axiom™



The 30 Year Horizon

Manuel Bronstein *William Burge* *Timothy Daly*
James Davenport *Michael Dewar* *Martin Dunstan*
Albrecht Fortenbacher *Patrizia Gianni* *Johannes Grabmeier*
Jocelyn Guidry *Richard Jenks* *Larry Lambe*
Michael Monagan *Scott Morrison* *William Sit*
Jonathan Steinbach *Robert Sutor* *Barry Trager*
Stephen Watt *Jim Wen* *Clifton Williamson*

Volume 10: Axiom Algebra: Categories

Portions Copyright (c) 2005 Timothy Daly

The Blue Bayou image Copyright (c) 2004 Jocelyn Guidry

Portions Copyright (c) 2004 Martin Dunstan

Portions Copyright (c) 1991-2002,
The Numerical ALgorithms Group Ltd.
All rights reserved.

This book and the Axiom software is licensed as follows:

Redistribution and use in source and binary forms, with or
without modification, are permitted provided that the following
conditions are
met:

- Redistributions of source code must retain the above
  copyright notice, this list of conditions and the
  following disclaimer.

- Redistributions in binary form must reproduce the above
  copyright notice, this list of conditions and the
  following disclaimer in the documentation and/or other
  materials provided with the distribution.

- Neither the name of The Numerical ALgorithms Group Ltd.
  nor the names of its contributors may be used to endorse
  or promote products derived from this software without
  specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.

Inclusion of names in the list of credits is based on historical information and is as accurate as possible. Inclusion of names does not in any way imply an endorsement but represents historical influence on Axiom development.

Cyril Alberga
George Andrews
Yurij Baransky
Gilbert Baumslag
Mark Botch
Martin Brock
William Burge
Bruce Char
Gregory V. Chudnovsky
Don Coppersmith
Gary Cornell
Timothy Daly Sr.
Jean Della Dora
Claire DiCrescendo
Martin Dunstan
Robert Edwards
Richard Fateman
Brian Ford
Constantine Frangos
Marc Gaetano
Patricia Gianni
Laureano Gonzalez-Vega
Matt Grayson
Oswald Gschnitzer
Vilya Harvey
Ralf Hemmecke
Pietro Iglio
Grant Keady
Klaus Kusche
Frederic Lehobey
Rudiger Loos
Camm Maguire
Ian Meikle
Gerard Milmeister
Michael Monagan
Mark Murray
John Nelder
Jinzhong Niu
Julian A. Padget
Susan Pelzel
Claude Quitte
Renaud Rioboo
Simon Robinson
Philip Santas
Gerhard Schneider
Fritz Schwarz
Elena Smirnova
Robert Sutor
James Thatcher
Dylan Thurston
Gregory Vanuxem
Juergen Weiss
James Wen
John M. Wiley
Stephen Wilson
Sandra Wityak

Roy Adler
Henry Baker
David R. Barton
Fred Blair
Alexandre Bouyer
Manuel Bronstein
Quentin Carpent
Cheekai Chin
Josh Cohen
George Corliss
Meino Cramer
Timothy Daly Jr.
Gabriel Dos Reis
Sam Dooley
Brian Dupee
Heow Eide-Goodman
Bertfried Fauser
Albrecht Fortenbacher
Timothy Freeman
Rudiger Gebauer
Holger Gollan
Stephen Gortler
James Griesmer
Jocelyn Guidry
Satoshi Hamaguchi
Henderson
Richard Jenks
Tony Kennedy
Bernhard Kutzler
Michel Levaud
Michael Lucks
Bob McElrath
David Mentre
Mohammed Mobarak
Marc Moreno-Maza
William Naylor
Godfrey Nolan
Michael O'Connor
Bill Page
Michel Petitot
Norman Ramsey
Jean Rivlin
Michael Rothstein
Alfred Scheerhorn
Martin Schoenert
Nick Simicich
Jonathan Steinbach
Moss E. Sweedler
Baldir Thomas
Barry Trager
Bernhard Wall
M. Weller
Thorsten Werther
Berhard Will
Shmuel Winograd
Waldemar Wiwianka

Richard Anderson
Stephen Balzac
Gerald Baumgartner
Vladimir Bondarenko
Peter A. Broadbery
Florian Bundschuh
Bob Caviness
David V. Chudnovsky
Christophe Conil
Robert Corless
Claire Di Crescenzo
James H. Davenport
Michael Dewar
Lionel Ducos
Dominique Duval
Lars Erickson
Stuart Feldman
George Frances
Korrinn Fu
Kathy Gerber
Teresa Gomez-Diaz
Johannes Grabmeier
Vladimir Grinberg
Steve Hague
Martin Hassner
Antoine Hersen
Kai Kaminski
Paul Kosinski
Larry Lambe
Howard Levy
Richard Luczak
Michael McGettrick
Victor S. Miller
H. Michael Moeller
Scott Morrison
C. Andrew Neff
Arthur Norman
Kostas Oikonomou
Jaap Weel
Didier Pinchon
Michael Richardson
Nicolas Robidoux
Martin Rubey
William Schelter
Marshall Schor
William Sit
Christine Sundaresan
Eugene Surowitz
Mike Thomas
Themos T. Tsikas
Stephen Watt
Mark Wegman
Michael Wester
Clifton J. Williamson
Robert Wisbauer
Knut Wolf

Contents

1 Categories **1**

2 Category Layer 1 **3**

2.0.1 Category (CATEGORY) . . . 3
2.0.2 ArcHyperbolicFunctionCategory (AHYP) . . . 5
2.0.3 ArcTrigonometricFunctionCategory (ATRIG) . . . 7
2.0.4 AttributeRegistry (ATTREG) . . . 10
2.0.5 BasicType (BASTYPE) . . . 14
2.0.6 CoercibleTo (KOERCE) . . . 17
2.0.7 CombinatorialFunctionCategory (CFCAT) . . . 20
2.0.8 ConvertibleTo (KONVERT) . . . 23
2.0.9 ElementaryFunctionCategory (ELEMFUN) . . . 27
2.0.10 Eltable (ELTAB) . . . 29
2.0.11 HyperbolicFunctionCategory (HYPCAT) . . . 32
2.0.12 InnerEvalable (IEVALAB) . . . 35
2.0.13 OpenMath (OM) . . . 39
2.0.14 PartialTranscendentalFunctions (PTRANFN) . . . 42
2.0.15 Patternable (PATAB) . . . 47
2.0.16 PrimitiveFunctionCategory (PRIMCAT) . . . 50
2.0.17 RadicalCategory (RADCAT) . . . 52
2.0.18 RetractableTo (RETRACT) . . . 55
2.0.19 SpecialFunctionCategory (SPFCAT) . . . 59
2.0.20 TrigonometricFunctionCategory (TRIGCAT) . . . 63
2.0.21 Type (TYPE) . . . 66

3 Category Layer 2 **69**

3.0.22 Aggregate (AGG) . . . 69
3.0.23 CombinatorialOpsCategory (COMBOPC) . . . 73
3.0.24 EltableAggregate (ELTAGG) . . . 76
3.0.25 Evalable (EVALAB) . . . 80
3.0.26 FortranProgramCategory (FORTCAT) . . . 84
3.0.27 FullyRetractableTo (FRETRCT) . . . 87
3.0.28 FullyPatternMatchable (FPATMAB) . . . 91
3.0.29 Logic (LOGIC) . . . 95

3.0.30 PlottablePlaneCurveCategory (PPCURVE) . . . 98
3.0.31 PlottableSpaceCurveCategory (PSCURVE) . . . 102
3.0.32 RealConstant (REAL) . . . 106
3.0.33 SegmentCategory (SEGCAT) . . . 109
3.0.34 SetCategory (SETCAT) . . . 113
3.0.35 TranscendentalFunctionCategory (TRANFUN) . . . 117

4 Category Layer 3 123
4.0.36 AbelianSemiGroup (ABELSG) . . . 123
4.0.37 BlowUpMethodCategory (BLMETCT) . . . 127
4.0.38 DesingTreeCategory (DSTRCAT) . . . 131
4.0.39 FortranFunctionCategory (FORTFN) . . . 136
4.0.40 FortranMatrixCategory (FMC) . . . 141
4.0.41 FortranMatrixFunctionCategory (FMFUN) . . . 145
4.0.42 FortranVectorCategory (FVC) . . . 150
4.0.43 FortranVectorFunctionCategory (FVFUN) . . . 154
4.0.44 FullyEvalableOver (FEVALAB) . . . 159
4.0.45 FileCategory (FILECAT) . . . 163
4.0.46 Finite (FINITE) . . . 168
4.0.47 FileNameCategory (FNCAT) . . . 172
4.0.48 GradedModule (GRMOD) . . . 176
4.0.49 HomogeneousAggregate (HOAGG) . . . 181
4.0.50 IndexedDirectProductCategory (IDPC) . . . 188
4.0.51 LiouvillianFunctionCategory (LFCAT) . . . 192
4.0.52 Monad (MONAD) . . . 197
4.0.53 NumericalIntegrationCategory (NUMINT) . . . 202
4.0.54 NumericalOptimizationCategory (OPTCAT) . . . 207
4.0.55 OrdinaryDifferentialEquationsSolverCategory (ODECAT) . . . 212
4.0.56 OrderedSet (ORDSET) . . . 216
4.0.57 PartialDifferentialEquationsSolverCategory (PDECAT) . . . 221
4.0.58 PatternMatchable (PATMAB) . . . 226
4.0.59 RealRootCharacterizationCategory (RRCC) . . . 230
4.0.60 SegmentExpansionCategory (SEGXCAT) . . . 235
4.0.61 SemiGroup (SGROUP) . . . 239
4.0.62 SetCategoryWithDegree (SETCATD) . . . 243
4.0.63 SExpressionCategory (SEXCAT) . . . 246
4.0.64 StepThrough (STEP) . . . 252
4.0.65 ThreeSpaceCategory (SPACEC) . . . 256

5 Category Layer 4 267
5.0.66 AbelianMonoid (ABELMON) . . . 267
5.0.67 AffineSpaceCategory (AFSPCAT) . . . 272
5.0.68 BagAggregate (BGAGG) . . . 277
5.0.69 CachableSet (CACHSET) . . . 283
5.0.70 Collection (CLAGG) . . . 287
5.0.71 DifferentialVariableCategory (DVARCAT) . . . 294

5.0.72 ExpressionSpace (ES) . . . 300
5.0.73 GradedAlgebra (GRALG) . . . 313
5.0.74 IndexedAggregate (IXAGG) . . . 318
5.0.75 MonadWithUnit (MONADWU) . . . 325
5.0.76 Monoid (MONOID) . . . 331
5.0.77 OrderedFinite (ORDFIN) . . . 336
5.0.78 PlacesCategory (PLACESC) . . . 340
5.0.79 ProjectiveSpaceCategory (PRSPCAT) . . . 345
5.0.80 RecursiveAggregate (RCAGG) . . . 351
5.0.81 TwoDimensionalArrayCategory (ARR2CAT) . . . 357

6 Category Layer 5 371
6.0.82 BinaryRecursiveAggregate (BRAGG) . . . 372
6.0.83 CancellationAbelianMonoid (CABMON) . . . 380
6.0.84 DictionaryOperations (DIOPS) . . . 385
6.0.85 DoublyLinkedAggregate (DLAGG) . . . 392
6.0.86 Group (GROUP) . . . 398
6.0.87 LinearAggregate (LNAGG) . . . 404
6.0.88 MatrixCategory (MATCAT) . . . 412
6.0.89 OrderedAbelianSemiGroup (OASGP) . . . 456
6.0.90 OrderedMonoid (ORDMON) . . . 461
6.0.91 PolynomialSetCategory (PSETCAT) . . . 465
6.0.92 PriorityQueueAggregate (PRQAGG) . . . 480
6.0.93 QueueAggregate (QUAGG) . . . 486
6.0.94 SetAggregate (SETAGG) . . . 492
6.0.95 StackAggregate (SKAGG) . . . 500
6.0.96 UnaryRecursiveAggregate (URAGG) . . . 506

7 Category Layer 6 519
7.0.97 AbelianGroup (ABELGRP) . . . 520
7.0.98 BinaryTreeCategory (BTCAT) . . . 526
7.0.99 Dictionary (DIAGG) . . . 533
7.0.100 DequeueAggregate (DQAGG) . . . 540
7.0.101 ExtensibleLinearAggregate (ELAGG) . . . 547
7.0.102 FiniteLinearAggregate (FLAGG) . . . 555
7.0.103 FreeAbelianMonoidCategory (FAMONC) . . . 564
7.0.104 MultiDictionary (MDAGG) . . . 570
7.0.105 OrderedAbelianMonoid (OAMON) . . . 576
7.0.106 PermutationCategory (PERMCAT) . . . 580
7.0.107 StreamAggregate (STAGG) . . . 585
7.0.108 TriangularSetCategory (TSETCAT) . . . 595

8 Category Layer 7 **615**
8.0.109 FiniteDivisorCategory (FDIVCAT) . . . 616
8.0.110 FiniteSetAggregate (FSAGG) . . . 621
8.0.111 KeyedDictionary (KDAGG) . . . 630
8.0.112 LazyStreamAggregate (LZSTAGG) . . . 637
8.0.113 LeftModule (LMODULE) . . . 655
8.0.114 ListAggregate (LSAGG) . . . 659
8.0.115 MultisetAggregate (MSETAGG) . . . 673
8.0.116 NonAssociativeRng (NARNG) . . . 679
8.0.117 OneDimensionalArrayAggregate (A1AGG) . . . 684
8.0.118 OrderedCancellationAbelianMonoid (OCAMON) . . . 696
8.0.119 RegularTriangularSetCategory (RSETCAT) . . . 700
8.0.120 RightModule (RMODULE) . . . 715
8.0.121 Rng (RNG) . . . 719

9 Category Layer 8 **725**
9.0.122 BiModule (BMODULE) . . . 726
9.0.123 BitAggregate (BTAGG) . . . 731
9.0.124 NonAssociativeRing (NASRING) . . . 740
9.0.125 NormalizedTriangularSetCategory (NTSCAT) . . . 745
9.0.126 OrderedAbelianGroup (OAGROUP) . . . 755
9.0.127 OrderedAbelianMonoidSup (OAMONS) . . . 759
9.0.128 OrderedMultisetAggregate (OMSAGG) . . . 763
9.0.129 Ring (RING) . . . 770
9.0.130 SquareFreeRegularTriangularSetCategory (SFRTCAT) . . . 775
9.0.131 StringAggregate (SRAGG) . . . 786
9.0.132 TableAggregate (TBAGG) . . . 797
9.0.133 VectorCategory (VECTCAT) . . . 808

10 Category Layer 9 **819**
10.0.134 AssociationListAggregate (ALAGG) . . . 819
10.0.135 CharacteristicNonZero (CHARNZ) . . . 833
10.0.136 CharacteristicZero (CHARZ) . . . 838
10.0.137 CommutativeRing (COMRING) . . . 843
10.0.138 DifferentialRing (DIFRING) . . . 848
10.0.139 EntireRing (ENTIRER) . . . 853
10.0.140 FreeModuleCat (FMCAT) . . . 858
10.0.141 LeftAlgebra (LALG) . . . 864
10.0.142 LinearlyExplicitRingOver (LINEXP) . . . 869
10.0.143 Module (MODULE) . . . 874
10.0.144 OrderedRing (ORDRING) . . . 879
10.0.145 PartialDifferentialRing (PDRING) . . . 885
10.0.146 PointCategory (PTCAT) . . . 893
10.0.147 RectangularMatrixCategory (RMATCAT) . . . 901
10.0.148 SquareFreeNormalizedTriangularSetCategory (SNTSCAT) . . . 910
10.0.149 StringCategory (STRICAT) . . . 920

10.0.150 UnivariateSkewPolynomialCategory (OREPCAT) 929
10.0.151 XAlgebra (XALG) 940

11 Category Layer 10 **947**
11.0.152 Algebra (ALGEBRA) 947
11.0.153 DifferentialExtension (DIFEXT) 953
11.0.154 FullyLinearlyExplicitRingOver (FLINEXP) 960
11.0.155 LieAlgebra (LIECAT) 966
11.0.156 LinearOrdinaryDifferentialOperatorCategory (LODOCAT) 971
11.0.157 NonAssociativeAlgebra (NAALG) 980
11.0.158 VectorSpace (VSPACE) 987
11.0.159 XFreeAlgebra (XFALG) 992

12 Category Layer 11 **1001**
12.0.160 DirectProductCategory (DIRPCAT) 1001
12.0.161 DivisionRing (DIVRING) 1013
12.0.162 FiniteRankNonAssociativeAlgebra (FINAALG) 1019
12.0.163 FreeLieAlgebra (FLALG) 1041
12.0.164 IntegralDomain (INTDOM) 1047
12.0.165 MonogenicLinearOperator (MLO) 1053
12.0.166 OctonionCategory (OC) 1059
12.0.167 QuaternionCategory (QUATCAT) 1071
12.0.168 SquareMatrixCategory (SMATCAT) 1082
12.0.169 XPolynomialsCat (XPOLYC) 1094

13 Category Layer 12 **1101**
13.0.170 AbelianMonoidRing (AMR) 1101
13.0.171 FortranMachineTypeCategory (FMTC) 1108
13.0.172 FramedNonAssociativeAlgebra (FRNAALG) 1115
13.0.173 GcdDomain (GCDDOM) 1129
13.0.174 OrderedIntegralDomain (OINTDOM) 1135

14 Category Layer 13 **1141**
14.0.175 FiniteAbelianMonoidRing (FAMR) 1141
14.0.176 IntervalCategory (INTCAT) 1150
14.0.177 PowerSeriesCategory (PSCAT) 1159
14.0.178 PrincipalIdealDomain (PID) 1166
14.0.179 UniqueFactorizationDomain (UFD) 1172

15 Category Layer 14 **1179**
15.0.180 DivisorCategory (DIVCAT) 1179
15.0.181 EuclideanDomain (EUCDOM) 1185
15.0.182 MultivariateTaylorSeriesCategory (MTSCAT) 1193
15.0.183 PolynomialFactorizationExplicit (PFECAT) 1202
15.0.184 UnivariatePowerSeriesCategory (UPSCAT) 1210

16 Category Layer 15 **1221**
16.0.185 Field (FIELD) . . . 1221
16.0.186 IntegerNumberSystem (INS) . . . 1228
16.0.187 LocalPowerSeriesCategory (LOCPOWC) . . . 1239
16.0.188 PAdicIntegerCategory (PADICCT) . . . 1249
16.0.189 PolynomialCategory (POLYCAT) . . . 1255
16.0.190 UnivariateTaylorSeriesCategory (UTSCAT) . . . 1277

17 Category Layer 16 **1293**
17.0.191 AlgebraicallyClosedField (ACF) . . . 1293
17.0.192 DifferentialPolynomialCategory (DPOLCAT) . . . 1306
17.0.193 FieldOfPrimeCharacteristic (FPC) . . . 1323
17.0.194 FiniteRankAlgebra (FINRALG) . . . 1329
17.0.195 FunctionSpace (FS) . . . 1336
17.0.196 InfinitlyClosePointCategory (INFCLCT) . . . 1364
17.0.197 PseudoAlgebraicClosureOfPerfectFieldCategory (PACPERC) . . . 1369
17.0.198 QuotientFieldCategory (QFCAT) . . . 1376
17.0.199 RealClosedField (RCFIELD) . . . 1390
17.0.200 RealNumberSystem (RNS) . . . 1400
17.0.201 RecursivePolynomialCategory (RPOLCAT) . . . 1408
17.0.202 UnivariateLaurentSeriesCategory (ULSCAT) . . . 1448
17.0.203 UnivariatePuiseuxSeriesCategory (UPXSCAT) . . . 1460
17.0.204 UnivariatePolynomialCategory (UPOLYC) . . . 1471

18 Category Layer 17 **1495**
18.0.205 AlgebraicallyClosedFunctionSpace (ACFS) . . . 1495
18.0.206 ExtensionField (XF) . . . 1510
18.0.207 FiniteFieldCategory (FFIELDC) . . . 1518
18.0.208 FloatingPointSystem (FPS) . . . 1530
18.0.209 FramedAlgebra (FRAMALG) . . . 1539
18.0.210 PseudoAlgebraicClosureOfFiniteFieldCategory (PACFFC) . . . 1546
18.0.211 UnivariateLaurentSeriesConstructorCategory (ULSCCAT) . . . 1553
18.0.212 UnivariatePuiseuxSeriesConstructorCategory (UPXSCCA) . . . 1570

19 Category Layer 18 **1583**
19.0.213 FiniteAlgebraicExtensionField (FAXF) . . . 1583
19.0.214 MonogenicAlgebra (MONOGEN) . . . 1598
19.0.215 PseudoAlgebraicClosureOfRationalNumberCategory (PACRATC) . . . 1610

20 Category Layer 19 **1619**
20.0.216 ComplexCategory (COMPCAT) . . . 1619
20.0.217 FunctionFieldCategory (FFCAT) . . . 1642
20.0.218 PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory (PACEXTC) 1665

21 The bootstrap code 1675
21.1 ABELGRP.lsp BOOTSTRAP 1675
21.2 ABELGRP-.lsp BOOTSTRAP 1676
21.3 ABELMON.lsp BOOTSTRAP 1678
21.4 ABELMON-.lsp BOOTSTRAP 1679
21.5 ABELSG.lsp BOOTSTRAP 1680
21.6 ABELSG-.lsp BOOTSTRAP 1681
21.7 ALAGG.lsp BOOTSTRAP 1683
21.8 CABMON.lsp BOOTSTRAP 1684
21.9 CLAGG.lsp BOOTSTRAP 1685
21.10CLAGG-.lsp BOOTSTRAP 1687
21.11COMRING.lsp BOOTSTRAP 1691
21.12DIFRING.lsp BOOTSTRAP 1692
21.13DIFRING-.lsp BOOTSTRAP 1693
21.14DIVRING.lsp BOOTSTRAP 1695
21.15DIVRING-.lsp BOOTSTRAP 1696
21.16ES.lsp BOOTSTRAP 1698
21.17ES-.lsp BOOTSTRAP 1700
21.18EUCDOM.lsp BOOTSTRAP 1716
21.18.1 The Lisp Implementation 1716
21.19EUCDOM-.lsp BOOTSTRAP 1719
21.19.1 The Lisp Implementation 1719
21.20ENTIRER.lsp BOOTSTRAP 1732
21.21FFIELDC.lsp BOOTSTRAP 1733
21.22FFIELDC-.lsp BOOTSTRAP 1734
21.23FPS.lsp BOOTSTRAP 1745
21.24FPS-.lsp BOOTSTRAP 1747
21.25GCDDOM.lsp BOOTSTRAP 1749
21.26GCDDOM-.lsp BOOTSTRAP 1750
21.27HOAGG.lsp BOOTSTRAP 1755
21.28HOAGG-.lsp BOOTSTRAP 1757
21.29INS.lsp BOOTSTRAP 1763
21.30INS-.lsp BOOTSTRAP 1765
21.31INTDOM.lsp BOOTSTRAP 1773
21.32INTDOM-.lsp BOOTSTRAP 1774
21.33LNAGG.lsp BOOTSTRAP 1776
21.34LNAGG-.lsp BOOTSTRAP 1778
21.35LSAGG.lsp BOOTSTRAP 1780
21.36LSAGG-.lsp BOOTSTRAP 1781
21.37MONOID.lsp BOOTSTRAP 1798
21.38MONOID-.lsp BOOTSTRAP 1799
21.39MTSCAT.lsp BOOTSTRAP 1801
21.40OINTDOM.lsp BOOTSTRAP 1803
21.41ORDRING.lsp BOOTSTRAP 1804
21.42ORDRING-.lsp BOOTSTRAP 1805
21.43POLYCAT.lsp BOOTSTRAP 1807

21.44POLYCAT-.lsp BOOTSTRAP 1809
21.45PSETCAT.lsp BOOTSTRAP 1840
21.46PSETCAT-.lsp BOOTSTRAP 1842
21.47QFCAT.lsp BOOTSTRAP 1859
21.48QFCAT-.lsp BOOTSTRAP 1861
21.49RCAGG.lsp BOOTSTRAP 1869
21.50RCAGG-.lsp BOOTSTRAP 1871
21.51RING.lsp BOOTSTRAP 1872
21.52RING-.lsp BOOTSTRAP 1873
21.53RNG.lsp BOOTSTRAP 1875
21.54RNS.lsp BOOTSTRAP 1875
21.55RNS-.lsp BOOTSTRAP 1877
21.56SETAGG.lsp BOOTSTRAP 1881
21.57SETAGG-.lsp BOOTSTRAP 1882
21.58SETCAT.lsp BOOTSTRAP 1884
21.59SETCAT-.lsp BOOTSTRAP 1885
21.60STAGG.lsp BOOTSTRAP 1886
21.61STAGG-.lsp BOOTSTRAP 1887
21.62TSETCAT.lsp BOOTSTRAP 1894
21.63TSETCAT-.lsp BOOTSTRAP 1897
21.64UFD.lsp BOOTSTRAP 1917
21.65UFD-.lsp BOOTSTRAP 1918
21.66ULSCAT.lsp BOOTSTRAP 1921
21.67UPOLYC.lsp BOOTSTRAP 1922
21.68UPOLYC-.lsp BOOTSTRAP 1926
21.69URAGG.lsp BOOTSTRAP 1953
21.70URAGG-.lsp BOOTSTRAP 1955

22 Chunk collections **1971**

New Foreword

On October 1, 2001 Axiom was withdrawn from the market and ended life as a commercial product. On September 3, 2002 Axiom was released under the Modified BSD license, including this document. On August 27, 2003 Axiom was released as free and open source software available for download from the Free Software Foundation's website, Savannah.

Work on Axiom has had the generous support of the Center for Algorithms and Interactive Scientific Computation (CAISS) at City College of New York. Special thanks go to Dr. Gilbert Baumslag for his support of the long term goal.

The online version of this documentation is roughly 1000 pages. In order to make printed versions we've broken it up into three volumes. The first volume is tutorial in nature. The second volume is for programmers. The third volume is reference material. We've also added a fourth volume for developers. All of these changes represent an experiment in print-on-demand delivery of documentation. Time will tell whether the experiment succeeded.

Axiom has been in existence for over thirty years. It is estimated to contain about three hundred man-years of research and has, as of September 3, 2003, 143 people listed in the credits. All of these people have contributed directly or indirectly to making Axiom available. Axiom is being passed to the next generation. I'm looking forward to future milestones.

With that in mind I've introduced the theme of the "30 year horizon". We must invent the tools that support the Computational Mathematician working 30 years from now. How will research be done when every bit of mathematical knowledge is online and instantly available? What happens when we scale Axiom by a factor of 100, giving us 1.1 million domains? How can we integrate theory with code? How will we integrate theorems and proofs of the mathematics with space-time complexity proofs and running code? What visualization tools are needed? How do we support the conceptual structures and semantics of mathematics in effective ways? How do we support results from the sciences? How do we teach the next generation to be effective Computational Mathematicians?

The "30 year horizon" is much nearer than it appears.

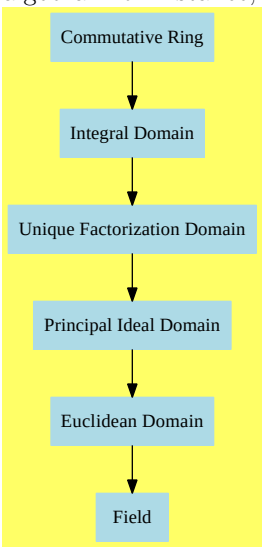
Tim Daly
CAISS, City College of New York
November 10, 2003 ((iHy))

Chapter 1

Categories

Axiom has 3 main algebra components, Categories, Domains, and Packages. If we make an analogy to dressmaking, you can consider the Categories to be hierarchies of properties of things, like patterns, colors, or fabrics. Domains are instances of things based on category choices, such as a dress with a particular style, fabric, color, etc. Packages are tools that work with dresses such as irons, sewing machines, etc.

Axiom is based on abstract algebra and uses it as a scaffolding for constructing well-formed algebra. For instance, in abstract algebra there is a strict subset hierarchy, like:



— **algebrahierarchy.dotpic** —

```
digraph pic {
 fontsize=10;
```

```
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"Commutative Ring"
  [color=lightblue,href="bookvol10.2.pdf#nameddest=COMRING"];
"Integral Domain"
  [color=lightblue,href="bookvol10.2.pdf#nameddest=INTDOM"];
"Unique Factorization Domain"
  [color=lightblue,href="bookvol10.2.pdf#nameddest=UFD"];
"Principal Ideal Domain"
  [color=lightblue,href="bookvol10.2.pdf#nameddest=PID"];
"Euclidean Domain"
  [color=lightblue,href="bookvol10.2.pdf#nameddest=EUCDOM"];
"Field"
  [color=lightblue,href="bookvol10.2.pdf#nameddest=FIELD"];

"Commutative Ring" -> "Integral Domain"
"Integral Domain" -> "Unique Factorization Domain"
"Unique Factorization Domain" -> "Principal Ideal Domain"
"Principal Ideal Domain" -> "Euclidean Domain"
"Euclidean Domain" -> "Field"

}
```

Chapter 2

Category Layer 1

In general, we use several colors in the graph images. The "lightblue" color indicates a category that is in the direct inheritance path. The "green" (#00EE00) color indicates a category or domain used in the exports. The "seagreen" (a dark green, indicates a category or domain which is used but does not correspond to the signature of an existing category. The system can infer that this "subsumption node" matches the category. A "yellow" color indicates a domain.

2.0.1 Category (CATEGORY)



This is the root of the category hierarchy and is not represented by code.

See:

⇒ "ArcHyperbolicFunctionCategory" (AHYP) 2.0.2 on page 5
⇒ "ArcTrigonometricFunctionCategory" (ATRIG) 2.0.3 on page 7
⇒ "BasicType" (BASTYPE) 2.0.5 on page 14
⇒ "CoercibleTo" (KOERCE) 2.0.6 on page 17
⇒ "CombinatorialFunctionCategory" (CFCAT) 2.0.7 on page 20
⇒ "ConvertibleTo" (KONVERT) 2.0.8 on page 23
⇒ "ElementaryFunctionCategory" (ELEMFUN) 2.0.9 on page 27
⇒ "Eltable" (ELTAB) 2.0.10 on page 29
⇒ "FullyEvalableOver" (FEVALAB) 4.0.44 on page 159
⇒ "HyperbolicFunctionCategory" (HYPCAT) 2.0.11 on page 32
⇒ "InnerEvalable" (IEVALAB) 2.0.12 on page 35

⇒ “Logic” (LOGIC) 3.0.29 on page 95
⇒ “OpenMath” (OM) 2.0.13 on page 39
⇒ “PartialTranscendentalFunctions” (PTRANFN) 2.0.14 on page 42
⇒ “Patternable” (PATAB) 2.0.15 on page 47
⇒ “PrimitiveFunctionCategory” (PRIMCAT) 2.0.16 on page 50
⇒ “RadicalCategory” (RADCAT) 2.0.17 on page 52
⇒ “RetractableTo” (RETRACT) 2.0.18 on page 55
⇒ “SpecialFunctionCategory” (SPFCAT) 2.0.19 on page 59
⇒ “TrigonometricFunctionCategory” (TRIGCAT) 2.0.20 on page 63
⇒ “Type” (TYPE) 2.0.21 on page 66

— **CATEGORY.dotabb** —

```
"CATEGORY"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=CATEGORY"];
```

————-

— **CATEGORY.dotfull** —

```
"Category"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=CATEGORY"];
```

————-

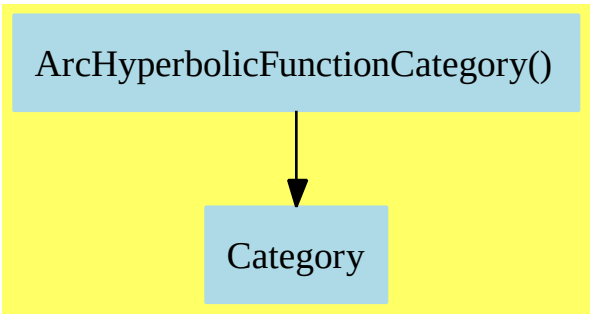
— **CATEGORY.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"Category" [color=lightblue];
}
```

————-

2.0.2 ArcHyperbolicFunctionCategory (AHYP)



— **ArcHyperbolicFunctionCategory.input** —

```
)set break resume
)sys rm -f ArcHyperbolicFunctionCategory.output
)spool ArcHyperbolicFunctionCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show ArcHyperbolicFunctionCategory
--R ArcHyperbolicFunctionCategory  is a category constructor
--R Abbreviation for ArcHyperbolicFunctionCategory is AHYP
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for AHYP
--R
--R------------------------------ Operations --------------------------------
--R acosh : % -> %                       acoth : % -> %
--R acsch : % -> %                       asech : % -> %
--R asinh : % -> %                       atanh : % -> %
--R
--E 1

)spool
)lisp (bye)
```

————-

— **ArcHyperbolicFunctionCategory.help** —

```
=======================================================================
ArcHyperbolicFunctionCategory examples
=======================================================================

See Also:
```

```
o )show ArcHyperbolicFunctionCategory
```

———-

See:

⇒ “TranscendentalFunctionCategory” (TRANFUN) 3.0.35 on page 117
⇐ “Category” (CATEGORY) 2.0.1 on page 3

Exports:

acosh acoth acsch asech asinh atanh

These are directly exported but not implemented:

```
acosh : % -> %
acoth : % -> %
acsch : % -> %
asech : % -> %
asinh : % -> %
atanh : % -> %
```

— **category AHYP ArcHyperbolicFunctionCategory** —

```
)abbrev category AHYP ArcHyperbolicFunctionCategory
++ Category for the inverse hyperbolic trigonometric functions
++ Author: ???
++ Date Created: ???
++ Date Last Updated: 14 May 1991
++ Description:
++ Category for the inverse hyperbolic trigonometric functions;

ArcHyperbolicFunctionCategory(): Category == with
    acosh: $ -> $ ++ acosh(x) returns the hyperbolic arc-cosine of x.
    acoth: $ -> $ ++ acoth(x) returns the hyperbolic arc-cotangent of x.
    acsch: $ -> $ ++ acsch(x) returns the hyperbolic arc-cosecant of x.
    asech: $ -> $ ++ asech(x) returns the hyperbolic arc-secant of x.
    asinh: $ -> $ ++ asinh(x) returns the hyperbolic arc-sine of x.
    atanh: $ -> $ ++ atanh(x) returns the hyperbolic arc-tangent of x.
```

———-

— **AHYP.dotabb** —

```
"AHYP"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=AHYP"];
```

```
"AHYP" -> "CATEGORY"
```

———-

— **AHYP.dotfull** —

```
"ArcHyperbolicFunctionCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=AHYP"];
"ArcHyperbolicFunctionCategory()" -> "Category"
```

———-

— **AHYP.dotpic** —

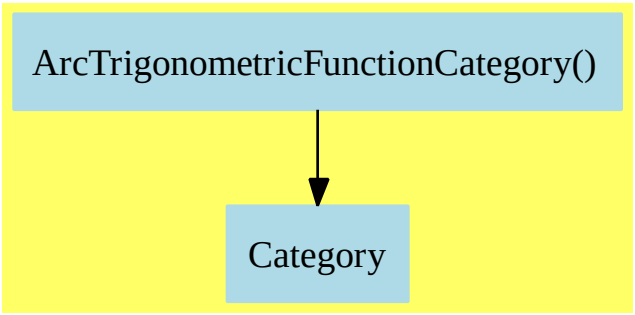
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"ArcHyperbolicFunctionCategory()" [color=lightblue];
"ArcHyperbolicFunctionCategory()" -> "Category"

"Category" [color=lightblue];
}
```

———-

2.0.3 ArcTrigonometricFunctionCategory (ATRIG)



The `asec` and `acsc` functions were modified to include an intermediate test to check that the argument has a reciprocal values.

— ArcTrigonometricFunctionCategory.input —

```
)set break resume
)sys rm -f ArcTrigonometricFunctionCategory.output
)spool ArcTrigonometricFunctionCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show ArcTrigonometricFunctionCategory
--R ArcTrigonometricFunctionCategory  is a category constructor
--R Abbreviation for ArcTrigonometricFunctionCategory is ATRIG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for ATRIG
--R
--R------------------------------ Operations --------------------------------
--R acos : % -> %                         acot : % -> %
--R acsc : % -> %                         asec : % -> %
--R asin : % -> %                         atan : % -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— ArcTrigonometricFunctionCategory.help —

```
======================================================================
ArcTrigonometricFunctionCategory examples
======================================================================

See Also:
o )show ArcTrigonometricFunctionCategory
```

———-

See:

⇒ “TranscendentalFunctionCategory” (TRANFUN) 3.0.35 on page 117
⇐ “Category” (CATEGORY) 2.0.1 on page 3

Exports:

acos acot acsc asec asin atan

These are directly exported but not implemented:

```
 acos : % -> %
 acot : % -> %
 asin : % -> %
 atan : % -> %
```

These are implemented by this category:

```
 acsc : % -> %
 asec : % -> %
```

— category ATRIG ArcTrigonometricFunctionCategory —

```
)abbrev category ATRIG ArcTrigonometricFunctionCategory
++ Category for the inverse trigonometric functions
++ Author: ???
++ Date Created: ???
++ Date Last Updated: 14 May 1991
++ Description:
++ Category for the inverse trigonometric functions;

ArcTrigonometricFunctionCategory(): Category == with
    acos: $ -> $       ++ acos(x) returns the arc-cosine of x.
    acot: $ -> $       ++ acot(x) returns the arc-cotangent of x.
    acsc: $ -> $       ++ acsc(x) returns the arc-cosecant of x.
    asec: $ -> $       ++ asec(x) returns the arc-secant of x.
    asin: $ -> $       ++ asin(x) returns the arc-sine of x.
    atan: $ -> $       ++ atan(x) returns the arc-tangent of x.
 add
    if $ has Ring then
       asec(x) ==
         (a := recip x) case "failed" => error "asec: no reciprocal"
         acos(a::$)
       acsc(x) ==
         (a := recip x) case "failed" => error "acsc: no reciprocal"
         asin(a::$)
```

———-

— ATRIG.dotabb —

```
"ATRIG"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ATRIG"];
"ATRIG" -> "CATEGORY"
```

———-

— **ATRIG.dotfull** —

```
"ArcTrigonometricFunctionCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ATRIG"];
"ArcTrigonometricFunctionCategory()" -> "Category"
```

———-

— **ATRIG.dotpic** —

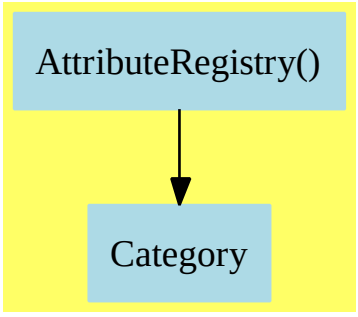
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"ArcTrigonometricFunctionCategory()" [color=lightblue];
"ArcTrigonometricFunctionCategory()" -> "Category"

"Category" [color=lightblue];
}
```

———-

2.0.4 AttributeRegistry (ATTREG)



See:

⇐ “Category” (CATEGORY) 2.0.1 on page 3

Exports: Nothing

Attributes Exported:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.
- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.
- **arbitraryPrecision** means the user can set the precision for subsequent calculations.
- **partiallyOrderedSet** is true if a set with $<$ which is transitive, but not($a < b$ or $a = b$) does not necessarily imply $b < a$.
- **central** is true if, given an algebra over a ring R, the image of R is the center of the algebra, i.e. the set of members of the algebra which commute with all others is precisely the image of R in the algebra.
- **noetherian** is true if all of its ideals are finitely generated.
- **additiveValuation** implies `euclideanSize(a*b)=euclideanSize(a)+euclideanSize(b)`.
- **multiplicativeValuation** implies `euclideanSize(a*b)=euclideanSize(a)*euclideanSize(b)`.
- **NullSquare** means that $[x, x] = 0$ holds. See `LieAlgebra`.
- **JacobiIdentity** means that $[x, [y, z]] + [y, [z, x]] + [z, [x, y]] = 0$ holds. See `LieAlgebra`.
- **canonical** is true if and only if distinct elements have distinct data structures. For example, a domain of mathematical objects which has the `canonical` attribute means that two objects are mathematically equal if and only if their data structures are equal.

- **approximate** means "is an approximation to the real numbers".
- **complex** means that this domain has $\sqrt{-1}$

— category ATTREG AttributeRegistry —

```
)abbrev category ATTREG AttributeRegistry
++ Description:
++ This category exports the attributes in the AXIOM Library

AttributeRegistry(): Category == with
  finiteAggregate
    ++ \spad{finiteAggregate} is true if it is an aggregate with a
    ++ finite number of elements.
  commutative("*")
    ++ \spad{commutative("*")} is true if it has an operation
    ++ \spad{"*": (D,D) -> D} which is commutative.
  shallowlyMutable
    ++ \spad{shallowlyMutable} is true if its values
    ++ have immediate components that are updateable (mutable).
    ++ Note that the properties of any component domain are irrevelant
    ++ to the \spad{shallowlyMutable} proper.
  unitsKnown
    ++ \spad{unitsKnown} is true if a monoid (a multiplicative semigroup
    ++ with a 1) has \spad{unitsKnown} means that
    ++ the operation \spadfun{recip} can only return "failed"
    ++ if its argument is not a unit.
  leftUnitary
    ++ \spad{leftUnitary} is true if \spad{1 * x = x} for all x.
  rightUnitary
    ++ \spad{rightUnitary} is true if \spad{x * 1 = x} for all x.
  noZeroDivisors
    ++ \spad{noZeroDivisors} is true if \spad{x * y \~~= 0} implies
    ++ both x and y are non-zero.
  canonicalUnitNormal
    ++ \spad{canonicalUnitNormal} is true if we can choose a canonical
    ++ representative for each class of associate elements, that is
    ++ \spad{associates?(a,b)} returns true if and only if
    ++ \spad{unitCanonical(a) = unitCanonical(b)}.
  canonicalsClosed
    ++ \spad{canonicalsClosed} is true if
    ++ \spad{unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)}.
  arbitraryPrecision
    ++ \spad{arbitraryPrecision} means the user can set the
    ++ precision for subsequent calculations.
  partiallyOrderedSet
    ++ \spad{partiallyOrderedSet} is true if
    ++ a set with \spadop{<} which is transitive,
    ++ but \spad{not(a < b or a = b)}
```

```
    ++ does not necessarily imply \spad{b<a}.
  central
    ++ \spad{central} is true if, given an algebra over a ring R,
    ++ the image of R is the center
    ++ of the algebra, i.e. the set of members of the algebra which commute
    ++ with all others is precisely the image of R in the algebra.
  noetherian
    ++ \spad{noetherian} is true if all of its ideals are finitely generated.
  additiveValuation
    ++ \spad{additiveValuation} implies
    ++ \spad{euclideanSize(a*b)=euclideanSize(a)+euclideanSize(b)}.
  multiplicativeValuation
    ++ \spad{multiplicativeValuation} implies
    ++ \spad{euclideanSize(a*b)=euclideanSize(a)*euclideanSize(b)}.
  NullSquare
    ++ \axiom{NullSquare} means that \axiom{[x,x] = 0} holds.
    ++ See \axiomType{LieAlgebra}.
  JacobiIdentity
    ++ \axiom{JacobiIdentity} means that
    ++ \axiom{[x,[y,z]]+[y,[z,x]]+[z,[x,y]] = 0} holds.
    ++ See \axiomType{LieAlgebra}.
  canonical
    ++ \spad{canonical} is true if and only if distinct elements have
    ++ distinct data structures. For example, a domain of mathematical
    ++ objects  which has the \spad{canonical} attribute means that two
    ++ objects are mathematically equal if and only if their data
    ++ structures are equal.
  approximate
    ++ \spad{approximate} means "is an approximation to the real numbers".
```

———-

— ATTREG.dotabb —

```
"ATTREG"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ATTREG"];
"ATTREG" -> "CATEGORY"
```

———-

— ATTREG.dotfull —

```
"AttributeRegistry()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ATTREG"];
"AttributeRegistry()" -> "Category"
```

———-

— **ATTREG.dotpic** —

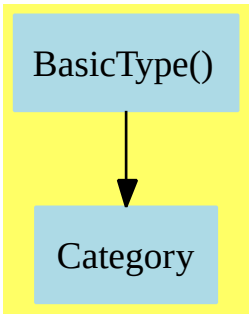
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"AttributeRegistry()" [color=lightblue];
"AttributeRegistry()" -> "Category"

"Category" [color=lightblue];
}
```

———-

2.0.5 BasicType (BASTYPE)



— **BasicType.input** —

```
)set break resume
)sys rm -f BasicType.output
)spool BasicType.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show BasicType
--R BasicType  is a category constructor
--R Abbreviation for BasicType is BASTYPE
```

```
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for BASTYPE
--R
--R------------------------------ Operations --------------------------------
--R ?=? : (%,%) -> Boolean               ?~=? : (%,%) -> Boolean
--R
--E 1

)spool
)lisp (bye)
```

———-

— BasicType.help —

```
=====================================================================
BasicType examples
=====================================================================

See Also:
o )show BasicType
```

———-

See:

⇒ “BlowUpMethodCategory” (BLMETCT) 4.0.37 on page 127
⇒ “SetCategory” (SETCAT) 3.0.34 on page 113
⇐ “Category” (CATEGORY) 2.0.1 on page 3

Exports:

?=? ?~=?

These are directly exported but not implemented:

```
?=? : (%,%) -> Boolean
```

These are implemented by this category:

```
?~=? : (%,%) -> Boolean
```

— category BASTYPE BasicType —

```
)abbrev category BASTYPE BasicType
--% BasicType
++ Author:
```

```
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ \spadtype{BasicType} is the basic category for describing a collection
++ of elements with \spadop{=} (equality).

BasicType(): Category == with
      "=": (%,%) -> Boolean    ++ x=y tests if x and y are equal.
      "~=": (%,%) -> Boolean   ++ x~=y tests if x and y are not equal.
   add
      _~_=(x:%,y:%) : Boolean == not(x=y)
```

———-

— BASTYPE.dotabb —

```
"BASTYPE"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=BASTYPE"];
"BASTYPE" -> "CATEGORY"
```

———-

— BASTYPE.dotfull —

```
"BasicType()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=BASTYPE"];
"BasicType()" -> "Category"
```

———-

— BASTYPE.dotpic —

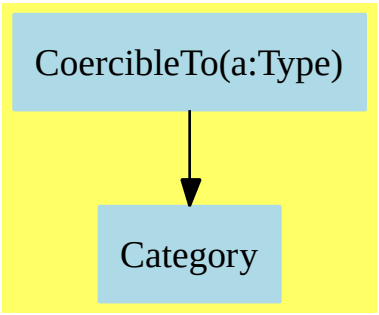
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"
```

```
"Category" [color=lightblue];
}
```

———-

2.0.6 CoercibleTo (KOERCE)



— **CoercibleTo.input** —

```
)set break resume
)sys rm -f CoercibleTo.output
)spool CoercibleTo.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show CoercibleTo
--R CoercibleTo S: Type  is a category constructor
--R Abbreviation for CoercibleTo is KOERCE
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for KOERCE
--R
--R------------------------------ Operations --------------------------------
--R coerce : % -> S
--R
--E 1

)spool
)lisp (bye)
```

———-

— CoercibleTo.help —

```
==================================================================
CoercibleTo examples
==================================================================

See Also:
o )show CoercibleTo
```

———-

See:

⇒ “BlowUpMethodCategory” (BLMETCT) 4.0.37 on page 127
⇒ “DirectProductCategory” (DIRPCAT) 12.0.160 on page 1001
⇒ “FortranProgramCategory” (FORTCAT) 3.0.26 on page 84
⇒ “PlottablePlaneCurveCategory” (PPCURVE) 3.0.30 on page 98
⇒ “PlottableSpaceCurveCategory” (PSCURVE) 3.0.31 on page 102
⇒ “PolynomialSetCategory” (PSETCAT) 6.0.91 on page 465
⇒ “SetCategory” (SETCAT) 3.0.34 on page 113
⇐ “Category” (CATEGORY) 2.0.1 on page 3

Exports:

coerce

This is directly exported but not implemented:

```
coerce : % -> S
```

— category KOERCE CoercibleTo —

```
)abbrev category KOERCE CoercibleTo
++ Category for coerce
++ Author: Manuel Bronstein
++ Date Created: ???
++ Date Last Updated: 14 May 1991
++ Description:
++ A is coercible to B means any element of A can automatically be
++ converted into an element of B by the interpreter.

CoercibleTo(S:Type): Category == with
    coerce: % -> S
      ++ coerce(a) transforms a into an element of S.
```

———-

— **KOERCE.dotabb** —

```
"KOERCE"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=KOERCE"];
"KOERCE" -> "CATEGORY"
```

———-

— **KOERCE.dotfull** —

```
"CoercibleTo(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=KOERCE"];
"CoercibleTo(a:Type)" -> "Category"

"CoercibleTo(OutputForm)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=KOERCE"];
"CoercibleTo(OutputForm)" ->
    "CoercibleTo(a:Type)"

"CoercibleTo(List(RecursivePolynomialCategory(Ring,OrderedAbelianMonoidSup(),OrderedSet()))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=KOERCE"];
"CoercibleTo(List(RecursivePolynomialCategory(Ring,OrderedAbelianMonoidSup(),OrderedSet()))"
  -> "CoercibleTo(a:Type)"
```

———-

— **KOERCE.dotpic** —

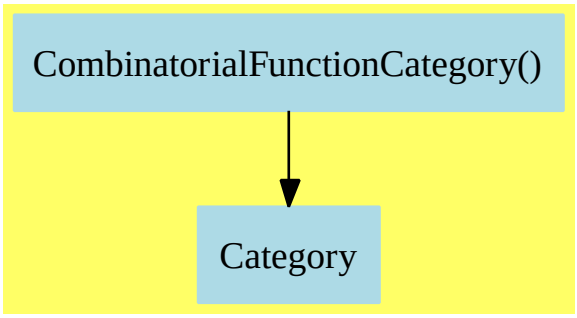
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];
}
```

———-

2.0.7 CombinatorialFunctionCategory (CFCAT)



— **CombinatorialFunctionCategory.input** —

```
)set break resume
)sys rm -f CombinatorialFunctionCategory.output
)spool CombinatorialFunctionCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show CombinatorialFunctionCategory
--R CombinatorialFunctionCategory  is a category constructor
--R Abbreviation for CombinatorialFunctionCategory is CFCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for CFCAT
--R
--R------------------------------ Operations --------------------------------
--R binomial : (%,%) -> %                factorial : % -> %
--R permutation : (%,%) -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **CombinatorialFunctionCategory.help** —

```
=======================================================================
CombinatorialFunctionCategory examples
=======================================================================

See Also:
```

```
o )show CombinatorialFunctionCategory
```

———-

See:

⇒ “CombinatorialOpsCategory” (COMBOPC) 3.0.23 on page 73
⇒ “IntegerNumberSystem” (INS) 16.0.186 on page 1228
⇐ “Category” (CATEGORY) 2.0.1 on page 3

Exports:

binomial factorial permutation

These are directly exported but not implemented:

```
binomial : (%,%) -> %
factorial : % -> %
permutation : (%,%) -> %
```

— category CFCAT CombinatorialFunctionCategory —

```
)abbrev category CFCAT CombinatorialFunctionCategory
++ Category for the usual combinatorial functions
++ Author: Manuel Bronstein
++ Date Created: ???
++ Date Last Updated: 14 May 1991
++ Description:
++ Category for the usual combinatorial functions;

CombinatorialFunctionCategory(): Category == with
    binomial   : ($, $) -> $
      ++ binomial(n,r) returns the \spad{(n,r)} binomial coefficient
      ++ (often denoted in the literature by \spad{C(n,r)}).
      ++ Note that \spad{C(n,r) = n!/(r!(n-r)!)} where \spad{n >= r >= 0}.
      ++
      ++X [binomial(5,i) for i in 0..5]
    factorial  : $ -> $
      ++ factorial(n) computes the factorial of n
      ++ (denoted in the literature by \spad{n!})
      ++ Note that \spad{n! = n (n-1)! when n > 0}; also, \spad{0! = 1}.
    permutation: ($, $) -> $
      ++ permutation(n, m) returns the number of
      ++ permutations of n objects taken m at a time.
      ++ Note that \spad{permutation(n,m) = n!/(n-m)!}.
```

———-

— **CFCAT.dotabb** —

```
"CFCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=CFCAT"];
"CFCAT" -> "CATEGORY"
```

————-

— **CFCAT.dotfull** —

```
"CombinatorialFunctionCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=CFCAT"];
"CombinatorialFunctionCategory()" -> "Category"
```

————-

— **CFCAT.dotpic** —

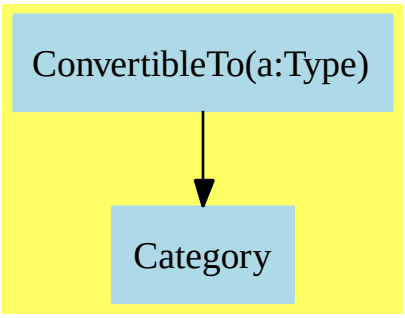
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"CombinatorialFunctionCategory()" [color=lightblue];
"CombinatorialFunctionCategory()" -> "Category"

"Category" [color=lightblue];
}
```

————-

2.0.8 ConvertibleTo (KONVERT)



— **ConvertibleTo.input** —

```
)set break resume
)sys rm -f ConvertibleTo.output
)spool ConvertibleTo.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show ConvertibleTo
--R ConvertibleTo S: Type  is a category constructor
--R Abbreviation for ConvertibleTo is KONVERT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for KONVERT
--R
--R------------------------------ Operations --------------------------------
--R convert : % -> S
--R
--E 1

)spool
)lisp (bye)
```

———-

— **ConvertibleTo.help** —

```
=======================================================================
ConvertibleTo examples
=======================================================================

See Also:
o )show ConvertibleTo
```

———-

See:

⇒ “Collection” (CLAGG) 5.0.70 on page 287
⇒ “FunctionSpace” (FS) 17.0.195 on page 1336
⇒ “IntegerNumberSystem” (INS) 16.0.186 on page 1228
⇒ “MonogenicAlgebra” (MONOGEN) 19.0.214 on page 1598
⇒ “QuotientFieldCategory” (QFCAT) 17.0.198 on page 1376
⇒ “RealConstant” (REAL) 3.0.32 on page 106
⇒ “RealNumberSystem” (RNS) 17.0.200 on page 1400
⇐ “Category” (CATEGORY) 2.0.1 on page 3

Exports:

convert

This is directly exported but not implemented:

```
convert : % -> S
```

— category KONVERT ConvertibleTo —

```
)abbrev category KONVERT ConvertibleTo
++ Category for convert
++ Author: Manuel Bronstein
++ Date Created: ???
++ Date Last Updated: 14 May 1991
++ Description:
++ A is convertible to B means any element of A
++ can be converted into an element of B,
++ but not automatically by the interpreter.

ConvertibleTo(S:Type): Category == with
    convert: % -> S
      ++ convert(a) transforms a into an element of S.
```

———-

— KONVERT.dotabb —

```
"KONVERT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=KONVERT"];
"KONVERT" -> "CATEGORY"
```

———-

— **KONVERT.dotfull** —

```
"ConvertibleTo(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=KONVERT"];
"ConvertibleTo(a:Type)" -> "Category"

"ConvertibleTo(DoubleFloat)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=KONVERT"];
"ConvertibleTo(DoubleFloat)" -> "ConvertibleTo(a:Type)"

"ConvertibleTo(Float)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=KONVERT"];
"ConvertibleTo(Float)" -> "ConvertibleTo(a:Type)"

"ConvertibleTo(InputForm)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=KONVERT"];
"ConvertibleTo(InputForm)" -> "ConvertibleTo(a:Type)"

"ConvertibleTo(Integer)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=KONVERT"];
"ConvertibleTo(Integer)" -> "ConvertibleTo(a:Type)"

"ConvertibleTo(Pattern(Integer))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=KONVERT"];
"ConvertibleTo(Pattern(Integer))" -> "ConvertibleTo(a:Type)"

"ConvertibleTo(Pattern(Float))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=KONVERT"];
"ConvertibleTo(Pattern(Float))" -> "ConvertibleTo(a:Type)"

"ConvertibleTo(Complex(Float))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=KONVERT"];
"ConvertibleTo(Complex(Float))" ->  "ConvertibleTo(a:Type)"

"ConvertibleTo(Complex(DoubleFloat))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=KONVERT"];
"ConvertibleTo(Complex(DoubleFloat))" ->  "ConvertibleTo(a:Type)"

"ConvertibleTo(String)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=KONVERT"];
"ConvertibleTo(String)" ->  "ConvertibleTo(a:Type)"

"ConvertibleTo(Symbol)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=KONVERT"];
"ConvertibleTo(Symbol)" ->  "ConvertibleTo(a:Type)"

"ConvertibleTo(SExpression)"
```

```
 [color=seagreen,href="bookvol10.2.pdf#nameddest=KONVERT"];
"ConvertibleTo(SExpression)" ->  "ConvertibleTo(a:Type)"

"ConvertibleTo(Pattern(Base))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=KONVERT"];
"ConvertibleTo(Pattern(Base))" ->  "ConvertibleTo(a:Type)"

"ConvertibleTo(List(Integer))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=KONVERT"];
"ConvertibleTo(List(Integer))" ->  "ConvertibleTo(a:Type)"

"ConvertibleTo(List(Character))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=KONVERT"];
"ConvertibleTo(List(Character))" ->  "ConvertibleTo(a:Type)"

"ConvertibleTo(UnivariatePolynomialCategory(CommutativeRing))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=KONVERT"];
"ConvertibleTo(UnivariatePolynomialCategory(CommutativeRing))" ->
    "ConvertibleTo(a:Type)"
```

———-

— KONVERT.dotpic —

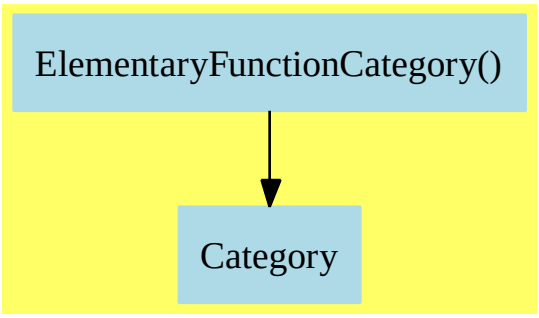
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"ConvertibleTo(a:Type)" [color=lightblue];
"ConvertibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];
}
```

———-

2.0.9 ElementaryFunctionCategory (ELEMFUN)



— **ElementaryFunctionCategory.input** —

```
)set break resume
)sys rm -f ElementaryFunctionCategory.output
)spool ElementaryFunctionCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show ElementaryFunctionCategory
--R ElementaryFunctionCategory  is a category constructor
--R Abbreviation for ElementaryFunctionCategory is ELEMFUN
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for ELEMFUN
--R
--R------------------------------ Operations --------------------------------
--R ?**? : (%,%) -> %                    exp : % -> %
--R log : % -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **ElementaryFunctionCategory.help** —

```
=====================================================================
ElementaryFunctionCategory examples
=====================================================================

See Also:
```

```
o )show ElementaryFunctionCategory
```

———-

See:

⇒ “TranscendentalFunctionCategory” (TRANFUN) 3.0.35 on page 117
⇐ “Category” (CATEGORY) 2.0.1 on page 3

Exports:

?**? exp log

These are directly exported but not implemented:

```
exp : % -> %
log : % -> %
```

These are implemented by this category:

```
?**? : (%,%) -> %
```

— category ELEMFUN ElementaryFunctionCategory —

```
)abbrev category ELEMFUN ElementaryFunctionCategory
++ Category for the elementary functions
++ Author: Manuel Bronstein
++ Date Created: ???
++ Date Last Updated: 14 May 1991
++ Description:
++ Category for the elementary functions;

ElementaryFunctionCategory(): Category == with
    log : $ -> $       ++ log(x) returns the natural logarithm of x.
    exp : $ -> $       ++ exp(x) returns %e to the power x.
    "**": ($, $) -> $  ++ x**y returns x to the power y.
 add
   if $ has Monoid then
     x ** y == exp(y * log x)
```

———-

— ELEMFUN.dotabb —

```
"ELEMFUN"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ELEMFUN"];
```

```
"ELEMFUN" -> "CATEGORY"
```

———-

— **ELEMFUN.dotfull** —

```
"ElementaryFunctionCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ELEMFUN"];
"ElementaryFunctionCategory()" -> "Category"
```

———-

— **ELEMFUN.dotpic** —

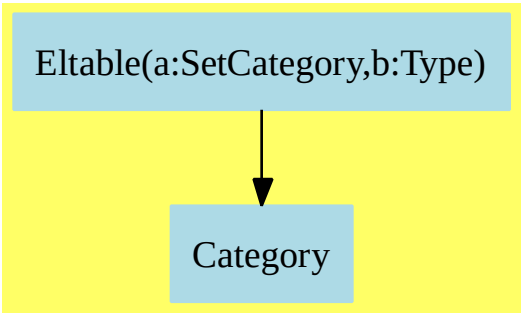
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"ElementaryFunctionCategory()" [color=lightblue];
"ElementaryFunctionCategory()" -> "Category"

"Category" [color=lightblue];
}
```

———-

2.0.10 Eltable (ELTAB)



— **Eltable.input** —

```
)set break resume
)sys rm -f Eltable.output
)spool Eltable.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show Eltable
--R Eltable(S: SetCategory,Index: Type)  is a category constructor
--R Abbreviation for Eltable is ELTAB
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for ELTAB
--R
--R------------------------------ Operations --------------------------------
--R ?.? : (%,S) -> Index
--R
--E 1

)spool
)lisp (bye)
```

———-

— Eltable.help —

```
========================================================================
Eltable examples
========================================================================

See Also:
o )show Eltable
```

———-

See:

⇒ “EltableAggregate” (ELTAGG) 3.0.24 on page 76
⇒ “LinearOrdinaryDifferentialOperatorCategory” (LODOCAT) 11.0.156 on page 971
⇒ “UnivariatePolynomialCategory” (UPOLYC) 17.0.204 on page 1471
⇐ “Category” (CATEGORY) 2.0.1 on page 3

Exports:

?.?

This syntax for elt is supported by the interpreter and compiler.

This is directly exported but not implemented:

```
 ?.? : (%,S) -> Index
```

— **category ELTAB Eltable** —

```
)abbrev category ELTAB Eltable
++ Author: Michael Monagan; revised by Manuel Bronstein and Manuel Bronstein
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ An eltable over domains D and I is a structure which can be viewed
++ as a function from D to I.
++ Examples of eltable structures range from data structures, e.g. those
++ of type \spadtype{List}, to algebraic structures like
++ \spadtype{Polynomial}.

Eltable(S:SetCategory, Index:Type): Category == with
  elt : (%, S) -> Index
     ++ elt(u,i) (also written: u . i) returns the element of u indexed by i.
     ++ Error: if i is not an index of u.
```

———-

— **ELTAB.dotabb** —

```
"ELTAB" [color=lightblue,href="bookvol10.2.pdf#nameddest=ELTAB"];
"ELTAB" -> "CATEGORY"
```

———-

— **ELTAB.dotfull** —

```
"Eltable(a:SetCategory,b:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ELTAB"];
"Eltable(a:SetCategory,b:Type)" -> "Category"

"Eltable(a:UnivariatePolynomialCategory(a:Ring),b:UnivariatePolynomialCategory(a:Ring))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=ELTAB"];
"Eltable(a:UnivariatePolynomialCategory(a:Ring),b:UnivariatePolynomialCategory(a:Ring))" ->
   "Eltable(a:SetCategory,b:Type)"
```

```
"Eltable(a:Ring,b:Ring)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=ELTAB"];
"Eltable(a:Ring,b:Ring)" ->
   "Eltable(a:SetCategory,b:Type)"

"Eltable(a:SetCategory,b:SetCategory)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=ELTAB"];
"Eltable(a:SetCategory,b:SetCategory)" ->
   "Eltable(a:SetCategory,b:Type)"
```

———-

— ELTAB.dotpic —

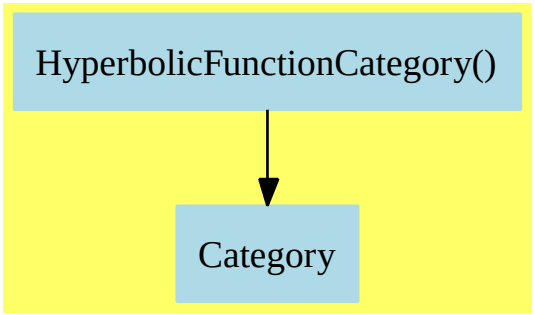
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"Eltable(a:SetCategory,b:Type)" [color=lightblue];
"Eltable(a:SetCategory,b:Type)" -> "Category"

"Category" [color=lightblue];
}
```

———-

2.0.11 HyperbolicFunctionCategory (HYPCAT)



The `csch` and `sech` functions were modified to include an intermediate test to check that the argument has a reciprocal values.

— **HyperbolicFunctionCategory.input** —

```
)set break resume
)sys rm -f HyperbolicFunctionCategory.output
)spool HyperbolicFunctionCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show HyperbolicFunctionCategory
--R HyperbolicFunctionCategory  is a category constructor
--R Abbreviation for HyperbolicFunctionCategory is HYPCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for HYPCAT
--R
--R------------------------------ Operations --------------------------------
--R cosh : % -> %                        coth : % -> %
--R csch : % -> %                        sech : % -> %
--R sinh : % -> %                        tanh : % -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **HyperbolicFunctionCategory.help** —

```
=======================================================================
HyperbolicFunctionCategory examples
=======================================================================

See Also:
o )show HyperbolicFunctionCategory
```

———-

See:

⇒ “TranscendentalFunctionCategory” (TRANFUN) 3.0.35 on page 117
⇐ “Category” (CATEGORY) 2.0.1 on page 3

Exports:

cosh coth csch sech sinh tanh

These are implemented by this category:

```
cosh : % -> %
coth : % -> %
csch : % -> %
sech : % -> %
sinh : % -> %
tanh : % -> %
```

— category HYPCAT HyperbolicFunctionCategory —

```
)abbrev category HYPCAT HyperbolicFunctionCategory
++ Category for the hyperbolic trigonometric functions
++ Author: ???
++ Date Created: ???
++ Date Last Updated: 14 May 1991
++ Description:
++ Category for the hyperbolic trigonometric functions;

HyperbolicFunctionCategory(): Category == with
    cosh: $ -> $       ++ cosh(x) returns the hyperbolic cosine of x.
    coth: $ -> $       ++ coth(x) returns the hyperbolic cotangent of x.
    csch: $ -> $       ++ csch(x) returns the hyperbolic cosecant of x.
    sech: $ -> $       ++ sech(x) returns the hyperbolic secant of x.
    sinh: $ -> $       ++ sinh(x) returns the hyperbolic sine of x.
    tanh: $ -> $       ++ tanh(x) returns the hyperbolic tangent of x.
 add
    if $ has Ring then
       csch x ==
         (a := recip(sinh x)) case "failed" => error "csch: no reciprocal"
         a::$
       sech x ==
         (a := recip(cosh x)) case "failed" => error "sech: no reciprocal"
         a::$
       tanh x == sinh x * sech x
       coth x == cosh x * csch x
       if $ has ElementaryFunctionCategory then
         cosh x ==
           e := exp x
           (e + recip(e)::$) * recip(2::$)::$
         sinh(x):$ ==
           e := exp x
           (e - recip(e)::$) * recip(2::$)::$
```

———-

— HYPCAT.dotabb —

```
"HYPCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=HYPCAT"];
"HYPCAT" -> "CATEGORY"
```

———-

— **HYPCAT.dotfull** —

```
"HyperbolicFunctionCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=HYPCAT"];
"HyperbolicFunctionCategory()" -> "Category"
```

———-

— **HYPCAT.dotpic** —

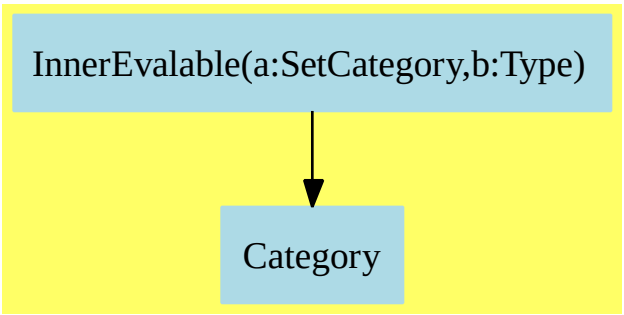
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"HyperbolicFunctionCategory()" [color=lightblue];
"HyperbolicFunctionCategory()" -> "Category"

"Category" [color=lightblue];
}
```

———-

2.0.12 InnerEvalable (IEVALAB)



— **InnerEvalable.input** —

```
)set break resume
)sys rm -f InnerEvalable.output
)spool InnerEvalable.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show InnerEvalable
--R InnerEvalable(A: SetCategory,B: Type)  is a category constructor
--R Abbreviation for InnerEvalable is IEVALAB
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for IEVALAB
--R
--R------------------------------ Operations ------------------------------
--R eval : (%,A,B) -> %                   eval : (%,List A,List B) -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **InnerEvalable.help** —

```
=====================================================================
InnerEvalable examples
=====================================================================

See Also:
o )show InnerEvalable
```

———-

See:

⇒ “DesingTreeCategory” (DSTRCAT) 4.0.38 on page 131
⇒ “Evalable” (EVALAB) 3.0.25 on page 80
⇒ “ExpressionSpace” (ES) 5.0.72 on page 300
⇒ “MultivariateTaylorSeriesCategory” (MTSCAT) 15.0.182 on page 1193
⇒ “PolynomialCategory” (POLYCAT) 16.0.189 on page 1255
⇐ “Category” (CATEGORY) 2.0.1 on page 3

Exports:

eval

These are directly exported but not implemented:

```
eval : (%,A,B) -> %
```

These are implemented by this category:

```
eval : (%,List A,List B) -> %
```

— category IEVALAB InnerEvalable —

```
)abbrev category IEVALAB InnerEvalable
-- FOR THE BENEFIT OF LIBAXO GENERATION
++ Author:
++ Date Created:
++ Date Last Updated: June 3, 1991
++ Basic Operations:
++ Related Domains:
++ Also See: Evalable
++ AMS Classifications:
++ Keywords: equation
++ Examples:
++ References:
++ Description:
++ This category provides \spadfun{eval} operations.
++ A domain may belong to this category if it is possible to make
++ ‘‘evaluation’’ substitutions.  The difference between this
++ and \spadtype{Evalable} is that the operations in this category
++ specify the substitution as a pair of arguments rather than as
++ an equation.

InnerEvalable(A:SetCategory, B:Type): Category == with
    eval: ($, A, B) -> $
       ++ eval(f, x, v) replaces x by v in f.
    eval: ($, List A, List B) -> $
       ++ eval(f, [x1,...,xn], [v1,...,vn]) replaces xi by vi in f.
 add
    eval(f:$, x:A, v:B) == eval(f, [x], [v])
```

———-

— IEVALAB.dotabb —

```
"IEVALAB"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=IEVALAB"];
"IEVALAB" -> "CATEGORY"
```

———-

— IEVALAB.dotfull —

```
"InnerEvalable(a:SetCategory,b:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=IEVALAB"];
"InnerEvalable(a:SetCategory,b:Type)" -> "Category"

"InnerEvalable(a:SetCategory,b:SetCategory)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=IEVALAB"];
"InnerEvalable(a:SetCategory,b:SetCategory)" ->
    "InnerEvalable(a:SetCategory,b:Type)"

"InnerEvalable(a:OrderedSet,b:Ring)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=IEVALAB"];
"InnerEvalable(a:OrderedSet,b:Ring)" ->
    "InnerEvalable(a:SetCategory,b:Type)"

"InnerEvalable(a:OrderedSet,b:PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=IEVALAB"];
"InnerEvalable(a:OrderedSet,b:PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
    "InnerEvalable(a:SetCategory,b:Type)"

"InnerEvalable(a:Ring,MultivariateTaylorSeriesCategory(a:Ring,b:OrderedSet))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=IEVALAB"];
"InnerEvalable(a:Ring,MultivariateTaylorSeriesCategory(a:Ring,b:OrderedSet))"
     -> "InnerEvalable(a:SetCategory,b:Type)"

"InnerEvalable(Kernal(ExpressionSpace),ExpressionSpace)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=IEVALAB"];
"InnerEvalable(Kernal(ExpressionSpace),ExpressionSpace)" ->
    "InnerEvalable(a:SetCategory,b:Type)"
```

———-

— IEVALAB.dotpic —

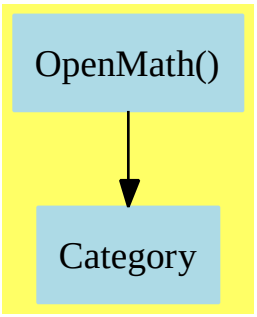
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"InnerEvalable(a:SetCategory,b:Type)" [color=lightblue];
"InnerEvalable(a:SetCategory,b:Type)" -> "Category"

"Category" [color=lightblue];
}
```

———-

2.0.13 OpenMath (OM)



— **OpenMath.input** —

```
)set break resume
)sys rm -f OpenMath.output
)spool OpenMath.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show OpenMath
--R OpenMath  is a category constructor
--R Abbreviation for OpenMath is OM
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for OM
--R
--R------------------------------ Operations --------------------------------
--R OMwrite : % -> String                OMwrite : (%,Boolean) -> String
--R OMwrite : (OpenMathDevice,%) -> Void
--R OMwrite : (OpenMathDevice,%,Boolean) -> Void
--R
--E 1

)spool
)lisp (bye)
```

———-

— **OpenMath.help** —

```
=======================================================================
OpenMath examples
=======================================================================

See Also:
o )show OpenMath
```

———-

See:

⇒ “StringCategory” (STRICAT) 10.0.149 on page 920
⇐ “Category” (CATEGORY) 2.0.1 on page 3

Exports:

OMwrite

These are directly exported but not implemented:

```
OMwrite : % -> String
OMwrite : (%,Boolean) -> String
OMwrite : (OpenMathDevice,%) -> Void
OMwrite : (OpenMathDevice,%,Boolean) -> Void
```

— category OM OpenMath —

```
)abbrev category OM OpenMath
++ Author: Mike Dewar & Vilya Harvey
++ Basic Functions: OMwrite
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ \spadtype{OpenMath} provides operations for exporting an object
++ in OpenMath format.

OpenMath(): Category == with
  OMwrite  : % -> String
  ++ OMwrite(u) returns the OpenMath XML encoding of \axiom{u} as a
  ++ complete OpenMath object.
  OMwrite  : (%, Boolean) -> String
  ++ OMwrite(u, true) returns the OpenMath XML encoding of \axiom{u}
  ++ as a complete OpenMath object; OMwrite(u, false) returns the
  ++ OpenMath XML encoding of \axiom{u} as an OpenMath fragment.
  OMwrite  : (OpenMathDevice, %) -> Void
```

```
  ++ OMwrite(dev, u) writes the OpenMath form of \axiom{u} to the
  ++ OpenMath device \axiom{dev} as a complete OpenMath object.
  OMwrite  : (OpenMathDevice, %, Boolean) -> Void
  ++ OMwrite(dev, u, true) writes the OpenMath form of \axiom{u} to
  ++ the OpenMath device \axiom{dev} as a complete OpenMath object;
  ++ OMwrite(dev, u, false) writes the object as an OpenMath fragment.
```

———-

— **OM.dotabb** —

```
"OM"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OM"];
"OM" -> "CATEGORY"
```

———-

— **OM.dotfull** —

```
"OpenMath()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OM"];
"OpenMath()" -> "Category"
```

———-

— **OM.dotpic** —

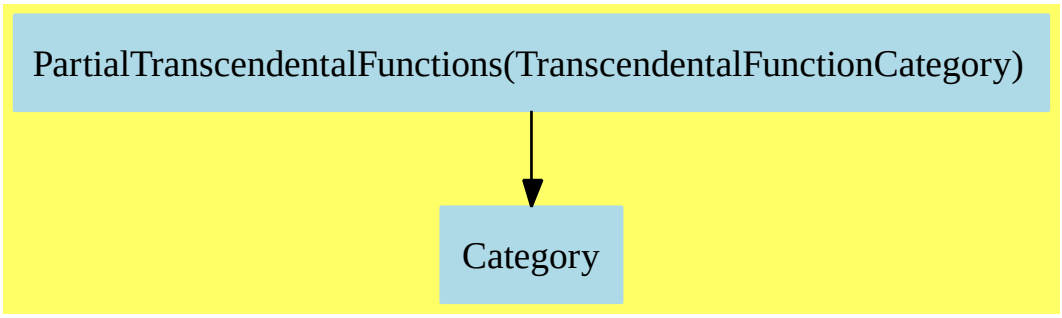
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"OpenMath()" [color=lightblue];
"OpenMath()" -> "Category"

"Category" [color=lightblue];
}
```

———-

2.0.14 PartialTranscendentalFunctions (PTRANFN)



— PartialTranscendentalFunctions.input —

```
)set break resume
)sys rm -f PartialTranscendentalFunctions.output
)spool PartialTranscendentalFunctions.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show PartialTranscendentalFunctions
--R PartialTranscendentalFunctions K: TranscendentalFunctionCategory  is a category constructor
--R Abbreviation for PartialTranscendentalFunctions is PTRANFN
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PTRANFN
--R
--R------------------------------ Operations ------------------------------
--R acosIfCan : K -> Union(K,"failed")    acotIfCan : K -> Union(K,"failed")
--R acscIfCan : K -> Union(K,"failed")    asecIfCan : K -> Union(K,"failed")
--R asinIfCan : K -> Union(K,"failed")    atanIfCan : K -> Union(K,"failed")
--R cosIfCan : K -> Union(K,"failed")     coshIfCan : K -> Union(K,"failed")
--R cotIfCan : K -> Union(K,"failed")     cothIfCan : K -> Union(K,"failed")
--R cscIfCan : K -> Union(K,"failed")     cschIfCan : K -> Union(K,"failed")
--R expIfCan : K -> Union(K,"failed")     logIfCan : K -> Union(K,"failed")
--R secIfCan : K -> Union(K,"failed")     sechIfCan : K -> Union(K,"failed")
--R sinIfCan : K -> Union(K,"failed")     sinhIfCan : K -> Union(K,"failed")
--R tanIfCan : K -> Union(K,"failed")     tanhIfCan : K -> Union(K,"failed")
--R acoshIfCan : K -> Union(K,"failed")
--R acothIfCan : K -> Union(K,"failed")
--R acschIfCan : K -> Union(K,"failed")
--R asechIfCan : K -> Union(K,"failed")
--R asinhIfCan : K -> Union(K,"failed")
--R atanhIfCan : K -> Union(K,"failed")
--R nthRootIfCan : (K,NonNegativeInteger) -> Union(K,"failed")
```

```
--R
--E 1

)spool
)lisp (bye)
```

———-

— **PartialTranscendentalFunctions.help** —

```
=====================================================================
PartialTranscendentalFunctions examples
=====================================================================

See Also:
o )show PartialTranscendentalFunctions
```

———-

See:

⇐ “Category” (CATEGORY) 2.0.1 on page 3

Exports:

acosIfCan	acoshIfCan	acotIfCan	acothIfCan	acscIfCan
acschIfCan	asecIfCan	asechIfCan	asinIfCan	asinhIfCan
atanIfCan	atanhIfCan	cosIfCan	coshIfCan	cotIfCan
cothIfCan	cscIfCan	cschIfCan	expIfCan	logIfCan
nthRootIfCan	secIfCan	sechIfCan	sinIfCan	sinhIfCan
tanIfCan	tanhIfCan			

These are directly exported but not implemented:

```
acosIfCan : K -> Union(K,"failed")
acoshIfCan : K -> Union(K,"failed")
acotIfCan : K -> Union(K,"failed")
acothIfCan : K -> Union(K,"failed")
acscIfCan : K -> Union(K,"failed")
acschIfCan : K -> Union(K,"failed")
asecIfCan : K -> Union(K,"failed")
asechIfCan : K -> Union(K,"failed")
asinIfCan : K -> Union(K,"failed")
asinhIfCan : K -> Union(K,"failed")
atanIfCan : K -> Union(K,"failed")
atanhIfCan : K -> Union(K,"failed")
cosIfCan : K -> Union(K,"failed")
coshIfCan : K -> Union(K,"failed")
```

```
 cotIfCan : K -> Union(K,"failed")
 cothIfCan : K -> Union(K,"failed")
 cscIfCan : K -> Union(K,"failed")
 cschIfCan : K -> Union(K,"failed")
 expIfCan : K -> Union(K,"failed")
 logIfCan : K -> Union(K,"failed")
 nthRootIfCan : (K,NonNegativeInteger) -> Union(K,"failed")
 secIfCan : K -> Union(K,"failed")
 sechIfCan : K -> Union(K,"failed")
 sinIfCan : K -> Union(K,"failed")
 sinhIfCan : K -> Union(K,"failed")
 tanIfCan : K -> Union(K,"failed")
 tanhIfCan : K -> Union(K,"failed")
```

— category PTRANFN PartialTranscendentalFunctions —

```
)abbrev category PTRANFN PartialTranscendentalFunctions
++ Description of a package which provides partial transcendental
++ functions, i.e. functions which return an answer or "failed"
++ Author: Clifton J. Williamson
++ Date Created: 12 February 1990
++ Date Last Updated: 14 February 1990
++ Keywords:
++ Examples:
++ References:
++ Description:
++ This is the description of any package which provides partial
++ functions on a domain belonging to TranscendentalFunctionCategory.

PartialTranscendentalFunctions(K): Category == Definition where
  K :     TranscendentalFunctionCategory
  NNI ==> NonNegativeInteger

  Definition ==> with

--% Exponentials and Logarithms

    nthRootIfCan: (K,NNI) -> Union(K,"failed")
      ++ nthRootIfCan(z,n) returns the nth root of z if possible,
      ++ and "failed" otherwise.
    expIfCan: K -> Union(K,"failed")
      ++ expIfCan(z) returns exp(z) if possible, and "failed" otherwise.
    logIfCan: K -> Union(K,"failed")
      ++ logIfCan(z) returns log(z) if possible, and "failed" otherwise.

--% TrigonometricFunctionCategory

    sinIfCan : K -> Union(K,"failed")
      ++ sinIfCan(z) returns sin(z) if possible, and "failed" otherwise.
```

```
    cosIfCan: K -> Union(K,"failed")
      ++ cosIfCan(z) returns cos(z) if possible, and "failed" otherwise.
    tanIfCan: K -> Union(K,"failed")
      ++ tanIfCan(z) returns tan(z) if possible, and "failed" otherwise.
    cotIfCan: K -> Union(K,"failed")
      ++ cotIfCan(z) returns cot(z) if possible, and "failed" otherwise.
    secIfCan: K -> Union(K,"failed")
      ++ secIfCan(z) returns sec(z) if possible, and "failed" otherwise.
    cscIfCan: K -> Union(K,"failed")
      ++ cscIfCan(z) returns csc(z) if possible, and "failed" otherwise.

--% ArcTrigonometricFunctionCategory

    asinIfCan: K -> Union(K,"failed")
      ++ asinIfCan(z) returns asin(z) if possible, and "failed" otherwise.
    acosIfCan: K -> Union(K,"failed")
      ++ acosIfCan(z) returns acos(z) if possible, and "failed" otherwise.
    atanIfCan: K -> Union(K,"failed")
      ++ atanIfCan(z) returns atan(z) if possible, and "failed" otherwise.
    acotIfCan: K -> Union(K,"failed")
      ++ acotIfCan(z) returns acot(z) if possible, and "failed" otherwise.
    asecIfCan: K -> Union(K,"failed")
      ++ asecIfCan(z) returns asec(z) if possible, and "failed" otherwise.
    acscIfCan: K -> Union(K,"failed")
      ++ acscIfCan(z) returns acsc(z) if possible, and "failed" otherwise.

--% HyperbolicFunctionCategory

    sinhIfCan: K -> Union(K,"failed")
      ++ sinhIfCan(z) returns sinh(z) if possible, and "failed" otherwise.
    coshIfCan: K -> Union(K,"failed")
      ++ coshIfCan(z) returns cosh(z) if possible, and "failed" otherwise.
    tanhIfCan: K -> Union(K,"failed")
      ++ tanhIfCan(z) returns tanh(z) if possible, and "failed" otherwise.
    cothIfCan: K -> Union(K,"failed")
      ++ cothIfCan(z) returns coth(z) if possible, and "failed" otherwise.
    sechIfCan: K -> Union(K,"failed")
      ++ sechIfCan(z) returns sech(z) if possible, and "failed" otherwise.
    cschIfCan: K -> Union(K,"failed")
      ++ cschIfCan(z) returns csch(z) if possible, and "failed" otherwise.

--% ArcHyperbolicFunctionCategory

    asinhIfCan: K -> Union(K,"failed")
      ++ asinhIfCan(z) returns asinh(z) if possible, and "failed" otherwise.
    acoshIfCan: K -> Union(K,"failed")
      ++ acoshIfCan(z) returns acosh(z) if possible, and "failed" otherwise.
    atanhIfCan: K -> Union(K,"failed")
      ++ atanhIfCan(z) returns atanh(z) if possible, and "failed" otherwise.
    acothIfCan: K -> Union(K,"failed")
```

```
      ++ acothIfCan(z) returns acoth(z) if possible, and "failed" otherwise.
    asechIfCan: K -> Union(K,"failed")
      ++ asechIfCan(z) returns asech(z) if possible, and "failed" otherwise.
    acschIfCan: K -> Union(K,"failed")
      ++ acschIfCan(z) returns acsch(z) if possible, and "failed" otherwise.
```

———-

— **PTRANFN.dotabb** —

```
"PTRANFN"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PTRANFN"];
"PTRANFN" -> "CATEGORY"
```

———-

— **PTRANFN.dotfull** —

```
"PartialTranscendentalFunctions(TranscendentalFunctionCategory)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PTRANFN"];
"PartialTranscendentalFunctions(TranscendentalFunctionCategory)" ->
   "Category()"
```

———-

— **PTRANFN.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

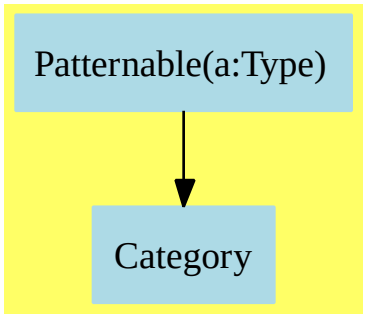
"PartialTranscendentalFunctions(TranscendentalFunctionCategory)"
 [color=lightblue];
"PartialTranscendentalFunctions(TranscendentalFunctionCategory)" ->
   "Category"

"Category" [color=lightblue];

}
```

———-

2.0.15 Patternable (PATAB)



— **Patternable.input** —

```
)set break resume
)sys rm -f Patternable.output
)spool Patternable.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show Patternable
--R Patternable R: Type  is a category constructor
--R Abbreviation for Patternable is PATAB
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PATAB
--R
--R------------------------------ Operations --------------------------------
--R convert : % -> Pattern Integer if R has KONVERT PATTERN INT
--R convert : % -> Pattern Float if R has KONVERT PATTERN FLOAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— **Patternable.help** —

```
=====================================================================
Patternable examples
=====================================================================

See Also:
```

```
o )show Patternable
```

———-

See:

⇒ “ComplexCategory” (COMPCAT) 20.0.216 on page 1619
⇒ “FunctionSpace” (FS) 17.0.195 on page 1336
⇒ “IntegerNumberSystem” (INS) 16.0.186 on page 1228
⇒ “QuotientFieldCategory” (QFCAT) 17.0.198 on page 1376
⇐ “Category” (CATEGORY) 2.0.1 on page 3

Exports:

convert

These exports come from (p23) ConvertibleTo(Pattern(Integer)):

```
convert : % -> Pattern Integer if R has KONVERT PATTERN INT
```

These exports come from (p23) ConvertibleTo(Pattern(Float)):

```
convert : % -> Pattern Float if R has KONVERT PATTERN FLOAT
```

— category PATAB Patternable —

```
)abbrev category PATAB Patternable
++ Category of sets that can be converted to useful patterns
++ Author: Manuel Bronstein
++ Date Created: 29 Nov 1989
++ Date Last Updated: 29 Nov 1989
++ Keywords: pattern, matching.
++ Description:
++ An object S is Patternable over an object R if S can
++ lift the conversions from R into \spadtype{Pattern(Integer)} and
++ \spadtype{Pattern(Float)} to itself;

Patternable(R:Type): Category == with
  if R has ConvertibleTo Pattern Integer then
           ConvertibleTo Pattern Integer
  if R has ConvertibleTo Pattern Float then
           ConvertibleTo Pattern Float
```

———-

— PATAB.dotabb —

```
"PATAB"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PATAB"];
"PATAB" -> "CATEGORY"
```

———-

— **PATAB.dotfull** —

```
"Patternable(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PATAB"];
"Patternable(a:Type)" -> "Category"

"Patternable(IntegralDomain)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=PATAB"];
"Patternable(IntegralDomain)" -> "Patternable(a:Type)"

"Patternable(OrderedSet)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=PATAB"];
"Patternable(OrderedSet)" -> "Patternable(a:Type)"

"Patternable(CommutativeRing)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=PATAB"];
"Patternable(CommutativeRing)" -> "Patternable(a:Type)"
```

———-

— **PATAB.dotpic** —

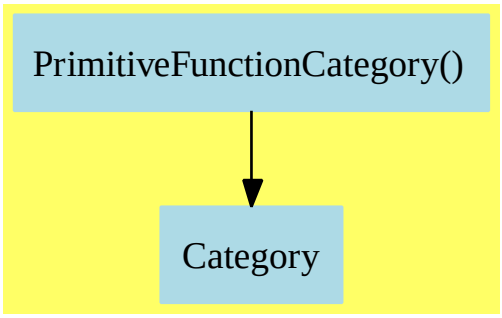
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"Patternable(a:Type)" [color=lightblue];
"Patternable(a:Type)" -> "Category"

"Category" [color=lightblue];
}
```

———-

2.0.16 PrimitiveFunctionCategory (PRIMCAT)



— **PrimitiveFunctionCategory.input** —

```
)set break resume
)sys rm -f PrimitiveFunctionCategory.output
)spool PrimitiveFunctionCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show PrimitiveFunctionCategory
--R PrimitiveFunctionCategory  is a category constructor
--R Abbreviation for PrimitiveFunctionCategory is PRIMCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PRIMCAT
--R
--R------------------------------ Operations --------------------------------
--R integral : (%,Symbol) -> %
--R integral : (%,SegmentBinding %) -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **PrimitiveFunctionCategory.help** —

```
==========================================================================
PrimitiveFunctionCategory examples
==========================================================================

See Also:
```

```
o )show PrimitiveFunctionCategory
```

See:

⇒ "LiouvillianFunctionCategory" (LFCAT) 4.0.51 on page 192
⇐ "Category" (CATEGORY) 2.0.1 on page 3

Exports:

integral

These are directly exported but not implemented:

```
integral : (%,Symbol) -> %
integral : (%,SegmentBinding %) -> %
```

— category PRIMCAT PrimitiveFunctionCategory —

```
)abbrev category PRIMCAT PrimitiveFunctionCategory
++ Category for the integral functions
++ Author: Manuel Bronstein
++ Date Created: ???
++ Date Last Updated: 14 May 1991
++ Description:
++ Category for the functions defined by integrals;

PrimitiveFunctionCategory(): Category == with
    integral: ($, Symbol) -> $
      ++ integral(f, x) returns the formal integral of f dx.
    integral: ($, SegmentBinding $) -> $
      ++ integral(f, x = a..b) returns the formal definite integral
      ++ of f dx for x between \spad{a} and b.
```

— PRIMCAT.dotabb —

```
"PRIMCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PRIMCAT"];
"PRIMCAT" -> "CATEGORY"
```

— PRIMCAT.dotfull —

```
"PrimitiveFunctionCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PRIMCAT"];
"PrimitiveFunctionCategory()" -> "Category"
```

———-

— PRIMCAT.dotpic —

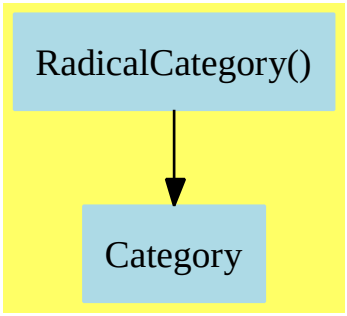
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"PrimitiveFunctionCategory()" [color=lightblue];
"PrimitiveFunctionCategory()" -> "Category"

"Category" [color=lightblue];
}
```

———-

2.0.17 RadicalCategory (RADCAT)



— RadicalCategory.input —

```
)set break resume
)sys rm -f RadicalCategory.output
)spool RadicalCategory.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 1
)show RadicalCategory
--R RadicalCategory  is a category constructor
--R Abbreviation for RadicalCategory is RADCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for RADCAT
--R
--R------------------------------ Operations --------------------------------
--R ?**? : (%,Fraction Integer) -> %      nthRoot : (%,Integer) -> %
--R sqrt : % -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— RadicalCategory.help —

```
========================================================================
RadicalCategory examples
========================================================================

See Also:
o )show RadicalCategory
```

———-

See:

⇒ “AlgebraicallyClosedField” (ACF) 17.0.191 on page 1293
⇒ “IntervalCategory” (INTCAT) 14.0.176 on page 1150
⇒ “RealClosedField” (RCFIELD) 17.0.199 on page 1390
⇒ “RealNumberSystem” (RNS) 17.0.200 on page 1400
⇒ “UnivariateLaurentSeriesCategory” (ULSCAT) 17.0.202 on page 1448
⇒ “UnivariatePuiseuxSeriesCategory” (UPXSCAT) 17.0.203 on page 1460
⇒ “UnivariateTaylorSeriesCategory” (UTSCAT) 16.0.190 on page 1277
⇐ “Category” (CATEGORY) 2.0.1 on page 3

Exports:

nthRoot sqrt ?**?

These are directly exported but not implemented:

```
?**? : (%,Fraction Integer) -> %
```

These are implemented by this category:

```
 nthRoot : (%,Integer) -> %
 sqrt : % -> %
```

— category RADCAT RadicalCategory —

```
)abbrev category RADCAT RadicalCategory
++ Author:
++ Date Created:
++ Change History:
++ Basic Operations: nthRoot, sqrt, **
++ Related Constructors:
++ Keywords: rational numbers
++ Description:
++ The \spad{RadicalCategory} is a model for the rational numbers.

RadicalCategory(): Category == with
  sqrt   : % -> %
      ++ sqrt(x) returns the square root of x.
  nthRoot: (%, Integer) -> %
      ++ nthRoot(x,n) returns the nth root of x.
  _*_*   : (%, Fraction Integer) -> %
      ++ x ** y is the rational exponentiation of x by the power y.
 add
  sqrt x        == x ** inv(2::Fraction(Integer))
  nthRoot(x, n) == x ** inv(n::Fraction(Integer))
```

———-

— RADCAT.dotabb —

```
"RADCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RADCAT"];
"RADCAT" -> "CATEGORY"
```

———-

— RADCAT.dotfull —

```
"RadicalCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RADCAT"];
"RadicalCategory()" -> "Category"
```

———-

— **RADCAT.dotpic** —

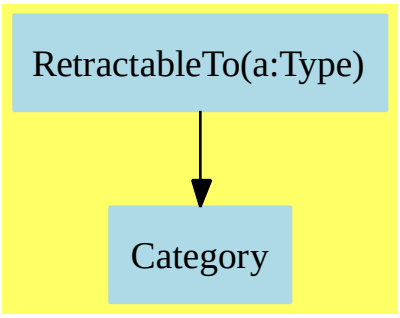
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"RadicalCategory()" [color=lightblue];
"RadicalCategory()" -> "Category"

"Category" [color=lightblue];
}
```

———-

2.0.18 RetractableTo (RETRACT)



— **RetractableTo.input** —

```
)set break resume
)sys rm -f RetractableTo.output
)spool RetractableTo.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show RetractableTo
--R RetractableTo S: Type  is a category constructor
--R Abbreviation for RetractableTo is RETRACT
```

```
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for RETRACT
--R
--R------------------------------ Operations ------------------------------
--R coerce : S -> %                       retract : % -> S
--R retractIfCan : % -> Union(S,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— RetractableTo.help —

```
=====================================================================
RetractableTo examples
=====================================================================

See Also:
o )show RetractableTo
```

———-

See:

⇒ “DifferentialPolynomialCategory” (DPOLCAT) 17.0.192 on page 1306
⇒ “DifferentialVariableCategory” (DVARCAT) 5.0.71 on page 294
⇒ “ExtensionField” (XF) 18.0.206 on page 1510
⇒ “ExpressionSpace” (ES) 5.0.72 on page 300
⇒ “FiniteAlgebraicExtensionField” (FAXF) 19.0.213 on page 1583
⇒ “FortranMachineTypeCategory” (FMTC) 13.0.171 on page 1108
⇒ “FreeAbelianMonoidCategory” (FAMONC) 7.0.103 on page 564
⇒ “FreeModuleCat” (FMCAT) 10.0.140 on page 858
⇒ “FullyRetractableTo” (FRETRCT) 3.0.27 on page 87
⇒ “FunctionSpace” (FS) 17.0.195 on page 1336
⇒ “GradedAlgebra” (GRALG) 5.0.73 on page 313
⇒ “IntegerNumberSystem” (INS) 16.0.186 on page 1228
⇒ “IntervalCategory” (INTCAT) 14.0.176 on page 1150
⇒ “PolynomialCategory” (POLYCAT) 16.0.189 on page 1255
⇒ “QuotientFieldCategory” (QFCAT) 17.0.198 on page 1376
⇒ “RealNumberSystem” (RNS) 17.0.200 on page 1400
⇒ “UnivariateLaurentSeriesConstructorCategory” (ULSCCAT) 18.0.211 on page 1553
⇒ “UnivariatePuiseuxSeriesConstructorCategory” (UPXSCCA) 18.0.212 on page 1570
⇒ “XFreeAlgebra” (XFALG) 11.0.159 on page 992
⇐ “Category” (CATEGORY) 2.0.1 on page 3

Exports:

coerce retract retractIfCan

These are directly exported but not implemented:

```
coerce : S -> %
retractIfCan : % -> Union(S,"failed")
```

These are implemented by this category:

```
retract : % -> S
```

— category RETRACT RetractableTo —

```
)abbrev category RETRACT RetractableTo
++ Category for retract
++ Author: ???
++ Date Created: ???
++ Date Last Updated: 14 May 1991
++ Description:
++ A is retractable to B means that some elementsif A can be converted
++ into elements of B and any element of B can be converted into an
++ element of A.

RetractableTo(S: Type): Category == with
    coerce:       S -> %
      ++ coerce(a) transforms a into an element of %.
    retractIfCan: % -> Union(S,"failed")
      ++ retractIfCan(a) transforms a into an element of S if possible.
      ++ Returns "failed" if a cannot be made into an element of S.
    retract:      % -> S
      ++ retract(a) transforms a into an element of S if possible.
      ++ Error: if a cannot be made into an element of S.
  add
    retract(s) ==
      (u:=retractIfCan s) case "failed" => error "not retractable"
      u
```

———-

— RETRACT.dotabb —

```
"RETRACT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RETRACT"];
"RETRACT" -> "CATEGORY"
```

———-

— RETRACT.dotfull —

```
"RetractableTo(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RETRACT"];
"RetractableTo(a:Type)" -> "Category"

"RetractableTo(SetCategory)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=RETRACT"];
"RetractableTo(SetCategory)" -> "RetractableTo(a:Type)"

"RetractableTo(OrderedSet)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=RETRACT"];
"RetractableTo(OrderedSet)" -> "RetractableTo(a:Type)"

"RetractableTo(Symbol)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=RETRACT"];
"RetractableTo(Symbol)" -> "RetractableTo(a:Type)"

"RetractableTo(Integer)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=RETRACT"];
"RetractableTo(Integer)" -> "RetractableTo(a:Type)"

"RetractableTo(NonNegativeInteger)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=RETRACT"];
"RetractableTo(NonNegativeInteger)" -> "RetractableTo(a:Type)"

"RetractableTo(Fraction(Integer))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=RETRACT"];
"RetractableTo(Fraction(Integer))" -> "RetractableTo(a:Type)"

"RetractableTo(Float)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=RETRACT"];
"RetractableTo(Float)" -> "RetractableTo(a:Type)"

"RetractableTo(Kernel(ExpressionSpace))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=RETRACT"];
"RetractableTo(Kernel(ExpressionSpace))" -> "RetractableTo(a:Type)"

"RetractableTo(CommutativeRing)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=RETRACT"];
"RetractableTo(CommutativeRing)" -> "RetractableTo(a:Type)"

"RetractableTo(UnivariatePuiseuxSeriesCategory(Ring))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=RETRACT"];
"RetractableTo(UnivariatePuiseuxSeriesCategory(Ring))"
  -> "RetractableTo(a:Type)"
```

```
"RetractableTo(Field)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=RETRACT"];
"RetractableTo(Field)" -> "RetractableTo(a:Type)"

"RetractableTo(IntegralDomain)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=RETRACT"];
"RetractableTo(IntegralDomain)" -> "RetractableTo(a:Type)"

"RetractableTo(OrderedFreeMonoid(OrderedSet))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=RETRACT"];
"RetractableTo(OrderedFreeMonoid(OrderedSet))" -> "RetractableTo(a:Type)"
```

———-

— RETRACT.dotpic —

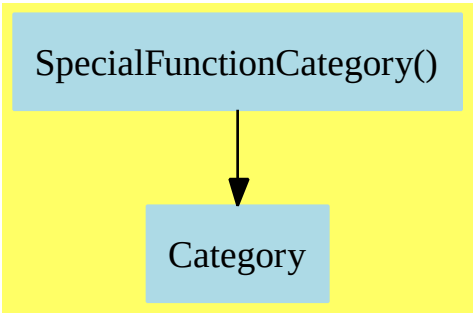
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"RetractableTo(a:Type)" [color=lightblue];
"RetractableTo(a:Type)" -> "Category"

"Category" [color=lightblue];
}
```

———-

2.0.19 SpecialFunctionCategory (SPFCAT)



— SpecialFunctionCategory.input —

```
)set break resume
)sys rm -f SpecialFunctionCategory.output
)spool SpecialFunctionCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show SpecialFunctionCategory
--R SpecialFunctionCategory  is a category constructor
--R Abbreviation for SpecialFunctionCategory is SPFCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for SPFCAT
--R
--R------------------------------ Operations --------------------------------
--R Beta : (%,%) -> %                    Gamma : % -> %
--R Gamma : (%,%) -> %                   abs : % -> %
--R airyAi : % -> %                      airyBi : % -> %
--R besselI : (%,%) -> %                 besselJ : (%,%) -> %
--R besselK : (%,%) -> %                 besselY : (%,%) -> %
--R digamma : % -> %                     polygamma : (%,%) -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— SpecialFunctionCategory.help —

```
=====================================================================
SpecialFunctionCategory examples
=====================================================================

See Also:
o )show SpecialFunctionCategory
```

———-

See:

⇐ “Category” (CATEGORY) 2.0.1 on page 3

Exports:

abs	airyAi	airyBi	besselI	besselJ
besselK	besselY	Beta	digamma	Gamma
polygamma				

These are directly exported but not implemented:

```
 abs : % -> %
 airyAi : % -> %
 airyBi : % -> %
 besselI : (%,%) -> %
 besselJ : (%,%) -> %
 besselK : (%,%) -> %
 besselY : (%,%) -> %
 Beta : (%,%) -> %
 digamma : % -> %
 Gamma : % -> %
 Gamma : (%,%) -> %
 polygamma : (%,%) -> %
```

— category SPFCAT SpecialFunctionCategory —

```
)abbrev category SPFCAT SpecialFunctionCategory
++ Category for the other special functions
++ Author: Manuel Bronstein
++ Date Created: ???
++ Date Last Updated: 11 May 1993
++ Description:
++ Category for the other special functions;

SpecialFunctionCategory(): Category == with
    abs :      $ -> $
        ++ abs(x) returns the absolute value of x.
    Gamma:     $ -> $
        ++ Gamma(x) is the Euler Gamma function.
    Beta:      ($,$)->$
        ++ Beta(x,y) is \spad{Gamma(x) * Gamma(y)/Gamma(x+y)}.
    digamma:   $ -> $
        ++ digamma(x) is the logarithmic derivative of \spad{Gamma(x)}
        ++ (often written \spad{psi(x)} in the literature).
    polygamma: ($, $) -> $
        ++ polygamma(k,x) is the \spad{k-th} derivative of \spad{digamma(x)},
        ++ (often written \spad{psi(k,x)} in the literature).
    Gamma:     ($, $) -> $
        ++ Gamma(a,x) is the incomplete Gamma function.
    besselJ:   ($,$) -> $
        ++ besselJ(v,z) is the Bessel function of the first kind.
    besselY:   ($,$) -> $
        ++ besselY(v,z) is the Bessel function of the second kind.
    besselI:   ($,$) -> $
        ++ besselI(v,z) is the modified Bessel function of the first kind.
    besselK:   ($,$) -> $
        ++ besselK(v,z) is the modified Bessel function of the second kind.
    airyAi:    $ -> $
```

```
        ++ airyAi(x) is the Airy function \spad{Ai(x)}.
    airyBi:    $ -> $
        ++ airyBi(x) is the Airy function \spad{Bi(x)}.
```

———-

— **SPFCAT.dotabb** —

```
"SPFCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SPFCAT"];
"SPFCAT" -> "CATEGORY"
```

———-

— **SPFCAT.dotfull** —

```
"SpecialFunctionCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SPFCAT"];
"SpecialFunctionCategory()" -> "Category"
```

———-

— **SPFCAT.dotpic** —

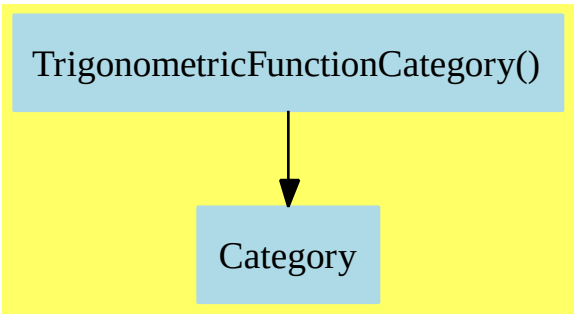
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"SpecialFunctionCategory()" [color=lightblue];
"SpecialFunctionCategory()" -> "Category"

"Category" [color=lightblue];
}
```

———-

2.0.20 TrigonometricFunctionCategory (TRIGCAT)



The csc and sec functions were modified to include an intermediate test to check that the argument has a reciprocal values.

— **TrigonometricFunctionCategory.input** —

```
)set break resume
)sys rm -f TrigonometricFunctionCategory.output
)spool TrigonometricFunctionCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show TrigonometricFunctionCategory
--R TrigonometricFunctionCategory  is a category constructor
--R Abbreviation for TrigonometricFunctionCategory is TRIGCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for TRIGCAT
--R
--R------------------------------ Operations --------------------------------
--R cos : % -> %                          cot : % -> %
--R csc : % -> %                          sec : % -> %
--R sin : % -> %                          tan : % -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **TrigonometricFunctionCategory.help** —

```
==========================================================================
```

```
TrigonometricFunctionCategory examples
======================================================================

See Also:
o )show TrigonometricFunctionCategory
```

———-

See:

⇒ “TranscendentalFunctionCategory” (TRANFUN) 3.0.35 on page 117
⇐ “Category” (CATEGORY) 2.0.1 on page 3

Exports:

cos cot csc sec sin tan

These are directly exported but not implemented:

```
cos : % -> %
sin : % -> %
```

These are implemented by this category:

```
cot : % -> %
csc : % -> %
sec : % -> %
tan : % -> %
```

— category TRIGCAT TrigonometricFunctionCategory —

```
)abbrev category TRIGCAT TrigonometricFunctionCategory
++ Category for the trigonometric functions
++ Author: ???
++ Date Created: ???
++ Date Last Updated: 14 May 1991
++ Description:
++ Category for the trigonometric functions;

TrigonometricFunctionCategory(): Category == with
    cos: $ -> $        ++ cos(x) returns the cosine of x.
    cot: $ -> $        ++ cot(x) returns the cotangent of x.
    csc: $ -> $        ++ csc(x) returns the cosecant of x.
    sec: $ -> $        ++ sec(x) returns the secant of x.
    sin: $ -> $        ++ sin(x) returns the sine of x.
    tan: $ -> $        ++ tan(x) returns the tangent of x.
 add
    if $ has Ring then
```

```
       csc x ==
         (a := recip(sin x)) case "failed" => error "csc: no reciprocal"
         a::$
       sec x ==
         (a := recip(cos x)) case "failed" => error "sec: no reciprocal"
         a::$
       tan x == sin x * sec x
       cot x == cos x * csc x
```

———-

— **TRIGCAT.dotabb** —

```
"TRIGCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=TRIGCAT"];
"TRIGCAT" -> "CATEGORY"
```

———-

— **TRIGCAT.dotfull** —

```
"TrigonometricFunctionCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=TRIGCAT"];
"TrigonometricFunctionCategory()" -> "Category"
```

———-

— **TRIGCAT.dotpic** —

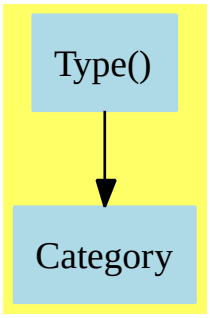
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"TrigonometricFunctionCategory()" [color=lightblue];
"TrigonometricFunctionCategory()" -> "Category"

"Category" [color=lightblue];
}
```

———-

2.0.21 Type (TYPE)



See:

⇒ “Aggregate” (AGG) 3.0.22 on page 69
⇒ “FortranProgramCategory” (FORTCAT) 3.0.26 on page 84
⇒ “FullyPatternMatchable” (FPATMAB) 3.0.28 on page 91
⇒ “SegmentCategory” (SEGCAT) 3.0.33 on page 109
⇐ “Category” (CATEGORY) 2.0.1 on page 3

Attributes exported:

- **nil**

— category TYPE Type —

```
)abbrev category TYPE Type
++ The new fundamental Type (keeping Object for 1.5 as well)
++ Author: Richard Jenks
++ Date Created: 14 May 1992
++ Date Last Updated: 14 May 1992
++ Description:
++ The fundamental Type;

Type(): Category == with nil
```

———-

— TYPE.dotabb —

```
"TYPE" [color=lightblue,href="bookvol10.2.pdf#nameddest=TYPE"];
"TYPE" -> "CATEGORY"
```

———-

— **TYPE.dotfull** —

```
"Type()" [color=lightblue,href="bookvol10.2.pdf#nameddest=TYPE"];
"Type()" -> "Category"
```

———-

— **TYPE.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"Type()" [color=lightblue];
"Type()" -> "Category"

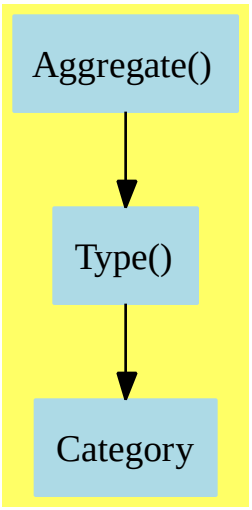
"Category" [color=lightblue];
}
```

———-

Chapter 3

Category Layer 2

3.0.22 Aggregate (AGG)



— **Aggregate.input** —

```
)set break resume
)sys rm -f Aggregate.output
)spool Aggregate.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
```

```
)show Aggregate
--R Aggregate  is a category constructor
--R Abbreviation for Aggregate is AGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for AGG
--R
--R------------------------------ Operations --------------------------------
--R copy : % -> %                          empty : () -> %
--R empty? : % -> Boolean                  eq? : (%,%) -> Boolean
--R sample : () -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R less? : (%,NonNegativeInteger) -> Boolean
--R more? : (%,NonNegativeInteger) -> Boolean
--R size? : (%,NonNegativeInteger) -> Boolean
--R
--E 1

)spool
)lisp (bye)
```

———-

— **Aggregate.help** —

```
=====================================================================
Aggregate examples
=====================================================================

See Also:
o )show Aggregate
```

———-

See:

⇒ “HomogeneousAggregate” (HOAGG) 4.0.49 on page 181
⇐ “Type” (TYPE) 2.0.21 on page 66

Attributes exported:

- **nil**

Exports:

empty? eq? less? more? sample size?

Attributes Used:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.

These are implemented by this category:

```
empty? : % -> Boolean
eq? : (%,%) -> Boolean
less? : (%,NonNegativeInteger) -> Boolean
more? : (%,NonNegativeInteger) -> Boolean
sample : () -> %
size? : (%,NonNegativeInteger) -> Boolean
```

— category AGG Aggregate —

```
)abbrev category AGG Aggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The notion of aggregate serves to model any data structure aggregate,
++ designating any collection of objects, with heterogenous or homogeneous
++ members, with a finite or infinite number of members, explicitly or
++ implicitly represented. An aggregate can in principle represent
++ everything from a string of characters to abstract sets such
++ as "the set of x satisfying relation r(x)"
++ An attribute \spadatt{finiteAggregate} is used to assert that a domain
++ has a finite number of elements.

Aggregate: Category == Type with
   eq?: (%,%) -> Boolean
     ++ eq?(u,v) tests if u and v are same objects.
   copy: % -> %
     ++ copy(u) returns a top-level (non-recursive) copy of u.
     ++ Note that for collections, \axiom{copy(u) == [x for x in u]}.
   empty: () -> %
     ++ empty()$D creates an aggregate of type D with 0 elements.
     ++ Note that The $D can be dropped if understood by context,
     ++ e.g. \axiom{u: D := empty()}.
   empty?: % -> Boolean
     ++ empty?(u) tests if u has 0 elements.
   less?: (%,NonNegativeInteger) -> Boolean
     ++ less?(u,n) tests if u has less than n elements.
   more?: (%,NonNegativeInteger) -> Boolean
     ++ more?(u,n) tests if u has greater than n elements.
   size?: (%,NonNegativeInteger) -> Boolean
     ++ size?(u,n) tests if u has exactly n elements.
```

```
   sample: constant -> %
     ++ sample yields a value of type %
   if % has finiteAggregate then
     "#": % -> NonNegativeInteger
       ++ # u returns the number of items in u.
 add
  eq?(a,b) == EQ(a,b)$Lisp
  sample() == empty()
  if % has finiteAggregate then
    empty? a   == #a = 0
    less?(a,n) == #a < n
    more?(a,n) == #a > n
    size?(a,n) == #a = n
```

———-

— **AGG.dotabb** —

```
"AGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=AGG"];
"AGG" -> "TYPE"
```

———-

— **AGG.dotfull** —

```
"Aggregate()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=AGG"];
"Aggregate()" -> "Type()"
```

———-

— **AGG.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"Aggregate()" [color=lightblue];
"Aggregate()" -> "Type()"

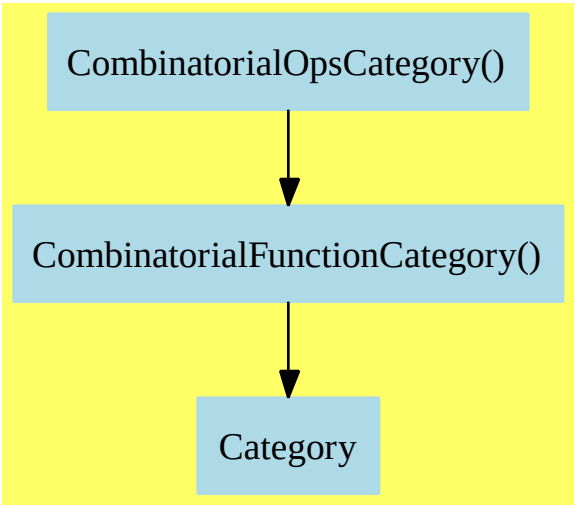
"Type()" [color=lightblue];
"Type()" -> "Category"

"Category" [color=lightblue];
```

```
}
```

———-

3.0.23 CombinatorialOpsCategory (COMBOPC)



— **CombinatorialOpsCategory.input** —

```
)set break resume
)sys rm -f CombinatorialOpsCategory.output
)spool CombinatorialOpsCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show CombinatorialOpsCategory
--R CombinatorialOpsCategory  is a category constructor
--R Abbreviation for CombinatorialOpsCategory is COMBOPC
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for COMBOPC
--R
--R------------------------------ Operations --------------------------------
--R binomial : (%,%) -> %                factorial : % -> %
--R factorials : (%,Symbol) -> %         factorials : % -> %
--R permutation : (%,%) -> %             product : (%,Symbol) -> %
```

```
--R summation : (%,Symbol) -> %
--R product : (%,SegmentBinding %) -> %
--R summation : (%,SegmentBinding %) -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **CombinatorialOpsCategory.help** —

```
=====================================================================
CombinatorialOpsCategory examples
=====================================================================

See Also:
o )show CombinatorialOpsCategory
```

———-

See:

⇐ “CombinatorialFunctionCategory” (CFCAT) 2.0.7 on page 20

Exports:

binomial	factorial	factorials	permutation	product
summation				

These are directly exported but not implemented:

```
factorials : % -> %
factorials : (%,Symbol) -> %
product : (%,Symbol) -> %
product : (%,SegmentBinding %) -> %
summation : (%,Symbol) -> %
summation : (%,SegmentBinding %) -> %
```

These exports come from (p20) CombinatorialFunctionCategory():

```
binomial : (%,%) -> %
factorial : % -> %
permutation : (%,%) -> %
```

— **category COMBOPC CombinatorialOpsCategory** —

```
)abbrev category COMBOPC CombinatorialOpsCategory
++ Category for summations and products
++ Author: Manuel Bronstein
++ Date Created: ???
++ Date Last Updated: 22 February 1993 (JHD/BMT)
++ Description:
++ CombinatorialOpsCategory is the category obtaining by adjoining
++ summations and products to the usual combinatorial operations;

CombinatorialOpsCategory(): Category ==
  CombinatorialFunctionCategory with
    factorials : $ -> $
      ++ factorials(f) rewrites the permutations and binomials in f
      ++ in terms of factorials;
    factorials : ($, Symbol) -> $
      ++ factorials(f, x) rewrites the permutations and binomials in f
      ++ involving x in terms of factorials;
    summation  : ($, Symbol)            -> $
      ++ summation(f(n), n) returns the formal sum S(n) which verifies
      ++ S(n+1) - S(n) = f(n);
    summation  : ($, SegmentBinding $)  -> $
      ++ summation(f(n), n = a..b) returns f(a) + ... + f(b) as a
      ++ formal sum;
    product    : ($, Symbol)            -> $
      ++ product(f(n), n) returns the formal product P(n) which verifies
      ++ P(n+1)/P(n) = f(n);
    product    : ($, SegmentBinding  $) -> $
      ++ product(f(n), n = a..b) returns f(a) * ... * f(b) as a
      ++ formal product;
```

———-

— COMBOPC.dotabb —

```
"COMBOPC"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=COMBOPC"];
"COMBOPC" -> "CFCAT"
```

———-

— COMBOPC.dotfull —

```
"CombinatorialOpsCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=COMBOPC"];
"CombinatorialOpsCategory()" -> "CombinatorialFunctionCategory()"
```

———-

— **COMBOPC.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

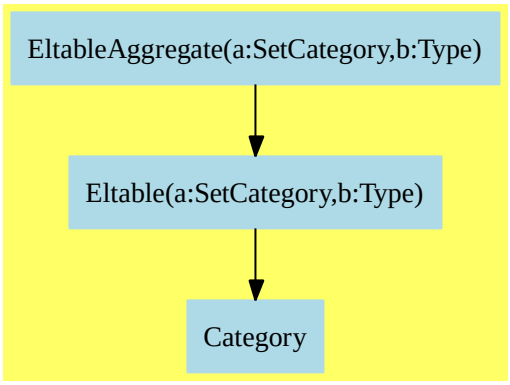
"CombinatorialOpsCategory()" [color=lightblue];
"CombinatorialOpsCategory()" -> "CombinatorialFunctionCategory()"

"CombinatorialFunctionCategory()" [color=lightblue];
"CombinatorialFunctionCategory()" -> "Category"

"Category" [color=lightblue];
}
```

———-

3.0.24 EltableAggregate (ELTAGG)



— **EltableAggregate.input** —

```
)set break resume
)sys rm -f EltableAggregate.output
)spool EltableAggregate.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 1
)show EltableAggregate
--R EltableAggregate(Dom: SetCategory,Im: Type)  is a category constructor
--R Abbreviation for EltableAggregate is ELTAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for ELTAGG
--R
--R------------------------------ Operations --------------------------------
--R elt : (%,Dom,Im) -> Im                ?.? : (%,Dom) -> Im
--R qelt : (%,Dom) -> Im
--R qsetelt! : (%,Dom,Im) -> Im if $ has shallowlyMutable
--R setelt : (%,Dom,Im) -> Im if $ has shallowlyMutable
--R
--E 1

)spool
)lisp (bye)
```

———-

— **EltableAggregate.help** —

```
=====================================================================
EltableAggregate examples
=====================================================================

See Also:
o )show EltableAggregate
```

———-

See:

⇒ “IndexedAggregate” (IXAGG) 5.0.74 on page 318
⇐ “Eltable” (ELTAB) 2.0.10 on page 29

Exports:

elt qelt qsetelt! setelt ?.?

Attributes Used:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.

These are part of this category’s direct exports:

```
elt : (%,Dom,Im) -> Im
setelt : (%,Dom,Im) -> Im if $ has shallowlyMutable
```

These are implemented by this category:

```
qelt : (%,Dom) -> Im
qsetelt! : (%,Dom,Im) -> Im if $ has shallowlyMutable
```

These exports come from (p29) Eltable():

```
?.? : (%,Dom) -> Im
```

— category ELTAGG EltableAggregate —

```
)abbrev category ELTAGG EltableAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ An eltable aggregate is one which can be viewed as a function.
++ For example, the list \axiom{[1,7,4]} can applied to 0,1, and 2
++ respectively will return the integers 1,7, and 4; thus this list may
++ be viewed as mapping 0 to 1, 1 to 7 and 2 to 4. In general, an aggregate
++ can map members of a domain Dom to an image domain Im.

EltableAggregate(Dom:SetCategory, Im:Type): Category ==
-- This is separated from Eltable
-- and series won't have to support qelt's and setelt's.
  Eltable(Dom, Im) with
    elt : (%, Dom, Im) -> Im
       ++ elt(u, x, y) applies u to x if x is in the domain of u,
       ++ and returns y otherwise.
       ++ For example, if u is a polynomial in \axiom{x} over the rationals,
       ++ \axiom{elt(u,n,0)} may define the coefficient of \axiom{x}
       ++ to the power n, returning 0 when n is out of range.
    qelt: (%, Dom) -> Im
       ++ qelt(u, x) applies \axiom{u} to \axiom{x} without checking whether
       ++ \axiom{x} is in the domain of \axiom{u}.  If \axiom{x} is not
       ++ in the domain of \axiom{u} a memory-access violation may occur.
       ++ If a check on whether \axiom{x} is in the domain of \axiom{u}
       ++ is required, use the function \axiom{elt}.
    if % has shallowlyMutable then
```

```
       setelt : (%, Dom, Im) -> Im
           ++ setelt(u,x,y) sets the image of x to be y under u,
           ++ assuming x is in the domain of u.
           ++ Error: if x is not in the domain of u.
           -- this function will soon be renamed as setelt!.
       qsetelt_!: (%, Dom, Im) -> Im
           ++ qsetelt!(u,x,y) sets the image of \axiom{x} to be \axiom{y}
           ++ under \axiom{u}, without checking that \axiom{x} is in
           ++ the domain of \axiom{u}.
           ++ If such a check is required use the function \axiom{setelt}.
 add
  qelt(a, x) == elt(a, x)
  if % has shallowlyMutable then
    qsetelt_!(a, x, y) == (a.x := y)
```

———-

— **ELTAGG.dotabb** —

```
"ELTAGG"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ELTAGG"];
"ELTAGG" -> "ELTAB"
```

———-

— **ELTAGG.dotfull** —

```
"EltableAggregate(a:SetCategory,b:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ELTAGG"];
"EltableAggregate(a:SetCategory,b:Type)" -> "Eltable(a:SetCategory,b:Type)"
```

———-

— **ELTAGG.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

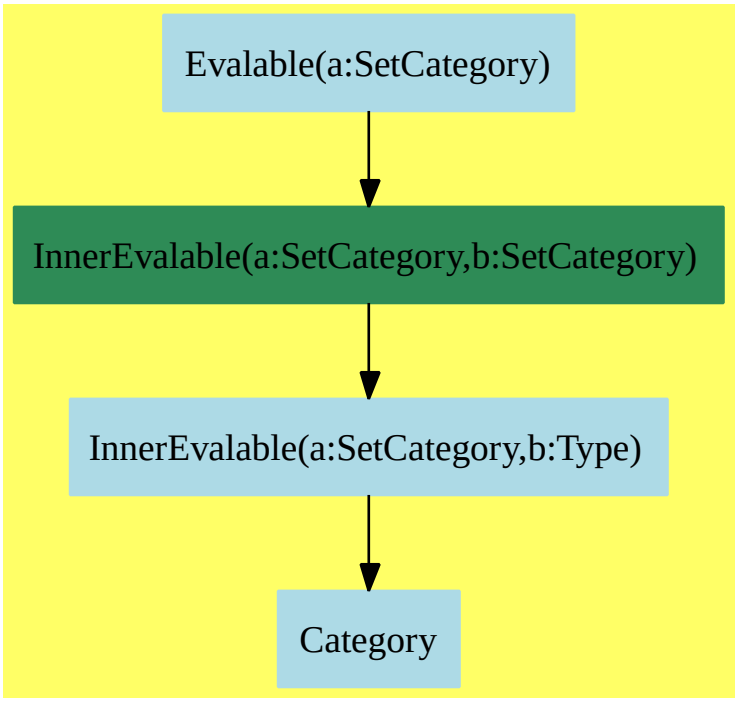
"EltableAggregate(a:SetCategory,b:Type)" [color=lightblue];
"EltableAggregate(a:SetCategory,b:Type)" -> "Eltable(a:SetCategory,b:Type)"

"Eltable(a:SetCategory,b:Type)" [color=lightblue];
"Eltable(a:SetCategory,b:Type)" -> "Category"
```

```
"Category" [color=lightblue];
}
```

———-

3.0.25 Evalable (EVALAB)



— Evalable.input —

```
)set break resume
)sys rm -f Evalable.output
)spool Evalable.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show Evalable
--R Evalable R: SetCategory  is a category constructor
```

```
--R Abbreviation for Evalable is EVALAB
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for EVALAB
--R
--R------------------------------ Operations --------------------------------
--R eval : (%,List Equation R) -> %       eval : (%,Equation R) -> %
--R eval : (%,R,R) -> %                   eval : (%,List R,List R) -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **Evalable.help** —

```
=====================================================================
Evalable examples
=====================================================================

See Also:
o )show Evalable
```

———-

See:

⇒ “ExpressionSpace” (ES) 5.0.72 on page 300
⇒ “MultivariateTaylorSeriesCategory” (MTSCAT) 15.0.182 on page 1193
⇒ “PolynomialCategory” (POLYCAT) 16.0.189 on page 1255
⇐ “InnerEvalable” (IEVALAB) 2.0.12 on page 35

Exports:

eval

These are directly exported but not implemented:

```
eval : (%,List Equation R) -> %
```

These are implemented by this category:

```
eval : (%,Equation R) -> %
eval : (%,List R,List R) -> %
```

These exports come from (p35) InnerEvalable(R:SetCategory,R:SetCategory):

```
 eval : (%,R,R) -> %
```

— category EVALAB Evalable —

```
)abbrev category EVALAB Evalable
++ Author:
++ Date Created:
++ Date Last Updated: June 3, 1991
++ Basic Operations:
++ Related Domains:
++ Also See: FullyEvalable
++ AMS Classifications:
++ Keywords: equation
++ Examples:
++ References:
++ Description:
++ This category provides \spadfun{eval} operations.
++ A domain may belong to this category if it is possible to make
++ ‘‘evaluation’’ substitutions.

Evalable(R:SetCategory): Category == InnerEvalable(R,R) with
    eval: ($, Equation R) -> $
       ++ eval(f,x = v) replaces x by v in f.
    eval: ($, List Equation R) -> $
       ++ eval(f, [x1 = v1,...,xn = vn]) replaces xi by vi in f.
 add
    eval(f:$, eq:Equation R) == eval(f, [eq])
    eval(f:$, xs:List R,vs:List R) == eval(f,[x=v for x in xs for v in vs])
```

———-

— EVALAB.dotabb —

```
"EVALAB"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=EVALAB"];
"EVALAB" -> "IEVALAB"
```

———-

— EVALAB.dotfull —

```
"Evalable(a:SetCategory)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=EVALAB"];
"Evalable(a:SetCategory)" -> "InnerEvalable(a:SetCategory,b:SetCategory)"
```

```
"Evalable(MultivariateTaylorSeriesCategory(a:Ring,b:OrderedSet))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=EVALAB"];
"Evalable(MultivariateTaylorSeriesCategory(a:Ring,b:OrderedSet))" ->
    "Evalable(a:SetCategory)"

"Evalable(ExpressionSpace)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=EVALAB"];
"Evalable(ExpressionSpace)" -> "Evalable(a:SetCategory)"

"Evalable(PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=EVALAB"];
"Evalable(PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet))"
  -> "Evalable(a:SetCategory)"
```

———-

— EVALAB.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"Evalable(a:SetCategory)" [color=lightblue];
"Evalable(a:SetCategory)" -> "InnerEvalable(a:SetCategory,b:SetCategory)"

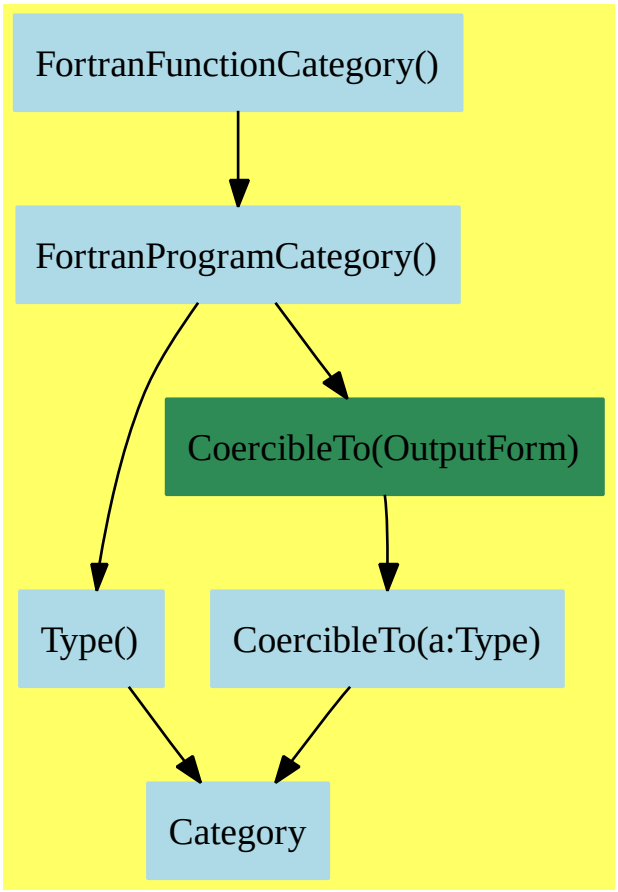
"InnerEvalable(a:SetCategory,b:SetCategory)" [color=seagreen];
"InnerEvalable(a:SetCategory,b:SetCategory)" ->
    "InnerEvalable(a:SetCategory,b:Type)"

"InnerEvalable(a:SetCategory,b:Type)" [color=lightblue];
"InnerEvalable(a:SetCategory,b:Type)" -> "Category"

"Category" [color=lightblue];
}
```

———-

3.0.26 FortranProgramCategory (FORTCAT)



— **FortranProgramCategory.input** —

```
)set break resume
)sys rm -f FortranProgramCategory.output
)spool FortranProgramCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FortranProgramCategory
--R FortranProgramCategory  is a category constructor
--R Abbreviation for FortranProgramCategory is FORTCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FORTCAT
```

```
--R
--R------------------------------ Operations --------------------------------
--R coerce : % -> OutputForm              outputAsFortran : % -> Void
--R
--E 1

)spool
)lisp (bye)
```

———-

— **FortranProgramCategory.help** —

```
=====================================================================
FortranProgramCategory examples
=====================================================================

See Also:
o )show FortranProgramCategory
```

———-

See:

⇒ “FortranFunctionCategory” (FORTFN) 4.0.39 on page 136
⇒ “FortranMatrixCategory” (FMC) 4.0.40 on page 141
⇒ “FortranMatrixFunctionCategory” (FMFUN) 4.0.41 on page 145
⇒ “FortranVectorCategory” (FVC) 4.0.42 on page 150
⇒ “FortranVectorFunctionCategory” (FVFUN) 4.0.43 on page 154
⇐ “CoercibleTo” (KOERCE) 2.0.6 on page 17
⇐ “Type” (TYPE) 2.0.21 on page 66

Exports:

coerce outputAsFortran

Attributes:

- **nil**

These are directly exported but not implemented:

```
outputAsFortran : % -> Void
```

These exports come from (p17) CoercibleTo(OutputForm):

```
coerce : % -> OutputForm
```

— category FORTCAT FortranProgramCategory —

```
)abbrev category FORTCAT FortranProgramCategory
++ Author: Mike Dewar
++ Date Created: November 1992
++ Date Last Updated:
++ Basic Operations:
++ Related Constructors: FortranType, FortranCode, Switch
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ \axiomType{FortranProgramCategory} provides various models of
++ FORTRAN subprograms.  These can be transformed into actual FORTRAN code.

FortranProgramCategory():Category == Join(Type,CoercibleTo OutputForm) with
    outputAsFortran : $ -> Void
    ++ \axiom{outputAsFortran(u)} translates \axiom{u} into a legal FORTRAN
    ++ subprogram.
```

———-

— FORTCAT.dotabb —

```
"FORTCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FORTCAT"];
"FORTCAT" -> "KOERCE"
"FORTCAT" -> "TYPE"
```

———-

— FORTCAT.dotfull —

```
"FortranProgramCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FORTCAT"];
"FortranProgramCategory()" -> "Type()"
"FortranProgramCategory()" -> "CoercibleTo(OutputForm)"
```

———-

— FORTCAT.dotpic —

```
digraph pic {
```

```
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FortranProgramCategory()" [color=lightblue];
"FortranProgramCategory()" -> "Type()"
"FortranProgramCategory()" -> "CoercibleTo(OutputForm)"

"Type()" [color=lightblue];
"Type()" -> "Category"

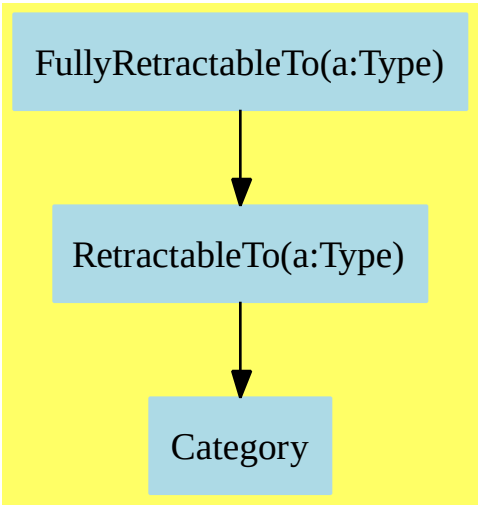
"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];
}
```

———-

3.0.27 FullyRetractableTo (FRETRCT)



— **FullyRetractableTo.input** —

```
)set break resume
)sys rm -f FullyRetractableTo.output
)spool FullyRetractableTo.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FullyRetractableTo
--R FullyRetractableTo S: Type  is a category constructor
--R Abbreviation for FullyRetractableTo is FRETRCT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FRETRCT
--R
--R------------------------------ Operations --------------------------------
--R coerce : S -> %                       retract : % -> S
--R coerce : Integer -> % if S has RETRACT INT
--R coerce : Fraction Integer -> % if S has RETRACT FRAC INT
--R retract : % -> Integer if S has RETRACT INT
--R retract : % -> Fraction Integer if S has RETRACT FRAC INT
--R retractIfCan : % -> Union(Integer,"failed") if S has RETRACT INT
--R retractIfCan : % -> Union(Fraction Integer,"failed") if S has RETRACT FRAC INT
--R retractIfCan : % -> Union(S,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **FullyRetractableTo.help** —

```
====================================================================
FullyRetractableTo examples
====================================================================

See Also:
o )show FullyRetractableTo
```

———-

See:

⇒ “ComplexCategory” (COMPCAT) 20.0.216 on page 1619
⇒ “DirectProductCategory” (DIRPCAT) 12.0.160 on page 1001
⇒ “FiniteAbelianMonoidRing” (FAMR) 14.0.175 on page 1141
⇒ “FunctionSpace” (FS) 17.0.195 on page 1336

⇒ “MonogenicAlgebra” (MONOGEN) 19.0.214 on page 1598
⇒ “OctonionCategory” (OC) 12.0.166 on page 1059
⇒ “QuaternionCategory” (QUATCAT) 12.0.167 on page 1071
⇒ “RealClosedField” (RCFIELD) 17.0.199 on page 1390
⇒ “SquareMatrixCategory” (SMATCAT) 12.0.168 on page 1082
⇒ “UnivariateSkewPolynomialCategory” (OREPCAT) 10.0.150 on page 929
⇐ “RetractableTo” (RETRACT) 2.0.18 on page 55

Exports:

coerce retract retractIfCan

These are implemented by this category:

```
coerce : Integer -> %
    if S has RETRACT INT
coerce : Fraction Integer -> %
    if S has RETRACT FRAC INT
retract : % -> Integer
    if S has RETRACT INT
retract : % -> Fraction Integer
    if S has RETRACT FRAC INT
retractIfCan : % -> Union(Integer,"failed")
    if S has RETRACT INT
retractIfCan : % -> Union(Fraction Integer,"failed")
    if S has RETRACT FRAC INT
```

These exports come from (p55) RetractableTo(S:Type):

```
coerce : S -> %
retract : % -> S
retractIfCan : % -> Union(S,"failed")
```

— category FRETRCT FullyRetractableTo —

```
)abbrev category FRETRCT FullyRetractableTo
++ Author: Manuel Bronstein
++ Date Created: March 1990
++ Date Last Updated: 9 April 1991
++ Description:
++ A is fully retractable to B means that A is retractable to B and
++ if B is retractable to the integers or rational numbers then so is A.
++ In particular, what we are asserting is that there are no integers
++ (rationals) in A which don't retract into B.

FullyRetractableTo(S: Type): Category == RetractableTo(S) with
    if (S has RetractableTo Integer) then RetractableTo Integer
    if (S has RetractableTo Fraction Integer) then
```

```
            RetractableTo Fraction Integer
  add
    if not(S is Integer) then
      if (S has RetractableTo Integer) then    -- induction
        coerce(n:Integer):%  == n::S::%
        retract(r:%):Integer == retract(retract(r)@S)

        retractIfCan(r:%):Union(Integer, "failed") ==
          (u:= retractIfCan(r)@Union(S,"failed")) case "failed"=> "failed"
          retractIfCan(u::S)

    if not(S is Fraction Integer) then
      if (S has RetractableTo Fraction Integer) then   -- induction
        coerce(n:Fraction Integer):%  == n::S::%
        retract(r:%):Fraction(Integer) == retract(retract(r)@S)

        retractIfCan(r:%):Union(Fraction Integer, "failed") ==
          (u:=retractIfCan(r)@Union(S,"failed")) case "failed"=>"failed"
          retractIfCan(u::S)
```

———-

— **FRETRCT.dotabb** —

```
"FRETRCT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FRETRCT"];
"FRETRCT" -> "RETRACT"
```

———-

— **FRETRCT.dotfull** —

```
"FullyRetractableTo(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FRETRCT"];
"FullyRetractableTo(a:Type)" -> "RetractableTo(a:Type)"

"FullyRetractableTo(a:Ring)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=FRETRCT"];
"FullyRetractableTo(a:Ring)" -> "FullyRetractableTo(a:Type)"

"FullyRetractableTo(a:CommutativeRing)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=FRETRCT"];
"FullyRetractableTo(a:CommutativeRing)" -> "FullyRetractableTo(a:Type)"

"FullyRetractableTo(a:SetCategory)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=FRETRCT"];
```

```
"FullyRetractableTo(a:SetCategory)" -> "FullyRetractableTo(a:Type)"

"FullyRetractableTo(Fraction(Integer))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=FRETRCT"];
"FullyRetractableTo(Fraction(Integer))" -> "FullyRetractableTo(a:Type)"
```

———-

— **FRETRCT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

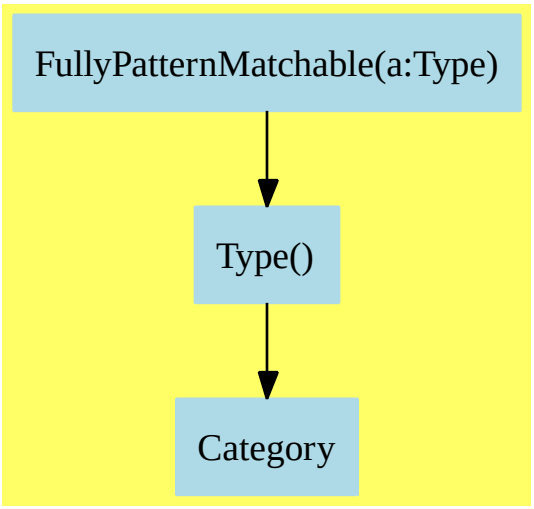
"FullyRetractableTo(a:Type)" [color=lightblue];
"FullyRetractableTo(a:Type)" -> "RetractableTo(a:Type)"

"RetractableTo(a:Type)" [color=lightblue];
"RetractableTo(a:Type)" -> "Category"

"Category" [color=lightblue];
}
```

———-

3.0.28 FullyPatternMatchable (FPATMAB)



— FullyPatternMatchable.input —

```
)set break resume
)sys rm -f FullyPatternMatchable.output
)spool FullyPatternMatchable.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FullyPatternMatchable
--R FullyPatternMatchable R: Type  is a category constructor
--R Abbreviation for FullyPatternMatchable is FPATMAB
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FPATMAB
--R
--R------------------------------ Operations ------------------------------
--R ?=? : (%,%) -> Boolean if R has PATMAB INT or R has PATMAB FLOAT
--R coerce : % -> OutputForm if R has PATMAB INT or R has PATMAB FLOAT
--R hash : % -> SingleInteger if R has PATMAB INT or R has PATMAB FLOAT
--R latex : % -> String if R has PATMAB INT or R has PATMAB FLOAT
--R patternMatch : (%,Pattern Integer,PatternMatchResult(Integer,%)) -> PatternMatchResult(Integ
--R patternMatch : (%,Pattern Float,PatternMatchResult(Float,%)) -> PatternMatchResult(Float,%)
--R ?~=? : (%,%) -> Boolean if R has PATMAB INT or R has PATMAB FLOAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— FullyPatternMatchable.help —

```
=====================================================================
FullyPatternMatchable examples
=====================================================================

See Also:
o )show FullyPatternMatchable
```

———-

See:

⇒ “ComplexCategory” (COMPCAT) 20.0.216 on page 1619

⇒ “FunctionSpace” (FS) 17.0.195 on page 1336
⇒ “QuotientFieldCategory” (QFCAT) 17.0.198 on page 1376
⇐ “Type” (TYPE) 2.0.21 on page 66

Attributes exported:

- **nil**

Exports:

coerce hash latex patternMatch ?=?
?~=?

These exports come from (p226) PatternMatchable(Integer):

```
coerce : % -> OutputForm
    if R has PATMAB INT
    or R has PATMAB FLOAT
hash : % -> SingleInteger
    if R has PATMAB INT
    or R has PATMAB FLOAT
latex : % -> String
    if R has PATMAB INT
    or R has PATMAB FLOAT
patternMatch :
  (%,Pattern Integer,PatternMatchResult(Integer,%))
     -> PatternMatchResult(Integer,%) if R has PATMAB INT
?=? : (%,%) -> Boolean
    if R has PATMAB INT
    or R has PATMAB FLOAT
?~=? : (%,%) -> Boolean
    if R has PATMAB INT
    or R has PATMAB FLOAT
```

These exports come from (p226) PatternMatchable(Float):

```
patternMatch :
  (%,Pattern Float,PatternMatchResult(Float,%))
     -> PatternMatchResult(Float,%) if R has PATMAB FLOAT
```

These exports come from (p66) Type():

— category FPATMAB FullyPatternMatchable —

```
)abbrev category FPATMAB FullyPatternMatchable
++ Category of sets that can be pattern-matched on
```

```
++ Author: Manuel Bronstein
++ Date Created: 28 Nov 1989
++ Date Last Updated: 29 Nov 1989
++ Keywords: pattern, matching.
++ Description:
++ A set S is PatternMatchable over R if S can lift the
++ pattern-matching functions of S over the integers and float
++ to itself (necessary for matching in towers).

FullyPatternMatchable(R:Type): Category == Type with
  if R has PatternMatchable Integer then PatternMatchable Integer
  if R has PatternMatchable Float   then PatternMatchable Float
```

———-

— **FPATMAB.dotabb** —

```
"FPATMAB"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FPATMAB"];
"FPATMAB" -> "TYPE"
```

———-

— **FPATMAB.dotfull** —

```
"FullyPatternMatchable(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FPATMAB"];
"FullyPatternMatchable(a:Type)" -> "Type()"

"FullyPatternMatchable(IntegralDomain)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=FPATMAB"];
"FullyPatternMatchable(IntegralDomain)" ->
  "FullyPatternMatchable(a:Type)"

"FullyPatternMatchable(OrderedSet)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=FPATMAB"];
"FullyPatternMatchable(OrderedSet)" ->
  "FullyPatternMatchable(a:Type)"

"FullyPatternMatchable(CommutativeRing)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=FPATMAB"];
"FullyPatternMatchable(CommutativeRing)" ->
  "FullyPatternMatchable(a:Type)"
```

———-

— **FPATMAB.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

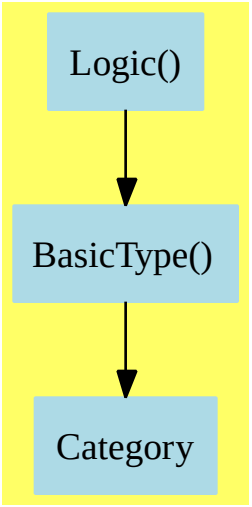
"FullyPatternMatchable(a:Type)" [color=lightblue];
"FullyPatternMatchable(a:Type)" -> "Type()"

"Type()" [color=lightblue];
"Type()" -> "Category"

"Category" [color=lightblue];
}
```

———-

3.0.29 Logic (LOGIC)



— **Logic.input** —

```
)set break resume
)sys rm -f Logic.output
)spool Logic.output
```

```
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show Logic
--R Logic  is a category constructor
--R Abbreviation for Logic is LOGIC
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for LOGIC
--R
--R------------------------------ Operations ------------------------------
--R ?/\? : (%,%) -> %                    ?=? : (%,%) -> Boolean
--R ?\/? : (%,%) -> %                    ~? : % -> %
--R ?~=? : (%,%) -> Boolean
--R
--E 1

)spool
)lisp (bye)
```

———-

— **Logic.help** —

```
==========================================================================
Logic examples
==========================================================================

See Also:
o )show Logic
```

———-

See:

⇒ “BitAggregate” (BTAGG) 9.0.123 on page 731
⇐ “BasicType” (BASTYPE) 2.0.5 on page 14

Exports:

?/\? ?=? ?\/? ~? ?~=?

These are directly exported but not implemented:

```
~? : % -> %
?/\? : (%,%) -> %
```

These are implemented by this category:

```
 ?\/? : (%,%) -> %
```

These exports come from (p14) BasicType():

```
 ?=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
```

— category LOGIC Logic —

```
)abbrev category LOGIC Logic
++ Author:
++ Date Created:
++ Change History:
++ Basic Operations: ~, /\, \/
++ Related Constructors:
++ Keywords: boolean
++ Description:
++ 'Logic' provides the basic operations for lattices, e.g., boolean algebra.

Logic: Category == BasicType with
       _~:        % -> %
        ++ ~(x) returns the logical complement of x.
       _/_\:       (%, %) -> %
        ++ \spadignore { /\ }returns the logical 'meet', e.g. 'and'.
       _\_/:       (%, %) -> %
        ++ \spadignore{ \/ } returns the logical 'join', e.g. 'or'.
  add
    _\_/(x: %,y: %) == _~( _/_\(_~(x), _~(y)))
```

———-

— LOGIC.dotabb —

```
"LOGIC"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=LOGIC"];
"LOGIC" -> "BASTYPE"
```

———-

— LOGIC.dotfull —

```
"Logic()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=LOGIC"];
"Logic()" -> "BasicType()"
```

———-

— **LOGIC.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

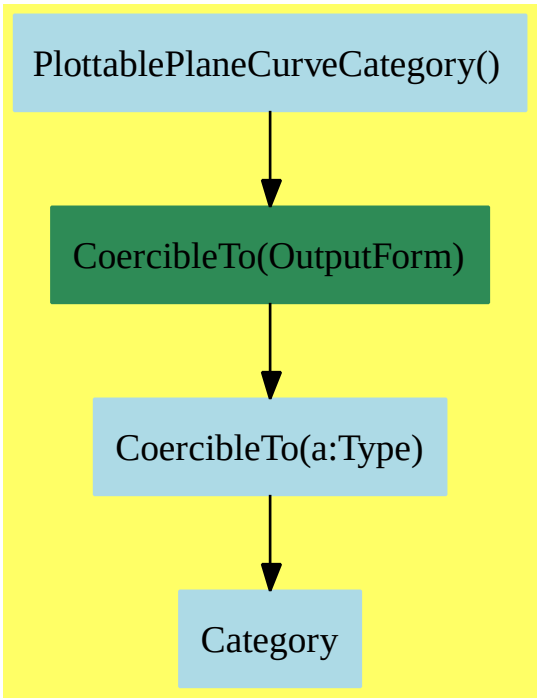
"Logic()" [color=lightblue];
"Logic()" -> "BasicType()"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"Category" [color=lightblue];
}
```

———-

3.0.30 PlottablePlaneCurveCategory (PPCURVE)



— PlottablePlaneCurveCategory.input —

```
)set break resume
)sys rm -f PlottablePlaneCurveCategory.output
)spool PlottablePlaneCurveCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show PlottablePlaneCurveCategory
--R PlottablePlaneCurveCategory  is a category constructor
--R Abbreviation for PlottablePlaneCurveCategory is PPCURVE
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PPCURVE
--R
--R------------------------------ Operations --------------------------------
--R coerce : % -> OutputForm              xRange : % -> Segment DoubleFloat
--R yRange : % -> Segment DoubleFloat
--R listBranches : % -> List List Point DoubleFloat
--R
--E 1

)spool
)lisp (bye)
```

———-

— PlottablePlaneCurveCategory.help —

```
==========================================================================
PlottablePlaneCurveCategory examples
==========================================================================

See Also:
o )show PlottablePlaneCurveCategory
```

———-

See:

⇐ “CoercibleTo” (KOERCE) 2.0.6 on page 17

Exports:

coerce listBranches xRange yRange

These are directly exported but not implemented:

```
listBranches : % -> List List Point DoubleFloat
xRange : % -> Segment DoubleFloat
yRange : % -> Segment DoubleFloat
```

These exports come from (p17) CoercibleTo(OutputForm):

```
coerce : % -> OutputForm
```

— category PPCURVE PlottablePlaneCurveCategory —

```
)abbrev category PPCURVE PlottablePlaneCurveCategory
++ Author: Clifton J. Williamson
++ Date Created: 11 January 1990
++ Date Last Updated: 15 June 1990
++ Basic Operations: listBranches, xRange, yRange
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords: plot, graphics
++ References:
++ Description:
++ PlottablePlaneCurveCategory is the category of curves in the
++ plane which may be plotted via the graphics facilities.  Functions are
++ provided for obtaining lists of lists of points, representing the
++ branches of the curve, and for determining the ranges of the
++ x-coordinates and y-coordinates of the points on the curve.

PlottablePlaneCurveCategory(): Category == Definition where
  L     ==> List
  SEG   ==> Segment
  SF    ==> DoubleFloat
  POINT ==> Point DoubleFloat

  Definition ==> CoercibleTo OutputForm with

    listBranches: % -> L L POINT
      ++ listBranches(c) returns a list of lists of points, representing the
      ++ branches of the curve c.
    xRange: % -> SEG SF
      ++ xRange(c) returns the range of the x-coordinates of the points
      ++ on the curve c.
    yRange: % -> SEG SF
      ++ yRange(c) returns the range of the y-coordinates of the points
      ++ on the curve c.
```

———-

— **PPCURVE.dotabb** —

```
"PPCURVE"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PPCURVE"];
"PPCURVE" -> "KOERCE"
```

———-

— **PPCURVE.dotfull** —

```
"PlottablePlaneCurveCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PPCURVE"];
"PlottablePlaneCurveCategory()" -> "CoercibleTo(OutputForm)"
```

———-

— **PPCURVE.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"PlottablePlaneCurveCategory()" [color=lightblue];
"PlottablePlaneCurveCategory()" -> "CoercibleTo(OutputForm)"

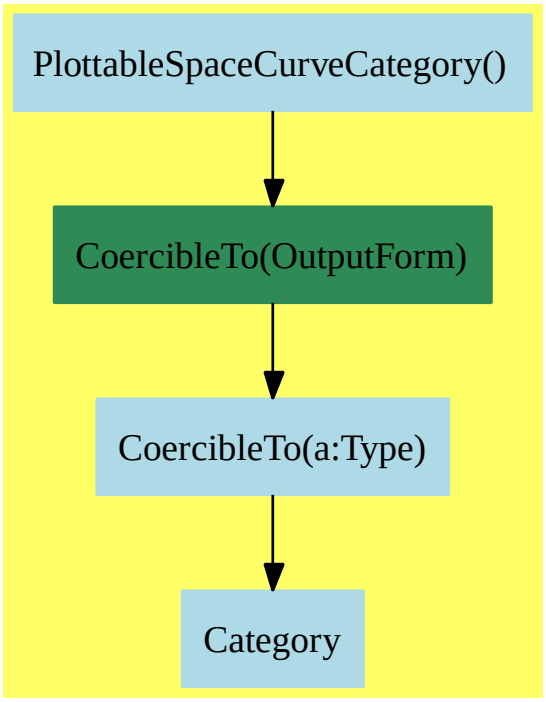
"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];
}
```

———-

3.0.31 PlottableSpaceCurveCategory (PSCURVE)



— PlottableSpaceCurveCategory.input —

```
)set break resume
)sys rm -f PlottableSpaceCurveCategory.output
)spool PlottableSpaceCurveCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show PlottableSpaceCurveCategory
--R PlottableSpaceCurveCategory  is a category constructor
--R Abbreviation for PlottableSpaceCurveCategory is PSCURVE
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PSCURVE
--R
--R------------------------------ Operations ------------------------------
--R coerce : % -> OutputForm              xRange : % -> Segment DoubleFloat
--R yRange : % -> Segment DoubleFloat     zRange : % -> Segment DoubleFloat
--R listBranches : % -> List List Point DoubleFloat
--R
```

```
--E 1

)spool
)lisp (bye)
```

———-

— PlottableSpaceCurveCategory.help —

```
====================================================================
PlottableSpaceCurveCategory examples
====================================================================

See Also:
o )show PlottableSpaceCurveCategory
```

———-

See:

⇐ “CoercibleTo” (KOERCE) 2.0.6 on page 17

Exports:

coerce listBranches xRange yRange zRange

These are directly exported but not implemented:

```
listBranches : % -> List List Point DoubleFloat
xRange : % -> Segment DoubleFloat
yRange : % -> Segment DoubleFloat
zRange : % -> Segment DoubleFloat
```

These exports come from (p17) CoercibleTo(OutputForm):

```
coerce : % -> OutputForm
```

— category PSCURVE PlottableSpaceCurveCategory —

```
)abbrev category PSCURVE PlottableSpaceCurveCategory
++ Author: Clifton J. Williamson
++ Date Created: 11 January 1990
++ Date Last Updated: 15 June 1990
++ Basic Operations: listBranches, xRange, yRange, zRange
++ Related Constructors:
++ Also See:
++ AMS Classifications:
```

```
++ Keywords: plot, graphics
++ References:
++ Description:
++ PlottableSpaceCurveCategory is the category of curves in
++ 3-space which may be plotted via the graphics facilities.  Functions are
++ provided for obtaining lists of lists of points, representing the
++ branches of the curve, and for determining the ranges of the
++ x-, y-, and z-coordinates of the points on the curve.

PlottableSpaceCurveCategory(): Category == Definition where
  L     ==> List
  SEG   ==> Segment
  SF    ==> DoubleFloat
  POINT ==> Point DoubleFloat

  Definition ==> CoercibleTo OutputForm with

    listBranches: % -> L L POINT
      ++ listBranches(c) returns a list of lists of points, representing the
      ++ branches of the curve c.
    xRange: % -> SEG SF
      ++ xRange(c) returns the range of the x-coordinates of the points
      ++ on the curve c.
    yRange: % -> SEG SF
      ++ yRange(c) returns the range of the y-coordinates of the points
      ++ on the curve c.
    zRange: % -> SEG SF
      ++ zRange(c) returns the range of the z-coordinates of the points
      ++ on the curve c.
```

———-

— PSCURVE.dotabb —

```
"PSCURVE"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PSCURVE"];
"PSCURVE" -> "KOERCE"
```

———-

— PSCURVE.dotfull —

```
"PlottableSpaceCurveCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PSCURVE"];
"PlottableSpaceCurveCategory()" -> "CoercibleTo(OutputForm)"
```

———-

— **PSCURVE.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"PlottableSpaceCurveCategory()" [color=lightblue];
"PlottableSpaceCurveCategory()" -> "CoercibleTo(OutputForm)"

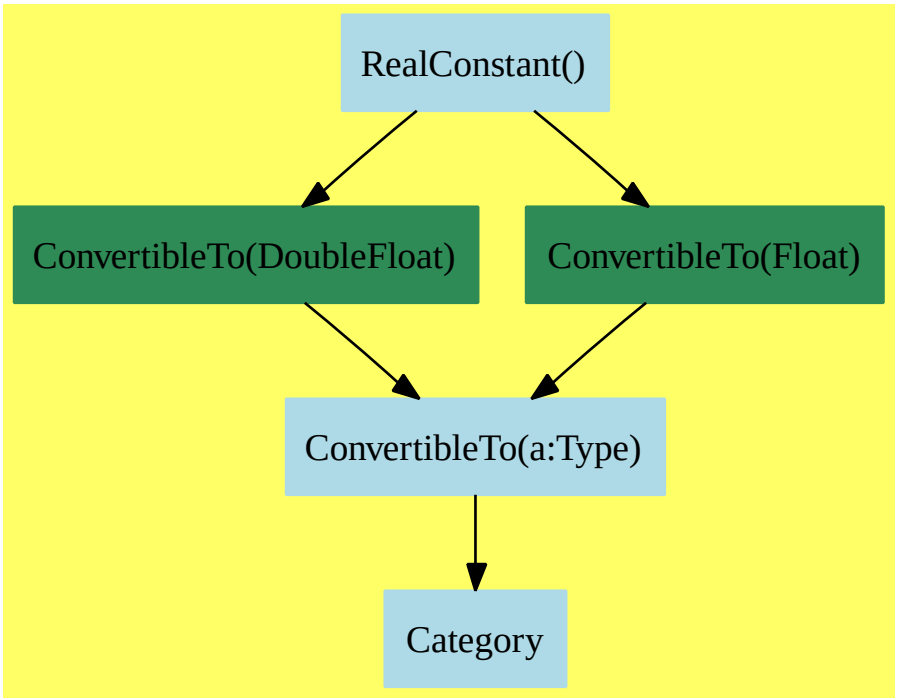
"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];
}
```

———-

3.0.32 RealConstant (REAL)



— RealConstant.input —

```
)set break resume
)sys rm -f RealConstant.output
)spool RealConstant.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show RealConstant
--R RealConstant  is a category constructor
--R Abbreviation for RealConstant is REAL
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for REAL
--R
--R------------------------------ Operations ------------------------------
--R convert : % -> Float                  convert : % -> DoubleFloat
--R
--E 1
```

```
)spool
)lisp (bye)
```

———-

— **RealConstant.help** —

```
==========================================================================
RealConstant examples
==========================================================================

See Also:
o )show RealConstant
```

———-

See:

⇒ “IntegerNumberSystem” (INS) 16.0.186 on page 1228
⇒ “QuotientFieldCategory” (QFCAT) 17.0.198 on page 1376
⇒ “RealNumberSystem” (RNS) 17.0.200 on page 1400
⇐ “ConvertibleTo” (KONVERT) 2.0.8 on page 23

Exports:

convert

These exports come from (p23) ConvertibleTo(DoubleFloat):

```
convert : % -> DoubleFloat
```

These exports come from (p23) ConvertibleTo(Float):

```
convert : % -> Float
```

— **category REAL RealConstant** —

```
)abbrev category REAL RealConstant
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
```

```
++ References:
++ Description:
++ The category of real numeric domains, i.e. convertible to floats.

RealConstant(): Category ==
  Join(ConvertibleTo DoubleFloat, ConvertibleTo Float)
```

———-

— REAL.dotabb —

```
"REAL"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=REAL"];
"REAL" -> "KONVERT"
```

———-

— REAL.dotfull —

```
"RealConstant()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=REAL"];
"RealConstant()" -> "ConvertibleTo(DoubleFloat)"
"RealConstant()" -> "ConvertibleTo(Float)"
```

———-

— REAL.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"RealConstant()" [color=lightblue];
"RealConstant()" -> "ConvertibleTo(DoubleFloat)"
"RealConstant()" -> "ConvertibleTo(Float)"

"ConvertibleTo(DoubleFloat)" [color=seagreen];
"ConvertibleTo(DoubleFloat)" -> "ConvertibleTo(a:Type)"

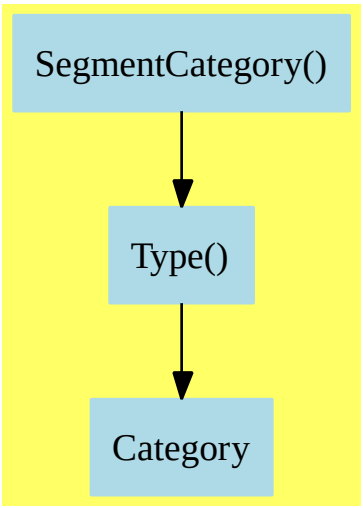
"ConvertibleTo(Float)" [color=seagreen];
"ConvertibleTo(Float)" -> "ConvertibleTo(a:Type)"

"ConvertibleTo(a:Type)" [color=lightblue];
"ConvertibleTo(a:Type)" -> "Category"
```

```
"Category" [color=lightblue];
}
```

———-

3.0.33 SegmentCategory (SEGCAT)



— **SegmentCategory.input** —

```
)set break resume
)sys rm -f SegmentCategory.output
)spool SegmentCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show SegmentCategory
--R SegmentCategory S: Type  is a category constructor
--R Abbreviation for SegmentCategory is SEGCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for SEGCAT
--R
--R------------------------------ Operations --------------------------------
--R BY : (%,Integer) -> %                ?..? : (S,S) -> %
```

```
--R convert : S -> %                    hi : % -> S
--R high : % -> S                       incr : % -> Integer
--R lo : % -> S                         low : % -> S
--R segment : (S,S) -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **SegmentCategory.help** —

```
=====================================================================
SegmentCategory examples
=====================================================================

See Also:
o )show SegmentCategory
```

———-

See:

⇒ “SegmentExpansionCategory” (SEGXCAT) 4.0.60 on page 235
⇐ “Type” (TYPE) 2.0.21 on page 66

Exports:

BY	convert	hi	high	incr
lo	low	segment	?..?	

Attributes Exported:

- **nil**

These are directly exported but not implemented:

```
BY : (%,Integer) -> %
convert : S -> %
hi : % -> S
high : % -> S
incr : % -> Integer
lo : % -> S
low : % -> S
segment : (S,S) -> %
?..? : (S,S) -> %
```

— category SEGCAT SegmentCategory —

```
)abbrev category SEGCAT SegmentCategory
++ Author:  Stephen M. Watt
++ Date Created:  December 1986
++ Date Last Updated: June 3, 1991
++ Basic Operations:
++ Related Domains:
++ Also See:
++ AMS Classifications:
++ Keywords: range, segment
++ Examples:
++ References:
++ Description:
++ This category provides operations on ranges, or segments
++ as they are called.

SegmentCategory(S:Type): Category == Type with
    SEGMENT: (S, S) -> %
        ++ \spad{l..h} creates a segment with l and h as the endpoints.
    BY: (%, Integer) -> %
        ++ \spad{s by n} creates a new segment in which only every
        ++ \spad{n}-th element is used.
    lo: % -> S
        ++ lo(s) returns the first endpoint of s.
        ++ Note that \spad{lo(l..h) = l}.
    hi: % -> S
        ++ hi(s) returns the second endpoint of s.
        ++ Note that \spad{hi(l..h) = h}.
    low: % -> S
        ++ low(s) returns the first endpoint of s.
        ++ Note that \spad{low(l..h) = l}.
    high: % -> S
        ++ high(s) returns the second endpoint of s.
        ++ Note that \spad{high(l..h) = h}.
    incr: % -> Integer
        ++ incr(s) returns \spad{n}, where s is a segment in which every
        ++ \spad{n}-th element is used.
        ++ Note that \spad{incr(l..h by n) = n}.
    segment: (S, S) -> %
        ++ segment(i,j) is an alternate way to create the segment
        ++ \spad{i..j}.
    convert: S -> %
        ++ convert(i) creates the segment \spad{i..i}.
```

———-

— **SEGCAT.dotabb** —

```
"SEGCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SEGCAT"];
"SEGCAT" -> "TYPE"
```

———-

— **SEGCAT.dotfull** —

```
"SegmentCategory(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SEGCAT"];
"SegmentCategory(a:Type)" -> "Type()"

"SegmentCategory(OrderedRing)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=SEGCAT"];
"SegmentCategory(OrderedRing)" -> "SegmentCategory(a:Type)"
```

———-

— **SEGCAT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

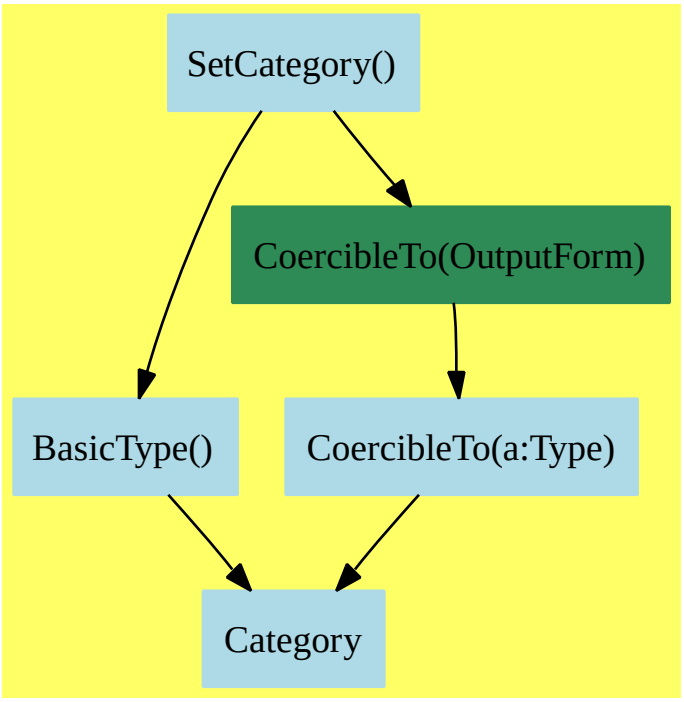
"SegmentCategory(a:Type)" [color=lightblue];
"SegmentCategory(a:Type)" -> "Type()"

"Type()" [color=lightblue];
"Type()" -> "Category"

"Category" [color=lightblue];
}
```

———-

3.0.34 SetCategory (SETCAT)



— **SetCategory.input** —

```
)set break resume
)sys rm -f SetCategory.output
)spool SetCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show SetCategory
--R SetCategory  is a category constructor
--R Abbreviation for SetCategory is SETCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for SETCAT
--R
--R------------------------------ Operations ------------------------------
--R ?=? : (%,%) -> Boolean                coerce : % -> OutputForm
--R hash : % -> SingleInteger             latex : % -> String
--R ?~=? : (%,%) -> Boolean
--R
```

```
--E 1

)spool
)lisp (bye)
```

———-

— **SetCategory.help** —

```
=====================================================================
SetCategory examples
=====================================================================

See Also:
o )show SetCategory
```

———-

See:

⇒ “AbelianSemiGroup” (ABELSG) 4.0.36 on page 123
⇒ “BlowUpMethodCategory” (BLMETCT) 4.0.37 on page 127
⇒ “FileCategory” (FILECAT) 4.0.45 on page 163
⇒ “FileNameCategory” (FNCAT) 4.0.47 on page 172
⇒ “Finite” (FINITE) 4.0.46 on page 168
⇒ “GradedModule” (GRMOD) 4.0.48 on page 176
⇒ “HomogeneousAggregate” (HOAGG) 4.0.49 on page 181
⇒ “IndexedDirectProductCategory” (IDPC) 4.0.50 on page 188
⇒ “NumericalIntegrationCategory” (NUMINT) 4.0.53 on page 202
⇒ “NumericalOptimizationCategory” (OPTCAT) 4.0.54 on page 207
⇒ “OrderedSet” (ORDSET) 4.0.56 on page 216
⇒ “OrdinaryDifferentialEquationsSolverCategory” (ODECAT) 4.0.55 on page 212
⇒ “PartialDifferentialEquationsSolverCategory” (PDECAT) 4.0.57 on page 221
⇒ “PatternMatchable” (PATMAB) 4.0.58 on page 226
⇒ “PolynomialSetCategory” (PSETCAT) 6.0.91 on page 465
⇒ “RealRootCharacterizationCategory” (RRCC) 4.0.59 on page 230
⇒ “SemiGroup” (SGROUP) 4.0.61 on page 239
⇒ “SetAggregate” (SETAGG) 6.0.94 on page 492
⇒ “SExpressionCategory” (SEXCAT) 4.0.63 on page 246
⇒ “StepThrough” (STEP) 4.0.64 on page 252
⇒ “StringCategory” (STRICAT) 10.0.149 on page 920
⇒ “ThreeSpaceCategory” (SPACEC) 4.0.65 on page 256
⇐ “BasicType” (BASTYPE) 2.0.5 on page 14
⇐ “CoercibleTo” (KOERCE) 2.0.6 on page 17

Exports:

coerce hash latex ?=? ?~=?

These are implemented by this category:

```
hash : % -> SingleInteger
latex : % -> String
```

These exports come from (p14) BasicType():

```
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
```

These exports come from (p17) CoercibleTo(OutputForm):

```
coerce : % -> OutputForm
```

— category SETCAT SetCategory —

```
)abbrev category SETCAT SetCategory
++ Author:
++ Date Created:
++ Date Last Updated: November 10, 2009 tpd happy birthday
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ \spadtype{SetCategory} is the basic category for describing a collection
++ of elements with \spadop{=} (equality) and \spadfun{coerce} to
++ output form.
++
++ Conditional Attributes\br
++ \tab{5}canonical\tab{5}data structure equality is the same as \spadop{=}

SetCategory(): Category == Join(BasicType,CoercibleTo OutputForm) with
    hash: % -> SingleInteger
      ++ hash(s) calculates a hash code for s.
    latex: % -> String
      ++ latex(s) returns a LaTeX-printable output
      ++ representation of s.
  add
    hash(s : %):  SingleInteger == SXHASH(s)$Lisp
    latex(s : %): String       == "\mbox{\bf Unimplemented}"
```

———-

— **SETCAT.dotabb** —

```
"SETCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SETCAT"];
"SETCAT" -> "BASTYPE"
"SETCAT" -> "KOERCE"
```

———-

— **SETCAT.dotfull** —

```
"SetCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SETCAT"];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"
```

———-

— **SETCAT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

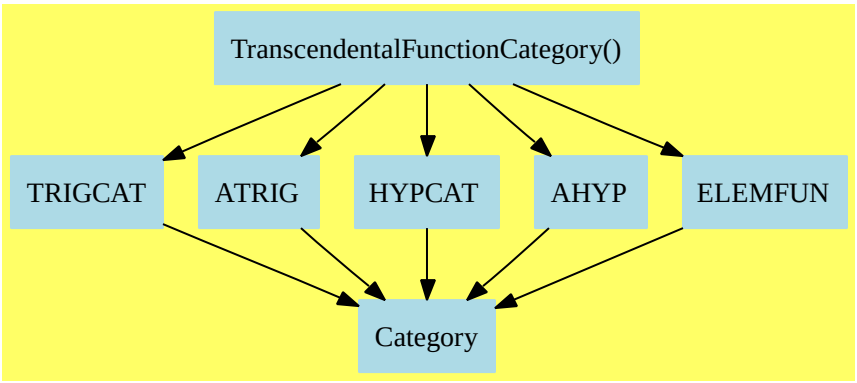
"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];

}
```

———-

3.0.35 TranscendentalFunctionCategory (TRANFUN)



— TranscendentalFunctionCategory.input —

```
)set break resume
)sys rm -f TranscendentalFunctionCategory.output
)spool TranscendentalFunctionCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show TranscendentalFunctionCategory
--R TranscendentalFunctionCategory  is a category constructor
--R Abbreviation for TranscendentalFunctionCategory is TRANFUN
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for TRANFUN
--R
--R------------------------------ Operations --------------------------------
--R ?**? : (%,%) -> %                    acos : % -> %
--R acosh : % -> %                       acot : % -> %
--R acoth : % -> %                       acsc : % -> %
--R acsch : % -> %                       asec : % -> %
--R asech : % -> %                       asin : % -> %
--R asinh : % -> %                       atan : % -> %
--R atanh : % -> %                       cos : % -> %
--R cosh : % -> %                        cot : % -> %
--R coth : % -> %                        csc : % -> %
--R csch : % -> %                        exp : % -> %
--R log : % -> %                         pi : () -> %
--R sec : % -> %                         sech : % -> %
--R sin : % -> %                         sinh : % -> %
--R tan : % -> %                         tanh : % -> %
--R
```

```
--E 1

)spool
)lisp (bye)
```

———-

— **TranscendentalFunctionCategory.help** —

```
=====================================================================
TranscendentalFunctionCategory examples
=====================================================================

See Also:
o )show TranscendentalFunctionCategory
```

———-

The `acsch`, `asech`, and `acoth` functions were modified to include an intermediate test to check that the argument has a reciprocal values.

See:

⇒ “IntervalCategory” (INTCAT) 14.0.176 on page 1150
⇒ “LiouvillianFunctionCategory” (LFCAT) 4.0.51 on page 192
⇒ “UnivariateLaurentSeriesCategory” (ULSCAT) 17.0.202 on page 1448
⇒ “UnivariatePuiseuxSeriesCategory” (UPXSCAT) 17.0.203 on page 1460
⇒ “UnivariateTaylorSeriesCategory” (UTSCAT) 16.0.190 on page 1277
⇐ “ArcHyperbolicFunctionCategory” (AHYP) 2.0.2 on page 5
⇐ “ArcTrigonometricFunctionCategory” (ATRIG) 2.0.3 on page 7
⇐ “ElementaryFunctionCategory” (ELEMFUN) 2.0.9 on page 27
⇐ “HyperbolicFunctionCategory” (HYPCAT) 2.0.11 on page 32
⇐ “TrigonometricFunctionCategory” (TRIGCAT) 2.0.20 on page 63

Exports:

?**?	acos	acosh	acot	acoth
acsc	acsch	asec	asech	asin
asinh	atan	atanh	cos	cosh
cot	coth	csc	csch	exp
log	pi	sec	sech	sin
sinh	tan	tanh		

These are implemented by this category:

```
pi : () -> %
```

These exports come from (p63) TrigonometricFunctionCategory():

```
cos : % -> %
cot : % -> %
csc : % -> %
sec : % -> %
sin : % -> %
tan : % -> %
```

These exports come from (p7) ArcTrigonometricFunctionCategory():

```
acos : % -> %
acot : % -> %
acsc : % -> %
asec : % -> %
asin : % -> %
atan : % -> %
```

These exports come from (p32) HyperbolicFunctionCategory():

```
cosh : % -> %
coth : % -> %
csch : % -> %
sech : % -> %
sinh : % -> %
tanh : % -> %
```

These exports come from (p5) ArcHyperbolicFunctionCategory():

```
acosh : % -> %
acoth : % -> %
acsch : % -> %
asech : % -> %
asinh : % -> %
atanh : % -> %
```

These exports come from (p27) ElementaryFunctionCategory():

```
?**? : (%,%) -> %
exp : % -> %
log : % -> %
```

— category TRANFUN TranscendentalFunctionCategory —

```
)abbrev category TRANFUN TranscendentalFunctionCategory
++ Category for the transcendental elementary functions
++ Author: Manuel Bronstein
++ Date Created: ???
++ Date Last Updated: 14 May 1991
++ Description:
```

```
++ Category for the transcendental elementary functions;

TranscendentalFunctionCategory(): Category ==
    Join(TrigonometricFunctionCategory,ArcTrigonometricFunctionCategory,
         HyperbolicFunctionCategory,ArcHyperbolicFunctionCategory,
         ElementaryFunctionCategory) with
           pi : () -> $        ++ pi() returns the constant pi.
   add
     if $ has Ring then
       pi()   == 2*asin(1)
       acsch x ==
         (a := recip x) case "failed" => error "acsch: no reciprocal"
         asinh(a::$)
       asech x ==
         (a := recip x) case "failed" => error "asech: no reciprocal"
         acosh(a::$)
       acoth x ==
         (a := recip x) case "failed" => error "acoth: no reciprocal"
         atanh(a::$)
     if $ has Field and $ has sqrt: $ -> $ then
       asin x == atan(x/sqrt(1-x**2))
       acos x == pi()/2::$ - asin x
       acot x == pi()/2::$ - atan x
       asinh x == log(x + sqrt(x**2 + 1))
       acosh x == 2*log(sqrt((x+1)/2::$) + sqrt((x-1)/2::$))
       atanh x == (log(1+x)-log(1-x))/2::$
```

———-

— TRANFUN.dotabb —

```
"TRANFUN"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=TRANFUN"];
"TRANFUN" -> "TRIGCAT"
"TRANFUN" -> "ATRIG"
"TRANFUN" -> "HYPCAT"
"TRANFUN" -> "AHYP"
"TRANFUN" -> "ELEMFUN"
```

———-

— TRANFUN.dotfull —

```
"TranscendentalFunctionCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=TRANFUN"];
"TranscendentalFunctionCategory()" ->
```

```
    "TrigonometricFunctionCategory()"
"TranscendentalFunctionCategory()" ->
    "ArcTrigonometricFunctionCategory()"
"TranscendentalFunctionCategory()" ->
    "HyperbolicFunctionCategory()"
"TranscendentalFunctionCategory()" ->
    "ArcHyperbolicFunctionCategory()"
"TranscendentalFunctionCategory()" ->
    "ElementaryFunctionCategory()"
```

———-

— TRANFUN.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"TranscendentalFunctionCategory()" [color=lightblue];
"TranscendentalFunctionCategory()" ->
    "TRIGCAT"
"TranscendentalFunctionCategory()" ->
    "ATRIG"
"TranscendentalFunctionCategory()" ->
    "HYPCAT"
"TranscendentalFunctionCategory()" ->
    "AHYP"
"TranscendentalFunctionCategory()" ->
    "ELEMFUN"

"TRIGCAT" [color=lightblue];
"TRIGCAT" -> "Category"

"ATRIG" [color=lightblue];
"ATRIG" -> "Category"

"HYPCAT" [color=lightblue];
"HYPCAT" -> "Category"

"AHYP" [color=lightblue];
"AHYP" -> "Category"

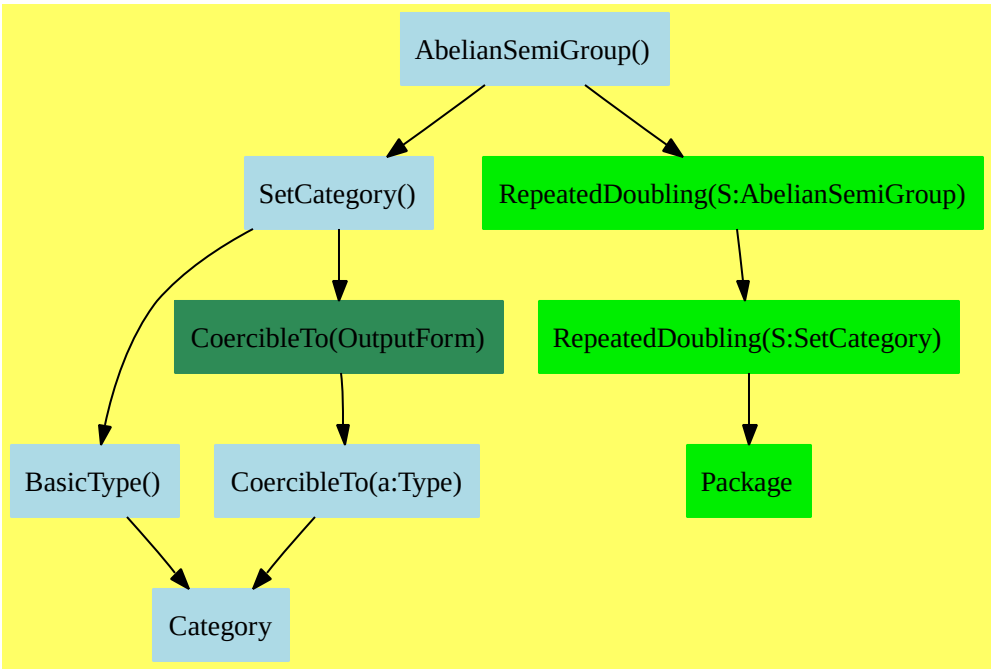
"ELEMFUN" [color=lightblue];
"ELEMFUN" -> "Category"

"Category" [color=lightblue];
}
```

Chapter 4

Category Layer 3

4.0.36 AbelianSemiGroup (ABELSG)



— **AbelianSemiGroup.input** —

```
)set break resume
)sys rm -f AbelianSemiGroup.output
)spool AbelianSemiGroup.output
```

```
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show AbelianSemiGroup
--R AbelianSemiGroup  is a category constructor
--R Abbreviation for AbelianSemiGroup is ABELSG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for ABELSG
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (PositiveInteger,%) -> %        ?+? : (%,%) -> %
--R ?=? : (%,%) -> Boolean                coerce : % -> OutputForm
--R hash : % -> SingleInteger             latex : % -> String
--R ?~=? : (%,%) -> Boolean
--R
--E 1

)spool
)lisp (bye)
```

———-

— **AbelianSemiGroup.help** —

```
====================================================================
AbelianSemiGroup examples
====================================================================

See Also:
o )show AbelianSemiGroup
```

———-

See:

⇒ “AbelianMonoid” (ABELMON) 5.0.66 on page 267
⇒ “FunctionSpace” (FS) 17.0.195 on page 1336
⇐ “SetCategory” (SETCAT) 3.0.34 on page 113

Exports:

coerce	hash	latex	?*?	?+?
?=?	?~=?			

These are directly exported but not implemented:

```
?+? : (%,%) -> %
```

These are implemented by this category:

```
?*? : (PositiveInteger,%) -> %
```

These exports come from (p113) SetCategory():

```
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
```

— category ABELSG AbelianSemiGroup —

```
)abbrev category ABELSG AbelianSemiGroup
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The class of all additive (commutative) semigroups, i.e.
++ a set with a commutative and associative operation \spadop{+}.
++
++ Axioms\br
++ \tab{5}\spad{associative("+":(%,%)->%)}\tab{5}\spad{ (x+y)+z = x+(y+z) }\br
++ \tab{6}\spad{commutative("+":(%,%)->%)}\tab{5}\spad{ x+y = y+x }

AbelianSemiGroup(): Category == SetCategory with
      "+": (%,%) -> %                  ++ x+y computes the sum of x and y.
      "*": (PositiveInteger,%) -> %
        ++ n*x computes the left-multiplication of x by the positive
        ++ integer n. This is equivalent to adding x to itself n times.
    add
      import RepeatedDoubling(%)
      if not (% has Ring) then
        n:PositiveInteger * x:% == double(n,x)
```

———-

— ABELSG.dotabb —

```
"ABELSG"
```

```
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ABELSG"];
"ABELSG" -> "SETCAT"
"ABELSG" -> "REPDB"
```

———-

— ABELSG.dotfull —

```
"AbelianSemiGroup()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ABELSG"];
"AbelianSemiGroup()" -> "SetCategory()"
"AbelianSemiGroup()" -> "RepeatedDoubling(a:SetCategory)"
```

———-

— ABELSG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"AbelianSemiGroup()" [color=lightblue];
"AbelianSemiGroup()" -> "SetCategory()"
"AbelianSemiGroup()" -> "RepeatedDoubling(AbelianSemiGroup)"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" ->
    "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"RepeatedDoubling(AbelianSemiGroup)" [color="#00EE00"];
"RepeatedDoubling(AbelianSemiGroup)" -> "RepeatedDoubling(a:SetCategory)"

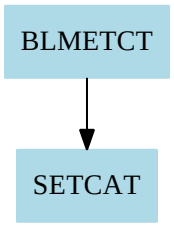
"RepeatedDoubling(a:SetCategory)" [color="#00EE00"];
"RepeatedDoubling(a:SetCategory)" -> "Package"
```

```
"Package" [color="#00EE00"];

"Category" [color=lightblue];
}
```

———-

4.0.37 BlowUpMethodCategory (BLMETCT)



— BlowUpMethodCategory.input —

```
)set break resume
)sys rm -f BlowUpMethodCategory.output
)spool BlowUpMethodCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show BlowUpMethodCategory
--R BlowUpMethodCategory  is a category constructor
--R Abbreviation for BlowUpMethodCategory is BLMETCT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for BLMETCT
--R
--R------------------------------ Operations --------------------------------
--R ?=? : (%,%) -> Boolean               chartCoord : % -> Integer
--R coerce : List Integer -> %           coerce : % -> OutputForm
--R excepCoord : % -> Integer            hash : % -> SingleInteger
--R infClsPt? : % -> Boolean             latex : % -> String
--R quotValuation : % -> Integer         ramifMult : % -> Integer
--R transCoord : % -> Integer            ?~=? : (%,%) -> Boolean
--R createHN : (Integer,Integer,Integer,Integer,Integer,Boolean,Union(left,center,right,vertical,horizontal)
--R type : % -> Union(left,center,right,vertical,horizontal)
--R
--E 1
```

```
)spool
)lisp (bye)
```

———-

— **BlowUpMethodCategory.help** —

```
=====================================================================
BlowUpMethodCategory examples
=====================================================================

See Also:
o )show BlowUpMethodCategory
```

———-

See:

⇐ "BasicType" (BASTYPE) 2.0.5 on page 14
⇐ "CoercibleTo" (KOERCE) 2.0.6 on page 17
⇐ "SetCategory" (SETCAT) 3.0.34 on page 113

Exports:

?=?	?~=?	chartCoord	coerce	createHN
excepCoord	hash	infClsPt?	latex	quotValuation
ramifMult	transCoord	type		

These are directly exported but not implemented:

```
coerce: List Integer -> %
excepCoord: % -> Integer
chartCoord: % -> Integer
transCoord: % -> Integer
createHN:  ( Integer, Integer, Integer , Integer, Integer, Boolean,_
             Union("left","center","right","vertical","horizontal")) -> %
ramifMult: % -> Integer
infClsPt_? : % -> Boolean
quotValuation : % -> Integer
type: % -> Union("left","center","right","vertical","horizontal")
```

These exports come from (p113) SetCategory():

```
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
```

— category BLMETCT BlowUpMethodCategory —

```
)abbrev category BLMETCT BlowUpMethodCategory
BlowUpMethodCategory:Category ==  SetCategory with

  coerce: List Integer -> %

  excepCoord: % -> Integer

  chartCoord: % -> Integer

  transCoord: % -> Integer

  createHN:  ( Integer, Integer, Integer , Integer, Integer, Boolean,_
               Union("left","center","right","vertical","horizontal")) -> %

  ramifMult: % -> Integer

  infClsPt_? : % -> Boolean

  quotValuation : % -> Integer

  type: % -> Union("left","center","right","vertical","horizontal")
```

———-

— BLMETCT.dotabb —

```
"BLMETCT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=BLMETCT"];
"BLMETCT" -> "SETCAT"
```

———-

— BLMETCT.dotfull —

```
"BlowUpMethodCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=BLMETCT"];
"BlowUpMethodCategory()" -> "SetCategory()"
```

———-

— BLMETCT.dotpic —

```
digraph pic {
```

```
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"BlowUpMethodCategory()" [color=lightblue];
"BlowUpMethodCategory()" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

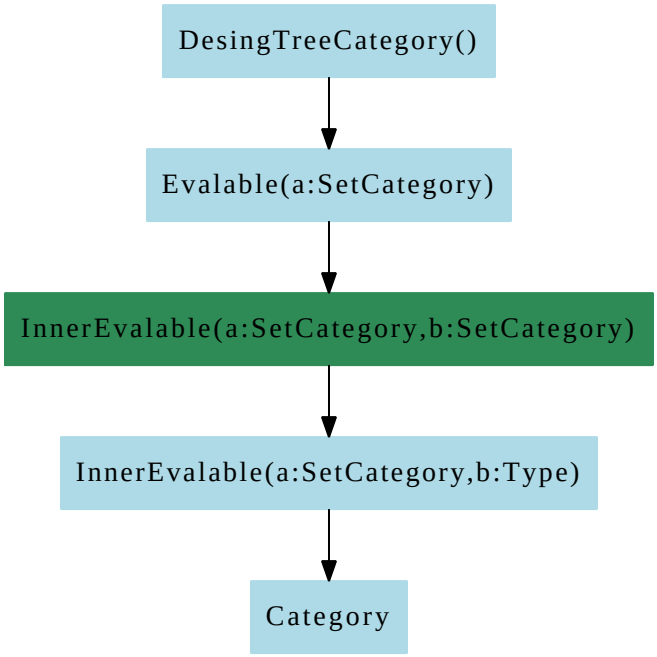
"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" ->
    "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];
}
```

4.0.38 DesingTreeCategory (DSTRCAT)



— DesingTreeCategory.input —

```
)set break resume
)sys rm -f DesingTreeCategory.output
)spool DesingTreeCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show DesingTreeCategory
--R DesingTreeCategory S: SetCategory  is a category constructor
--R Abbreviation for DesingTreeCategory is DSTRCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for DSTRCAT
--R
--R------------------------------ Operations --------------------------------
--R children : % -> List %                copy : % -> %
--R cyclic? : % -> Boolean                distance : (%,%) -> Integer
--R ?.value : (%,value) -> S              empty : () -> %
--R empty? : % -> Boolean                 eq? : (%,%) -> Boolean
--R leaf? : % -> Boolean                  leaves : % -> List S
--R map : ((S -> S),%) -> %               nodes : % -> List %
--R sample : () -> %                      tree : List S -> %
```

```
--R tree : S -> %                          tree : (S,List %) -> %
--R value : % -> S
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R child? : (%,%) -> Boolean if S has SETCAT
--R coerce : % -> OutputForm if S has SETCAT
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R hash : % -> SingleInteger if S has SETCAT
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R node? : (%,%) -> Boolean if S has SETCAT
--R parts : % -> List S if $ has finiteAggregate
--R setchildren! : (%,List %) -> % if $ has shallowlyMutable
--R setelt : (%,value,S) -> S if $ has shallowlyMutable
--R setvalue! : (%,S) -> S if $ has shallowlyMutable
--R size? : (%,NonNegativeInteger) -> Boolean
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— DesingTreeCategory.help —

```
=====================================================================
DesingTreeCategory examples
=====================================================================

See Also:
o )show DesingTreeCategory
```

———-

See:

⇒ "Evalable" (EVALAB) 3.0.25 on page 80

Exports:

#?	?.value	?=?	?~=?	any?
child?	children	coerce	copy	count
count	cyclic?	distance	empty	empty?
eq?	eval	every?	hash	latex
leaf?	leaves	less?	map	map!
member?	members	more?	node?	nodes
parts	sample	setchildren!	setelt	setvalue!
size?	tree	value		

Attributes Exported:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.

These are directly exported but not implemented:

```
tree : (S,List %) -> %
tree : List S -> %
tree : S -> %
```

These exports come from (p351) RecursiveAggregate(Type)

```
#? : % -> NonNegativeInteger if $ has finiteAggregate
?.value : (%,value) -> S
?=? : (%,%) -> Boolean if S has SETCAT
?~=? : (%,%) -> Boolean if S has SETCAT
any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
child? : (%,%) -> Boolean if S has SETCAT
children : % -> List %
coerce : % -> OutputForm if S has SETCAT
copy : % -> %
count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
cyclic? : % -> Boolean
distance : (%,%) -> Integer
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
```

```
 eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
 every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
 hash : % -> SingleInteger if S has SETCAT
 latex : % -> String if S has SETCAT
 leaf? : % -> Boolean
 leaves : % -> List S
 less? : (%,NonNegativeInteger) -> Boolean
 map : ((S -> S),%) -> %
 map! : ((S -> S),%) -> % if $ has shallowlyMutable
 member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
 members : % -> List S if $ has finiteAggregate
 more? : (%,NonNegativeInteger) -> Boolean
 node? : (%,%) -> Boolean if S has SETCAT
 nodes : % -> List %
 parts : % -> List S if $ has finiteAggregate
 sample : () -> %
 setchildren! : (%,List %) -> % if $ has shallowlyMutable
 setelt : (%,value,S) -> S if $ has shallowlyMutable
 setvalue! : (%,S) -> S if $ has shallowlyMutable
 size? : (%,NonNegativeInteger) -> Boolean
 value : % -> S
```

— category DSTRCAT DesingTreeCategory —

```
)abbrev category DSTRCAT DesingTreeCategory
++ Authors: Gaetan Hache
++ Date Created: jan 1998
++ Date Last Updated: May 2010 by Tim Daly
++ Description:
++ This category is part of the PAFF package
DesingTreeCategory(S: SetCategory):Category == RecursiveAggregate(S) with

  shallowlyMutable
  finiteAggregate

  tree: (S,List %) -> %
    ++ tree(nd,ls) creates a tree with value nd, and children ls.

  tree: S -> %
    ++ tree(nd) creates a tree with value nd, and no children

  tree: List(S) -> %
    ++ tree(l) creates a chain tree from the list l
```

———-

— DSTRCAT.dotabb —

```
"DSTRCAT" [color=lightblue,href="bookvol10.2.pdf#nameddest=DSTRCAT"];
"EVALAB" [color="#4488FF",href="bookvol10.2.pdf#nameddest=EVALAB"]
"DSTRCAT" -> "EVALAB"
```

———-

— DSTRCAT.dotfull —

```
"DesingTreeCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=DSTRCAT"];
"DesingTreeCategory()" -> "Evalable()"
```

———-

— DSTRCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"DesingTreeCategory()" [color=lightblue];
"DesingTreeCategory()" -> "Evalable(a:SetCategory)"

"Evalable(a:SetCategory)" [color=lightblue];
"Evalable(a:SetCategory)" -> "InnerEvalable(a:SetCategory,b:SetCategory)"

"InnerEvalable(a:SetCategory,b:SetCategory)" [color=seagreen];
"InnerEvalable(a:SetCategory,b:SetCategory)" ->
    "InnerEvalable(a:SetCategory,b:Type)"

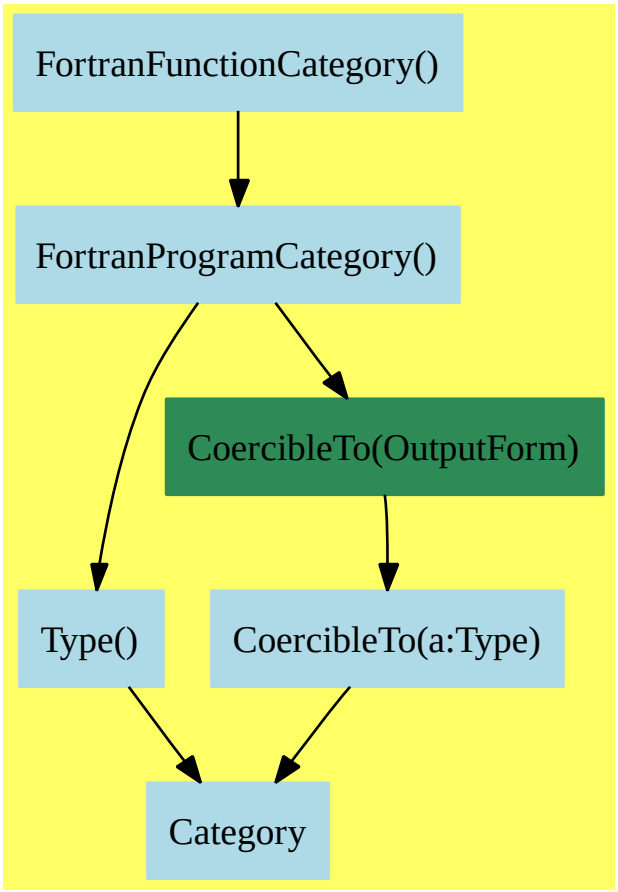
"InnerEvalable(a:SetCategory,b:Type)" [color=lightblue];
"InnerEvalable(a:SetCategory,b:Type)" -> "Category"

"Category" [color=lightblue];

}
```

———-

4.0.39 FortranFunctionCategory (FORTFN)



— **FortranFunctionCategory.input** —

```
)set break resume
)sys rm -f FortranFunctionCategory.output
)spool FortranFunctionCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FortranFunctionCategory
--R FortranFunctionCategory  is a category constructor
--R Abbreviation for FortranFunctionCategory is FORTFN
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FORTFN
```

```
--R
--R------------------------------ Operations ------------------------------
--R coerce : FortranCode -> %             coerce : List FortranCode -> %
--R coerce : % -> OutputForm              outputAsFortran : % -> Void
--R retract : Polynomial Integer -> %     retract : Polynomial Float -> %
--R retract : Expression Integer -> %     retract : Expression Float -> %
--R coerce : Record(localSymbols: SymbolTable,code: List FortranCode) -> %
--R retract : Fraction Polynomial Integer -> %
--R retract : Fraction Polynomial Float -> %
--R retractIfCan : Fraction Polynomial Integer -> Union(%,"failed")
--R retractIfCan : Fraction Polynomial Float -> Union(%,"failed")
--R retractIfCan : Polynomial Integer -> Union(%,"failed")
--R retractIfCan : Polynomial Float -> Union(%,"failed")
--R retractIfCan : Expression Integer -> Union(%,"failed")
--R retractIfCan : Expression Float -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **FortranFunctionCategory.help** —

```
=====================================================================
FortranFunctionCategory examples
=====================================================================

See Also:
o )show FortranFunctionCategory
```

———-

See:

⇐ “FortranProgramCategory” (FORTCAT) 3.0.26 on page 84

Exports:

coerce outputAsFortran retract retractIfCan

Attributes:

- **nil**

These are directly exported but not implemented:

```
coerce : FortranCode -> %
```

```
 coerce : List FortranCode -> %
 coerce : Record(localSymbols: SymbolTable,code: List FortranCode) -> %
 retract : Expression Float -> %
 retract : Expression Integer -> %
 retract : Polynomial Float -> %
 retract : Polynomial Integer -> %
 retract : Fraction Polynomial Integer -> %
 retract : Fraction Polynomial Float -> %
 retractIfCan : Fraction Polynomial Integer -> Union(%,"failed")
 retractIfCan : Fraction Polynomial Float -> Union(%,"failed")
 retractIfCan : Polynomial Integer -> Union(%,"failed")
 retractIfCan : Polynomial Float -> Union(%,"failed")
 retractIfCan : Expression Integer -> Union(%,"failed")
 retractIfCan : Expression Float -> Union(%,"failed")
```

These exports come from (p84) FortranProgramCategory():

```
 coerce : % -> OutputForm
 outputAsFortran : % -> Void
```

— category FORTFN FortranFunctionCategory —

```
)abbrev category FORTFN FortranFunctionCategory
++ Author: Mike Dewar
++ Date Created: 13 January 1994
++ Date Last Updated: 18 March 1994
++ Related Constructors: FortranProgramCategory.
++ Description:
++ \axiomType{FortranFunctionCategory} is the category of arguments to
++ NAG Library routines which return (sets of) function values.

FortranFunctionCategory():Category == FortranProgramCategory with
  coerce : List FortranCode -> $
    ++ coerce(e) takes an object from \spadtype{List FortranCode} and
    ++  uses it as the body of an ASP.
  coerce : FortranCode -> $
    ++ coerce(e) takes an object from \spadtype{FortranCode} and
    ++  uses it as the body of an ASP.
  coerce : Record(localSymbols:SymbolTable,code:List(FortranCode)) -> $
    ++ coerce(e) takes the component of \spad{e} from
    ++ \spadtype{List FortranCode} and uses it as the body of the ASP,
    ++ making the declarations in the \spadtype{SymbolTable} component.
  retract : Expression Float -> $
    ++ retract(e) tries to convert \spad{e} into an ASP, checking that
    ++  legal Fortran-77 is produced.
  retractIfCan : Expression Float -> Union($,"failed")
    ++ retractIfCan(e) tries to convert \spad{e} into an ASP, checking that
    ++  legal Fortran-77 is produced.
  retract : Expression Integer -> $
```

```
    ++ retract(e) tries to convert \spad{e} into an ASP, checking that
    ++  legal Fortran-77 is produced.
  retractIfCan : Expression Integer -> Union($,"failed")
    ++ retractIfCan(e) tries to convert \spad{e} into an ASP, checking that
    ++  legal Fortran-77 is produced.
  retract : Polynomial Float -> $
    ++ retract(e) tries to convert \spad{e} into an ASP, checking that
    ++  legal Fortran-77 is produced.
  retractIfCan : Polynomial Float -> Union($,"failed")
    ++ retractIfCan(e) tries to convert \spad{e} into an ASP, checking that
    ++  legal Fortran-77 is produced.
  retract : Polynomial Integer -> $
    ++ retract(e) tries to convert \spad{e} into an ASP, checking that
    ++  legal Fortran-77 is produced.
  retractIfCan : Polynomial Integer -> Union($,"failed")
    ++ retractIfCan(e) tries to convert \spad{e} into an ASP, checking that
    ++  legal Fortran-77 is produced.
  retract : Fraction Polynomial Float -> $
    ++ retract(e) tries to convert \spad{e} into an ASP, checking that
    ++  legal Fortran-77 is produced.
  retractIfCan : Fraction Polynomial Float -> Union($,"failed")
    ++ retractIfCan(e) tries to convert \spad{e} into an ASP, checking that
    ++  legal Fortran-77 is produced.
  retract : Fraction Polynomial Integer -> $
    ++ retract(e) tries to convert \spad{e} into an ASP, checking that
    ++  legal Fortran-77 is produced.
  retractIfCan : Fraction Polynomial Integer -> Union($,"failed")
    ++ retractIfCan(e) tries to convert \spad{e} into an ASP, checking that
    ++  legal Fortran-77 is produced.

  -- NB: These ASPs also have a coerce from an appropriate instantiation
  --     of FortranExpression.
```

———-

— FORTFN.dotabb —

```
"FORTFN"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FORTFN"];
"FORTFN" -> "FORTCAT"
```

———-

— FORTFN.dotfull —

```
"FortranFunctionCategory()"
```

```
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FORTFN"];
"FortranFunctionCategory()" -> "FortranProgramCategory()"
```

———-

— FORTFN.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FortranFunctionCategory()" [color=lightblue];
"FortranFunctionCategory()" -> "FortranProgramCategory()"

"FortranProgramCategory()" [color=lightblue];
"FortranProgramCategory()" -> "Type()"
"FortranProgramCategory()" -> "CoercibleTo(OutputForm)"

"Type()" [color=lightblue];
"Type()" -> "Category"

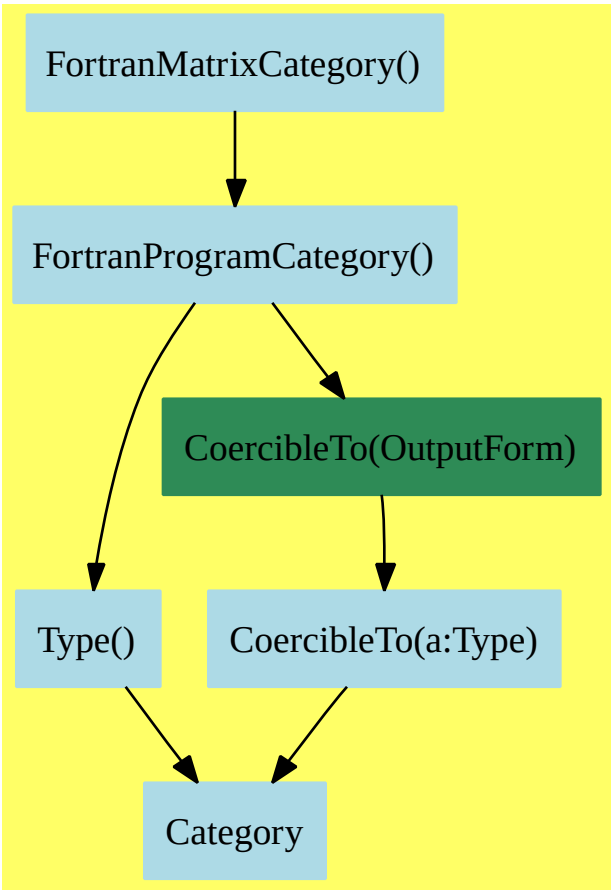
"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];
}
```

———-

4.0.40 FortranMatrixCategory (FMC)



— **FortranMatrixCategory.input** —

```
)set break resume
)sys rm -f FortranMatrixCategory.output
)spool FortranMatrixCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FortranMatrixCategory
--R FortranMatrixCategory  is a category constructor
--R Abbreviation for FortranMatrixCategory is FMC
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FMC
```

```
--R
--R------------------------------ Operations --------------------------------
--R coerce : FortranCode -> %              coerce : List FortranCode -> %
--R coerce : Matrix MachineFloat -> %      coerce : % -> OutputForm
--R outputAsFortran : % -> Void
--R coerce : Record(localSymbols: SymbolTable,code: List FortranCode) -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **FortranMatrixCategory.help** —

```
=====================================================================
FortranMatrixCategory examples
=====================================================================

See Also:
o )show FortranMatrixCategory
```

———-

See:

⇐ “FortranProgramCategory” (FORTCAT) 3.0.26 on page 84

Exports:

coerce outputAsFortran

Attributes:

- **nil**

These are directly exported but not implemented:

```
coerce : Matrix MachineFloat -> %
coerce : List FortranCode -> %
coerce : FortranCode -> %
coerce : Record(localSymbols: SymbolTable,code: List FortranCode) -> %
```

These exports come from (p84) FortranProgramCategory():

```
coerce : % -> OutputForm
outputAsFortran : % -> Void
```

— **category FMC FortranMatrixCategory** —

```
)abbrev category FMC FortranMatrixCategory
++ Author: Mike Dewar
++ Date Created: 21 March 1994
++ Date Last Updated:
++ Related Constructors: FortranProgramCategory.
++ Description:
++ \axiomType{FortranMatrixCategory} provides support for
++ producing Functions and Subroutines when the input to these
++ is an AXIOM object of type \axiomType{Matrix} or in domains
++ involving \axiomType{FortranCode}.

FortranMatrixCategory():Category == FortranProgramCategory with
    coerce : Matrix MachineFloat -> $
      ++ coerce(v) produces an ASP which returns the value of \spad{v}.
    coerce : List FortranCode -> $
      ++ coerce(e) takes an object from \spadtype{List FortranCode} and
      ++  uses it as the body of an ASP.
    coerce : FortranCode -> $
      ++ coerce(e) takes an object from \spadtype{FortranCode} and
      ++  uses it as the body of an ASP.
    coerce : Record(localSymbols:SymbolTable,code:List(FortranCode)) -> $
      ++ coerce(e) takes the component of \spad{e} from
      ++ \spadtype{List FortranCode} and uses it as the body of the ASP,
      ++ making the declarations in the \spadtype{SymbolTable} component.
```

———-

— **FMC.dotabb** —

```
"FMC"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FMC"];
"FMC" -> "FORTCAT"
```

———-

— **FMC.dotfull** —

```
"FortranMatrixCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FMC"];
"FortranMatrixCategory()" -> "FortranProgramCategory()"
```

———-

— FMC.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FortranMatrixCategory()" [color=lightblue];
"FortranMatrixCategory()" -> "FortranProgramCategory()"

"FortranProgramCategory()" [color=lightblue];
"FortranProgramCategory()" -> "Type()"
"FortranProgramCategory()" -> "CoercibleTo(OutputForm)"

"Type()" [color=lightblue];
"Type()" -> "Category"

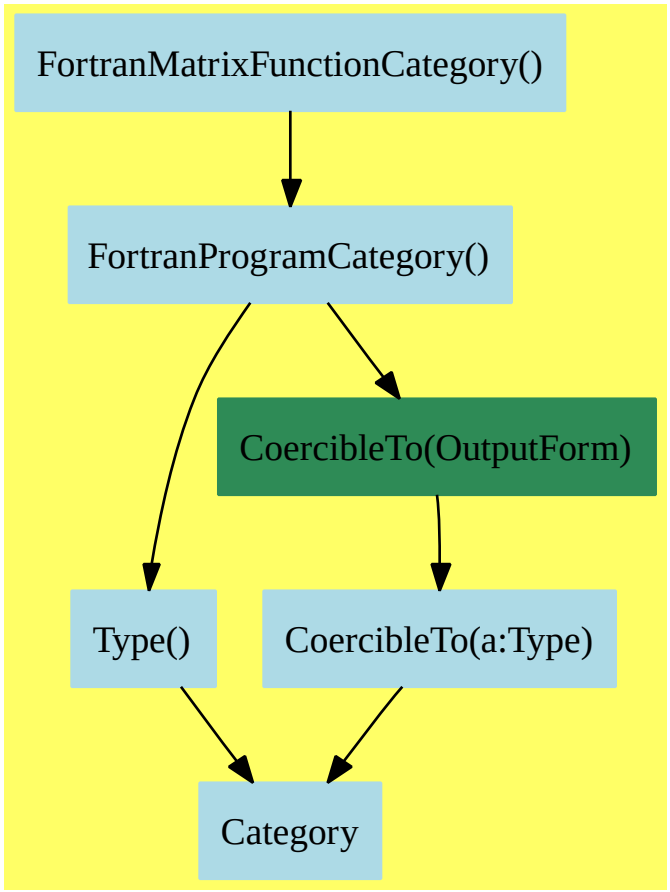
"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];
}
```

———-

4.0.41 FortranMatrixFunctionCategory (FMFUN)



— **FortranMatrixFunctionCategory.input** —

```
)set break resume
)sys rm -f FortranMatrixFunctionCategory.output
)spool FortranMatrixFunctionCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FortranMatrixFunctionCategory
--R FortranMatrixFunctionCategory  is a category constructor
--R Abbreviation for FortranMatrixFunctionCategory is FMFUN
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FMFUN
```

```
--R
--R------------------------------ Operations ------------------------------
--R coerce : FortranCode -> %             coerce : List FortranCode -> %
--R coerce : % -> OutputForm              outputAsFortran : % -> Void
--R coerce : Record(localSymbols: SymbolTable,code: List FortranCode) -> %
--R retract : Matrix Fraction Polynomial Integer -> %
--R retract : Matrix Fraction Polynomial Float -> %
--R retract : Matrix Polynomial Integer -> %
--R retract : Matrix Polynomial Float -> %
--R retract : Matrix Expression Integer -> %
--R retract : Matrix Expression Float -> %
--R retractIfCan : Matrix Fraction Polynomial Integer -> Union(%,"failed")
--R retractIfCan : Matrix Fraction Polynomial Float -> Union(%,"failed")
--R retractIfCan : Matrix Polynomial Integer -> Union(%,"failed")
--R retractIfCan : Matrix Polynomial Float -> Union(%,"failed")
--R retractIfCan : Matrix Expression Integer -> Union(%,"failed")
--R retractIfCan : Matrix Expression Float -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— FortranMatrixFunctionCategory.help —

```
=====================================================================
FortranMatrixFunctionCategory examples
=====================================================================

See Also:
o )show FortranMatrixFunctionCategory
```

———-

See:

⇐ “FortranProgramCategory” (FORTCAT) 3.0.26 on page 84

Exports:

coerce outputAsFortran retract retractIfCan

Attributes:

- **nil**

These are directly exported but not implemented:

```
 coerce : List FortranCode -> %
 coerce : FortranCode -> %
 coerce : Record(localSymbols: SymbolTable,code: List FortranCode) -> %
 retract : Matrix Expression Float -> %
 retract : Matrix Expression Integer -> %
 retract : Matrix Polynomial Float -> %
 retract : Matrix Polynomial Integer -> %
 retract : Matrix Fraction Polynomial Float -> %
 retract : Matrix Fraction Polynomial Integer -> %
 retractIfCan : Matrix Fraction Polynomial Integer -> Union(%,"failed")
 retractIfCan : Matrix Fraction Polynomial Float -> Union(%,"failed")
 retractIfCan : Matrix Polynomial Integer -> Union(%,"failed")
 retractIfCan : Matrix Polynomial Float -> Union(%,"failed")
 retractIfCan : Matrix Expression Integer -> Union(%,"failed")
 retractIfCan : Matrix Expression Float -> Union(%,"failed")
```

These exports come from (p84) FortranProgramCategory():

```
 coerce : % -> OutputForm
 outputAsFortran : % -> Void
```

— category FMFUN FortranMatrixFunctionCategory —

```
)abbrev category FMFUN FortranMatrixFunctionCategory
++ Author: Mike Dewar
++ Date Created: March 18 1994
++ Date Last Updated:
++ Related Constructors: FortranProgramCategory.
++ Description:
++ \axiomType{FortranMatrixFunctionCategory} provides support for
++ producing Functions and Subroutines representing matrices of
++ expressions.

FortranMatrixFunctionCategory():Category == FortranProgramCategory with
    coerce : List FortranCode -> $
      ++ coerce(e) takes an object from \spadtype{List FortranCode} and
      ++  uses it as the body of an ASP.
    coerce : FortranCode -> $
      ++ coerce(e) takes an object from \spadtype{FortranCode} and
      ++  uses it as the body of an ASP.
    coerce : Record(localSymbols:SymbolTable,code:List(FortranCode)) -> $
      ++ coerce(e) takes the component of \spad{e} from
      ++ \spadtype{List FortranCode} and uses it as the body of the ASP,
      ++ making the declarations in the \spadtype{SymbolTable} component.
    retract : Matrix Expression Float -> $
      ++ retract(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.
    retractIfCan : Matrix Expression Float -> Union($,"failed")
      ++ retractIfCan(e) tries to convert \spad{e} into an ASP, checking that
```

```
      ++  legal Fortran-77 is produced.
    retract : Matrix Expression Integer -> $
      ++ retract(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.
    retractIfCan : Matrix Expression Integer -> Union($,"failed")
      ++ retractIfCan(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.
    retract : Matrix Polynomial Float -> $
      ++ retract(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.
    retractIfCan : Matrix Polynomial Float -> Union($,"failed")
      ++ retractIfCan(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.
    retract : Matrix Polynomial Integer -> $
      ++ retract(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.
    retractIfCan : Matrix Polynomial Integer -> Union($,"failed")
      ++ retractIfCan(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.
    retract : Matrix Fraction Polynomial Float -> $
      ++ retract(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.
    retractIfCan : Matrix Fraction Polynomial Float -> Union($,"failed")
      ++ retractIfCan(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.
    retract : Matrix Fraction Polynomial Integer -> $
      ++ retract(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.
    retractIfCan : Matrix Fraction Polynomial Integer -> Union($,"failed")
      ++ retractIfCan(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.

    -- NB: These ASPs also have a coerce from an appropriate instantiation
    --     of Matrix FortranExpression.
```

———-

— FMFUN.dotabb —

```
"FMFUN"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FMFUN"];
"FMFUN" -> "FORTCAT"
```

———-

— FMFUN.dotfull —

```
"FortranMatrixFunctionCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FMFUN"];
"FortranMatrixFunctionCategory()" -> "FortranProgramCategory()"
```

———-

— **FMFUN.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FortranMatrixFunctionCategory()" [color=lightblue];
"FortranMatrixFunctionCategory()" -> "FortranProgramCategory()"

"FortranProgramCategory()" [color=lightblue];
"FortranProgramCategory()" -> "Type()"
"FortranProgramCategory()" -> "CoercibleTo(OutputForm)"

"Type()" [color=lightblue];
"Type()" -> "Category"

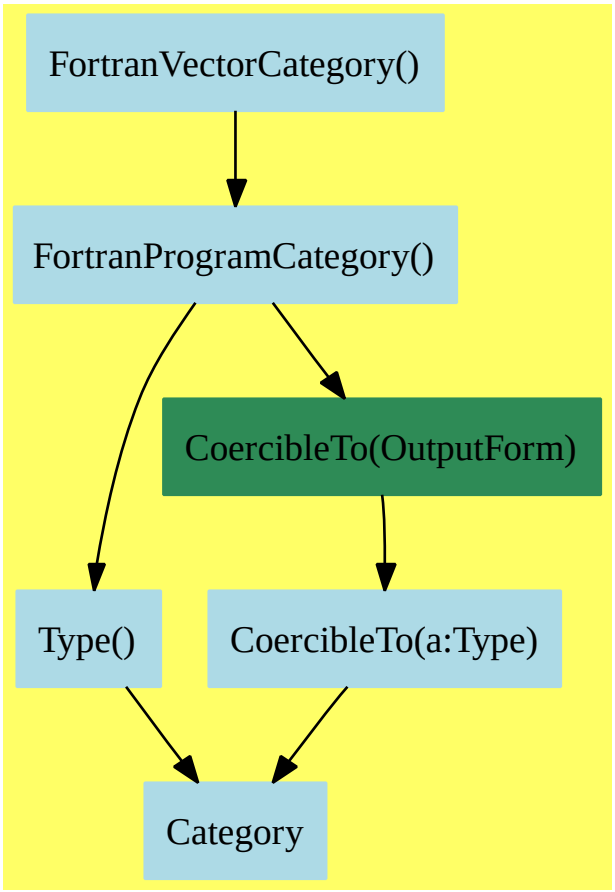
"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];
}
```

———-

4.0.42 FortranVectorCategory (FVC)



— **FortranVectorCategory.input** —

```
)set break resume
)sys rm -f FortranVectorCategory.output
)spool FortranVectorCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FortranVectorCategory
--R FortranVectorCategory  is a category constructor
--R Abbreviation for FortranVectorCategory is FVC
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FVC
```

```
--R
--R------------------------------ Operations ------------------------------
--R coerce : FortranCode -> %             coerce : List FortranCode -> %
--R coerce : Vector MachineFloat -> %     coerce : % -> OutputForm
--R outputAsFortran : % -> Void
--R coerce : Record(localSymbols: SymbolTable,code: List FortranCode) -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **FortranVectorCategory.help** —

```
=====================================================================
FortranVectorCategory examples
=====================================================================

See Also:
o )show FortranVectorCategory
```

———-

See:

⇐ “FortranProgramCategory” (FORTCAT) 3.0.26 on page 84

Exports:

Attributes:

- **nil**

These are directly exported but not implemented:

```
coerce : FortranCode -> %
coerce : List FortranCode -> %
coerce : Vector MachineFloat -> %
coerce : Record(localSymbols: SymbolTable,code: List FortranCode) -> %
```

These exports come from (p84) FortranProgramCategory():

```
coerce : % -> OutputForm
outputAsFortran : % -> Void
```

— category FVC FortranVectorCategory —

```
)abbrev category FVC FortranVectorCategory
++ Author: Mike Dewar
++ Date Created: October 1993
++ Date Last Updated: 18 March 1994
++ Related Constructors: FortranProgramCategory.
++ Description:
++ \axiomType{FortranVectorCategory} provides support for
++ producing Functions and Subroutines when the input to these
++ is an AXIOM object of type \axiomType{Vector} or in domains
++ involving \axiomType{FortranCode}.

FortranVectorCategory():Category == FortranProgramCategory with
    coerce : Vector MachineFloat -> $
      ++ coerce(v) produces an ASP which returns the value of \spad{v}.
    coerce : List FortranCode -> $
      ++ coerce(e) takes an object from \spadtype{List FortranCode} and
      ++  uses it as the body of an ASP.
    coerce : FortranCode -> $
      ++ coerce(e) takes an object from \spadtype{FortranCode} and
      ++  uses it as the body of an ASP.
    coerce : Record(localSymbols:SymbolTable,code:List(FortranCode)) -> $
      ++ coerce(e) takes the component of \spad{e} from
      ++ \spadtype{List FortranCode} and uses it as the body of the ASP,
      ++ making the declarations in the \spadtype{SymbolTable} component.
```

———-

— FVC.dotabb —

```
"FVC"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FVC"];
"FVC" -> "FORTCAT"
```

———-

— FVC.dotfull —

```
"FortranVectorCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FVC"];
"FortranVectorCategory()" -> "FortranProgramCategory()"
```

———-

— **FVC.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FortranVectorCategory()" [color=lightblue];
"FortranVectorCategory()" -> "FortranProgramCategory()"

"FortranProgramCategory()" [color=lightblue];
"FortranProgramCategory()" -> "Type()"
"FortranProgramCategory()" -> "CoercibleTo(OutputForm)"

"Type()" [color=lightblue];
"Type()" -> "Category"

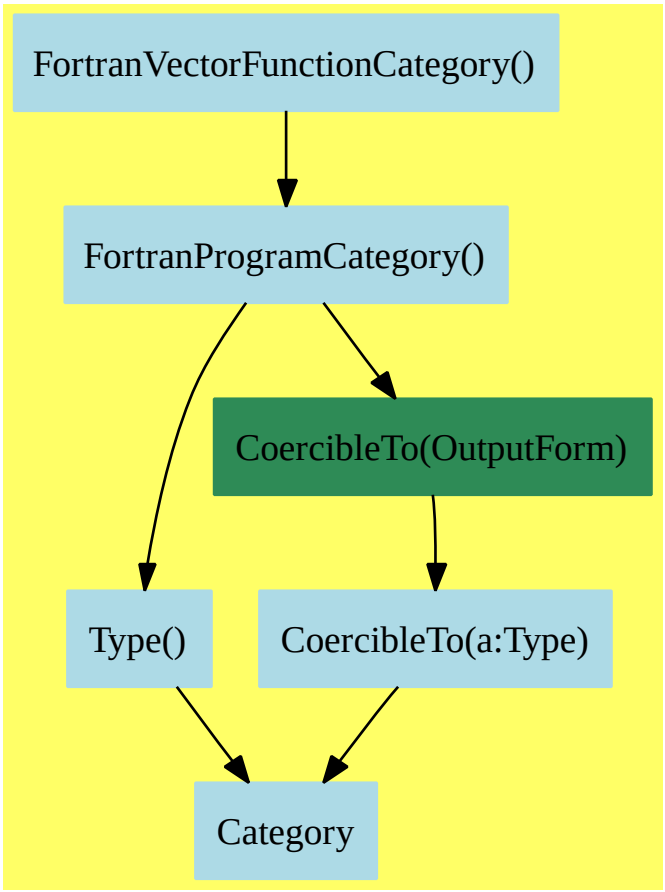
"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];
}
```

———-

4.0.43 FortranVectorFunctionCategory (FVFUN)



— **FortranVectorFunctionCategory.input** —

```
)set break resume
)sys rm -f FortranVectorFunctionCategory.output
)spool FortranVectorFunctionCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FortranVectorFunctionCategory
--R FortranVectorFunctionCategory  is a category constructor
--R Abbreviation for FortranVectorFunctionCategory is FVFUN
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FVFUN
```

```
--R
--R------------------------------ Operations --------------------------------
--R coerce : FortranCode -> %             coerce : List FortranCode -> %
--R coerce : % -> OutputForm              outputAsFortran : % -> Void
--R coerce : Record(localSymbols: SymbolTable,code: List FortranCode) -> %
--R retract : Vector Fraction Polynomial Integer -> %
--R retract : Vector Fraction Polynomial Float -> %
--R retract : Vector Polynomial Integer -> %
--R retract : Vector Polynomial Float -> %
--R retract : Vector Expression Integer -> %
--R retract : Vector Expression Float -> %
--R retractIfCan : Vector Fraction Polynomial Integer -> Union(%,"failed")
--R retractIfCan : Vector Fraction Polynomial Float -> Union(%,"failed")
--R retractIfCan : Vector Polynomial Integer -> Union(%,"failed")
--R retractIfCan : Vector Polynomial Float -> Union(%,"failed")
--R retractIfCan : Vector Expression Integer -> Union(%,"failed")
--R retractIfCan : Vector Expression Float -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **FortranVectorFunctionCategory.help** —

```
=====================================================================
FortranVectorFunctionCategory examples
=====================================================================

See Also:
o )show FortranVectorFunctionCategory
```

———-

See:

⇐ “FortranProgramCategory” (FORTCAT) 3.0.26 on page 84

Exports:

coerce outputAsFortran retract retractIfCan

Attributes:

- **nil**

These are directly exported but not implemented:

```
 coerce : FortranCode -> %
 coerce : List FortranCode -> %
 coerce : Record(localSymbols: SymbolTable,code: List FortranCode) -> %
 retract : Vector Fraction Polynomial Integer -> %
 retract : Vector Fraction Polynomial Float -> %
 retract : Vector Polynomial Integer -> %
 retract : Vector Polynomial Float -> %
 retract : Vector Expression Integer -> %
 retract : Vector Expression Float -> %
 retractIfCan : Vector Fraction Polynomial Integer -> Union(%,"failed")
 retractIfCan : Vector Fraction Polynomial Float -> Union(%,"failed")
 retractIfCan : Vector Polynomial Integer -> Union(%,"failed")
 retractIfCan : Vector Polynomial Float -> Union(%,"failed")
 retractIfCan : Vector Expression Integer -> Union(%,"failed")
 retractIfCan : Vector Expression Float -> Union(%,"failed")
```

These exports come from (p84) FortranProgramCategory():

```
 coerce : % -> OutputForm
 outputAsFortran : % -> Void
```

— category FVFUN FortranVectorFunctionCategory —

```
)abbrev category FVFUN FortranVectorFunctionCategory
++ Author: Mike Dewar
++ Date Created: 11 March 1994
++ Date Last Updated: 18 March 1994
++ Related Constructors: FortranProgramCategory.
++ Description:
++ \axiomType{FortranVectorFunctionCategory} is the catagory of arguments
++ to NAG Library routines which return the values of vectors of functions.

FortranVectorFunctionCategory():Category == FortranProgramCategory with
    coerce : List FortranCode -> $
      ++ coerce(e) takes an object from \spadtype{List FortranCode} and
      ++  uses it as the body of an ASP.
    coerce : FortranCode -> $
      ++ coerce(e) takes an object from \spadtype{FortranCode} and
      ++  uses it as the body of an ASP.
    coerce : Record(localSymbols:SymbolTable,code:List(FortranCode)) -> $
      ++ coerce(e) takes the component of \spad{e} from
      ++ \spadtype{List FortranCode} and uses it as the body of the ASP,
      ++ making the declarations in the \spadtype{SymbolTable} component.
    retract : Vector Expression Float -> $
      ++ retract(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.
    retractIfCan : Vector Expression Float -> Union($,"failed")
      ++ retractIfCan(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.
```

```
    retract : Vector Expression Integer -> $
      ++ retract(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.
    retractIfCan : Vector Expression Integer -> Union($,"failed")
      ++ retractIfCan(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.
    retract : Vector Polynomial Float -> $
      ++ retract(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.
    retractIfCan : Vector Polynomial Float -> Union($,"failed")
      ++ retractIfCan(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.
    retract : Vector Polynomial Integer -> $
      ++ retract(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.
    retractIfCan : Vector Polynomial Integer -> Union($,"failed")
      ++ retractIfCan(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.
    retract : Vector Fraction Polynomial Float -> $
      ++ retract(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.
    retractIfCan : Vector Fraction Polynomial Float -> Union($,"failed")
      ++ retractIfCan(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.
    retract : Vector Fraction Polynomial Integer -> $
      ++ retract(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.
    retractIfCan : Vector Fraction Polynomial Integer -> Union($,"failed")
      ++ retractIfCan(e) tries to convert \spad{e} into an ASP, checking that
      ++  legal Fortran-77 is produced.

    -- NB: These ASPs also have a coerce from an appropriate instantiation
    --     of Vector FortranExpression.
```

———-

— FVFUN.dotabb —

```
"FVFUN"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FVFUN"];
"FVFUN" -> "FORTCAT"
```

———-

— FVFUN.dotfull —

```
"FortranVectorFunctionCategory()"
```

```
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FVFUN"];
"FortranVectorFunctionCategory()" -> "FortranProgramCategory()"
```

———-

— FVFUN.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FortranVectorFunctionCategory()" [color=lightblue];
"FortranVectorFunctionCategory()" -> "FortranProgramCategory()"

"FortranProgramCategory()" [color=lightblue];
"FortranProgramCategory()" -> "Type()"
"FortranProgramCategory()" -> "CoercibleTo(OutputForm)"

"Type()" [color=lightblue];
"Type()" -> "Category"

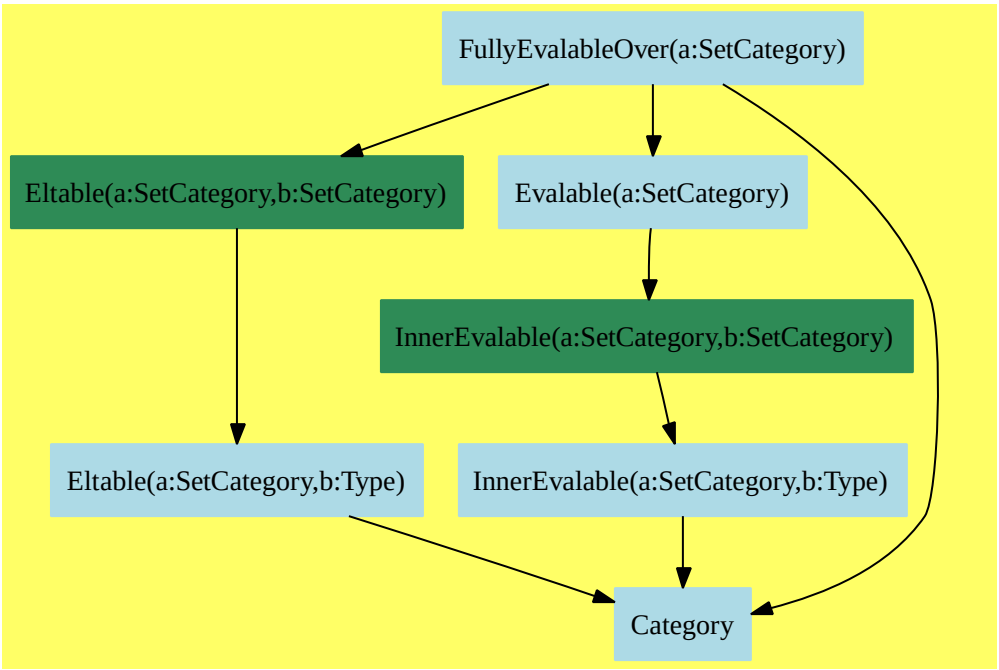
"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];
}
```

———-

4.0.44 FullyEvalableOver (FEVALAB)



— FullyEvalableOver.input —

```
)set break resume
)sys rm -f FullyEvalableOver.output
)spool FullyEvalableOver.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FullyEvalableOver
--R FullyEvalableOver R: SetCategory  is a category constructor
--R Abbreviation for FullyEvalableOver is FEVALAB
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FEVALAB
--R
--R------------------------------ Operations --------------------------------
--R map : ((R -> R),%) -> %
--R ?.? : (%,R) -> % if R has ELTAB(R,R)
--R eval : (%,List R,List R) -> % if R has EVALAB R
--R eval : (%,R,R) -> % if R has EVALAB R
--R eval : (%,Equation R) -> % if R has EVALAB R
```

```
--R eval : (%,List Equation R) -> % if R has EVALAB R
--R eval : (%,List Symbol,List R) -> % if R has IEVALAB(SYMBOL,R)
--R eval : (%,Symbol,R) -> % if R has IEVALAB(SYMBOL,R)
--R
--E 1

)spool
)lisp (bye)
```

———-

— **FullyEvalableOver.help** —

```
=====================================================================
FullyEvalableOver examples
=====================================================================

See Also:
o )show FullyEvalableOver
```

———-

See:

⇒ “ComplexCategory” (COMPCAT) 20.0.216 on page 1619
⇒ “OctonionCategory” (OC) 12.0.166 on page 1059
⇒ “QuaternionCategory” (QUATCAT) 12.0.167 on page 1071
⇒ “QuotientFieldCategory” (QFCAT) 17.0.198 on page 1376
⇐ “Category” (CATEGORY) 2.0.1 on page 3

Exports:

eval map ?.?

These are directly exported but not implemented:

```
map : ((R -> R),%) -> %
```

These are implemented by this category:

```
?.? : (%,R) -> % if R has ELTAB(R,R)
eval : (%,Equation R) -> % if R has EVALAB R
eval : (%,List Symbol,List R) -> % if R has IEVALAB(SYMBOL,R)
```

These exports come from (p80) Evalable(a:Type):

```
eval : (%,List Equation R) -> % if R has EVALAB R
eval : (%,R,R) -> % if R has EVALAB R
eval : (%,List R,List R) -> % if R has EVALAB R
```

These exports come from (p35) InnerEvalable(a:Symbol,b:SetCategory):

```
 eval : (%,Symbol,R) -> % if R has IEVALAB(SYMBOL,R)
```

— category FEVALAB FullyEvalableOver —

```
)abbrev category FEVALAB FullyEvalableOver
++ Author:
++ Date Created:
++ Date Last Updated: June 3, 1991
++ Basic Operations:
++ Related Domains: Equation
++ Also See:
++ AMS Classifications:
++ Keywords: equation
++ Examples:
++ References:
++ Description:
++ This category provides a selection of evaluation operations
++ depending on what the argument type R provides.

FullyEvalableOver(R:SetCategory): Category == with
    map: (R -> R, $) -> $
        ++ map(f, ex) evaluates ex, applying f to values of type R in ex.
    if R has Eltable(R, R) then Eltable(R, $)
    if R has Evalable(R) then Evalable(R)
    if R has InnerEvalable(Symbol, R) then InnerEvalable(Symbol, R)
 add
    if R has Eltable(R, R) then
      elt(x:$, r:R) == map(y +-> y(r), x)

    if R has Evalable(R) then
      eval(x:$, l:List Equation R) == map(y +-> eval(y, l), x)

    if R has InnerEvalable(Symbol, R) then
      eval(x:$, ls:List Symbol, lv:List R) == map(y +-> eval(y, ls, lv), x)
```

———-

— FEVALAB.dotabb —

```
"FEVALAB"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FEVALAB"];
"FEVALAB" -> "ELTAB"
"FEVALAB" -> "EVALAB"
"FEVALAB" -> "IEVALAB"
"FEVALAB" -> "CATEGORY"
```

———-

— **FEVALAB.dotfull** —

```
"FullyEvalableOver(a:SetCategory)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FEVALAB"];
"FullyEvalableOver(a:SetCategory)" -> "Eltable(a:SetCategory,b:Type)"
"FullyEvalableOver(a:SetCategory)" -> "Evalable(a:SetCategory)"
"FullyEvalableOver(a:SetCategory)" -> "Category"

"FullyEvalableOver(IntegralDomain)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=FEVALAB"];
"FullyEvalableOver(IntegralDomain)" ->
  "FullyEvalableOver(a:SetCategory)"

"FullyEvalableOver(CommutativeRing)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=FEVALAB"];
"FullyEvalableOver(CommutativeRing)" ->
  "FullyEvalableOver(a:SetCategory)"
```

———-

— **FEVALAB.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FullyEvalableOver(a:SetCategory)" [color=lightblue];
"FullyEvalableOver(a:SetCategory)" -> "Eltable(a:SetCategory,b:SetCategory)"
"FullyEvalableOver(a:SetCategory)" -> "Evalable(a:SetCategory)"
"FullyEvalableOver(a:SetCategory)" -> "Category"

"Eltable(a:SetCategory,b:SetCategory)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=ELTAB"];
"Eltable(a:SetCategory,b:SetCategory)" ->
   "Eltable(a:SetCategory,b:Type)"

"Eltable(a:SetCategory,b:Type)" [color=lightblue];
"Eltable(a:SetCategory,b:Type)" -> "Category"

"Evalable(a:SetCategory)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=EVALAB"];
"Evalable(a:SetCategory)" -> "InnerEvalable(a:SetCategory,b:SetCategory)"

"InnerEvalable(a:SetCategory,b:SetCategory)" [color=seagreen];
"InnerEvalable(a:SetCategory,b:SetCategory)" ->
```

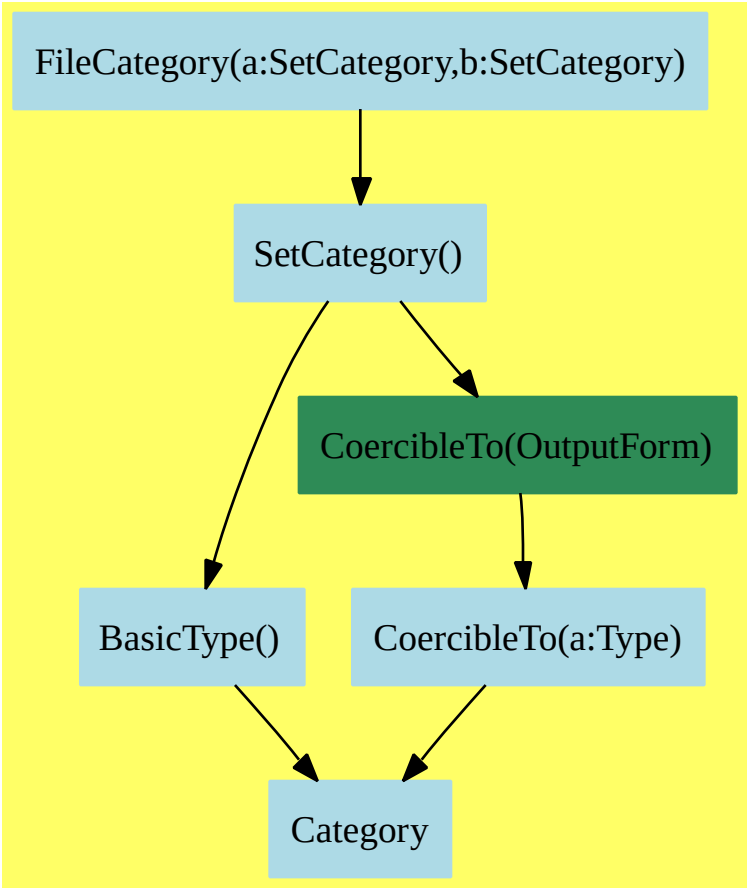
```
    "InnerEvalable(a:SetCategory,b:Type)"

"InnerEvalable(a:SetCategory,b:Type)" [color=lightblue];
"InnerEvalable(a:SetCategory,b:Type)" -> "Category"

"Category" [color=lightblue];
}
```

———-

4.0.45 FileCategory (FILECAT)



— **FileCategory.input** —

```
)set break resume
)sys rm -f FileCategory.output
)spool FileCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FileCategory
--R FileCategory(Name: SetCategory,S: SetCategory)  is a category constructor
--R Abbreviation for FileCategory is FILECAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FILECAT
--R
--R------------------------------ Operations ------------------------------
--R ?=? : (%,%) -> Boolean               close! : % -> %
--R coerce : % -> OutputForm             flush : % -> Void
--R hash : % -> SingleInteger            iomode : % -> String
--R latex : % -> String                  name : % -> Name
--R open : (Name,String) -> %            open : Name -> %
--R read! : % -> S                       reopen! : (%,String) -> %
--R write! : (%,S) -> S                  ?~=? : (%,%) -> Boolean
--R
--E 1

)spool
)lisp (bye)
```

———-

— FileCategory.help —

```
====================================================================
FileCategory examples
====================================================================

See Also:
o )show FileCategory
```

———-

See:

⇐ “SetCategory” (SETCAT) 3.0.34 on page 113

Exports:

close!	coerce	hash	iomode	latex
name	open	read!	reopen!	write!
?=?	?~=?			

These are directly exported but not implemented:

```
close! : % -> %
iomode : % -> String
name : % -> Name
open : Name -> %
open : (Name,String) -> %
read! : % -> S
reopen! : (%,String) -> %
write! : (%,S) -> S
```

These exports come from SetCategory():

```
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
```

— category FILECAT FileCategory —

```
)abbrev category FILECAT FileCategory
++ Author: Stephen M. Watt, Victor Miller
++ Date Created:
++ Date Last Updated: June 4, 1991
++ Basic Operations:
++ Related Domains: File
++ Also See:
++ AMS Classifications:
++ Keywords:
++ Examples:
++ References:
++ Description:
++ This category provides an interface to operate on files in the
++ computer's file system.  The precise method of naming files
++ is determined by the Name parameter.  The type of the contents
++ of the file is determined by S.

FileCategory(Name, S): Category == FCdefinition where
    Name:      SetCategory
    S:         SetCategory
    IOMode ==> String  -- Union("input", "output", "closed")

    FCdefinition == SetCategory with
        open: Name -> %
```

```
      ++ open(s) returns the file s open for input.

    open: (Name, IOMode) -> %
      ++ open(s,mode) returns a file s open for operation in the
      ++ indicated mode: "input" or "output".

    reopen_!: (%, IOMode) -> %
      ++ reopen!(f,mode) returns a file f reopened for operation in the
      ++ indicated mode: "input" or "output".
      ++ \spad{reopen!(f,"input")} will reopen the file f for input.

    close_!: % -> %
      ++ close!(f) returns the file f closed to input and output.

    name: % -> Name
      ++ name(f) returns the external name of the file f.

    iomode: % -> IOMode
      ++ iomode(f) returns the status of the file f. The input/output
      ++ status of f may be "input", "output" or "closed" mode.

    read_!: % -> S
      ++ read!(f) extracts a value from file f.  The state of f is
      ++ modified so a subsequent call to \spadfun{read!} will return
      ++ the next element.

    write_!: (%,S) -> S
      ++ write!(f,s) puts the value s into the file f.
      ++ The state of f is modified so subsequents call to \spad{write!}
      ++ will append one after another.

    flush: % -> Void
      ++ flush(f) makes sure that buffered data is written out
```

———-

— FILECAT.dotabb —

```
"FILECAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FILECAT"];
"FILECAT" -> "SETCAT"
```

———-

— FILECAT.dotfull —

```
"FileCategory(a:SetCategory,b:SetCategory)"
```

```
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FILECAT"];
"FileCategory(a:SetCategory,b:SetCategory)" -> "SetCategory()"
```

———-

— FILECAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FileCategory(a:SetCategory,b:SetCategory)" [color=lightblue];
"FileCategory(a:SetCategory,b:SetCategory)" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

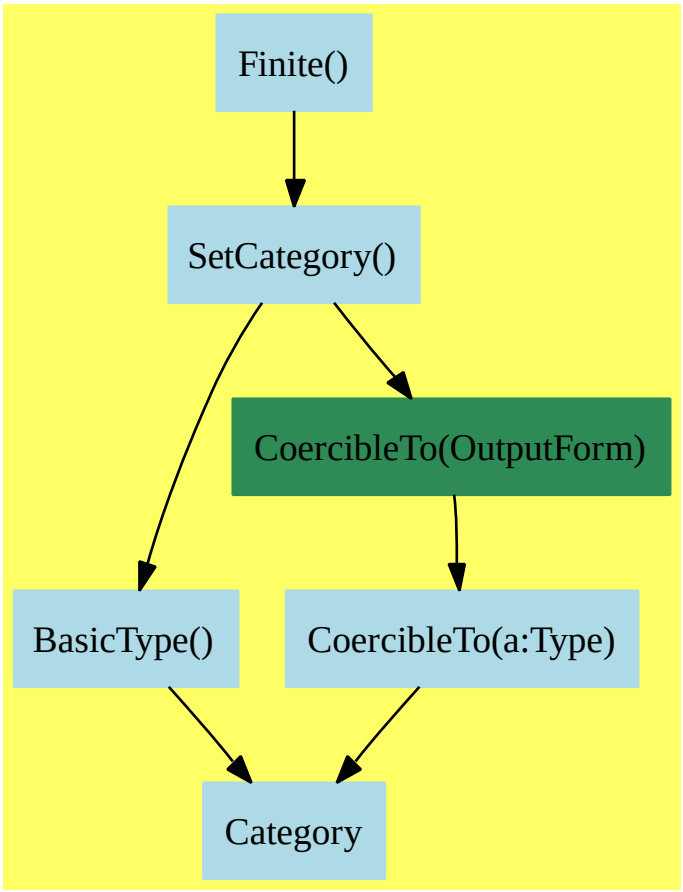
"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];

}
```

———-

4.0.46 Finite (FINITE)



— **Finite.input** —

```
)set break resume
)sys rm -f Finite.output
)spool Finite.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show Finite
--R Finite  is a category constructor
--R Abbreviation for Finite is FINITE
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FINITE
```

```
--R
--R------------------------------ Operations --------------------------------
--R ?=? : (%,%) -> Boolean                coerce : % -> OutputForm
--R hash : % -> SingleInteger             index : PositiveInteger -> %
--R latex : % -> String                   lookup : % -> PositiveInteger
--R random : () -> %                      size : () -> NonNegativeInteger
--R ?~=? : (%,%) -> Boolean
--R
--E 1

)spool
)lisp (bye)
```

———-

— **Finite.help** —

```
=====================================================================
Finite examples
=====================================================================

See Also:
o )show Finite
```

———-

See:

⇒ “DirectProductCategory” (DIRPCAT) 12.0.160 on page 1001
⇒ “FiniteFieldCategory” (FFIELDC) 18.0.207 on page 1518
⇒ “OrderedFinite” (ORDFIN) 5.0.77 on page 336
⇐ “SetCategory” (SETCAT) 3.0.34 on page 113

Exports:

coerce	hash	index	latex	lookup	random
size		?=?	?~=?		

These are directly exported but not implemented:

```
index : PositiveInteger -> %
lookup : % -> PositiveInteger
random : () -> %
size : () -> NonNegativeInteger
```

These exports come from (p113) SetCategory():

```
coerce : % -> OutputForm
```

```
 hash : % -> SingleInteger
 latex : % -> String
 ?=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
```

— category FINITE Finite —

```
)abbrev category FINITE Finite
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The category of domains composed of a finite set of elements.
++ We include the functions \spadfun{lookup} and \spadfun{index}
++ to give a bijection between the finite set and an initial
++ segment of positive integers.
++
++ Axioms:\br
++ \tab{5}\spad{lookup(index(n)) = n}\br
++ \tab{5}\spad{index(lookup(s)) = s}

Finite(): Category == SetCategory with
      size: () ->  NonNegativeInteger
        ++ size() returns the number of elements in the set.
      index: PositiveInteger -> %
        ++ index(i) takes a positive integer i less than or equal
        ++ to \spad{size()} and
        ++ returns the \spad{i}-th element of the set.
        ++ This operation establishs a bijection
        ++ between the elements of the finite set and \spad{1..size()}.
      lookup: % -> PositiveInteger
        ++ lookup(x) returns a positive integer such that
        ++ \spad{x = index lookup x}.
      random: () -> %
        ++ random() returns a random element from the set.
```

———-

— FINITE.dotabb —

```
"FINITE"
```

```
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FINITE"];
"FINITE" -> "SETCAT"
```

———-

— **FINITE.dotfull** —

```
"Finite()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FINITE"];
"Finite()" -> "SetCategory()"
```

———-

— **FINITE.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"Finite()" [color=lightblue];
"Finite()" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

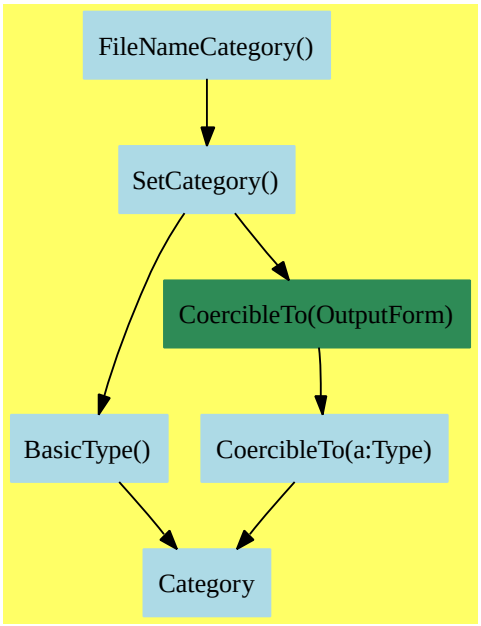
"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];

}
```

———-

4.0.47 FileNameCategory (FNCAT)



— FileNameCategory.input —

```
)set break resume
)sys rm -f FileNameCategory.output
)spool FileNameCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FileNameCategory
--R FileNameCategory  is a category constructor
--R Abbreviation for FileNameCategory is FNCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FNCAT
--R
--R------------------------------ Operations ------------------------------
--R ?=? : (%,%) -> Boolean                coerce : % -> String
--R coerce : String -> %                  coerce : % -> OutputForm
--R directory : % -> String               exists? : % -> Boolean
--R extension : % -> String               hash : % -> SingleInteger
--R latex : % -> String                   name : % -> String
--R new : (String,String,String) -> %     readable? : % -> Boolean
--R writable? : % -> Boolean              ?~=? : (%,%) -> Boolean
```

```
--R filename : (String,String,String) -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **FileNameCategory.help** —

```
==========================================================================
FileNameCategory examples
==========================================================================

See Also:
o )show FileNameCategory
```

———-

See:

⇐ “SetCategory” (SETCAT) 3.0.34 on page 113

Exports:

coerce	directory	exists?	extension	filename
hash	latex	name	new	readable?
writable?	?=?	?~=?		

These are directly exported but not implemented:

```
coerce : String -> %
coerce : % -> String
directory : % -> String
exists? : % -> Boolean
extension : % -> String
filename : (String,String,String) -> %
name : % -> String
new : (String,String,String) -> %
readable? : % -> Boolean
writable? : % -> Boolean
```

These exports come from (p113) SetCategory():

```
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
```

```
 ?=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
```

— category FNCAT FileNameCategory —

```
)abbrev category FNCAT FileNameCategory
++ Author: Stephen M. Watt
++ Date Created: 1985
++ Date Last Updated: June 20, 1991
++ Basic Operations:
++ Related Domains:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ Examples:
++ References:
++ Description:
++ This category provides an interface to names in the file system.

FileNameCategory(): Category == SetCategory with
   coerce: String -> %
     ++ coerce(s) converts a string to a file name
     ++ according to operating system-dependent conventions.
   coerce: % -> String
     ++ coerce(fn) produces a string for a file name
     ++ according to operating system-dependent conventions.
   filename: (String, String, String) -> %
     ++ filename(d,n,e) creates a file name with
     ++ d as its directory, n as its name and e as its extension.
     ++ This is a portable way to create file names.
     ++ When d or t is the empty string, a default is used.
   directory: % -> String
     ++ directory(f) returns the directory part of the file name.
   name: % -> String
     ++ name(f) returns the name part of the file name.
   extension: % -> String
     ++ extension(f) returns the type part of the file name.
   exists?: % -> Boolean
     ++ exists?(f) tests if the file exists in the file system.
   readable?: % -> Boolean
     ++ readable?(f) tests if the named file exist and can it be opened
     ++ for reading.
   writable?: % -> Boolean
     ++ writable?(f) tests if the named file be opened for writing.
     ++ The named file need not already exist.
   new: (String, String, String) -> %
     ++ new(d,pref,e) constructs the name of a new writable file with
     ++ d as its directory, pref as a prefix of its name and
     ++ e as its extension.
```

```
     ++ When d or t is the empty string, a default is used.
     ++ An error occurs if a new file cannot be written in the given
     ++ directory.
```

———-

— **FNCAT.dotabb** —

```
"FNCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FNCAT"];
"FNCAT" -> "SETCAT"
```

———-

— **FNCAT.dotfull** —

```
"FileNameCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FNCAT"];
"FileNameCategory()" -> "SetCategory()"
```

———-

— **FNCAT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FileNameCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FNCAT"];
"FileNameCategory()" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

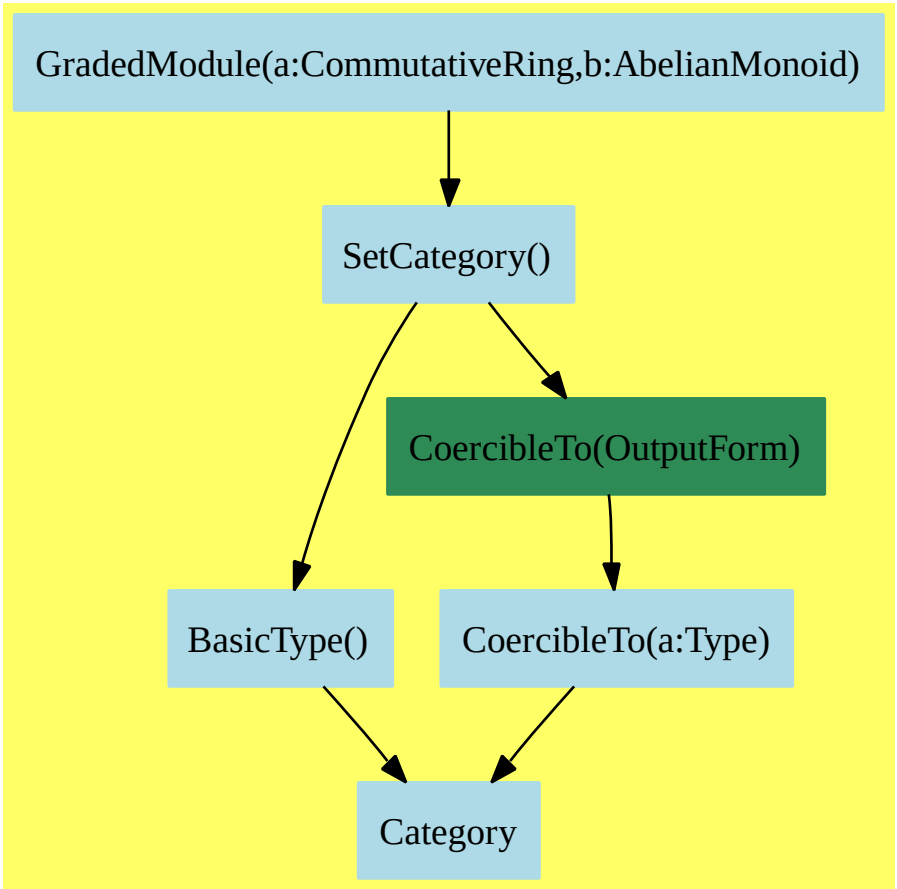
"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"
```

```
"Category" [color=lightblue];

}
```

———-

4.0.48 GradedModule (GRMOD)



— **GradedModule.input** —

```
)set break resume
)sys rm -f GradedModule.output
```

```
)spool GradedModule.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show GradedModule
--R GradedModule(R: CommutativeRing,E: AbelianMonoid)  is a category constructor
--R Abbreviation for GradedModule is GRMOD
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for GRMOD
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,R) -> %                    ?*? : (R,%) -> %
--R ?+? : (%,%) -> %                    ?-? : (%,%) -> %
--R -? : % -> %                         ?=? : (%,%) -> Boolean
--R 0 : () -> %                         coerce : % -> OutputForm
--R degree : % -> E                     hash : % -> SingleInteger
--R latex : % -> String                 ?~=? : (%,%) -> Boolean
--R
--E 1

)spool
)lisp (bye)
```

———-

— **GradedModule.help** —

```
=====================================================================
GradedModule examples
=====================================================================

See Also:
o )show GradedModule
```

———-

See:

⇒ “GradedAlgebra” (GRALG) 5.0.73 on page 313
⇐ “SetCategory” (SETCAT) 3.0.34 on page 113

Exports:

0	coerce	degree	hash	latex
?~=?	?*?	?+?	?-?	-?
?=?				

These are directly exported but not implemented:

```
0 : () -> %
degree : % -> E
?*? : (%,R) -> %
?*? : (R,%) -> %
-? : % -> %
?+? : (%,%) -> %
```

These are implemented by this category:

```
?-? : (%,%) -> %
```

These exports come from (p113) SetCategory():

```
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
?~=? : (%,%) -> Boolean
?=? : (%,%) -> Boolean
```

— category GRMOD GradedModule —

```
)abbrev category GRMOD GradedModule
++ Author: Stephen M. Watt
++ Date Created: May 20, 1991
++ Date Last Updated: May 20, 1991
++ Basic Operations: +, *, degree
++ Related Domains: CartesianTensor(n,dim,R)
++ Also See:
++ AMS Classifications:
++ Keywords: graded module, tensor, multi-linear algebra
++ Examples:
++ References: Algebra 2d Edition, MacLane and Birkhoff, MacMillan 1979
++ Description:
++ GradedModule(R,E) denotes ‘‘E-graded R-module’’, i.e. collection of
++ R-modules indexed by an abelian monoid E.
++ An element \spad{g} of \spad{G[s]} for some specific \spad{s} in \spad{E}
++ is said to be an element of \spad{G} with degree \spad{s}.
++ Sums are defined in each module \spad{G[s]} so two elements of \spad{G}
++ have a sum if they have the same degree.
++
++ Morphisms can be defined and composed by degree to give the
++ mathematical category of graded modules.

GradedModule(R: CommutativeRing, E: AbelianMonoid): Category ==
    SetCategory with
        degree: % -> E
```

```
            ++ degree(g) names the degree of g.  The set of all elements
            ++ of a given degree form an R-module.
        0: constant -> %
            ++ 0 denotes the zero of degree 0.
        _*: (R, %) -> %
            ++ r*g is left module multiplication.
        _*: (%, R) -> %
            ++ g*r is right module multiplication.

        _-: % -> %
            ++ -g is the additive inverse of g in the module of elements
            ++ of the same grade as g.
        _+: (%, %) -> %
            ++ g+h is the sum of g and h in the module of elements of
            ++ the same degree as g and h.  Error: if g and h
            ++ have different degrees.
        _-: (%, %) -> %
            ++ g-h is the difference of g and h in the module of elements of
            ++ the same degree as g and h.  Error: if g and h
            ++ have different degrees.
  add
        (x: %) - (y: %) == x+(-y)
```

———-

— **GRMOD.dotabb** —

```
"GRMOD"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=GRMOD"];
"GRMOD" -> "SETCAT"
```

———-

— **GRMOD.dotfull** —

```
"GradedModule(a:CommutativeRing,b:AbelianMonoid)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=GRMOD"];
"GradedModule(a:CommutativeRing,b:AbelianMonoid)" -> "SetCategory()"
```

———-

— **GRMOD.dotpic** —

```
digraph pic {
 fontsize=10;
```

```
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"GradedModule(a:CommutativeRing,b:AbelianMonoid)" [color=lightblue];
"GradedModule(a:CommutativeRing,b:AbelianMonoid)" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

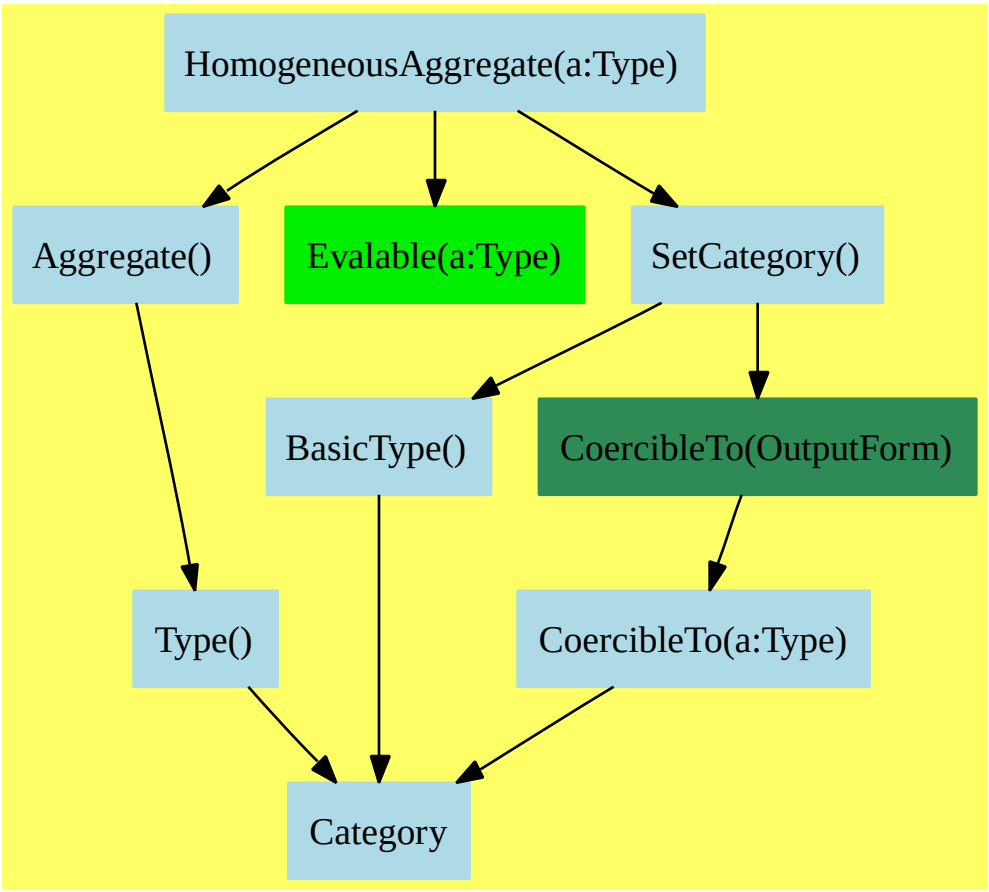
"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];

}
```

———-

4.0.49 HomogeneousAggregate (HOAGG)



— **HomogeneousAggregate.input** —

```
)set break resume
)sys rm -f HomogeneousAggregate.output
)spool HomogeneousAggregate.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show HomogeneousAggregate
--R HomogeneousAggregate S: Type  is a category constructor
--R Abbreviation for HomogeneousAggregate is HOAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for HOAGG
```

```
--R
--R------------------------------ Operations ------------------------------
--R copy : % -> %                          empty : () -> %
--R empty? : % -> Boolean                 eq? : (%,%) -> Boolean
--R map : ((S -> S),%) -> %                sample : () -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R coerce : % -> OutputForm if S has SETCAT
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R hash : % -> SingleInteger if S has SETCAT
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List S if $ has finiteAggregate
--R size? : (%,NonNegativeInteger) -> Boolean
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— HomogeneousAggregate.help —

```
=====================================================================
HomogeneousAggregate examples
=====================================================================

See Also:
o )show HomogeneousAggregate
```

———-

See:

⇒ “BagAggregate” (BGAGG) 5.0.68 on page 277

⇒ “Collection” (CLAGG) 5.0.70 on page 287
⇒ “IndexedAggregate” (IXAGG) 5.0.74 on page 318
⇒ “RectangularMatrixCategory” (RMATCAT) 10.0.147 on page 901
⇒ “RecursiveAggregate” (RCAGG) 5.0.80 on page 351
⇒ “TwoDimensionalArrayCategory” (ARR2CAT) 5.0.81 on page 357
⇐ “Aggregate” (AGG) 3.0.22 on page 69
⇐ “Evalable” (EVALAB) 3.0.25 on page 80
⇐ “SetCategory” (SETCAT) 3.0.34 on page 113

Attributes exported:

- **nil**

Exports:

any?	coerce	copy	count	empty
empty?	eq?	eval	every?	hash
latex	less?	map	map!	members
member?	more?	parts	sample	size?
#?	?=?	?~=?		

Attributes Used:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.

These are directly exported but not implemented:

```
map : ((S -> S),%) -> %
map! : ((S -> S),%) -> % if $ has shallowlyMutable
parts : % -> List S if $ has finiteAggregate
```

These are implemented by this category:

```
any? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
coerce : % -> OutputForm if S has SETCAT
count : (S,%) -> NonNegativeInteger
         if S has SETCAT and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
eval : (%,List Equation S) -> %
         if S has EVALAB S and S has SETCAT
every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
members : % -> List S if $ has finiteAggregate
member? : (S,%) -> Boolean
```

```
          if S has SETCAT and $ has finiteAggregate
 #? : % -> NonNegativeInteger if $ has finiteAggregate
 ?=? : (%,%) -> Boolean if S has SETCAT
```

These exports come from (p69) Aggregate:

```
 copy : % -> %
 empty : () -> %
 empty? : % -> Boolean
 eq? : (%,%) -> Boolean
 less? : (%,NonNegativeInteger) -> Boolean
 more? : (%,NonNegativeInteger) -> Boolean
 sample : () -> %
 size? : (%,NonNegativeInteger) -> Boolean
```

These exports come from (p80) Evalable(a:Type):

```
 eval : (%,List S,List S) -> %
          if S has EVALAB S and S has SETCAT
 eval : (%,S,S) -> %
          if S has EVALAB S and S has SETCAT
 eval : (%,Equation S) -> %
          if S has EVALAB S and S has SETCAT
```

These exports come from (p113) SetCategory():

```
 hash : % -> SingleInteger if S has SETCAT
 ?~=? : (%,%) -> Boolean if S has SETCAT
 latex : % -> String if S has SETCAT
```

— category HOAGG HomogeneousAggregate —

```
)abbrev category HOAGG HomogeneousAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991, May 1995
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A homogeneous aggregate is an aggregate of elements all of the
++ same type.
++ In the current system, all aggregates are homogeneous.
++ Two attributes characterize classes of aggregates.
++ Aggregates from domains with attribute \spadatt{finiteAggregate}
```

```
++ have a finite number of members.
++ Those with attribute \spadatt{shallowlyMutable} allow an element
++ to be modified or updated without changing its overall value.

HomogeneousAggregate(S:Type): Category == Aggregate with
   if S has SetCategory then SetCategory
   if S has SetCategory then
      if S has Evalable S then Evalable S
   map           : (S->S,%) -> %
     ++ map(f,u) returns a copy of u with each element x replaced by f(x).
     ++ For collections, \axiom{map(f,u) = [f(x) for x in u]}.
   if % has shallowlyMutable then
     map_!: (S->S,%) -> %
        ++ map!(f,u) destructively replaces each element x of u
        ++ by \axiom{f(x)}.
   if % has finiteAggregate then
      any?: (S->Boolean,%) -> Boolean
        ++ any?(p,u) tests if \axiom{p(x)} is true for any element x of u.
        ++ Note that for collections,
        ++ \axiom{any?(p,u) = reduce(or,map(f,u),false,true)}.
      every?: (S->Boolean,%) -> Boolean
        ++ every?(f,u) tests if p(x) is true for all elements x of u.
        ++ Note that for collections,
        ++ \axiom{every?(p,u) = reduce(and,map(f,u),true,false)}.
      count: (S->Boolean,%) -> NonNegativeInteger
        ++ count(p,u) returns the number of elements x in u
        ++ such that \axiom{p(x)} is true. For collections,
        ++ \axiom{count(p,u) = reduce(+,[1 for x in u | p(x)],0)}.
      parts: % -> List S
        ++ parts(u) returns a list of the consecutive elements of u.
        ++ For collections, \axiom{parts([x,y,...,z]) = (x,y,...,z)}.
      members: % -> List S
        ++ members(u) returns a list of the consecutive elements of u.
        ++ For collections, \axiom{parts([x,y,...,z]) = (x,y,...,z)}.
      if S has SetCategory then
        count: (S,%) -> NonNegativeInteger
          ++ count(x,u) returns the number of occurrences of x in u. For
          ++ collections, \axiom{count(x,u) = reduce(+,[x=y for y in u],0)}.
        member?: (S,%) -> Boolean
          ++ member?(x,u) tests if x is a member of u.
          ++ For collections,
          ++ \axiom{member?(x,u) = reduce(or,[x=y for y in u],false)}.
  add
   if S has Evalable S then
     eval(u:%,l:List Equation S):% == map(x +-> eval(x,l),u)
   if % has finiteAggregate then
     #c          == # parts c
     any?(f, c)  == _or/[f x for x in parts c]
     every?(f, c) == _and/[f x for x in parts c]
     count(f:S -> Boolean, c:%) == _+/[1 for x in parts c | f x]
```

```
    members x                  == parts x
    if S has SetCategory then
      count(s:S, x:%) == count(y +-> s = y, x)
      member?(e, c)   == any?(x +-> e = x,c)
      x = y ==
         size?(x, #y) and _and/[a = b for a in parts x for b in parts y]
      coerce(x:%):OutputForm ==
        bracket
           commaSeparate [a::OutputForm for a in parts x]$List(OutputForm)
```

———-

— HOAGG.dotabb —

```
"HOAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=HOAGG"];
"HOAGG" -> "AGG"
```

———-

— HOAGG.dotfull —

```
"HomogeneousAggregate(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=HOAGG"];
"HomogeneousAggregate(a:Type)" -> "Aggregate()"

"HomogeneousAggregate(Ring)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=HOAGG"];
"HomogeneousAggregate(Ring)"
  -> "HomogeneousAggregate(a:Type)"
```

———-

— HOAGG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "Aggregate()"
"HomogeneousAggregate(a:Type)" -> "Evalable(a:Type)"
"HomogeneousAggregate(a:Type)" -> "SetCategory()"

"Evalable(a:Type)" [color="#00EE00"];
```

```
"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Aggregate()" [color=lightblue];
"Aggregate()" -> "Type()"

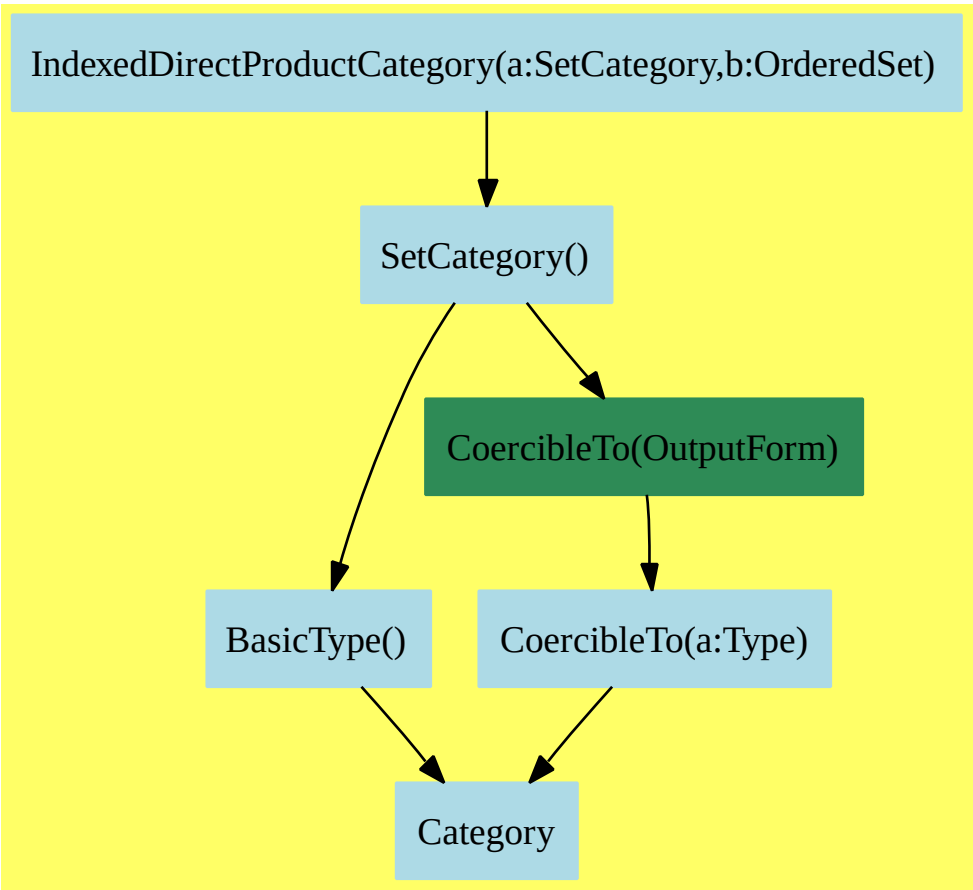
"Type()" [color=lightblue];
"Type()" -> "Category"

"Category" [color=lightblue];

}
```

———-

4.0.50 IndexedDirectProductCategory (IDPC)



— **IndexedDirectProductCategory.input** —

```
)set break resume
)sys rm -f IndexedDirectProductCategory.output
)spool IndexedDirectProductCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show IndexedDirectProductCategory
--R IndexedDirectProductCategory(A: SetCategory,S: OrderedSet)  is a category constructor
--R Abbreviation for IndexedDirectProductCategory is IDPC
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for IDPC
```

```
--R
--R------------------------------ Operations --------------------------------
--R ?=? : (%,%) -> Boolean                coerce : % -> OutputForm
--R hash : % -> SingleInteger             latex : % -> String
--R leadingCoefficient : % -> A           leadingSupport : % -> S
--R map : ((A -> A),%) -> %               monomial : (A,S) -> %
--R reductum : % -> %                     ?~=? : (%,%) -> Boolean
--R
--E 1

)spool
)lisp (bye)
```

———-

— **IndexedDirectProductCategory.help** —

```
=====================================================================
IndexedDirectProductCategory examples
=====================================================================

See Also:
o )show IndexedDirectProductCategory
```

———-

See:

⇐ “SetCategory” (SETCAT) 3.0.34 on page 113

Exports:

coerce	hash	latex	leadingCoefficient	leadingSupport
map	monomial	reductum	?=?	?~=?

These are directly exported but not implemented:

```
leadingCoefficient : % -> A
leadingSupport : % -> S
map : ((A -> A),%) -> %
monomial : (A,S) -> %
reductum : % -> %
```

These exports come from (p113) SetCategory():

```
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
```

```
 ?=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
```

— category IDPC IndexedDirectProductCategory —

```
)abbrev category IDPC IndexedDirectProductCategory
++ Author: James Davenport
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ This category represents the direct product of some set with
++ respect to an ordered indexing set.

IndexedDirectProductCategory(A:SetCategory,S:OrderedSet): Category ==
  SetCategory with
    map:           (A -> A, %) -> %
       ++ map(f,z) returns the new element created by applying the
       ++ function f to each component of the direct product element z.
    monomial:         (A, S) -> %
       ++ monomial(a,s) constructs a direct product element with the s
       ++ component set to \spad{a}
    leadingCoefficient:   % -> A
       ++ leadingCoefficient(z) returns the coefficient of the leading
       ++ (with respect to the ordering on the indexing set)
       ++ monomial of z.
       ++ Error: if z has no support.
    leadingSupport:   % -> S
       ++ leadingSupport(z) returns the index of leading
       ++ (with respect to the ordering on the indexing set) monomial of z.
       ++ Error: if z has no support.
    reductum:      % -> %
       ++ reductum(z) returns a new element created by removing the
       ++ leading coefficient/support pair from the element z.
       ++ Error: if z has no support.
```

———-

— IDPC.dotabb —

```
"IDPC"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=IDPC"];
```

```
"IDPC" -> "SETCAT"
```

———-

— IDPC.dotfull —

```
"IndexedDirectProductCategory(a:SetCategory,b:OrderedSet)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=IDPC"];
"IndexedDirectProductCategory(a:SetCategory,b:OrderedSet)" ->
   "SetCategory()"
```

———-

— IDPC.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"IndexedDirectProductCategory(a:SetCategory,b:OrderedSet)" [color=lightblue];
"IndexedDirectProductCategory(a:SetCategory,b:OrderedSet)" ->
   "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

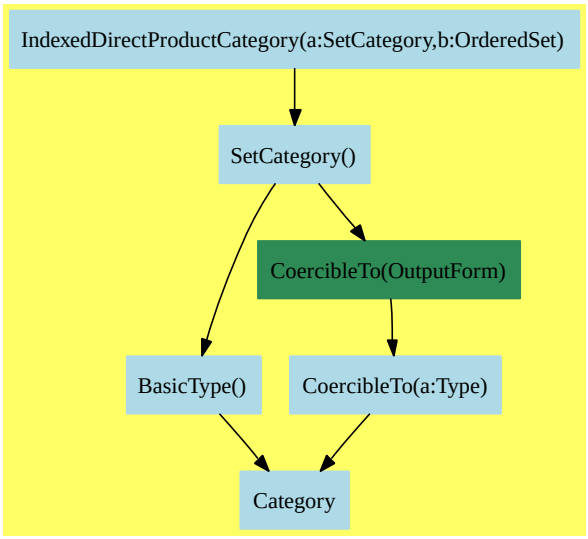
"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];

}
```

———-

4.0.51 LiouvillianFunctionCategory (LFCAT)



— LiouvillianFunctionCategory.input —

```
)set break resume
)sys rm -f LiouvillianFunctionCategory.output
)spool LiouvillianFunctionCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show LiouvillianFunctionCategory
--R LiouvillianFunctionCategory  is a category constructor
--R Abbreviation for LiouvillianFunctionCategory is LFCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for LFCAT
--R
--R------------------------------ Operations --------------------------------
--R ?**? : (%,%) -> %                      Ci : % -> %
--R Ei : % -> %                            Si : % -> %
--R acos : % -> %                          acosh : % -> %
--R acot : % -> %                          acoth : % -> %
--R acsc : % -> %                          acsch : % -> %
--R asec : % -> %                          asech : % -> %
--R asin : % -> %                          asinh : % -> %
--R atan : % -> %                          atanh : % -> %
--R cos : % -> %                           cosh : % -> %
--R cot : % -> %                           coth : % -> %
```

```
--R csc : % -> %                          csch : % -> %
--R dilog : % -> %                        erf : % -> %
--R exp : % -> %                          fresnelC : % -> %
--R fresnelS : % -> %                     integral : (%,Symbol) -> %
--R li : % -> %                           log : % -> %
--R pi : () -> %                          sec : % -> %
--R sech : % -> %                         sin : % -> %
--R sinh : % -> %                         tan : % -> %
--R tanh : % -> %
--R integral : (%,SegmentBinding %) -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **LiouvillianFunctionCategory.help** —

```
=====================================================================
LiouvillianFunctionCategory examples
=====================================================================

See Also:
o )show LiouvillianFunctionCategory
```

———-

See:

⇐ “PrimitiveFunctionCategory” (PRIMCAT) 2.0.16 on page 50
⇐ “TranscendentalFunctionCategory” (TRANFUN) 3.0.35 on page 117

Exports:

Ci	Ei	Si	acos	acosh
acot	acoth	acsc	acsch	asec
asech	asin	asinh	atan	atanh
cos	cosh	cot	coth	csc
csch	dilog	erf	exp	frsenelC
frsenelS	integral	li	log	pi
sec	sech	sin	sinh	tan
tanh	?**?			

These are directly exported but not implemented:

```
Ci : % -> %
```

```
dilog : % -> %
Ei : % -> %
erf : % -> %
li : % -> %
Si : % -> %
fresnelC : % -> %
fresnelS : % -> %
```

These exports come from (p50) PrimitiveFunctionCategory()

```
integral : (%,Symbol) -> %
integral : (%,SegmentBinding %) -> %
```

These exports come from (p117) TranscendentalFunctionCategory():

```
?**? : (%,%) -> %
acos : % -> %
acosh : % -> %
acot : % -> %
acoth : % -> %
acsc : % -> %
acsch : % -> %
asec : % -> %
asech : % -> %
asin : % -> %
asinh : % -> %
atan : % -> %
atanh : % -> %
cos : % -> %
cosh : % -> %
cot : % -> %
coth : % -> %
csc : % -> %
csch : % -> %
exp : % -> %
log : % -> %
pi : () -> %
sec : % -> %
sech : % -> %
sin : % -> %
sinh : % -> %
tan : % -> %
tanh : % -> %
```

— category LFCAT LiouvillianFunctionCategory —

```
)abbrev category LFCAT LiouvillianFunctionCategory
++ Category for the transcendental Liouvillian functions
```

```
++ Author: Manuel Bronstein
++ Date Created: ???
++ Date Last Updated: 14 May 1991
++ Description:
++ Category for the transcendental Liouvillian functions;

LiouvillianFunctionCategory(): Category ==
  Join(PrimitiveFunctionCategory, TranscendentalFunctionCategory) with
    Ei      : $  -> $
      ++ Ei(x) returns the exponential integral of x, i.e.
      ++ the integral of \spad{exp(x)/x dx}.
    Si      : $  -> $
      ++ Si(x) returns the sine integral of x, i.e.
      ++ the integral of \spad{sin(x) / x dx}.
    Ci      : $  -> $
      ++ Ci(x) returns the cosine integral of x, i.e.
      ++ the integral of \spad{cos(x) / x dx}.
    li      : $  -> $
      ++ li(x) returns the logarithmic integral of x, i.e.
      ++ the integral of \spad{dx / log(x)}.
    dilog   : $  -> $
      ++ dilog(x) returns the dilogarithm of x, i.e.
      ++ the integral of \spad{log(x) / (1 - x) dx}.
    erf     : $  -> $
      ++ erf(x) returns the error function of x, i.e.
      ++ \spad{2 / sqrt(%pi)} times the integral of \spad{exp(-x**2) dx}.
    fresnelS : $ -> $
      ++ fresnelS is the Fresnel integral S, defined by
      ++ S(x) = integrate(sin(t^2),t=0..x)
    fresnelC : $ -> $
      ++ fresnelC is the Fresnel integral C, defined by
      ++ C(x) = integrate(cos(t^2),t=0..x)
```

———-

— LFCAT.dotabb —

```
"LFCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=LFCAT"];
"LFCAT" -> "PRIMCAT"
"LFCAT" -> "TRANFUN"
```

———-

— LFCAT.dotfull —

```
"LiouvillianFunctionCategory()"
```

```
 [color=lightblue,href="bookvol10.2.pdf#nameddest=LFCAT"];
"LiouvillianFunctionCategory()" -> "PrimitiveFunctionCategory()"
"LiouvillianFunctionCategory()" -> "TranscendentalFunctionCategory()"
```

———-

— LFCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"LiouvillianFunctionCategory()" [color=lightblue];
"LiouvillianFunctionCategory()" -> "PrimitiveFunctionCategory()"
"LiouvillianFunctionCategory()" -> "TranscendentalFunctionCategory()"

"PrimitiveFunctionCategory()" [color=lightblue];
"PrimitiveFunctionCategory()" -> "Category"

"TranscendentalFunctionCategory()" [color=lightblue];
"TranscendentalFunctionCategory()" ->
    "TRIGCAT"
"TranscendentalFunctionCategory()" ->
    "ATRIG"
"TranscendentalFunctionCategory()" ->
    "HYPCAT"
"TranscendentalFunctionCategory()" ->
    "AHYP"
"TranscendentalFunctionCategory()" ->
    "ELEMFUN"

"TRIGCAT" [color=lightblue];
"TRIGCAT" -> "Category"

"ATRIG" [color=lightblue];
"ATRIG" -> "Category"

"HYPCAT" [color=lightblue];
"HYPCAT" -> "Category"

"AHYP" [color=lightblue];
"AHYP" -> "Category"

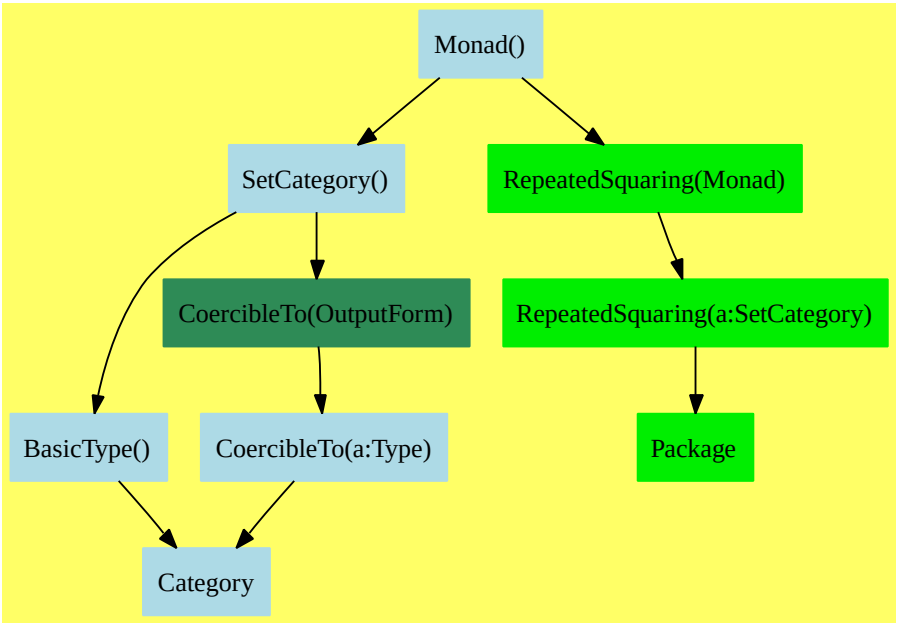
"ELEMFUN" [color=lightblue];
"ELEMFUN" -> "Category"

"Category" [color=lightblue];
```

```
}
```

———-

4.0.52 Monad (MONAD)



— **Monad.input** —

```
)set break resume
)sys rm -f Monad.output
)spool Monad.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show Monad
--R Monad  is a category constructor
--R Abbreviation for Monad is MONAD
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for MONAD
--R
```

```
--R------------------------------ Operations ------------------------------
--R ?*? : (%,%) -> %                       ?**? : (%,PositiveInteger) -> %
--R ?=? : (%,%) -> Boolean                 coerce : % -> OutputForm
--R hash : % -> SingleInteger              latex : % -> String
--R ?~=? : (%,%) -> Boolean
--R leftPower : (%,PositiveInteger) -> %
--R rightPower : (%,PositiveInteger) -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **Monad.help** —

```
========================================================================
Monad examples
========================================================================

See Also:
o )show Monad
```

———-

See:

⇒ “MonadWithUnit” (MONADWU) 5.0.75 on page 325
⇒ “NonAssociativeRng” (NARNG) 8.0.116 on page 679

Exports:

coerce	hash	latex	leftPower	rightPower
?**?	?*?	?=?	?~=?	

These are directly exported but not implemented:

```
?*? : (%,%) -> %
```

These are implemented by this category:

```
leftPower : (%,PositiveInteger) -> %
rightPower : (%,PositiveInteger) -> %
?**? : (%,PositiveInteger) -> %
```

These exports come from (p113) SetCategory():

```
 coerce : % -> OutputForm
 hash : % -> SingleInteger
 latex : % -> String
 ?=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
```

— category MONAD Monad —

```
)abbrev category MONAD Monad
++ Authors: J. Grabmeier, R. Wisbauer
++ Date Created: 01 March 1991
++ Date Last Updated: 11 June 1991
++ Basic Operations: *, **
++ Related Constructors: SemiGroup, Monoid, MonadWithUnit
++ Also See:
++ AMS Classifications:
++ Keywords: Monad,  binary operation
++ Reference:
++  N. Jacobson: Structure and Representations of Jordan Algebras
++  AMS, Providence, 1968
++ Description:
++ Monad is the class of all multiplicative monads, i.e. sets
++ with a binary operation.

Monad(): Category == SetCategory with
      "*": (%,%) -> %
        ++ a*b is the product of \spad{a} and b in a set with
        ++ a binary operation.
      rightPower: (%,PositiveInteger) -> %
        ++ rightPower(a,n) returns the \spad{n}-th right power of \spad{a},
        ++ i.e. \spad{rightPower(a,n) := rightPower(a,n-1) * a} and
        ++ \spad{rightPower(a,1) := a}.
      leftPower: (%,PositiveInteger) -> %
        ++ leftPower(a,n) returns the \spad{n}-th left power of \spad{a},
        ++ i.e. \spad{leftPower(a,n) := a * leftPower(a,n-1)} and
        ++ \spad{leftPower(a,1) := a}.
      "**": (%,PositiveInteger) -> %
        ++ a**n returns the \spad{n}-th power of \spad{a},
        ++ defined by repeated squaring.
    add
      import RepeatedSquaring(%)
      x:% ** n:PositiveInteger == expt(x,n)
      rightPower(a,n) ==
--        one? n => a
        (n = 1) => a
        res := a
        for i in 1..(n-1) repeat res := res * a
        res
      leftPower(a,n) ==
```

```
--        one? n => a
        (n = 1) => a
        res := a
        for i in 1..(n-1) repeat res := a * res
        res
```

———-

— **MONAD.dotabb** —

```
"MONAD"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=MONAD"];
"MONAD" -> "SETCAT"
```

———-

— **MONAD.dotfull** —

```
"Monad()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=MONAD"];
"Monad()" -> "SetCategory()"
"Monad()" -> "RepeatedSquaring(Monad)"
```

———-

— **MONAD.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"Monad()" [color=lightblue];
"Monad()" -> "SetCategory()"
"Monad()" -> "RepeatedSquaring(Monad)"

"RepeatedSquaring(Monad)" [color="#00EE00"];
"RepeatedSquaring(Monad)" -> "RepeatedSquaring(a:SetCategory)"

"RepeatedSquaring(a:SetCategory)" [color="#00EE00"];
"RepeatedSquaring(a:SetCategory)" -> "Package"

"Package" [color="#00EE00"];

"SetCategory()" [color=lightblue];
```

```
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

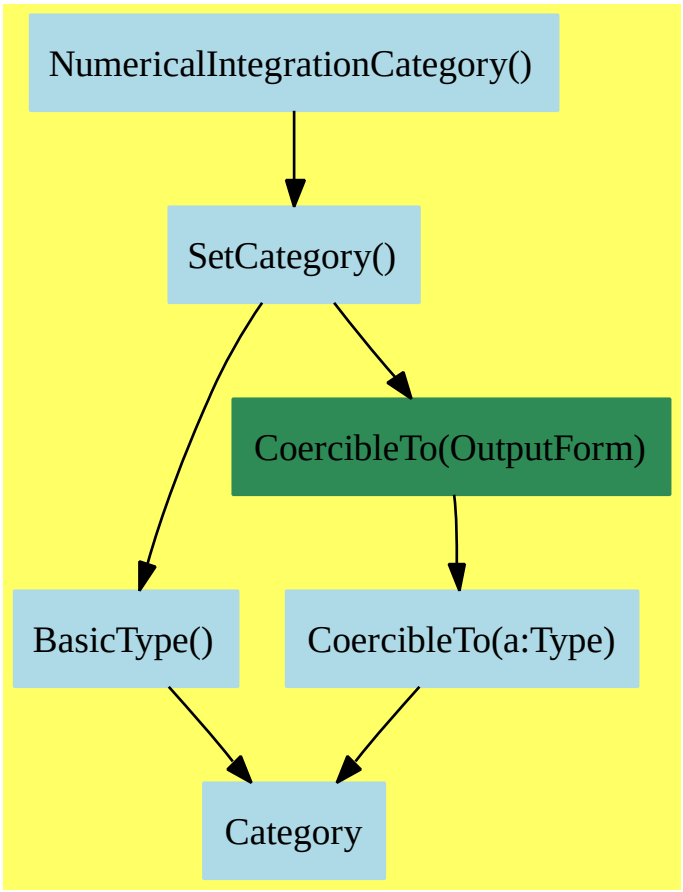
"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];

}
```

———————-

4.0.53 NumericalIntegrationCategory (NUMINT)



— **NumericalIntegrationCategory.input** —

```
)set break resume
)sys rm -f NumericalIntegrationCategory.output
)spool NumericalIntegrationCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show NumericalIntegrationCategory
--R NumericalIntegrationCategory  is a category constructor
--R Abbreviation for NumericalIntegrationCategory is NUMINT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for NUMINT
```

```
--R
--R------------------------------ Operations ------------------------------
--R ?=? : (%,%) -> Boolean                coerce : % -> OutputForm
--R hash : % -> SingleInteger             latex : % -> String
--R ?~=? : (%,%) -> Boolean
--R measure : (RoutinesTable,Record(fn: Expression DoubleFloat,range: List Segment OrderedCompletion Double
--R measure : (RoutinesTable,Record(var: Symbol,fn: Expression DoubleFloat,range: Segment OrderedCompletion
--R numericalIntegration : (Record(fn: Expression DoubleFloat,range: List Segment OrderedCompletion DoubleF
--R numericalIntegration : (Record(var: Symbol,fn: Expression DoubleFloat,range: Segment OrderedCompletion
--R
--E 1

)spool
)lisp (bye)
```

———-

— **NumericalIntegrationCategory.help** —

```
====================================================================
NumericalIntegrationCategory examples
====================================================================

See Also:
o )show NumericalIntegrationCategory
```

———-

See:

⇐ “SetCategory” (SETCAT) 3.0.34 on page 113

Exports:

coerce	hash	latex	measure
numericalIntegration	?=?	?~=?	

These are directly exported but not implemented:

```
measure :
 (RoutinesTable,
  Record(fn: Expression DoubleFloat,
         range: List Segment OrderedCompletion DoubleFloat,
         abserr: DoubleFloat,
         relerr: DoubleFloat)) ->
    Record(measure: Float,explanations: String,extra: Result)
measure :
 (RoutinesTable,
  Record(var: Symbol,
```

```
         fn: Expression DoubleFloat,
         range: Segment OrderedCompletion DoubleFloat,
         abserr: DoubleFloat,
         relerr: DoubleFloat)) ->
    Record(measure: Float,explanations: String,extra: Result)
 numericalIntegration :
  (Record(fn: Expression DoubleFloat,
         range: List Segment OrderedCompletion DoubleFloat,
         abserr: DoubleFloat,relerr: DoubleFloat),
   Result) ->
  Result
 numericalIntegration :
  (Record(var: Symbol,
         fn: Expression DoubleFloat,
         range: Segment OrderedCompletion DoubleFloat,
         abserr: DoubleFloat,
         relerr: DoubleFloat),
   Result) ->
  Result
```

These exports come from (p113) SetCategory():

```
 coerce : % -> OutputForm
 hash : % -> SingleInteger
 latex : % -> String
 ?=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
```

— category NUMINT NumericalIntegrationCategory —

```
)abbrev category NUMINT NumericalIntegrationCategory
++ Author: Brian Dupee
++ Date Created: February 1994
++ Date Last Updated: March 1996
++ Description:
++ \axiomType{NumericalIntegrationCategory} is the \axiom{category} for
++ describing the set of Numerical Integration \axiom{domains} with
++ \axiomFun{measure} and \axiomFun{numericalIntegration}.

NumericalIntegrationCategory(): Category == Exports where

 EDFE   ==> Expression DoubleFloat
 SOCDFE ==> Segment OrderedCompletion DoubleFloat
 DFE    ==> DoubleFloat
 NIAE   ==> Record(var:Symbol,fn:EDFE,range:SOCDFE,abserr:DFE,relerr:DFE)
 MDNIAE ==> Record(fn:EDFE,range:List SOCDFE,abserr:DFE,relerr:DFE)

 Exports ==> SetCategory with
```

```
  measure:(RoutinesTable,NIAE) -> _
   Record(measure:Float,explanations:String,extra:Result)
    ++ measure(R,args) calculates an estimate of the ability of a particular
    ++ method to solve a problem.
    ++
    ++ This method may be either a specific NAG routine or a strategy (such
    ++ as transforming the function from one which is difficult to one which
    ++ is easier to solve).
    ++
    ++ It will call whichever agents are needed to perform analysis on the
    ++ problem in order to calculate the measure. There is a parameter,
    ++ labelled \axiom{sofar}, which would contain the best compatibility
    ++ found so far.

  numericalIntegration: (NIAE, Result) -> Result
    ++ numericalIntegration(args,hints) performs the integration of the
    ++ function given the strategy or method returned by \axiomFun{measure}.

  measure:(RoutinesTable,MDNIAE) -> _
   Record(measure:Float,explanations:String,extra:Result)
    ++ measure(R,args) calculates an estimate of the ability of a particular
    ++ method to solve a problem.
    ++
    ++ This method may be either a specific NAG routine or a strategy (such
    ++ as transforming the function from one which is difficult to one which
    ++ is easier to solve).
    ++
    ++ It will call whichever agents are needed to perform analysis on the
    ++ problem in order to calculate the measure. There is a parameter,
    ++ labelled \axiom{sofar}, which would contain the best compatibility
    ++ found so far.

  numericalIntegration: (MDNIAE, Result) -> Result
    ++ numericalIntegration(args,hints) performs the integration of the
    ++ function given the strategy or method returned by \axiomFun{measure}.
```

———-

— NUMINT.dotabb —

```
"NUMINT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=NUMINT"];
"NUMINT" -> "SETCAT"
```

———-

— NUMINT.dotfull —

```
"NumericalIntegrationCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=NUMINT"];
"NumericalIntegrationCategory()" -> "SetCategory()"
```

———-

— NUMINT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"NumericalIntegrationCategory()" [color=lightblue];
"NumericalIntegrationCategory()" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

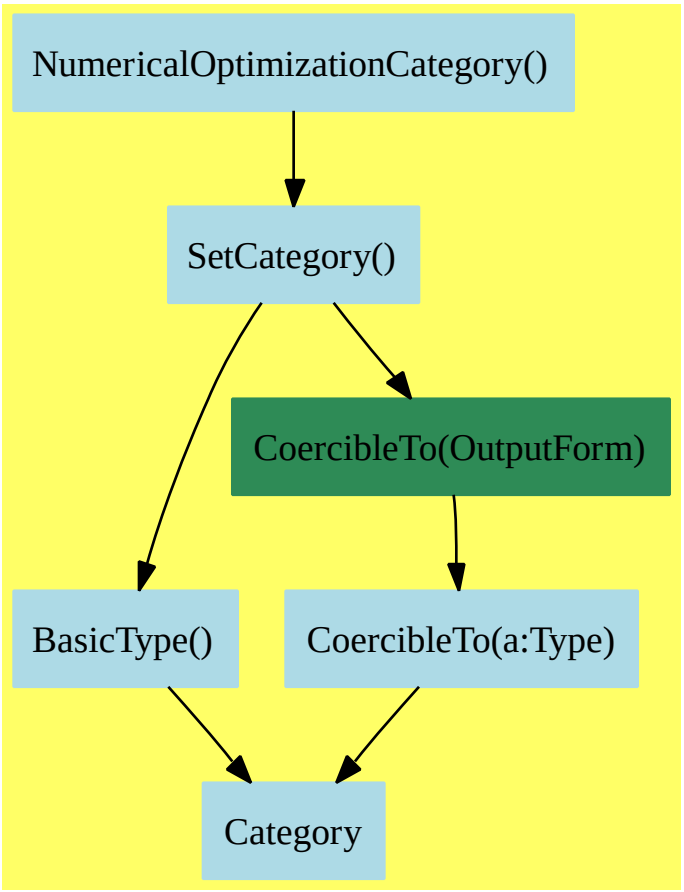
"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];

}
```

———-

4.0.54 NumericalOptimizationCategory (OPTCAT)



— NumericalOptimizationCategory.input —

```
)set break resume
)sys rm -f NumericalOptimizationCategory.output
)spool NumericalOptimizationCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show NumericalOptimizationCategory
--R NumericalOptimizationCategory  is a category constructor
--R Abbreviation for NumericalOptimizationCategory is OPTCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for OPTCAT
```

```
--R
--R------------------------------ Operations ------------------------------
--R ?=? : (%,%) -> Boolean                coerce : % -> OutputForm
--R hash : % -> SingleInteger             latex : % -> String
--R ?~=? : (%,%) -> Boolean
--R measure : (RoutinesTable,Record(lfn: List Expression DoubleFloat,init: List DoubleFloat)) ->
--R measure : (RoutinesTable,Record(fn: Expression DoubleFloat,init: List DoubleFloat,lb: List O
--R numericalOptimization : Record(fn: Expression DoubleFloat,init: List DoubleFloat,lb: List Or
--R numericalOptimization : Record(lfn: List Expression DoubleFloat,init: List DoubleFloat) -> R
--R
--E 1

)spool
)lisp (bye)
```

———-

— NumericalOptimizationCategory.help —

```
=====================================================================
NumericalOptimizationCategory examples
=====================================================================

See Also:
o )show NumericalOptimizationCategory
```

———-

See:

⇐ “SetCategory” (SETCAT) 3.0.34 on page 113

Exports:

coerce	hash	latex	measure
numericalOptimization	?=?	?~=?	

These are directly exported but not implemented:

```
measure :
 (RoutinesTable,
  Record(lfn: List Expression DoubleFloat,
         init: List DoubleFloat)) ->
   Record(measure: Float,explanations: String)
measure :
 (RoutinesTable,
  Record(fn: Expression DoubleFloat,
         init: List DoubleFloat,
         lb: List OrderedCompletion DoubleFloat,
```

```
          cf: List Expression DoubleFloat,
          ub: List OrderedCompletion DoubleFloat)) ->
    Record(measure: Float,explanations: String)
 numericalOptimization :
  Record(fn: Expression DoubleFloat,
         init: List DoubleFloat,
         lb: List OrderedCompletion DoubleFloat,
         cf: List Expression DoubleFloat,
         ub: List OrderedCompletion DoubleFloat) ->
   Result
 numericalOptimization :
  Record(lfn: List Expression DoubleFloat,
         init: List DoubleFloat) ->
   Result
```

These exports come from (p113) SetCategory():

```
 coerce : % -> OutputForm
 hash : % -> SingleInteger
 latex : % -> String
 ?=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
```

— category OPTCAT NumericalOptimizationCategory —

```
)abbrev category OPTCAT NumericalOptimizationCategory
++ Author: Brian Dupee
++ Date Created: January 1996
++ Date Last Updated: March 1996
++ Description:
++ \axiomType{NumericalOptimizationCategory} is the \axiom{category} for
++ describing the set of Numerical Optimization \axiom{domains} with
++ \axiomFun{measure} and \axiomFun{optimize}.

NumericalOptimizationCategory(): Category == Exports where

 LDFH   ==> List DoubleFloat
 LEDFH  ==> List Expression DoubleFloat
 LSAH   ==> Record(lfn:LEDFH, init:LDFH)
 EDFH   ==> Expression DoubleFloat
 LOCDFH ==> List OrderedCompletion DoubleFloat
 NOAH   ==> Record(fn:EDFH, init:LDFH, lb:LOCDFH, cf:LEDFH, ub:LOCDFH)

 Exports ==> SetCategory with

  measure:(RoutinesTable,NOAH)->Record(measure:Float,explanations:String)
    ++ measure(R,args) calculates an estimate of the ability of a particular
    ++ method to solve an optimization problem.
    ++
```

```
    ++ This method may be either a specific NAG routine or a strategy (such
    ++ as transforming the function from one which is difficult to one which
    ++ is easier to solve).
    ++
    ++ It will call whichever agents are needed to perform analysis on the
    ++ problem in order to calculate the measure. There is a parameter,
    ++ labelled \axiom{sofar}, which would contain the best compatibility
    ++ found so far.

  measure:(RoutinesTable,LSAH)->Record(measure:Float,explanations:String)
    ++ measure(R,args) calculates an estimate of the ability of a particular
    ++ method to solve an optimization problem.
    ++
    ++ This method may be either a specific NAG routine or a strategy (such
    ++ as transforming the function from one which is difficult to one which
    ++ is easier to solve).
    ++
    ++ It will call whichever agents are needed to perform analysis on the
    ++ problem in order to calculate the measure. There is a parameter,
    ++ labelled \axiom{sofar}, which would contain the best compatibility
    ++ found so far.

  numericalOptimization:LSAH -> Result
    ++ numericalOptimization(args) performs the optimization of the
    ++ function given the strategy or method returned by \axiomFun{measure}.

  numericalOptimization:NOAH -> Result
    ++ numericalOptimization(args) performs the optimization of the
    ++ function given the strategy or method returned by \axiomFun{measure}.
```

— OPTCAT.dotabb —

```
"OPTCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OPTCAT"];
"OPTCAT" -> "SETCAT"
```

— OPTCAT.dotfull —

```
"NumericalOptimizationCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OPTCAT"];
"NumericalOptimizationCategory()" -> "SetCategory()"
```

———-

— OPTCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"NumericalOptimizationCategory()" [color=lightblue];
"NumericalOptimizationCategory()" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

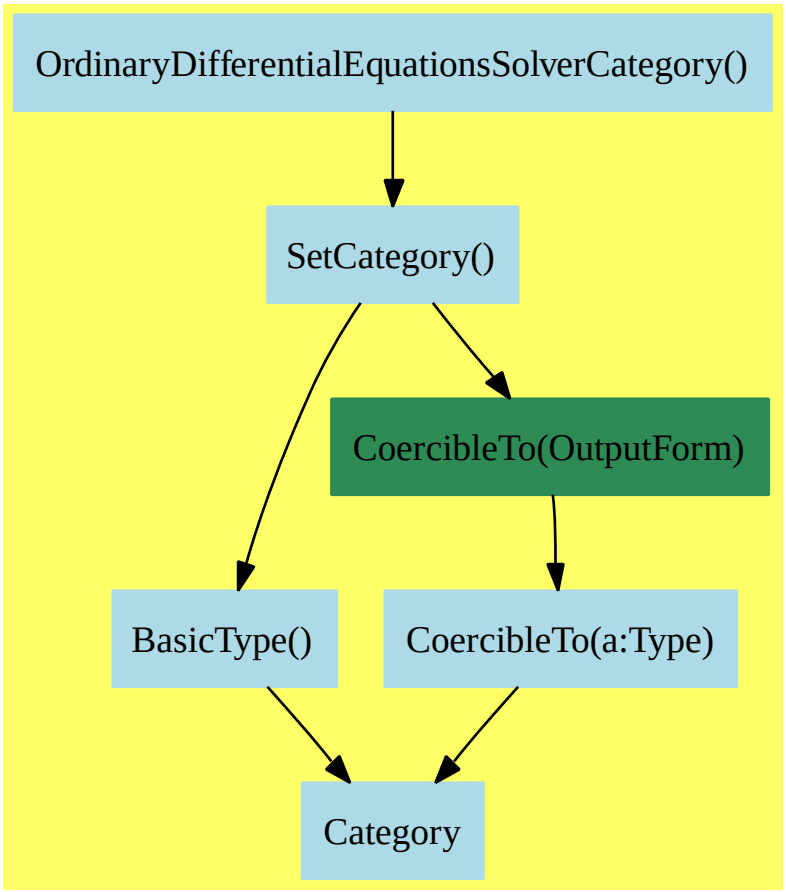
"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];

}
```

———-

4.0.55 OrdinaryDifferentialEquationsSolverCategory (ODECAT)



— **OrdinaryDifferentialEquationsSolverCategory.input** —

```
)set break resume
)sys rm -f OrdinaryDifferentialEquationsSolverCategory.output
)spool OrdinaryDifferentialEquationsSolverCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show OrdinaryDifferentialEquationsSolverCategory
--R OrdinaryDifferentialEquationsSolverCategory  is a category constructor
--R Abbreviation for OrdinaryDifferentialEquationsSolverCategory is ODECAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for ODECAT
```

```
--R
--R------------------------------ Operations ------------------------------
--R ?=? : (%,%) -> Boolean                coerce : % -> OutputForm
--R hash : % -> SingleInteger             latex : % -> String
--R ?~=? : (%,%) -> Boolean
--R ODESolve : Record(xinit: DoubleFloat,xend: DoubleFloat,fn: Vector Expression DoubleFloat,yinit: List Dou
--R measure : (RoutinesTable,Record(xinit: DoubleFloat,xend: DoubleFloat,fn: Vector Expression DoubleFloat,y
--R
--E 1

)spool
)lisp (bye)
```

———-

— OrdinaryDifferentialEquationsSolverCategory.help —

```
=====================================================================
OrdinaryDifferentialEquationsSolverCategory examples
=====================================================================

See Also:
o )show OrdinaryDifferentialEquationsSolverCategory
```

———-

See:

⇐ “SetCategory” (SETCAT) 3.0.34 on page 113

Exports:

coerce	hash	latex	measure	ODESolve
?=?	?~=?			

These are directly exported but not implemented:

```
ODESolve :
 Record(xinit: DoubleFloat,
        xend: DoubleFloat,
        fn: Vector Expression DoubleFloat,
        yinit: List DoubleFloat,
        intvals: List DoubleFloat,
        g: Expression DoubleFloat,
        abserr: DoubleFloat,
        relerr: DoubleFloat) ->
  Result
measure :
 (RoutinesTable,
```

```
   Record(xinit: DoubleFloat,
          xend: DoubleFloat,
          fn: Vector Expression DoubleFloat,
          yinit: List DoubleFloat,
          intvals: List DoubleFloat,
          g: Expression DoubleFloat,
          abserr: DoubleFloat,
          relerr: DoubleFloat)) ->
    Record(measure: Float,explanations: String)
```

These exports come from (p113) SetCategory():

```
 coerce : % -> OutputForm
 hash : % -> SingleInteger
 latex : % -> String
 ?=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
```

— **category ODECAT OrdinaryDifferentialEquationsSolverCategory** —

```
)abbrev category ODECAT OrdinaryDifferentialEquationsSolverCategory
++ Author: Brian Dupee
++ Date Created: February 1995
++ Date Last Updated: June 1995
++ Basic Operations:
++ Description:
++ \axiomType{OrdinaryDifferentialEquationsSolverCategory} is the
++ \axiom{category} for describing the set of ODE solver \axiom{domains}
++ with \axiomFun{measure} and \axiomFun{ODEsolve}.

OrdinaryDifferentialEquationsSolverCategory(): Category == Exports where

 DFF   ==> DoubleFloat
 VEDFF ==> Vector Expression DoubleFloat
 LDFF  ==> List DoubleFloat
 EDFF  ==> Expression DoubleFloat
 ODEAF ==> Record(xinit:DFF,xend:DFF,fn:VEDFF,yinit:LDFF,intvals:LDFF,g:EDFF,abserr:DFF,relerr:D

 Exports ==> SetCategory with

  measure:(RoutinesTable,ODEAF) -> Record(measure:Float,explanations:String)
    ++ measure(R,args) calculates an estimate of the ability of a particular
    ++ method to solve a problem.
    ++
    ++ This method may be either a specific NAG routine or a strategy (such
    ++ as transforming the function from one which is difficult to one which
    ++ is easier to solve).
```

```
    ++
    ++ It will call whichever agents are needed to perform analysis on the
    ++ problem in order to calculate the measure. There is a parameter,
    ++ labelled \axiom{sofar}, which would contain the best compatibility
    ++ found so far.

  ODESolve: ODEAF -> Result
    ++ ODESolve(args) performs the integration of the
    ++ function given the strategy or method returned by \axiomFun{measure}.
```

———-

— ODECAT.dotabb —

```
"ODECAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ODECAT"];
"ODECAT" -> "SETCAT"
```

———-

— ODECAT.dotfull —

```
"OrdinaryDifferentialEquationsSolverCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ODECAT"];
"OrdinaryDifferentialEquationsSolverCategory()" -> "SetCategory()"
```

———-

— ODECAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"OrdinaryDifferentialEquationsSolverCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ODECAT"];
"OrdinaryDifferentialEquationsSolverCategory()" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"
```

```
"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

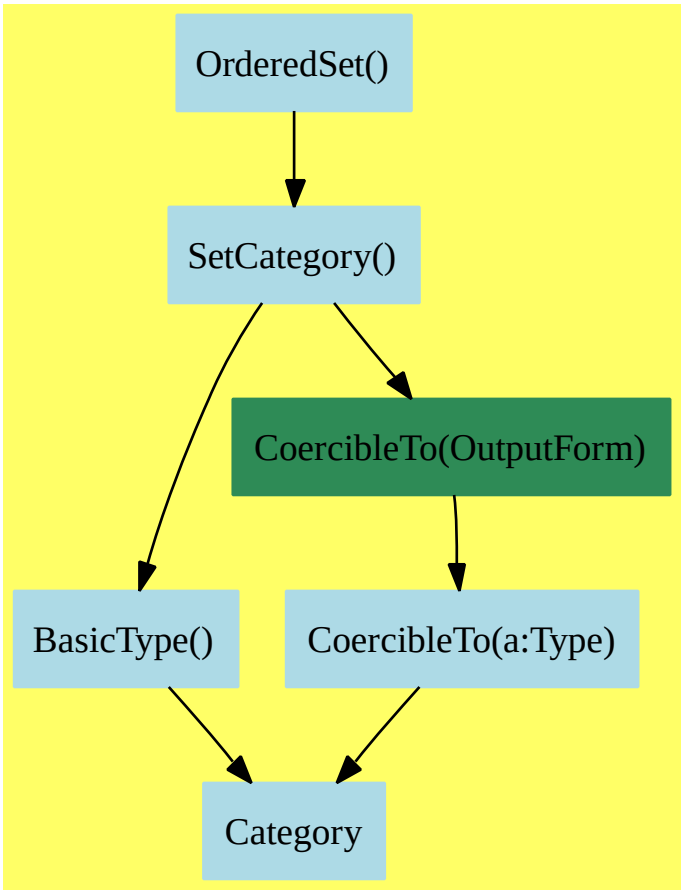
"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];

}
```

———-

4.0.56 OrderedSet (ORDSET)



— **OrderedSet.input** —

```
)set break resume
)sys rm -f OrderedSet.output
)spool OrderedSet.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show OrderedSet
--R OrderedSet  is a category constructor
--R Abbreviation for OrderedSet is ORDSET
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for ORDSET
--R
--R------------------------------ Operations --------------------------------
--R ?<? : (%,%) -> Boolean               ?<=? : (%,%) -> Boolean
--R ?=? : (%,%) -> Boolean               ?>? : (%,%) -> Boolean
--R ?>=? : (%,%) -> Boolean              coerce : % -> OutputForm
--R hash : % -> SingleInteger            latex : % -> String
--R max : (%,%) -> %                     min : (%,%) -> %
--R ?~=? : (%,%) -> Boolean
--R
--E 1

)spool
)lisp (bye)
```

———-

— **OrderedSet.help** —

```
=====================================================================
OrderedSet examples
=====================================================================

See Also:
o )show OrderedSet
```

———-

See:

⇒ “BitAggregate” (BTAGG) 9.0.123 on page 731
⇒ “CachableSet” (CACHSET) 5.0.69 on page 283
⇒ “DifferentialVariableCategory” (DVARCAT) 5.0.71 on page 294

⇒ "ExpressionSpace" (ES) 5.0.72 on page 300
⇒ "FortranMachineTypeCategory" (FMTC) 13.0.171 on page 1108
⇒ "IntervalCategory" (INTCAT) 14.0.176 on page 1150
⇒ "OrderedAbelianSemiGroup" (OASGP) 6.0.89 on page 456
⇒ "OrderedFinite" (ORDFIN) 5.0.77 on page 336
⇒ "OrderedMonoid" (ORDMON) 6.0.90 on page 461
⇒ "PolynomialCategory" (POLYCAT) 16.0.189 on page 1255
⇒ "QuotientFieldCategory" (QFCAT) 17.0.198 on page 1376
⇐ "SetCategory" (SETCAT) 3.0.34 on page 113

Exports:

coerce	hash	latex	max	min
?<?	?<=?	?=?	?>?	?>=?
?~=?				

These are directly exported but not implemented:

```
?<? : (%,%) -> Boolean
```

These are implemented by this category:

```
max : (%,%) -> %
min : (%,%) -> %
?>? : (%,%) -> Boolean
?>=? : (%,%) -> Boolean
?<=? : (%,%) -> Boolean
```

These exports come from (p113) SetCategory():

```
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
```

— category ORDSET OrderedSet —

```
)abbrev category ORDSET OrderedSet
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
```

```
++ Description:
++ The class of totally ordered sets, that is, sets such that for each
++ pair of elements \spad{(a,b)}
++ exactly one of the following relations holds \spad{a<b or a=b or b<a}
++ and the relation is transitive, i.e.  \spad{a<b and b<c => a<c}.

OrderedSet(): Category == SetCategory with
    "<": (%,%) -> Boolean
      ++ x < y is a strict total ordering on the elements of the set.
    ">":         (%, %) -> Boolean
      ++ x > y is a greater than test.
    ">=":        (%, %) -> Boolean
      ++ x >= y is a greater than or equal test.
    "<=":        (%, %) -> Boolean
      ++ x <= y is a less than or equal test.
    max: (%,%) -> %
      ++ max(x,y) returns the maximum of x and y relative to "<".
    min: (%,%) -> %
      ++ min(x,y) returns the minimum of x and y relative to "<".
  add
    x,y: %
  -- These really ought to become some sort of macro
    max(x,y) ==
      x > y => x
      y
    min(x,y) ==
      x > y => y
      x
    ((x: %) >  (y: %)) : Boolean == y < x
    ((x: %) >= (y: %)) : Boolean == not (x < y)
    ((x: %) <= (y: %)) : Boolean == not (y < x)
```

———-

— **ORDSET.dotabb** —

```
"ORDSET"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ORDSET"];
"ORDSET" -> "SETCAT"
```

———-

— **ORDSET.dotfull** —

```
"OrderedSet()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ORDSET"];
```

```
"OrderedSet()" -> "SetCategory()"
```

———-

— ORDSET.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"OrderedSet()" [color=lightblue];
"OrderedSet()" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

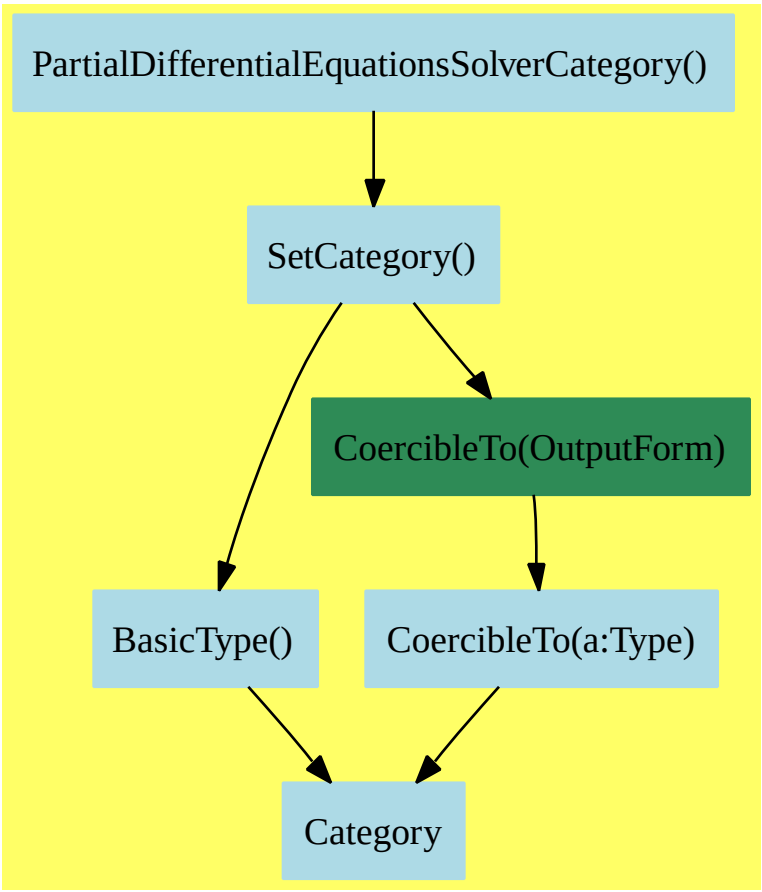
"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];
}
```

———-

4.0.57 PartialDifferentialEquationsSolverCategory (PDECAT)



— **PartialDifferentialEquationsSolverCategory.input** —

```
)set break resume
)sys rm -f PartialDifferentialEquationsSolverCategory.output
)spool PartialDifferentialEquationsSolverCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show PartialDifferentialEquationsSolverCategory
--R PartialDifferentialEquationsSolverCategory  is a category constructor
--R Abbreviation for PartialDifferentialEquationsSolverCategory is PDECAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PDECAT
```

```
--R
--R------------------------------ Operations ------------------------------
--R ?=? : (%,%) -> Boolean                coerce : % -> OutputForm
--R hash : % -> SingleInteger             latex : % -> String
--R ?~=? : (%,%) -> Boolean
--R PDESolve : Record(pde: List Expression DoubleFloat,constraints: List Record(start: DoubleFlo
--R measure : (RoutinesTable,Record(pde: List Expression DoubleFloat,constraints: List Record(st
--R
--E 1

)spool
)lisp (bye)
```

———-

— PartialDifferentialEquationsSolverCategory.help —

```
====================================================================
PartialDifferentialEquationsSolverCategory examples
====================================================================

See Also:
o )show PartialDifferentialEquationsSolverCategory
```

———-

See:

⇐ “SetCategory” (SETCAT) 3.0.34 on page 113

Exports:

coerce	hash	latex	measure	PDESolve
?=?	?~=?			

These are directly exported but not implemented:

```
measure :
(RoutinesTable,
 Record(pde: List Expression DoubleFloat,
        constraints:
         List Record(start: DoubleFloat,
                     finish: DoubleFloat,
                     grid: NonNegativeInteger,
                     boundaryType: Integer,
                     dStart: Matrix DoubleFloat,
                     dFinish: Matrix DoubleFloat),
        f: List List Expression DoubleFloat,
        st: String,
```

```
        tol: DoubleFloat)) ->
   Record(measure: Float,explanations: String)
 PDESolve :
  Record(pde: List Expression DoubleFloat,
         constraints:
          List Record(start: DoubleFloat,
                      finish: DoubleFloat,
                      grid: NonNegativeInteger,
                      boundaryType: Integer,
                      dStart: Matrix DoubleFloat,
                      dFinish: Matrix DoubleFloat),
         f: List List Expression DoubleFloat,
         st: String,
         tol: DoubleFloat) ->
   Result
```

These exports come from (p533) Dictionary(S:SetCategory):

```
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
```

— category PDECAT PartialDifferentialEquationsSolverCategory —

```
)abbrev category PDECAT PartialDifferentialEquationsSolverCategory
++ Author: Brian Dupee
++ Date Created: February 1995
++ Date Last Updated: June 1995
++ Basic Operations:
++ Description:
++ \axiomType{PartialDifferentialEquationsSolverCategory} is the
++ \axiom{category} for describing the set of PDE solver \axiom{domains}
++ with \axiomFun{measure} and \axiomFun{PDEsolve}.

-- PDEA        ==> Record(xmin:F,xmax:F,ymin:F,ymax:F,ngx:NNI,ngy:NNI,_
--             pde:List Expression Float, bounds:List List Expression Float,_
--             st:String, tol:DF)

--  measure:(RoutinesTable,PDEA) -> Record(measure:F,explanations:String)
--    ++ measure(R,args) calculates an estimate of the ability of a particular
--    ++ method to solve a problem.
--    ++
--    ++ This method may be either a specific NAG routine or a strategy (such
--    ++ as transforming the function from one which is difficult to one which
--    ++ is easier to solve).
--    ++
--    ++ It will call whichever agents are needed to perform analysis on the
```

```
--    ++ problem in order to calculate the measure. There is a parameter,
--    ++ labelled \axiom{sofar}, which would contain the best compatibility
--    ++ found so far.

--  PDESolve: PDEA -> Result
--    ++ PDESolve(args) performs the integration of the
--    ++ function given the strategy or method returned by \axiomFun{measure}.

PartialDifferentialEquationsSolverCategory(): Category == Exports where

 DFG   ==> DoubleFloat
 NNIG  ==> NonNegativeInteger
 INTG  ==> Integer
 MDFG  ==> Matrix DoubleFloat
 PDECG ==> Record(start:DFG, finish:DFG, grid:NNIG, boundaryType:INTG,
                  dStart:MDFG, dFinish:MDFG)
 LEDFG ==> List Expression DoubleFloat
 PDEBG ==> Record(pde:LEDFG, constraints:List PDECG, f:List LEDFG,
                    st:String, tol:DFG)
 Exports ==> SetCategory with

  measure:(RoutinesTable,PDEBG) -> Record(measure:Float,explanations:String)
    ++ measure(R,args) calculates an estimate of the ability of a particular
    ++ method to solve a problem.
    ++
    ++ This method may be either a specific NAG routine or a strategy (such
    ++ as transforming the function from one which is difficult to one which
    ++ is easier to solve).
    ++
    ++ It will call whichever agents are needed to perform analysis on the
    ++ problem in order to calculate the measure. There is a parameter,
    ++ labelled \axiom{sofar}, which would contain the best compatibility
    ++ found so far.

  PDESolve: PDEBG -> Result
    ++ PDESolve(args) performs the integration of the
    ++ function given the strategy or method returned by \axiomFun{measure}.
```

— PDECAT.dotabb —

```
"PDECAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PDECAT"];
"PDECAT" -> "SETCAT"
```

— **PDECAT.dotfull** —

```
"PartialDifferentialEquationsSolverCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PDECAT"];
"PartialDifferentialEquationsSolverCategory()" -> "SetCategory()"
```

———-

— **PDECAT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"PartialDifferentialEquationsSolverCategory()" [color=lightblue];
"PartialDifferentialEquationsSolverCategory()" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

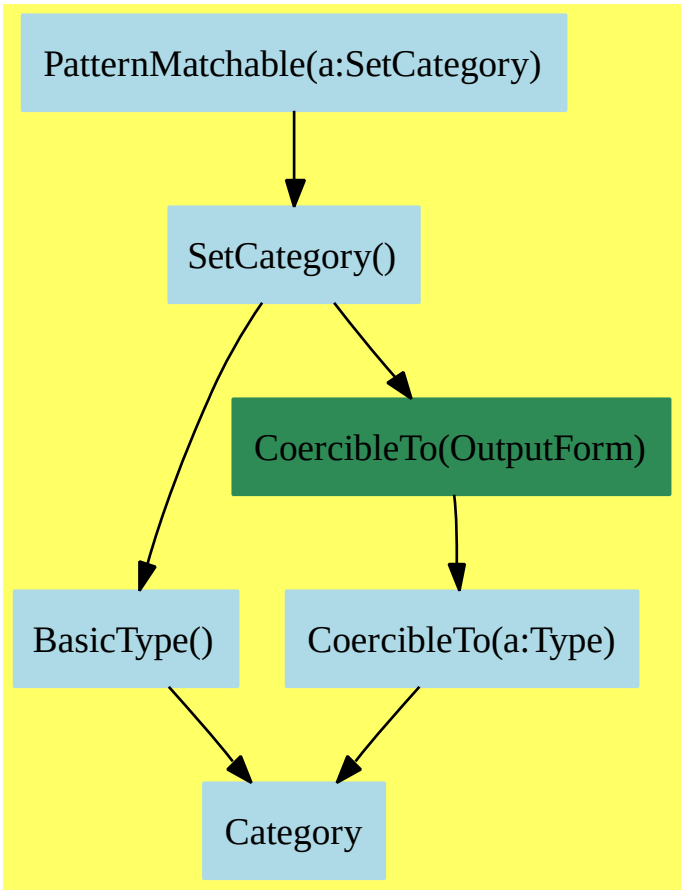
"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];

}
```

———-

4.0.58 PatternMatchable (PATMAB)



— **PatternMatchable.input** —

```
)set break resume
)sys rm -f PatternMatchable.output
)spool PatternMatchable.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show PatternMatchable
--R PatternMatchable S: SetCategory  is a category constructor
--R Abbreviation for PatternMatchable is PATMAB
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PATMAB
```

```
--R
--R------------------------------ Operations ------------------------------
--R ?=? : (%,%) -> Boolean                coerce : % -> OutputForm
--R hash : % -> SingleInteger             latex : % -> String
--R ?~=? : (%,%) -> Boolean
--R patternMatch : (%,Pattern S,PatternMatchResult(S,%)) -> PatternMatchResult(S,%)
--R
--E 1

)spool
)lisp (bye)
```

———-

— **PatternMatchable.help** —

```
=====================================================================
PatternMatchable examples
=====================================================================

See Also:
o )show PatternMatchable
```

———-

See:

⇒ “RealNumberSystem” (RNS) 17.0.200 on page 1400
⇐ “SetCategory” (SETCAT) 3.0.34 on page 113

Exports:

coerce hash latex patternMatch ?=?
?~=?

These are directly exported but not implemented:

```
patternMatch :
  (%,Pattern S,PatternMatchResult(S,%))
      -> PatternMatchResult(S,%)
```

These exports come from (p113) SetCategory():

```
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
```

— category PATMAB PatternMatchable —

```
)abbrev category PATMAB PatternMatchable
++ Category of sets that can be pattern-matched on
++ Author: Manuel Bronstein
++ Date Created: 28 Nov 1989
++ Date Last Updated: 15 Mar 1990
++ Keywords: pattern, matching.
++ Description:
++ A set R is PatternMatchable over S if elements of R can
++ be matched to patterns over S.

PatternMatchable(S:SetCategory): Category == SetCategory with
  patternMatch: (%, Pattern S, PatternMatchResult(S, %)) ->
                                              PatternMatchResult(S, %)
    ++ patternMatch(expr, pat, res) matches the pattern pat to the
    ++ expression expr. res contains the variables of pat which
    ++ are already matched and their matches (necessary for recursion).
    ++ Initially, res is just the result of \spadfun{new}
    ++ which is an empty list of matches.
```

———-

— PATMAB.dotabb —

```
"PATMAB"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PATMAB"];
"PATMAB" -> "SETCAT"
```

———-

— PATMAB.dotfull —

```
"PatternMatchable(a:SetCategory)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PATMAB"];
"PatternMatchable(a:SetCategory)" -> "SetCategory()"

"PatternMatchable(Integer)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=PATMAB"];
"PatternMatchable(Integer)" -> "PatternMatchable(a:SetCategory)"

"PatternMatchable(Float)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=PATMAB"];
"PatternMatchable(Float)" -> "PatternMatchable(a:SetCategory)"
```

———-

— PATMAB.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"PatternMatchable(a:SetCategory)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PATMAB"];
"PatternMatchable(a:SetCategory)" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

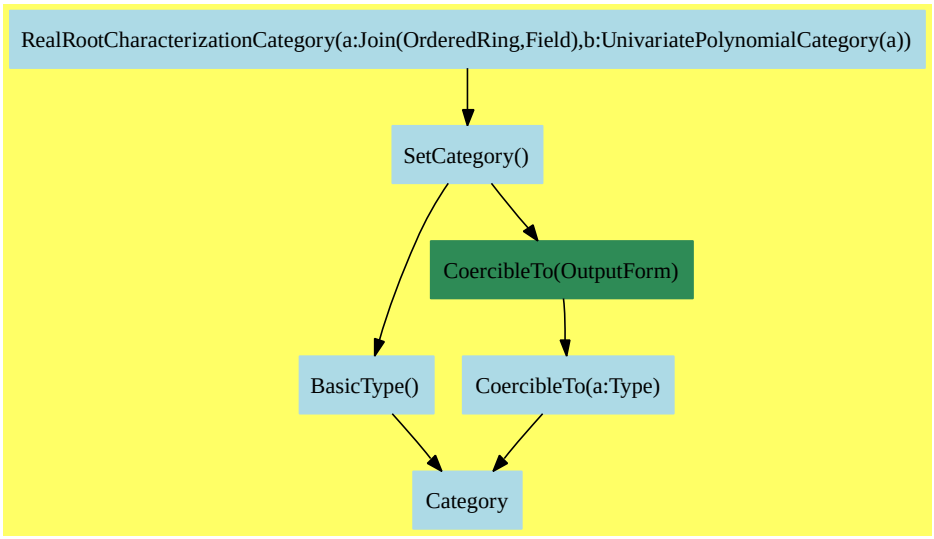
"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];

}
```

———-

4.0.59 RealRootCharacterizationCategory (RRCC)



— RealRootCharacterizationCategory.input —

```
)set break resume
)sys rm -f RealRootCharacterizationCategory.output
)spool RealRootCharacterizationCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show RealRootCharacterizationCategory
--R RealRootCharacterizationCategory(TheField: Join(OrderedRing,Field),ThePols: UnivariatePolyno
--R Abbreviation for RealRootCharacterizationCategory is RRCC
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for RRCC
--R
--R------------------------------ Operations ------------------------------
--R ?=? : (%,%) -> Boolean                allRootsOf : ThePols -> List %
--R coerce : % -> OutputForm              definingPolynomial : % -> ThePols
--R hash : % -> SingleInteger             latex : % -> String
--R sign : (ThePols,%) -> Integer         zero? : (ThePols,%) -> Boolean
--R ?~=? : (%,%) -> Boolean
--R approximate : (ThePols,%,TheField) -> TheField
--R negative? : (ThePols,%) -> Boolean
--R positive? : (ThePols,%) -> Boolean
--R recip : (ThePols,%) -> Union(ThePols,"failed")
--R relativeApprox : (ThePols,%,TheField) -> TheField
```

```
--R rootOf : (ThePols,PositiveInteger) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **RealRootCharacterizationCategory.help** —

```
=====================================================================
RealRootCharacterizationCategory examples
=====================================================================

See Also:
o )show RealRootCharacterizationCategory
```

———-

See:

⇐ "SetCategory" (SETCAT) 3.0.34 on page 113

Exports:

allRootsOf	approximate	coerce	definingPolynomial
hash	latex	negative?	positive?
recip	relativeApprox	rootOf	sign
zero?	?=?	? =?	

These are directly exported but not implemented:

```
approximate : (ThePols,%,TheField) -> TheField
allRootsOf : ThePols -> List %
definingPolynomial : % -> ThePols
relativeApprox : (ThePols,%,TheField) -> TheField
sign : (ThePols,%) -> Integer
```

These are implemented by this category:

```
negative? : (ThePols,%) -> Boolean
positive? : (ThePols,%) -> Boolean
recip : (ThePols,%) -> Union(ThePols,"failed")
rootOf : (ThePols,PositiveInteger) -> Union(%,"failed")
zero? : (ThePols,%) -> Boolean
```

These exports come from (p113) SetCategory():

```
 coerce : % -> OutputForm
 hash : % -> SingleInteger
 latex : % -> String
 ?=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
```

— category RRCC RealRootCharacterizationCategory —

```
)abbrev category RRCC RealRootCharacterizationCategory
++ Author: Renaud Rioboo
++ Date Created: summer 1992
++ Date Last Updated: January 2004
++ Basic Functions: provides operations with generic real roots of
++                  polynomials
++ Related Constructors: RealClosure, RightOpenIntervalRootCharacterization
++ Also See:
++ AMS Classifications:
++ Keywords: Real Algebraic Numbers
++ References:
++ Description:
++ \axiomType{RealRootCharacterizationCategory} provides common acces
++ functions for all real root codings.

RealRootCharacterizationCategory(TheField, ThePols ) : Category == PUB where

   TheField : Join(OrderedRing, Field)
   ThePols : UnivariatePolynomialCategory(TheField)

   Z ==> Integer
   N ==> PositiveInteger

   PUB ==>
     SetCategory with

        sign:                ( ThePols, $ )   ->            Z
              ++ \axiom{sign(pol,aRoot)} gives the sign of \axiom{pol}
              ++ interpreted as \axiom{aRoot}
        zero? :              ( ThePols, $ )   ->         Boolean
              ++ \axiom{zero?(pol,aRoot)} answers if \axiom{pol}
              ++ interpreted as \axiom{aRoot} is \axiom{0}
        negative?:           ( ThePols, $ )   ->         Boolean
              ++ \axiom{negative?(pol,aRoot)} answers if \axiom{pol}
              ++ interpreted as \axiom{aRoot} is negative
        positive?:           ( ThePols, $ )   ->         Boolean
              ++ \axiom{positive?(pol,aRoot)} answers if \axiom{pol}
              ++ interpreted as \axiom{aRoot} is positive
        recip:               ( ThePols, $ )   ->   Union(ThePols,"failed")
              ++ \axiom{recip(pol,aRoot)} tries to inverse \axiom{pol}
              ++ interpreted as \axiom{aRoot}
```

```
  definingPolynomial:         $         ->         ThePols
        ++ \axiom{definingPolynomial(aRoot)} gives a polynomial
        ++ such that \axiom{definingPolynomial(aRoot).aRoot = 0}
  allRootsOf:              ThePols      ->          List $
        ++ \axiom{allRootsOf(pol)} creates all the roots of \axiom{pol}
        ++ in the Real Closure, assumed in order.
  rootOf:              ( ThePols, N )   ->      Union($,"failed")
        ++ \axiom{rootOf(pol,n)} gives the nth root for the order of the
        ++ Real Closure
  approximate :  (ThePols,$,TheField)   ->    TheField
        ++ \axiom{approximate(term,root,prec)} gives an approximation
        ++ of \axiom{term} over \axiom{root} with precision \axiom{prec}

  relativeApprox :  (ThePols,$,TheField)   ->    TheField
        ++ \axiom{approximate(term,root,prec)} gives an approximation
        ++ of \axiom{term} over \axiom{root} with precision \axiom{prec}

add

  zero?(toTest, rootChar) ==
    sign(toTest, rootChar) = 0

  negative?(toTest, rootChar) ==
    sign(toTest, rootChar) < 0

  positive?(toTest, rootChar) ==
    sign(toTest, rootChar) > 0

  rootOf(pol,n) ==
    liste:List($):= allRootsOf(pol)
    # liste > n => "failed"
    liste.n

  recip(toInv,rootChar) ==
    degree(toInv) = 0 =>
      res := recip(leadingCoefficient(toInv))
      if (res case "failed") then "failed" else (res::TheField::ThePols)
    defPol := definingPolynomial(rootChar)
    d := principalIdeal([defPol,toInv])
    zero?(d.generator,rootChar) => "failed"
    if (degree(d.generator) ^= 0 )
    then
      defPol := (defPol exquo (d.generator))::ThePols
      d := principalIdeal([defPol,toInv])
    d.coef.2


      ———————-
```

— **RRCC.dotabb** —

```
"RRCC"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RRCC"];
"RRCC" -> "SETCAT"
```

———-

— **RRCC.dotfull** —

```
"RealRootCharacterizationCategory(a:Join(OrderedRing,Field),b:UnivariatePolynomialCategory(a))"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RRCC"];
"RealRootCharacterizationCategory(a:Join(OrderedRing,Field),b:UnivariatePolynomialCategory(a))"
  -> "SetCategory()"
```

———-

— **RRCC.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"RealRootCharacterizationCategory(a:Join(OrderedRing,Field),b:UnivariatePolynomialCategory(a))"
 [color=lightblue];
"RealRootCharacterizationCategory(a:Join(OrderedRing,Field),b:UnivariatePolynomialCategory(a))"
  -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

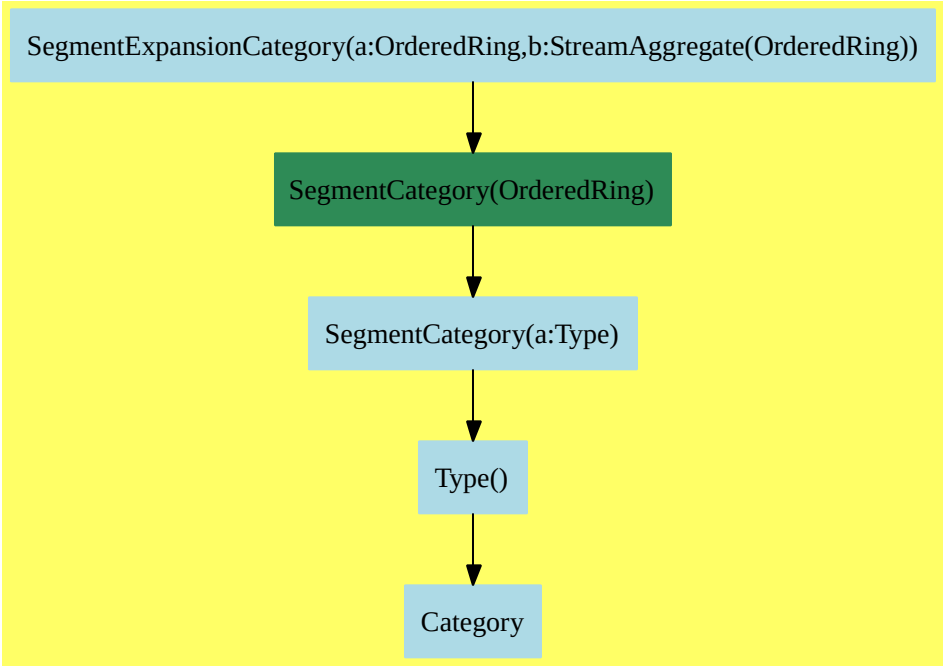
"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];

}
```

———-

4.0.60 SegmentExpansionCategory (SEGXCAT)



— **SegmentExpansionCategory.input** —

```
)set break resume
)sys rm -f SegmentExpansionCategory.output
)spool SegmentExpansionCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show SegmentExpansionCategory
--R SegmentExpansionCategory(S: OrderedRing,L: StreamAggregate t#1)  is a category constructor
--R Abbreviation for SegmentExpansionCategory is SEGXCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for SEGXCAT
--R
--R------------------------------ Operations ------------------------------
--R BY : (%,Integer) -> %                 ?..? : (S,S) -> %
```

```
--R convert : S -> %                    expand : % -> L
--R expand : List % -> L                hi : % -> S
--R high : % -> S                       incr : % -> Integer
--R lo : % -> S                         low : % -> S
--R map : ((S -> S),%) -> L             segment : (S,S) -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **SegmentExpansionCategory.help** —

```
=====================================================================
SegmentExpansionCategory examples
=====================================================================

See Also:
o )show SegmentExpansionCategory
```

———-

See:

⇐ “SegmentCategory” (SEGCAT) 3.0.33 on page 109

Exports:

BY	convert	expand	hi	high
incr	lo	low	map	segment
?..?				

Attributes exported:

- **nil**

These are directly exported but not implemented:

```
expand : % -> L
expand : List % -> L
map : ((S -> S),%) -> L
```

These exports come from (p109) SegmentCategory(OrderedRing):

```
BY : (%,Integer) -> %
convert : S -> %
```

```
 hi : % -> S
 high : % -> S
 incr : % -> Integer
 lo : % -> S
 low : % -> S
 segment : (S,S) -> %
 ?..? : (S,S) -> %
```

— category SEGXCAT SegmentExpansionCategory —

```
)abbrev category SEGXCAT SegmentExpansionCategory
++ Author:  Stephen M. Watt
++ Date Created: June 5, 1991
++ Date Last Updated:
++ Basic Operations:
++ Related Domains: Segment, UniversalSegment
++ Also See:
++ AMS Classifications:
++ Keywords:
++ Examples:
++ References:
++ Description:
++ This category provides an interface for expanding segments to
++ a stream of elements.

SegmentExpansionCategory(S: OrderedRing, L: StreamAggregate(S)): Category ==
    SegmentCategory(S) with
      expand: List % -> L
        ++ expand(l) creates a new value of type L in which each segment
        ++ \spad{l..h by k} is replaced with \spad{l, l+k, ... lN},
        ++ where \spad{lN <= h < lN+k}.
        ++ For example, \spad{expand [1..4, 7..9] = [1,2,3,4,7,8,9]}.
      expand: % -> L
        ++ expand(l..h by k) creates value of type L with elements
        ++ \spad{l, l+k, ... lN} where \spad{lN <= h < lN+k}.
        ++ For example, \spad{expand(1..5 by 2) = [1,3,5]}.
      map: (S -> S, %) -> L
        ++ map(f,l..h by k) produces a value of type L by applying f
        ++ to each of the succesive elements of the segment, that is,
        ++ \spad{[f(l), f(l+k), ..., f(lN)]}, where \spad{lN <= h < lN+k}.
```

———-

— SEGXCAT.dotabb —

```
"SEGXCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SEGXCAT"];
```

```
"SEGXCAT" -> "SEGCAT"
```

———-

— SEGXCAT.dotfull —

```
"SegmentExpansionCategory(a:OrderedRing,b:StreamAggregate(OrderedRing))"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SEGXCAT"];
"SegmentExpansionCategory(a:OrderedRing,b:StreamAggregate(OrderedRing))"
   -> "SegmentCategory(OrderedRing)"
```

———-

— SEGXCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"SegmentExpansionCategory(a:OrderedRing,b:StreamAggregate(OrderedRing))"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SEGXCAT"];
"SegmentExpansionCategory(a:OrderedRing,b:StreamAggregate(OrderedRing))"
   -> "SegmentCategory(OrderedRing)"

"SegmentCategory(OrderedRing)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=SEGCAT"];
"SegmentCategory(OrderedRing)" -> "SegmentCategory(a:Type)"

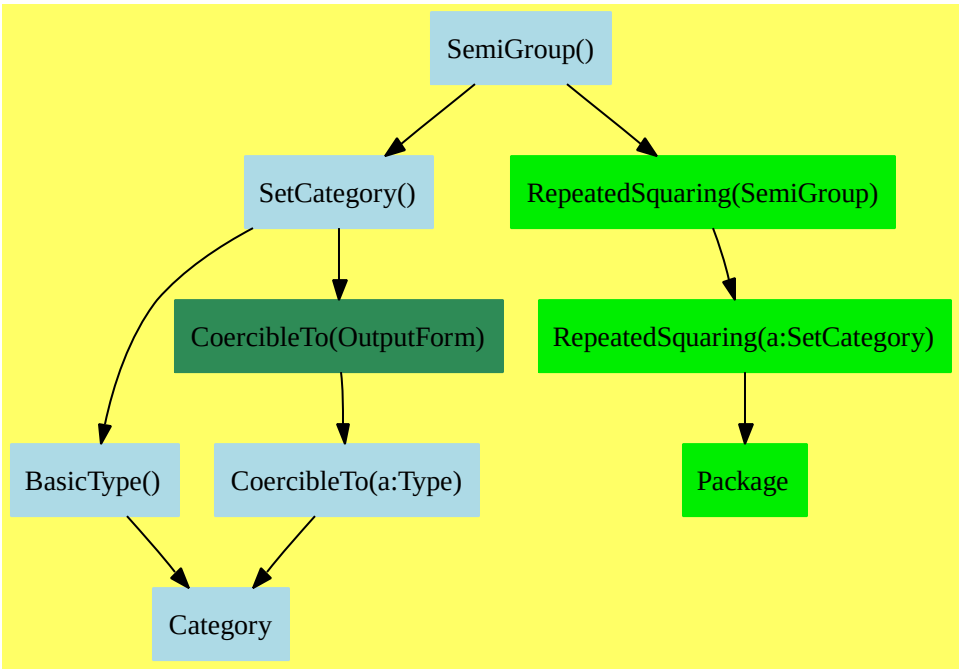
"SegmentCategory(a:Type)" [color=lightblue];
"SegmentCategory(a:Type)" -> "Type()"

"Type()" [color=lightblue];
"Type()" -> "Category"

"Category" [color=lightblue];
}
```

———-

4.0.61 SemiGroup (SGROUP)



A Semigroup is defined as a set S with a binary multiplicative operator "*". A Semigroup $G(S, *)$ is:

- a set S which can be null
- a binary multiplicative operator "*"
- associative. $\forall a, b, c \in S, a * (b * c) = (a * b) * c$

— SemiGroup.input —

```
)set break resume
)sys rm -f SemiGroup.output
)spool SemiGroup.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show SemiGroup
--R SemiGroup  is a category constructor
--R Abbreviation for SemiGroup is SGROUP
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for SGROUP
```

```
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (%,%) -> %                      ?**? : (%,PositiveInteger) -> %
--R ?=? : (%,%) -> Boolean                ?^? : (%,PositiveInteger) -> %
--R coerce : % -> OutputForm              hash : % -> SingleInteger
--R latex : % -> String                   ?~=? : (%,%) -> Boolean
--R
--E 1

)spool
)lisp (bye)
```

———-

— **SemiGroup.help** —

```
======================================================================
SemiGroup examples
======================================================================

See Also:
o )show SemiGroup
```

———-

See:

⇒ “FunctionSpace” (FS) 17.0.195 on page 1336
⇒ “Monoid” (MONOID) 5.0.76 on page 331
⇒ “Rng” (RNG) 8.0.121 on page 719
⇐ “SetCategory” (SETCAT) 3.0.34 on page 113

Exports:

coerce	hash	latex	?*?	?**?
?=?	?^?	?~=?		

These are directly exported but not implemented:

```
?*? : (%,%) -> %
```

These are implemented by this category:

```
?**? : (%,PositiveInteger) -> %
?^? : (%,PositiveInteger) -> %
```

These exports come from (p113) SetCategory():

```
 coerce : % -> OutputForm
 hash : % -> SingleInteger
 latex : % -> String
 ?=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
```

— category SGROUP SemiGroup —

```
)abbrev category SGROUP SemiGroup
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ the class of all multiplicative semigroups, i.e. a set
++ with an associative operation \spadop{*}.
++
++ Axioms\br
++ \tab{5}\spad{associative("*":(%,%)->%)}\tab{5}\spad{ (x*y)*z = x*(y*z)}
++
++ Conditional attributes\br
++ \tab{5}\spad{commutative("*":(%,%)->%)}\tab{5}\spad{ x*y = y*x }

SemiGroup(): Category == SetCategory with
      "*": (%,%) -> %                 ++ x*y returns the product of x and y.
      "**": (%,PositiveInteger) -> %  ++ x**n returns the repeated product
                                      ++ of x n times, i.e. exponentiation.
      "^": (%,PositiveInteger) -> %   ++ x^n returns the repeated product
                                      ++ of x n times, i.e. exponentiation.
    add
      import RepeatedSquaring(%)
      x:% ** n:PositiveInteger == expt(x,n)
      _^(x:%, n:PositiveInteger):% == x ** n
```

———-

— SGROUP.dotabb —

```
"SGROUP"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SGROUP"];
"SGROUP" -> "SETCAT"
```

———-

— SGROUP.dotfull —

```
"SemiGroup()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SGROUP"];
"SemiGroup()" -> "SetCategory()"
"SemiGroup()" -> "RepeatedSquaring(a:SemiGroup)"
```

———-

— SGROUP.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SetCategory()"
"SemiGroup()" -> "RepeatedSquaring(a:SemiGroup)"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"RepeatedSquaring(a:SemiGroup)" [color="#00EE00"];
"RepeatedSquaring(a:SemiGroup)" -> "RepeatedSquaring(a:SetCategory)"

"RepeatedSquaring(a:SetCategory)" [color="#00EE00"];
"RepeatedSquaring(a:SetCategory)" -> "Package"

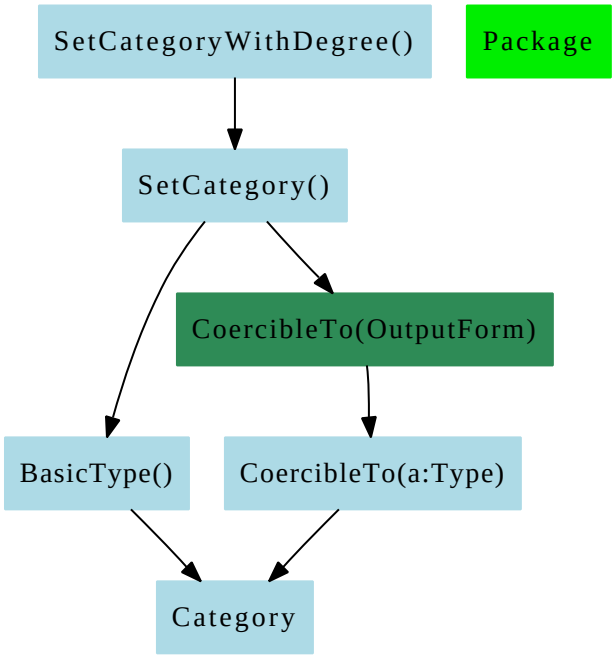
"Package" [color="#00EE00"];

"Category" [color=lightblue];

}
```

———-

4.0.62 SetCategoryWithDegree (SETCATD)



— **SetCategoryWithDegree.input** —

```
)set break resume
)sys rm -f SetCategoryWithDegree.output
)spool SetCategoryWithDegree.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show SetCategoryWithDegree
--R SetCategoryWithDegree  is a category constructor
--R Abbreviation for SetCategoryWithDegree is SETCATD
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for SETCATD
--R
--R------------------------------ Operations ------------------------------
--R ?=? : (%,%) -> Boolean                coerce : % -> OutputForm
```

```
--R degree : % -> PositiveInteger          hash : % -> SingleInteger
--R latex : % -> String                    ?~=? : (%,%) -> Boolean
--R
--E 1

)spool
)lisp (bye)
```

———-

— **SetCategoryWithDegree.help** —

```
=====================================================================
SetCategoryWithDegree examples
=====================================================================

See Also:
o )show SetCategoryWithDegree
```

———-

See:

⇒ “ProjectiveSpaceCategory” (PRSPCAT) 5.0.79 on page 345
⇐ “BasicType” (BASTYPE) 2.0.5 on page 14
⇐ “CoercibleTo” (KOERCE) 2.0.6 on page 17
⇐ “SetCategory” (SETCAT) 3.0.34 on page 113

Exports:

coerce degree hash latex ?=? ?~=?

These are directly exported but not implemented:

```
degree : % -> PositiveInteger
```

These exports come from (p113) SetCategory():

```
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
```

— **category SETCATD SetCategoryWithDegree** —

```
)abbrev category SETCATD SetCategoryWithDegree
++ Author: Gaetan Hache
++ Date Created: 17 nov 1992
++ Date Last Updated: May 2010 by Tim Daly
++ Keywords:
++ Description:
++ This is part of the PAFF package, related to projective space.
SetCategoryWithDegree:Category == SetCategory with
    degree: % -> PositiveInteger
```

———-

— SETCATD.dotabb —

```
"SETCATD"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SETCATD"];
"SETCATD" -> "SETCAT"
```

———-

— SETCATD.dotfull —

```
"SetCategoryWithDegree()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SETCATD"];
"SetCategoryWithDegree()" -> "SetCategory()"
```

———-

— SETCATD.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"SetCategoryWithDegree()" [color=lightblue];
"SetCategoryWithDegree()" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"
```

```
"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

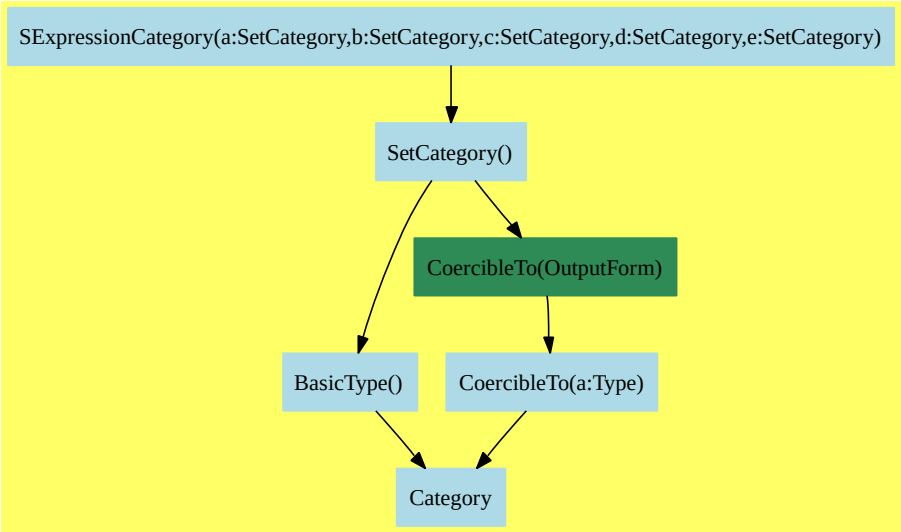
"Package" [color="#00EE00"];

"Category" [color=lightblue];

}
```

———-

4.0.63 SExpressionCategory (SEXCAT)



— SExpressionCategory.input —

```
)set break resume
)sys rm -f SExpressionCategory.output
)spool SExpressionCategory.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 1
)show SExpressionCategory
--R SExpressionCategory(Str: SetCategory,Sym: SetCategory,Int: SetCategory,Flt: SetCategory,Expr: SetCategor
--R Abbreviation for SExpressionCategory is SEXCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for SEXCAT
--R
--R------------------------------ Operations --------------------------------
--R #? : % -> Integer                    ?=? : (%,%) -> Boolean
--R atom? : % -> Boolean                 car : % -> %
--R cdr : % -> %                         coerce : % -> OutputForm
--R convert : Expr -> %                  convert : Flt -> %
--R convert : Int -> %                   convert : Sym -> %
--R convert : Str -> %                   convert : List % -> %
--R destruct : % -> List %               ?.? : (%,List Integer) -> %
--R ?.? : (%,Integer) -> %               eq : (%,%) -> Boolean
--R expr : % -> Expr                     float : % -> Flt
--R float? : % -> Boolean                hash : % -> SingleInteger
--R integer : % -> Int                   integer? : % -> Boolean
--R latex : % -> String                  list? : % -> Boolean
--R null? : % -> Boolean                 pair? : % -> Boolean
--R string : % -> Str                    string? : % -> Boolean
--R symbol : % -> Sym                    symbol? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R
--E 1

)spool
)lisp (bye)
```

———-

— SExpressionCategory.help —

```
=====================================================================
SExpressionCategory examples
=====================================================================

See Also:
o )show SExpressionCategory
```

———-

See:

⇐ “SetCategory” (SETCAT) 3.0.34 on page 113

Exports:

atom?	car	cdr	coerce	convert
destruct	eq	expr	float	float?
hash	integer	integer?	latex	list?
null?	pair?	string	string?	symbol
symbol?	#?	?=?	?~=?	?.?

These are directly exported but not implemented:

```
atom? : % -> Boolean
car : % -> %
cdr : % -> %
convert : Expr -> %
convert : Flt -> %
convert : Int -> %
convert : Sym -> %
convert : Str -> %
convert : List % -> %
destruct : % -> List %
eq : (%,%) -> Boolean
expr : % -> Expr
float : % -> Flt
float? : % -> Boolean
integer : % -> Int
integer? : % -> Boolean
list? : % -> Boolean
null? : % -> Boolean
pair? : % -> Boolean
string : % -> Str
string? : % -> Boolean
symbol : % -> Sym
symbol? : % -> Boolean
#? : % -> Integer
?.? : (%,List Integer) -> %
?.? : (%,Integer) -> %
```

These exports come from (p113) SetCategory():

```
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
```

— category SEXCAT SExpressionCategory —

```
)abbrev category SEXCAT SExpressionCategory
++ Category for Lisp values
++ Author: S.M.Watt
++ Date Created: July 1987
```

```
++ Date Last Modified: 23 May 1991
++ Description:
++ This category allows the manipulation of Lisp values while keeping
++ the grunge fairly localized.
--  The coerce to expression lets the
--  values be displayed in the usual parenthesized way (displaying
--  them as type Expression can cause the formatter to die, since
--  certain magic cookies are in unexpected places).
--  SMW July 87

SExpressionCategory(Str, Sym, Int, Flt, Expr): Category == Decl where
    Str, Sym, Int, Flt, Expr: SetCategory

    Decl ==> SetCategory with
        eq:        (%,%) -> Boolean
          ++ eq(s, t) is true if EQ(s,t) is true in Lisp.
        null?:     % -> Boolean
          ++ null?(s) is true if s is the S-expression ().
        atom?:     % -> Boolean
          ++ atom?(s) is true if s is a Lisp atom.
        pair?:     % -> Boolean
          ++ pair?(s) is true if s has is a non-null Lisp list.
        list?:     % -> Boolean
          ++ list?(s) is true if s is a Lisp list, possibly ().
        string?:   % -> Boolean
          ++ string?(s) is true if s is an atom and belong to Str.
        symbol?:   % -> Boolean
          ++ symbol?(s) is true if s is an atom and belong to Sym.
        integer?:  % -> Boolean
          ++ integer?(s) is true if s is an atom and belong to Int.
        float?:    % -> Boolean
          ++ float?(s) is true if s is an atom and belong to Flt.
        destruct:  % -> List %
          ++ destruct((a1,...,an)) returns the list [a1,...,an].
        string:    % -> Str
          ++ string(s) returns s as an element of Str.
          ++ Error: if s is not an atom that also belongs to Str.
        symbol:    % -> Sym
          ++ symbol(s) returns s as an element of Sym.
          ++ Error: if s is not an atom that also belongs to Sym.
        integer:   % -> Int
          ++ integer(s) returns s as an element of Int.
          ++ Error: if s is not an atom that also belongs to Int.
        float:     % -> Flt
          ++ float(s) returns s as an element of Flt;
          ++ Error: if s is not an atom that also belongs to Flt.
        expr:      % -> Expr
          ++ expr(s) returns s as an element of Expr;
          ++ Error: if s is not an atom that also belongs to Expr.
        convert:   List % -> %
```

```
          ++ convert([a1,...,an]) returns an S-expression \spad{(a1,...,an)}.
        convert:   Str    -> %
          ++ convert(x) returns the Lisp atom x;
        convert:   Sym    -> %
          ++ convert(x) returns the Lisp atom x.
        convert:   Int    -> %
          ++ convert(x) returns the Lisp atom x.
        convert:   Flt    -> %
          ++ convert(x) returns the Lisp atom x.
        convert:   Expr   -> %
          ++ convert(x) returns the Lisp atom x.
        car:       % -> %
          ++ car((a1,...,an)) returns a1.
        cdr:       % -> %
          ++ cdr((a1,...,an)) returns \spad{(a2,...,an)}.
        "#":       % -> Integer
          ++ #((a1,...,an)) returns n.
        elt:       (%, Integer)      -> %
          ++ elt((a1,...,an), i) returns \spad{ai}.
        elt:       (%, List Integer) -> %
          ++ elt((a1,...,an), [i1,...,im]) returns \spad{(a_i1,...,a_im)}.
```

———-

— SEXCAT.dotabb —

```
"SEXCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SEXCAT"];
"SEXCAT" -> "SETCAT"
```

———-

— SEXCAT.dotfull —

```
"SExpressionCategory(a:SetCategory,b:SetCategory,c:SetCategory,d:SetCategory,e:SetCategory)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SEXCAT"];
"SExpressionCategory(a:SetCategory,b:SetCategory,c:SetCategory,d:SetCategory,e:SetCategory)" ->
   "SetCategory()"
```

———-

— SEXCAT.dotpic —

```
digraph pic {
 fontsize=10;
```

```
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"SExpressionCategory(a:SetCategory,b:SetCategory,c:SetCategory,d:SetCategory,e:SetCategory)"
 [color=lightblue];
"SExpressionCategory(a:SetCategory,b:SetCategory,c:SetCategory,d:SetCategory,e:SetCategory)" ->
   "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

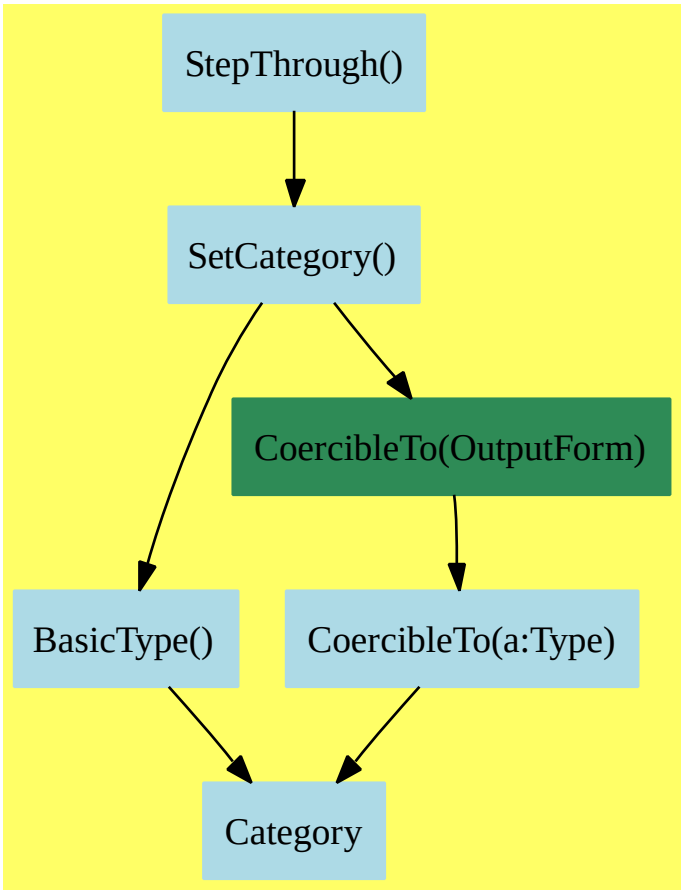
"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];

}
```

———————

4.0.64 StepThrough (STEP)



— **StepThrough.input** —

```
)set break resume
)sys rm -f StepThrough.output
)spool StepThrough.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show StepThrough
--R StepThrough  is a category constructor
--R Abbreviation for StepThrough is STEP
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for STEP
```

```
--R
--R------------------------------ Operations --------------------------------
--R ?=? : (%,%) -> Boolean                coerce : % -> OutputForm
--R hash : % -> SingleInteger             init : () -> %
--R latex : % -> String                   ?~=? : (%,%) -> Boolean
--R nextItem : % -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **StepThrough.help** —

```
========================================================================
StepThrough examples
========================================================================

See Also:
o )show StepThrough
```

———-

See:

⇒ “FiniteFieldCategory” (FFIELDC) 18.0.207 on page 1518
⇒ “IntegerNumberSystem” (INS) 16.0.186 on page 1228
⇒ “QuotientFieldCategory” (QFCAT) 17.0.198 on page 1376
⇒ “UnivariatePolynomialCategory” (UPOLYC) 17.0.204 on page 1471
⇐ “SetCategory” (SETCAT) 3.0.34 on page 113

Exports:

coerce	hash	init	latex	?=?
?~=?	nextItem			

These are directly exported but not implemented:

```
init : () -> %
nextItem : % -> Union(%,"failed")
```

These exports come from (p113) SetCategory():

```
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
```

```
 ?=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
```

— category STEP StepThrough —

```
)abbrev category STEP StepThrough
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A class of objects which can be 'stepped through'.
++ Repeated applications of \spadfun{nextItem} is guaranteed never to
++ return duplicate items and only return "failed" after exhausting
++ all elements of the domain.
++ This assumes that the sequence starts with \spad{init()}.
++ For infinite domains, repeated application
++ of \spadfun{nextItem} is not required to reach all possible domain elements
++ starting from any initial element.
++
++ Conditional attributes\br
++ \tab{5}infinite\tab{5}repeated nextItem's are never "failed".

StepThrough(): Category == SetCategory with
      init: constant -> %
        ++ init() chooses an initial object for stepping.
      nextItem: % -> Union(%,"failed")
        ++ nextItem(x) returns the next item, or "failed"
        ++ if domain is exhausted.
```

———-

— STEP.dotabb —

```
"STEP" [color=lightblue,href="bookvol10.2.pdf#nameddest=STEP"];
"STEP" -> "SETCAT"
```

———-

— STEP.dotfull —

```
"StepThrough()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=STEP"];
"StepThrough()" -> "SetCategory()"
```

———-

— STEP.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"StepThrough()" [color=lightblue];
"StepThrough()" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

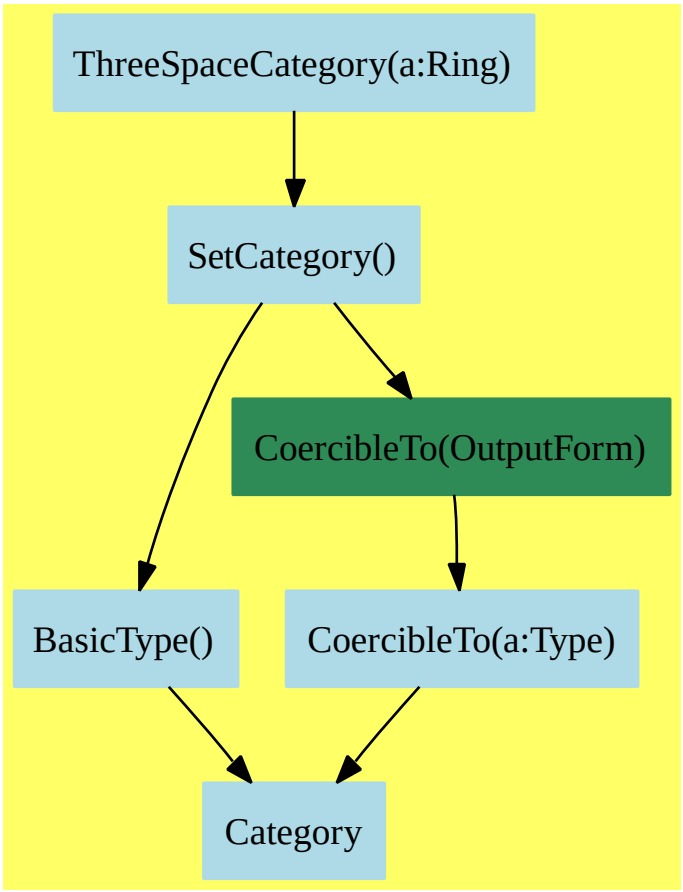
"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];

}
```

———-

4.0.65 ThreeSpaceCategory (SPACEC)



— **ThreeSpaceCategory.input** —

```
)set break resume
)sys rm -f ThreeSpaceCategory.output
)spool ThreeSpaceCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show ThreeSpaceCategory
--R ThreeSpaceCategory R: Ring  is a category constructor
--R Abbreviation for ThreeSpaceCategory is SPACEC
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for SPACEC
```

```
--R
--R------------------------------ Operations ------------------------------
--R ?=? : (%,%) -> Boolean                check : % -> %
--R closedCurve : % -> List Point R       closedCurve : List Point R -> %
--R closedCurve? : % -> Boolean           coerce : % -> OutputForm
--R components : % -> List %              composite : List % -> %
--R composites : % -> List %              copy : % -> %
--R create3Space : SubSpace(3,R) -> %     create3Space : () -> %
--R curve : % -> List Point R             curve : List Point R -> %
--R curve : (%,List List R) -> %          curve : (%,List Point R) -> %
--R curve? : % -> Boolean                 hash : % -> SingleInteger
--R latex : % -> String                   lp : % -> List Point R
--R merge : (%,%) -> %                    merge : List % -> %
--R mesh : % -> List List Point R         mesh : List List Point R -> %
--R mesh? : % -> Boolean                  point : % -> Point R
--R point : Point R -> %                  point : (%,List R) -> %
--R point : (%,Point R) -> %              point? : % -> Boolean
--R polygon : % -> List Point R           polygon : List Point R -> %
--R polygon : (%,List List R) -> %        polygon : (%,List Point R) -> %
--R polygon? : % -> Boolean               subspace : % -> SubSpace(3,R)
--R ?~=? : (%,%) -> Boolean
--R closedCurve : (%,List List R) -> %
--R closedCurve : (%,List Point R) -> %
--R enterPointData : (%,List Point R) -> NonNegativeInteger
--R lllip : % -> List List List NonNegativeInteger
--R lllp : % -> List List List Point R
--R llprop : % -> List List SubSpaceComponentProperty
--R lprop : % -> List SubSpaceComponentProperty
--R mesh : (List List Point R,Boolean,Boolean) -> %
--R mesh : (%,List List List R,Boolean,Boolean) -> %
--R mesh : (%,List List Point R,Boolean,Boolean) -> %
--R mesh : (%,List List List R,List SubSpaceComponentProperty,SubSpaceComponentProperty) -> %
--R mesh : (%,List List Point R,List SubSpaceComponentProperty,SubSpaceComponentProperty) -> %
--R modifyPointData : (%,NonNegativeInteger,Point R) -> %
--R numberOfComponents : % -> NonNegativeInteger
--R numberOfComposites : % -> NonNegativeInteger
--R objects : % -> Record(points: NonNegativeInteger,curves: NonNegativeInteger,polygons: NonNegativeIntege
--R point : (%,NonNegativeInteger) -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— ThreeSpaceCategory.help —

```
=====================================================================
```

```
ThreeSpaceCategory examples
===========================================================================

See Also:
o )show ThreeSpaceCategory
```

See:

⇐ "SetCategory" (SETCAT) 3.0.34 on page 113

Exports:

check	closedCurve	closedCurve?	coerce
components	composite	composites	copy
create3Space	curve	curve?	enterPointData
hash	latex	lllip	lllp
llprop	lp	lprop	merge
mesh	mesh?	modifyPointData	numberOfComponents
numberOfComposites	objects	point	point?
polygon	polygon?	subspace	?=?
?~=?			

These are directly exported but not implemented:

```
check : % -> %
closedCurve : (%,List List R) -> %
closedCurve : (%,List Point R) -> %
closedCurve : List Point R -> %
closedCurve : % -> List Point R
closedCurve? : % -> Boolean
coerce : % -> OutputForm
components : % -> List %
composite : List % -> %
composites : % -> List %
copy : % -> %
create3Space : () -> %
create3Space : SubSpace(3,R) -> %
curve : (%,List List R) -> %
curve : (%,List Point R) -> %
curve : List Point R -> %
curve : % -> List Point R
curve? : % -> Boolean
enterPointData : (%,List Point R) -> NonNegativeInteger
lllip : % -> List List List NonNegativeInteger
lllp : % -> List List List Point R
llprop : % -> List List SubSpaceComponentProperty
lp : % -> List Point R
```

```
lprop : % -> List SubSpaceComponentProperty
merge : List % -> %
merge : (%,%) -> %
mesh : % -> List List Point R
mesh : List List Point R -> %
mesh : (List List Point R,Boolean,Boolean) -> %
mesh : (%,List List List R,Boolean,Boolean) -> %
mesh : (%,List List Point R,Boolean,Boolean) -> %
mesh : (%,List List List R,
        List SubSpaceComponentProperty,
        SubSpaceComponentProperty) -> %
mesh : (%,List List Point R,
        List SubSpaceComponentProperty,
        SubSpaceComponentProperty) -> %
mesh? : % -> Boolean
modifyPointData : (%,NonNegativeInteger,Point R) -> %
numberOfComponents : % -> NonNegativeInteger
numberOfComposites : % -> NonNegativeInteger
objects : % ->
  Record(points: NonNegativeInteger,
         curves: NonNegativeInteger,
         polygons: NonNegativeInteger,
         constructs: NonNegativeInteger)
point : (%,Point R) -> %
point : (%,List R) -> %
point : (%,NonNegativeInteger) -> %
point : Point R -> %
point : % -> Point R
point? : % -> Boolean
polygon : (%,List Point R) -> %
polygon : (%,List List R) -> %
polygon : List Point R -> %
polygon : % -> List Point R
polygon? : % -> Boolean
subspace : % -> SubSpace(3,R)
```

These exports come from (p113) SetCategory():

```
hash : % -> SingleInteger
latex : % -> String
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
```

— category SPACEC ThreeSpaceCategory —

```
)abbrev category SPACEC ThreeSpaceCategory
++ Author:
++ Date Created:
++ Date Last Updated:
```

```
++ Basic Operations: create3Space, numberOfComponents, numberOfComposites,
++ merge, composite, components, copy, enterPointData, modifyPointData,
++ point, point?, curve, curve?, closedCurve, closedCurve?, polygon,
++ polygon? mesh, mesh?, lp, lllip, lllp, llprop, lprop, objects,
++ check, subspace, coerce
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The category ThreeSpaceCategory is used for creating
++ three dimensional objects using functions for defining points, curves,
++ polygons, constructs and the subspaces containing them.

ThreeSpaceCategory(R:Ring): Exports == Implementation where
 I    ==> Integer
 PI   ==> PositiveInteger
 NNI  ==> NonNegativeInteger
 L    ==> List
 B    ==> Boolean
 O    ==> OutputForm
 SUBSPACE ==> SubSpace(3,R)
 POINT   ==> Point(R)
 PROP    ==> SubSpaceComponentProperty()
 REP3D ==> Record(lp:L POINT,llliPt:L L L NNI, llProp:L L PROP, lProp:L PROP)
 OBJ3D   ==> Record(points:NNI, curves:NNI, polygons:NNI, constructs:NNI)

 Exports ==> Category
 Implementation ==>
  SetCategory with
    create3Space : () -> %
      ++ create3Space() creates a \spadtype{ThreeSpace} object capable of
      ++ holding point, curve, mesh components and any combination.
    create3Space : SUBSPACE -> %
      ++ create3Space(s) creates a \spadtype{ThreeSpace} object containing
      ++ objects pre-defined within some \spadtype{SubSpace} s.
    numberOfComponents : % -> NNI
      ++ numberOfComponents(s) returns the number of distinct
      ++ object components in the indicated \spadtype{ThreeSpace}, s, such
      ++ as points, curves, polygons, and constructs.
    numberOfComposites : % -> NNI
      ++ numberOfComposites(s) returns the number of supercomponents,
      ++ or composites, in the \spadtype{ThreeSpace}, s; Composites are
      ++ arbitrary groupings of otherwise distinct and unrelated components;
      ++ A \spadtype{ThreeSpace} need not have any composites defined at all
      ++ and, outside of the requirement that no component can belong
      ++ to more than one composite at a time, the definition and
      ++ interpretation of composites are unrestricted.
    merge : L % -> %
```

```
  ++ merge([s1,s2,...,sn]) will create a new \spadtype{ThreeSpace} that
  ++ has the components of all the ones in the list; Groupings of
  ++ components into composites are maintained.
merge : (%,%) -> %
  ++ merge(s1,s2) will create a new \spadtype{ThreeSpace} that has the
  ++ components of \spad{s1} and \spad{s2}; Groupings of components
  ++ into composites are maintained.
composite : L % -> %
  ++ composite([s1,s2,...,sn]) will create a new \spadtype{ThreeSpace}
  ++ that is a union of all the components from each
  ++ \spadtype{ThreeSpace} in the parameter list, grouped as a composite.
components : % -> L %
  ++ components(s) takes the \spadtype{ThreeSpace} s, and creates a list
  ++ containing a unique \spadtype{ThreeSpace} for each single component
  ++ of s. If s has no components defined, the list returned is empty.
composites : % -> L %
  ++ composites(s) takes the \spadtype{ThreeSpace} s, and creates a list
  ++ containing a unique \spadtype{ThreeSpace} for each single composite
  ++ of s. If s has no composites defined (composites need to be
  ++ explicitly created), the list returned is empty. Note that not all
  ++ the components need to be part of a composite.
copy : % -> %
  ++ copy(s) returns a new \spadtype{ThreeSpace} that is an exact copy
  ++ of s.
enterPointData  : (%,L POINT) -> NNI
  ++ enterPointData(s,[p0,p1,...,pn]) adds a list of points from p0
  ++ through pn to the \spadtype{ThreeSpace}, s, and returns the index,
  ++ to the starting point of the list;
modifyPointData : (%,NNI,POINT) -> %
  ++ modifyPointData(s,i,p) changes the point at the indexed
  ++ location i in the \spadtype{ThreeSpace}, s, to that of point p.
  ++ This is useful for making changes to a point which has been
  ++ transformed.

-- 3D primitives
point        : (%,POINT) -> %
  ++ point(s,p) adds a point component defined by the point, p,
  ++ specified as a list from \spad{List(R)}, to the
  ++ \spadtype{ThreeSpace}, s, where R is the \spadtype{Ring} over
  ++ which the point is defined.
point        : (%,L R) -> %
  ++ point(s,[x,y,z]) adds a point component defined by a list of
  ++ elements which are from the \spad{PointDomain(R)} to the
  ++ \spadtype{ThreeSpace}, s, where R is the \spadtype{Ring} over
  ++ which the point elements are defined.
point        : (%,NNI) -> %
  ++ point(s,i) adds a point component which is placed into a component
  ++ list of the \spadtype{ThreeSpace}, s, at the index given by i.
point        : POINT -> %
  ++ point(p) returns a \spadtype{ThreeSpace} object which is composed
```

```
  ++ of one component, the point p.
point        : % -> POINT
  ++ point(s) checks to see if the \spadtype{ThreeSpace}, s, is
  ++ composed of only a single point and if so, returns the point.
  ++ An error is signaled otherwise.
point?       : % -> B
  ++ point?(s) queries whether the \spadtype{ThreeSpace}, s, is
  ++ composed of a single component which is a point and returns the
  ++ boolean result.
curve        : (%,L POINT) -> %
  ++ curve(s,[p0,p1,...,pn]) adds a space curve component defined by a
  ++ list of points \spad{p0} through \spad{pn}, to the
  ++ \spadtype{ThreeSpace} s.
curve        : (%,L L R) -> %
  ++ curve(s,[[p0],[p1],...,[pn]]) adds a space curve which is a list of
  ++ points p0 through pn defined by lists of elements from the domain
  ++ \spad{PointDomain(m,R)}, where R is the \spadtype{Ring} over which
  ++ the point elements are defined and m is the dimension of the
  ++ points, to the \spadtype{ThreeSpace} s.
curve         : L POINT -> %
  ++ curve([p0,p1,p2,...,pn]) creates a space curve defined
  ++ by the list of points \spad{p0} through \spad{pn}, and returns the
  ++ \spadtype{ThreeSpace} whose component is the curve.
curve         : % -> L POINT
  ++ curve(s) checks to see if the \spadtype{ThreeSpace}, s, is
  ++ composed of a single curve defined by a list of points and if so,
  ++ returns the curve, i.e., list of points. An error is signaled
  ++ otherwise.
curve?        : % -> B
  ++ curve?(s) queries whether the \spadtype{ThreeSpace}, s, is a curve,
  ++ i.e., has one component, a list of list of points, and returns
  ++ true if it is, or false otherwise.
closedCurve  : (%,L POINT) -> %
  ++ closedCurve(s,[p0,p1,...,pn,p0]) adds a closed curve component
  ++ which is a list of points defined by the first element p0 through
  ++ the last element pn and back to the first element p0 again, to the
  ++ \spadtype{ThreeSpace} s.
closedCurve  : (%,L L R) -> %
  ++ closedCurve(s,[[lr0],[lr1],...,[lrn],[lr0]]) adds a closed curve
  ++ component defined by a list of points \spad{lr0} through
  ++ \spad{lrn}, which are lists of elements from the domain
  ++ \spad{PointDomain(m,R)}, where R is the \spadtype{Ring} over which
  ++ the point elements are defined and m is the dimension of the
  ++ points, in which the last element of the list of points contains
  ++ a copy of the first element list, lr0.
  ++ The closed curve is added to the \spadtype{ThreeSpace}, s.
closedCurve         : L POINT -> %
  ++ closedCurve(lp) sets a list of points defined by the first element
  ++ of lp through the last element of lp and back to the first elelment
  ++ again and returns a \spadtype{ThreeSpace} whose component is the
```

```
  ++ closed curve defined by lp.
closedCurve         : % -> L POINT
  ++ closedCurve(s) checks to see if the \spadtype{ThreeSpace}, s, is
  ++ composed of a single closed curve component defined by a list of
  ++ points in which the first point is also the last point, all of
  ++ which are from the domain \spad{PointDomain(m,R)} and if so,
  ++ returns the list of points.  An error is signaled otherwise.
closedCurve?        : % -> B
  ++ closedCurve?(s) returns true if the \spadtype{ThreeSpace} s
  ++ contains a single closed curve component, i.e., the first element
  ++ of the curve is also the last element, or false otherwise.
polygon      : (%,L POINT) -> %
  ++ polygon(s,[p0,p1,...,pn]) adds a polygon component defined by a
  ++ list of points, p0 throught pn, to the \spadtype{ThreeSpace} s.
polygon      : (%,L L R) -> %
  ++ polygon(s,[[r0],[r1],...,[rn]]) adds a polygon component defined
  ++ by a list of points \spad{r0} through \spad{rn}, which are lists of
  ++ elements from the domain \spad{PointDomain(m,R)} to the
  ++ \spadtype{ThreeSpace} s, where m is the dimension of the points
  ++ and R is the \spadtype{Ring} over which the points are defined.
polygon         : L POINT -> %
  ++ polygon([p0,p1,...,pn]) creates a polygon defined by a list of
  ++ points, p0 through pn, and returns a \spadtype{ThreeSpace} whose
  ++ component is the polygon.
polygon         : % -> L POINT
  ++ polygon(s) checks to see if the \spadtype{ThreeSpace}, s, is
  ++ composed of a single polygon component defined by a list of
  ++ points, and if so, returns the list of points;  An error is
  ++ signaled otherwise.
polygon?        : % -> B
  ++ polygon?(s) returns true if the \spadtype{ThreeSpace} s contains
  ++ a single polygon component, or false otherwise.
mesh         : (%,L L POINT,L PROP,PROP) -> %
  ++ mesh(s,[[p0],[p1],...,[pn]],[props],prop) adds a surface component,
  ++ defined over a list curves which contains lists of points, to the
  ++ \spadtype{ThreeSpace} s; props is a list which contains the
  ++ subspace component properties for each surface parameter, and
  ++ prop is the subspace component property by which the points are
  ++ defined.
mesh         : (%,L L L R,L PROP,PROP) -> %
  ++ mesh(s,[ [[r10]...,[r1m]],[[r20]...,[r2m]],...,[[rn0]...,[rnm]] ],
  ++        [props], prop)
  ++ adds a surface component to the \spadtype{ThreeSpace} s, which is
  ++ defined over a rectangular domain of size WxH where W is the number
  ++ of lists of points from the domain \spad{PointDomain(R)} and H is
  ++ the number of elements in each of those lists; lprops is the list
  ++ of the subspace component properties for each curve list, and
  ++ prop is the subspace component property by which the points are
  ++ defined.
mesh         : (%,L L POINT,B,B) -> %
```

```
  ++ mesh(s,[[p0],[p1],...,[pn]], close1, close2) adds a surface
  ++ component to the \spadtype{ThreeSpace}, which is defined over a
  ++ list of curves, in which each of these curves is a list of points.
  ++ The boolean arguments close1 and close2 indicate how the surface
  ++ is to be closed. Argument close1 equal true
  ++ means that each individual list (a curve) is to be closed, i.e. the
  ++ last point of the list is to be connected to the first point.
  ++ Argument close2 equal true
  ++ means that the boundary at one end of the surface is to be
  ++ connected to the boundary at the other end, i.e. the boundaries
  ++ are defined as the first list of points (curve) and
  ++ the last list of points (curve).
mesh         : (%,L L L R,B,B) -> %
  ++ mesh(s,[ [[r10]...,[r1m]],[[r20]...,[r2m]],...,[[rn0]...,[rnm]] ],
  ++      close1, close2)
  ++ adds a surface component to the \spadtype{ThreeSpace} s, which is
  ++ defined over a rectangular domain of size WxH where W is the number
  ++ of lists of points from the domain \spad{PointDomain(R)} and H is
  ++ the number of elements in each of those lists; the booleans close1
  ++ and close2 indicate how the surface is to be closed: if close1 is
  ++ true this means that each individual list (a curve) is to be
  ++ closed (i.e.,
  ++ the last point of the list is to be connected to the first point);
  ++ if close2 is true, this means that the boundary at one end of the
  ++ surface is to be connected to the boundary at the other end
  ++ (the boundaries are defined as the first list of points (curve)
  ++ and the last list of points (curve)).
mesh         : L L POINT -> %
  ++ mesh([[p0],[p1],...,[pn]]) creates a surface defined by a list of
  ++ curves which are lists, p0 through pn, of points, and returns a
  ++ \spadtype{ThreeSpace} whose component is the surface.
mesh         : (L L POINT,B,B) -> %
  ++ mesh([[p0],[p1],...,[pn]], close1, close2) creates a surface
  ++ defined over a list of curves, p0 through pn, which are lists of
  ++ points; the booleans close1 and close2 indicate how the surface is
  ++ to be closed: close1 set to true means that each individual list
  ++ (a curve) is to be closed (that is, the last point of the list is
  ++ to be connected to the first point); close2 set to true means
  ++ that the boundary at one end of the surface is to be connected to
  ++ the boundary at the other end (the boundaries are defined as the
  ++ first list of points (curve) and the last list of points (curve));
  ++ the \spadtype{ThreeSpace} containing this surface is returned.
mesh         : % -> L L POINT
  ++ mesh(s) checks to see if the \spadtype{ThreeSpace}, s, is
  ++ composed of a single surface component defined by a list curves
  ++ which contain lists of points, and if so, returns the list of
  ++ lists of points;  An error is signaled otherwise.
mesh?        : % -> B
  ++ mesh?(s) returns true if the \spadtype{ThreeSpace} s is composed
  ++ of one component, a mesh comprising a list of curves which are lists
```

```
      ++ of points, or returns false if otherwise
    lp           : % -> L POINT
      ++ lp(s) returns the list of points component which the
      ++ \spadtype{ThreeSpace}, s, contains; these points are used by
      ++ reference, i.e., the component holds indices referring to the
      ++ points rather than the points themselves. This allows for sharing
      ++ of the points.
    lllip        : % -> L L L NNI
      ++ lllip(s) checks to see if the \spadtype{ThreeSpace}, s, is
      ++ composed of a list of components, which are lists of curves,
      ++ which are lists of indices to points, and if so, returns the list
      ++ of lists of lists;  An error is signaled otherwise.
    lllp         : % -> L L L POINT   -- used by view3D
      ++ lllp(s) checks to see if the \spadtype{ThreeSpace}, s, is
      ++ composed of a list of components, which are lists of curves,
      ++ which are lists of points, and if so, returns the list of
      ++ lists of lists;  An error is signaled otherwise.
    llprop       : % -> L L PROP      -- used by view3D
      ++ llprop(s) checks to see if the \spadtype{ThreeSpace}, s, is
      ++ composed of a list of curves which are lists of the
      ++ subspace component properties of the curves, and if so, returns the
      ++ list of lists;  An error is signaled otherwise.
    lprop        : % -> L PROP        -- used by view3D
      ++ lprop(s) checks to see if the \spadtype{ThreeSpace}, s, is
      ++ composed of a list of subspace component properties, and if so,
      ++ returns the list;  An error is signaled otherwise.
    objects      : % -> OBJ3D
      ++ objects(s) returns the \spadtype{ThreeSpace}, s, in the form of a
      ++ 3D object record containing information on the number of points,
      ++ curves, polygons and constructs comprising the
      ++ \spadtype{ThreeSpace}..
    check        : % -> %             -- used by mesh
      ++ check(s) returns lllpt, list of lists of lists of point information
      ++ about the \spadtype{ThreeSpace} s.
    subspace     : % -> SUBSPACE
      ++ subspace(s) returns the \spadtype{SubSpace} which holds all the
      ++ point information in the \spadtype{ThreeSpace}, s.
    coerce       : % -> O
      ++ coerce(s) returns the \spadtype{ThreeSpace} s to Output format.
```

———-

— SPACEC.dotabb —

```
"SPACEC"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SPACEC"];
"SPACEC" -> "SETCAT"
```

———-

— **SPACEC.dotfull** —

```
"ThreeSpaceCategory(a:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SPACEC"];
"ThreeSpaceCategory(a:Ring)" -> "SetCategory()"
```

———-

— **SPACEC.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"ThreeSpaceCategory(a:Ring)" [color=lightblue];
"ThreeSpaceCategory(a:Ring)" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];

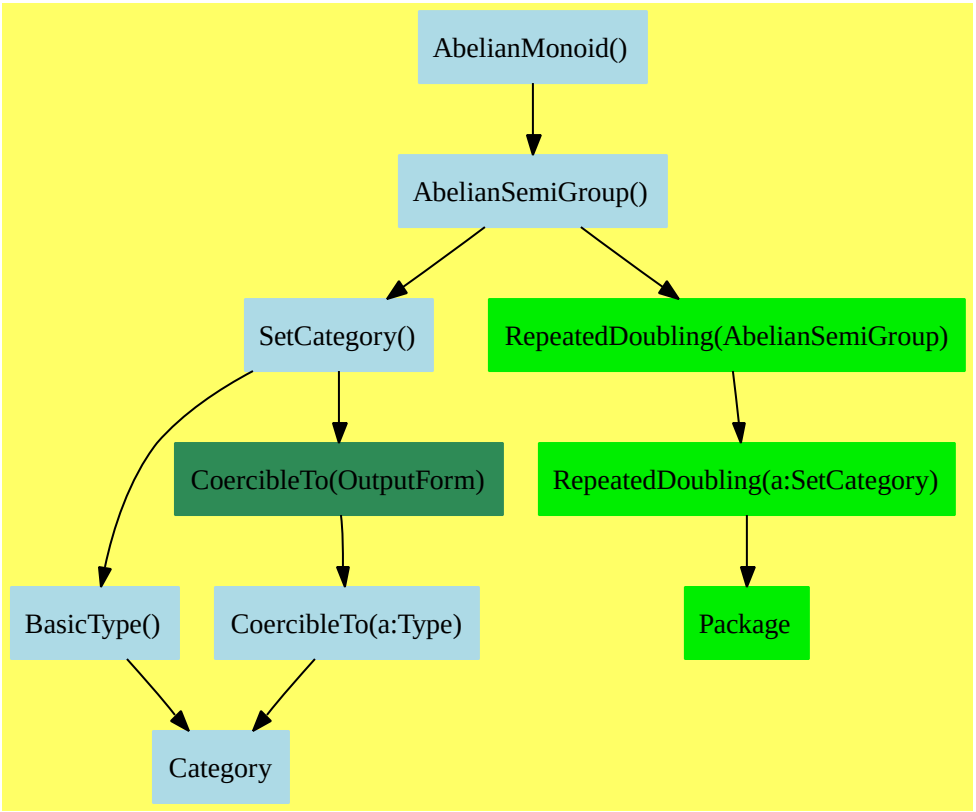
}
```

———-

Chapter 5

Category Layer 4

5.0.66 AbelianMonoid (ABELMON)



— **AbelianMonoid.input** —

```
)set break resume
)sys rm -f AbelianMonoid.output
)spool AbelianMonoid.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show AbelianMonoid
--R AbelianMonoid  is a category constructor
--R Abbreviation for AbelianMonoid is ABELMON
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for ABELMON
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (PositiveInteger,%) -> %        ?+? : (%,%) -> %
--R ?=? : (%,%) -> Boolean                0 : () -> %
--R coerce : % -> OutputForm              hash : % -> SingleInteger
--R latex : % -> String                   sample : () -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **AbelianMonoid.help** —

```
=====================================================================
AbelianMonoid examples
=====================================================================

See Also:
o )show AbelianMonoid
```

———-

See:

⇒ “CancellationAbelianMonoid” (CABMON) 6.0.83 on page 380
⇒ “FunctionSpace” (FS) 17.0.195 on page 1336
⇒ “OrderedAbelianMonoid” (OAMON) 7.0.105 on page 576

⇒ “OrderedAbelianSemiGroup” (OASGP) 6.0.89 on page 456
⇐ “AbelianSemiGroup” (ABELSG) 4.0.36 on page 123

Exports:

0	coerce	hash	latex	sample
zero?	?*?	?+?	?=?	?~=?

These are directly exported but not implemented:

```
0 : () -> %
```

These are implemented by this category:

```
sample : () -> %
zero? : % -> Boolean
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
```

These exports come from (p123) AbelianSemiGroup():

```
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?+? : (%,%) -> %
```

— category ABELMON AbelianMonoid —

```
)abbrev category ABELMON AbelianMonoid
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The class of multiplicative monoids, i.e. semigroups with an
++ additive identity element.
++
++ Axioms\br
++ \tab{5}\spad{leftIdentity("+":(%,%)->%,0)}\tab{5}\spad{ 0+x=x }\br
++ \tab{5}\spad{rightIdentity("+":(%,%)->%,0)}\tab{4}\spad{ x+0=x }
-- following domain must be compiled with subsumption disabled
-- define SourceLevelSubset to be EQUAL
```

```
AbelianMonoid(): Category == AbelianSemiGroup with
      0: constant -> %
        ++ 0 is the additive identity element.
      sample: constant -> %
        ++ sample yields a value of type %
      zero?: % -> Boolean
        ++ zero?(x) tests if x is equal to 0.
      "*": (NonNegativeInteger,%) -> %
        ++ n * x is left-multiplication by a non negative integer
    add
      import RepeatedDoubling(%)
      zero? x == x = 0
      n:PositiveInteger * x:% == (n::NonNegativeInteger) * x
      sample() == 0
      if not (% has Ring) then
        n:NonNegativeInteger * x:% ==
          zero? n => 0
          double(n pretend PositiveInteger,x)
```

———-

— **ABELMON.dotabb** —

```
"ABELMON"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ABELMON"];
"ABELMON" -> "ABELSG"
```

———-

— **ABELMON.dotfull** —

```
"AbelianMonoid()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ABELMON"];
"AbelianMonoid()" -> "AbelianSemiGroup()"
```

———-

— **ABELMON.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];
```

```
"AbelianMonoid()" [color=lightblue];
"AbelianMonoid()" -> "AbelianSemiGroup()"

"AbelianSemiGroup()" [color=lightblue];
"AbelianSemiGroup()" -> "SetCategory()"
"AbelianSemiGroup()" -> "RepeatedDoubling(AbelianSemiGroup)"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" ->
    "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

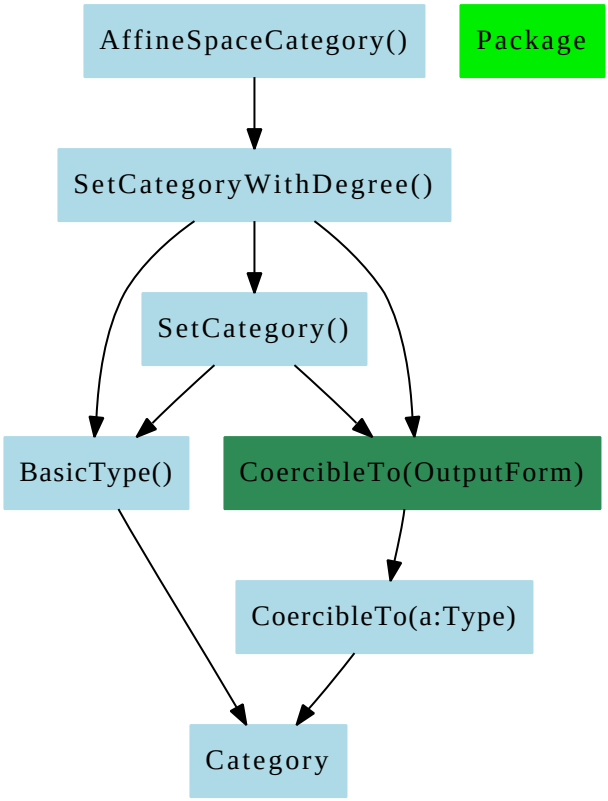
"RepeatedDoubling(AbelianSemiGroup)" [color="#00EE00"];
"RepeatedDoubling(AbelianSemiGroup)" -> "RepeatedDoubling(a:SetCategory)"

"RepeatedDoubling(a:SetCategory)" [color="#00EE00"];
"RepeatedDoubling(a:SetCategory)" -> "Package"

"Package" [color="#00EE00"];

"Category" [color=lightblue];
}
```

5.0.67 AffineSpaceCategory (AFSPCAT)



— **AffineSpaceCategory.input** —

```
)set break resume
)sys rm -f AffineSpaceCategory.output
)spool AffineSpaceCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show AffineSpaceCategory
--R AffineSpaceCategory K: Field  is a category constructor
--R Abbreviation for AffineSpaceCategory is AFSPCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for AFSPCAT
--R
--R------------------------------ Operations --------------------------------
```

```
--R ?=? : (%,%) -> Boolean                affinePoint : List K -> %
--R coerce : List K -> %                  coerce : % -> List K
--R coerce : % -> OutputForm              conjugate : % -> %
--R definingField : % -> K                degree : % -> PositiveInteger
--R ?.? : (%,Integer) -> K                hash : % -> SingleInteger
--R latex : % -> String                   list : % -> List K
--R orbit : % -> List %                   origin : () -> %
--R pointValue : % -> List K              rational? : % -> Boolean
--R setelt : (%,Integer,K) -> K           ?~=? : (%,%) -> Boolean
--R conjugate : (%,NonNegativeInteger) -> %
--R orbit : (%,NonNegativeInteger) -> List %
--R rational? : (%,NonNegativeInteger) -> Boolean
--R removeConjugate : List % -> List %
--R removeConjugate : (List %,NonNegativeInteger) -> List %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **AffineSpaceCategory.help** —

```
==========================================================================
AffineSpaceCategory examples
==========================================================================

See Also:
o )show AffineSpaceCategory
```

———-

See:

⇐ “SetCategoryWithDegree” (SETCATD) 4.0.62 on page 243

Exports:

?.?	?=?	?~=?	affinePoint	coerce
conjugate	definingField	degree	hash	latex
list	orbit	origin	pointValue	rational?
removeConjugate	setelt			

These are directly exported but not implemented

```
?.? : (%,Integer) -> K
affinePoint : List K -> %
```

```
coerce : % -> List K
coerce : List K -> %
conjugate : % -> %
conjugate : (%,NNI) -> %
definingField : % -> K
list : % -> List K
orbit : % -> List %
orbit : (%,NNI) -> List %
origin : () -> %
pointValue : % -> List K
rational? : % -> Boolean
rational? : (%,NNI) -> Boolean
removeConjugate : (List %,NNI) -> List %
removeConjugate : List % -> List %
setelt : (%,Integer,K) -> K
```

These exports come from (p243) SetCategoryWithDegree

```
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
coerce : % -> OutputForm
degree : % -> PositiveInteger
hash : % -> SingleInteger
latex : % -> String
```

— category AFSPCAT AffineSpaceCategory —

```
)abbrev category AFSPCAT AffineSpaceCategory
++ Author: Gaetan Hache
++ Date Created: 17 nov 1992
++ Date Last Updated: May 2010 by Tim Daly
++ Description:
++ The following is all the categories and domains related to projective
++ space and part of the PAFF package
AffineSpaceCategory(K:Field):Category == Implementation where

  LIST ==> List
  INT  ==> Integer
  NNI  ==> NonNegativeInteger

  Implementation ==>
   SetCategoryWithDegree with

    origin: () -> %

    affinePoint: LIST(K) -> %
      ++ affinePoint creates a affine point from a list

    definingField: % -> K
```

```
coerce: % -> List(K)

coerce: LIST(K) -> %
  ++ coerce a list of K to a affine point.

orbit: % -> LIST %
  ++ orbit(p) returns the orbit of the point p according to the
  ++ characteristic of K, that is, for q= char K,
  ++ orbit(p) = \{ p, p**q, p**(q**2), p**(q**3), ..... \}

orbit: (%,NNI) -> LIST %
  ++ orbit(p,n) returns the orbit of the point p according to n, that is
  ++ orbit(p,n) = \{ p, p**n, p**(n**2), p**(n**3), ..... \}

conjugate: (%,NNI) -> %
  ++ conjugate(p,n) returns p**n, that is all the coordinates of p
  ++ to the power of n

conjugate: % -> %
  ++ conjugate(p) returns conjugate(p,n) where n is the characteristic
  ++ of the ground field.

removeConjugate: (LIST(%),NNI) -> LIST(%)
  ++ removeConjugate(lp,n) returns a list of points such that no points
  ++ in the list is the conjugate (according to n) of another point.

removeConjugate: LIST(%)  -> LIST(%)
  ++ removeConjugate(lp) returns removeConjugate(lp,n)
  ++ where n is the characteristic of the ground field.

rational?: (%,NNI) -> Boolean
  ++ rational?(p,n) test if the point is rational according to n.

rational?: % -> Boolean
  ++ rational?(p) test if the point is rational according to the
  ++ characteristic of the ground field.

list  : % -> LIST(K)
  ++ list returns the list of the coordinates

elt: (%,INT) -> K
  ++ elt returns the value of a specified coordinates

setelt: (%,INT,K) -> K
  ++ setelt sets the value of a specified coordinates

pointValue: % -> LIST(K)
 ++ pointValue returns the coordinates of the point or of the point
 ++ of origin that represent an infinitly close point
```

———-

— **AFSPCAT.dotabb** —

```
"AFSPCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=AFSPCAT"];
"AFSPCAT" -> "SETCATD"
```

———-

— **AFSPCAT.dotfull** —

```
"AffineSpaceCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=AFSPCAT"];
"AffineSpaceCategory()" -> "SetCategoryWithDegree()"
```

———-

— **AFSPCAT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"AffineSpaceCategory()" [color=lightblue];
"AffineSpaceCategory()" -> "SetCategoryWithDegree()"

"SetCategoryWithDegree()" [color=lightblue];
"SetCategoryWithDegree()" -> "BasicType()"
"SetCategoryWithDegree()" -> "CoercibleTo(OutputForm)"
"SetCategoryWithDegree()" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
```

```
"CoercibleTo(a:Type)" -> "Category"

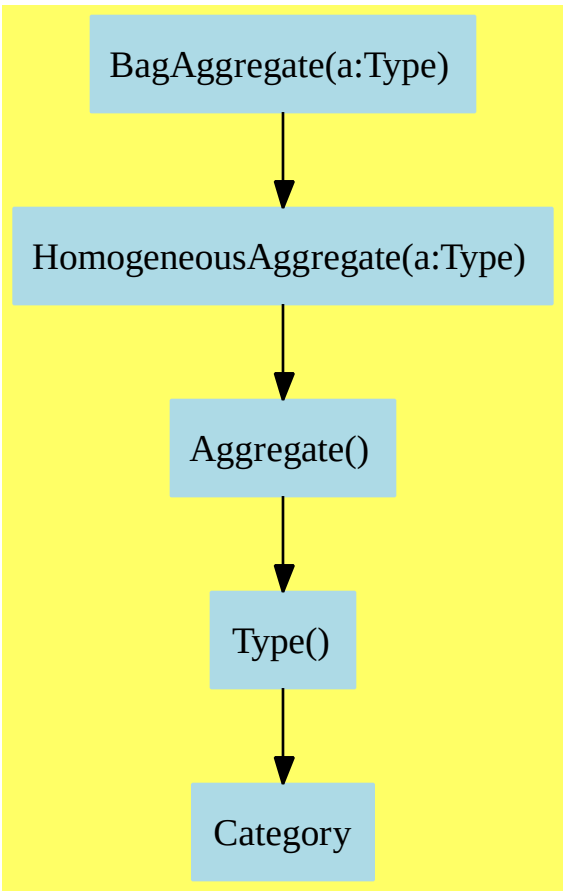
"Package" [color="#00EE00"];

"Category" [color=lightblue];

}
```

———-

5.0.68 BagAggregate (BGAGG)



— BagAggregate.input —

```
)set break resume
)sys rm -f BagAggregate.output
)spool BagAggregate.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show BagAggregate
--R BagAggregate S: Type  is a category constructor
--R Abbreviation for BagAggregate is BGAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for BGAGG
--R
--R------------------------------ Operations ------------------------------
--R bag : List S -> %                    copy : % -> %
--R empty : () -> %                      empty? : % -> Boolean
--R eq? : (%,%) -> Boolean               extract! : % -> S
--R insert! : (S,%) -> %                 inspect : % -> S
--R map : ((S -> S),%) -> %              sample : () -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R coerce : % -> OutputForm if S has SETCAT
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R hash : % -> SingleInteger if S has SETCAT
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List S if $ has finiteAggregate
--R size? : (%,NonNegativeInteger) -> Boolean
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

— **BagAggregate.help** —

```
====================================================================
BagAggregate examples
====================================================================

See Also:
o )show BagAggregate
```

———-

See:

⇒ “DictionaryOperations” (DIOPS) 6.0.84 on page 385
⇒ “PriorityQueueAggregate” (PRQAGG) 6.0.92 on page 480
⇒ “QueueAggregate” (QUAGG) 6.0.93 on page 486
⇒ “StackAggregate” (SKAGG) 6.0.95 on page 500
⇐ “HomogeneousAggregate” (HOAGG) 4.0.49 on page 181

Exports:

any?	bag	coerce	copy	count
empty	empty?	eq?	eval	every?
extract!	hash	insert!	inspect	latex
less?	map	map!	member?	members
more?	parts	sample	size?	#?
?=?	?~=?			

Attributes Exported:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **nil**

These are directly exported but not implemented:

```
extract! : % -> S
insert! : (S,%) -> %
inspect : % -> S
```

These are implemented by this category:

```
bag : List S -> %
```

These exports come from (p181) HomogeneousAggregate(S:Type):

```
any? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
coerce : % -> OutputForm if S has SETCAT
copy : % -> %
count : (S,%) -> NonNegativeInteger
         if S has SETCAT and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,S,S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,Equation S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,List Equation S) -> %
         if S has EVALAB S and S has SETCAT
every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
hash : % -> SingleInteger if S has SETCAT
latex : % -> String if S has SETCAT
less? : (%,NonNegativeInteger) -> Boolean
map : ((S -> S),%) -> %
map! : ((S -> S),%) -> % if $ has shallowlyMutable
members : % -> List S if $ has finiteAggregate
member? : (S,%) -> Boolean
         if S has SETCAT and $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
parts : % -> List S if $ has finiteAggregate
sample : () -> %
size? : (%,NonNegativeInteger) -> Boolean
\#? : % -> NonNegativeInteger if $ has finiteAggregate
?=? : (%,%) -> Boolean if S has SETCAT
?~=? : (%,%) -> Boolean if S has SETCAT
```

— category BGAGG BagAggregate —

```
)abbrev category BGAGG BagAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
```

```
++ A bag aggregate is an aggregate for which one can insert and extract
++ objects, and where the order in which objects are inserted determines
++ the order of extraction.
++ Examples of bags are stacks, queues, and dequeues.

BagAggregate(S:Type): Category == HomogeneousAggregate S with
   shallowlyMutable
     ++ shallowlyMutable means that elements of bags may be
     ++ destructively changed.
   bag: List S -> %
     ++ bag([x,y,...,z]) creates a bag with elements x,y,...,z.
   extract_!: % -> S
     ++ extract!(u) destructively removes a (random) item from bag u.
   insert_!: (S,%) -> %
     ++ insert!(x,u) inserts item x into bag u.
   inspect: % -> S
     ++ inspect(u) returns an (random) element from a bag.
 add
   bag(l) ==
     x:=empty()
     for s in l repeat x:=insert_!(s,x)
     x
```

———-

— BGAGG.dotabb —

```
"BGAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=BGAGG"];
"BGAGG" -> "HOAGG"
```

———-

— BGAGG.dotfull —

```
"BagAggregate(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=BGAGG"];
"BagAggregate(a:Type)" -> "HomogeneousAggregate(a:Type)"

"BagAggregate(a:SetCategory)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=BGAGG"];
"BagAggregate(a:SetCategory)" -> "BagAggregate(a:Type)"
```

———-

— BGAGG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"BagAggregate(a:Type)" [color=lightblue];
"BagAggregate(a:Type)" -> "HomogeneousAggregate(a:Type)"

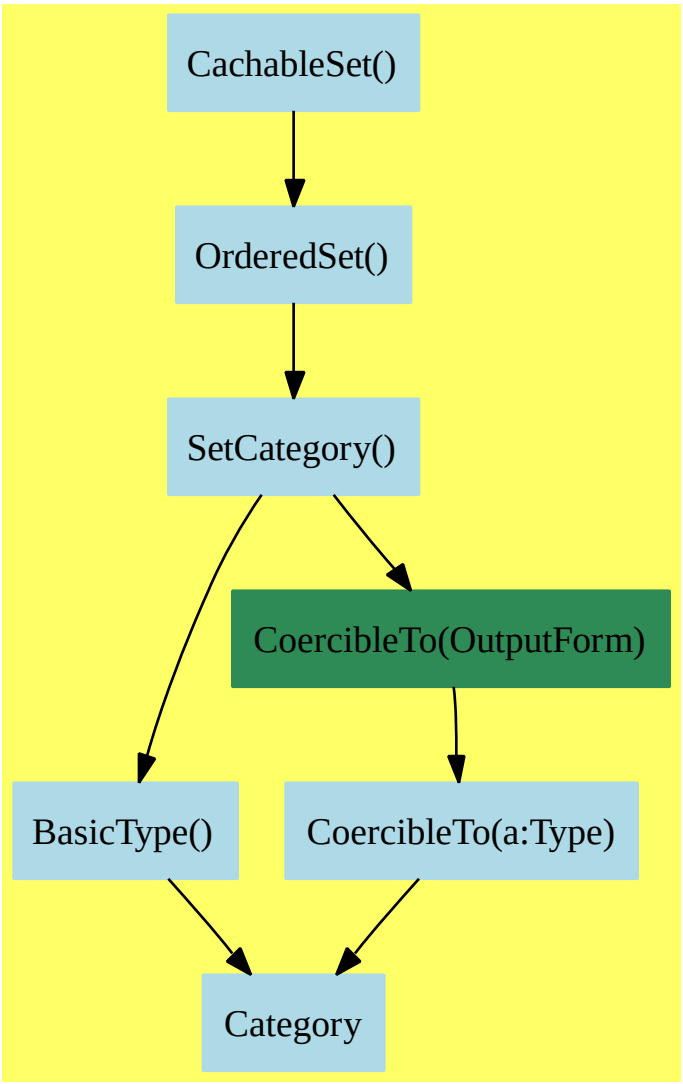
"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "Aggregate()"

"Aggregate()" [color=lightblue];
"Aggregate()" -> "Type()"

"Type()" [color=lightblue];
"Type()" -> "Category"

"Category" [color=lightblue];
}
```

5.0.69 CachableSet (CACHSET)



— CachableSet.input —

```
)set break resume
)sys rm -f CachableSet.output
)spool CachableSet.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 1
)show CachableSet
--R CachableSet  is a category constructor
--R Abbreviation for CachableSet is CACHSET
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for CACHSET
--R
--R------------------------------ Operations ------------------------------
--R ?<? : (%,%) -> Boolean                ?<=? : (%,%) -> Boolean
--R ?=? : (%,%) -> Boolean                ?>? : (%,%) -> Boolean
--R ?>=? : (%,%) -> Boolean               coerce : % -> OutputForm
--R hash : % -> SingleInteger             latex : % -> String
--R max : (%,%) -> %                      min : (%,%) -> %
--R ?~=? : (%,%) -> Boolean
--R position : % -> NonNegativeInteger
--R setPosition : (%,NonNegativeInteger) -> Void
--R
--E 1

)spool
)lisp (bye)
```

———-

— **CachableSet.help** —

```
=====================================================================
CachableSet examples
=====================================================================

See Also:
o )show CachableSet
```

———-

See:

⇐ “OrderedSet” (ORDSET) 4.0.56 on page 216

Exports:

coerce	hash	latex	max	min
position	setPosition	?~=?	?<?	?<=?
?=?	?>?	?>=?		

These are directly exported but not implemented:

```
position : % -> NonNegativeInteger
```

```
 setPosition : (%,NonNegativeInteger) -> Void
```

These exports come from (p216) OrderedSet():

```
 coerce : % -> OutputForm
 hash : % -> SingleInteger
 latex : % -> String
 max : (%,%) -> %
 min : (%,%) -> %
 ?~=? : (%,%) -> Boolean
 ?<? : (%,%) -> Boolean
 ?<=? : (%,%) -> Boolean
 ?=? : (%,%) -> Boolean
 ?>? : (%,%) -> Boolean
 ?>=? : (%,%) -> Boolean
```

— category CACHSET CachableSet —

```
)abbrev category CACHSET CachableSet
++ Sets whose elements can cache an integer
++ Author: Manuel Bronstein
++ Date Created: 31 Oct 1988
++ Date Last Updated: 14 May 1991
++ Description:
++ A cachable set is a set whose elements keep an integer as part
++ of their structure.

CachableSet: Category == OrderedSet with
  position   : % -> NonNegativeInteger
    ++ position(x) returns the integer n associated to x.
  setPosition: (%, NonNegativeInteger) -> Void
    ++ setPosition(x, n) associates the integer n to x.
```

———-

— CACHSET.dotabb —

```
"CACHSET"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=CACHSET"];
"CACHSET" -> "ORDSET"
```

———-

— CACHSET.dotfull —

```
"CachableSet()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=CACHSET"];
"CachableSet()" -> "OrderedSet()"
```

— CACHSET.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"CachableSet()" [color=lightblue];
"CachableSet()" -> "OrderedSet()"

"OrderedSet()" [color=lightblue];
"OrderedSet()" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

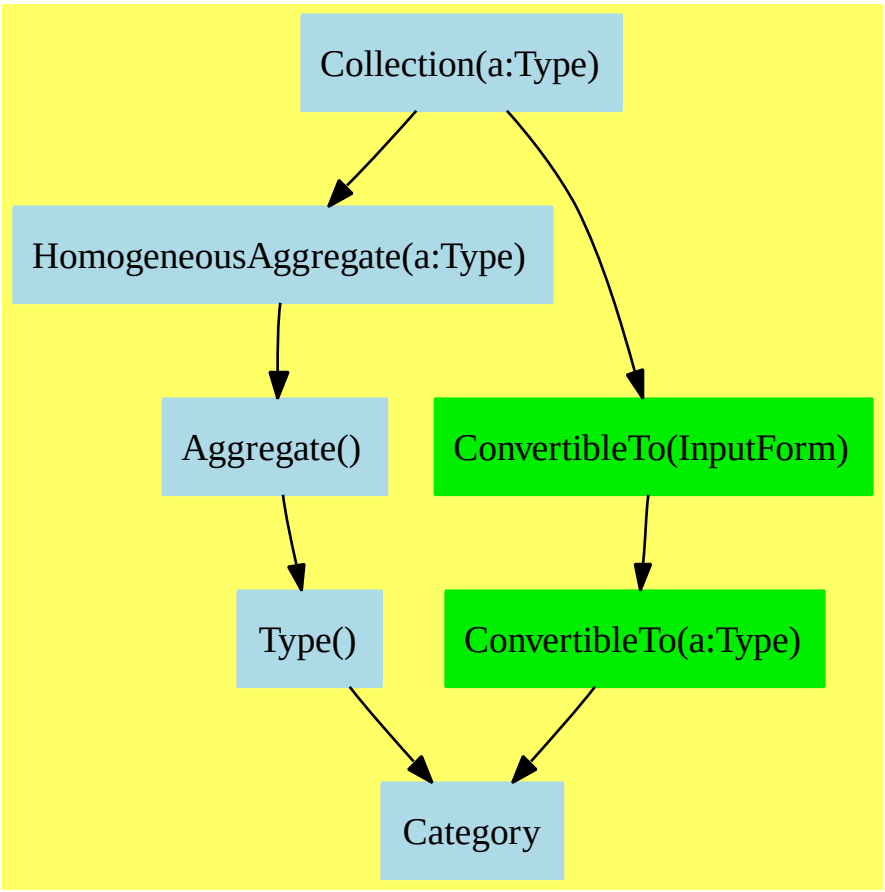
"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];
}
```

5.0.70 Collection (CLAGG)



— Collection.input —

```
)set break resume
)sys rm -f Collection.output
)spool Collection.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show Collection
--R Collection S: Type  is a category constructor
--R Abbreviation for Collection is CLAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for CLAGG
```

```
--R
--R------------------------------ Operations ------------------------------
--R construct : List S -> %               copy : % -> %
--R empty : () -> %                       empty? : % -> Boolean
--R eq? : (%,%) -> Boolean                map : ((S -> S),%) -> %
--R sample : () -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R coerce : % -> OutputForm if S has SETCAT
--R convert : % -> InputForm if S has KONVERT INFORM
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R find : ((S -> Boolean),%) -> Union(S,"failed")
--R hash : % -> SingleInteger if S has SETCAT
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S,S) -> S if S has SETCAT and $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%) -> S if $ has finiteAggregate
--R remove : (S,%) -> % if S has SETCAT and $ has finiteAggregate
--R remove : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R removeDuplicates : % -> % if S has SETCAT and $ has finiteAggregate
--R select : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R size? : (%,NonNegativeInteger) -> Boolean
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— Collection.help —

```
=====================================================================
Collection examples
=====================================================================
```

```
See Also:
o )show Collection
```

———-

See:

⇒ “DictionaryOperations” (DIOPS) 6.0.84 on page 385
⇒ “LinearAggregate” (LNAGG) 6.0.87 on page 404
⇒ “PolynomialSetCategory” (PSETCAT) 6.0.91 on page 465
⇒ “SetAggregate” (SETAGG) 6.0.94 on page 492
⇐ “ConvertibleTo” (KONVERT) 2.0.8 on page 23
⇐ “HomogeneousAggregate” (HOAGG) 4.0.49 on page 181

Exports:

any?	coerce	construct	copy	convert
count	empty	empty?	eq?	eval
every?	find	hash	latex	less?
map	map!	member?	members	more?
parts	reduce	remove	removeDuplicates	sample
select	size?	#?	?=?	?~=?

Attributes exported:

- **nil**

Attributes Used:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.

These are directly exported but not implemented:

```
construct: List S -> %
```

These are implemented by this category:

```
any? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
find : ((S -> Boolean),%) -> Union(S,"failed")
reduce : (((S,S) -> S),%,S,S) -> S
         if S has SETCAT and $ has finiteAggregate
reduce : (((S,S) -> S),%,S) -> S if $ has finiteAggregate
reduce : (((S,S) -> S),%) -> S if $ has finiteAggregate
```

```
remove : (S,%) -> % if S has SETCAT and $ has finiteAggregate
remove : ((S -> Boolean),%) -> % if $ has finiteAggregate
removeDuplicates : % -> %
         if S has SETCAT and $ has finiteAggregate
select : ((S -> Boolean),%) -> % if $ has finiteAggregate
#? : % -> NonNegativeInteger if $ has finiteAggregate
```

These exports come from (p181) HomogeneousAggregate(S:Type):

```
coerce : % -> OutputForm if S has SETCAT
copy : % -> %
count : (S,%) -> NonNegativeInteger
            if S has SETCAT and $ has finiteAggregate
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,S,S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,Equation S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,List Equation S) -> %
         if S has EVALAB S and S has SETCAT
every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
hash : % -> SingleInteger if S has SETCAT
latex : % -> String if S has SETCAT
less? : (%,NonNegativeInteger) -> Boolean
map : ((S -> S),%) -> %
map! : ((S -> S),%) -> % if $ has shallowlyMutable
member? : (S,%) -> Boolean
         if S has SETCAT and $ has finiteAggregate
members : % -> List S if $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
parts : % -> List S if $ has finiteAggregate
sample : () -> %
size? : (%,NonNegativeInteger) -> Boolean
?=? : (%,%) -> Boolean if S has SETCAT
?~=? : (%,%) -> Boolean if S has SETCAT
```

These exports come from (p23) ConvertibleTo(S:Type):

```
convert : % -> InputForm if S has KONVERT INFORM
```

— category CLAGG Collection —

```
)abbrev category CLAGG Collection
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
```

```
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A collection is a homogeneous aggregate which can built from
++ list of members. The operation used to build the aggregate is
++ generically named \spadfun{construct}. However, each collection
++ provides its own special function with the same name as the
++ data type, except with an initial lower case letter, e.g.
++ \spadfun{list} for \spadtype{List},
++ \spadfun{flexibleArray} for \spadtype{FlexibleArray}, and so on.

Collection(S:Type): Category == HomogeneousAggregate(S) with
   construct: List S -> %
     ++ \axiom{construct(x,y,...,z)} returns the collection of elements
     ++ \axiom{x,y,...,z} ordered as given. Equivalently written as
     ++ \axiom{[x,y,...,z]$D}, where
     ++ D is the domain. D may be omitted for those of type List.
   find: (S->Boolean, %) -> Union(S, "failed")
     ++ find(p,u) returns the first x in u such that \axiom{p(x)} is true,
     ++ and "failed" otherwise.
   if % has finiteAggregate then
      reduce: ((S,S)->S,%) -> S
        ++ reduce(f,u) reduces the binary operation f across u. For example,
        ++ if u is \axiom{[x,y,...,z]} then \axiom{reduce(f,u)}
        ++ returns \axiom{f(..f(f(x,y),...),z)}.
        ++ Note that if u has one element x, \axiom{reduce(f,u)} returns x.
        ++ Error: if u is empty.
        ++
        ++C )clear all
        ++X reduce(+,[C[i]*x**i for i in 1..5])

      reduce: ((S,S)->S,%,S) -> S
        ++ reduce(f,u,x) reduces the binary operation f across u, where x is
        ++ the identity operation of f.
        ++ Same as \axiom{reduce(f,u)} if u has 2 or more elements.
        ++ Returns \axiom{f(x,y)} if u has one element y,
        ++ x if u is empty.
        ++ For example, \axiom{reduce(+,u,0)} returns the
        ++ sum of the elements of u.
      remove: (S->Boolean,%) -> %
        ++ remove(p,u) returns a copy of u removing all elements x such that
        ++ \axiom{p(x)} is true.
        ++ Note that \axiom{remove(p,u) == [x for x in u | not p(x)]}.
      select: (S->Boolean,%) -> %
```

```
        ++ select(p,u) returns a copy of u containing only those elements
        ++ such \axiom{p(x)} is true.
        ++ Note that \axiom{select(p,u) == [x for x in u | p(x)]}.
      if S has SetCategory then
        reduce: ((S,S)->S,%,S,S) -> S
          ++ reduce(f,u,x,z) reduces the binary operation f across u,
          ++ stopping when an "absorbing element" z is encountered.
          ++ As for \axiom{reduce(f,u,x)}, x is the identity operation of f.
          ++ Same as \axiom{reduce(f,u,x)} when u contains no element z.
          ++ Thus the third argument x is returned when u is empty.
        remove: (S,%) -> %
          ++ remove(x,u) returns a copy of u with all
          ++ elements \axiom{y = x} removed.
          ++ Note that \axiom{remove(y,c) == [x for x in c | x ^= y]}.
        removeDuplicates: % -> %
          ++ removeDuplicates(u) returns a copy of u with all duplicates
          ++ removed.
  if S has ConvertibleTo InputForm then ConvertibleTo InputForm
add
  if % has finiteAggregate then
    #c                          == # parts c
    count(f:S -> Boolean, c:%) == _+/[1 for x in parts c | f x]
    any?(f, c)                  == _or/[f x for x in parts c]
    every?(f, c)          == _and/[f x for x in parts c]
    find(f:S -> Boolean, c:%) == find(f, parts c)
    reduce(f:(S,S)->S, x:%) == reduce(f, parts x)
    reduce(f:(S,S)->S, x:%, s:S) == reduce(f, parts x, s)
    remove(f:S->Boolean, x:%) ==
      construct remove(f, parts x)
    select(f:S->Boolean, x:%) ==
      construct select(f, parts x)

    if S has SetCategory then
      remove(s:S, x:%) == remove(y +-> y = s, x)
      reduce(f:(S,S)->S, x:%, s1:S, s2:S) == reduce(f, parts x, s1, s2)
      removeDuplicates(x) == construct removeDuplicates parts x
```

———-

— CLAGG.dotabb —

```
"CLAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=CLAGG"];
"CLAGG" -> "HOAGG"
```

———-

— CLAGG.dotfull —

```
"Collection(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=CLAGG"];
"Collection(a:Type)" -> "HomogeneousAggregate(a:Type)"

"Collection(a:SetCategory)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=CLAGG"];
"Collection(a:SetCategory)" -> "Collection(a:Type)"

"Collection(RecursivePolynomialCategory(Ring,OrderedAbelianMonoidSup(),OrderedSet())"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=CLAGG"];
"Collection(RecursivePolynomialCategory(Ring,OrderedAbelianMonoidSup(),OrderedSet())"
  -> "Collection(a:Type)"
```

———-

— CLAGG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"Collection(a:Type)" [color=lightblue];
"Collection(a:Type)" -> "HomogeneousAggregate(a:Type)"
"Collection(a:Type)" -> "ConvertibleTo(InputForm)"

"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "Aggregate()"

"Aggregate()" [color=lightblue];
"Aggregate()" -> "Type()"

"Type()" [color=lightblue];
"Type()" -> "Category"

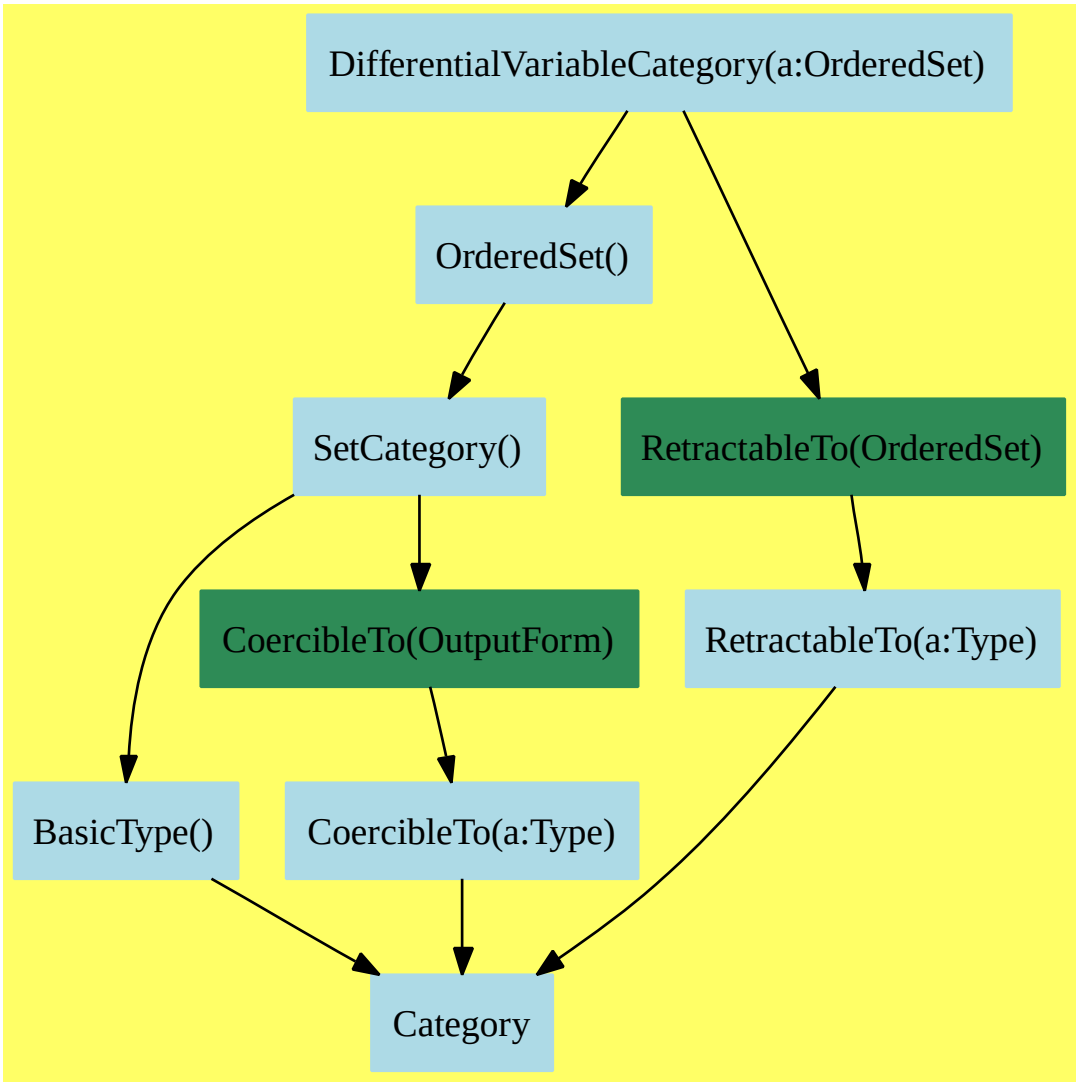
"ConvertibleTo(InputForm)" [color="#00EE00"];
"ConvertibleTo(InputForm)" -> "ConvertibleTo(a:Type)"

"ConvertibleTo(a:Type)" [color="#00EE00"];
"ConvertibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];
}
```

———-

5.0.71 DifferentialVariableCategory (DVARCAT)



— **DifferentialVariableCategory.input** —

```
)set break resume
)sys rm -f DifferentialVariableCategory.output
)spool DifferentialVariableCategory.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 1
)show DifferentialVariableCategory
--R DifferentialVariableCategory S: OrderedSet  is a category constructor
--R Abbreviation for DifferentialVariableCategory is DVARCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for DVARCAT
--R
--R------------------------------ Operations ------------------------------
--R ?<? : (%,%) -> Boolean                ?<=? : (%,%) -> Boolean
--R ?=? : (%,%) -> Boolean                ?>? : (%,%) -> Boolean
--R ?>=? : (%,%) -> Boolean               coerce : S -> %
--R coerce : % -> OutputForm              differentiate : % -> %
--R hash : % -> SingleInteger             latex : % -> String
--R max : (%,%) -> %                      min : (%,%) -> %
--R order : % -> NonNegativeInteger       retract : % -> S
--R variable : % -> S                     weight : % -> NonNegativeInteger
--R ?~=? : (%,%) -> Boolean
--R differentiate : (%,NonNegativeInteger) -> %
--R makeVariable : (S,NonNegativeInteger) -> %
--R retractIfCan : % -> Union(S,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **DifferentialVariableCategory.help** —

```
=====================================================================
DifferentialVariableCategory examples
=====================================================================

See Also:
o )show DifferentialVariableCategory
```

———-

See:

⇐ “OrderedSet” (ORDSET) 4.0.56 on page 216
⇐ “RetractableTo” (RETRACT) 2.0.18 on page 55

Exports:

coerce	differentiate	hash	latex	makeVariable
max	min	order	retract	retractIfCan
variable	weight	?~=?	?<?	?<=?
?=?	?>?	?>=?		

These are directly exported but not implemented:

```
makeVariable : (S,NonNegativeInteger) -> %
order : % -> NonNegativeInteger
variable : % -> S
```

These are implemented by this category:

```
coerce : S -> %
coerce : % -> OutputForm
differentiate : % -> %
differentiate : (%,NonNegativeInteger) -> %
retract : % -> S
retractIfCan : % -> Union(S,"failed")
weight : % -> NonNegativeInteger
?<? : (%,%) -> Boolean
?=? : (%,%) -> Boolean
```

These exports come from (p216) OrderedSet():

```
hash : % -> SingleInteger
latex : % -> String
max : (%,%) -> %
min : (%,%) -> %
?<=? : (%,%) -> Boolean
?>? : (%,%) -> Boolean
?>=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
```

These exports come from (p55) RetractableTo(S:OrderedSet):

— category DVARCAT DifferentialVariableCategory —

```
)abbrev category DVARCAT DifferentialVariableCategory
++ Author:  William Sit
++ Date Created: 19 July 1990
++ Date Last Updated: 13 September 1991
++ Basic Operations:
++ Related Constructors:DifferentialPolynomialCategory
++ See Also:OrderedDifferentialVariable,
++          SequentialDifferentialVariable,
```

```
++          DifferentialSparseMultivariatePolynomial.
++ AMS Classifications:12H05
++ Keywords: differential indeterminates, ranking, order, weight
++ References:Ritt, J.F. "Differential Algebra" (Dover, 1950).
++ Description:
++ \spadtype{DifferentialVariableCategory} constructs the
++ set of derivatives of a given set of
++ (ordinary) differential indeterminates.
++ If x,...,y is an ordered set of differential indeterminates,
++ and the prime notation is used for differentiation, then
++ the set of derivatives (including
++ zero-th order) of the differential indeterminates is
++ x,\spad{x'},\spad{x''},..., y,\spad{y'},\spad{y''},...
++ (Note that in the interpreter, the n-th derivative of y is displayed as
++ y with a subscript n.)  This set is
++ viewed as a set of algebraic indeterminates, totally ordered in a
++ way compatible with differentiation and the given order on the
++ differential indeterminates.  Such a total order is called a
++ ranking of the differential indeterminates.
++
++ A domain in this category is needed to construct a differential
++ polynomial domain.  Differential polynomials are ordered
++ by a ranking on the derivatives,  and by an order (extending the
++ ranking) on
++ on the set of differential monomials.  One may thus associate
++ a domain in this category with a ranking of the differential
++ indeterminates, just as one associates a domain in the category
++ \spadtype{OrderedAbelianMonoidSup} with an ordering of the set of
++ monomials in a set of algebraic indeterminates.  The ranking
++ is specified through the binary relation \spadfun{<}.
++ For example, one may define
++ one derivative to be less than another by lexicographically comparing
++ first the \spadfun{order}, then the given order of the differential
++ indeterminates appearing in the derivatives.  This is the default
++ implementation.
++
++ The notion of weight generalizes that of degree.  A
++ polynomial domain may be made into a graded ring
++ if a weight function is given on the set of indeterminates,
++ Very often, a grading is the first step in ordering the set of
++ monomials.  For differential polynomial domains, this
++ constructor provides a function \spadfun{weight}, which
++ allows the assignment of a non-negative number to each derivative of a
++ differential indeterminate.  For example, one may define
++ the weight of a derivative to be simply its \spadfun{order}
++ (this is the default assignment).
++ This weight function can then be extended to the set of
++ all differential polynomials, providing a graded ring structure.

DifferentialVariableCategory(S:OrderedSet): Category ==
```

```
 Join(OrderedSet, RetractableTo S) with
   -- Examples:
   -- v:=makeVariable('s,5)
   makeVariable  : (S, NonNegativeInteger) -> $
      ++ makeVariable(s, n) returns the n-th derivative of a
      ++ differential indeterminate s as an algebraic indeterminate.
      -- Example: makeVariable('s, 5)
   order         : $ -> NonNegativeInteger
      ++ order(v) returns n if v is the n-th derivative of any
      ++ differential indeterminate.
      -- Example: order(v)
   variable      : $ -> S
      ++ variable(v) returns s if v is any derivative of the differential
      ++ indeterminate s.
      -- Example: variable(v)
         --  default implementation using above primitives --

   weight        : $ -> NonNegativeInteger
      ++ weight(v) returns the weight of the derivative v.
      -- Example: weight(v)
   differentiate : $ -> $
      ++ differentiate(v) returns the derivative of v.
      -- Example: differentiate(v)
   differentiate : ($, NonNegativeInteger) -> $
      ++ differentiate(v, n) returns the n-th derivative of v.
      -- Example: differentiate(v,2)
   coerce        : S  -> $
      ++ coerce(s) returns s, viewed as the zero-th order derivative of s.
      -- Example: coerce('s); differentiate(%,5)
add
   import NumberFormats

   coerce (s:S):$ == makeVariable(s, 0)

   differentiate v     == differentiate(v, 1)

   differentiate(v, n) == makeVariable(variable v, n + order v)

   retractIfCan v == (zero?(order v) => variable v; "failed")

   v = u      == (variable v = variable u) and (order v = order u)

   coerce(v:$):OutputForm ==
     a := variable(v)::OutputForm
     zero?(nn := order v) => a
     sub(a, outputForm nn)

   retract v ==
     zero?(order v) => variable v
     error "Not retractable"
```

```
    v < u ==
      -- the ranking below is orderly, and is the default --
      order v = order u => variable v < variable u
      order v < order u

    weight v == order v
      --  the default weight is just the order
```

———-

— **DVARCAT.dotabb** —

```
"DVARCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=DVARCAT"];
"DVARCAT" -> "ORDSET"
"DVARCAT" -> "RETRACT"
```

———-

— **DVARCAT.dotfull** —

```
"DifferentialVariableCategory(a:OrderedSet)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=DVARCAT"];
"DifferentialVariableCategory(a:OrderedSet)" -> "OrderedSet()"
"DifferentialVariableCategory(a:OrderedSet)" -> "RetractableTo(OrderedSet)"
```

———-

— **DVARCAT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"DifferentialVariableCategory(a:OrderedSet)" [color=lightblue];
"DifferentialVariableCategory(a:OrderedSet)" -> "OrderedSet()"
"DifferentialVariableCategory(a:OrderedSet)" -> "RetractableTo(OrderedSet)"

"RetractableTo(OrderedSet)" [color=seagreen];
"RetractableTo(OrderedSet)" -> "RetractableTo(a:Type)"

"RetractableTo(a:Type)" [color=lightblue];
"RetractableTo(a:Type)" -> "Category"
```

```
"OrderedSet()" [color=lightblue];
"OrderedSet()" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

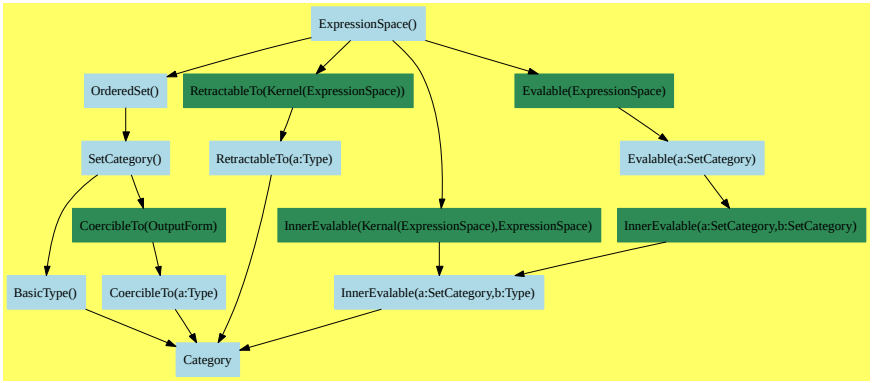
"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];
}
```

———-

5.0.72 ExpressionSpace (ES)



— ExpressionSpace.input —

```
)set break resume
)sys rm -f ExpressionSpace.output
)spool ExpressionSpace.output
)set message test on
```

```
)set message auto off
)clear all

--S 1 of 1
)show ExpressionSpace
--R ExpressionSpace  is a category constructor
--R Abbreviation for ExpressionSpace is ES
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for ES
--R
--R------------------------------ Operations --------------------------------
--R ?<? : (%,%) -> Boolean                ?<=? : (%,%) -> Boolean
--R ?=? : (%,%) -> Boolean                ?>? : (%,%) -> Boolean
--R ?>=? : (%,%) -> Boolean               belong? : BasicOperator -> Boolean
--R box : List % -> %                     box : % -> %
--R coerce : Kernel % -> %                coerce : % -> OutputForm
--R distribute : (%,%) -> %               distribute : % -> %
--R elt : (BasicOperator,%,%) -> %        elt : (BasicOperator,%) -> %
--R eval : (%,List %,List %) -> %         eval : (%,%,%) -> %
--R eval : (%,Equation %) -> %            eval : (%,List Equation %) -> %
--R eval : (%,Kernel %,%) -> %            freeOf? : (%,Symbol) -> Boolean
--R freeOf? : (%,%) -> Boolean            hash : % -> SingleInteger
--R height : % -> NonNegativeInteger      is? : (%,Symbol) -> Boolean
--R kernel : (BasicOperator,%) -> %       kernels : % -> List Kernel %
--R latex : % -> String                   map : ((% -> %),Kernel %) -> %
--R max : (%,%) -> %                      min : (%,%) -> %
--R paren : List % -> %                   paren : % -> %
--R retract : % -> Kernel %               subst : (%,Equation %) -> %
--R tower : % -> List Kernel %            ?~=? : (%,%) -> Boolean
--R definingPolynomial : % -> % if $ has RING
--R elt : (BasicOperator,List %) -> %
--R elt : (BasicOperator,%,%,%,%) -> %
--R elt : (BasicOperator,%,%,%) -> %
--R eval : (%,BasicOperator,(% -> %)) -> %
--R eval : (%,BasicOperator,(List % -> %)) -> %
--R eval : (%,List BasicOperator,List (List % -> %)) -> %
--R eval : (%,List BasicOperator,List (% -> %)) -> %
--R eval : (%,Symbol,(% -> %)) -> %
--R eval : (%,Symbol,(List % -> %)) -> %
--R eval : (%,List Symbol,List (List % -> %)) -> %
--R eval : (%,List Symbol,List (% -> %)) -> %
--R eval : (%,List Kernel %,List %) -> %
--R even? : % -> Boolean if $ has RETRACT INT
--R is? : (%,BasicOperator) -> Boolean
--R kernel : (BasicOperator,List %) -> %
--R mainKernel : % -> Union(Kernel %,"failed")
--R minPoly : Kernel % -> SparseUnivariatePolynomial % if $ has RING
--R odd? : % -> Boolean if $ has RETRACT INT
--R operator : BasicOperator -> BasicOperator
--R operators : % -> List BasicOperator
```

```
--R retractIfCan : % -> Union(Kernel %,"failed")
--R subst : (%,List Kernel %,List %) -> %
--R subst : (%,List Equation %) -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **ExpressionSpace.help** —

```
=====================================================================
ExpressionSpace examples
=====================================================================

See Also:
o )show ExpressionSpace
```

———-

See:

⇒ “FunctionSpace” (FS) 17.0.195 on page 1336
⇐ “Evalable” (EVALAB) 3.0.25 on page 80
⇐ “InnerEvalable” (IEVALAB) 2.0.12 on page 35
⇐ “OrderedSet” (ORDSET) 4.0.56 on page 216
⇐ “RetractableTo” (RETRACT) 2.0.18 on page 55

Exports:

belong?	box	coerce	definingPolynomial
distribute	elt	eval	even?
freeOf?	hash	height	is?
kernel	kernels	latex	mainKernel
map	max	min	minPoly
odd?	operator	operators	paren
retract	retractIfCan	subst	tower
?<?	?<=?	?=?	?>?
?>=?	?~=?		

These are directly exported but not implemented:

```
definingPolynomial : % -> % if $ has RING
eval : (%,List Symbol,List (List % -> %)) -> %
eval : (%,List Kernel %,List %) -> %
eval : (%,List %,List %) -> %
```

```
eval : (%,%,%) -> %
eval : (%,Equation %) -> %
eval : (%,Kernel %,%) -> %
kernels : % -> List Kernel %
minPoly : Kernel % -> SparseUnivariatePolynomial % if $ has RING
subst : (%,List Kernel %,List %) -> %
```

These are implemented by this category:

```
belong? : BasicOperator -> Boolean
box : List % -> %
box : % -> %
distribute : % -> %
distribute : (%,%) -> %
elt : (BasicOperator,%,%,%,%) -> %
elt : (BasicOperator,%,%,%) -> %
elt : (BasicOperator,%,%) -> %
elt : (BasicOperator,%) -> %
elt : (BasicOperator,List %) -> %
eval : (%,List Equation %) -> %
eval : (%,Symbol,(% -> %)) -> %
eval : (%,Symbol,(List % -> %)) -> %
eval : (%,BasicOperator,(% -> %)) -> %
eval : (%,BasicOperator,(List % -> %)) -> %
eval : (%,List BasicOperator,List (% -> %)) -> %
eval : (%,List Symbol,List (% -> %)) -> %
eval : (%,List BasicOperator,List (List % -> %)) -> %
even? : % -> Boolean if $ has RETRACT INT
freeOf? : (%,Symbol) -> Boolean
freeOf? : (%,%) -> Boolean
height : % -> NonNegativeInteger
is? : (%,BasicOperator) -> Boolean
is? : (%,Symbol) -> Boolean
kernel : (BasicOperator,%) -> %
kernel : (BasicOperator,List %) -> %
mainKernel : % -> Union(Kernel %,"failed")
map : ((% -> %),Kernel %) -> %
odd? : % -> Boolean if $ has RETRACT INT
operator : BasicOperator -> BasicOperator
operators : % -> List BasicOperator
paren : % -> %
paren : List % -> %
retract : % -> Kernel %
retractIfCan : % -> Union(Kernel %,"failed")
subst : (%,Equation %) -> %
subst : (%,List Equation %) -> %
tower : % -> List Kernel %
```

These exports come from (p216) OrderedSet():

```
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
max : (%,%) -> %
min : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?<? : (%,%) -> Boolean
?>? : (%,%) -> Boolean
?>=? : (%,%) -> Boolean
?<=? : (%,%) -> Boolean
```

These exports come from (p55) RetractableTo(a:Type):

```
coerce : Kernel % -> %
```

These exports come from (p35) InnerEvalable(a:SetCategory,b:Type):

These exports come from (p80) Evalable(a:SetCategory):

— category ES ExpressionSpace —

```
)abbrev category ES ExpressionSpace
++ Category for domains on which operators can be applied
++ Author: Manuel Bronstein
++ Date Created: 22 March 1988
++ Date Last Updated: 27 May 1994
++ Keywords: operator, kernel, expression, space.
++ Description:
++ An expression space is a set which is closed under certain operators;

ExpressionSpace(): Category == Defn where
  N   ==> NonNegativeInteger
  K   ==> Kernel %
  OP  ==> BasicOperator
  SY  ==> Symbol
  PAREN  ==> "%paren"::SY
  BOX    ==> "%box"::SY
  DUMMYVAR ==> "%dummyVar"

  Defn ==> Join(OrderedSet, RetractableTo K,
                InnerEvalable(K, %), Evalable %) with
    elt          : (OP, %) -> %
      ++ elt(op,x) or op(x) applies the unary operator op to x.
```

```
elt          : (OP, %, %) -> %
  ++ elt(op,x,y) or op(x, y) applies the binary operator op to x and y.
elt          : (OP, %, %, %) -> %
  ++ elt(op,x,y,z) or op(x, y, z) applies the ternary operator op
  ++ to x, y and z.
elt          : (OP, %, %, %, %) -> %
  ++ elt(op,x,y,z,t) or op(x, y, z, t) applies the 4-ary operator op
  ++ to x, y, z and t.
elt          : (OP, List %) -> %
  ++ elt(op,[x1,...,xn]) or op([x1,...,xn]) applies the n-ary operator
  ++ op to x1,...,xn.
subst        : (%, Equation %) -> %
  ++ subst(f, k = g) replaces the kernel k by g formally in f.
subst        : (%, List Equation %) -> %
  ++ subst(f, [k1 = g1,...,kn = gn]) replaces the kernels k1,...,kn
  ++ by g1,...,gn formally in f.
subst        : (%, List K, List %) -> %
  ++ subst(f, [k1...,kn], [g1,...,gn]) replaces the kernels k1,...,kn
  ++ by g1,...,gn formally in f.
box          : % -> %
  ++ box(f) returns f with a 'box' around it that prevents f from
  ++ being evaluated when operators are applied to it. For example,
  ++ \spad{log(1)} returns 0, but \spad{log(box 1)}
  ++ returns the formal kernel log(1).
box          : List % -> %
  ++ box([f1,...,fn]) returns \spad{(f1,...,fn)} with a 'box'
  ++ around them that
  ++ prevents the fi from being evaluated when operators are applied to
  ++ them, and makes them applicable to a unary operator. For example,
  ++ \spad{atan(box [x, 2])} returns the formal kernel \spad{atan(x, 2)}.
paren        : % -> %
  ++ paren(f) returns (f). This prevents f from
  ++ being evaluated when operators are applied to it. For example,
  ++ \spad{log(1)} returns 0, but \spad{log(paren 1)} returns the
  ++ formal kernel log((1)).
paren        : List % -> %
  ++ paren([f1,...,fn]) returns \spad{(f1,...,fn)}. This
  ++ prevents the fi from being evaluated when operators are applied to
  ++ them, and makes them applicable to a unary operator. For example,
  ++ \spad{atan(paren [x, 2])} returns the formal
  ++ kernel \spad{atan((x, 2))}.
distribute   : % -> %
  ++ distribute(f) expands all the kernels in f that are
  ++ formally enclosed by a \spadfunFrom{box}{ExpressionSpace}
  ++ or \spadfunFrom{paren}{ExpressionSpace} expression.
distribute   : (%, %) -> %
  ++ distribute(f, g) expands all the kernels in f that contain g in their
  ++ arguments and that are formally
  ++ enclosed by a \spadfunFrom{box}{ExpressionSpace}
  ++ or a \spadfunFrom{paren}{ExpressionSpace} expression.
```

```
height       : %  -> N
  ++ height(f) returns the highest nesting level appearing in f.
  ++ Constants have height 0. Symbols have height 1. For any
  ++ operator op and expressions f1,...,fn, \spad{op(f1,...,fn)} has
  ++ height equal to \spad{1 + max(height(f1),...,height(fn))}.
mainKernel   : %  -> Union(K, "failed")
  ++ mainKernel(f) returns a kernel of f with maximum nesting level, or
  ++ if f has no kernels (i.e. f is a constant).
kernels      : %  -> List K
  ++ kernels(f) returns the list of all the top-level kernels
  ++ appearing in f, but not the ones appearing in the arguments
  ++ of the top-level kernels.
tower        : %  -> List K
  ++ tower(f) returns all the kernels appearing in f, no matter
  ++ what their levels are.
operators    : %  -> List OP
  ++ operators(f) returns all the basic operators appearing in f,
  ++ no matter what their levels are.
operator     : OP -> OP
  ++ operator(op) returns a copy of op with the domain-dependent
  ++ properties appropriate for %.
belong?      : OP -> Boolean
  ++ belong?(op) tests if % accepts op as applicable to its
  ++ elements.
is?          : (%, OP)     -> Boolean
  ++ is?(x, op) tests if x is a kernel and is its operator is op.
is?          : (%, SY) -> Boolean
  ++ is?(x, s) tests if x is a kernel and is the name of its
  ++ operator is s.
kernel       : (OP, %) -> %
  ++ kernel(op, x) constructs op(x) without evaluating it.
kernel       : (OP, List %) -> %
  ++ kernel(op, [f1,...,fn]) constructs \spad{op(f1,...,fn)} without
  ++ evaluating it.
map          : (% -> %, K) -> %
  ++ map(f, k) returns \spad{op(f(x1),...,f(xn))} where
  ++ \spad{k = op(x1,...,xn)}.
freeOf?      : (%, %)  -> Boolean
  ++ freeOf?(x, y) tests if x does not contain any occurrence of y,
  ++ where y is a single kernel.
freeOf?      : (%, SY) -> Boolean
  ++ freeOf?(x, s) tests if x does not contain any operator
  ++ whose name is s.
eval         : (%, List SY, List(% -> %)) -> %
  ++ eval(x, [s1,...,sm], [f1,...,fm]) replaces
  ++ every \spad{si(a)} in x by \spad{fi(a)} for any \spad{a}.
eval         : (%, List SY, List(List % -> %)) -> %
  ++ eval(x, [s1,...,sm], [f1,...,fm]) replaces
  ++ every \spad{si(a1,...,an)} in x by
  ++ \spad{fi(a1,...,an)} for any \spad{a1},...,\spad{an}.
```

```
    eval         : (%, SY, List % -> %) -> %
      ++ eval(x, s, f) replaces every \spad{s(a1,..,am)} in x
      ++ by \spad{f(a1,..,am)} for any \spad{a1},...,\spad{am}.
    eval         : (%, SY, % -> %) -> %
      ++ eval(x, s, f) replaces every \spad{s(a)} in x by \spad{f(a)}
      ++ for any \spad{a}.
    eval         : (%, List OP, List(% -> %)) -> %
      ++ eval(x, [s1,...,sm], [f1,...,fm]) replaces
      ++ every \spad{si(a)} in x by \spad{fi(a)} for any \spad{a}.
    eval         : (%, List OP, List(List % -> %)) -> %
      ++ eval(x, [s1,...,sm], [f1,...,fm]) replaces
      ++ every \spad{si(a1,...,an)} in x by
      ++ \spad{fi(a1,...,an)} for any \spad{a1},...,\spad{an}.
    eval         : (%, OP, List % -> %) -> %
      ++ eval(x, s, f) replaces every \spad{s(a1,..,am)} in x
      ++ by \spad{f(a1,..,am)} for any \spad{a1},...,\spad{am}.
    eval         : (%, OP, % -> %) -> %
      ++ eval(x, s, f) replaces every \spad{s(a)} in x by \spad{f(a)}
      ++ for any \spad{a}.
    if % has Ring then
      minPoly: K -> SparseUnivariatePolynomial %
        ++ minPoly(k) returns p such that \spad{p(k) = 0}.
      definingPolynomial: % -> %
        ++ definingPolynomial(x) returns an expression p such that
        ++ \spad{p(x) = 0}.
    if % has RetractableTo Integer then
      even?: % -> Boolean
        ++ even? x is true if x is an even integer.
      odd? : % -> Boolean
        ++ odd? x is true if x is an odd integer.

  add

-- the 7 functions not provided are:
--       kernels   minPoly   definingPolynomial
--       coerce:K -> %  eval:(%, List K, List %) -> %
--       subst:(%, List K, List %) -> %
--       eval:(%, List Symbol, List(List % -> %)) -> %

    allKernels: %      -> Set K
    listk     : %      -> List K
    allk      : List % -> Set K
    unwrap    : (List K, %) -> %
    okkernel  : (OP, List %) -> %
    mkKerLists: List Equation % -> Record(lstk: List K, lstv:List %)

    oppren := operator(PAREN)$CommonOperators()
    opbox  := operator(BOX)$CommonOperators()

    box(x:%)     == box [x]
```

```
paren(x:%)   == paren [x]
belong? op   == op = oppren or op = opbox
listk f      == parts allKernels f
tower f      == sort_! listk f
allk l       == reduce("union", [allKernels f for f in l], {})
operators f  == [operator k for k in listk f]
height f     == reduce("max", [height k for k in kernels f], 0)
freeOf?(x:%, s:SY)       == not member?(s, [name k for k in listk x])
distribute x == unwrap([k for k in listk x | is?(k, oppren)], x)
box(l:List %)                 == opbox l
paren(l:List %)               == oppren l
freeOf?(x:%, k:%)             == not member?(retract k, listk x)
kernel(op:OP, arg:%)          == kernel(op, [arg])
elt(op:OP, x:%)               == op [x]
elt(op:OP, x:%, y:%)          == op [x, y]
elt(op:OP, x:%, y:%, z:%)     == op [x, y, z]
elt(op:OP, x:%, y:%, z:%, t:%) == op [x, y, z, t]
eval(x:%, s:SY, f:List % -> %) == eval(x, [s], [f])
eval(x:%, s:OP, f:List % -> %) == eval(x, [name s], [f])
eval(x:%, s:SY, f:% -> %)     ==
  eval(x, [s], [(y:List %):% +-> f(first y)])
eval(x:%, s:OP, f:% -> %)     ==
  eval(x, [s], [(y:List %):% +-> f(first y)])
subst(x:%, e:Equation %)      == subst(x, [e])

eval(x:%, ls:List OP, lf:List(% -> %)) ==
  eval(x, ls, [y +-> f(first y) for f in lf]$List(List % -> %))

eval(x:%, ls:List SY, lf:List(% -> %)) ==
  eval(x, ls, [y +-> f(first y) for f in lf]$List(List % -> %))

eval(x:%, ls:List OP, lf:List(List % -> %)) ==
  eval(x, [name s for s in ls]$List(SY), lf)

map(fn, k) ==
  (l := [fn x for x in argument k]$List(%)) = argument k => k::%
  (operator k) l

operator op ==
  is?(op, PAREN) => oppren
  is?(op, BOX) => opbox
  error "Unknown operator"

mainKernel x ==
  empty?(l := kernels x) => "failed"
  n := height(k := first l)
  for kk in rest l repeat
    if height(kk) > n then
      n := height kk
      k := kk
```

```
      k

-- takes all the kernels except for the dummy variables, which are second
-- arguments of rootOf's, integrals, sums and products which appear only in
-- their first arguments
    allKernels f ==
      s := brace(l := kernels f)
      for k in l repeat
          t :=
              (u := property(operator k, DUMMYVAR)) case None =>
                  arg := argument k
                  s0  := remove_!(retract(second arg)@K, allKernels first arg)
                  arg := rest rest arg
                  n   := (u::None) pretend N
                  if n > 1 then arg := rest arg
                  union(s0, allk arg)
              allk argument k
          s := union(s, t)
      s

    kernel(op:OP, args:List %) ==
      not belong? op => error "Unknown operator"
      okkernel(op, args)

    okkernel(op, l) ==
      kernel(op, l, 1 + reduce("max", [height f for f in l], 0))$K :: %

    elt(op:OP, args:List %) ==
      not belong? op => error "Unknown operator"
      ((u := arity op) case N) and (#args ^= u::N)
                                    => error "Wrong number of arguments"
      (v := evaluate(op,args)$BasicOperatorFunctions1(%)) case % => v::%
      okkernel(op, args)

    retract f ==
      (k := mainKernel f) case "failed" => error "not a kernel"
      k::K::% ^= f => error "not a kernel"
      k::K

    retractIfCan f ==
      (k := mainKernel f) case "failed" => "failed"
      k::K::% ^= f => "failed"
      k

    is?(f:%, s:SY) ==
      (k := retractIfCan f) case "failed" => false
      is?(k::K, s)

    is?(f:%, op:OP) ==
      (k := retractIfCan f) case "failed" => false
```

```
  is?(k::K, op)

unwrap(l, x) ==
  for k in reverse_! l repeat
    x := eval(x, k, first argument k)
  x

distribute(x, y) ==
  ky := retract y
  unwrap([k for k in listk x |
          is?(k, "%paren"::SY) and member?(ky, listk(k::%))], x)

-- in case of conflicting substitutions e.g. [x = a, x = b],
-- the first one prevails.
-- this is not part of the semantics of the function, but just
-- a feature of this implementation.
eval(f:%, leq:List Equation %) ==
  rec := mkKerLists leq
  eval(f, rec.lstk, rec.lstv)

subst(f:%, leq:List Equation %) ==
  rec := mkKerLists leq
  subst(f, rec.lstk, rec.lstv)

mkKerLists leq ==
  lk := empty()$List(K)
  lv := empty()$List(%)
  for eq in leq repeat
    (k := retractIfCan(lhs eq)@Union(K, "failed")) case "failed" =>
                      error "left hand side must be a single kernel"
    if not member?(k::K, lk) then
      lk := concat(k::K, lk)
      lv := concat(rhs eq, lv)
  [lk, lv]

if % has RetractableTo Integer then
   intpred?: (%, Integer -> Boolean) -> Boolean

   even? x == intpred?(x, even?)
   odd? x  == intpred?(x, odd?)

   intpred?(x, pred?) ==
       (u := retractIfCan(x)@Union(Integer, "failed")) case Integer
              and pred?(u::Integer)
```

———-

— **ES.dotabb** —

```
"ES"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ES"];
"ES" -> "ORDSET"
"ES" -> "RETRACT"
"ES" -> "IEVALAB"
"ES" -> "EVALAB"
```

———-

— **ES.dotfull** —

```
"ExpressionSpace()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ES"];
"ExpressionSpace()" -> "OrderedSet()"
"ExpressionSpace()" -> "RetractableTo(Kernel(ExpressionSpace))"
"ExpressionSpace()" ->
    "InnerEvalable(Kernal(ExpressionSpace),ExpressionSpace)"
"ExpressionSpace()" -> "Evalable(ExpressionSpace)"
```

———-

— **ES.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"ExpressionSpace()" [color=lightblue];
"ExpressionSpace()" -> "OrderedSet()"
"ExpressionSpace()" -> "RetractableTo(Kernel(ExpressionSpace))"
"ExpressionSpace()" ->
    "InnerEvalable(Kernal(ExpressionSpace),ExpressionSpace)"
"ExpressionSpace()" -> "Evalable(ExpressionSpace)"

"Evalable(ExpressionSpace)" [color=seagreen];
"Evalable(ExpressionSpace)" -> "Evalable(a:SetCategory)"

"Evalable(a:SetCategory)" [color=lightblue];
"Evalable(a:SetCategory)" -> "InnerEvalable(a:SetCategory,b:SetCategory)"

"InnerEvalable(Kernal(ExpressionSpace),ExpressionSpace)" [color=seagreen];
"InnerEvalable(Kernal(ExpressionSpace),ExpressionSpace)" ->
    "InnerEvalable(a:SetCategory,b:Type)"

"InnerEvalable(a:SetCategory,b:SetCategory)" [color=seagreen];
```

```
"InnerEvalable(a:SetCategory,b:SetCategory)" ->
    "InnerEvalable(a:SetCategory,b:Type)"

"InnerEvalable(a:SetCategory,b:Type)" [color=lightblue];
"InnerEvalable(a:SetCategory,b:Type)" -> "Category"

"RetractableTo(Kernel(ExpressionSpace))" [color=seagreen];
"RetractableTo(Kernel(ExpressionSpace))" -> "RetractableTo(a:Type)"

"RetractableTo(a:Type)" [color=lightblue];
"RetractableTo(a:Type)" -> "Category"

"OrderedSet()" [color=lightblue];
"OrderedSet()" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

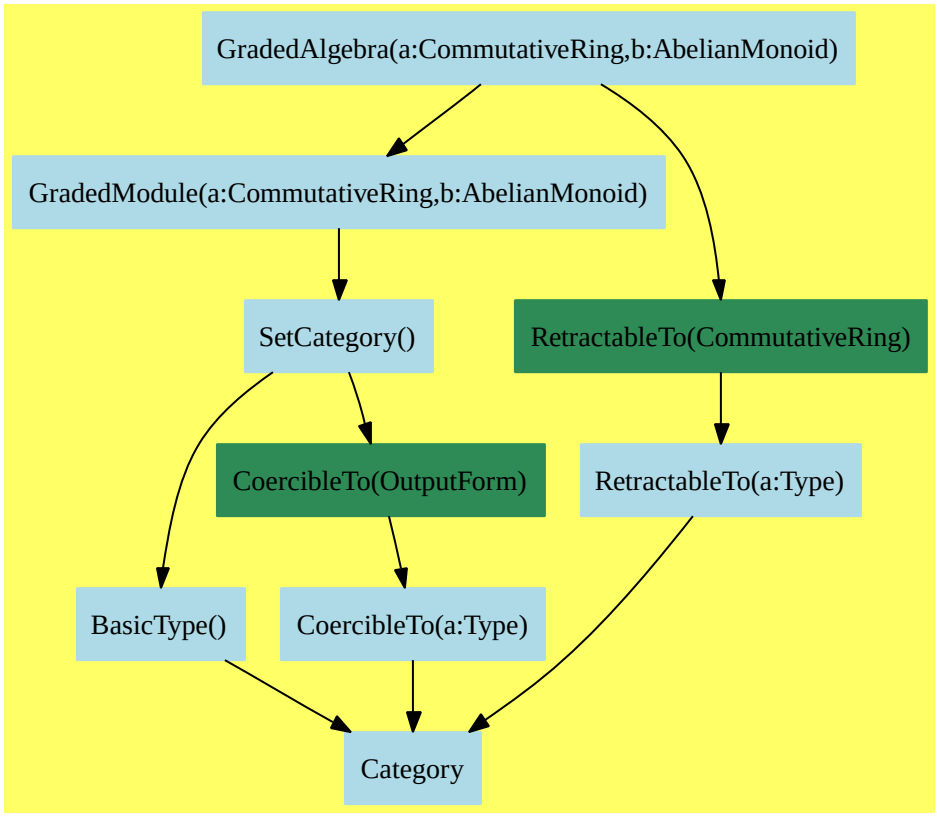
"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];
}
```

———-

5.0.73 GradedAlgebra (GRALG)



— **GradedAlgebra.input** —

```
)set break resume
)sys rm -f GradedAlgebra.output
)spool GradedAlgebra.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show GradedAlgebra
--R GradedAlgebra(R: CommutativeRing,E: AbelianMonoid)  is a category constructor
--R Abbreviation for GradedAlgebra is GRALG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for GRALG
--R
--R------------------------------ Operations --------------------------------
```

```
--R ?*? : (%,R) -> %                        ?*? : (R,%) -> %
--R ?+? : (%,%) -> %                        ?-? : (%,%) -> %
--R -? : % -> %                             ?=? : (%,%) -> Boolean
--R 1 : () -> %                             0 : () -> %
--R coerce : R -> %                         coerce : % -> OutputForm
--R degree : % -> E                         hash : % -> SingleInteger
--R latex : % -> String                     product : (%,%) -> %
--R retract : % -> R                        ?~=? : (%,%) -> Boolean
--R retractIfCan : % -> Union(R,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— GradedAlgebra.help —

```
=====================================================================
GradedAlgebra examples
=====================================================================

See Also:
o )show GradedAlgebra
```

———-

See:

⇐ “GradedModule” (GRMOD) 4.0.48 on page 176
⇐ “RetractableTo” (RETRACT) 2.0.18 on page 55

Exports:

0	1	coerce	degree	hash
latex	product	retract	retractIfCan	?~=?
?*?	?+?	?-?	-?	?=?

These are directly exported but not implemented:

```
product : (%,%) -> %
```

These are implemented by this category:

```
0 : () -> %
1 : () -> %
?*? : (%,R) -> %
?*? : (R,%) -> %
```

These exports come from (p176) GradedModule(R, E)
where R:CommutativeRing and E:AbelianMonoid:

```
coerce : % -> OutputForm
degree : % -> E
hash : % -> SingleInteger
latex : % -> String
?~=? : (%,%) -> Boolean
?=? : (%,%) -> Boolean
?-? : (%,%) -> %
-? : % -> %
?+? : (%,%) -> %
```

These exports come from (p55) RetractableTo(R:CommutativeRing):

```
coerce : R -> %
retract : % -> R
retractIfCan : % -> Union(R,"failed")
```

— category GRALG GradedAlgebra —

```
)abbrev category GRALG GradedAlgebra
++ Author: Stephen M. Watt
++ Date Created: May 20, 1991
++ Date Last Updated: May 20, 1991
++ Basic Operations: +, *, degree
++ Related Domains: CartesianTensor(n,dim,R)
++ Also See:
++ AMS Classifications:
++ Keywords: graded module, tensor, multi-linear algebra
++ Examples:
++ References: Encyclopedic Dictionary of Mathematics, MIT Press, 1977
++ Description:
++ GradedAlgebra(R,E) denotes ‘‘E-graded R-algebra’’.
++ A graded algebra is a graded module together with a degree preserving
++ R-linear map, called the product.
++
++ The name ‘‘product’’ is written out in full so inner and outer products
++ with the same mapping type can be distinguished by name.

GradedAlgebra(R: CommutativeRing, E: AbelianMonoid): Category ==
    Join(GradedModule(R, E),RetractableTo(R)) with
        1: constant -> %
            ++ 1 is the identity for \spad{product}.
        product: (%, %) -> %
            ++ product(a,b) is the degree-preserving R-linear product:
            ++
            ++   \spad{degree product(a,b) = degree a + degree b}
```

```
            ++   \spad{product(a1+a2,b) = product(a1,b) + product(a2,b)}
            ++   \spad{product(a,b1+b2) = product(a,b1) + product(a,b2)}
            ++   \spad{product(r*a,b) = product(a,r*b) = r*product(a,b)}
            ++   \spad{product(a,product(b,c)) = product(product(a,b),c)}
  add
        if not (R is %) then
            0: % == (0$R)::%
            1: % == 1$R::%
            (r: R)*(x: %) == product(r::%, x)
            (x: %)*(r: R) == product(x, r::%)
```

— GRALG.dotabb —

```
"GRALG"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=GRALG"];
"GRALG" -> "GRMOD"
"GRALG" -> "RETRACT"
```

— GRALG.dotfull —

```
"GradedAlgebra(a:CommutativeRing,b:AbelianMonoid)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=GRALG"];
"GradedAlgebra(a:CommutativeRing,b:AbelianMonoid)" ->
    "GradedModule(a:CommutativeRing,b:AbelianMonoid)"
"GradedAlgebra(a:CommutativeRing,b:AbelianMonoid)" ->
    "RetractableTo(CommutativeRing)"
```

— GRALG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"GradedAlgebra(a:CommutativeRing,b:AbelianMonoid)" [color=lightblue];
"GradedAlgebra(a:CommutativeRing,b:AbelianMonoid)" ->
    "GradedModule(a:CommutativeRing,b:AbelianMonoid)"
"GradedAlgebra(a:CommutativeRing,b:AbelianMonoid)" ->
    "RetractableTo(CommutativeRing)"
```

```
"RetractableTo(CommutativeRing)" [color=seagreen];
"RetractableTo(CommutativeRing)" -> "RetractableTo(a:Type)"

"RetractableTo(a:Type)" [color=lightblue];
"RetractableTo(a:Type)" -> "Category"

"GradedModule(a:CommutativeRing,b:AbelianMonoid)" [color=lightblue];
"GradedModule(a:CommutativeRing,b:AbelianMonoid)" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

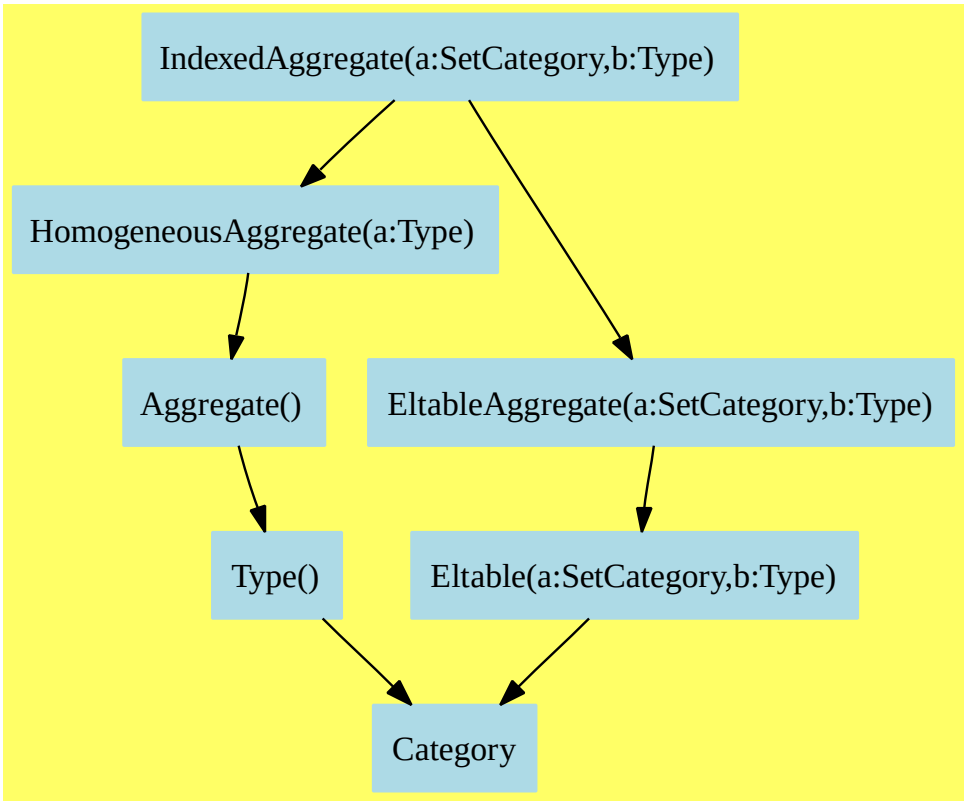
"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];

}
```

———————-

5.0.74 IndexedAggregate (IXAGG)



— **IndexedAggregate.input** —

```
)set break resume
)sys rm -f IndexedAggregate.output
)spool IndexedAggregate.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show IndexedAggregate
--R IndexedAggregate(Index: SetCategory,Entry: Type)  is a category constructor
--R Abbreviation for IndexedAggregate is IXAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for IXAGG
--R
--R------------------------------ Operations ------------------------------
--R copy : % -> %                          ?.? : (%,Index) -> Entry
```

```
--R elt : (%,Index,Entry) -> Entry        empty : () -> %
--R empty? : % -> Boolean                 entries : % -> List Entry
--R eq? : (%,%) -> Boolean                index? : (Index,%) -> Boolean
--R indices : % -> List Index             map : ((Entry -> Entry),%) -> %
--R qelt : (%,Index) -> Entry             sample : () -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if Entry has SETCAT
--R any? : ((Entry -> Boolean),%) -> Boolean if $ has finiteAggregate
--R coerce : % -> OutputForm if Entry has SETCAT
--R count : (Entry,%) -> NonNegativeInteger if Entry has SETCAT and $ has finiteAggregate
--R count : ((Entry -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R entry? : (Entry,%) -> Boolean if $ has finiteAggregate and Entry has SETCAT
--R eval : (%,List Entry,List Entry) -> % if Entry has EVALAB Entry and Entry has SETCAT
--R eval : (%,Entry,Entry) -> % if Entry has EVALAB Entry and Entry has SETCAT
--R eval : (%,Equation Entry) -> % if Entry has EVALAB Entry and Entry has SETCAT
--R eval : (%,List Equation Entry) -> % if Entry has EVALAB Entry and Entry has SETCAT
--R every? : ((Entry -> Boolean),%) -> Boolean if $ has finiteAggregate
--R fill! : (%,Entry) -> % if $ has shallowlyMutable
--R first : % -> Entry if Index has ORDSET
--R hash : % -> SingleInteger if Entry has SETCAT
--R latex : % -> String if Entry has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((Entry -> Entry),%) -> % if $ has shallowlyMutable
--R maxIndex : % -> Index if Index has ORDSET
--R member? : (Entry,%) -> Boolean if Entry has SETCAT and $ has finiteAggregate
--R members : % -> List Entry if $ has finiteAggregate
--R minIndex : % -> Index if Index has ORDSET
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List Entry if $ has finiteAggregate
--R qsetelt! : (%,Index,Entry) -> Entry if $ has shallowlyMutable
--R setelt : (%,Index,Entry) -> Entry if $ has shallowlyMutable
--R size? : (%,NonNegativeInteger) -> Boolean
--R swap! : (%,Index,Index) -> Void if $ has shallowlyMutable
--R ?~=? : (%,%) -> Boolean if Entry has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— IndexedAggregate.help —

```
=====================================================================
IndexedAggregate examples
=====================================================================

See Also:
```

```
o )show IndexedAggregate
```

See:

⇒ "DirectProductCategory" (DIRPCAT) 12.0.160 on page 1001
⇒ "LinearAggregate" (LNAGG) 6.0.87 on page 404
⇒ "TableAggregate" (TBAGG) 9.0.132 on page 797
⇐ "EltableAggregate" (ELTAGG) 3.0.24 on page 76
⇐ "HomogeneousAggregate" (HOAGG) 4.0.49 on page 181

Exports:

any?	coerce	copy	count	elt
empty	empty?	entries	entry?	eq?
eval	every?	fill!	first	hash
index?	indices	latex	less?	map
map!	maxIndex	member?	members	minIndex
more?	parts	qelt	qsetelt!	sample
setelt	size?	swap!	?.?	?~=?
#?	?=?			

Attributes exported:

- **nil**

Attributes Used:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.

These are directly exported but not implemented:

```
index? : (Index,%) -> Boolean
indices : % -> List Index
```

These are implemented by this category:

```
elt : (%,Index,Entry) -> Entry
entries : % -> List Entry
entry? : (Entry,%) -> Boolean
          if $ has finiteAggregate and Entry has SETCAT
fill! : (%,Entry) -> % if $ has shallowlyMutable
```

```
first : % -> Entry if Index has ORDSET
map : ((Entry -> Entry),%) -> %
map! : ((Entry -> Entry),%) -> % if $ has shallowlyMutable
maxIndex : % -> Index if Index has ORDSET
minIndex : % -> Index if Index has ORDSET
swap! : (%,Index,Index) -> Void if $ has shallowlyMutable
```

These exports come from (p181) HomogeneousAggregate(Entry:Type):

```
any? : ((Entry -> Boolean),%) -> Boolean
         if $ has finiteAggregate
coerce : % -> OutputForm if Entry has SETCAT
copy : % -> %
count : (Entry,%) -> NonNegativeInteger
         if Entry has SETCAT and $ has finiteAggregate
count : ((Entry -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List Entry,List Entry) -> %
      if Entry has EVALAB Entry and Entry has SETCAT
eval : (%,Entry,Entry) -> %
      if Entry has EVALAB Entry and Entry has SETCAT
eval : (%,Equation Entry) -> %
      if Entry has EVALAB Entry and Entry has SETCAT
eval : (%,List Equation Entry) -> %
      if Entry has EVALAB Entry and Entry has SETCAT
every? : ((Entry -> Boolean),%) -> Boolean
      if $ has finiteAggregate
hash : % -> SingleInteger if Entry has SETCAT
latex : % -> String if Entry has SETCAT
less? : (%,NonNegativeInteger) -> Boolean
member? : (Entry,%) -> Boolean
          if Entry has SETCAT and $ has finiteAggregate
members : % -> List Entry if $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
parts : % -> List Entry if $ has finiteAggregate
sample : () -> %
size? : (%,NonNegativeInteger) -> Boolean
?~=? : (%,%) -> Boolean if Entry has SETCAT
#? : % -> NonNegativeInteger if $ has finiteAggregate
?=? : (%,%) -> Boolean if Entry has SETCAT
```

These exports come from (p76) EltableAggregate(Index:SetCategory,Entry:Type):

```
qelt : (%,Index) -> Entry
qsetelt! : (%,Index,Entry) -> Entry if $ has shallowlyMutable
setelt : (%,Index,Entry) -> Entry if $ has shallowlyMutable
?.? : (%,Index) -> Entry
```

— category IXAGG IndexedAggregate —

```
)abbrev category IXAGG IndexedAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ An indexed aggregate is a many-to-one mapping of indices to entries.
++ For example, a one-dimensional-array is an indexed aggregate where
++ the index is an integer.  Also, a table is an indexed aggregate
++ where the indices and entries may have any type.

IndexedAggregate(Index: SetCategory, Entry: Type): Category ==
  Join(HomogeneousAggregate(Entry), EltableAggregate(Index, Entry)) with
   entries: % -> List Entry
      ++ entries(u) returns a list of all the entries of aggregate u
      ++ in no assumed order.
      -- to become entries: % -> Entry* and
      -- entries: % -> Iterator(Entry,Entry)
   index?: (Index,%) -> Boolean
      ++ index?(i,u) tests if i is an index of aggregate u.
   indices: % -> List Index
      ++ indices(u) returns a list of indices of aggregate u in no
      ++ particular order. to become indices:
      --  % -> Index* and indices: % -> Iterator(Index,Index).
-- map: ((Entry,Entry)->Entry,%,%,Entry) -> %
--    ++ exists c = map(f,a,b,x), i:Index where
--    ++    c.i = f(a(i,x),b(i,x)) | index?(i,a) or index?(i,b)
   if Entry has SetCategory and % has finiteAggregate then
      entry?: (Entry,%) -> Boolean
        ++ entry?(x,u) tests if x equals \axiom{u . i} for some index i.
   if Index has OrderedSet then
      maxIndex: % -> Index
        ++ maxIndex(u) returns the maximum index i of aggregate u.
        ++ Note that in general,
        ++ \axiom{maxIndex(u) = reduce(max,[i for i in indices u])};
        ++ if u is a list, \axiom{maxIndex(u) = #u}.
      minIndex: % -> Index
        ++ minIndex(u) returns the minimum index i of aggregate u.
        ++ Note that in general,
        ++ \axiom{minIndex(a) = reduce(min,[i for i in indices a])};
        ++ for lists, \axiom{minIndex(a) = 1}.
      first   : % -> Entry
```

```
      ++ first(u) returns the first element x of u.
      ++ Note that for collections, \axiom{first([x,y,...,z]) = x}.
      ++ Error: if u is empty.

  if % has shallowlyMutable then
     fill_!: (%,Entry) -> %
       ++ fill!(u,x) replaces each entry in aggregate u by x.
       ++ The modified u is returned as value.
     swap_!: (%,Index,Index) -> Void
       ++ swap!(u,i,j) interchanges elements i and j of aggregate u.
       ++ No meaningful value is returned.
add
 elt(a, i, x) == (index?(i, a) => qelt(a, i); x)

 if % has finiteAggregate then
   entries x == parts x
   if Entry has SetCategory then
     entry?(x, a) == member?(x, a)

 if Index has OrderedSet then
   maxIndex a == "max"/indices(a)
   minIndex a == "min"/indices(a)
   first a    == a minIndex a

 if % has shallowlyMutable then
   map(f, a) == map_!(f, copy a)

   map_!(f, a) ==
     for i in indices a repeat qsetelt_!(a, i, f qelt(a, i))
     a

   fill_!(a, x) ==
     for i in indices a repeat qsetelt_!(a, i, x)
     a

   swap_!(a, i, j) ==
     t := a.i
     qsetelt_!(a, i, a.j)
     qsetelt_!(a, j, t)
     void
```

———-

— IXAGG.dotabb —

```
"IXAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=IXAGG"];
"IXAGG" -> "HOAGG"
"IXAGG" -> "ELTAGG"
```

———-

— IXAGG.dotfull —

```
"IndexedAggregate(a:SetCategory,b:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=IXAGG"];
"IndexedAggregate(a:SetCategory,b:Type)" ->
    "HomogeneousAggregate(a:Type)"
"IndexedAggregate(a:SetCategory,b:Type)" ->
    "EltableAggregate(a:SetCategory,b:Type)"

"IndexedAggregate(a:SetCategory,b:SetCategory)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=IXAGG"];
"IndexedAggregate(a:SetCategory,b:SetCategory)" ->
    "IndexedAggregate(a:SetCategory,b:Type)"

"IndexedAggregate(b:Integer,a:Type)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=IXAGG"];
"IndexedAggregate(b:Integer,a:Type)" ->
    "IndexedAggregate(a:SetCategory,b:Type)"
```

———-

— IXAGG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"IndexedAggregate(a:SetCategory,b:Type)" [color=lightblue];
"IndexedAggregate(a:SetCategory,b:Type)" ->
    "HomogeneousAggregate(a:Type)"
"IndexedAggregate(a:SetCategory,b:Type)" ->
    "EltableAggregate(a:SetCategory,b:Type)"

"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "Aggregate()"

"EltableAggregate(a:SetCategory,b:Type)" [color=lightblue];
"EltableAggregate(a:SetCategory,b:Type)" -> "Eltable(a:SetCategory,b:Type)"

"Eltable(a:SetCategory,b:Type)" [color=lightblue];
"Eltable(a:SetCategory,b:Type)" -> "Category"
```

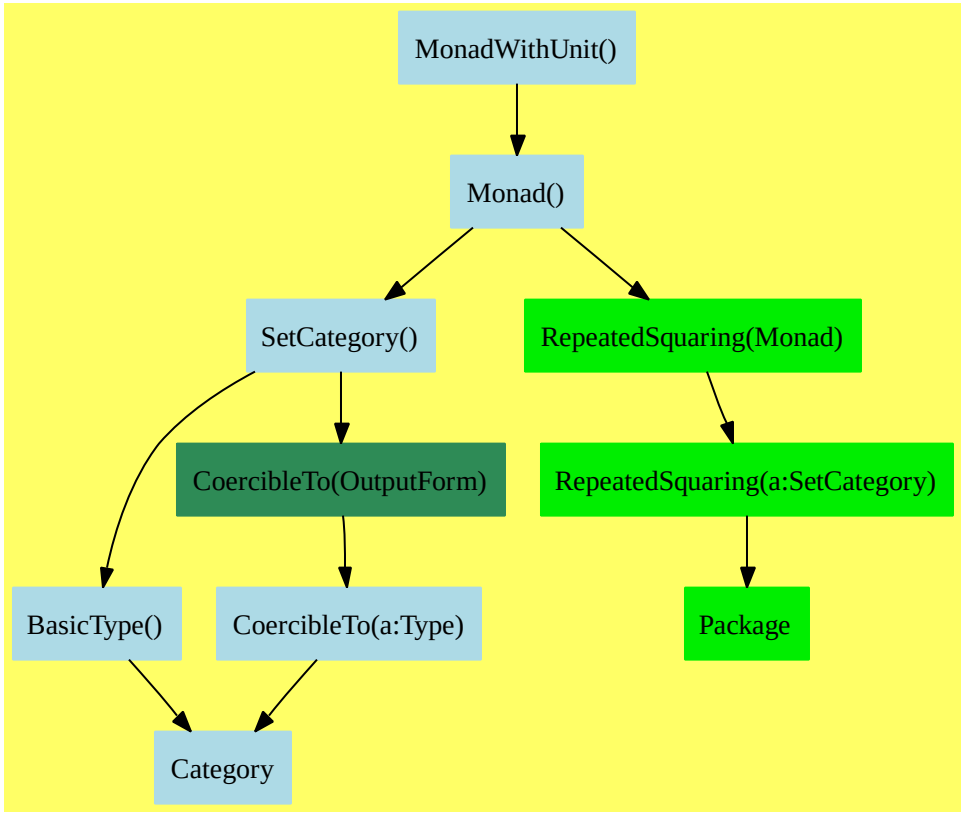
```
"Aggregate()" [color=lightblue];
"Aggregate()" -> "Type()"

"Type()" [color=lightblue];
"Type()" -> "Category"

"Category" [color=lightblue];
}
```

———-

5.0.75 MonadWithUnit (MONADWU)



— **MonadWithUnit.input** —

```
)set break resume
```

```
)sys rm -f MonadWithUnit.output
)spool MonadWithUnit.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show MonadWithUnit
--R MonadWithUnit  is a category constructor
--R Abbreviation for MonadWithUnit is MONADWU
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for MONADWU
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (%,%) -> %                    ?**? : (%,PositiveInteger) -> %
--R ?=? : (%,%) -> Boolean              1 : () -> %
--R coerce : % -> OutputForm            hash : % -> SingleInteger
--R latex : % -> String                 one? : % -> Boolean
--R recip : % -> Union(%,"failed")      ?~=? : (%,%) -> Boolean
--R ?**? : (%,NonNegativeInteger) -> %
--R leftPower : (%,NonNegativeInteger) -> %
--R leftPower : (%,PositiveInteger) -> %
--R leftRecip : % -> Union(%,"failed")
--R rightPower : (%,NonNegativeInteger) -> %
--R rightPower : (%,PositiveInteger) -> %
--R rightRecip : % -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— MonadWithUnit.help —

```
=====================================================================
MonadWithUnit examples
=====================================================================

See Also:
o )show MonadWithUnit
```

———-

See:

⇒ “NonAssociativeRing” (NASRING) 9.0.124 on page 740

⇐ "Monad" (MONAD) 4.0.52 on page 197

Exports:

1	coerce	hash	latex	one?
recip	leftPower	leftRecip	rightPower	rightRecip
?*?	?~=?	?**?	?=?	

These are directly exported but not implemented:

```
1 : () -> %
leftRecip : % -> Union(%,"failed")
recip : % -> Union(%,"failed")
rightRecip : % -> Union(%,"failed")
```

These are implemented by this category:

```
leftPower : (%,NonNegativeInteger) -> %
one? : % -> Boolean
rightPower : (%,NonNegativeInteger) -> %
?**? : (%,NonNegativeInteger) -> %
```

These exports come from (p197) Monad():

```
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
leftPower : (%,PositiveInteger) -> %
rightPower : (%,PositiveInteger) -> %
?**? : (%,PositiveInteger) -> %
?*? : (%,%) -> %
?~=? : (%,%) -> Boolean
?=? : (%,%) -> Boolean
```

— category MONADWU MonadWithUnit —

```
)abbrev category MONADWU MonadWithUnit
++ Authors: J. Grabmeier, R. Wisbauer
++ Date Created: 01 March 1991
++ Date Last Updated: 11 June 1991
++ Basic Operations: *, **, 1
++ Related Constructors: SemiGroup, Monoid, Monad
++ Also See:
++ AMS Classifications:
++ Keywords:
++ Keywords: Monad with unit, binary operation
++ Reference:
++ N. Jacobson: Structure and Representations of Jordan Algebras
++ AMS, Providence, 1968
```

```
++ Description:
++ MonadWithUnit is the class of multiplicative monads with unit,
++ i.e. sets with a binary operation and a unit element.
++
++ Axioms\br
++ \tab{5}leftIdentity("*":(%,%)->%,1) e.g. 1*x=x\br
++ \tab{5}rightIdentity("*":(%,%)->%,1) e.g x*1=x
++
++ Common Additional Axioms\br
++ \tab{5}unitsKnown - if "recip" says "failed", it PROVES input wasn't a unit

MonadWithUnit(): Category == Monad with
      1: constant ->  %
        ++ 1 returns the unit element, denoted by 1.
      one?: % -> Boolean
        ++ one?(a) tests whether \spad{a} is the unit 1.
      rightPower: (%,NonNegativeInteger) -> %
        ++ rightPower(a,n) returns the \spad{n}-th right power of \spad{a},
        ++ i.e. \spad{rightPower(a,n) := rightPower(a,n-1) * a} and
        ++ \spad{rightPower(a,0) := 1}.
      leftPower: (%,NonNegativeInteger) -> %
        ++ leftPower(a,n) returns the \spad{n}-th left power of \spad{a},
        ++ i.e. \spad{leftPower(a,n) := a * leftPower(a,n-1)} and
        ++ \spad{leftPower(a,0) := 1}.
      "**": (%,NonNegativeInteger) -> %
        ++ \spad{a**n} returns the \spad{n}-th power of \spad{a},
        ++ defined by repeated squaring.
      recip: % -> Union(%,"failed")
        ++ recip(a) returns an element, which is both a left and a right
        ++ inverse of \spad{a},
        ++ or \spad{"failed"} if such an element doesn't exist or cannot
        ++ be determined (see unitsKnown).
      leftRecip: % -> Union(%,"failed")
        ++ leftRecip(a) returns an element, which is a left inverse of
        ++ \spad{a}, or \spad{"failed"} if such an element doesn't exist
        ++ or cannot be determined (see unitsKnown).
      rightRecip: % -> Union(%,"failed")
        ++ rightRecip(a) returns an element, which is a right inverse of
        ++ \spad{a}, or \spad{"failed"} if such an element doesn't exist
        ++ or cannot be determined (see unitsKnown).
    add
      import RepeatedSquaring(%)
      one? x == x = 1
      x:% ** n:NonNegativeInteger ==
         zero? n => 1
         expt(x,n pretend PositiveInteger)
      rightPower(a,n) ==
        zero? n => 1
        res := 1
        for i in 1..n repeat res := res * a
```

```
        res
      leftPower(a,n) ==
        zero? n => 1
        res := 1
        for i in 1..n repeat res := a * res
        res
```

———-

— **MONADWU.dotabb** —

```
"MONADWU"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=MONADWU"];
"MONADWU" -> "MONAD"
```

———-

— **MONADWU.dotfull** —

```
"MonadWithUnit()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=MONADWU"];
"MonadWithUnit()" -> "Monad()"
```

———-

— **MONADWU.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"MonadWithUnit()" [color=lightblue];
"MonadWithUnit()" -> "Monad()"

"Monad()" [color=lightblue];
"Monad()" -> "SetCategory()"
"Monad()" -> "RepeatedSquaring(Monad)"

"RepeatedSquaring(Monad)" [color="#00EE00"];
"RepeatedSquaring(Monad)" -> "RepeatedSquaring(a:SetCategory)"

"RepeatedSquaring(a:SetCategory)" [color="#00EE00"];
"RepeatedSquaring(a:SetCategory)" -> "Package"
```

```
"Package" [color="#00EE00"];

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

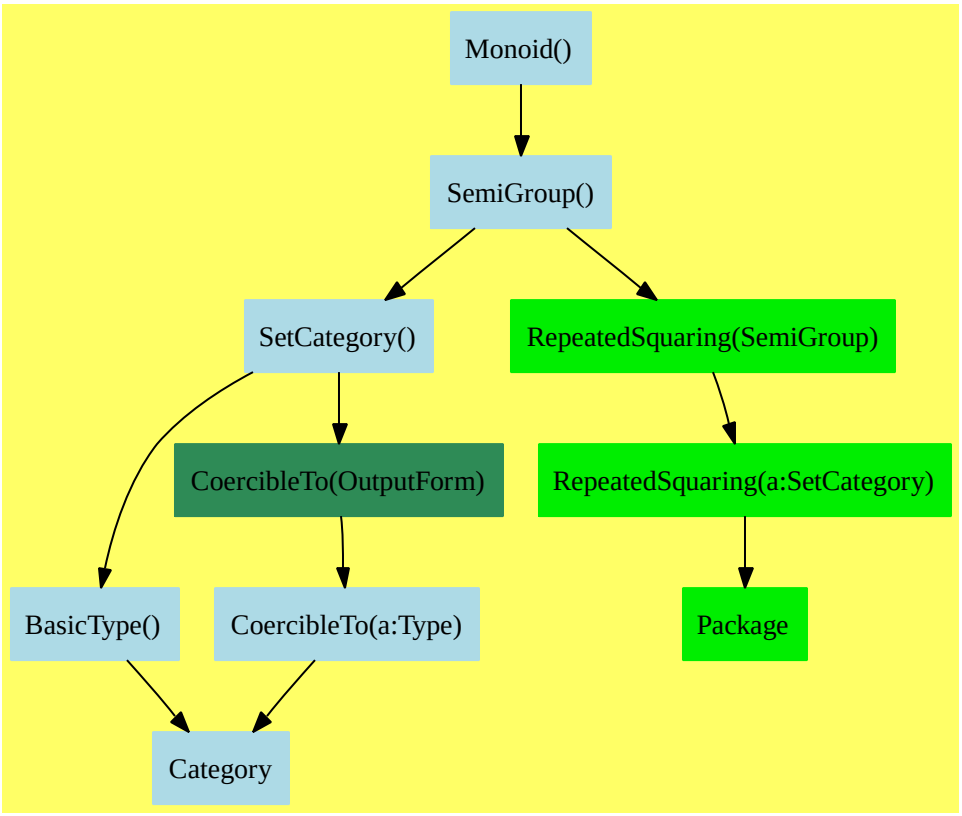
"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];

}
```

5.0.76 Monoid (MONOID)



— Monoid.input —

```
)set break resume
)sys rm -f Monoid.output
)spool Monoid.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show Monoid
--R Monoid  is a category constructor
--R Abbreviation for Monoid is MONOID
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for MONOID
--R
--R------------------------------ Operations --------------------------------
```

```
--R ?*? : (%,%) -> %                        ?**? : (%,PositiveInteger) -> %
--R ?=? : (%,%) -> Boolean                  1 : () -> %
--R ?^? : (%,PositiveInteger) -> %          coerce : % -> OutputForm
--R hash : % -> SingleInteger               latex : % -> String
--R one? : % -> Boolean                     recip : % -> Union(%,"failed")
--R sample : () -> %                        ?~=? : (%,%) -> Boolean
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **Monoid.help** —

```
====================================================================
Monoid examples
====================================================================

See Also:
o )show Monoid
```

———-

See:

⇒ “FunctionSpace” (FS) 17.0.195 on page 1336
⇒ “Group” (GROUP) 6.0.86 on page 398
⇒ “OrderedMonoid” (ORDMON) 6.0.90 on page 461
⇒ “OrderedRing” (ORDRING) 10.0.144 on page 879
⇒ “Ring” (RING) 9.0.129 on page 770
⇐ “SemiGroup” (SGROUP) 4.0.61 on page 239

Exports:

1	coerce	hash	latex	one?
recip	sample	?*?	?=?	?~=?
?**?	?^?			

These are directly exported but not implemented:

```
1 : () -> %
```

These are implemented by this category:

```
 one? : % -> Boolean
 recip : % -> Union(%,"failed")
 sample : () -> %
 ?^? : (%,NonNegativeInteger) -> %
 ?**? : (%,NonNegativeInteger) -> %
```

These exports come from (p239) SemiGroup():

```
 coerce : % -> OutputForm
 hash : % -> SingleInteger
 latex : % -> String
 ?*? : (%,%) -> %
 ?**? : (%,PositiveInteger) -> %
 ?^? : (%,PositiveInteger) -> %
 ?=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
```

— category MONOID Monoid —

```
)abbrev category MONOID Monoid
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The class of multiplicative monoids, i.e. semigroups with a
++ multiplicative identity element.
++
++ Axioms\br
++ \tab{5}\spad{leftIdentity("*":(%,%)->%,1)}\tab{5}\spad{1*x=x}\br
++ \tab{5}\spad{rightIdentity("*":(%,%)->%,1)}\tab{4}\spad{x*1=x}
++
++ Conditional attributes\br
++ \tab{5}unitsKnown - \spadfun{recip} only returns "failed" on non-units

Monoid(): Category == SemiGroup with
      1: constant ->  %
        ++ 1 is the multiplicative identity.
      sample: constant -> %
        ++ sample yields a value of type %
      one?: % -> Boolean
        ++ one?(x) tests if x is equal to 1.
      "**": (%,NonNegativeInteger) -> %
        ++ x**n returns the repeated product
```

```
        ++ of x n times, i.e. exponentiation.
      "^" : (%,NonNegativeInteger) -> %
        ++ x^n returns the repeated product
        ++ of x n times, i.e. exponentiation.
      recip: % -> Union(%,"failed")
        ++ recip(x) tries to compute the multiplicative inverse for x
        ++ or "failed" if it cannot find the inverse (see unitsKnown).
    add
      import RepeatedSquaring(%)
      _^(x:%, n:NonNegativeInteger):% == x ** n
      one? x == x = 1
      sample() == 1
      recip x ==
        (x = 1) => x
        "failed"
      x:% ** n:NonNegativeInteger ==
         zero? n => 1
         expt(x,n pretend PositiveInteger)
```

———-

— **MONOID.dotabb** —

```
"MONOID"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=MONOID"];
"MONOID" -> "SGROUP"
```

———-

— **MONOID.dotfull** —

```
"Monoid()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=MONOID"];
"Monoid()" -> "SemiGroup()"
```

———-

— **MONOID.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"Monoid()" [color=lightblue];
```

```
"Monoid()" -> "SemiGroup()"

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SetCategory()"
"SemiGroup()" -> "RepeatedSquaring(a:SemiGroup)"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"RepeatedSquaring(a:SemiGroup)" [color="#00EE00"];
"RepeatedSquaring(a:SemiGroup)" -> "RepeatedSquaring(a:SetCategory)"

"RepeatedSquaring(a:SetCategory)" [color="#00EE00"];
"RepeatedSquaring(a:SetCategory)" -> "Package"

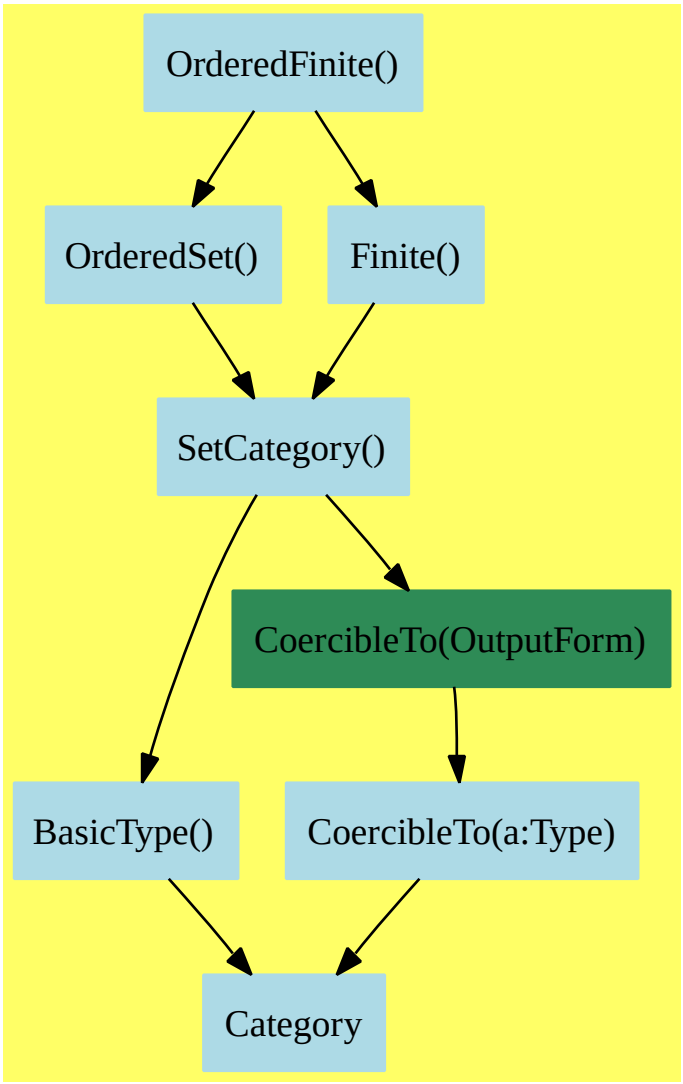
"Package" [color="#00EE00"];

"Category" [color=lightblue];

}
```

———————-

5.0.77 OrderedFinite (ORDFIN)



— OrderedFinite.input —

```
)set break resume
)sys rm -f OrderedFinite.output
)spool OrderedFinite.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 1
)show OrderedFinite
--R OrderedFinite  is a category constructor
--R Abbreviation for OrderedFinite is ORDFIN
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for ORDFIN
--R
--R------------------------------ Operations --------------------------------
--R ?<? : (%,%) -> Boolean               ?<=? : (%,%) -> Boolean
--R ?=? : (%,%) -> Boolean               ?>? : (%,%) -> Boolean
--R ?>=? : (%,%) -> Boolean              coerce : % -> OutputForm
--R hash : % -> SingleInteger            index : PositiveInteger -> %
--R latex : % -> String                  lookup : % -> PositiveInteger
--R max : (%,%) -> %                     min : (%,%) -> %
--R random : () -> %                     size : () -> NonNegativeInteger
--R ?~=? : (%,%) -> Boolean
--R
--E 1

)spool
)lisp (bye)
```

———-

— **OrderedFinite.help** —

```
=====================================================================
OrderedFinite examples
=====================================================================

See Also:
o )show OrderedFinite
```

———-

See:

⇐ “Finite” (FINITE) 4.0.46 on page 168
⇐ “OrderedSet” (ORDSET) 4.0.56 on page 216

Exports:

coerce	hash	index	latex	lookup
max	min	random	size	?~=?
?<?	?<=?	?=?	?>?	?>=?

These exports come from (p216) OrderedSet():

```
 coerce : % -> OutputForm
 hash : % -> SingleInteger
 latex : % -> String
 max : (%,%) -> %
 min : (%,%) -> %
 ?=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
 ?<? : (%,%) -> Boolean
 ?<=? : (%,%) -> Boolean
 ?>? : (%,%) -> Boolean
 ?>=? : (%,%) -> Boolean
```

These exports come from (p168) Finite():

```
 index : PositiveInteger -> %
 lookup : % -> PositiveInteger
 random : () -> %
 size : () -> NonNegativeInteger
```

— category ORDFIN OrderedFinite —

```
)abbrev category ORDFIN OrderedFinite
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ Ordered finite sets.

OrderedFinite(): Category == Join(OrderedSet, Finite)
```

———-

— ORDFIN.dotabb —

```
"ORDFIN"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ORDFIN"];
"ORDFIN" -> "ORDSET"
"ORDFIN" -> "FINITE"
```

———-

— **ORDFIN.dotfull** —

```
"OrderedFinite()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ORDFIN"];
"OrderedFinite()" -> "OrderedSet()"
"OrderedFinite()" -> "Finite()"
```

———-

— **ORDFIN.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"OrderedFinite()" [color=lightblue];
"OrderedFinite()" -> "OrderedSet()"
"OrderedFinite()" -> "Finite()"

"Finite()" [color=lightblue];
"Finite()" -> "SetCategory()"

"OrderedSet()" [color=lightblue];
"OrderedSet()" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

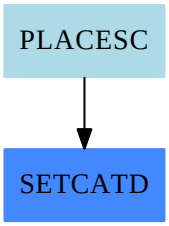
"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];

}
```

———-

5.0.78 PlacesCategory (PLACESC)



— PlacesCategory.input —

```
)set break resume
)sys rm -f PlacesCategory.output
)spool PlacesCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show PlacesCategory
--R PlacesCategory(K: Field,PCS: LocalPowerSeriesCategory t#1)  is a category constructor
--R Abbreviation for PlacesCategory is PLACESC
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PLACESC
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (Integer,%) -> Divisor %        ?+? : (%,%) -> Divisor %
--R -? : % -> Divisor %                   ?-? : (%,%) -> Divisor %
--R ?=? : (%,%) -> Boolean                coerce : % -> OutputForm
--R create : Symbol -> %                  create : List K -> %
--R degree : % -> PositiveInteger         ?.? : (%,Integer) -> K
--R foundPlaces : () -> List %            hash : % -> SingleInteger
--R itsALeaf! : % -> Void                 latex : % -> String
--R leaf? : % -> Boolean                  localParam : % -> List PCS
--R reduce : List % -> Divisor %          setParam! : (%,List PCS) -> Void
--R ?~=? : (%,%) -> Boolean
--R ?+? : (%,Divisor %) -> Divisor %
--R ?+? : (Divisor %,%) -> Divisor %
--R ?-? : (%,Divisor %) -> Divisor %
--R ?-? : (Divisor %,%) -> Divisor %
--R setDegree! : (%,PositiveInteger) -> Void
--R setFoundPlacesToEmpty : () -> List %
--R
--E 1

)spool
```

```
)lisp (bye)
```

———-

— **PlacesCategory.help** —

```
====================================================================
PlacesCategory examples
====================================================================

See Also:
o )show PlacesCategory
```

———-

See:

⇐ “SetCategoryWithDegree” (SETCATD) 4.0.62 on page 243

Exports:

-?	?*?	?+?	?-?
?.?	?=?	?~=?	coerce
create	degree	foundPlaces	hash
itsALeaf!	latex	leaf?	localParam
reduce	setDegree!	setFoundPlacesToEmpty	setParam!

These are directly exported but not implemented:

```
-? : % -> Divisor %
?*? : (Integer,%) -> Divisor %
?+? : (%,%) -> Divisor %
?+? : (%,Divisor %) -> Divisor %
?+? : (Divisor %,%) -> Divisor %
?-? : (%,%) -> Divisor %
?-? : (%,Divisor %) -> Divisor %
?-? : (Divisor %,%) -> Divisor %
?.? : (%,Integer) -> K
create : List K -> %
create : Symbol -> %
foundPlaces : () -> List %
itsALeaf! : % -> Void
leaf? : % -> Boolean
localParam : % -> List PCS
reduce : List % -> Divisor %
setDegree! : (%,PositiveInteger) -> Void
setFoundPlacesToEmpty : () -> List %
setParam! : (%,List PCS) -> Void
```

These exports come from (p243) SetCategoryWithDegree

```
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
coerce : % -> OutputForm
degree : % -> PositiveInteger
hash : % -> SingleInteger
latex : % -> String
```

— category PLACESC PlacesCategory —

```
)abbrev category PLACESC PlacesCategory
++ Author: Gaetan Hache
++ Date Created: 17 nov 1992
++ Date Last Updated: May 2010 by Tim Daly
++ Description:
++ This is part of the PAFF package, related to projective space.
PlacesCategory(K:Field,PCS:LocalPowerSeriesCategory(K)):Category
  == SetCategoryWithDegree with

    "+": (%,%)       -> Divisor(%)

    "+": (Divisor(%) , % ) -> Divisor(%)

    "+": (%, Divisor(%)) -> Divisor(%)

    "-": (%,%)       -> Divisor(%)

    "-": (Divisor(%) , % ) -> Divisor(%)

    "-": (%, Divisor(%)) -> Divisor(%)

    "-": % -> Divisor(%)

    "*": (Integer,%) -> Divisor(%)

    reduce: List %   -> Divisor(%)

    create: List K -> %

    create: Symbol -> %

    localParam: % -> List PCS
      ++ localParam(pl) returns the local parametrization associated
      ++ to the place pl.

    setParam_!: (%,List PCS)  -> Void
      ++ setParam!(pl,ls) set the local parametrization of pl to ls.
```

```
    setDegree_!: (%,PositiveInteger)  -> Void
      ++ setDegree!(pl,ls) set the degree.

    leaf?: % -> Boolean
      ++ leaf?(pl) test if the place pl correspond to a leaf of a
      ++ desingularisation tree.

    itsALeaf_!: % -> Void

    foundPlaces: () -> List %
      ++ foundPlaces() returns the list of all "created" places up to now.

    setFoundPlacesToEmpty: () -> List %
      ++ setFoundPlacesToEmpty() does what it says.
      ++ (this should not be used)!!!

    elt: (%,Integer) -> K
      ++ elt returns the value of a specified coordinates if the places
      ++ correspnd to a simple point
```

———-

— PLACESC.dotabb —

```
"PLACESC" [color=lightblue,href="bookvol10.2.pdf#nameddest=PLACESC"];
"SETCATD" [color="#4488FF",href="bookvol10.2.pdf#nameddest=SETCATD"]
"PLACESC" -> "SETCATD"
```

———-

— PLACESC.dotfull —

```
"PlacesCategory()" [color=lightblue,href="bookvol10.2.pdf#nameddest=PLACESC"];
"PlacesCategory()" -> "SetCategoryWithDegree()"
```

———-

— PLACESC.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"PlacesCategory()" [color=lightblue];
```

```
"PlacesCategory()" -> "SetCategoryWithDegree()"

"SetCategoryWithDegree()" [color=lightblue];
"SetCategoryWithDegree()" -> "BasicType()"
"SetCategoryWithDegree()" -> "CoercibleTo(OutputForm)"
"SetCategoryWithDegree()" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

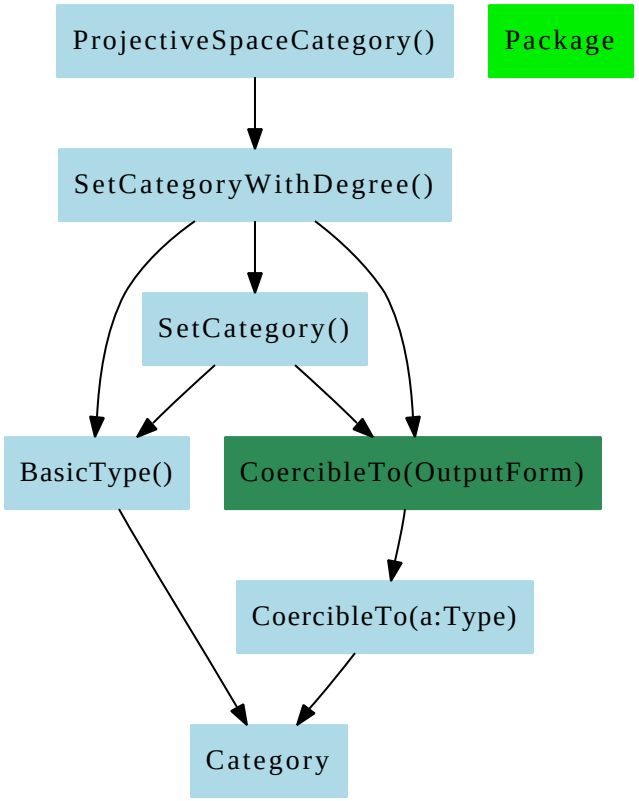
"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Package" [color="#00EE00"];

"Category" [color=lightblue];

}
```

5.0.79 ProjectiveSpaceCategory (PRSPCAT)



— ProjectiveSpaceCategory.input —

```
)set break resume
)sys rm -f ProjectiveSpaceCategory.output
)spool ProjectiveSpaceCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show ProjectiveSpaceCategory
--R ProjectiveSpaceCategory K: Field  is a category constructor
--R Abbreviation for ProjectiveSpaceCategory is PRSPCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PRSPCAT
--R
--R------------------------------ Operations --------------------------------
```

```
--R ?=? : (%,%) -> Boolean                coerce : List K -> %
--R coerce : % -> List K                  coerce : % -> OutputForm
--R conjugate : % -> %                    definingField : % -> K
--R degree : % -> PositiveInteger         ?.? : (%,Integer) -> K
--R hash : % -> SingleInteger             homogenize : % -> %
--R homogenize : (%,Integer) -> %         lastNonNul : % -> Integer
--R lastNonNull : % -> Integer            latex : % -> String
--R list : % -> List K                    orbit : % -> List %
--R pointValue : % -> List K              projectivePoint : List K -> %
--R rational? : % -> Boolean              setelt : (%,Integer,K) -> K
--R ?~=? : (%,%) -> Boolean
--R conjugate : (%,NonNegativeInteger) -> %
--R orbit : (%,NonNegativeInteger) -> List %
--R rational? : (%,NonNegativeInteger) -> Boolean
--R removeConjugate : List % -> List %
--R removeConjugate : (List %,NonNegativeInteger) -> List %
--R
--E 1

)spool
)lisp (bye)
```

———-

— ProjectiveSpaceCategory.help —

```
=====================================================================
ProjectiveSpaceCategory examples
=====================================================================

See Also:
o )show ProjectiveSpaceCategory
```

———-

See:

⇐ “SetCategoryWithDegree” (SETCATD) 4.0.62 on page 243

Exports:

?.?	?=?	? =?	coerce
conjugate	definingField	degree	hash
homogenize	lastNonNul	lastNonNull	latex
list	orbit	orbit	pointValue
projectivePoint	rational?	removeConjugate	setelt

These are directly exported but not implemented:

```
 ?.? : (%,Integer) -> K
 coerce : % -> List K
 coerce : List K -> %
 conjugate : % -> %
 conjugate : (%,NonNegativeInteger) -> %
 definingField : % -> K
 homogenize : % -> %
 homogenize : (%,Integer) -> %
 lastNonNul : % -> Integer
 lastNonNull : % -> Integer
 list : % -> List K
 orbit : % -> List %
 orbit : (%,NonNegativeInteger) -> List %
 pointValue : % -> List K
 projectivePoint : List K -> %
 rational? : % -> Boolean
 rational? : (%,NonNegativeInteger) -> Boolean
 removeConjugate : (List %,NonNegativeInteger) -> List %
 removeConjugate : List % -> List %
 setelt : (%,Integer,K) -> K
```

These exports come from (p243) SetCategoryWithDegree

```
 ?=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
 coerce : % -> OutputForm
 degree : % -> PositiveInteger
 hash : % -> SingleInteger
 latex : % -> String
```

— category PRSPCAT ProjectiveSpaceCategory —

```
)abbrev category PRSPCAT ProjectiveSpaceCategory
++ Author: Gaetan Hache
++ Date Created: 17 nov 1992
++ Date Last Updated: May 2010 by Tim Daly
++ Description:
++ This is part of the PAFF package, related to projective space.
ProjectiveSpaceCategory(K:Field):Category == Implementation where

 LIST ==> List
 INT  ==> Integer
 NNI  ==> NonNegativeInteger

 Implementation ==>
  SetCategoryWithDegree with

    lastNonNul: % -> INT
```

```
homogenize: (%,INT) -> %
  ++ homogenize the point according to the coordinate specified
  ++ by the integer

projectivePoint: LIST(K) -> %
  ++ projectivePoint creates a projective point from a list

definingField: % -> K

coerce: % -> List(K)
  ++ coerce a a projective point list of K

coerce: LIST(K) -> %
  ++ coerce a list of K to a projective point.

orbit: % -> LIST %
  ++ orbit(p) returns the orbit of the point p according to the
  ++ characteristic of K, that is, for q= char K,
  ++ orbit(p) = \{ p, p**q, p**(q**2), p**(q**3), ..... \}

orbit: (%,NNI) -> LIST %
  ++ orbit(p,n) returns the orbit of the point p according to n, that is
  ++ orbit(p,n) = \{ p, p**n, p**(n**2), p**(n**3), ..... \}

conjugate: (%,NNI) -> %
  ++ conjugate(p,n) returns p**n, that is all the coordinates of p
  ++ to the power of n

conjugate: % -> %
  ++ conjugate(p) returns conjugate(p,n) where n is the
  ++ characteristic of the ground field.

removeConjugate: (LIST(%),NNI) -> LIST(%)
  ++ removeConjugate(lp,n) returns a list of points such that no points
  ++ in the list is the conjugate (according to n) of another point.

removeConjugate: LIST(%)  -> LIST(%)
  ++ removeConjugate(lp) returns removeConjugate(lp,n)
  ++ where n is the characteristic of the ground field.

rational?: (%,NNI) -> Boolean
  ++ rational?(p,n) test if the point is rational according to n.

rational?: % -> Boolean
  ++ rational?(p) test if the point is rational according to the
  ++ characteristic of the ground field.

homogenize: % -> %
  ++ homogenize(pt) the point according to the coordinate
  ++ which is the last non null.
```

```
    lastNonNull: % -> INT
      ++ lastNonNull returns the integer corresponding to the last
      ++ non null coordinates.

    list  : % -> LIST(K)
      ++ list returns the list of the coordinates

    elt: (%,INT) -> K
      ++ elt returns the value of a specified coordinates

    setelt: (%,INT,K) -> K
      ++ setelt sets the value of a specified coordinates

    pointValue: % -> LIST(K)
     ++ pointValue returns the coordinates of the point or of the point
     ++ of origin that represent an infinitly close point
```

———-

— PRSPCAT.dotabb —

```
"PRSPCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PRSPCAT"];
"PRSPCAT" -> "SETCATD"
```

———-

— PRSPCAT.dotfull —

```
"ProjectiveSpaceCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PRSPCAT"];
"ProjectiveSpaceCategory()" -> "SetCategoryWithDegree()"
```

———-

— PRSPCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"ProjectiveSpaceCategory()" [color=lightblue];
"ProjectiveSpaceCategory()" -> "SetCategoryWithDegree()"
```

```
"SetCategoryWithDegree()" [color=lightblue];
"SetCategoryWithDegree()" -> "BasicType()"
"SetCategoryWithDegree()" -> "CoercibleTo(OutputForm)"
"SetCategoryWithDegree()" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

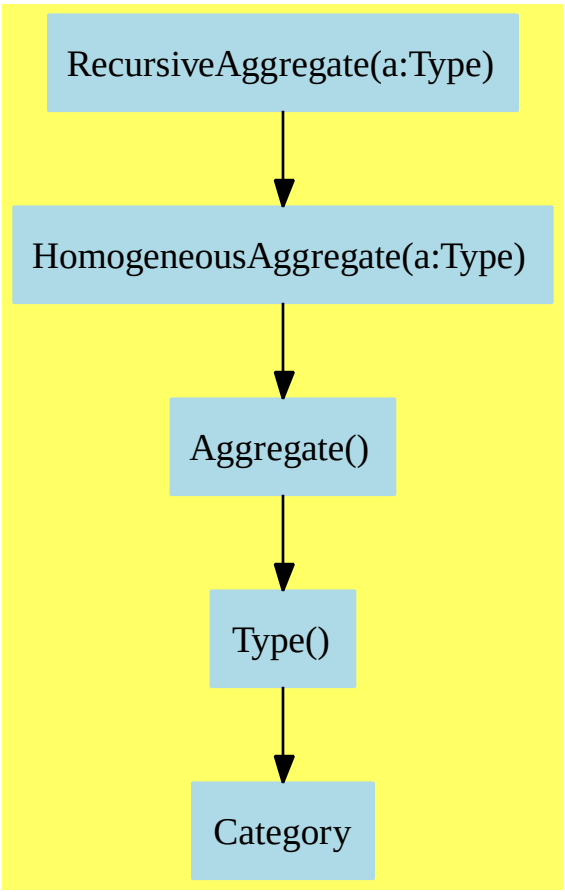
"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Package" [color="#00EE00"];

"Category" [color=lightblue];

}
```

5.0.80 RecursiveAggregate (RCAGG)



— RecursiveAggregate.input —

```
)set break resume
)sys rm -f RecursiveAggregate.output
)spool RecursiveAggregate.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show RecursiveAggregate
--R RecursiveAggregate S: Type  is a category constructor
--R Abbreviation for RecursiveAggregate is RCAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for RCAGG
```

```
--R
--R------------------------------ Operations ------------------------------
--R children : % -> List %                copy : % -> %
--R cyclic? : % -> Boolean                distance : (%,%) -> Integer
--R ?.value : (%,value) -> S              empty : () -> %
--R empty? : % -> Boolean                 eq? : (%,%) -> Boolean
--R leaf? : % -> Boolean                  leaves : % -> List S
--R map : ((S -> S),%) -> %               nodes : % -> List %
--R sample : () -> %                      value : % -> S
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R child? : (%,%) -> Boolean if S has SETCAT
--R coerce : % -> OutputForm if S has SETCAT
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R hash : % -> SingleInteger if S has SETCAT
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R node? : (%,%) -> Boolean if S has SETCAT
--R parts : % -> List S if $ has finiteAggregate
--R setchildren! : (%,List %) -> % if $ has shallowlyMutable
--R setelt : (%,value,S) -> S if $ has shallowlyMutable
--R setvalue! : (%,S) -> S if $ has shallowlyMutable
--R size? : (%,NonNegativeInteger) -> Boolean
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

— RecursiveAggregate.help —

```
=====================================================================
RecursiveAggregate examples
=====================================================================
```

```
See Also:
o )show RecursiveAggregate
```

See:

⇒ "BinaryRecursiveAggregate" (BRAGG) 6.0.82 on page 372
⇒ "DoublyLinkedAggregate" (DLAGG) 6.0.85 on page 392
⇒ "UnaryRecursiveAggregate" (URAGG) 6.0.96 on page 506
⇐ "HomogeneousAggregate" (HOAGG) 4.0.49 on page 181

Exports:

any?	child?	children	coerce	copy
count	cyclic?	distance	empty	empty?
eq?	eval	every?	hash	latex
leaf?	leaves	less?	map	map!
member?	members	more?	nodes	node?
parts	sample	setchildren!	setelt	setvalue!
size?	value	?.value	?~=?	#?
?=?				

Attributes exported:

- **nil**

Attributes Used:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.

These are directly exported but not implemented:

```
children : % -> List %
cyclic? : % -> Boolean
distance : (%,%) -> Integer
nodes : % -> List %
leaf? : % -> Boolean
leaves : % -> List S
node? : (%,%) -> Boolean if S has SETCAT
setchildren! : (%,List %) -> % if $ has shallowlyMutable
setvalue! : (%,S) -> S if $ has shallowlyMutable
value : % -> S
```

These are implemented by this category:

```
child? : (%,%) -> Boolean if S has SETCAT
setelt : (%,value,S) -> S if $ has shallowlyMutable
?.value : (%,value) -> S
```

These exports come from (p181) HomogeneousAggregate(S:Type):

```
any? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
coerce : % -> OutputForm if S has SETCAT
copy : % -> %
count : (S,%) -> NonNegativeInteger
         if S has SETCAT and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> %
      if S has EVALAB S and S has SETCAT
eval : (%,S,S) -> %
      if S has EVALAB S and S has SETCAT
eval : (%,Equation S) -> %
      if S has EVALAB S and S has SETCAT
eval : (%,List Equation S) -> %
      if S has EVALAB S and S has SETCAT
every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
hash : % -> SingleInteger if S has SETCAT
latex : % -> String if S has SETCAT
less? : (%,NonNegativeInteger) -> Boolean
map : ((S -> S),%) -> %
map! : ((S -> S),%) -> % if $ has shallowlyMutable
member? : (S,%) -> Boolean
          if S has SETCAT and $ has finiteAggregate
members : % -> List S if $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
parts : % -> List S if $ has finiteAggregate
sample : () -> %
size? : (%,NonNegativeInteger) -> Boolean
?~=? : (%,%) -> Boolean if S has SETCAT
#? : % -> NonNegativeInteger if $ has finiteAggregate
?=? : (%,%) -> Boolean if S has SETCAT
```

— category RCAGG RecursiveAggregate —

```
)abbrev category RCAGG RecursiveAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
```

```
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A recursive aggregate over a type S is a model for a
++ a directed graph containing values of type S.
++ Recursively, a recursive aggregate is a node
++ consisting of a \spadfun{value} from S and 0 or more \spadfun{children}
++ which are recursive aggregates.
++ A node with no children is called a \spadfun{leaf} node.
++ A recursive aggregate may be cyclic for which some operations as noted
++ may go into an infinite loop.

RecursiveAggregate(S:Type): Category == HomogeneousAggregate(S) with
   children: % -> List %
     ++ children(u) returns a list of the children of aggregate u.
   -- should be % -> %* and also needs children: % -> Iterator(S,S)
   nodes: % -> List %
     ++ nodes(u) returns a list of all of the nodes of aggregate u.
   -- to become % -> %* and also nodes: % -> Iterator(S,S)
   leaf?: % -> Boolean
     ++ leaf?(u) tests if u is a terminal node.
   value: % -> S
     ++ value(u) returns the value of the node u.
   elt: (%,"value") -> S
     ++ elt(u,"value") (also written: \axiom{a. value}) is
     ++ equivalent to \axiom{value(a)}.
   cyclic?: % -> Boolean
     ++ cyclic?(u) tests if u has a cycle.
   leaves: % -> List S
     ++ leaves(t) returns the list of values in obtained by visiting the
     ++ nodes of tree \axiom{t} in left-to-right order.
   distance: (%,%) -> Integer
     ++ distance(u,v) returns the path length (an integer) from node u to v.
   if S has SetCategory then
      child?: (%,%) -> Boolean
        ++ child?(u,v) tests if node u is a child of node v.
      node?: (%,%) -> Boolean
        ++ node?(u,v) tests if node u is contained in node v
        ++ (either as a child, a child of a child, etc.).
   if % has shallowlyMutable then
      setchildren_!: (%,List %)->%
        ++ setchildren!(u,v) replaces the current children of node u
        ++ with the members of v in left-to-right order.
      setelt: (%,"value",S) -> S
        ++ setelt(a,"value",x) (also written \axiom{a . value := x})
        ++ is equivalent to \axiom{setvalue!(a,x)}
      setvalue_!: (%,S) -> S
```

```
        ++ setvalue!(u,x) sets the value of node u to x.
 add
   elt(x,"value") == value x
   if % has shallowlyMutable then
     setelt(x,"value",y) == setvalue_!(x,y)
   if S has SetCategory then
     child?(x,l) == member?(x,children(l))
```

———-

— **RCAGG.dotabb** —

```
"RCAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=RCAGG"];
"RCAGG" -> "HOAGG"
```

———-

— **RCAGG.dotfull** —

```
"RecursiveAggregate(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RCAGG"];
"RecursiveAggregate(a:Type)" -> "HomogeneousAggregate(a:Type)"
```

———-

— **RCAGG.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"RecursiveAggregate(a:Type)" [color=lightblue];
"RecursiveAggregate(a:Type)" -> "HomogeneousAggregate(a:Type)"

"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "Aggregate()"

"Aggregate()" [color=lightblue];
"Aggregate()" -> "Type()"

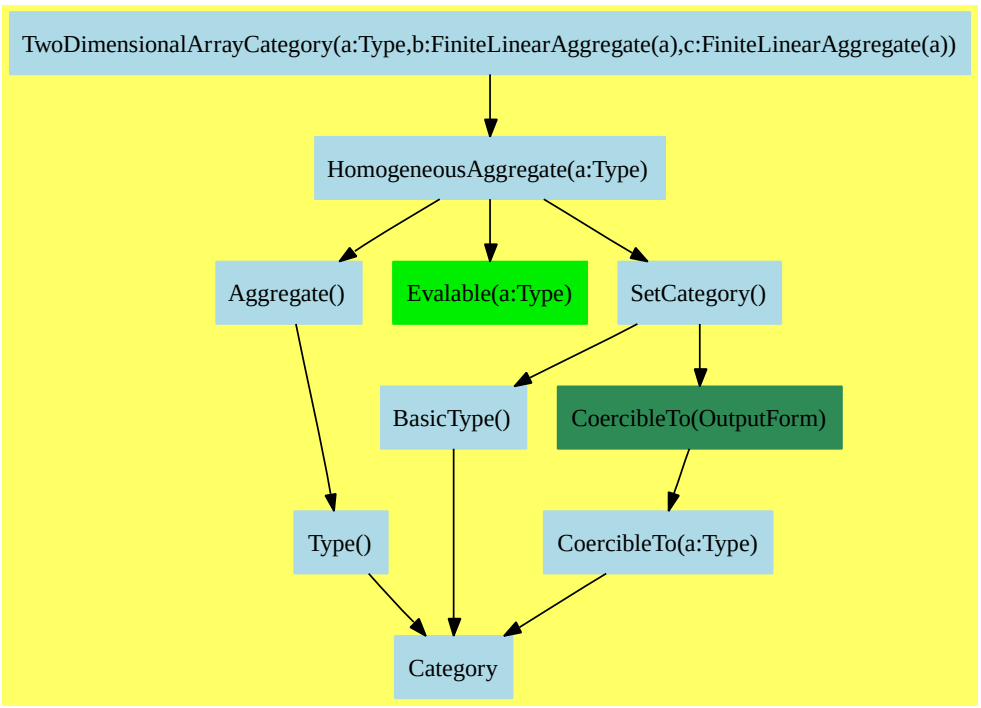
"Type()" [color=lightblue];
"Type()" -> "Category"

"Category" [color=lightblue];
```

```
}
```

———-

5.0.81 TwoDimensionalArrayCategory (ARR2CAT)



TwoDimensionalArrayCategory is a general array category which allows different representations and indexing schemes. Rows and columns may be extracted with rows returned as objects of type Row and columns returned as objects of type Col. The index of the 'first' row may be obtained by calling the function 'minRowIndex'. The index of the 'first' column may be obtained by calling the function 'minColIndex'. The index of the first element of a 'Row' is the same as the index of the first column in an array and vice versa.

— **TwoDimensionalArrayCategory.input** —

```
)set break resume
)sys rm -f TwoDimensionalArrayCategory.output
)spool TwoDimensionalArrayCategory.output
)set message test on
```

```
)set message auto off
)clear all

--S 1 of 1
)show TwoDimensionalArrayCategory
--R TwoDimensionalArrayCategory(R: Type,Row: FiniteLinearAggregate t#1,Col: FiniteLinearAggregat
--R Abbreviation for TwoDimensionalArrayCategory is ARR2CAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for ARR2CAT
--R
--R------------------------------ Operations --------------------------------
--R column : (%,Integer) -> Col           copy : % -> %
--R elt : (%,Integer,Integer,R) -> R      elt : (%,Integer,Integer) -> R
--R empty : () -> %                       empty? : % -> Boolean
--R eq? : (%,%) -> Boolean                fill! : (%,R) -> %
--R map : (((R,R) -> R),%,%,R) -> %       map : (((R,R) -> R),%,%) -> %
--R map : ((R -> R),%) -> %               map! : ((R -> R),%) -> %
--R maxColIndex : % -> Integer            maxRowIndex : % -> Integer
--R minColIndex : % -> Integer            minRowIndex : % -> Integer
--R ncols : % -> NonNegativeInteger       nrows : % -> NonNegativeInteger
--R parts : % -> List R                   qelt : (%,Integer,Integer) -> R
--R row : (%,Integer) -> Row              sample : () -> %
--R setRow! : (%,Integer,Row) -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if R has SETCAT
--R any? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
--R coerce : % -> OutputForm if R has SETCAT
--R count : (R,%) -> NonNegativeInteger if R has SETCAT and $ has finiteAggregate
--R count : ((R -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R eval : (%,List R,List R) -> % if R has EVALAB R and R has SETCAT
--R eval : (%,R,R) -> % if R has EVALAB R and R has SETCAT
--R eval : (%,Equation R) -> % if R has EVALAB R and R has SETCAT
--R eval : (%,List Equation R) -> % if R has EVALAB R and R has SETCAT
--R every? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
--R hash : % -> SingleInteger if R has SETCAT
--R latex : % -> String if R has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R member? : (R,%) -> Boolean if R has SETCAT and $ has finiteAggregate
--R members : % -> List R if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R new : (NonNegativeInteger,NonNegativeInteger,R) -> %
--R qsetelt! : (%,Integer,Integer,R) -> R
--R setColumn! : (%,Integer,Col) -> %
--R setelt : (%,Integer,Integer,R) -> R
--R size? : (%,NonNegativeInteger) -> Boolean
--R ?~=? : (%,%) -> Boolean if R has SETCAT
--R
--E 1

)spool
```

```
)lisp (bye)
```

———-

— **TwoDimensionalArrayCategory.help** —

```
====================================================================
TwoDimensionalArrayCategory examples
====================================================================

See Also:
o )show TwoDimensionalArrayCategory
```

———-

See:

⇒ "MatrixCategory" (MATCAT) 6.0.88 on page 412
⇐ "HomogeneousAggregate" (HOAGG) 4.0.49 on page 181

Exports:

any?	column	coerce	copy	count
elt	empty	empty?	eq?	eval
every?	fill!	hash	latex	less?
map	map!	maxColIndex	maxRowIndex	member?
members	minColIndex	minRowIndex	more?	ncols
new	nrows	parts	qelt	qsetelt!
row	sample	setColumn!	setRow!	setelt
size?	#?	?=?	?~=?	

Attributes Exported:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **nil**

Attributes Used:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.

These are directly exported but not implemented:

```
elt : (%,Integer,Integer) -> R
maxColIndex : % -> Integer
maxRowIndex : % -> Integer
minColIndex : % -> Integer
minRowIndex : % -> Integer
new : (NonNegativeInteger,NonNegativeInteger,R) -> %
ncols : % -> NonNegativeInteger
nrows : % -> NonNegativeInteger
qelt : (%,Integer,Integer) -> R
qsetelt! : (%,Integer,Integer,R) -> R
setelt : (%,Integer,Integer,R) -> R
```

These are implemented by this category:

```
any? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
coerce : % -> OutputForm if R has SETCAT
column : (%,Integer) -> Col
copy : % -> %
count : ((R -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
count : (R,%) -> NonNegativeInteger if R has SETCAT and $ has finiteAggregate
elt : (%,Integer,Integer,R) -> R
every? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
fill! : (%,R) -> %
less? : (%,NonNegativeInteger) -> Boolean
map : ((R -> R),%) -> %
map : (((R,R) -> R),%,%) -> %
map : (((R,R) -> R),%,%,R) -> %
map! : ((R -> R),%) -> %
member? : (R,%) -> Boolean if R has SETCAT and $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
parts : % -> List R
row : (%,Integer) -> Row
setColumn! : (%,Integer,Col) -> %
setRow! : (%,Integer,Row) -> %
size? : (%,NonNegativeInteger) -> Boolean
#? : % -> NonNegativeInteger if $ has finiteAggregate
?=? : (%,%) -> Boolean if R has SETCAT
```

These exports come from (p181) HomogeneousAggregate(R:Type)

```
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List R,List R) -> % if R has EVALAB R and R has SETCAT
eval : (%,R,R) -> % if R has EVALAB R and R has SETCAT
eval : (%,Equation R) -> % if R has EVALAB R and R has SETCAT
eval : (%,List Equation R) -> % if R has EVALAB R and R has SETCAT
hash : % -> SingleInteger if R has SETCAT
latex : % -> String if R has SETCAT
members : % -> List R if $ has finiteAggregate
```

```
 sample : () -> %
 ?~=? : (%,%) -> Boolean if R has SETCAT
```

— category ARR2CAT TwoDimensionalArrayCategory —

```
)abbrev category ARR2CAT TwoDimensionalArrayCategory
++ Author:
++ Date Created: 27 October 1989
++ Date Last Updated: 27 June 1990
++ Keywords: array, data structure
++ Examples:
++ References:
++ Description:
++ Two dimensional array categories and domains

TwoDimensionalArrayCategory(R,Row,Col): Category == Definition where
 R   : Type
 Row : FiniteLinearAggregate R
 Col : FiniteLinearAggregate R

 Definition == HomogeneousAggregate(R) with

   shallowlyMutable
    ++ one may destructively alter arrays

   finiteAggregate
    ++ two-dimensional arrays are finite

--% Array creation

   new: (NonNegativeInteger,NonNegativeInteger,R) -> %
    ++ new(m,n,r) is an m-by-n array all of whose entries are r
    ++
    ++X arr : ARRAY2 INT := new(5,4,0)

   fill_!: (%,R) -> %
    ++ fill!(m,r) fills m with r's
    ++
    ++X arr : ARRAY2 INT := new(5,4,0)
    ++X fill!(arr,10)

--% Size inquiries

   minRowIndex : % -> Integer
    ++ minRowIndex(m) returns the index of the 'first' row of the array m
    ++
    ++X arr : ARRAY2 INT := new(5,4,10)
    ++X minRowIndex(arr)
```

```
   maxRowIndex : % -> Integer
    ++ maxRowIndex(m) returns the index of the 'last' row of the array m
    ++
    ++X arr : ARRAY2 INT := new(5,4,10)
    ++X maxRowIndex(arr)

   minColIndex : % -> Integer
    ++ minColIndex(m) returns the index of the 'first' column of the array m
    ++
    ++X arr : ARRAY2 INT := new(5,4,10)
    ++X minColIndex(arr)

   maxColIndex : % -> Integer
    ++ maxColIndex(m) returns the index of the 'last' column of the array m
    ++
    ++X arr : ARRAY2 INT := new(5,4,10)
    ++X maxColIndex(arr)

   nrows : % -> NonNegativeInteger
    ++ nrows(m) returns the number of rows in the array m
    ++
    ++X arr : ARRAY2 INT := new(5,4,10)
    ++X nrows(arr)

   ncols : % -> NonNegativeInteger
    ++ ncols(m) returns the number of columns in the array m
    ++
    ++X arr : ARRAY2 INT := new(5,4,10)
    ++X ncols(arr)

--% Part extractions

   elt: (%,Integer,Integer) -> R
    ++ elt(m,i,j) returns the element in the ith row and jth
    ++ column of the array m
    ++ error check to determine if indices are in proper ranges
    ++
    ++X arr : ARRAY2 INT := new(5,4,10)
    ++X elt(arr,1,1)

   qelt: (%,Integer,Integer) -> R
    ++ qelt(m,i,j) returns the element in the ith row and jth
    ++ column of the array m
    ++ NO error check to determine if indices are in proper ranges
    ++
    ++X arr : ARRAY2 INT := new(5,4,10)
    ++X qelt(arr,1,1)

   elt: (%,Integer,Integer,R) -> R
    ++ elt(m,i,j,r) returns the element in the ith row and jth
```

```
     ++ column of the array m, if m has an ith row and a jth column,
     ++ and returns r otherwise
     ++
     ++X arr : ARRAY2 INT := new(5,4,10)
     ++X elt(arr,1,1,6)
     ++X elt(arr,1,10,6)

    row: (%,Integer) -> Row
     ++ row(m,i) returns the ith row of m
     ++ error check to determine if index is in proper ranges
     ++
     ++X arr : ARRAY2 INT := new(5,4,10)
     ++X row(arr,1)

    column: (%,Integer) -> Col
     ++ column(m,j) returns the jth column of m
     ++ error check to determine if index is in proper ranges
     ++
     ++X arr : ARRAY2 INT := new(5,4,10)
     ++X column(arr,1)

    parts: % -> List R
     ++ parts(m) returns a list of the elements of m in row major order
     ++
     ++X arr : ARRAY2 INT := new(5,4,10)
     ++X parts(arr)

--% Part assignments

    setelt: (%,Integer,Integer,R) -> R
     -- will become setelt_!
     ++ setelt(m,i,j,r) sets the element in the ith row and jth
     ++ column of m to r
     ++ error check to determine if indices are in proper ranges
     ++
     ++X arr : ARRAY2 INT := new(5,4,0)
     ++X setelt(arr,1,1,17)

    qsetelt_!: (%,Integer,Integer,R) -> R
     ++ qsetelt!(m,i,j,r) sets the element in the ith row and jth
     ++ column of m to r
     ++ NO error check to determine if indices are in proper ranges
     ++
     ++X arr : ARRAY2 INT := new(5,4,0)
     ++X qsetelt!(arr,1,1,17)

    setRow_!: (%,Integer,Row) -> %
     ++ setRow!(m,i,v) sets to ith row of m to v
     ++
     ++X T1:=TwoDimensionalArray Integer
```

```
    ++X arr:T1:= new(5,4,0)
    ++X T2:=OneDimensionalArray Integer
    ++X arow:=construct([1,2,3,4]::List(INT))$T2
    ++X setRow!(arr,1,arow)$T1

   setColumn_!: (%,Integer,Col) -> %
    ++ setColumn!(m,j,v) sets to jth column of m to v
    ++
    ++X T1:=TwoDimensionalArray Integer
    ++X arr:T1:= new(5,4,0)
    ++X T2:=OneDimensionalArray Integer
    ++X acol:=construct([1,2,3,4,5]::List(INT))$T2
    ++X setColumn!(arr,1,acol)$T1

--% Map and Zip

   map: (R -> R,%) -> %
    ++ map(f,a) returns \spad{b}, where \spad{b(i,j) = f(a(i,j))}
    ++ for all \spad{i, j}
    ++
    ++X arr : ARRAY2 INT := new(5,4,10)
    ++X map(-,arr)
    ++X map((x +-> x + x),arr)

   map_!: (R -> R,%) -> %
    ++ map!(f,a)  assign \spad{a(i,j)} to \spad{f(a(i,j))}
    ++ for all \spad{i, j}
    ++X arr : ARRAY2 INT := new(5,4,10)
    ++X map!(-,arr)

   map:((R,R) -> R,%,%) -> %
    ++ map(f,a,b) returns \spad{c}, where \spad{c(i,j) = f(a(i,j),b(i,j))}
    ++ for all \spad{i, j}
    ++
    ++X adder(a:Integer,b:Integer):Integer == a+b
    ++X arr : ARRAY2 INT := new(5,4,10)
    ++X map(adder,arr,arr)

   map:((R,R) -> R,%,%,R) -> %
    ++ map(f,a,b,r) returns \spad{c}, where \spad{c(i,j) = f(a(i,j),b(i,j))}
    ++ when both \spad{a(i,j)} and \spad{b(i,j)} exist;
    ++ else \spad{c(i,j) = f(r, b(i,j))} when \spad{a(i,j)} does not exist;
    ++ else \spad{c(i,j) = f(a(i,j),r)} when \spad{b(i,j)} does not exist;
    ++ otherwise \spad{c(i,j) = f(r,r)}.
    ++
    ++X adder(a:Integer,b:Integer):Integer == a+b
    ++X arr1 : ARRAY2 INT := new(5,4,10)
    ++X arr2 : ARRAY2 INT := new(3,3,10)
    ++X map(adder,arr1,arr2,17)
```

```
  add

--% Predicates

    any?(f,m) ==
      for i in minRowIndex(m)..maxRowIndex(m) repeat
        for j in minColIndex(m)..maxColIndex(m) repeat
          f(qelt(m,i,j)) => return true
      false

    every?(f,m) ==
      for i in minRowIndex(m)..maxRowIndex(m) repeat
        for j in minColIndex(m)..maxColIndex(m) repeat
          not f(qelt(m,i,j)) => return false
      true

    size?(m,n) == nrows(m) * ncols(m) = n
    less?(m,n) == nrows(m) * ncols(m) < n
    more?(m,n) == nrows(m) * ncols(m) > n

--% Size inquiries

    # m == nrows(m) * ncols(m)

--% Part extractions

    elt(m,i,j,r) ==
      i < minRowIndex(m) or i > maxRowIndex(m) => r
      j < minColIndex(m) or j > maxColIndex(m) => r
      qelt(m,i,j)

    count(f:R -> Boolean,m:%) ==
      num : NonNegativeInteger := 0
      for i in minRowIndex(m)..maxRowIndex(m) repeat
        for j in minColIndex(m)..maxColIndex(m) repeat
          if f(qelt(m,i,j)) then num := num + 1
      num

    parts m ==
      entryList : List R := nil()
      for i in maxRowIndex(m)..minRowIndex(m) by -1 repeat
        for j in maxColIndex(m)..minColIndex(m) by -1 repeat
          entryList := concat(qelt(m,i,j),entryList)
      entryList

--% Creation

    copy m ==
      ans := new(nrows m,ncols m,NIL$Lisp)
      for i in minRowIndex(m)..maxRowIndex(m) repeat
```

```
    for j in minColIndex(m)..maxColIndex(m) repeat
      qsetelt_!(ans,i,j,qelt(m,i,j))
  ans

fill_!(m,r) ==
  for i in minRowIndex(m)..maxRowIndex(m) repeat
    for j in minColIndex(m)..maxColIndex(m) repeat
      qsetelt_!(m,i,j,r)
  m

map(f,m) ==
  ans := new(nrows m,ncols m,NIL$Lisp)
  for i in minRowIndex(m)..maxRowIndex(m) repeat
    for j in minColIndex(m)..maxColIndex(m) repeat
      qsetelt_!(ans,i,j,f(qelt(m,i,j)))
  ans

map_!(f,m) ==
  for i in minRowIndex(m)..maxRowIndex(m) repeat
    for j in minColIndex(m)..maxColIndex(m) repeat
      qsetelt_!(m,i,j,f(qelt(m,i,j)))
  m

map(f,m,n) ==
  (nrows(m) ^= nrows(n)) or (ncols(m) ^= ncols(n)) =>
    error "map: arguments must have same dimensions"
  ans := new(nrows m,ncols m,NIL$Lisp)
  for i in minRowIndex(m)..maxRowIndex(m) repeat
    for j in minColIndex(m)..maxColIndex(m) repeat
      qsetelt_!(ans,i,j,f(qelt(m,i,j),qelt(n,i,j)))
  ans

map(f,m,n,r) ==
  maxRow := max(maxRowIndex m,maxRowIndex n)
  maxCol := max(maxColIndex m,maxColIndex n)
  ans := new(max(nrows m,nrows n),max(ncols m,ncols n),NIL$Lisp)
  for i in minRowIndex(m)..maxRow repeat
    for j in minColIndex(m)..maxCol repeat
      qsetelt_!(ans,i,j,f(elt(m,i,j,r),elt(n,i,j,r)))
  ans

setRow_!(m,i,v) ==
  i < minRowIndex(m) or i > maxRowIndex(m) =>
    error "setRow!: index out of range"
  for j in minColIndex(m)..maxColIndex(m) _
    for k in minIndex(v)..maxIndex(v) repeat
      qsetelt_!(m,i,j,v.k)
  m

setColumn_!(m,j,v) ==
```

```
  j < minColIndex(m) or j > maxColIndex(m) =>
    error "setColumn!: index out of range"
  for i in minRowIndex(m)..maxRowIndex(m) _
    for k in minIndex(v)..maxIndex(v) repeat
      qsetelt_!(m,i,j,v.k)
  m

if R has _= : (R,R) -> Boolean then

  m = n ==
    eq?(m,n) => true
    (nrows(m) ^= nrows(n)) or (ncols(m) ^= ncols(n)) => false
    for i in minRowIndex(m)..maxRowIndex(m) repeat
      for j in minColIndex(m)..maxColIndex(m) repeat
        not (qelt(m,i,j) = qelt(n,i,j)) => return false
    true

  member?(r,m) ==
    for i in minRowIndex(m)..maxRowIndex(m) repeat
      for j in minColIndex(m)..maxColIndex(m) repeat
        qelt(m,i,j) = r => return true
    false

  count(r:R,m:%) == count(x +-> x = r,m)

if Row has shallowlyMutable then

  row(m,i) ==
    i < minRowIndex(m) or i > maxRowIndex(m) =>
      error "row: index out of range"
    v : Row := new(ncols m,NIL$Lisp)
    for j in minColIndex(m)..maxColIndex(m) _
      for k in minIndex(v)..maxIndex(v) repeat
        qsetelt_!(v,k,qelt(m,i,j))
    v

if Col has shallowlyMutable then

  column(m,j) ==
    j < minColIndex(m) or j > maxColIndex(m) =>
      error "column: index out of range"
    v : Col := new(nrows m,NIL$Lisp)
    for i in minRowIndex(m)..maxRowIndex(m) _
      for k in minIndex(v)..maxIndex(v) repeat
        qsetelt_!(v,k,qelt(m,i,j))
    v

if R has CoercibleTo(OutputForm) then

  coerce(m:%) ==
```

```
        l : List List OutputForm
        l := [[qelt(m,i,j) :: OutputForm _
                  for j in minColIndex(m)..maxColIndex(m)] _
                  for i in minRowIndex(m)..maxRowIndex(m)]
        matrix l
```

———-

— **ARR2CAT.dotabb** —

```
"ARR2CAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ARR2CAT"];
"ARR2CAT" -> "HOAGG"
```

———-

— **ARR2CAT.dotfull** —

```
"TwoDimensionalArrayCategory(a:Type,b:FiniteLinearAggregate(a),c:FiniteLinearAggregate(a))"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ARR2CAT"];
"TwoDimensionalArrayCategory(a:Type,b:FiniteLinearAggregate(a),c:FiniteLinearAggregate(a))"
    -> "HomogeneousAggregate(a:Type)"

"TwoDimensionalArrayCategory(a:Type,d:IndexedOneDimensionalArray(a,b),e:IndexedOneDimensionalArr
 [color=seagreen,href="bookvol10.2.pdf#nameddest=ARR2CAT"];
"TwoDimensionalArrayCategory(a:Type,d:IndexedOneDimensionalArray(a,b),e:IndexedOneDimensionalArr
-> "TwoDimensionalArrayCategory(a:Type,b:FiniteLinearAggregate(a),c:FiniteLinearAggregate(a))"
```

———-

— **ARR2CAT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"TwoDimensionalArrayCategory(a:Type,b:FiniteLinearAggregate(a),c:FiniteLinearAggregate(a))"
 [color=lightblue];
"TwoDimensionalArrayCategory(a:Type,b:FiniteLinearAggregate(a),c:FiniteLinearAggregate(a))"
    -> "HomogeneousAggregate(a:Type)"

"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "Aggregate()"
"HomogeneousAggregate(a:Type)" -> "Evalable(a:Type)"
```

```
"HomogeneousAggregate(a:Type)" -> "SetCategory()"

"Evalable(a:Type)" [color="#00EE00"];

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Aggregate()" [color=lightblue];
"Aggregate()" -> "Type()"

"Type()" [color=lightblue];
"Type()" -> "Category"

"Category" [color=lightblue];

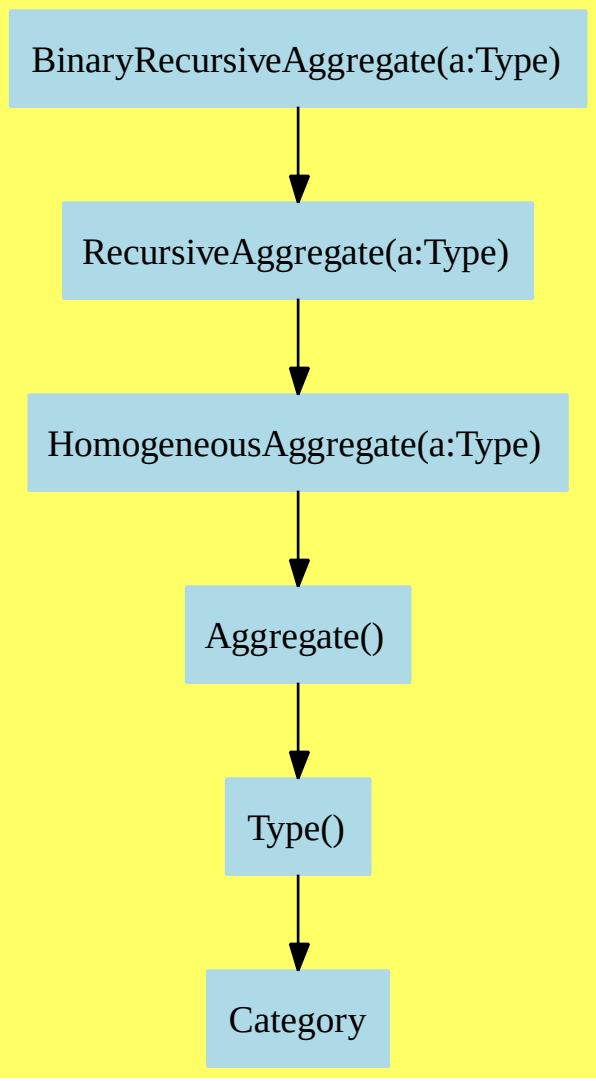
}
```

———————-

Chapter 6

Category Layer 5

6.0.82 BinaryRecursiveAggregate (BRAGG)



— BinaryRecursiveAggregate.input —

```
)set break resume
)sys rm -f BinaryRecursiveAggregate.output
)spool BinaryRecursiveAggregate.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show BinaryRecursiveAggregate
--R BinaryRecursiveAggregate S: Type  is a category constructor
--R Abbreviation for BinaryRecursiveAggregate is BRAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for BRAGG
--R
--R------------------------------ Operations --------------------------------
--R children : % -> List %               copy : % -> %
--R cyclic? : % -> Boolean               distance : (%,%) -> Integer
--R ?.right : (%,right) -> %             ?.left : (%,left) -> %
--R ?.value : (%,value) -> S             empty : () -> %
--R empty? : % -> Boolean                eq? : (%,%) -> Boolean
--R leaf? : % -> Boolean                 leaves : % -> List S
--R left : % -> %                        map : ((S -> S),%) -> %
--R nodes : % -> List %                  right : % -> %
--R sample : () -> %                     value : % -> S
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R child? : (%,%) -> Boolean if S has SETCAT
--R coerce : % -> OutputForm if S has SETCAT
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R hash : % -> SingleInteger if S has SETCAT
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R node? : (%,%) -> Boolean if S has SETCAT
--R parts : % -> List S if $ has finiteAggregate
```

```
--R setchildren! : (%,List %) -> % if $ has shallowlyMutable
--R setelt : (%,right,%) -> % if $ has shallowlyMutable
--R setelt : (%,left,%) -> % if $ has shallowlyMutable
--R setelt : (%,value,S) -> S if $ has shallowlyMutable
--R setleft! : (%,%) -> % if $ has shallowlyMutable
--R setright! : (%,%) -> % if $ has shallowlyMutable
--R setvalue! : (%,S) -> S if $ has shallowlyMutable
--R size? : (%,NonNegativeInteger) -> Boolean
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— BinaryRecursiveAggregate.help —

```
=====================================================================
BinaryRecursiveAggregate examples
=====================================================================

See Also:
o )show BinaryRecursiveAggregate
```

———-

See:

⇒ “BinaryTreeCategory” (BTCAT) 7.0.98 on page 526
⇐ “RecursiveAggregate” (RCAGG) 5.0.80 on page 351

Exports:

any?	children	child?	coerce	copy
count	cyclic?	distance	empty	empty?
eq?	eval	every?	hash	latex
leaf?	leaves	left	less?	map
map!	member?	members	more?	nodes
node?	parts	right	sample	setchildren!
setelt	setleft!	setright!	setvalue!	size?
value	#?	?=?	?~=?	?.right
?.left	?.value			

Attributes exported:

- **nil**

Attributes Used:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.

These are directly exported but not implemented:

```
left : % -> %
right : % -> %
setelt : (%,right,%) -> % if $ has shallowlyMutable
setelt : (%,left,%) -> % if $ has shallowlyMutable
setleft! : (%,%) -> % if $ has shallowlyMutable
setright! : (%,%) -> % if $ has shallowlyMutable
```

These are implemented by this category:

```
children : % -> List %
coerce : % -> OutputForm if S has SETCAT
cyclic? : % -> Boolean
leaf? : % -> Boolean
leaves : % -> List S
member? : (S,%) -> Boolean
         if S has SETCAT and $ has finiteAggregate
nodes : % -> List %
node? : (%,%) -> Boolean if S has SETCAT
#? : % -> NonNegativeInteger if $ has finiteAggregate
?=? : (%,%) -> Boolean if S has SETCAT
?.right : (%,right) -> %
?.left : (%,left) -> %
```

These exports come from (p351) RecursiveAggregate(S:Type)

```
any? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
child? : (%,%) -> Boolean if S has SETCAT
copy : % -> %
count : (S,%) -> NonNegativeInteger
         if S has SETCAT and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
distance : (%,%) -> Integer
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> %
         if S has EVALAB S and S has SETCAT
```

```
 eval : (%,S,S) -> %
          if S has EVALAB S and S has SETCAT
 eval : (%,Equation S) -> %
          if S has EVALAB S and S has SETCAT
 eval : (%,List Equation S) -> %
          if S has EVALAB S and S has SETCAT
 every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
 hash : % -> SingleInteger if S has SETCAT
 latex : % -> String if S has SETCAT
 less? : (%,NonNegativeInteger) -> Boolean
 map : ((S -> S),%) -> %
 map! : ((S -> S),%) -> % if $ has shallowlyMutable
 members : % -> List S if $ has finiteAggregate
 more? : (%,NonNegativeInteger) -> Boolean
 parts : % -> List S if $ has finiteAggregate
 sample : () -> %
 setchildren! : (%,List %) -> % if $ has shallowlyMutable
 setelt : (%,value,S) -> S if $ has shallowlyMutable
 setvalue! : (%,S) -> S if $ has shallowlyMutable
 size? : (%,NonNegativeInteger) -> Boolean
 value : % -> S
 ?~=? : (%,%) -> Boolean if S has SETCAT
 ?.value : (%,value) -> S
```

— category BRAGG BinaryRecursiveAggregate —

```
)abbrev category BRAGG BinaryRecursiveAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A binary-recursive aggregate has 0, 1 or 2 children and serves
++ as a model for a binary tree or a doubly-linked aggregate structure

BinaryRecursiveAggregate(S:Type):Category == RecursiveAggregate S with
   -- needs preorder, inorder and postorder iterators
   left: % -> %
     ++ left(u) returns the left child.
   elt: (%,"left") -> %
     ++ elt(u,"left") (also written: \axiom{a . left}) is
     ++ equivalent to \axiom{left(a)}.
   right: % -> %
     ++ right(a) returns the right child.
```

```
  elt: (%,"right") -> %
    ++ elt(a,"right") (also written: \axiom{a . right})
    ++ is equivalent to \axiom{right(a)}.
  if % has shallowlyMutable then
     setelt: (%,"left",%) -> %
       ++ setelt(a,"left",b) (also written \axiom{a . left := b}) is
       ++ equivalent to \axiom{setleft!(a,b)}.
     setleft_!: (%,%) -> %
        ++ setleft!(a,b) sets the left child of \axiom{a} to be b.
     setelt: (%,"right",%) -> %
        ++ setelt(a,"right",b) (also written \axiom{b . right := b})
        ++ is equivalent to \axiom{setright!(a,b)}.
     setright_!: (%,%) -> %
        ++ setright!(a,x) sets the right child of t to be x.
add
  cycleMax ==> 1000

  elt(x,"left")  == left x
  elt(x,"right") == right x
  leaf? x == empty? x or empty? left x and empty? right x
  leaves t ==
    empty? t => empty()$List(S)
    leaf? t => [value t]
    concat(leaves left t,leaves right t)
  nodes x ==
    l := empty()$List(%)
    empty? x => l
    concat(nodes left x,concat([x],nodes right x))
  children x ==
    l := empty()$List(%)
    empty? x => l
    empty? left x  => [right x]
    empty? right x => [left x]
    [left x, right x]
  if % has SetAggregate(S) and S has SetCategory then
    node?(u,v) ==
      empty? v => false
      u = v => true
      for y in children v repeat node?(u,y) => return true
      false
    x = y ==
      empty?(x) => empty?(y)
      empty?(y) => false
      value x = value y and left x = left y and right x = right y
    if % has finiteAggregate then
      member?(x,u) ==
        empty? u => false
        x = value u => true
        member?(x,left u) or member?(x,right u)
```

```
  if S has SetCategory then
    coerce(t:%): OutputForm ==
      empty? t =>  "[]"::OutputForm
      v := value(t):: OutputForm
      empty? left t =>
        empty? right t => v
        r := coerce(right t)@OutputForm
        bracket ["."::OutputForm, v, r]
      l := coerce(left t)@OutputForm
      r :=
        empty? right t => "."::OutputForm
        coerce(right t)@OutputForm
      bracket [l, v, r]

  if % has finiteAggregate then
    aggCount: (%,NonNegativeInteger) -> NonNegativeInteger
    #x == aggCount(x,0)
    aggCount(x,k) ==
      empty? x => 0
      k := k + 1
      k = cycleMax and cyclic? x => error "cyclic tree"
      for y in children x repeat k := aggCount(y,k)
      k

  isCycle?:  (%, List %) -> Boolean
  eqMember?: (%, List %) -> Boolean
  cyclic? x         == not empty? x and isCycle?(x,empty()$(List %))
  isCycle?(x,acc) ==
    empty? x => false
    eqMember?(x,acc) => true
    for y in children x | not empty? y repeat
      isCycle?(y,acc) => return true
    false
  eqMember?(y,l) ==
    for x in l repeat eq?(x,y) => return true
    false
  if % has shallowlyMutable then
    setelt(x,"left",b)  == setleft_!(x,b)
    setelt(x,"right",b) == setright_!(x,b)
```

———-

— BRAGG.dotabb —

```
"BRAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=BRAGG"];
"BRAGG" -> "RCAGG"
```

———-

— **BRAGG.dotfull** —

```
"BinaryRecursiveAggregate(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=BRAGG"];
"BinaryRecursiveAggregate(a:Type)" -> "RecursiveAggregate(a:Type)"

"BinaryRecursiveAggregate(a:SetCategory)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=BRAGG"];
"BinaryRecursiveAggregate(a:SetCategory)" ->
   "BinaryRecursiveAggregate(a:Type)"
```

———-

— **BRAGG.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"BinaryRecursiveAggregate(a:Type)" [color=lightblue];
"BinaryRecursiveAggregate(a:Type)" -> "RecursiveAggregate(a:Type)"

"RecursiveAggregate(a:Type)" [color=lightblue];
"RecursiveAggregate(a:Type)" -> "HomogeneousAggregate(a:Type)"

"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "Aggregate()"

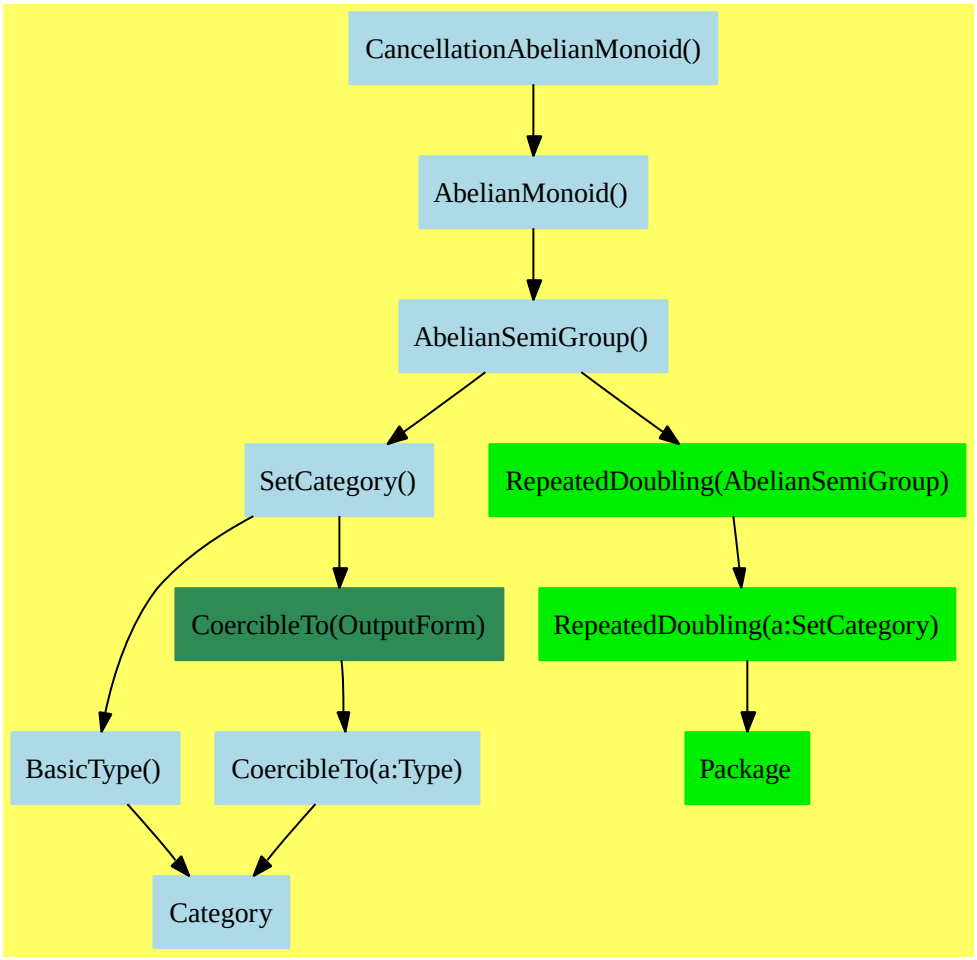
"Aggregate()" [color=lightblue];
"Aggregate()" -> "Type()"

"Type()" [color=lightblue];
"Type()" -> "Category"

"Category" [color=lightblue];
}
```

———-

6.0.83 CancellationAbelianMonoid (CABMON)



— CancellationAbelianMonoid.input —

```
)set break resume
)sys rm -f CancellationAbelianMonoid.output
)spool CancellationAbelianMonoid.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show CancellationAbelianMonoid
--R CancellationAbelianMonoid  is a category constructor
```

```
--R Abbreviation for CancellationAbelianMonoid is CABMON
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for CABMON
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (PositiveInteger,%) -> %        ?+? : (%,%) -> %
--R ?=? : (%,%) -> Boolean                0 : () -> %
--R coerce : % -> OutputForm              hash : % -> SingleInteger
--R latex : % -> String                   sample : () -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— CancellationAbelianMonoid.help —

```
=====================================================================
CancellationAbelianMonoid examples
=====================================================================

See Also:
o )show CancellationAbelianMonoid
```

———-

See:

⇒ “AbelianGroup” (ABELGRP) 7.0.97 on page 520
⇒ “FreeAbelianMonoidCategory” (FAMONC) 7.0.103 on page 564
⇒ “OrderedCancellationAbelianMonoid” (OCAMON) 8.0.118 on page 696
⇐ “AbelianMonoid” (ABELMON) 5.0.66 on page 267

Exports:

0	coerce	hash	latex	sample
subtractIfCan	zero?	?~=?	?*?	?+?
?=?				

These are directly exported but not implemented:

```
subtractIfCan : (%,%) -> Union(%,"failed")
```

These exports come from (p267) AbelianMonoid():

```
 0 : () -> %
 coerce : % -> OutputForm
 hash : % -> SingleInteger
 latex : % -> String
 sample : () -> %
 zero? : % -> Boolean
 ?*? : (NonNegativeInteger,%) -> %
 ?*? : (PositiveInteger,%) -> %
 ?~=? : (%,%) -> Boolean
 ?+? : (%,%) -> %
 ?=? : (%,%) -> Boolean
```

— category CABMON CancellationAbelianMonoid —

```
)abbrev category CABMON CancellationAbelianMonoid
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References: Davenport & Trager I
++ Description:
++ This is an \spadtype{AbelianMonoid} with the cancellation property, i.e.\br
++ \tab{5}\spad{ a+b = a+c => b=c }.\br
++ This is formalised by the partial subtraction operator,
++ which satisfies the  Axioms\br
++ \tab{5}\spad{c = a+b <=> c-b = a}

CancellationAbelianMonoid(): Category == AbelianMonoid with
      subtractIfCan: (%,%) -> Union(%,"failed")
         ++ subtractIfCan(x, y) returns an element z such that \spad{z+y=x}
         ++ or "failed" if no such element exists.
```

— CABMON.dotabb —

```
"CABMON"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=CABMON"];
"CABMON" -> "ABELMON"
```

— **CABMON.dotfull** —

```
"CancellationAbelianMonoid()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=CABMON"];
"CancellationAbelianMonoid()" -> "AbelianMonoid()"
```

———-

— **CABMON.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"CancellationAbelianMonoid()" [color=lightblue];
"CancellationAbelianMonoid()" -> "AbelianMonoid()"

"AbelianMonoid()" [color=lightblue];
"AbelianMonoid()" -> "AbelianSemiGroup()"

"AbelianSemiGroup()" [color=lightblue];
"AbelianSemiGroup()" -> "SetCategory()"
"AbelianSemiGroup()" -> "RepeatedDoubling(AbelianSemiGroup)"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" ->
    "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

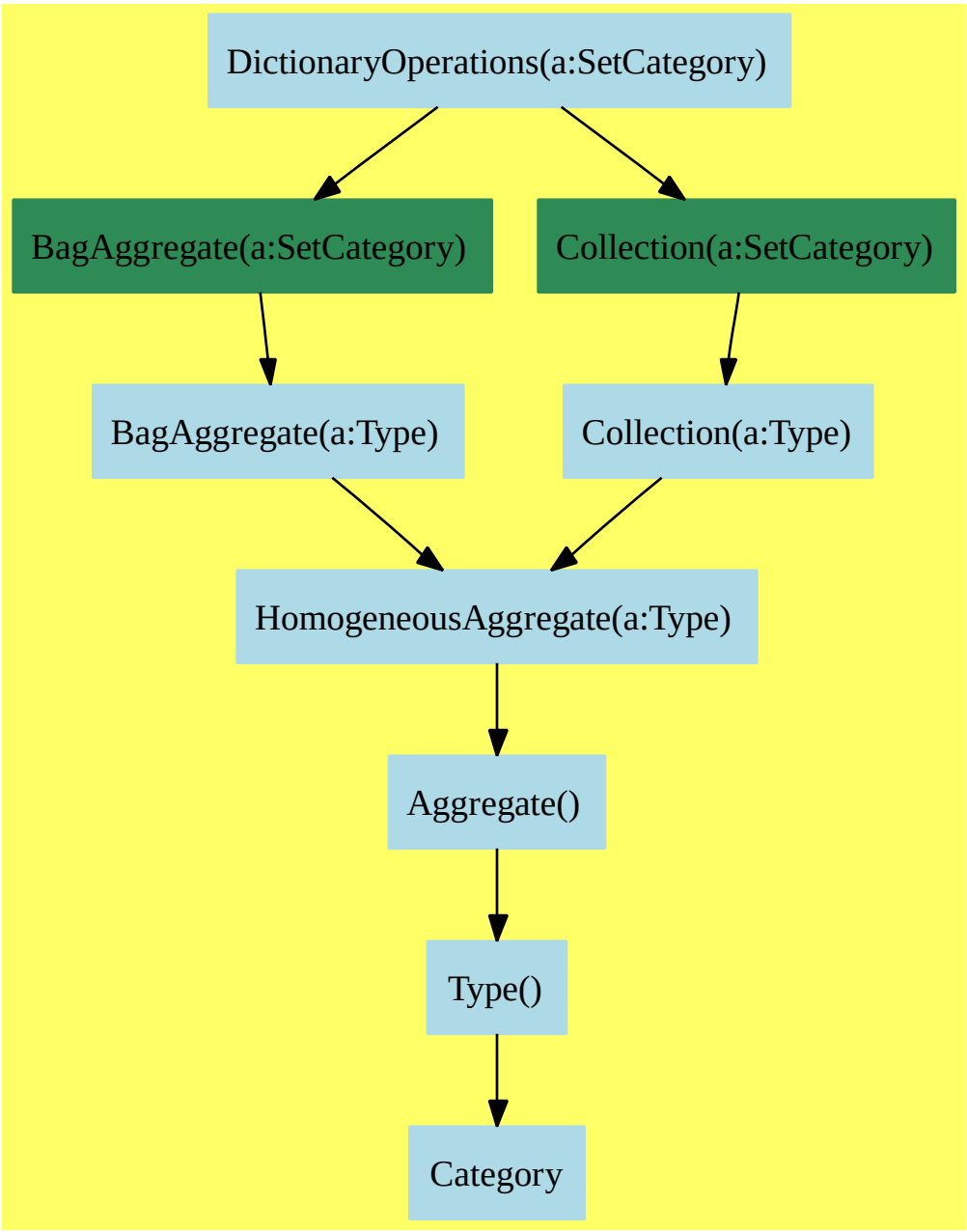
"RepeatedDoubling(AbelianSemiGroup)" [color="#00EE00"];
"RepeatedDoubling(AbelianSemiGroup)" -> "RepeatedDoubling(a:SetCategory)"

"RepeatedDoubling(a:SetCategory)" [color="#00EE00"];
"RepeatedDoubling(a:SetCategory)" -> "Package"

"Package" [color="#00EE00"];
```

```
"Category" [color=lightblue];
}
```

6.0.84 DictionaryOperations (DIOPS)



— **DictionaryOperations.input** —

```
)set break resume
)sys rm -f DictionaryOperations.output
)spool DictionaryOperations.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show DictionaryOperations
--R DictionaryOperations S: SetCategory  is a category constructor
--R Abbreviation for DictionaryOperations is DIOPS
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for DIOPS
--R
--R------------------------------ Operations ------------------------------
--R bag : List S -> %                    construct : List S -> %
--R copy : % -> %                        dictionary : List S -> %
--R dictionary : () -> %                 empty : () -> %
--R empty? : % -> Boolean                eq? : (%,%) -> Boolean
--R extract! : % -> S                    insert! : (S,%) -> %
--R inspect : % -> S                     map : ((S -> S),%) -> %
--R sample : () -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R coerce : % -> OutputForm if S has SETCAT
--R convert : % -> InputForm if S has KONVERT INFORM
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R find : ((S -> Boolean),%) -> Union(S,"failed")
--R hash : % -> SingleInteger if S has SETCAT
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S,S) -> S if S has SETCAT and $ has finiteAggregate
--R remove : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (S,%) -> % if S has SETCAT and $ has finiteAggregate
--R remove! : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R remove! : (S,%) -> % if $ has finiteAggregate
```

```
--R removeDuplicates : % -> % if S has SETCAT and $ has finiteAggregate
--R select : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R select! : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R size? : (%,NonNegativeInteger) -> Boolean
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— **DictionaryOperations.help** —

```
=====================================================================
DictionaryOperations examples
=====================================================================

See Also:
o )show DictionaryOperations
```

———-

See:

⇒ “Dictionary” (DIAGG) 7.0.99 on page 533
⇒ “MultiDictionary” (MDAGG) 7.0.104 on page 570
⇐ “BagAggregate” (BGAGG) 5.0.68 on page 277
⇐ “Collection” (CLAGG) 5.0.70 on page 287

Exports:

any?	bag	coerce	construct	convert
copy	count	dictionary	empty	empty?
eq?	eval	every?	extract!	find
hash	insert!	inspect	latex	less?
map	map!	member?	members	more?
parts	reduce	remove	remove!	removeDuplicates
sample	select	select!	size?	#?
?=?	?~=?			

Attributes exported:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.

- **nil**

Attributes Used:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.

These are directly exported but not implemented:

```
dictionary : List S -> %
remove! : ((S -> Boolean),%) -> % if $ has finiteAggregate
remove! : (S,%) -> % if $ has finiteAggregate
select! : ((S -> Boolean),%) -> % if $ has finiteAggregate
```

These are implemented by this category:

```
coerce : % -> OutputForm if S has SETCAT
construct : List S -> %
copy : % -> %
dictionary : () -> %
```

These exports come from (p277) BagAggregate(S:SetCategory):

```
any? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
bag : List S -> %
count : (S,%) -> NonNegativeInteger
         if S has SETCAT and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,S,S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,Equation S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,List Equation S) -> %
         if S has EVALAB S and S has SETCAT
every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
extract! : % -> S
hash : % -> SingleInteger if S has SETCAT
insert! : (S,%) -> %
inspect : % -> S
latex : % -> String if S has SETCAT
less? : (%,NonNegativeInteger) -> Boolean
map : ((S -> S),%) -> %
map! : ((S -> S),%) -> % if $ has shallowlyMutable
```

```
member? : (S,%) -> Boolean
         if S has SETCAT and $ has finiteAggregate
members : % -> List S if $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
parts : % -> List S if $ has finiteAggregate
sample : () -> %
size? : (%,NonNegativeInteger) -> Boolean
#? : % -> NonNegativeInteger if $ has finiteAggregate
?=? : (%,%) -> Boolean if S has SETCAT
?~=? : (%,%) -> Boolean if S has SETCAT
```

These exports come from (p287) Collection(S:SetCategory)

```
convert : % -> InputForm if S has KONVERT INFORM
find : ((S -> Boolean),%) -> Union(S,"failed")
reduce : (((S,S) -> S),%) -> S if $ has finiteAggregate
reduce : (((S,S) -> S),%,S) -> S if $ has finiteAggregate
reduce : (((S,S) -> S),%,S,S) -> S
                if S has SETCAT and $ has finiteAggregate
remove : ((S -> Boolean),%) -> % if $ has finiteAggregate
remove : (S,%) -> % if S has SETCAT and $ has finiteAggregate
removeDuplicates : % -> %
         if S has SETCAT and $ has finiteAggregate
select : ((S -> Boolean),%) -> % if $ has finiteAggregate
```

— category DIOPS DictionaryOperations —

```
)abbrev category DIOPS DictionaryOperations
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ This category is a collection of operations common to both
++ categories \spadtype{Dictionary} and \spadtype{MultiDictionary}

DictionaryOperations(S:SetCategory): Category ==
  Join(BagAggregate S, Collection(S)) with
   dictionary: () -> %
     ++ dictionary()$D creates an empty dictionary of type D.
   dictionary: List S -> %
     ++ dictionary([x,y,...,z]) creates a dictionary consisting of
     ++ entries \axiom{x,y,...,z}.
-- insert: (S,%) -> S                    ++ insert an entry
```

```
-- member?: (S,%) -> Boolean                      ++ search for an entry
-- remove_!: (S,%,NonNegativeInteger) -> %
--   ++ remove!(x,d,n) destructively changes dictionary d by removing
--   ++ up to n entries y such that \axiom{y = x}.
-- remove_!: (S->Boolean,%,NonNegativeInteger) -> %
--   ++ remove!(p,d,n) destructively changes dictionary d by removing
--   ++ up to n entries x such that \axiom{p(x)} is true.
   if % has finiteAggregate then
     remove_!: (S,%) -> %
       ++ remove!(x,d) destructively changes dictionary d by removing
       ++ all entries y such that \axiom{y = x}.
     remove_!: (S->Boolean,%) -> %
       ++ remove!(p,d) destructively changes dictionary d by removeing
       ++ all entries x such that \axiom{p(x)} is true.
     select_!: (S->Boolean,%) -> %
       ++ select!(p,d) destructively changes dictionary d by removing
       ++ all entries x such that \axiom{p(x)} is not true.
 add
   construct l == dictionary l
   dictionary() == empty()
   if % has finiteAggregate then
     copy d == dictionary parts d
     coerce(s:%):OutputForm ==
       prefix("dictionary"@String :: OutputForm,
                                    [x::OutputForm for x in parts s])
```

———-

— DIOPS.dotabb —

```
"DIOPS" [color=lightblue,href="bookvol10.2.pdf#nameddest=DIOPS"];
"DIOPS" -> "BGAGG"
"DIOPS" -> "CLAGG"
```

———-

— DIOPS.dotfull —

```
"DictionaryOperations(a:SetCategory)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=DIOPS"];
"DictionaryOperations(a:SetCategory)" -> "BagAggregate(a:SetCategory)"
"DictionaryOperations(a:SetCategory)" -> "Collection(a:SetCategory)"
```

———-

— DIOPS.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"DictionaryOperations(a:SetCategory)" [color=lightblue];
"DictionaryOperations(a:SetCategory)" -> "BagAggregate(a:SetCategory)"
"DictionaryOperations(a:SetCategory)" -> "Collection(a:SetCategory)"

"BagAggregate(a:SetCategory)" [color=seagreen];
"BagAggregate(a:SetCategory)" -> "BagAggregate(a:Type)"

"BagAggregate(a:Type)" [color=lightblue];
"BagAggregate(a:Type)" -> "HomogeneousAggregate(a:Type)"

"Collection(a:SetCategory)" [color=seagreen];
"Collection(a:SetCategory)" -> "Collection(a:Type)"

"Collection(a:Type)" [color=lightblue];
"Collection(a:Type)" -> "HomogeneousAggregate(a:Type)"

"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "Aggregate()"

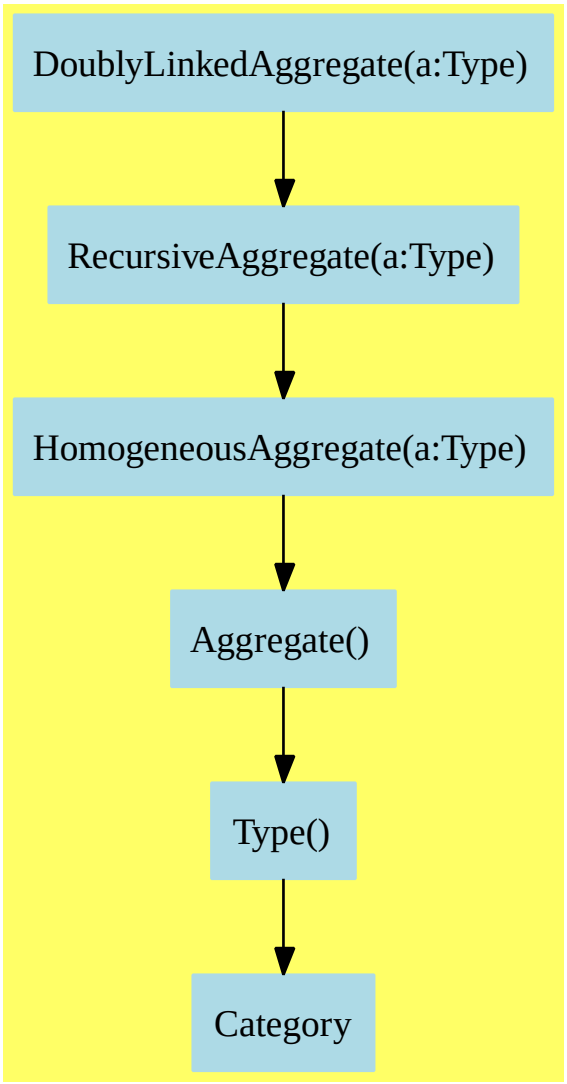
"Aggregate()" [color=lightblue];
"Aggregate()" -> "Type()"

"Type()" [color=lightblue];
"Type()" -> "Category"

"Category" [color=lightblue];
}
```

———-

6.0.85 DoublyLinkedAggregate (DLAGG)



— DoublyLinkedAggregate.input —

```
)set break resume
)sys rm -f DoublyLinkedAggregate.output
)spool DoublyLinkedAggregate.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 1
)show DoublyLinkedAggregate
--R DoublyLinkedAggregate S: Type  is a category constructor
--R Abbreviation for DoublyLinkedAggregate is DLAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for DLAGG
--R
--R------------------------------ Operations --------------------------------
--R children : % -> List %               copy : % -> %
--R cyclic? : % -> Boolean               distance : (%,%) -> Integer
--R ?.value : (%,value) -> S             empty : () -> %
--R empty? : % -> Boolean                eq? : (%,%) -> Boolean
--R head : % -> %                        last : % -> S
--R leaf? : % -> Boolean                 leaves : % -> List S
--R map : ((S -> S),%) -> %              next : % -> %
--R nodes : % -> List %                  previous : % -> %
--R sample : () -> %                     tail : % -> %
--R value : % -> S
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R child? : (%,%) -> Boolean if S has SETCAT
--R coerce : % -> OutputForm if S has SETCAT
--R concat! : (%,%) -> % if $ has shallowlyMutable
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R hash : % -> SingleInteger if S has SETCAT
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R node? : (%,%) -> Boolean if S has SETCAT
--R parts : % -> List S if $ has finiteAggregate
--R setchildren! : (%,List %) -> % if $ has shallowlyMutable
--R setelt : (%,value,S) -> S if $ has shallowlyMutable
--R setnext! : (%,%) -> % if $ has shallowlyMutable
--R setprevious! : (%,%) -> % if $ has shallowlyMutable
--R setvalue! : (%,S) -> S if $ has shallowlyMutable
--R size? : (%,NonNegativeInteger) -> Boolean
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1
```

```
)spool
)lisp (bye)
```

———-

— **DoublyLinkedAggregate.help** —

```
====================================================================
DoublyLinkedAggregate examples
====================================================================

See Also:
o )show DoublyLinkedAggregate
```

———-

See:

⇐ “RecursiveAggregate” (RCAGG) 5.0.80 on page 351

Exports:

any?	children	child?	coerce	concat!
copy	count	cyclic?	distance	empty
empty?	eq?	eval	every?	hash
head	last	latex	leaf?	leaves
less?	map	map!	member?	members
more?	next	nodes	node?	parts
previous	sample	setchildren!	setelt	setnext!
setprevious!	setvalue!	size?	tail	value
#?	?=?	?~=?	?.value	

Attributes exported:

- **nil**

Attributes Used:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.

These are directly exported but not implemented:

```
concat! : (%,%) -> % if $ has shallowlyMutable
head : % -> %
```

```
last : % -> S
next : % -> %
previous : % -> %
setnext! : (%,%) -> % if $ has shallowlyMutable
setprevious! : (%,%) -> % if $ has shallowlyMutable
tail : % -> %
```

These exports come from (p351) RecursiveAggregate(S:Type):

```
any? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
children : % -> List %
child? : (%,%) -> Boolean if S has SETCAT
coerce : % -> OutputForm if S has SETCAT
copy : % -> %
count : (S,%) -> NonNegativeInteger
         if S has SETCAT and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
cyclic? : % -> Boolean
distance : (%,%) -> Integer
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,S,S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,Equation S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,List Equation S) -> %
         if S has EVALAB S and S has SETCAT
every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
hash : % -> SingleInteger if S has SETCAT
latex : % -> String if S has SETCAT
leaf? : % -> Boolean
leaves : % -> List S
less? : (%,NonNegativeInteger) -> Boolean
map : ((S -> S),%) -> %
map! : ((S -> S),%) -> % if $ has shallowlyMutable
member? : (S,%) -> Boolean
         if S has SETCAT and $ has finiteAggregate
members : % -> List S if $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
nodes : % -> List %
node? : (%,%) -> Boolean if S has SETCAT
parts : % -> List S if $ has finiteAggregate
sample : () -> %
setchildren! : (%,List %) -> % if $ has shallowlyMutable
setelt : (%,value,S) -> S if $ has shallowlyMutable
```

```
 setvalue! : (%,S) -> S if $ has shallowlyMutable
 size? : (%,NonNegativeInteger) -> Boolean
 value : % -> S
 #? : % -> NonNegativeInteger if $ has finiteAggregate
 ?=? : (%,%) -> Boolean if S has SETCAT
 ?~=? : (%,%) -> Boolean if S has SETCAT
 ?.value : (%,value) -> S
```

— category DLAGG DoublyLinkedAggregate —

```
)abbrev category DLAGG DoublyLinkedAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A doubly-linked aggregate serves as a model for a doubly-linked
++ list, that is, a list which can has links to both next and previous
++ nodes and thus can be efficiently traversed in both directions.

DoublyLinkedAggregate(S:Type): Category == RecursiveAggregate S with
   last: % -> S
     ++ last(l) returns the last element of a doubly-linked aggregate l.
     ++ Error: if l is empty.
   head: % -> %
     ++ head(l) returns the first element of a doubly-linked aggregate l.
     ++ Error: if l is empty.
   tail: % -> %
     ++ tail(l) returns the doubly-linked aggregate l starting at
     ++ its second element.
     ++ Error: if l is empty.
   previous: % -> %
     ++ previous(l) returns the doubly-link list beginning with its previous
     ++ element.
     ++ Error: if l has no previous element.
     ++ Note that \axiom{next(previous(l)) = l}.
   next: % -> %
     ++ next(l) returns the doubly-linked aggregate beginning with its next
     ++ element.
     ++ Error: if l has no next element.
     ++ Note that \axiom{next(l) = rest(l)} and \axiom{previous(next(l)) = l}.
   if % has shallowlyMutable then
      concat_!: (%,%) -> %
        ++ concat!(u,v) destructively concatenates doubly-linked aggregate v
```

```
        ++ to the end of doubly-linked aggregate u.
      setprevious_!: (%,%) -> %
        ++ setprevious!(u,v) destructively sets the previous node of
        ++ doubly-linked aggregate u to v, returning v.
      setnext_!: (%,%) -> %
        ++ setnext!(u,v) destructively sets the next node of doubly-linked
        ++ aggregate u to v, returning v.
```

———-

— **DLAGG.dotabb** —

```
"DLAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=DLAGG"];
"DLAGG" -> "RCAGG"
```

———-

— **DLAGG.dotfull** —

```
"DoublyLinkedAggregate(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=DLAGG"];
"DoublyLinkedAggregate(a:Type)" -> "RecursiveAggregate(a:Type)"
```

———-

— **DLAGG.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"DoublyLinkedAggregate(a:Type)" [color=lightblue];
"DoublyLinkedAggregate(a:Type)" -> "RecursiveAggregate(a:Type)"

"RecursiveAggregate(a:Type)" [color=lightblue];
"RecursiveAggregate(a:Type)" -> "HomogeneousAggregate(a:Type)"

"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "Aggregate()"

"Aggregate()" [color=lightblue];
"Aggregate()" -> "Type()"

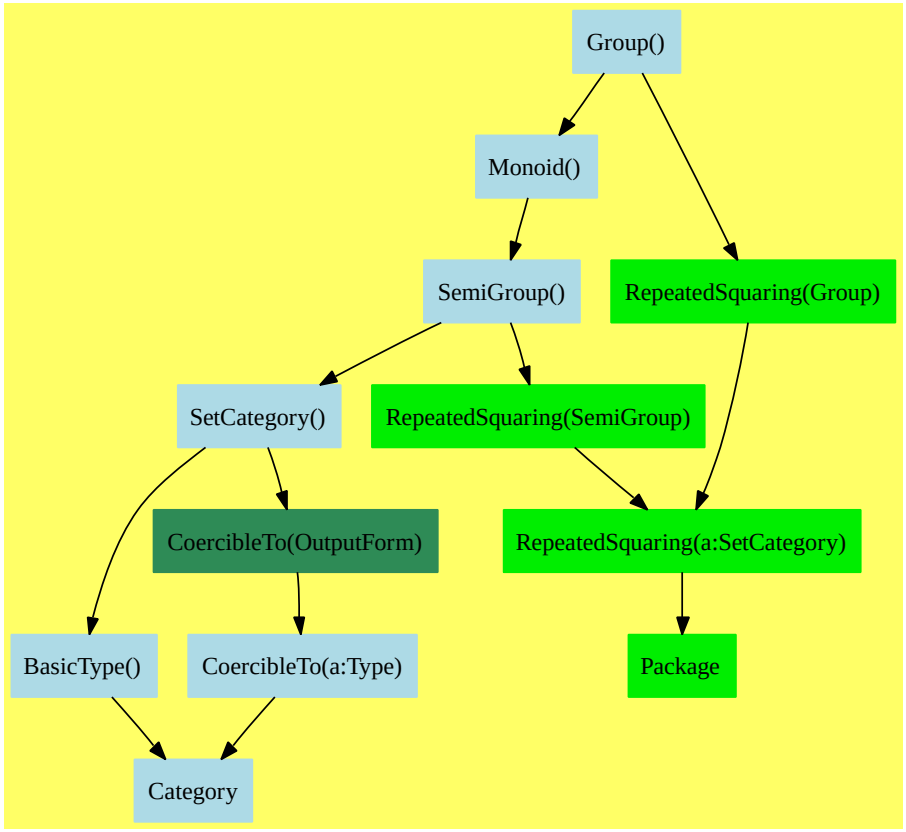
"Type()" [color=lightblue];
```

```
"Type()" -> "Category"

"Category" [color=lightblue];
}
```

———-

6.0.86 Group (GROUP)



— Group.input —

```
)set break resume
)sys rm -f Group.output
)spool Group.output
)set message test on
```

```
)set message auto off
)clear all

--S 1 of 1
)show Group
--R Group  is a category constructor
--R Abbreviation for Group is GROUP
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for GROUP
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,%) -> %                      ?**? : (%,Integer) -> %
--R ?**? : (%,PositiveInteger) -> %       ?/? : (%,%) -> %
--R ?=? : (%,%) -> Boolean                1 : () -> %
--R ?^? : (%,Integer) -> %                ?^? : (%,PositiveInteger) -> %
--R coerce : % -> OutputForm              commutator : (%,%) -> %
--R conjugate : (%,%) -> %                hash : % -> SingleInteger
--R inv : % -> %                          latex : % -> String
--R one? : % -> Boolean                   recip : % -> Union(%,"failed")
--R sample : () -> %                      ?~=? : (%,%) -> Boolean
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— Group.help —

```
========================================================================
Group examples
========================================================================

See Also:
o )show Group
```

———-

See:

⇒ “FunctionSpace” (FS) 17.0.195 on page 1336
⇒ “PermutationCategory” (PERMCAT) 7.0.106 on page 580
⇐ “Monoid” (MONOID) 5.0.76 on page 331

Exports:

1	coerce	commutator	conjugate	hash
inv	latex	one?	recip	sample
?~=?	?*?	?**?	?/?	?=?
?^?				

Attributes Exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.

These are directly exported but not implemented:

```
inv : % -> %
```

These are implemented by this category:

```
commutator : (%,%) -> %
conjugate : (%,%) -> %
recip : % -> Union(%,"failed")
?/? : (%,%) -> %
?^? : (%,Integer) -> %
?**? : (%,Integer) -> %
```

These exports come from (p331) Monoid():

```
1 : () -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
sample : () -> %
?^? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
?*? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
```

— category GROUP Group —

```
)abbrev category GROUP Group
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
```

```
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The class of multiplicative groups, i.e. monoids with
++ multiplicative inverses.
++
++ Axioms\br
++ \tab{5}\spad{leftInverse("*":(%,%)->%,inv)}\tab{5}\spad{ inv(x)*x = 1 }\br
++ \tab{5}\spad{rightInverse("*":(%,%)->%,inv)}\tab{4}\spad{ x*inv(x) = 1 }

Group(): Category == Monoid with
      inv: % -> %
        ++ inv(x) returns the inverse of x.
      "/": (%,%) -> %
        ++ x/y is the same as x times the inverse of y.
      "**": (%,Integer) -> %
        ++ x**n returns x raised to the integer power n.
      "^": (%,Integer) -> %
        ++ x^n returns x raised to the integer power n.
      unitsKnown
        ++ unitsKnown asserts that recip only returns
        ++ "failed" for non-units.
      conjugate: (%,%) -> %
        ++ conjugate(p,q) computes \spad{inv(q) * p * q}; this is
        ++ 'right action by conjugation'.
      commutator: (%,%) -> %
        ++ commutator(p,q) computes \spad{inv(p) * inv(q) * p * q}.
    add
      import RepeatedSquaring(%)
      x:% / y:% == x*inv(y)
      recip(x:%) == inv(x)
      _^(x:%, n:Integer):% == x ** n
      x:% ** n:Integer ==
         zero? n => 1
         n<0 => expt(inv(x),(-n) pretend PositiveInteger)
         expt(x,n pretend PositiveInteger)
      conjugate(p,q) == inv(q) * p * q
      commutator(p,q) == inv(p) * inv(q) * p * q
```

———-

— GROUP.dotabb —

```
"GROUP"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=GROUP"];
"GROUP" -> "MONOID"
```

———-

— **GROUP.dotfull** —

```
"Group()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=GROUP"];
"Group()" -> "Monoid()"
```

———-

— **GROUP.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"Group()" [color=lightblue];
"Group()" -> "Monoid()"
"Group()" -> "RepeatedSquaring(Group)"

"Monoid()" [color=lightblue];
"Monoid()" -> "SemiGroup()"

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SetCategory()"
"SemiGroup()" -> "RepeatedSquaring(SemiGroup)"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"RepeatedSquaring(Group)" [color="#00EE00"];
"RepeatedSquaring(Group)" -> "RepeatedSquaring(a:SetCategory)"

"RepeatedSquaring(SemiGroup)" [color="#00EE00"];
"RepeatedSquaring(SemiGroup)" -> "RepeatedSquaring(a:SetCategory)"

"RepeatedSquaring(a:SetCategory)" [color="#00EE00"];
```

```
"RepeatedSquaring(a:SetCategory)" -> "Package"

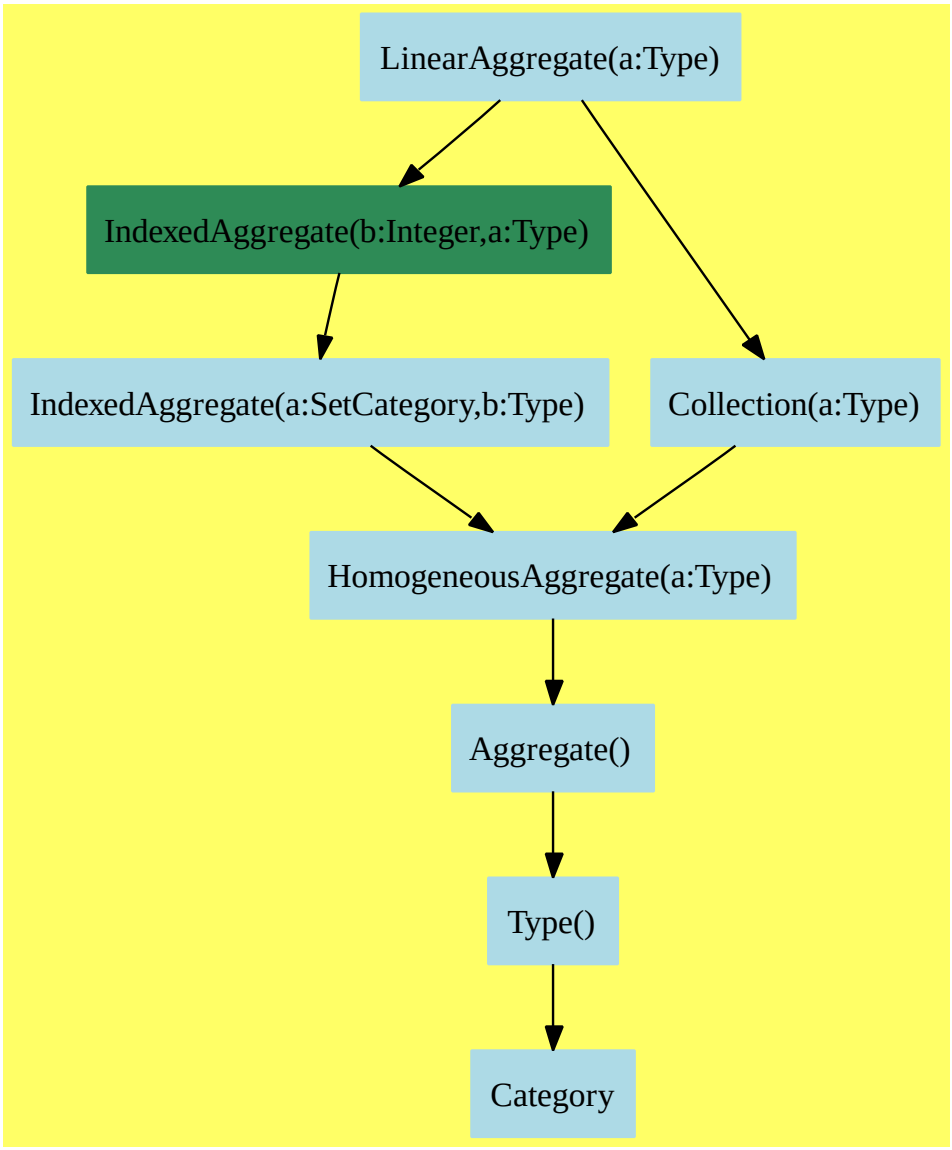
"Package" [color="#00EE00"];

"Category" [color=lightblue];

}
```

———————

6.0.87 LinearAggregate (LNAGG)



— **LinearAggregate.input** —

```
)set break resume
)sys rm -f LinearAggregate.output
)spool LinearAggregate.output
)set message test on
```

```
)set message auto off
)clear all

--S 1 of 1
)show LinearAggregate
--R LinearAggregate S: Type  is a category constructor
--R Abbreviation for LinearAggregate is LNAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for LNAGG
--R
--R------------------------------ Operations --------------------------------
--R concat : List % -> %                  concat : (%,%) -> %
--R concat : (S,%) -> %                   concat : (%,S) -> %
--R construct : List S -> %               copy : % -> %
--R delete : (%,Integer) -> %             ?.? : (%,Integer) -> S
--R elt : (%,Integer,S) -> S              empty : () -> %
--R empty? : % -> Boolean                 entries : % -> List S
--R eq? : (%,%) -> Boolean                index? : (Integer,%) -> Boolean
--R indices : % -> List Integer           insert : (%,%,Integer) -> %
--R insert : (S,%,Integer) -> %           map : (((S,S) -> S),%,%) -> %
--R map : ((S -> S),%) -> %               new : (NonNegativeInteger,S) -> %
--R qelt : (%,Integer) -> S               sample : () -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R coerce : % -> OutputForm if S has SETCAT
--R convert : % -> InputForm if S has KONVERT INFORM
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R delete : (%,UniversalSegment Integer) -> %
--R ?.? : (%,UniversalSegment Integer) -> %
--R entry? : (S,%) -> Boolean if $ has finiteAggregate and S has SETCAT
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R fill! : (%,S) -> % if $ has shallowlyMutable
--R find : ((S -> Boolean),%) -> Union(S,"failed")
--R first : % -> S if Integer has ORDSET
--R hash : % -> SingleInteger if S has SETCAT
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R maxIndex : % -> Integer if Integer has ORDSET
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R minIndex : % -> Integer if Integer has ORDSET
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List S if $ has finiteAggregate
```

```
--R qsetelt! : (%,Integer,S) -> S if $ has shallowlyMutable
--R reduce : (((S,S) -> S),%) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S,S) -> S if S has SETCAT and $ has finiteAggregate
--R remove : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (S,%) -> % if S has SETCAT and $ has finiteAggregate
--R removeDuplicates : % -> % if S has SETCAT and $ has finiteAggregate
--R select : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R setelt : (%,UniversalSegment Integer,S) -> S if $ has shallowlyMutable
--R setelt : (%,Integer,S) -> S if $ has shallowlyMutable
--R size? : (%,NonNegativeInteger) -> Boolean
--R swap! : (%,Integer,Integer) -> Void if $ has shallowlyMutable
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— **LinearAggregate.help** —

```
=====================================================================
LinearAggregate examples
=====================================================================

See Also:
o )show LinearAggregate
```

———-

See:

⇒ “ExtensibleLinearAggregate” (ELAGG) 7.0.101 on page 547
⇒ “FiniteLinearAggregate” (FLAGG) 7.0.102 on page 555
⇒ “StreamAggregate” (STAGG) 7.0.107 on page 585
⇐ “Collection” (CLAGG) 5.0.70 on page 287
⇐ “IndexedAggregate” (IXAGG) 5.0.74 on page 318

Exports:

any?	coerce	concat	construct	convert
copy	count	delete	elt	empty
empty?	entries	entry?	eq?	eval
every?	fill!	find	first	hash
index?	indices	insert	latex	less?
map	map!	maxIndex	member?	members
minIndex	more?	new	parts	qelt
qsetelt!	reduce	remove	removeDuplicates	sample
setelt	size?	swap!	?~=?	# ?
?=?	?.?			

Attributes exported:

- **nil**

Attributes Used:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.

These are directly exported but not implemented:

```
concat : (%,%) -> %
concat : List % -> %
delete : (%,Integer) -> %
delete : (%,UniversalSegment Integer) -> %
insert : (%,%,Integer) -> %
map : (((S,S) -> S),%,%) -> %
new : (NonNegativeInteger,S) -> %
setelt : (%,UniversalSegment Integer,S) -> S
         if $ has shallowlyMutable
?.? : (%,UniversalSegment Integer) -> %
```

These are implemented by this category:

```
concat : (%,S) -> %
concat : (S,%) -> %
index? : (Integer,%) -> Boolean
indices : % -> List Integer
insert : (S,%,Integer) -> %
maxIndex : % -> Integer if Integer has ORDSET
```

These exports come from (p318) IndexedAggregate(Integer,S:Type)

```
any? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
```

```
coerce : % -> OutputForm if S has SETCAT
copy : % -> %
count : (S,%) -> NonNegativeInteger
         if S has SETCAT and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
elt : (%,Integer,S) -> S
empty : () -> %
empty? : % -> Boolean
entries : % -> List S
entry? : (S,%) -> Boolean
         if $ has finiteAggregate and S has SETCAT
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,S,S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,Equation S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,List Equation S) -> %
         if S has EVALAB S and S has SETCAT
every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
fill! : (%,S) -> % if $ has shallowlyMutable
first : % -> S if Integer has ORDSET
hash : % -> SingleInteger if S has SETCAT
latex : % -> String if S has SETCAT
less? : (%,NonNegativeInteger) -> Boolean
map : ((S -> S),%) -> %
map! : ((S -> S),%) -> % if $ has shallowlyMutable
member? : (S,%) -> Boolean
         if S has SETCAT and $ has finiteAggregate
members : % -> List S if $ has finiteAggregate
minIndex : % -> Integer if Integer has ORDSET
more? : (%,NonNegativeInteger) -> Boolean
parts : % -> List S if $ has finiteAggregate
qelt : (%,Integer) -> S
qsetelt! : (%,Integer,S) -> S if $ has shallowlyMutable
sample : () -> %
setelt : (%,Integer,S) -> S if $ has shallowlyMutable
size? : (%,NonNegativeInteger) -> Boolean
swap! : (%,Integer,Integer) -> Void if $ has shallowlyMutable
#? : % -> NonNegativeInteger if $ has finiteAggregate
?.? : (%,Integer) -> S
?~=? : (%,%) -> Boolean if S has SETCAT
?=? : (%,%) -> Boolean if S has SETCAT
```

These exports come from (p287) Collection(S:Type):

```
construct : List S -> %
convert : % -> InputForm if S has KONVERT INFORM
```

```
 find : ((S -> Boolean),%) -> Union(S,"failed")
 reduce : (((S,S) -> S),%) -> S if $ has finiteAggregate
 reduce : (((S,S) -> S),%,S) -> S if $ has finiteAggregate
 reduce : (((S,S) -> S),%,S,S) -> S
                 if S has SETCAT and $ has finiteAggregate
 remove : ((S -> Boolean),%) -> % if $ has finiteAggregate
 remove : (S,%) -> % if S has SETCAT and $ has finiteAggregate
 removeDuplicates : % -> %
          if S has SETCAT and $ has finiteAggregate
 select : ((S -> Boolean),%) -> % if $ has finiteAggregate
```

— category LNAGG LinearAggregate —

```
)abbrev category LNAGG LinearAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A linear aggregate is an aggregate whose elements are indexed by integers.
++ Examples of linear aggregates are strings, lists, and
++ arrays.
++ Most of the exported operations for linear aggregates are non-destructive
++ but are not always efficient for a particular aggregate.
++ For example, \spadfun{concat} of two lists needs only to copy its first
++ argument, whereas \spadfun{concat} of two arrays needs to copy both
++ arguments. Most of the operations exported here apply to infinite
++ objects (e.g. streams) as well to finite ones.
++ For finite linear aggregates, see \spadtype{FiniteLinearAggregate}.

LinearAggregate(S:Type): Category ==
  Join(IndexedAggregate(Integer, S), Collection(S)) with
   new         : (NonNegativeInteger,S) -> %
     ++ new(n,x) returns \axiom{fill!(new n,x)}.
   concat: (%,S) -> %
     ++ concat(u,x) returns aggregate u with additional element x at the end.
     ++ Note that for lists, \axiom{concat(u,x) == concat(u,[x])}
   concat: (S,%) -> %
     ++ concat(x,u) returns aggregate u with additional element at the front.
     ++ Note that for lists: \axiom{concat(x,u) == concat([x],u)}.
   concat: (%,%) -> %
      ++ concat(u,v) returns an aggregate consisting of the elements of u
      ++ followed by the elements of v.
      ++ Note that if \axiom{w = concat(u,v)} then
```

```
      ++ \axiom{w.i = u.i for i in indices u}
      ++ and \axiom{w.(j + maxIndex u) = v.j for j in indices v}.
   concat: List % -> %
      ++ concat(u), where u is a lists of aggregates \axiom{[a,b,...,c]},
      ++ returns a single aggregate consisting of the elements of \axiom{a}
      ++ followed by those
      ++ of b followed ... by the elements of c.
      ++ Note that \axiom{concat(a,b,...,c) = concat(a,concat(b,...,c))}.
   map: ((S,S)->S,%,%) -> %
     ++ map(f,u,v) returns a new collection w with elements
     ++ \axiom{z = f(x,y)} for corresponding elements x and y from u and v.
     ++ Note that for linear aggregates, \axiom{w.i = f(u.i,v.i)}.
   elt: (%,UniversalSegment(Integer)) -> %
      ++ elt(u,i..j) (also written: \axiom{a(i..j)}) returns the aggregate of
      ++ elements \axiom{u} for k from i to j in that order.
      ++ Note that in general, \axiom{a.s = [a.k for i in s]}.
   delete: (%,Integer) -> %
      ++ delete(u,i) returns a copy of u with the \axiom{i}th
      ++ element deleted. Note that for lists,
      ++ \axiom{delete(a,i) == concat(a(0..i - 1),a(i + 1,..))}.
   delete: (%,UniversalSegment(Integer)) -> %
      ++ delete(u,i..j) returns a copy of u with the \axiom{i}th through
      ++ \axiom{j}th element deleted.
      ++ Note that \axiom{delete(a,i..j) = concat(a(0..i-1),a(j+1..))}.
   insert: (S,%,Integer) -> %
      ++ insert(x,u,i) returns a copy of u having x as its
      ++ \axiom{i}th element.
      ++ Note that \axiom{insert(x,a,k) = concat(concat(a(0..k-1),x),a(k..))}.
   insert: (%,%,Integer) -> %
      ++ insert(v,u,k) returns a copy of u having v inserted beginning at the
      ++ \axiom{i}th element.
      ++ Note that \axiom{insert(v,u,k) = concat( u(0..k-1), v, u(k..) )}.
   if % has shallowlyMutable then
    setelt: (%,UniversalSegment(Integer),S) -> S
      ++ setelt(u,i..j,x) (also written: \axiom{u(i..j) := x}) destructively
      ++ replaces each element in the segment \axiom{u(i..j)} by x.
      ++ The value x is returned.
      ++ Note that u is destructively change so
      ++ that \axiom{u.k := x for k in i..j};
      ++ its length remains unchanged.
 add
  indices a         == [i for i in minIndex a .. maxIndex a]
  index?(i, a)         == i >= minIndex a and i <= maxIndex a
  concat(a:%, x:S)        == concat(a, new(1, x))
  concat(x:S, y:%)        == concat(new(1, x), y)
  insert(x:S, a:%, i:Integer) == insert(new(1, x), a, i)
  if % has finiteAggregate then
    maxIndex l == #l - 1 + minIndex l

--if % has shallowlyMutable then new(n, s)  == fill_!(new n, s)
```

———-

— **LNAGG.dotabb** —

```
"LNAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=LNAGG"];
"LNAGG" -> "IXAGG"
"LNAGG" -> "CLAGG"
```

———-

— **LNAGG.dotfull** —

```
"LinearAggregate(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=LNAGG"];
"LinearAggregate(a:Type)" -> "IndexedAggregate(b:Integer,a:Type)"
"LinearAggregate(a:Type)" -> "Collection(a:Type)"
```

———-

— **LNAGG.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"LinearAggregate(a:Type)" [color=lightblue];
"LinearAggregate(a:Type)" -> "IndexedAggregate(b:Integer,a:Type)"
"LinearAggregate(a:Type)" -> "Collection(a:Type)"

"IndexedAggregate(b:Integer,a:Type)" [color=seagreen];
"IndexedAggregate(b:Integer,a:Type)" ->
    "IndexedAggregate(a:SetCategory,b:Type)"

"IndexedAggregate(a:SetCategory,b:Type)" [color=lightblue];
"IndexedAggregate(a:SetCategory,b:Type)" ->
    "HomogeneousAggregate(a:Type)"

"Collection(a:Type)" [color=lightblue];
"Collection(a:Type)" -> "HomogeneousAggregate(a:Type)"

"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "Aggregate()"
```

```
"Aggregate()" [color=lightblue];
"Aggregate()" -> "Type()"

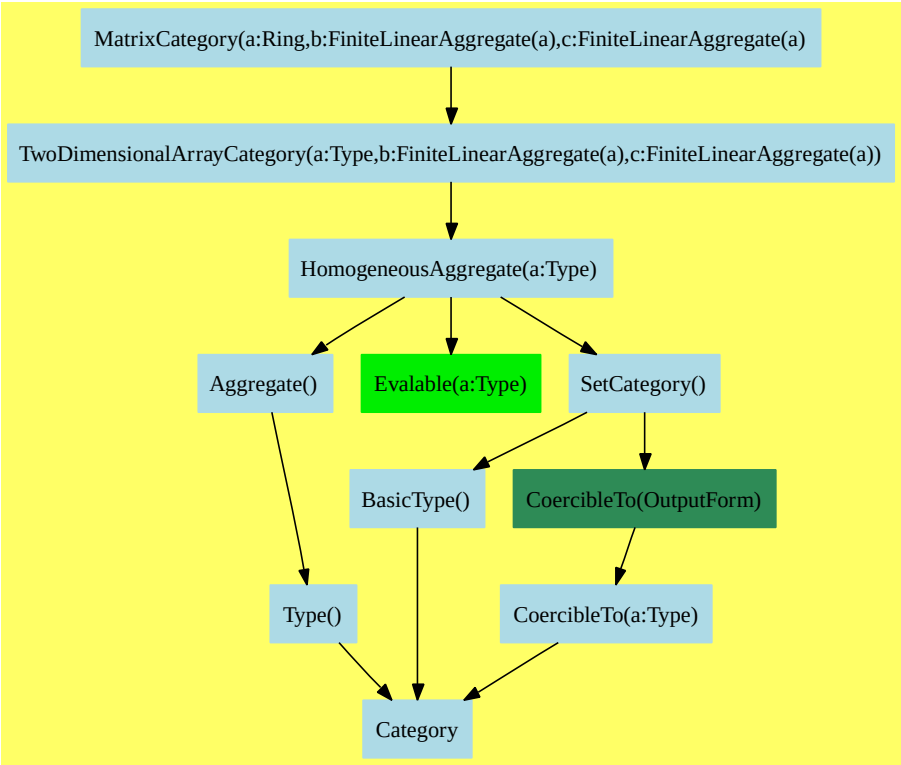
"Type()" [color=lightblue];
"Type()" -> "Category"

"Category" [color=lightblue];

}
```

———-

6.0.88 MatrixCategory (MATCAT)



— MatrixCategory.input —

```
)set break resume
)sys rm -f MatrixCategory.output
```

```
)spool MatrixCategory.output
)set message test on
)set message auto off
)clear all


--S 1 of 59
square? matrix [[j**i for i in 0..4] for j in 1..5]
--R
--R
--R   (1)  true
--R                                                              Type: Boolean
--E 1

--S 2 of 59
diagonal? matrix [[j**i for i in 0..4] for j in 1..5]
--R
--R
--R   (2)  false
--R                                                              Type: Boolean
--E 2

--S 3 of 59
symmetric? matrix [[j**i for i in 0..4] for j in 1..5]
--R
--R
--R   (3)  false
--R                                                              Type: Boolean
--E 3

--S 4 of 59
antisymmetric? matrix [[j**i for i in 0..4] for j in 1..5]
--R
--R
--R   (4)  false
--R                                                              Type: Boolean
--E 4

--S 5 of 59
z:Matrix(INT):=zero(3,3)
--R
--R
--R        +0  0  0+
--R        |       |
--R   (5)  |0  0  0|
--R        |       |
--R        +0  0  0+
--R                                                  Type: Matrix Integer
--E 5
```

```
--S 6 of 59
matrix [[1,2,3],[4,5,6],[7,8,9],[1,1,1]]
--R
--R
--R        +1  2  3+
--R        |       |
--R        |4  5  6|
--R   (6)  |       |
--R        |7  8  9|
--R        |       |
--R        +1  1  1+
--R                                                    Type: Matrix Integer
--E 6

--S 7 of 59
z:Matrix(INT):=scalarMatrix(3,5)
--R
--R
--R        +5  0  0+
--R        |       |
--R   (7)  |0  5  0|
--R        |       |
--R        +0  0  5+
--R                                                    Type: Matrix Integer
--E 7

--S 8 of 59
diagonalMatrix [1,2,3]
--R
--R
--R        +1  0  0+
--R        |       |
--R   (8)  |0  2  0|
--R        |       |
--R        +0  0  3+
--R                                                    Type: Matrix Integer
--E 8

--S 9 of 59
diagonalMatrix [matrix [[1,2],[3,4]], matrix [[4,5],[6,7]]]
--R
--R
--R        +1  2  0  0+
--R        |          |
--R        |3  4  0  0|
--R   (9)  |          |
--R        |0  0  4  5|
--R        |          |
--R        +0  0  6  7+
--R                                                    Type: Matrix Integer
```

```
--E 9

--S 10 of 59
coerce([1,2,3])@Matrix(INT)
--R
--R
--R         +1+
--R         | |
--R   (10)  |2|
--R         | |
--R         +3+
--R                                                    Type: Matrix Integer
--E 10

--S 11 of 59
transpose([1,2,3])@Matrix(INT)
--R
--R
--R   (11)  [1  2  3]
--R                                                    Type: Matrix Integer
--E 11

--S 12 of 59
transpose matrix [[j**i for i in 0..4] for j in 1..5]
--R
--R
--R         +1  1   1    1    1 +
--R         |                   |
--R         |1  2   3    4    5 |
--R         |                   |
--R   (12)  |1  4   9   16   25 |
--R         |                   |
--R         |1  8   27  64   125|
--R         |                   |
--R         +1  16  81  256  625+
--R                                                    Type: Matrix Integer
--E 12

--S 13 of 59
squareTop matrix [[j**i for i in 0..2] for j in 1..5]
--R
--R
--R         +1  1  1+
--R         |      |
--R   (13)  |1  2  4|
--R         |      |
--R         +1  3  9+
--R                                                    Type: Matrix Integer
--E 13
```

```
--S 14 of 59
t1:=matrix [[j**i for i in 0..4] for j in 1..5]
--R
--R
--R         +1  1  1    1    1 +
--R         |                  |
--R         |1  2  4    8   16 |
--R         |                  |
--R   (14)  |1  3  9   27   81 |
--R         |                  |
--R         |1  4  16  64   256|
--R         |                  |
--R         +1  5  25  125  625+
--R                                                  Type: Matrix Integer
--E 14

--S 15 of 59
horizConcat(t1,t1)
--R
--R
--R         +1  1  1    1    1   1  1  1    1    1 +
--R         |                                     |
--R         |1  2  4    8   16   1  2  4    8   16 |
--R         |                                     |
--R   (15)  |1  3  9   27   81   1  3  9   27   81 |
--R         |                                     |
--R         |1  4  16  64   256  1  4  16  64   256|
--R         |                                     |
--R         +1  5  25  125  625  1  5  25  125  625+
--R                                                  Type: Matrix Integer
--E 15

--S 16 of 59
t2:=matrix [[j**i for i in 0..4] for j in 1..5]
--R
--R
--R         +1  1  1    1    1 +
--R         |                  |
--R         |1  2  4    8   16 |
--R         |                  |
--R   (16)  |1  3  9   27   81 |
--R         |                  |
--R         |1  4  16  64   256|
--R         |                  |
--R         +1  5  25  125  625+
--R                                                  Type: Matrix Integer
--E 16

--S 17 of 59
vertConcat(t2,t2)
```

```
--R
--R
--R         +1  1  1    1    1 +
--R         |                  |
--R         |1  2  4    8   16 |
--R         |                  |
--R         |1  3  9   27   81 |
--R         |                  |
--R         |1  4  16  64   256|
--R         |                  |
--R         |1  5  25  125  625|
--R   (17)  |                  |
--R         |1  1  1    1    1 |
--R         |                  |
--R         |1  2  4    8   16 |
--R         |                  |
--R         |1  3  9   27   81 |
--R         |                  |
--R         |1  4  16  64   256|
--R         |                  |
--R         +1  5  25  125  625+
--R                                                    Type: Matrix Integer
--E 17

--S 18 of 59
t3:=matrix [[j**i for i in 0..4] for j in 1..5]
--R
--R
--R         +1  1  1    1    1 +
--R         |                  |
--R         |1  2  4    8   16 |
--R         |                  |
--R   (18)  |1  3  9   27   81 |
--R         |                  |
--R         |1  4  16  64   256|
--R         |                  |
--R         +1  5  25  125  625+
--R                                                    Type: Matrix Integer
--E 18

--S 19 of 59
listOfLists t3
--R
--R
--R   (19)
--R   [[1,1,1,1,1],[1,2,4,8,16],[1,3,9,27,81],[1,4,16,64,256],[1,5,25,125,625]]
--R                                                 Type: List List Integer
--E 19

--S 20 of 59
```

```
t4:=matrix [[j**i for i in 0..4] for j in 1..5]
--R
--R
--R         +1  1  1    1    1 +
--R         |                  |
--R         |1  2  4    8   16 |
--R         |                  |
--R   (20)  |1  3  9   27   81 |
--R         |                  |
--R         |1  4  16  64   256|
--R         |                  |
--R         +1  5  25  125  625+
--R                                                    Type: Matrix Integer
--E 20

--S 21 of 59
elt(t4,3,3)
--R
--R
--R   (21)  9
--R                                                   Type: PositiveInteger
--E 21

--S 22 of 59
t5:=matrix [[j**i for i in 0..4] for j in 1..5]
--R
--R
--R         +1  1  1    1    1 +
--R         |                  |
--R         |1  2  4    8   16 |
--R         |                  |
--R   (22)  |1  3  9   27   81 |
--R         |                  |
--R         |1  4  16  64   256|
--R         |                  |
--R         +1  5  25  125  625+
--R                                                    Type: Matrix Integer
--E 22

--S 23 of 59
setelt(t5,3,3,10)
--R
--R
--R   (23)  10
--R                                                   Type: PositiveInteger
--E 23

--S 24 of 59
t6:=matrix [[j**i for i in 0..4] for j in 1..5]
--R
```

```
--R
--R         +1  1  1    1    1 +
--R         |                  |
--R         |1  2  4    8   16 |
--R         |                  |
--R   (24)  |1  3  9   27   81 |
--R         |                  |
--R         |1  4  16  64   256|
--R         |                  |
--R         +1  5  25  125  625+
--R                                                  Type: Matrix Integer
--E 24

--S 25 of 59
swapRows!(t6,2,4)
--R
--R
--R         +1  1  1    1    1 +
--R         |                  |
--R         |1  4  16  64   256|
--R         |                  |
--R   (25)  |1  3  9   27   81 |
--R         |                  |
--R         |1  2  4    8   16 |
--R         |                  |
--R         +1  5  25  125  625+
--R                                                  Type: Matrix Integer
--E 25

--S 26 of 59
t7:=matrix [[j**i for i in 0..4] for j in 1..5]
--R
--R
--R         +1  1  1    1    1 +
--R         |                  |
--R         |1  2  4    8   16 |
--R         |                  |
--R   (26)  |1  3  9   27   81 |
--R         |                  |
--R         |1  4  16  64   256|
--R         |                  |
--R         +1  5  25  125  625+
--R                                                  Type: Matrix Integer
--E 26

--S 27 of 59
swapColumns!(t7,2,4)
--R
--R
--R         +1   1   1   1   1 +
```

```
--R         |                  |
--R         |1   8   4   2  16 |
--R         |                  |
--R   (27)  |1  27   9   3  81 |
--R         |                  |
--R         |1  64   16  4  256|
--R         |                  |
--R         +1  125  25  5  625+
--R                                                  Type: Matrix Integer
--E 27

--S 28 of 59
t8:=matrix [[j**i for i in 0..4] for j in 1..5]
--R
--R
--R         +1  1  1    1    1 +
--R         |                  |
--R         |1  2  4    8   16 |
--R         |                  |
--R   (28)  |1  3  9   27   81 |
--R         |                  |
--R         |1  4  16  64   256|
--R         |                  |
--R         +1  5  25  125  625+
--R                                                  Type: Matrix Integer
--E 28

--S 29 of 59
subMatrix(t8,1,3,2,4)
--R
--R
--R         +1  1  1 +
--R         |        |
--R   (29)  |2  4  8 |
--R         |        |
--R         +3  9  27+
--R                                                  Type: Matrix Integer
--E 29

--S 30 of 59
t9:=matrix [[j**i for i in 0..4] for j in 1..5]
--R
--R
--R         +1  1  1    1    1 +
--R         |                  |
--R         |1  2  4    8   16 |
--R         |                  |
--R   (30)  |1  3  9   27   81 |
--R         |                  |
--R         |1  4  16  64   256|
```

```
--R         |                  |
--R         +1  5  25  125  625+
--R                                                    Type: Matrix Integer
--E 30

--S 31 of 59
setsubMatrix!(t9,2,2,matrix [[3,3],[3,3]])
--R
--R
--R         +1  1  1    1    1 +
--R         |                  |
--R         |1  3  3    8   16 |
--R         |                  |
--R   (31)  |1  3  3   27   81 |
--R         |                  |
--R         |1  4  16  64   256|
--R         |                  |
--R         +1  5  25  125  625+
--R                                                    Type: Matrix Integer
--E 31

--S 32 of 59
t0:=matrix [[j**i for i in 0..4] for j in 1..5]
--R
--R
--R         +1  1  1    1    1 +
--R         |                  |
--R         |1  2  4    8   16 |
--R         |                  |
--R   (32)  |1  3  9   27   81 |
--R         |                  |
--R         |1  4  16  64   256|
--R         |                  |
--R         +1  5  25  125  625+
--R                                                    Type: Matrix Integer
--E 32

--S 33 of 59
t0+t0
--R
--R
--R         +2  2   2    2    2  +
--R         |                    |
--R         |2  4   8   16    32 |
--R         |                    |
--R   (33)  |2  6   18  54   162 |
--R         |                    |
--R         |2  8   32  128  512 |
--R         |                    |
--R         +2  10  50  250  1250+
```

```
--R                                                          Type: Matrix Integer
--E 33

--S 34 of 59
t0-t0
--R
--R
--R         +0  0  0  0  0+
--R         |              |
--R         |0  0  0  0  0|
--R         |              |
--R   (34)  |0  0  0  0  0|
--R         |              |
--R         |0  0  0  0  0|
--R         |              |
--R         +0  0  0  0  0+
--R                                                          Type: Matrix Integer
--E 34

--S 35 of 59
-t0
--R
--R
--R         +- 1  - 1  - 1    - 1    - 1 +
--R         |                            |
--R         |- 1  - 2  - 4    - 8   - 16 |
--R         |                            |
--R   (35)  |- 1  - 3  - 9   - 27   - 81 |
--R         |                            |
--R         |- 1  - 4  - 16  - 64   - 256|
--R         |                            |
--R         +- 1  - 5  - 25  - 125  - 625+
--R                                                          Type: Matrix Integer
--E 35

--S 36 of 59
t0*t0
--R
--R
--R         + 5    15    55     225    979  +
--R         |                               |
--R         |31   129    573   2637   12405 |
--R         |                               |
--R   (36)  |121  547   2551   12121  58315 |
--R         |                               |
--R         |341  1593  7585   36561  177745|
--R         |                               |
--R         +781  3711  17871  86841  424731+
--R                                                          Type: Matrix Integer
--E 36
```

```
--S 37 of 59
1/3*t0
--R
--R
--R         +1  1  1    1    1 +
--R         |-  -  -    -    - |
--R         |3  3  3    3    3 |
--R         |                   |
--R         |1  2  4    8   16 |
--R         |-  -  -    -   -- |
--R         |3  3  3    3    3 |
--R         |                   |
--R         |1                  |
--R   (37)  |-  1  3    9   27 |
--R         |3                  |
--R         |                   |
--R         |1  4  16  64   256|
--R         |-  -  --  --   ---|
--R         |3  3   3   3    3 |
--R         |                   |
--R         |1  5  25  125  625|
--R         |-  -  --  ---  ---|
--R         +3  3   3   3    3 +
--R                                                Type: Matrix Fraction Integer
--E 37

--S 38 of 59
m:=matrix [[j**i for i in 0..4] for j in 1..5]
--R
--R
--R         +1  1  1    1    1 +
--R         |                   |
--R         |1  2  4    8   16 |
--R         |                   |
--R   (38)  |1  3  9   27   81 |
--R         |                   |
--R         |1  4  16  64   256|
--R         |                   |
--R         +1  5  25  125  625+
--R                                                   Type: Matrix Integer
--E 38

--S 39 of 59
t0*1/3
--R
--R
--R         +1  1  1    1    1 +
--R         |-  -  -    -    - |
--R         |3  3  3    3    3 |
```

```
--R         |                    |
--R         |1  2  4    8   16  |
--R         |-  -  -    -   --  |
--R         |3  3  3    3    3  |
--R         |                    |
--R         |1                   |
--R   (39)  |-  1  3    9   27  |
--R         |3                   |
--R         |                    |
--R         |1  4  16  64   256 |
--R         |-  -  --  --   --- |
--R         |3  3   3   3    3  |
--R         |                    |
--R         |1  5  25  125  625 |
--R         |-  -  --  ---  --- |
--R         +3  3   3   3    3  +
--R                                        Type: Matrix Fraction Integer
--E 39

--S 40 of 59
3*t0
--R
--R
--R         +3  3   3    3    3  +
--R         |                    |
--R         |3  6   12  24   48  |
--R         |                    |
--R   (40)  |3  9   27  81   243 |
--R         |                    |
--R         |3  12  48  192  768 |
--R         |                    |
--R         +3  15  75  375  1875+
--R                                          Type: Matrix Integer
--E 40

--S 41 of 59
c:=coerce([1,2,3,4,5])@Matrix(INT)
--R
--R
--R         +1+
--R         | |
--R         |2|
--R         | |
--R   (41)  |3|
--R         | |
--R         |4|
--R         | |
--R         +5+
--R                                          Type: Matrix Integer
--E 41
```

```
--S 42 of 59
t0*c
--R
--R
--R         + 15 +
--R         |    |
--R         |129 |
--R         |    |
--R   (42)  |547 |
--R         |    |
--R         |1593|
--R         |    |
--R         +3711+
--R                                                    Type: Matrix Integer
--E 42

--S 43 of 59
r:=transpose([1,2,3,4,5])@Matrix(INT)
--R
--R
--R   (43)  [1  2  3  4  5]
--R                                                    Type: Matrix Integer
--E 43

--S 44 of 59
r*t0
--R
--R
--R   (44)  [15  55  225  979  4425]
--R                                                    Type: Matrix Integer
--E 44

--S 45 of 59
t0**3
--R
--R
--R         + 1279    5995     28635     138385    674175  +
--R         |                                              |
--R         |15775    74581    358021   1735927    8476705 |
--R         |                                              |
--R   (45)  |73655   348927   1677079   8138493   39765355 |
--R         |                                              |
--R         |223825  1061251  5103579   24775909  121090455|
--R         |                                              |
--R         +533935  2532835  12184195  59162185  289195879+
--R                                                    Type: Matrix Integer
--E 45

--S 46 of 59
```

```
t10:=matrix [[2**i for i in 2..4] for j in 1..5]
--R
--R
--R         +4  8  16+
--R         |        |
--R         |4  8  16|
--R         |        |
--R   (46)  |4  8  16|
--R         |        |
--R         |4  8  16|
--R         |        |
--R         +4  8  16+
--R                                                  Type: Matrix Integer
--E 46

--S 47 of 59
exquo(t10,2)
--R
--R
--R         +2  4  8+
--R         |       |
--R         |2  4  8|
--R         |       |
--R   (47)  |2  4  8|
--R         |       |
--R         |2  4  8|
--R         |       |
--R         +2  4  8+
--R                                        Type: Union(Matrix Integer,...)
--E 47

--S 48 of 59
t10/4
--R
--R
--R         +1  2  4+
--R         |       |
--R         |1  2  4|
--R         |       |
--R   (48)  |1  2  4|
--R         |       |
--R         |1  2  4|
--R         |       |
--R         +1  2  4+
--R                                          Type: Matrix Fraction Integer
--E 48

--S 49 of 59
rowEchelon matrix [[j**i for i in 0..4] for j in 1..5]
--R
```

```
--R
--R         +1  0  0  0  0 +
--R         |               |
--R         |0  1  1  1  1 |
--R         |               |
--R   (49)  |0  0  2  0  2 |
--R         |               |
--R         |0  0  0  6  12|
--R         |               |
--R         +0  0  0  0  24+
--R                                                    Type: Matrix Integer
--E 49

--S 50 of 59
columnSpace matrix [[1,2,3],[4,5,6],[7,8,9],[1,1,1]]
--R
--R
--R   (50)  [[1,4,7,1],[2,5,8,1]]
--R                                                 Type: List Vector Integer
--E 50

--S 51 of 59
rank matrix [[1,2,3],[4,5,6],[7,8,9]]
--R
--R
--R   (51)  2
--R                                                     Type: PositiveInteger
--E 51

--S 52 of 59
nullity matrix [[1,2,3],[4,5,6],[7,8,9]]
--R
--R
--R   (52)  1
--R                                                     Type: PositiveInteger
--E 52

--S 53 of 59
nullSpace matrix [[1,2,3],[4,5,6],[7,8,9]]
--R
--R
--R   (53)  [[1,- 2,1]]
--R                                                 Type: List Vector Integer
--E 53

--S 54 of 59
determinant matrix [[j**i for i in 0..4] for j in 1..5]
--R
--R
--R   (54)  288
```

```
--R                                                          Type: PositiveInteger
--E 54

--S 55 of 59
minordet matrix [[j**i for i in 0..4] for j in 1..5]
--R
--R
--R   (55)  288
--R                                                          Type: PositiveInteger
--E 55

--S 56 of 59
pfaffian [[0,1,0,0],[-1,0,0,0],[0,0,0,1],[0,0,-1,0]]
--R
--R
--R   (56)  1
--R                                                          Type: PositiveInteger
--E 56

--S 57 of 59
inverse matrix [[j**i for i in 0..4] for j in 1..5]
--R
--R
--R         + 5    - 10   10   - 5    1  +
--R         |                           |
--R         |  77  107     39   61     25|
--R         |- --  ---   - --   --   - --|
--R         |  12   6       2    6     12|
--R         |                           |
--R         | 71     59   49     41   35 |
--R         | --   - --   --   - --   -- |
--R   (57)  | 24      6    4      6   24 |
--R         |                           |
--R         |   7   13          11      5|
--R         |- --   --   - 3    --   - --|
--R         |  12    6           6     12|
--R         |                           |
--R         |  1     1    1      1     1 |
--R         | --   - -    -    - -    -- |
--R         + 24     6    4      6    24 +
--R                                    Type: Union(Matrix Fraction Integer,...)
--E 57

--S 58 of 59
(matrix [[j**i for i in 0..4] for j in 1..5]) ** 2
--R
--R
--R         + 5    15    55     225    979  +
--R         |                              |
--R         |31   129    573   2637   12405 |
```

```
--R         |                                  |
--R   (58)  |121  547   2551   12121  58315 |
--R         |                                  |
--R         |341  1593  7585   36561  177745|
--R         |                                  |
--R         +781  3711  17871  86841  424731+
--R                                                    Type: Matrix Integer
--E 58

--S 59 of 59
)show MatrixCategory
--R
--R MatrixCategory(R: Ring,Row: FiniteLinearAggregate t#1,Col: FiniteLinearAggregate t#1)  is a category co
--R Abbreviation for MatrixCategory is MATCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for MATCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (Row,%) -> Row                 ?*? : (%,Col) -> Col
--R ?*? : (Integer,%) -> %               ?*? : (%,R) -> %
--R ?*? : (R,%) -> %                     ?*? : (%,%) -> %
--R ?+? : (%,%) -> %                     -? : % -> %
--R ?-? : (%,%) -> %                     antisymmetric? : % -> Boolean
--R coerce : Col -> %                    column : (%,Integer) -> Col
--R copy : % -> %                        diagonal? : % -> Boolean
--R diagonalMatrix : List % -> %         diagonalMatrix : List R -> %
--R elt : (%,Integer,Integer,R) -> R     elt : (%,Integer,Integer) -> R
--R empty : () -> %                      empty? : % -> Boolean
--R eq? : (%,%) -> Boolean               fill! : (%,R) -> %
--R horizConcat : (%,%) -> %             listOfLists : % -> List List R
--R map : (((R,R) -> R),%,%,R) -> %      map : (((R,R) -> R),%,%) -> %
--R map : ((R -> R),%) -> %              map! : ((R -> R),%) -> %
--R matrix : List List R -> %            maxColIndex : % -> Integer
--R maxRowIndex : % -> Integer           minColIndex : % -> Integer
--R minRowIndex : % -> Integer           ncols : % -> NonNegativeInteger
--R nrows : % -> NonNegativeInteger      parts : % -> List R
--R qelt : (%,Integer,Integer) -> R      row : (%,Integer) -> Row
--R sample : () -> %                     setRow! : (%,Integer,Row) -> %
--R square? : % -> Boolean               squareTop : % -> %
--R symmetric? : % -> Boolean            transpose : % -> %
--R transpose : Row -> %                 vertConcat : (%,%) -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?**? : (%,Integer) -> % if R has FIELD
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (%,R) -> % if R has FIELD
--R ?=? : (%,%) -> Boolean if R has SETCAT
--R any? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
--R coerce : % -> OutputForm if R has SETCAT
--R columnSpace : % -> List Col if R has EUCDOM
--R count : (R,%) -> NonNegativeInteger if R has SETCAT and $ has finiteAggregate
```

```
--R count : ((R -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R determinant : % -> R if R has commutative *
--R elt : (%,List Integer,List Integer) -> %
--R eval : (%,List R,List R) -> % if R has EVALAB R and R has SETCAT
--R eval : (%,R,R) -> % if R has EVALAB R and R has SETCAT
--R eval : (%,Equation R) -> % if R has EVALAB R and R has SETCAT
--R eval : (%,List Equation R) -> % if R has EVALAB R and R has SETCAT
--R every? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
--R exquo : (%,R) -> Union(%,"failed") if R has INTDOM
--R hash : % -> SingleInteger if R has SETCAT
--R inverse : % -> Union(%,"failed") if R has FIELD
--R latex : % -> String if R has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R member? : (R,%) -> Boolean if R has SETCAT and $ has finiteAggregate
--R members : % -> List R if $ has finiteAggregate
--R minordet : % -> R if R has commutative *
--R more? : (%,NonNegativeInteger) -> Boolean
--R new : (NonNegativeInteger,NonNegativeInteger,R) -> %
--R nullSpace : % -> List Col if R has INTDOM
--R nullity : % -> NonNegativeInteger if R has INTDOM
--R pfaffian : % -> R if R has COMRING
--R qsetelt! : (%,Integer,Integer,R) -> R
--R rank : % -> NonNegativeInteger if R has INTDOM
--R rowEchelon : % -> % if R has EUCDOM
--R scalarMatrix : (NonNegativeInteger,R) -> %
--R setColumn! : (%,Integer,Col) -> %
--R setelt : (%,List Integer,List Integer,%) -> %
--R setelt : (%,Integer,Integer,R) -> R
--R setsubMatrix! : (%,Integer,Integer,%) -> %
--R size? : (%,NonNegativeInteger) -> Boolean
--R subMatrix : (%,Integer,Integer,Integer,Integer) -> %
--R swapColumns! : (%,Integer,Integer) -> %
--R swapRows! : (%,Integer,Integer) -> %
--R zero : (NonNegativeInteger,NonNegativeInteger) -> %
--R ?~=? : (%,%) -> Boolean if R has SETCAT
--R
--E 59

)spool
)lisp (bye)
```

———-

— MatrixCategory.help —

```
=====================================================================
MatrixCategory examples
=====================================================================
```

```
Predicates:

square?(m) returns true if m is a square matrix
(if m has the same number of rows as columns) and false otherwise.

  square matrix [[j**i for i in 0..4] for j in 1..5]

diagonal?(m) returns true if the matrix m is square and
diagonal (i.e. all entries of m not on the diagonal are zero) and
false otherwise.

  diagonal? matrix [[j**i for i in 0..4] for j in 1..5]

symmetric?(m) returns true if the matrix m is square and
symmetric (i.e. \spad{m[i,j] = m[j,i]} for all i and j) and false
otherwise.

  symmetric? matrix [[j**i for i in 0..4] for j in 1..5]

antisymmetric?(m) returns true if the matrix m is square and
antisymmetric (i.e. m[i,j] = -m[j,i] for all i and j)
and false otherwise.

  antisymmetric? matrix [[j**i for i in 0..4] for j in 1..5]


Creation

zero(m,n) returns an m-by-n zero matrix.

  z:Matrix(INT):=zero(3,3)

matrix(l) converts the list of lists l to a matrix, where the
list of lists is viewed as a list of the rows of the matrix.

  matrix [[1,2,3],[4,5,6],[7,8,9],[1,1,1]]

scalarMatrix(n,r) returns an n-by-n matrix with r's on the
diagonal and zeroes elsewhere.

  z:Matrix(INT):=scalarMatrix(3,5)

diagonalMatrix(l) returns a diagonal matrix with the elements
of l on the diagonal.

  diagonalMatrix [1,2,3]

diagonalMatrix([m1,...,mk]) creates a block diagonal matrix
M with block matrices m1,...,mk down the diagonal,
```

```
with 0 block matrices elsewhere.

More precisly: if ri := nrows mi, ci := ncols mi,
then m is an (r1+..+rk) by (c1+..+ck) - matrix  with entries
m.i.j = ml.(i-r1-..-r(l-1)).(j-n1-..-n(l-1)),
if (r1+..+r(l-1)) < i <= r1+..+rl and
   (c1+..+c(l-1)) < i <= c1+..+cl,
    m.i.j = 0  otherwise.

  diagonalMatrix [matrix [[1,2],[3,4]], matrix [[4,5],[6,7]]]

coerce(col) converts the column col to a column matrix.

  coerce([1,2,3])@Matrix(INT)

transpose(r) converts the row r to a row matrix.

  transpose([1,2,3])@Matrix(INT)

Creation of new matrices from old

transpose(m) returns the transpose of the matrix m.

  m:=matrix [[j**i for i in 0..4] for j in 1..5]
  transpose m

squareTop(m) returns an n-by-n matrix consisting of the first
n rows of the m-by-n matrix m. Error: if m < n.

  m:=matrix [[j**i for i in 0..2] for j in 1..5]
  squareTop m

horizConcat(x,y) horizontally concatenates two matrices with
an equal number of rows. The entries of y appear to the right
of the entries of x.  Error: if the matrices
do not have the same number of rows.

  m:=matrix [[j**i for i in 0..4] for j in 1..5]
  horizConcat(m,m)

vertConcat(x,y) vertically concatenates two matrices with an
equal number of columns. The entries of y appear below
of the entries of x.  Error: if the matrices
do not have the same number of columns.

  m:=matrix [[j**i for i in 0..4] for j in 1..5]
  vertConcat(m,m)

Part extractions/assignments
```

```
listOfLists(m) returns the rows of the matrix m as a list of lists

  m:=matrix [[j**i for i in 0..4] for j in 1..5]
  listOfLists m

elt(x,rowList,colList) returns an m-by-n matrix consisting
of elements of x, where m = # rowList and n = # colList
If rowList = [i<1>,i<2>,...,i<m>] and
   colList = [j<1>,j<2>,...,j<n>],
 then the (k,l)-th entry of elt(x,rowList,colList) is x(i<k>,j<l>).

  m:=matrix [[j**i for i in 0..4] for j in 1..5]
  elt(m,3,3)

setelt(x,rowList,colList,y) destructively alters the matrix x.
If y is m-by-n,
   rowList = [i<1>,i<2>,...,i<m>] and
   colList = [j<1>,j<2>,...,j<n>],
 then x(i<k>,j<l>)
 is set to y(k,l) for k = 1,...,m and l = 1,...,n

  m:=matrix [[j**i for i in 0..4] for j in 1..5]
  setelt(m,3,3,10)

swapRows!(m,i,j) interchanges the i-th and j-th
rows of m. This destructively alters the matrix.

  m:=matrix [[j**i for i in 0..4] for j in 1..5]
  swapRows!(m,2,4)

swapColumns!(m,i,j) interchanges the i-th and j-th
columns of m. This destructively alters the matrix.

  m:=matrix [[j**i for i in 0..4] for j in 1..5]
  swapColumns!(m,2,4)

subMatrix(x,i1,i2,j1,j2) extracts the submatrix [x(i,j)]
where the index i ranges from i1 to i2
and   the index j ranges from j1 to j2.

  m:=matrix [[j**i for i in 0..4] for j in 1..5]
  subMatrix(m,1,3,2,4)

setsubMatrix(x,i1,j1,y) destructively alters the matrix x.
Here x(i,j) is set to y(i-i1+1,j-j1+1) for
i = i1,...,i1-1+nrows y and j = j1,...,j1-1+ncols y.

  m:=matrix [[j**i for i in 0..4] for j in 1..5]
  setsubMatrix!(m,2,2,matrix [[3,3],[3,3]])
```

```
Arithmetic

x + y is the sum of the matrices x and y.
It is an error if the dimensions are incompatible.

  m:=matrix [[j**i for i in 0..4] for j in 1..5]
  m+m

x - y is the difference of the matrices x and y.
It is an error if the dimensions are incompatible.

  m:=matrix [[j**i for i in 0..4] for j in 1..5]
  m-m

-x returns the negative of the matrix x.

  m:=matrix [[j**i for i in 0..4] for j in 1..5]
  -m

x * y is the product of the matrices x and y.
It is an error if the dimensions are incompatible.

  m:=matrix [[j**i for i in 0..4] for j in 1..5]
  m*m

r*x is the left scalar multiple of the scalar r and the matrix x.

  m:=matrix [[j**i for i in 0..4] for j in 1..5]
  1/3*m

x * r is the right scalar multiple of the scalar r and the matrix x.

  m:=matrix [[j**i for i in 0..4] for j in 1..5]
  m*1/3

n * x is an integer multiple.

  m:=matrix [[j**i for i in 0..4] for j in 1..5]
  3*m

x * c is the product of the matrix x and the column vector c.
It is an error if the dimensions are incompatible.

  m:=matrix [[j**i for i in 0..4] for j in 1..5]
  c:=coerce([1,2,3,4,5])@Matrix(INT)
  m *c

r * x is the product of the row vector r and the matrix x.
It is an error if the dimensions are incompatible.
```

```
  m:=matrix [[j**i for i in 0..4] for j in 1..5]
  r:=transpose([1,2,3,4,5])@Matrix(INT)
  r*m

x ** n computes a non-negative integral power of the matrix x.
It is an error if the matrix is not square.

  m:=matrix [[j**i for i in 0..4] for j in 1..5]
  m**3

exquo(m,r) computes the exact quotient of the elements
of m by r, returning "failed" if this is not possible.

  m:=matrix [[2**i for i in 2..4] for j in 1..5]
  exquo(m,2)

m/r divides the elements of m by r, r must be non-zero.

  m:=matrix [[2**i for i in 2..4] for j in 1..5]
  m/4

Linear algebra

rowEchelon(m) returns the row echelon form of the matrix m.

  rowEchelon matrix [[j**i for i in 0..4] for j in 1..5]

columnSpace(m) returns a sublist of columns of the matrix m

  columnSpace matrix [[1,2,3],[4,5,6],[7,8,9],[1,1,1]]

rank(m) returns the rank of the matrix m.

  rank matrix [[1,2,3],[4,5,6],[7,8,9]]

nullity(m) returns the nullity of the matrix m. This is
the dimension of the null space of the matrix m.

  nullity matrix [[1,2,3],[4,5,6],[7,8,9]]

nullSpace(m) returns a basis for the null space of the matrix m.

  nullSpace matrix [[1,2,3],[4,5,6],[7,8,9]]

determinant(m) returns the determinant of the matrix m.
It is an error if the matrix is not square.

  determinant matrix [[j**i for i in 0..4] for j in 1..5]
```

```
minordet(m) computes the determinant of the matrix m using minors.
It is an error if the matrix is not square.

  minordet matrix [[j**i for i in 0..4] for j in 1..5]

pfaffian(m) returns the Pfaffian of the matrix m.
It is an error if the matrix is not antisymmetric

  pfaffian [[0,1,0,0],[-1,0,0,0],[0,0,0,1],[0,0,-1,0]]

inverse(m) returns the inverse of the matrix m.
If the matrix is not invertible, "failed" is returned.
It is an error if the matrix is not square.

  inverse matrix [[j**i for i in 0..4] for j in 1..5]

m**n computes an integral power of the matrix m.
It is an error if matrix is not square or
if the matrix is square but not invertible.

  (matrix [[j**i for i in 0..4] for j in 1..5]) ** 2
```

———-

We define three categories for matrices

- MatrixCategory is the category of all matrices
- RectangularMatrixCategory is the category of all matrices of a given dimension
- SquareMatrixCategory inherits from RectangularMatrixCategory

The Matrix domain is the domain of all matrices.

All three domains share the same representation, inherited from Matrix. Most algorithms are only implemented for Matrix but implemented in separate packages.

- MatrixLinearAlgebraFunctions is the top-level package that calls the other packages
- InnerMatrixLinearAlgebraFunctions contains implementations that work over a Field
- InnerMatrixQuotientFieldFunctions contain implementations that work over a quotient field

Implementations that rely on the representation of matrices used in Matrix should be put into these packages.

See:

⇐ “TwoDimensionalArrayCategory” (ARR2CAT) 5.0.81 on page 357

⇒ "RectangularMatrixCategory" (RMATCAT) 10.0.147 on page 901
⇒ "SquareMatrixCategory" (SMATCAT) 12.0.168 on page 1082

Exports:

antisymmetric?	any?	coerce	column	columnSpace
copy	count	determinant	diagonal?	diagonalMatrix
elt	empty	empty?	eq?	eval
every?	exquo	fill!	hash	horizConcat
inverse	latex	less?	listOfLists	map
map!	matrix	maxColIndex	maxRowIndex	member?
members	minColIndex	minordet	minRowIndex	more?
ncols	new	nrows	nullSpace	nullity
parts	pfaffian	qelt	qsetelt!	rank
row	rowEchelon	sample	scalarMatrix	setColumn!
setelt	setRow!	setsubMatrix!	size?	square?
squareTop	subMatrix	swapColumns!	swapRows!	symmetric?
transpose	vertConcat	zero	#?	?**?
?/?	?=?	?~=?	?*?	?+?
-?	?-?			

Attributes Exported:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **nil**

These are directly exported but not implemented:

```
determinant : % -> R if R has commutative *
inverse : % -> Union(%,"failed") if R has FIELD
minordet : % -> R if R has commutative *
nullity : % -> NonNegativeInteger if R has INTDOM
nullSpace : % -> List Col if R has INTDOM
rowEchelon : % -> % if R has EUCDOM
rank : % -> NonNegativeInteger if R has INTDOM
```

These are implemented by this category:

```
antisymmetric? : % -> Boolean
coerce : Col -> %
columnSpace: % -> List Col if R has EUCDOM
diagonal? : % -> Boolean
diagonalMatrix : List % -> %
diagonalMatrix : List R -> %
```

```
elt : (%,List Integer,List Integer) -> %
exquo : (%,R) -> Union(%,"failed") if R has INTDOM
horizConcat : (%,%) -> %
listOfLists : % -> List List R
pfaffian: % -> R if R has COMRING
matrix : List List R -> %
scalarMatrix : (NonNegativeInteger,R) -> %
setelt : (%,List Integer,List Integer,%) -> %
setsubMatrix! : (%,Integer,Integer,%) -> %
square? : % -> Boolean
squareTop : % -> %
subMatrix : (%,Integer,Integer,Integer,Integer) -> %
swapColumns! : (%,Integer,Integer) -> %
swapRows! : (%,Integer,Integer) -> %
symmetric? : % -> Boolean
transpose : Row -> %
transpose : % -> %
vertConcat : (%,%) -> %
zero : (NonNegativeInteger,NonNegativeInteger) -> %
?/? : (%,R) -> % if R has FIELD
?+? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
?*? : (%,R) -> %
?*? : (R,%) -> %
?*? : (Integer,%) -> %
?**? : (%,NonNegativeInteger) -> %
?**? : (%,Integer) -> % if R has FIELD
?*? : (%,Col) -> Col
?*? : (Row,%) -> Row
?*? : (%,%) -> %
```

These exports come from (p357) TwoDimensionalArrayCategory(R,Row,Col) where R:Ring, Row:FiniteLinearAggregate(R), Col:FiniteLinearAggregate(R):

```
any? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
coerce : % -> OutputForm if R has SETCAT
column : (%,Integer) -> Col
copy : % -> %
count : (R,%) -> NonNegativeInteger if R has SETCAT and $ has finiteAggregate
count : ((R -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
elt : (%,Integer,Integer,R) -> R
elt : (%,Integer,Integer) -> R
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List R,List R) -> % if R has EVALAB R and R has SETCAT
eval : (%,R,R) -> % if R has EVALAB R and R has SETCAT
eval : (%,Equation R) -> % if R has EVALAB R and R has SETCAT
```

```
eval : (%,List Equation R) -> % if R has EVALAB R and R has SETCAT
every? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
fill! : (%,R) -> %
hash : % -> SingleInteger if R has SETCAT
latex : % -> String if R has SETCAT
less? : (%,NonNegativeInteger) -> Boolean
map : (((R,R) -> R),%,%,R) -> %
map : (((R,R) -> R),%,%) -> %
map : ((R -> R),%) -> %
map! : ((R -> R),%) -> %
maxColIndex : % -> Integer
maxRowIndex : % -> Integer
member? : (R,%) -> Boolean if R has SETCAT and $ has finiteAggregate
members : % -> List R if $ has finiteAggregate
minColIndex : % -> Integer
minRowIndex : % -> Integer
more? : (%,NonNegativeInteger) -> Boolean
new : (NonNegativeInteger,NonNegativeInteger,R) -> %
ncols : % -> NonNegativeInteger
nrows : % -> NonNegativeInteger
parts : % -> List R
qelt : (%,Integer,Integer) -> R
qsetelt! : (%,Integer,Integer,R) -> R
row : (%,Integer) -> Row
sample : () -> %
setColumn! : (%,Integer,Col) -> %
setelt : (%,Integer,Integer,R) -> R
setRow! : (%,Integer,Row) -> %
size? : (%,NonNegativeInteger) -> Boolean
#? : % -> NonNegativeInteger if $ has finiteAggregate
?~=? : (%,%) -> Boolean if R has SETCAT
?=? : (%,%) -> Boolean if R has SETCAT
```

— category MATCAT MatrixCategory —

```
)abbrev category MATCAT MatrixCategory
++ Authors: Grabmeier, Gschnitzer, Williamson
++ Date Created: 1987
++ Date Last Updated: July 1990
++ Basic Operations:
++ Related Domains: Matrix(R)
++ Also See:
++ AMS Classifications:
++ Keywords: matrix, linear algebra
++ Examples:
++ References:
++ Description:
++ \spadtype{MatrixCategory} is a general matrix category which allows
++ different representations and indexing schemes.  Rows and
```

```
++ columns may be extracted with rows returned as objects of
++ type Row and colums returned as objects of type Col.
++ A domain belonging to this category will be shallowly mutable.
++ The index of the 'first' row may be obtained by calling the
++ function \spadfun{minRowIndex}.  The index of the 'first' column may
++ be obtained by calling the function \spadfun{minColIndex}.  The index of
++ the first element of a Row is the same as the index of the
++ first column in a matrix and vice versa.

MatrixCategory(R,Row,Col): Category == Definition where
  R   : Ring
  Row : FiniteLinearAggregate R
  Col : FiniteLinearAggregate R

  Definition ==> TwoDimensionalArrayCategory(R,Row,Col) with
     shallowlyMutable
       ++ One may destructively alter matrices

     finiteAggregate
       ++ matrices are finite

--% Predicates

     square?  : % -> Boolean
       ++ \spad{square?(m)} returns true if m is a square matrix
       ++ (if m has the same number of rows as columns) and false otherwise.
       ++
       ++X square matrix [[j**i for i in 0..4] for j in 1..5]

     diagonal?: % -> Boolean
       ++ \spad{diagonal?(m)} returns true if the matrix m is square and
       ++ diagonal (i.e. all entries of m not on the diagonal are zero) and
       ++ false otherwise.
       ++
       ++X diagonal? matrix [[j**i for i in 0..4] for j in 1..5]

     symmetric?: % -> Boolean
       ++ \spad{symmetric?(m)} returns true if the matrix m is square and
       ++ symmetric (i.e. \spad{m[i,j] = m[j,i]} for all i and j) and false
       ++ otherwise.
       ++
       ++X symmetric? matrix [[j**i for i in 0..4] for j in 1..5]

     antisymmetric?: % -> Boolean
       ++ \spad{antisymmetric?(m)} returns true if the matrix m is square and
       ++ antisymmetric (i.e. \spad{m[i,j] = -m[j,i]} for all i and j)
       ++ and false otherwise.
       ++
       ++X antisymmetric? matrix [[j**i for i in 0..4] for j in 1..5]
```

```
--% Creation

     zero: (NonNegativeInteger,NonNegativeInteger) -> %
       ++ \spad{zero(m,n)} returns an m-by-n zero matrix.
       ++
       ++X z:Matrix(INT):=zero(3,3)

     matrix: List List R -> %
       ++ \spad{matrix(l)} converts the list of lists l to a matrix, where the
       ++ list of lists is viewed as a list of the rows of the matrix.
       ++
       ++X matrix [[1,2,3],[4,5,6],[7,8,9],[1,1,1]]

     scalarMatrix: (NonNegativeInteger,R) -> %
       ++ \spad{scalarMatrix(n,r)} returns an n-by-n matrix with r's on the
       ++ diagonal and zeroes elsewhere.
       ++
       ++X z:Matrix(INT):=scalarMatrix(3,5)

     diagonalMatrix: List R -> %
       ++ \spad{diagonalMatrix(l)} returns a diagonal matrix with the elements
       ++ of l on the diagonal.
       ++
       ++X diagonalMatrix [1,2,3]

     diagonalMatrix: List % -> %
       ++ \spad{diagonalMatrix([m1,...,mk])} creates a block diagonal matrix
       ++ M with block matrices m1,...,mk down the diagonal,
       ++ with 0 block matrices elsewhere.
       ++ More precisly: if \spad{ri := nrows mi}, \spad{ci := ncols mi},
       ++ then m is an (r1+..+rk) by (c1+..+ck) - matrix  with entries
       ++ \spad{m.i.j = ml.(i-r1-..-r(l-1)).(j-n1-..-n(l-1))}, if
       ++ \spad{(r1+..+r(l-1)) < i <= r1+..+rl} and
       ++ \spad{(c1+..+c(l-1)) < i <= c1+..+cl},
       ++ \spad{m.i.j} = 0  otherwise.
       ++
       ++X diagonalMatrix [matrix [[1,2],[3,4]], matrix [[4,5],[6,7]]]

     coerce: Col -> %
       ++ \spad{coerce(col)} converts the column col to a column matrix.
       ++
       ++X coerce([1,2,3])@Matrix(INT)

     transpose: Row -> %
       ++ \spad{transpose(r)} converts the row r to a row matrix.
       ++
       ++X transpose([1,2,3])@Matrix(INT)

--% Creation of new matrices from old
```

```
    transpose: % -> %
      ++ \spad{transpose(m)} returns the transpose of the matrix m.
      ++
      ++X m:=matrix [[j**i for i in 0..4] for j in 1..5]
      ++X transpose m

    squareTop: % -> %
      ++ \spad{squareTop(m)} returns an n-by-n matrix consisting of the first
      ++ n rows of the m-by-n matrix m. Error: if
      ++ \spad{m < n}.
      ++
      ++X m:=matrix [[j**i for i in 0..2] for j in 1..5]
      ++X squareTop m

    horizConcat: (%,%) -> %
      ++ \spad{horizConcat(x,y)} horizontally concatenates two matrices with
      ++ an equal number of rows. The entries of y appear to the right
      ++ of the entries of x.  Error: if the matrices
      ++ do not have the same number of rows.
      ++
      ++X m:=matrix [[j**i for i in 0..4] for j in 1..5]
      ++X horizConcat(m,m)

    vertConcat: (%,%) -> %
      ++ \spad{vertConcat(x,y)} vertically concatenates two matrices with an
      ++ equal number of columns. The entries of y appear below
      ++ of the entries of x.  Error: if the matrices
      ++ do not have the same number of columns.
      ++
      ++X m:=matrix [[j**i for i in 0..4] for j in 1..5]
      ++X vertConcat(m,m)

--% Part extractions/assignments

    listOfLists: % -> List List R
      ++ \spad{listOfLists(m)} returns the rows of the matrix m as a list
      ++ of lists.
      ++
      ++X m:=matrix [[j**i for i in 0..4] for j in 1..5]
      ++X listOfLists m

    elt: (%,List Integer,List Integer) -> %
      ++ \spad{elt(x,rowList,colList)} returns an m-by-n matrix consisting
      ++ of elements of x, where \spad{m = # rowList} and \spad{n = # colList}
      ++ If \spad{rowList = [i<1>,i<2>,...,i<m>]} and \spad{colList =
      ++ [j<1>,j<2>,...,j<n>]}, then the \spad{(k,l)}th entry of
      ++ \spad{elt(x,rowList,colList)} is \spad{x(i<k>,j<l>)}.
      ++
      ++X m:=matrix [[j**i for i in 0..4] for j in 1..5]
```

```
    ++X elt(m,3,3)

  setelt: (%,List Integer,List Integer, %) -> %
    ++ \spad{setelt(x,rowList,colList,y)} destructively alters the matrix x.
    ++ If y is \spad{m}-by-\spad{n}, \spad{rowList = [i<1>,i<2>,...,i<m>]}
    ++ and \spad{colList = [j<1>,j<2>,...,j<n>]}, then \spad{x(i<k>,j<l>)}
    ++ is set to \spad{y(k,l)} for \spad{k = 1,...,m} and \spad{l = 1,...,n}
    ++
    ++X m:=matrix [[j**i for i in 0..4] for j in 1..5]
    ++X setelt(m,3,3,10)

  swapRows_!: (%,Integer,Integer) -> %
    ++ \spad{swapRows!(m,i,j)} interchanges the \spad{i}th and \spad{j}th
    ++ rows of m. This destructively alters the matrix.
    ++
    ++X m:=matrix [[j**i for i in 0..4] for j in 1..5]
    ++X swapRows!(m,2,4)

  swapColumns_!: (%,Integer,Integer) -> %
    ++ \spad{swapColumns!(m,i,j)} interchanges the \spad{i}th and \spad{j}th
    ++ columns of m. This destructively alters the matrix.
    ++
    ++X m:=matrix [[j**i for i in 0..4] for j in 1..5]
    ++X swapColumns!(m,2,4)

  subMatrix: (%,Integer,Integer,Integer,Integer) -> %
    ++ \spad{subMatrix(x,i1,i2,j1,j2)} extracts the submatrix
    ++ \spad{[x(i,j)]} where the index i ranges from \spad{i1} to \spad{i2}
    ++ and the index j ranges from \spad{j1} to \spad{j2}.
    ++
    ++X m:=matrix [[j**i for i in 0..4] for j in 1..5]
    ++X subMatrix(m,1,3,2,4)

  setsubMatrix_!: (%,Integer,Integer,%) -> %
    ++ \spad{setsubMatrix(x,i1,j1,y)} destructively alters the
    ++ matrix x. Here \spad{x(i,j)} is set to \spad{y(i-i1+1,j-j1+1)} for
    ++ \spad{i = i1,...,i1-1+nrows y} and \spad{j = j1,...,j1-1+ncols y}.
    ++
    ++X m:=matrix [[j**i for i in 0..4] for j in 1..5]
    ++X setsubMatrix!(m,2,2,matrix [[3,3],[3,3]])


--% Arithmetic

  "+": (%,%) -> %
    ++ \spad{x + y} is the sum of the matrices x and y.
    ++ Error: if the dimensions are incompatible.
    ++
    ++X m:=matrix [[j**i for i in 0..4] for j in 1..5]
    ++X m+m
```

```
"-": (%,%) -> %
  ++ \spad{x - y} is the difference of the matrices x and y.
  ++ Error: if the dimensions are incompatible.
  ++
  ++X m:=matrix [[j**i for i in 0..4] for j in 1..5]
  ++X m-m

"-":  %    -> %
  ++ \spad{-x} returns the negative of the matrix x.
  ++
  ++X m:=matrix [[j**i for i in 0..4] for j in 1..5]
  ++X -m

"*": (%,%) -> %
  ++ \spad{x * y} is the product of the matrices x and y.
  ++ Error: if the dimensions are incompatible.
  ++
  ++X m:=matrix [[j**i for i in 0..4] for j in 1..5]
  ++X m*m

"*": (R,%) -> %
  ++ \spad{r*x} is the left scalar multiple of the scalar r and the
  ++ matrix x.
  ++
  ++X m:=matrix [[j**i for i in 0..4] for j in 1..5]
  ++X 1/3*m

"*": (%,R) -> %
  ++ \spad{x * r} is the right scalar multiple of the scalar r and the
  ++ matrix x.
  ++
  ++X m:=matrix [[j**i for i in 0..4] for j in 1..5]
  ++X m*1/3

"*": (Integer,%) -> %
  ++ \spad{n * x} is an integer multiple.
  ++
  ++X m:=matrix [[j**i for i in 0..4] for j in 1..5]
  ++X 3*m

"*": (%,Col) -> Col
  ++ \spad{x * c} is the product of the matrix x and the column vector c.
  ++ Error: if the dimensions are incompatible.
  ++
  ++X m:=matrix [[j**i for i in 0..4] for j in 1..5]
  ++X c:=coerce([1,2,3,4,5])@Matrix(INT)
  ++X m*c

"*": (Row,%) -> Row
```

```
      ++ \spad{r * x} is the product of the row vector r and the matrix x.
      ++ Error: if the dimensions are incompatible.
      ++
      ++X m:=matrix [[j**i for i in 0..4] for j in 1..5]
      ++X r:=transpose([1,2,3,4,5])@Matrix(INT)
      ++X r*m

    "**": (%,NonNegativeInteger) -> %
      ++ \spad{x ** n} computes a non-negative integral power of the matrix x.
      ++ Error: if the matrix is not square.
      ++
      ++X m:=matrix [[j**i for i in 0..4] for j in 1..5]
      ++X m**3

    if R has IntegralDomain then
      "exquo": (%,R) -> Union(%,"failed")
        ++ \spad{exquo(m,r)} computes the exact quotient of the elements
        ++ of m by r, returning \axiom{"failed"} if this is not possible.
        ++
        ++X m:=matrix [[2**i for i in 2..4] for j in 1..5]
        ++X exquo(m,2)

    if R has Field then
      "/": (%,R) -> %
        ++ \spad{m/r} divides the elements of m by r. Error: if \spad{r = 0}.
        ++
        ++X m:=matrix [[2**i for i in 2..4] for j in 1..5]
        ++X m/4

--% Linear algebra

    if R has EuclideanDomain then
      rowEchelon: % -> %
        ++ \spad{rowEchelon(m)} returns the row echelon form of the matrix m.
        ++
        ++X rowEchelon matrix [[j**i for i in 0..4] for j in 1..5]

      columnSpace: % -> List Col
        ++ \spad{columnSpace(m)} returns a sublist of columns of the matrix m
        ++ forming a basis of its column space
        ++
        ++X columnSpace matrix [[1,2,3],[4,5,6],[7,8,9],[1,1,1]]

    if R has IntegralDomain then
      rank: % -> NonNegativeInteger
        ++ \spad{rank(m)} returns the rank of the matrix m.
        ++
        ++X rank matrix [[1,2,3],[4,5,6],[7,8,9]]

      nullity: % -> NonNegativeInteger
```

```
      ++ \spad{nullity(m)} returns the nullity of the matrix m. This is
      ++ the dimension of the null space of the matrix m.
      ++
      ++X nullity matrix [[1,2,3],[4,5,6],[7,8,9]]

    nullSpace: % -> List Col
      ++ \spad{nullSpace(m)} returns a basis for the null space of
      ++ the matrix m.
      ++
      ++X nullSpace matrix [[1,2,3],[4,5,6],[7,8,9]]

  if R has commutative("*") then
    determinant: % -> R
      ++ \spad{determinant(m)} returns the determinant of the matrix m.
      ++ Error: if the matrix is not square.
      ++
      ++X determinant matrix [[j**i for i in 0..4] for j in 1..5]

    minordet: % -> R
      ++ \spad{minordet(m)} computes the determinant of the matrix m using
      ++ minors. Error: if the matrix is not square.
      ++
      ++X minordet matrix [[j**i for i in 0..4] for j in 1..5]

  if R has CommutativeRing then
    pfaffian: % -> R
      ++ \spad{pfaffian(m)} returns the Pfaffian of the matrix m.
      ++ Error if the matrix is not antisymmetric
      ++
      ++X pfaffian [[0,1,0,0],[-1,0,0,0],[0,0,0,1],[0,0,-1,0]]
  if R has Field then
    inverse: % -> Union(%,"failed")
      ++ \spad{inverse(m)} returns the inverse of the matrix m.
      ++ If the matrix is not invertible, "failed" is returned.
      ++ Error: if the matrix is not square.
      ++
      ++X inverse matrix [[j**i for i in 0..4] for j in 1..5]

    "**": (%,Integer) -> %
      ++ \spad{m**n} computes an integral power of the matrix m.
      ++ Error: if matrix is not square or if the matrix
      ++ is square but not invertible.
      ++
      ++X (matrix [[j**i for i in 0..4] for j in 1..5]) ** 2

  add
    minr ==> minRowIndex
    maxr ==> maxRowIndex
    minc ==> minColIndex
    maxc ==> maxColIndex
```

```
    mini ==> minIndex
    maxi ==> maxIndex

--% Predicates

    square? x == nrows x = ncols x

    diagonal? x ==
      not square? x => false
      for i in minr x .. maxr x repeat
        for j in minc x .. maxc x | (j - minc x) ^= (i - minr x) repeat
          not zero? qelt(x, i, j) => return false
      true

    symmetric? x ==
      (nRows := nrows x) ^= ncols x => false
      mr := minRowIndex x; mc := minColIndex x
      for i in 0..(nRows - 1) repeat
        for j in (i + 1)..(nRows - 1) repeat
          qelt(x,mr + i,mc + j) ^= qelt(x,mr + j,mc + i) => return false
      true

    antisymmetric? x ==
      (nRows := nrows x) ^= ncols x => false
      mr := minRowIndex x; mc := minColIndex x
      for i in 0..(nRows - 1) repeat
        for j in i..(nRows - 1) repeat
          qelt(x,mr + i,mc + j) ^= -qelt(x,mr + j,mc + i) =>
            return false
      true

--% Creation of matrices

    zero(rows,cols) == new(rows,cols,0)

    matrix(l: List List R) ==
      null l => new(0,0,0)
      -- error check: this is a top level function
      rows : NonNegativeInteger := 1; cols := # first l
      cols = 0 => error "matrices with zero columns are not supported"
      for ll in rest l repeat
        cols ^= # ll => error "matrix: rows of different lengths"
        rows := rows + 1
      ans := new(rows,cols,0)
      for i in minr(ans)..maxr(ans) for ll in l repeat
        for j in minc(ans)..maxc(ans) for r in ll repeat
          qsetelt_!(ans,i,j,r)
      ans

    scalarMatrix(n,r) ==
```

```
  ans := zero(n,n)
  for i in minr(ans)..maxr(ans) for j in minc(ans)..maxc(ans) repeat
    qsetelt_!(ans,i,j,r)
  ans

diagonalMatrix(l: List R) ==
  n := #l; ans := zero(n,n)
  for i in minr(ans)..maxr(ans) for j in minc(ans)..maxc(ans) _
      for r in l repeat qsetelt_!(ans,i,j,r)
  ans

diagonalMatrix(list: List %) ==
  rows : NonNegativeInteger := 0
  cols : NonNegativeInteger := 0
  for mat in list repeat
    rows := rows + nrows mat
    cols := cols + ncols mat
  ans := zero(rows,cols)
  loR := minr ans; loC := minc ans
  for mat in list repeat
    hiR := loR + nrows(mat) - 1; hiC := loC + nrows(mat) - 1
    for i in loR..hiR for k in minr(mat)..maxr(mat) repeat
      for j in loC..hiC for l in minc(mat)..maxc(mat) repeat
        qsetelt_!(ans,i,j,qelt(mat,k,l))
    loR := hiR + 1; loC := hiC + 1
  ans

coerce(v:Col) ==
  x := new(#v,1,0)
  one := minc(x)
  for i in minr(x)..maxr(x) for k in mini(v)..maxi(v) repeat
    qsetelt_!(x,i,one,qelt(v,k))
  x

transpose(v:Row) ==
  x := new(1,#v,0)
  one := minr(x)
  for j in minc(x)..maxc(x) for k in mini(v)..maxi(v) repeat
    qsetelt_!(x,one,j,qelt(v,k))
  x

transpose(x:%) ==
  ans := new(ncols x,nrows x,0)
  for i in minr(ans)..maxr(ans) repeat
    for j in minc(ans)..maxc(ans) repeat
      qsetelt_!(ans,i,j,qelt(x,j,i))
  ans

squareTop x ==
  nrows x < (cols := ncols x) =>
```

```
        error "squareTop: number of columns exceeds number of rows"
      ans := new(cols,cols,0)
      for i in minr(x)..(minr(x) + cols - 1) repeat
        for j in minc(x)..maxc(x) repeat
          qsetelt_!(ans,i,j,qelt(x,i,j))
      ans

    horizConcat(x,y) ==
      (rows := nrows x) ^= nrows y =>
        error "HConcat: matrices must have same number of rows"
      ans := new(rows,(cols := ncols x) + ncols y,0)
      for i in minr(x)..maxr(x) repeat
        for j in minc(x)..maxc(x) repeat
          qsetelt_!(ans,i,j,qelt(x,i,j))
      for i in minr(y)..maxr(y) repeat
        for j in minc(y)..maxc(y) repeat
          qsetelt_!(ans,i,j + cols,qelt(y,i,j))
      ans

    vertConcat(x,y) ==
      (cols := ncols x) ^= ncols y =>
        error "HConcat: matrices must have same number of columns"
      ans := new((rows := nrows x) + nrows y,cols,0)
      for i in minr(x)..maxr(x) repeat
        for j in minc(x)..maxc(x) repeat
          qsetelt_!(ans,i,j,qelt(x,i,j))
      for i in minr(y)..maxr(y) repeat
        for j in minc(y)..maxc(y) repeat
          qsetelt_!(ans,i + rows,j,qelt(y,i,j))
      ans

--% Part extraction/assignment

    listOfLists x ==
      ll : List List R := nil()
      for i in maxr(x)..minr(x) by -1 repeat
        l : List R := nil()
        for j in maxc(x)..minc(x) by -1 repeat
          l := cons(qelt(x,i,j),l)
        ll := cons(l,ll)
      ll

    swapRows_!(x,i1,i2) ==
      (i1 < minr(x)) or (i1 > maxr(x)) or (i2 < minr(x)) or _
          (i2 > maxr(x)) => error "swapRows!: index out of range"
      i1 = i2 => x
      for j in minc(x)..maxc(x) repeat
        r := qelt(x,i1,j)
        qsetelt_!(x,i1,j,qelt(x,i2,j))
        qsetelt_!(x,i2,j,r)
```

```
  x

swapColumns_!(x,j1,j2) ==
  (j1 < minc(x)) or (j1 > maxc(x)) or (j2 < minc(x)) or _
      (j2 > maxc(x)) => error "swapColumns!: index out of range"
  j1 = j2 => x
  for i in minr(x)..maxr(x) repeat
    r := qelt(x,i,j1)
    qsetelt_!(x,i,j1,qelt(x,i,j2))
    qsetelt_!(x,i,j2,r)
  x

elt(x:%,rowList:List Integer,colList:List Integer) ==
  for ei in rowList repeat
    (ei < minr(x)) or (ei > maxr(x)) =>
      error "elt: index out of range"
  for ej in colList repeat
    (ej < minc(x)) or (ej > maxc(x)) =>
      error "elt: index out of range"
  y := new(# rowList,# colList,0)
  for ei in rowList for i in minr(y)..maxr(y) repeat
    for ej in colList for j in minc(y)..maxc(y) repeat
      qsetelt_!(y,i,j,qelt(x,ei,ej))
  y

setelt(x:%,rowList:List Integer,colList:List Integer,y:%) ==
  for ei in rowList repeat
    (ei < minr(x)) or (ei > maxr(x)) =>
      error "setelt: index out of range"
  for ej in colList repeat
    (ej < minc(x)) or (ej > maxc(x)) =>
      error "setelt: index out of range"
  ((# rowList) ^= (nrows y)) or ((# colList) ^= (ncols y)) =>
    error "setelt: matrix has bad dimensions"
  for ei in rowList for i in minr(y)..maxr(y) repeat
    for ej in colList for j in minc(y)..maxc(y) repeat
      qsetelt_!(x,ei,ej,qelt(y,i,j))
  y

subMatrix(x,i1,i2,j1,j2) ==
  (i2 < i1) => error "subMatrix: bad row indices"
  (j2 < j1) => error "subMatrix: bad column indices"
  (i1 < minr(x)) or (i2 > maxr(x)) =>
    error "subMatrix: index out of range"
  (j1 < minc(x)) or (j2 > maxc(x)) =>
    error "subMatrix: index out of range"
  rows := (i2 - i1 + 1) pretend NonNegativeInteger
  cols := (j2 - j1 + 1) pretend NonNegativeInteger
  y := new(rows,cols,0)
  for i in minr(y)..maxr(y) for k in i1..i2 repeat
```

```
        for j in minc(y)..maxc(y) for l in j1..j2 repeat
          qsetelt_!(y,i,j,qelt(x,k,l))
      y

    setsubMatrix_!(x,i1,j1,y) ==
      i2 := i1 + nrows(y) -1
      j2 := j1 + ncols(y) -1
      (i1 < minr(x)) or (i2 > maxr(x)) =>
       error _
        "setsubMatrix!: inserted matrix too big, use subMatrix to restrict it"
      (j1 < minc(x)) or (j2 > maxc(x)) =>
       error _
        "setsubMatrix!: inserted matrix too big, use subMatrix to restrict it"
      for i in minr(y)..maxr(y) for k in i1..i2 repeat
        for j in minc(y)..maxc(y) for l in j1..j2 repeat
          qsetelt_!(x,k,l,qelt(y,i,j))
      x

--% Arithmetic

    x + y ==
      ((r := nrows x) ^= nrows y) or ((c := ncols x) ^= ncols y) =>
        error "can't add matrices of different dimensions"
      ans := new(r,c,0)
      for i in minr(x)..maxr(x) repeat
        for j in minc(x)..maxc(x) repeat
          qsetelt_!(ans,i,j,qelt(x,i,j) + qelt(y,i,j))
      ans

    x - y ==
      ((r := nrows x) ^= nrows y) or ((c := ncols x) ^= ncols y) =>
        error "can't subtract matrices of different dimensions"
      ans := new(r,c,0)
      for i in minr(x)..maxr(x) repeat
        for j in minc(x)..maxc(x) repeat
          qsetelt_!(ans,i,j,qelt(x,i,j) - qelt(y,i,j))
      ans

    - x == map((r1:R):R +-> - r1,x)

    a:R * x:% == map((r1:R):R +-> a * r1,x)

    x:% * a:R == map((r1:R):R +-> r1 * a,x)

    m:Integer * x:% == map((r1:R):R +-> m * r1,x)

    x:% * y:% ==
      (ncols x ^= nrows y) =>
        error "can't multiply matrices of incompatible dimensions"
      ans := new(nrows x,ncols y,0)
```

```
      for i in minr(x)..maxr(x) repeat
        for j in minc(y)..maxc(y) repeat
          entry :=
            sum : R := 0
            for k in minr(y)..maxr(y) for l in minc(x)..maxc(x) repeat
              sum := sum + qelt(x,i,l) * qelt(y,k,j)
            sum
          qsetelt_!(ans,i,j,entry)
      ans

    positivePower:(%,Integer) -> %
    positivePower(x,n) ==
--      one? n => x
      (n = 1) => x
      odd? n => x * positivePower(x,n - 1)
      y := positivePower(x,n quo 2)
      y * y

    x:% ** n:NonNegativeInteger ==
      not((nn:= nrows x) = ncols x) => error "**: matrix must be square"
      zero? n => scalarMatrix(nn,1)
      positivePower(x,n)

    --if R has ConvertibleTo InputForm then
      --convert(x:%):InputForm ==
        --convert [convert("matrix"::Symbol)@InputForm,
                  --convert listOfLists x]$List(InputForm)

    if Col has shallowlyMutable then

      x:% * v:Col ==
        ncols(x) ^= #v =>
          error "can't multiply matrix A and vector v if #cols A ^= #v"
        w : Col := new(nrows x,0)
        for i in minr(x)..maxr(x) for k in mini(w)..maxi(w) repeat
          w.k :=
            sum : R := 0
            for j in minc(x)..maxc(x) for l in mini(v)..maxi(v) repeat
              sum := sum + qelt(x,i,j) * v(l)
            sum
        w

    if Row has shallowlyMutable then

      v:Row * x:% ==
        nrows(x) ^= #v =>
          error "can't multiply vector v and matrix A if #rows A ^= #v"
        w : Row := new(ncols x,0)
        for j in minc(x)..maxc(x) for k in mini(w)..maxi(w) repeat
          w.k :=
```

```
        sum : R := 0
        for i in minr(x)..maxr(x) for l in mini(v)..maxi(v) repeat
          sum := sum + qelt(x,i,j) * v(l)
        sum
    w

if R has EuclideanDomain then
  columnSpace M ==
    M2 := rowEchelon M
    basis: List Col := []
    n: Integer := ncols M
    m: Integer := nrows M
    indRow: Integer := 1
    for k in 1..n while indRow <= m repeat
      if not zero?(M2.(indRow,k)) then
        basis := cons(column(M,k),basis)
        indRow := indRow + 1
    reverse! basis

if R has CommutativeRing then
  skewSymmetricUnitMatrix(n:PositiveInteger):% ==
    matrix [[(if i=j+1 and odd? j
                then -1
                else if i=j-1 and odd? i
                       then 1
                       else 0) for j in 1..n] for i in 1..n]

  SUPR ==> SparseUnivariatePolynomial R

  PfChar(A:%):SUPR ==
    n := nrows A
    (n = 2) => monomial(1$R,2)$SUPR + qelt(A,1,2)::SUPR
    M:=subMatrix(A,3,n,3,n)
    r:=subMatrix(A,1,1,3,n)
    s:=subMatrix(A,3,n,2,2)
    p:=PfChar(M)
    d:=degree(p)$SUPR
    B:=skewSymmetricUnitMatrix((n-2)::PositiveInteger)
    C:=r*B
    g:List R := [qelt(C*s,1,1), qelt(A,1,2), 1]
    if d >= 4 then
      B:=M*B
      for i in 4..d by 2 repeat
        C:=C*B
        g:=cons(qelt(C*s,1,1),g)
    g:=reverse! g
    res:SUPR := 0
    for i in 0..d by 2 for j in 2..d+2 repeat
      c:=coefficient(p,i)
      for e in first(g,j) for k in 2..-d by -2 repeat
```

```
            res:=res+monomial(c*e,(k+i)::NonNegativeInteger)$SUPR
          res

        pfaffian a ==
          if antisymmetric? a
            then if odd? nrows a
                   then 0
                   else PfChar(a).0
            else
              error "pfaffian: only defined for antisymmetric square matrices"

      if R has IntegralDomain then
        x exquo a ==
          ans := new(nrows x,ncols x,0)
          for i in minr(x)..maxr(x) repeat
            for j in minc(x)..maxc(x) repeat
              entry :=
                (r := (qelt(x,i,j) exquo a)) case "failed" =>
                  return "failed"
                r :: R
              qsetelt_!(ans,i,j,entry)
          ans

      if R has Field then
        x / r == map((r1:R):R +-> r1 / r,x)

        x:% ** n:Integer ==
          not((nn:= nrows x) = ncols x) => error "**: matrix must be square"
          zero? n => scalarMatrix(nn,1)
          positive? n => positivePower(x,n)
          (xInv := inverse x) case "failed" =>
            error "**: matrix must be invertible"
          positivePower(xInv :: %,-n)
```

———-

— MATCAT.dotabb —

```
"MATCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=MATCAT"];
"MATCAT" -> "ARR2CAT"
```

———-

— MATCAT.dotfull —

```
"MatrixCategory(a:Ring,b:FiniteLinearAggregate(a),c:FiniteLinearAggregate(a)"
```

```
 [color=lightblue,href="bookvol10.2.pdf#nameddest=MATCAT"];
"MatrixCategory(a:Ring,b:FiniteLinearAggregate(a),c:FiniteLinearAggregate(a)"
 ->
"TwoDimensionalArrayCategory(a:Type,b:FiniteLinearAggregate(a),c:FiniteLinearAggregate(a))"
```

———-

— MATCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"MatrixCategory(a:Ring,b:FiniteLinearAggregate(a),c:FiniteLinearAggregate(a)"
 [color=lightblue];
"MatrixCategory(a:Ring,b:FiniteLinearAggregate(a),c:FiniteLinearAggregate(a)"
 ->
"TwoDimensionalArrayCategory(a:Type,b:FiniteLinearAggregate(a),c:FiniteLinearAggregate(a))"

"TwoDimensionalArrayCategory(a:Type,b:FiniteLinearAggregate(a),c:FiniteLinearAggregate(a))"
 [color=lightblue];
"TwoDimensionalArrayCategory(a:Type,b:FiniteLinearAggregate(a),c:FiniteLinearAggregate(a))"
    -> "HomogeneousAggregate(a:Type)"

"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "Aggregate()"
"HomogeneousAggregate(a:Type)" -> "Evalable(a:Type)"
"HomogeneousAggregate(a:Type)" -> "SetCategory()"

"Evalable(a:Type)" [color="#00EE00"];

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Aggregate()" [color=lightblue];
"Aggregate()" -> "Type()"
```

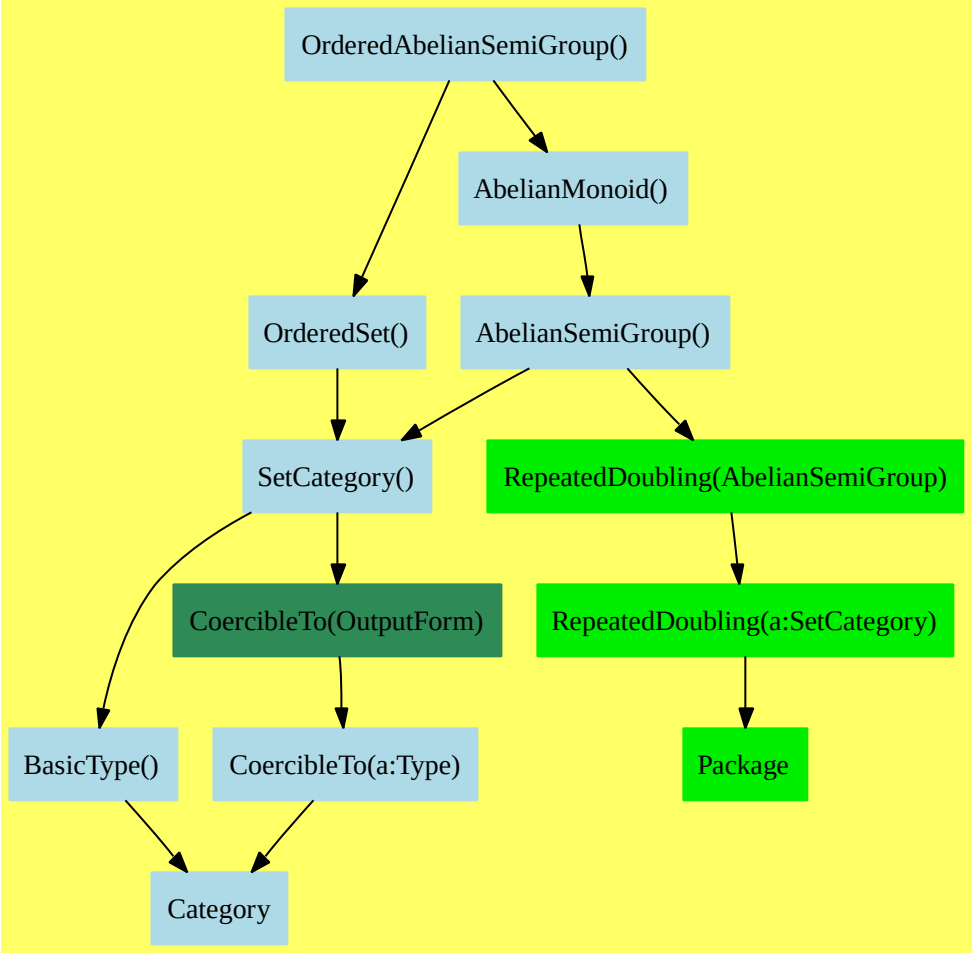
```
"Type()" [color=lightblue];
"Type()" -> "Category"

"Category" [color=lightblue];

}
```

———-

6.0.89 OrderedAbelianSemiGroup (OASGP)



— OrderedAbelianSemiGroup.input —

```
)set break resume
)sys rm -f OrderedAbelianSemiGroup.output
)spool OrderedAbelianSemiGroup.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show OrderedAbelianSemiGroup
--R OrderedAbelianSemiGroup  is a category constructor
--R Abbreviation for OrderedAbelianSemiGroup is OASGP
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for OASGP
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (PositiveInteger,%) -> %        ?+? : (%,%) -> %
--R ?<? : (%,%) -> Boolean                ?<=? : (%,%) -> Boolean
--R ?=? : (%,%) -> Boolean                ?>? : (%,%) -> Boolean
--R ?>=? : (%,%) -> Boolean               0 : () -> %
--R coerce : % -> OutputForm              hash : % -> SingleInteger
--R latex : % -> String                   max : (%,%) -> %
--R min : (%,%) -> %                      sample : () -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— OrderedAbelianSemiGroup.help —

```
=====================================================================
OrderedAbelianSemiGroup examples
=====================================================================

See Also:
o )show OrderedAbelianSemiGroup
```

———-

See:

⇒ “OrderedAbelianMonoid” (OAMON) 7.0.105 on page 576

⇐ "AbelianMonoid" (ABELMON) 5.0.66 on page 267
⇐ "OrderedSet" (ORDSET) 4.0.56 on page 216

Exports:

0	coerce	hash	latex	max
min	sample	zero?	?~=?	?*?
?+?	?<?	?<=?	?=?	?>?
?>=?				

These exports come from (p216) OrderedSet():

```
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
max : (%,%) -> %
min : (%,%) -> %
?<? : (%,%) -> Boolean
?>? : (%,%) -> Boolean
?<=? : (%,%) -> Boolean
?>=? : (%,%) -> Boolean
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
```

These exports come from (p267) AbelianMonoid():

```
0 : () -> %
sample : () -> %
zero? : % -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?+? : (%,%) -> %
```

— category OASGP OrderedAbelianSemiGroup —

```
)abbrev category OASGP OrderedAbelianSemiGroup
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ Ordered sets which are also abelian semigroups, such that the addition
++ preserves the ordering.\br
++
```

```
++ Axiom\br
++ \tab{5} x < y => x+z < y+z

OrderedAbelianSemiGroup(): Category == Join(OrderedSet, AbelianMonoid)
```

———-

— **OASGP.dotabb** —

```
"OASGP"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OASGP"];
"OASGP" -> "ORDSET"
"OASGP" -> "ABELMON"
```

———-

— **OASGP.dotfull** —

```
"OrderedAbelianSemiGroup()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OASGP"];
"OrderedAbelianSemiGroup()" -> "OrderedSet()"
"OrderedAbelianSemiGroup()" -> "AbelianMonoid()"
```

———-

— **OASGP.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"OrderedAbelianSemiGroup()" [color=lightblue];
"OrderedAbelianSemiGroup()" -> "OrderedSet()"
"OrderedAbelianSemiGroup()" -> "AbelianMonoid()"

"OrderedSet()" [color=lightblue];
"OrderedSet()" -> "SetCategory()"

"AbelianMonoid()" [color=lightblue];
"AbelianMonoid()" -> "AbelianSemiGroup()"

"AbelianSemiGroup()" [color=lightblue];
"AbelianSemiGroup()" -> "SetCategory()"
"AbelianSemiGroup()" -> "RepeatedDoubling(AbelianSemiGroup)"
```

```
"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" ->
    "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

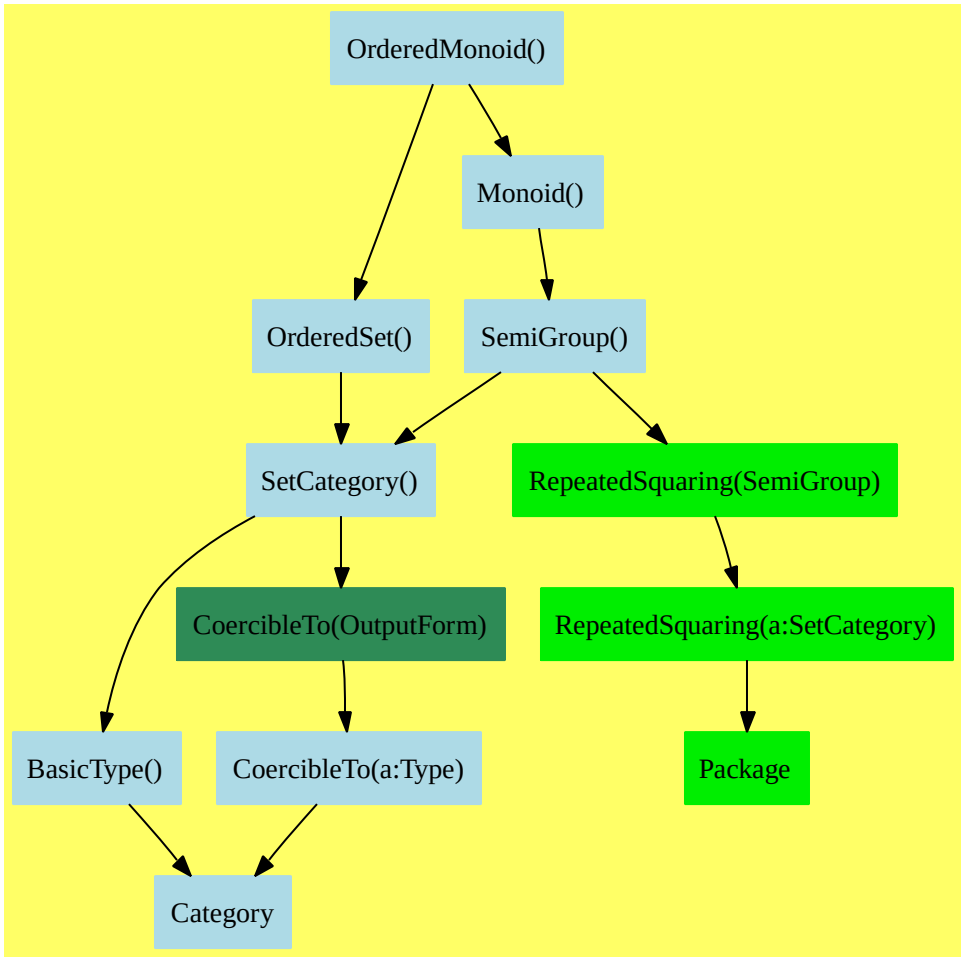
"RepeatedDoubling(AbelianSemiGroup)" [color="#00EE00"];
"RepeatedDoubling(AbelianSemiGroup)" -> "RepeatedDoubling(a:SetCategory)"

"RepeatedDoubling(a:SetCategory)" [color="#00EE00"];
"RepeatedDoubling(a:SetCategory)" -> "Package"

"Package" [color="#00EE00"];

"Category" [color=lightblue];
}
```

6.0.90 OrderedMonoid (ORDMON)



— OrderedMonoid.input —

```
)set break resume
)sys rm -f OrderedMonoid.output
)spool OrderedMonoid.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show OrderedMonoid
--R OrderedMonoid  is a category constructor
```

```
--R Abbreviation for OrderedMonoid is ORDMON
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for ORDMON
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (%,%) -> %                      ?**? : (%,PositiveInteger) -> %
--R ?<? : (%,%) -> Boolean                ?<=? : (%,%) -> Boolean
--R ?=? : (%,%) -> Boolean                ?>? : (%,%) -> Boolean
--R ?>=? : (%,%) -> Boolean               1 : () -> %
--R ?^? : (%,PositiveInteger) -> %        coerce : % -> OutputForm
--R hash : % -> SingleInteger             latex : % -> String
--R max : (%,%) -> %                      min : (%,%) -> %
--R one? : % -> Boolean                   recip : % -> Union(%,"failed")
--R sample : () -> %                      ?~=? : (%,%) -> Boolean
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **OrderedMonoid.help** —

```
====================================================================
OrderedMonoid examples
====================================================================

See Also:
o )show OrderedMonoid
```

———-

See:

⇐ “Monoid” (MONOID) 5.0.76 on page 331
⇐ “OrderedSet” (ORDSET) 4.0.56 on page 216

Exports:

1	coerce	hash	latex	max
min	one?	recip	sample	?*?
?**?	?<?	?<=?	?=?	?>?
?>=?	?~=?	?^?		

These exports come from (p331) Monoid():

```
1 : () -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
?*? : (%,%) -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
```

These exports come from (p216) OrderedSet():

```
max : (%,%) -> %
min : (%,%) -> %
?<? : (%,%) -> Boolean
?<=? : (%,%) -> Boolean
?>? : (%,%) -> Boolean
?>=? : (%,%) -> Boolean
```

— category ORDMON OrderedMonoid —

```
)abbrev category ORDMON OrderedMonoid
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ Ordered sets which are also monoids, such that multiplication
++ preserves the ordering.
++
++ Axioms\br
++ \tab{5}\spad{x < y => x*z < y*z}\br
++ \tab{5}\spad{x < y => z*x < z*y}

OrderedMonoid(): Category == Join(OrderedSet, Monoid)
```

———-

— **ORDMON.dotabb** —

```
"ORDMON"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ORDMON"];
"ORDMON" -> "ORDSET"
"ORDMON" -> "MONOID"
```

———-

— **ORDMON.dotfull** —

```
"OrderedMonoid()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ORDMON"];
"OrderedMonoid()" -> "OrderedSet()"
"OrderedMonoid()" -> "Monoid()"
```

———-

— **ORDMON.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"OrderedMonoid()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ORDMON"];
"OrderedMonoid()" -> "OrderedSet()"
"OrderedMonoid()" -> "Monoid()"

"OrderedSet()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ORDSET"];
"OrderedSet()" -> "SetCategory()"

"Monoid()" [color=lightblue];
"Monoid()" -> "SemiGroup()"

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SetCategory()"
"SemiGroup()" -> "RepeatedSquaring(SemiGroup)"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"
```

```
"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"RepeatedSquaring(SemiGroup)" [color="#00EE00"];
"RepeatedSquaring(SemiGroup)" -> "RepeatedSquaring(a:SetCategory)"

"RepeatedSquaring(a:SetCategory)" [color="#00EE00"];
"RepeatedSquaring(a:SetCategory)" -> "Package"

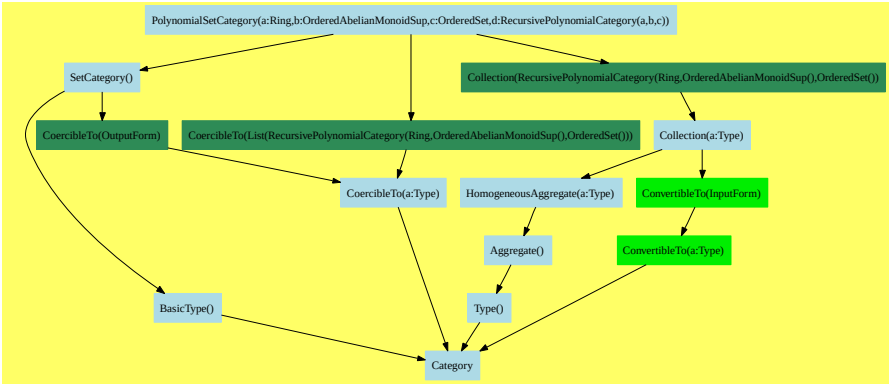
"Package" [color="#00EE00"];

"Category" [color=lightblue];

}
```

———-

6.0.91 PolynomialSetCategory (PSETCAT)



— PolynomialSetCategory.input —

```
)set break resume
)sys rm -f PolynomialSetCategory.output
)spool PolynomialSetCategory.output
```

```
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show PolynomialSetCategory
--R PolynomialSetCategory(R: Ring,E: OrderedAbelianMonoidSup,VarSet: OrderedSet,P: RecursivePoly
--R Abbreviation for PolynomialSetCategory is PSETCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PSETCAT
--R
--R------------------------------ Operations --------------------------------
--R ?=? : (%,%) -> Boolean                coerce : % -> List P
--R coerce : % -> OutputForm              collect : (%,VarSet) -> %
--R collectUnder : (%,VarSet) -> %        collectUpper : (%,VarSet) -> %
--R construct : List P -> %               copy : % -> %
--R empty : () -> %                       empty? : % -> Boolean
--R eq? : (%,%) -> Boolean                hash : % -> SingleInteger
--R latex : % -> String                   mainVariables : % -> List VarSet
--R map : ((P -> P),%) -> %               mvar : % -> VarSet
--R retract : List P -> %                 sample : () -> %
--R trivialIdeal? : % -> Boolean          variables : % -> List VarSet
--R ?~=? : (%,%) -> Boolean
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R any? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
--R convert : % -> InputForm if P has KONVERT INFORM
--R count : ((P -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R count : (P,%) -> NonNegativeInteger if P has SETCAT and $ has finiteAggregate
--R eval : (%,List Equation P) -> % if P has EVALAB P and P has SETCAT
--R eval : (%,Equation P) -> % if P has EVALAB P and P has SETCAT
--R eval : (%,P,P) -> % if P has EVALAB P and P has SETCAT
--R eval : (%,List P,List P) -> % if P has EVALAB P and P has SETCAT
--R every? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
--R find : ((P -> Boolean),%) -> Union(P,"failed")
--R headRemainder : (P,%) -> Record(num: P,den: R) if R has INTDOM
--R less? : (%,NonNegativeInteger) -> Boolean
--R mainVariable? : (VarSet,%) -> Boolean
--R map! : ((P -> P),%) -> % if $ has shallowlyMutable
--R member? : (P,%) -> Boolean if P has SETCAT and $ has finiteAggregate
--R members : % -> List P if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List P if $ has finiteAggregate
--R reduce : (((P,P) -> P),%) -> P if $ has finiteAggregate
--R reduce : (((P,P) -> P),%,P) -> P if $ has finiteAggregate
--R reduce : (((P,P) -> P),%,P,P) -> P if P has SETCAT and $ has finiteAggregate
--R remainder : (P,%) -> Record(rnum: R,polnum: P,den: R) if R has INTDOM
--R remove : ((P -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (P,%) -> % if P has SETCAT and $ has finiteAggregate
--R removeDuplicates : % -> % if P has SETCAT and $ has finiteAggregate
--R retractIfCan : List P -> Union(%,"failed")
```

```
--R rewriteIdealWithHeadRemainder : (List P,%) -> List P if R has INTDOM
--R rewriteIdealWithRemainder : (List P,%) -> List P if R has INTDOM
--R roughBase? : % -> Boolean if R has INTDOM
--R roughEqualIdeals? : (%,%) -> Boolean if R has INTDOM
--R roughSubIdeal? : (%,%) -> Boolean if R has INTDOM
--R roughUnitIdeal? : % -> Boolean if R has INTDOM
--R select : ((P -> Boolean),%) -> % if $ has finiteAggregate
--R size? : (%,NonNegativeInteger) -> Boolean
--R sort : (%,VarSet) -> Record(under: %,floor: %,upper: %)
--R triangular? : % -> Boolean if R has INTDOM
--R
--E 1

)spool
)lisp (bye)
```

———-

— PolynomialSetCategory.help —

```
=====================================================================
PolynomialSetCategory examples
=====================================================================

See Also:
o )show PolynomialSetCategory
```

———-

See:

⇒ “TriangularSetCategory” (TSETCAT) 7.0.108 on page 595
⇐ “CoercibleTo” (KOERCE) 2.0.6 on page 17
⇐ “Collection” (CLAGG) 5.0.70 on page 287
⇐ “SetCategory” (SETCAT) 3.0.34 on page 113

Exports:

any?	coerce	collect
collectUnder	collectUpper	construct
convert	copy	count
empty	empty?	eq?
eval	every?	find
hash	headRemainder	latex
less?	mainVariables	mainVariable?
map	map!	member?
members	more?	mvar
parts	reduce	remainder
remove	removeDuplicates	retract
retractIfCan	rewriteIdealWithHeadRemainder	rewriteIdealWithRemainder
roughBase?	roughEqualIdeals?	roughSubIdeal?
roughUnitIdeal?	sample	select
size?	sort	triangular?
trivialIdeal?	variables	#?
?~=?	?=?	

Attributes Exported:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **nil**

These are directly exported but not implemented:

```
mvar : % -> VarSet
retract : List P -> %
retractIfCan : List P -> Union(%,"failed")
```

These are implemented by this category:

```
collect : (%,VarSet) -> %
collectUnder : (%,VarSet) -> %
collectUpper : (%,VarSet) -> %
headRemainder : (P,%) -> Record(num: P,den: R) if R has INTDOM
mainVariables : % -> List VarSet
mainVariable? : (VarSet,%) -> Boolean
remainder : (P,%) -> Record(rnum: R,polnum: P,den: R) if R has INTDOM
rewriteIdealWithHeadRemainder : (List P,%) -> List P if R has INTDOM
rewriteIdealWithRemainder : (List P,%) -> List P if R has INTDOM
roughBase? : % -> Boolean if R has INTDOM
roughEqualIdeals? : (%,%) -> Boolean if R has INTDOM
roughSubIdeal? : (%,%) -> Boolean if R has INTDOM
roughUnitIdeal? : % -> Boolean if R has INTDOM
sort : (%,VarSet) -> Record(under: %,floor: %,upper: %)
triangular? : % -> Boolean if R has INTDOM
trivialIdeal? : % -> Boolean
variables : % -> List VarSet
?=? : (%,%) -> Boolean
```

These exports come from (p113) SetCategory():

```
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
?~=? : (%,%) -> Boolean
```

These exports come from (p287) Collection(P)
where P:RecursivePolynomialCategory(R,E,V)
where R:Ring, E:OrderedAbelianMonoidSup, V:OrderedSet

```
any? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
construct : List P -> %
copy : % -> %
count : (P,%) -> NonNegativeInteger if P has SETCAT and $ has finiteAggregate
count : ((P -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List Equation P) -> % if P has EVALAB P and P has SETCAT
eval : (%,Equation P) -> % if P has EVALAB P and P has SETCAT
eval : (%,P,P) -> % if P has EVALAB P and P has SETCAT
eval : (%,List P,List P) -> % if P has EVALAB P and P has SETCAT
every? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
find : ((P -> Boolean),%) -> Union(P,"failed")
less? : (%,NonNegativeInteger) -> Boolean
map : ((P -> P),%) -> %
map! : ((P -> P),%) -> % if $ has shallowlyMutable
member? : (P,%) -> Boolean if P has SETCAT and $ has finiteAggregate
members : % -> List P if $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
parts : % -> List P if $ has finiteAggregate
reduce : (((P,P) -> P),%) -> P if $ has finiteAggregate
reduce : (((P,P) -> P),%,P,P) -> P if P has SETCAT and $ has finiteAggregate
reduce : (((P,P) -> P),%,P) -> P if $ has finiteAggregate
remove : (P,%) -> % if P has SETCAT and $ has finiteAggregate
remove : ((P -> Boolean),%) -> % if $ has finiteAggregate
removeDuplicates : % -> % if P has SETCAT and $ has finiteAggregate
sample : () -> %
select : ((P -> Boolean),%) -> % if $ has finiteAggregate
size? : (%,NonNegativeInteger) -> Boolean
#? : % -> NonNegativeInteger if $ has finiteAggregate
```

These exports come from (p17) CoercibleTo(List(P))
where P:RecursivePolynomialCategory(R,E,V)
where R:Ring, E:OrderedAbelianMonoidSup, V:OrderedSet

```
coerce : % -> List P
convert : % -> InputForm if P has KONVERT INFORM
```

These exports come from (p1047) IntegralDomain():

— category PSETCAT PolynomialSetCategory —

```
)abbrev category PSETCAT PolynomialSetCategory
++ Author: Marc Moreno Maza
++ Date Created: 04/26/1994
++ Date Last Updated: 12/15/1998
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords: polynomial, multivariate, ordered variables set
++ References:
++ Description:
++ A category for finite subsets of a polynomial ring.
++ Such a set is only regarded as a set of polynomials and not
++ identified to the ideal it generates. So two distinct sets may
++ generate the same the ideal. Furthermore, for \spad{R} being an
++ integral domain, a set of polynomials may be viewed as a representation
++ of the ideal it generates in the polynomial ring \spad{(R)^(-1) P},
++ or the set of its zeros (described for instance by the radical of the
++ previous ideal, or a split of the associated affine variety) and so on.
++ So this category provides operations about those different notions.

PolynomialSetCategory(R:Ring, E:OrderedAbelianMonoidSup,_
  VarSet:OrderedSet, P:RecursivePolynomialCategory(R,E,VarSet)): Category ==
    Join(SetCategory,Collection(P),CoercibleTo(List(P))) with
     finiteAggregate
     retractIfCan : List(P) -> Union($,"failed")
         ++ \axiom{retractIfCan(lp)} returns an element of the domain
         ++ whose elements are the members of \axiom{lp} if such an element
         ++ exists, otherwise \axiom{"failed"} is returned.
     retract : List(P) -> $
         ++ \axiom{retract(lp)} returns an element of the domain whose elements
         ++ are the members of \axiom{lp} if such an element exists, otherwise
         ++ an error is produced.
     mvar : $ -> VarSet
         ++  \axiom{mvar(ps)} returns the main variable of the non constant
         ++ polynomial with the greatest main variable, if any, else an
         ++ error is returned.
     variables : $ -> List VarSet
         ++  \axiom{variables(ps)} returns the decreasingly sorted list of the
         ++  variables which are variables of some polynomial in \axiom{ps}.
     mainVariables : $  -> List VarSet
         ++ \axiom{mainVariables(ps)} returns the decreasingly sorted list
         ++ of the variables which are main variables of some polynomial
```

```
    ++ in \axiom{ps}.
mainVariable? : (VarSet,$) -> Boolean
    ++ \axiom{mainVariable?(v,ps)} returns true iff \axiom{v} is the
    ++ main variable of some polynomial in \axiom{ps}.
collectUnder : ($,VarSet) -> $
    ++ \axiom{collectUnder(ps,v)} returns the set consisting of the
    ++ polynomials of \axiom{ps} with main variable less than \axiom{v}.
collect : ($,VarSet) -> $
    ++ \axiom{collect(ps,v)}  returns the set consisting of the
    ++ polynomials of \axiom{ps} with \axiom{v} as main variable.
collectUpper : ($,VarSet) -> $
    ++ \axiom{collectUpper(ps,v)} returns the set consisting of the
    ++ polynomials of \axiom{ps} with main variable greater
    ++ than \axiom{v}.
sort : ($,VarSet) -> Record(under:$,floor:$,upper:$)
    ++ \axiom{sort(v,ps)} returns \axiom{us,vs,ws} such that \axiom{us}
    ++ is \axiom{collectUnder(ps,v)}, \axiom{vs} is \axiom{collect(ps,v)}
    ++ and \axiom{ws} is \axiom{collectUpper(ps,v)}.
trivialIdeal?: $ -> Boolean
    ++ \axiom{trivialIdeal?(ps)} returns true iff \axiom{ps} does
    ++ not contain non-zero elements.
if R has IntegralDomain
then
  roughBase? : $ -> Boolean
      ++ \axiom{roughBase?(ps)} returns true iff for every pair
      ++ \axiom{{p,q}} of polynomials in \axiom{ps} their leading
      ++ monomials are relatively prime.
  roughSubIdeal?  : ($,$) -> Boolean
      ++  \axiom{roughSubIdeal?(ps1,ps2)} returns true iff it can proved
      ++ that all polynomials in  \axiom{ps1} lie in the ideal generated
      ++ by \axiom{ps2} in \axiom{\axiom{(R)^(-1) P}} without computing
      ++ Groebner bases.
  roughEqualIdeals? : ($,$) -> Boolean
      ++ \axiom{roughEqualIdeals?(ps1,ps2)} returns true iff it can
      ++ proved that \axiom{ps1} and \axiom{ps2} generate the same ideal
      ++ in \axiom{(R)^(-1) P} without computing Groebner bases.
  roughUnitIdeal? : $ -> Boolean
      ++ \axiom{roughUnitIdeal?(ps)} returns true iff \axiom{ps} contains
      ++  some non null element lying in the base ring \axiom{R}.
  headRemainder : (P,$) -> Record(num:P,den:R)
      ++ \axiom{headRemainder(a,ps)} returns \axiom{[b,r]} such that the
      ++ leading monomial of \axiom{b} is reduced in the sense of
      ++ Groebner bases w.r.t. \axiom{ps} and \axiom{r*a - b} lies in
      ++ the ideal generated by \axiom{ps}.
  remainder : (P,$) -> Record(rnum:R,polnum:P,den:R)
      ++ \axiom{remainder(a,ps)} returns \axiom{[c,b,r]} such that
      ++ \axiom{b} is fully reduced in the sense of Groebner bases
      ++ w.r.t. \axiom{ps}, \axiom{r*a - c*b} lies in the ideal
      ++ generated by \axiom{ps}. Furthermore, if \axiom{R} is a
      ++ gcd-domain, \axiom{b} is primitive.
```

```
      rewriteIdealWithHeadRemainder : (List(P),$) -> List(P)
          ++ \axiom{rewriteIdealWithHeadRemainder(lp,cs)} returns \axiom{lr}
          ++ such that the leading monomial of every polynomial in \axiom{lr}
          ++ is reduced in the sense of Groebner bases w.r.t. \axiom{cs}
          ++ and \axiom{(lp,cs)} and \axiom{(lr,cs)} generate the same
          ++ ideal in \axiom{(R)^(-1) P}.
      rewriteIdealWithRemainder : (List(P),$) -> List(P)
          ++ \axiom{rewriteIdealWithRemainder(lp,cs)} returns \axiom{lr}
          ++ such that every polynomial in \axiom{lr} is fully reduced in
          ++ the sense of Groebner bases w.r.t. \axiom{cs} and
          ++ \axiom{(lp,cs)} and \axiom{(lr,cs)} generate the same ideal
          ++ in \axiom{(R)^(-1) P}.
      triangular? : $ -> Boolean
          ++ \axiom{triangular?(ps)} returns true iff \axiom{ps} is a
          ++ triangular set, i.e. two distinct polynomials have distinct
          ++ main variables and no constant lies in \axiom{ps}.

  add

    NNI ==> NonNegativeInteger
    B ==> Boolean

    elements: $ -> List(P)

    elements(ps:$):List(P) ==
      lp : List(P) := members(ps)$$

    variables1(lp:List(P)):(List VarSet) ==
      lvars : List(List(VarSet)) := [variables(p)$P for p in lp]
      sort((z1:VarSet,z2:VarSet):Boolean +-> z1 > z2,
            removeDuplicates(concat(lvars)$List(VarSet)))

    variables2(lp:List(P)):(List VarSet) ==
      lvars : List(VarSet) := [mvar(p)$P for p in lp]
      sort((z1:VarSet,z2:VarSet):Boolean +-> z1 > z2,
            removeDuplicates(lvars)$List(VarSet))

    variables (ps:$) ==
      variables1(elements(ps))

    mainVariables (ps:$) ==
      variables2(remove(ground?,elements(ps)))

    mainVariable? (v,ps) ==
      lp : List(P) := remove(ground?,elements(ps))
      while (not empty? lp) and (not (mvar(first(lp)) = v)) repeat
        lp := rest lp
      (not empty? lp)

    collectUnder (ps,v) ==
```

```
  lp : List P := elements(ps)
  lq : List P := []
  while (not empty? lp) repeat
    p := first lp
    lp := rest lp
    if (ground?(p)) or (mvar(p) < v)
      then
        lq := cons(p,lq)
  construct(lq)$$

collectUpper (ps,v) ==
  lp : List P := elements(ps)
  lq : List P := []
  while (not empty? lp) repeat
    p := first lp
    lp := rest lp
    if (not ground?(p)) and (mvar(p) > v)
      then
        lq := cons(p,lq)
  construct(lq)$$

collect (ps,v) ==
  lp : List P := elements(ps)
  lq : List P := []
  while (not empty? lp) repeat
    p := first lp
    lp := rest lp
    if (not ground?(p)) and (mvar(p) = v)
      then
        lq := cons(p,lq)
  construct(lq)$$

sort (ps,v) ==
  lp : List P := elements(ps)
  us : List P := []
  vs : List P := []
  ws : List P := []
  while (not empty? lp) repeat
    p := first lp
    lp := rest lp
    if (ground?(p)) or (mvar(p) < v)
      then
        us := cons(p,us)
      else
        if (mvar(p) = v)
          then
            vs := cons(p,vs)
          else
            ws := cons(p,ws)
  [construct(us)$$,_
```

```
    construct(vs)$$,_
    construct(ws)$$]$Record(under:$,floor:$,upper:$)

  ps1 = ps2 ==
    {p for p in elements(ps1)} =$(Set P) {p for p in elements(ps2)}

  exactQuo : (R,R) -> R

  localInf? (p:P,q:P):B ==
    degree(p) <$E degree(q)

  localTriangular? (lp:List(P)):B ==
    lp := remove(zero?, lp)
    empty? lp => true
    any? (ground?, lp) => false
    lp := sort((z1:P,z2:P):Boolean +-> mvar(z1)$P > mvar(z2)$P, lp)
    p,q : P
    p := first lp
    lp := rest lp
    while (not empty? lp) and (mvar(p) > mvar((q := first(lp)))) repeat
      p := q
      lp := rest lp
    empty? lp

  triangular? ps ==
    localTriangular? elements ps

  trivialIdeal? ps ==
    empty?(remove(zero?,elements(ps))$(List(P)))$(List(P))

  if R has IntegralDomain
  then

    roughUnitIdeal? ps ==
      any?(ground?,remove(zero?,elements(ps))$(List(P)))$(List P)

    relativelyPrimeLeadingMonomials? (p:P,q:P):B ==
      dp : E := degree(p)
      dq : E := degree(q)
      (sup(dp,dq)$E =$E dp +$E dq)@B

    roughBase? ps ==
      lp := remove(zero?,elements(ps))$(List(P))
      empty? lp => true
      rB? : B := true
      while (not empty? lp) and rB? repeat
        p := first lp
        lp := rest lp
        copylp := lp
        while (not empty? copylp) and rB? repeat
```

```
        rB? := relativelyPrimeLeadingMonomials?(p,first(copylp))
        copylp := rest copylp
    rB?

  roughSubIdeal?(ps1,ps2) ==
    lp: List(P) := rewriteIdealWithRemainder(elements(ps1),ps2)
    empty? (remove(zero?,lp))

  roughEqualIdeals? (ps1,ps2) ==
    ps1 =$$ ps2 => true
    roughSubIdeal?(ps1,ps2) and roughSubIdeal?(ps2,ps1)

if (R has GcdDomain) and (VarSet has ConvertibleTo (Symbol))
then

  LPR ==> List Polynomial R
  LS ==> List Symbol

  if R has EuclideanDomain
    then
      exactQuo(r:R,s:R):R ==
        r quo$R s
    else
      exactQuo(r:R,s:R):R ==
        (r exquo$R s)::R

  headRemainder (a,ps) ==
    lp1 : List(P) := remove(zero?, elements(ps))$(List(P))
    empty? lp1 => [a,1$R]
    any?(ground?,lp1) => [reductum(a),1$R]
    r : R := 1$R
    lp1 := sort(localInf?, reverse elements(ps))
    lp2 := lp1
    e : Union(E, "failed")
    while (not zero? a) and (not empty? lp2) repeat
      p := first lp2
      if ((e:= subtractIfCan(degree(a),degree(p))) case E)
        then
          g := gcd((lca := leadingCoefficient(a)),_
                   (lcp := leadingCoefficient(p)))$R
          (lca,lcp) := (exactQuo(lca,g),exactQuo(lcp,g))
          a := lcp * reductum(a) - monomial(lca, e::E)$P * reductum(p)
          r := r * lcp
          lp2 := lp1
        else
          lp2 := rest lp2
    [a,r]

  makeIrreducible! (frac:Record(num:P,den:R)):Record(num:P,den:R) ==
    g := gcd(frac.den,frac.num)$P
```

```
--         one? g => frac
          (g = 1) => frac
          frac.num := exactQuotient!(frac.num,g)
          frac.den := exactQuo(frac.den,g)
          frac

        remainder (a,ps) ==
          hRa := makeIrreducible! headRemainder (a,ps)
          a := hRa.num
          r : R := hRa.den
          zero? a => [1$R,a,r]
          b : P := monomial(1$R,degree(a))$P
          c : R := leadingCoefficient(a)
          while not zero?(a := reductum a) repeat
            hRa := makeIrreducible!  headRemainder (a,ps)
            a := hRa.num
            r := r * hRa.den
            g := gcd(c,(lca := leadingCoefficient(a)))$R
            b := ((hRa.den) * exactQuo(c,g)) * b + _
                  monomial(exactQuo(lca,g),degree(a))$P
            c := g
          [c,b,r]

        rewriteIdealWithHeadRemainder(ps,cs) ==
          trivialIdeal? cs => ps
          roughUnitIdeal? cs => [0$P]
          ps := remove(zero?,ps)
          empty? ps => ps
          any?(ground?,ps) => [1$P]
          rs : List P := []
          while not empty? ps repeat
            p := first ps
            ps := rest ps
            p := (headRemainder(p,cs)).num
            if not zero? p
              then
                if ground? p
                  then
                    ps := []
                    rs := [1$P]
                  else
                    primitivePart! p
                    rs := cons(p,rs)
          removeDuplicates rs

        rewriteIdealWithRemainder(ps,cs) ==
          trivialIdeal? cs => ps
          roughUnitIdeal? cs => [0$P]
          ps := remove(zero?,ps)
          empty? ps => ps
```

```
        any?(ground?,ps) => [1$P]
        rs : List P := []
        while not empty? ps repeat
          p := first ps
          ps := rest ps
          p := (remainder(p,cs)).polnum
          if not zero? p
            then
              if ground? p
                then
                  ps := []
                  rs := [1$P]
                else
                  rs := cons(unitCanonical(p),rs)
        removeDuplicates rs
```

———-

— **PSETCAT.dotabb** —

```
"PSETCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PSETCAT"];
"PSETCAT" -> "KOERCE"
"PSETCAT" -> "CLAGG"
"PSETCAT" -> "SETCAT"
```

———-

— **PSETCAT.dotfull** —

```
"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCategory(a,b,c))"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PSETCAT"];
"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCategory(a,b,c))"
 -> "SetCategory()"
"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCategory(a,b,c))"
 -> "Collection(RecursivePolynomialCategory(Ring,OrderedAbelianMonoidSup(),OrderedSet())"
"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCategory(a,b,c))"
 -> "CoercibleTo(List(RecursivePolynomialCategory(Ring,OrderedAbelianMonoidSup(),OrderedSet()))"

"PolynomialSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCategory
 [color=seagreen,href="bookvol10.2.pdf#nameddest=PSETCAT"];
"PolynomialSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCategory
  -> "PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCategory(a,b
```

———-

— PSETCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCatego
 [color=lightblue];
"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCatego
 -> "SetCategory()"
"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCatego
 -> "Collection(RecursivePolynomialCategory(Ring,OrderedAbelianMonoidSup(),OrderedSet())"
"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCatego
 -> "CoercibleTo(List(RecursivePolynomialCategory(Ring,OrderedAbelianMonoidSup(),OrderedSet()))"

"CoercibleTo(List(RecursivePolynomialCategory(Ring,OrderedAbelianMonoidSup(),OrderedSet()))"
 [color=seagreen];
"CoercibleTo(List(RecursivePolynomialCategory(Ring,OrderedAbelianMonoidSup(),OrderedSet()))"
  -> "CoercibleTo(a:Type)"

"Collection(RecursivePolynomialCategory(Ring,OrderedAbelianMonoidSup(),OrderedSet())" [color=sea
"Collection(RecursivePolynomialCategory(Ring,OrderedAbelianMonoidSup(),OrderedSet())"
  -> "Collection(a:Type)"

"Collection(a:Type)" [color=lightblue];
"Collection(a:Type)" -> "HomogeneousAggregate(a:Type)"
"Collection(a:Type)" -> "ConvertibleTo(InputForm)"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "Aggregate()"

"Aggregate()" [color=lightblue];
"Aggregate()" -> "Type()"

"Type()" [color=lightblue];
```

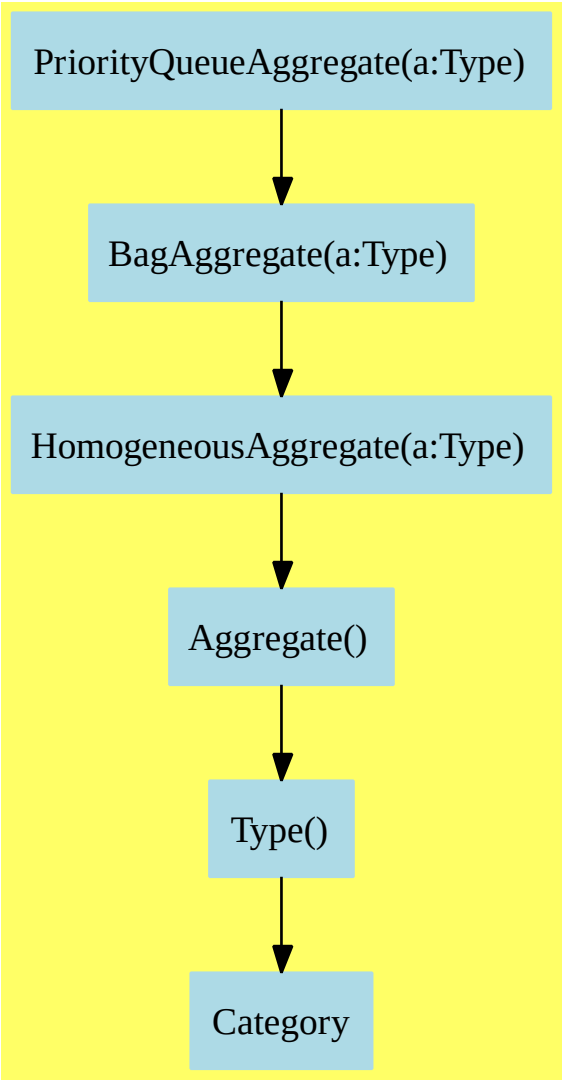
```
"Type()" -> "Category"

"ConvertibleTo(InputForm)" [color="#00EE00"];
"ConvertibleTo(InputForm)" -> "ConvertibleTo(a:Type)"

"ConvertibleTo(a:Type)" [color="#00EE00"];
"ConvertibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];
}
```

6.0.92 PriorityQueueAggregate (PRQAGG)



— PriorityQueueAggregate.input —

```
)set break resume
)sys rm -f PriorityQueueAggregate.output
)spool PriorityQueueAggregate.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 1
)show PriorityQueueAggregate
--R PriorityQueueAggregate S: OrderedSet  is a category constructor
--R Abbreviation for PriorityQueueAggregate is PRQAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PRQAGG
--R
--R------------------------------ Operations --------------------------------
--R bag : List S -> %                    copy : % -> %
--R empty : () -> %                      empty? : % -> Boolean
--R eq? : (%,%) -> Boolean               extract! : % -> S
--R insert! : (S,%) -> %                 inspect : % -> S
--R map : ((S -> S),%) -> %              max : % -> S
--R merge : (%,%) -> %                   merge! : (%,%) -> %
--R sample : () -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R coerce : % -> OutputForm if S has SETCAT
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R hash : % -> SingleInteger if S has SETCAT
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List S if $ has finiteAggregate
--R size? : (%,NonNegativeInteger) -> Boolean
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— PriorityQueueAggregate.help —

```
=====================================================================
PriorityQueueAggregate examples
```

```
================================================================

See Also:
o )show PriorityQueueAggregate
```

See:

⇒ "OrderedMultisetAggregate" (OMSAGG) 9.0.128 on page 763
⇐ "BagAggregate" (BGAGG) 5.0.68 on page 277

Exports:

any?	bag	copy	coerce	count
empty	empty?	eq?	eval	every?
extract!	hash	insert!	inspect	latex
less?	map	map!	max	member?
members	merge	merge!	more?	parts
sample	size?	#	?=?	?~=?

Attributes Exported:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.

These are directly exported but not implemented:

```
max : % -> S
merge : (%,%) -> %
merge! : (%,%) -> %
```

These exports come from (p277) BagAggregate(S:OrderedSet):

```
any? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
bag : List S -> %
coerce : % -> OutputForm if S has SETCAT
copy : % -> %
count : (S,%) -> NonNegativeInteger
         if S has SETCAT and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
empty : () -> %
empty? : % -> Boolean
```

```
 eq? : (%,%) -> Boolean
 eval : (%,List S,List S) -> %
          if S has EVALAB S and S has SETCAT
 eval : (%,S,S) -> %
          if S has EVALAB S and S has SETCAT
 eval : (%,Equation S) -> %
          if S has EVALAB S and S has SETCAT
 eval : (%,List Equation S) -> %
          if S has EVALAB S and S has SETCAT
 every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
 extract! : % -> S
 hash : % -> SingleInteger if S has SETCAT
 insert! : (S,%) -> %
 inspect : % -> S
 latex : % -> String if S has SETCAT
 less? : (%,NonNegativeInteger) -> Boolean
 map : ((S -> S),%) -> %
 map! : ((S -> S),%) -> % if $ has shallowlyMutable
 member? : (S,%) -> Boolean
          if S has SETCAT and $ has finiteAggregate
 members : % -> List S if $ has finiteAggregate
 more? : (%,NonNegativeInteger) -> Boolean
 parts : % -> List S if $ has finiteAggregate
 sample : () -> %
 size? : (%,NonNegativeInteger) -> Boolean
 #? : % -> NonNegativeInteger if $ has finiteAggregate
 ?=? : (%,%) -> Boolean if S has SETCAT
 ?~=? : (%,%) -> Boolean if S has SETCAT
```

— category PRQAGG PriorityQueueAggregate —

```
)abbrev category PRQAGG PriorityQueueAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A priority queue is a bag of items from an ordered set where the item
++ extracted is always the maximum element.

PriorityQueueAggregate(S:OrderedSet): Category == BagAggregate S with
   finiteAggregate
   max: % -> S
     ++ max(q) returns the maximum element of priority queue q.
```

```
   merge: (%,%) -> %
     ++ merge(q1,q2) returns combines priority queues q1 and q2 to return
     ++ a single priority queue q.
   merge_!: (%,%) -> %
     ++ merge!(q,q1) destructively changes priority queue q to include the
     ++ values from priority queue q1.
```

———-

— **PRQAGG.dotabb** —

```
"PRQAGG"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PRQAGG"];
"PRQAGG" -> "BGAGG"
```

———-

— **PRQAGG.dotfull** —

```
"PriorityQueueAggregate(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PRQAGG"];
"PriorityQueueAggregate(a:Type)" -> "BagAggregate(a:Type)"

"PriorityQueueAggregate(a:SetCategory)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=PRQAGG"];
"PriorityQueueAggregate(a:SetCategory)" -> "PriorityQueueAggregate(a:Type)"

"PriorityQueueAggregate(a:OrderedSet)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=PRQAGG"];
"PriorityQueueAggregate(a:OrderedSet)" ->
   "PriorityQueueAggregate(a:SetCategory)"
```

———-

— **PRQAGG.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"PriorityQueueAggregate(a:Type)" [color=lightblue];
"PriorityQueueAggregate(a:Type)" -> "BagAggregate(a:Type)"

"BagAggregate(a:Type)" [color=lightblue];
```

```
"BagAggregate(a:Type)" -> "HomogeneousAggregate(a:Type)"

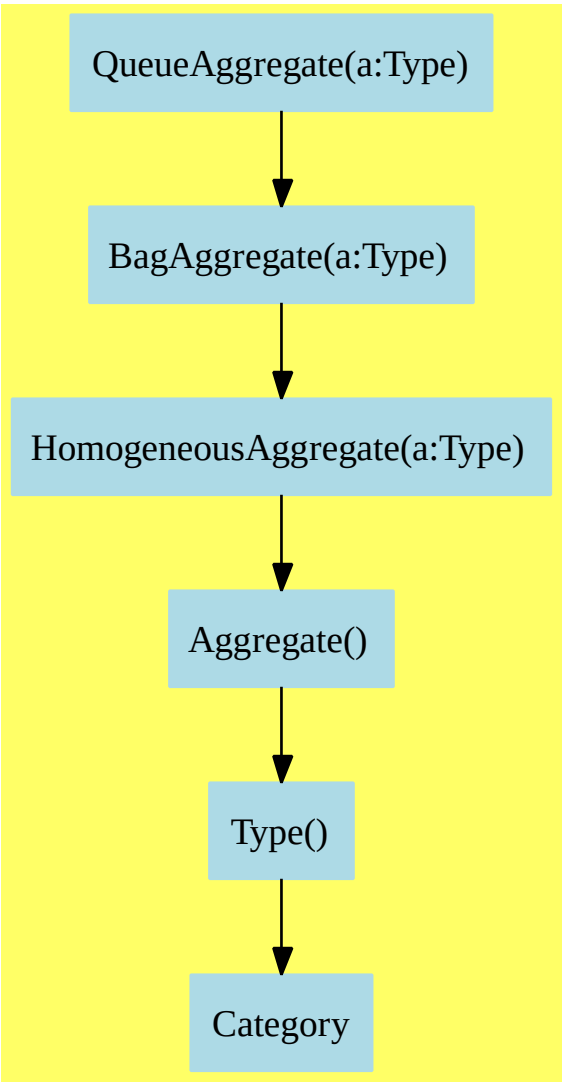
"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "Aggregate()"

"Aggregate()" [color=lightblue];
"Aggregate()" -> "Type()"

"Type()" [color=lightblue];
"Type()" -> "Category"

"Category" [color=lightblue];
}
```

6.0.93 QueueAggregate (QUAGG)



— **QueueAggregate.input** —

```
)set break resume
)sys rm -f QueueAggregate.output
)spool QueueAggregate.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 1
)show QueueAggregate
--R QueueAggregate S: Type  is a category constructor
--R Abbreviation for QueueAggregate is QUAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for QUAGG
--R
--R------------------------------ Operations --------------------------------
--R back : % -> S                        bag : List S -> %
--R copy : % -> %                        dequeue! : % -> S
--R empty : () -> %                      empty? : % -> Boolean
--R enqueue! : (S,%) -> S                eq? : (%,%) -> Boolean
--R extract! : % -> S                    front : % -> S
--R insert! : (S,%) -> %                 inspect : % -> S
--R length : % -> NonNegativeInteger     map : ((S -> S),%) -> %
--R rotate! : % -> %                     sample : () -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R coerce : % -> OutputForm if S has SETCAT
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R hash : % -> SingleInteger if S has SETCAT
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List S if $ has finiteAggregate
--R size? : (%,NonNegativeInteger) -> Boolean
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— QueueAggregate.help —

```
=====================================================================
```

```
QueueAggregate examples
=======================================================================

See Also:
o )show QueueAggregate
```

———-

See:

⇒ “DequeueAggregate” (DQAGG) 7.0.100 on page 540
⇐ “BagAggregate” (BGAGG) 5.0.68 on page 277

Exports:

any?	bag	back	coerce	copy
count	dequeue!	empty	empty?	enqueue!
eq?	eval	every?	extract!	front
hash	insert!	inspect	latex	length
less?	map	map!	member?	members
more?	parts	rotate!	sample	size?
#?	?=?	?~=?		

Attributes Exported:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.

These are implemented by this category:

```
back : % -> S
dequeue! : % -> S
enqueue! : (S,%) -> S
front : % -> S
length : % -> NonNegativeInteger
rotate! : % -> %
```

These exports come from (p277) BagAggregate(S:Type):

```
any? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
bag : List S -> %
coerce : % -> OutputForm if S has SETCAT
copy : % -> %
count : (S,%) -> NonNegativeInteger
```

```
          if S has SETCAT and $ has finiteAggregate
 count : ((S -> Boolean),%) -> NonNegativeInteger
          if $ has finiteAggregate
 empty : () -> %
 empty? : % -> Boolean
 eq? : (%,%) -> Boolean
 eval : (%,List S,List S) -> %
          if S has EVALAB S and S has SETCAT
 eval : (%,S,S) -> %
          if S has EVALAB S and S has SETCAT
 eval : (%,Equation S) -> %
          if S has EVALAB S and S has SETCAT
 eval : (%,List Equation S) -> %
          if S has EVALAB S and S has SETCAT
 every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
 extract! : % -> S
 hash : % -> SingleInteger if S has SETCAT
 insert! : (S,%) -> %
 inspect : % -> S
 latex : % -> String if S has SETCAT
 less? : (%,NonNegativeInteger) -> Boolean
 map : ((S -> S),%) -> %
 map! : ((S -> S),%) -> % if $ has shallowlyMutable
 member? : (S,%) -> Boolean
          if S has SETCAT and $ has finiteAggregate
 members : % -> List S if $ has finiteAggregate
 more? : (%,NonNegativeInteger) -> Boolean
 parts : % -> List S if $ has finiteAggregate
 sample : () -> %
 size? : (%,NonNegativeInteger) -> Boolean
 #? : % -> NonNegativeInteger if $ has finiteAggregate
 ?=? : (%,%) -> Boolean if S has SETCAT
 ?~=? : (%,%) -> Boolean if S has SETCAT
```

— category QUAGG QueueAggregate —

```
)abbrev category QUAGG QueueAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A queue is a bag where the first item inserted is the first
++ item extracted.
```

```
QueueAggregate(S:Type): Category == BagAggregate S with
   finiteAggregate
   enqueue_!: (S, %) -> S
     ++ enqueue!(x,q) inserts x into the queue q at the back end.
   dequeue_!: % -> S
     ++ dequeue! s destructively extracts the first (top) element
     ++ from queue q. The element previously second in the queue becomes
     ++ the first element. Error: if q is empty.
   rotate_!: % -> %
     ++ rotate! q rotates queue q so that the element at the front of
     ++ the queue goes to the back of the queue.
     ++ Note that rotate! q is equivalent to enqueue!(dequeue!(q)).
   length: % -> NonNegativeInteger
     ++ length(q) returns the number of elements in the queue.
     ++ Note that \axiom{length(q) = #q}.
   front: % -> S
     ++ front(q) returns the element at the front of the queue.
     ++ The queue q is unchanged by this operation.
     ++ Error: if q is empty.
   back: % -> S
     ++ back(q) returns the element at the back of the queue.
     ++ The queue q is unchanged by this operation.
     ++ Error: if q is empty.
```

— QUAGG.dotabb —

```
"QUAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=QUAGG"];
"QUAGG" -> "BGAGG"
```

— QUAGG.dotfull —

```
"QueueAggregate(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=QUAGG"];
"QueueAggregate(a:Type)" -> "BagAggregate(a:Type)"

"QueueAggregate(a:SetCategory)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=QUAGG"];
"QueueAggregate(a:SetCategory)" -> "QueueAggregate(a:Type)"
```

— QUAGG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"QueueAggregate(a:Type)" [color=lightblue];
"QueueAggregate(a:Type)" -> "BagAggregate(a:Type)"

"BagAggregate(a:Type)" [color=lightblue];
"BagAggregate(a:Type)" -> "HomogeneousAggregate(a:Type)"

"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "Aggregate()"

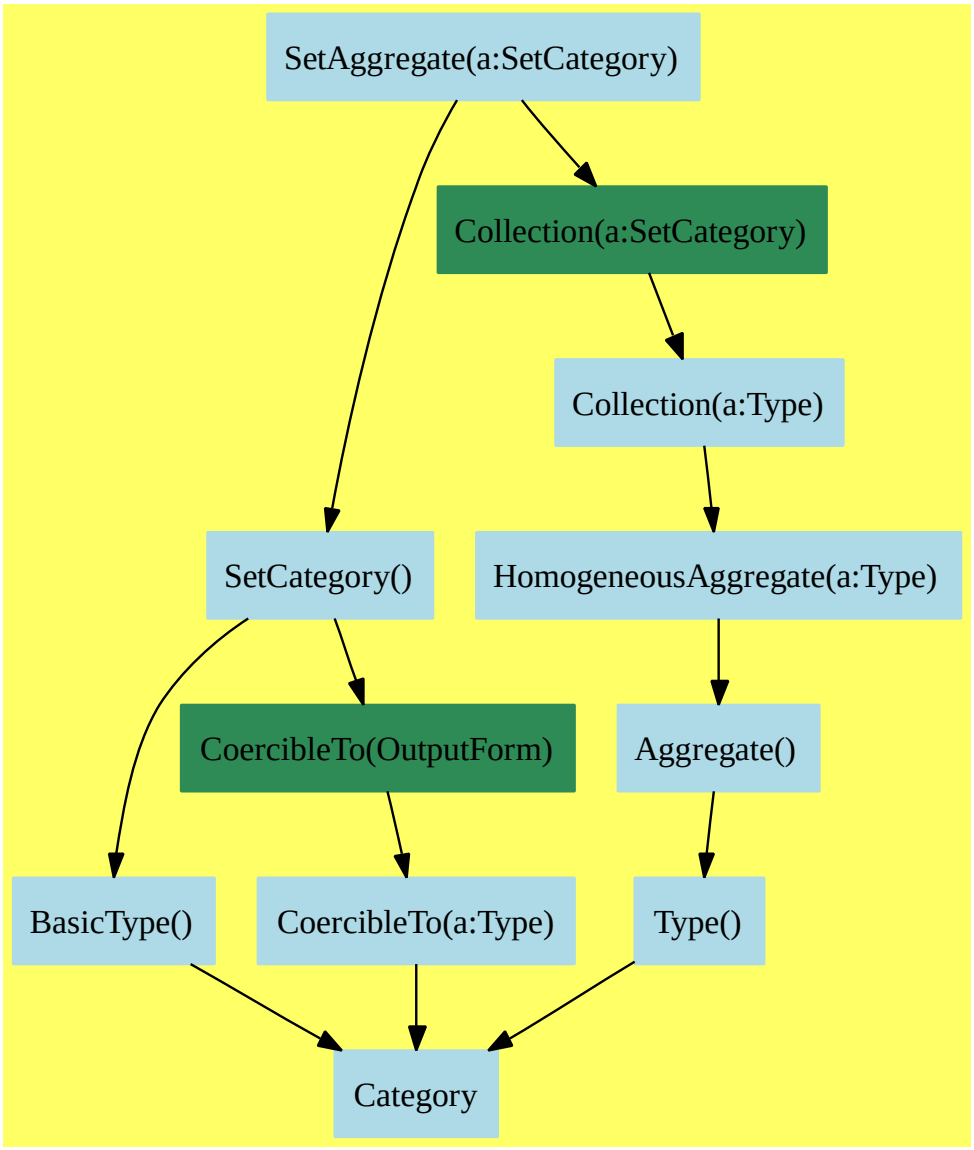
"Aggregate()" [color=lightblue];
"Aggregate()" -> "Type()"

"Type()" [color=lightblue];
"Type()" -> "Category"

"Category" [color=lightblue];
}
```

———-

6.0.94 SetAggregate (SETAGG)



— SetAggregate.input —

```
)set break resume
)sys rm -f SetAggregate.output
)spool SetAggregate.output
)set message test on
```

```
)set message auto off
)clear all

--S 1 of 1
)show SetAggregate
--R SetAggregate S: SetCategory  is a category constructor
--R Abbreviation for SetAggregate is SETAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for SETAGG
--R
--R------------------------------ Operations --------------------------------
--R ?<? : (%,%) -> Boolean               ?=? : (%,%) -> Boolean
--R brace : List S -> %                  brace : () -> %
--R coerce : % -> OutputForm             construct : List S -> %
--R copy : % -> %                        difference : (%,S) -> %
--R difference : (%,%) -> %              empty : () -> %
--R empty? : % -> Boolean                eq? : (%,%) -> Boolean
--R hash : % -> SingleInteger            intersect : (%,%) -> %
--R latex : % -> String                  map : ((S -> S),%) -> %
--R sample : () -> %                     set : List S -> %
--R set : () -> %                        subset? : (%,%) -> Boolean
--R union : (S,%) -> %                   union : (%,S) -> %
--R union : (%,%) -> %                   ?~=? : (%,%) -> Boolean
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R convert : % -> InputForm if S has KONVERT INFORM
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R find : ((S -> Boolean),%) -> Union(S,"failed")
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S,S) -> S if S has SETCAT and $ has finiteAggregate
--R remove : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (S,%) -> % if S has SETCAT and $ has finiteAggregate
--R removeDuplicates : % -> % if S has SETCAT and $ has finiteAggregate
--R select : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R size? : (%,NonNegativeInteger) -> Boolean
--R symmetricDifference : (%,%) -> %
--R
```

```
--E 1

)spool
)lisp (bye)
```

———-

— **SetAggregate.help** —

```
====================================================================
SetAggregate examples
====================================================================

See Also:
o )show SetAggregate
```

———-

See:

⇒ “FiniteSetAggregate” (FSAGG) 8.0.110 on page 621
⇒ “MultisetAggregate” (MSETAGG) 8.0.115 on page 673
⇐ “Collection” (CLAGG) 5.0.70 on page 287
⇐ “SetCategory” (SETCAT) 3.0.34 on page 113

Exports:

any?	brace	coerce	construct	convert
copy	count	difference	empty	empty?
eq?	eval	every?	find	hash
intersect	latex	less?	map	map!
member?	members	more?	parts	reduce
remove	removeDuplicates	sample	select	set
size?	subset?	symmetricDifference	union	#?
?<?	?=?	?~=?		

Attributes Exported:

- **partiallyOrderedSet** is true if a set with $<$ which is transitive, but not($a < b$ or $a = b$) does not necessarily imply $b < a$.
- **nil**

These are directly exported but not implemented:

```
brace : List S -> %
brace : () -> %
```

```
difference : (%,%) -> %
intersect : (%,%) -> %
set : List S -> %
set : () -> %
subset? : (%,%) -> Boolean
union : (%,%) -> %
?<? : (%,%) -> Boolean
```

These are implemented by this category:

```
difference : (%,S) -> %
symmetricDifference : (%,%) -> %
union : (S,%) -> %
union : (%,S) -> %
```

These exports come from (p113) SetCategory():

```
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
```

These exports come from (p287) Collection(S:SetCategory):

```
any? : ((S -> Boolean),%) -> Boolean
    if $ has finiteAggregate
construct : List S -> %
convert : % -> InputForm if S has KONVERT INFORM
copy : % -> %
count : ((S -> Boolean),%) -> NonNegativeInteger
    if $ has finiteAggregate
count : (S,%) -> NonNegativeInteger
    if S has SETCAT
    and $ has finiteAggregate
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List Equation S) -> %
    if S has EVALAB S
    and S has SETCAT
eval : (%,Equation S) -> %
    if S has EVALAB S
    and S has SETCAT
eval : (%,S,S) -> %
    if S has EVALAB S
    and S has SETCAT
eval : (%,List S,List S) -> %
    if S has EVALAB S
```

```
    and S has SETCAT
 every? : ((S -> Boolean),%) -> Boolean
    if $ has finiteAggregate
 find : ((S -> Boolean),%) -> Union(S,"failed")
 less? : (%,NonNegativeInteger) -> Boolean
 map : ((S -> S),%) -> %
 map! : ((S -> S),%) -> % if $ has shallowlyMutable
 members : % -> List S if $ has finiteAggregate
 member? : (S,%) -> Boolean
    if S has SETCAT and $ has finiteAggregate
 more? : (%,NonNegativeInteger) -> Boolean
 parts : % -> List S if $ has finiteAggregate
 reduce : (((S,S) -> S),%,S,S) -> S
    if S has SETCAT
    and $ has finiteAggregate
 reduce : (((S,S) -> S),%,S) -> S
    if $ has finiteAggregate
 reduce : (((S,S) -> S),%) -> S
    if $ has finiteAggregate
 remove : ((S -> Boolean),%) -> %
    if $ has finiteAggregate
 remove : (S,%) -> %
    if S has SETCAT
    and $ has finiteAggregate
 removeDuplicates : % -> %
    if S has SETCAT
    and $ has finiteAggregate
 sample : () -> %
 select : ((S -> Boolean),%) -> %
    if $ has finiteAggregate
 size? : (%,NonNegativeInteger) -> Boolean
 #? : % -> NonNegativeInteger
    if $ has finiteAggregate
```

— category SETAGG SetAggregate —

```
)abbrev category SETAGG SetAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: 14 Oct, 1993 by RSS
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A set category lists a collection of set-theoretic operations
++ useful for both finite sets and multisets.
```

```
++ Note however that finite sets are distinct from multisets.
++ Although the operations defined for set categories are
++ common to both, the relationship between the two cannot
++ be described by inclusion or inheritance.

SetAggregate(S:SetCategory):
  Category == Join(SetCategory, Collection(S)) with
   partiallyOrderedSet
   "<"         : (%, %) -> Boolean
     ++ s < t returns true if all elements of set aggregate s are also
     ++ elements of set aggregate t.
   brace       : () -> %
     ++ brace()$D (otherwise written {}$D)
     ++ creates an empty set aggregate of type D.
     ++ This form is considered obsolete. Use \axiomFun{set} instead.
   brace       : List S -> %
     ++ brace([x,y,...,z])
     ++ creates a set aggregate containing items x,y,...,z.
     ++ This form is considered obsolete. Use \axiomFun{set} instead.
   set               : () -> %
     ++ set()$D creates an empty set aggregate of type D.
   set               : List S -> %
     ++ set([x,y,...,z]) creates a set aggregate containing items x,y,...,z.
   intersect: (%, %) -> %
     ++ intersect(u,v) returns the set aggregate w consisting of
     ++ elements common to both set aggregates u and v.
     ++ Note that equivalent to the notation (not currently supported)
     ++ {x for x in u | member?(x,v)}.
   difference  : (%, %) -> %
     ++ difference(u,v) returns the set aggregate w consisting of
     ++ elements in set aggregate u but not in set aggregate v.
     ++ If u and v have no elements in common, \axiom{difference(u,v)}
     ++ returns a copy of u.
     ++ Note that equivalent to the notation (not currently supported)
     ++ \axiom{{x for x in u | not member?(x,v)}}.
   difference  : (%, S) -> %
     ++ difference(u,x) returns the set aggregate u with element x removed.
     ++ If u does not contain x, a copy of u is returned.
     ++ Note that \axiom{difference(s, x) = difference(s, {x})}.
   symmetricDifference : (%, %) -> %
     ++ symmetricDifference(u,v) returns the set aggregate of elements x
     ++ which are members of set aggregate u or set aggregate v but
     ++ not both. If u and v have no elements in common,
     ++ \axiom{symmetricDifference(u,v)} returns a copy of u.
     ++ Note that \axiom{symmetricDifference(u,v) =
     ++  union(difference(u,v),difference(v,u))}
   subset?     : (%, %) -> Boolean
     ++ subset?(u,v) tests if u is a subset of v.
     ++ Note that equivalent to
     ++ \axiom{reduce(and,{member?(x,v) for x in u},true,false)}.
```

```
   union       : (%, %) -> %
     ++ union(u,v) returns the set aggregate of elements which are members
     ++ of either set aggregate u or v.
   union       : (%, S) -> %
     ++ union(u,x) returns the set aggregate u with the element x added.
     ++ If u already contains x, \axiom{union(u,x)} returns a copy of u.
   union       : (S, %) -> %
     ++ union(x,u) returns the set aggregate u with the element x added.
     ++ If u already contains x, \axiom{union(x,u)} returns a copy of u.
 add
  symmetricDifference(x, y)    == union(difference(x, y), difference(y, x))
  union(s:%, x:S) == union(s, {x})
  union(x:S, s:%) == union(s, {x})
  difference(s:%, x:S) == difference(s, {x})
```

———-

— SETAGG.dotabb —

```
"SETAGG"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SETAGG"];
"SETAGG" -> "SETCAT"
"SETAGG" -> "CLAGG"
```

———-

— SETAGG.dotfull —

```
"SetAggregate(a:SetCategory)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SETAGG"];
"SetAggregate(a:SetCategory)" -> "SetCategory()"
"SetAggregate(a:SetCategory)" -> "Collection(a:SetCategory)"
```

———-

— SETAGG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"SetAggregate(a:SetCategory)" [color=lightblue];
"SetAggregate(a:SetCategory)" -> "SetCategory()"
"SetAggregate(a:SetCategory)" -> "Collection(a:SetCategory)"
```

```
"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Collection(a:SetCategory)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=CLAGG"];
"Collection(a:SetCategory)" -> "Collection(a:Type)"

"Collection(a:Type)" [color=lightblue];
"Collection(a:Type)" -> "HomogeneousAggregate(a:Type)"

"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "Aggregate()"

"Aggregate()" [color=lightblue];
"Aggregate()" -> "Type()"

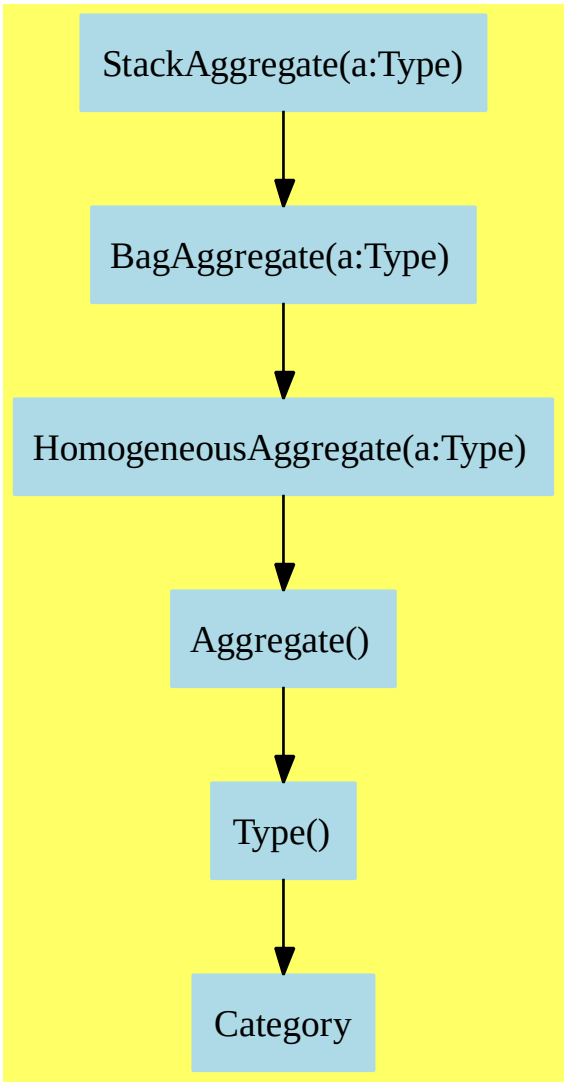
"Type()" [color=lightblue];
"Type()" -> "Category"

"Category" [color=lightblue];

}
```

———-

6.0.95 StackAggregate (SKAGG)



— **StackAggregate.input** —

```
)set break resume
)sys rm -f StackAggregate.output
)spool StackAggregate.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 1
)show StackAggregate
--R StackAggregate S: Type  is a category constructor
--R Abbreviation for StackAggregate is SKAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for SKAGG
--R
--R------------------------------ Operations --------------------------------
--R bag : List S -> %                    copy : % -> %
--R depth : % -> NonNegativeInteger      empty : () -> %
--R empty? : % -> Boolean                eq? : (%,%) -> Boolean
--R extract! : % -> S                    insert! : (S,%) -> %
--R inspect : % -> S                     map : ((S -> S),%) -> %
--R pop! : % -> S                       push! : (S,%) -> S
--R sample : () -> %                     top : % -> S
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R coerce : % -> OutputForm if S has SETCAT
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R hash : % -> SingleInteger if S has SETCAT
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List S if $ has finiteAggregate
--R size? : (%,NonNegativeInteger) -> Boolean
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— StackAggregate.help —

```
=====================================================================
StackAggregate examples
```

```
================================================================
```

```
See Also:
o )show StackAggregate
```

———-

See:

⇒ “DequeueAggregate” (DQAGG) 7.0.100 on page 540
⇐ “BagAggregate” (BGAGG) 5.0.68 on page 277

Exports:

any?	bag	coerce	copy	count
depth	empty	empty?	eq?	eval
every?	extract!	hash	insert!	inspect
latex	less?	map	map!	member?
members	more?	parts	pop!	push!
sample	size?	top	#?	?=?
?~=?				

Attributes Exported:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.

These are directly exported but not implemented:

```
depth : % -> NonNegativeInteger
pop! : % -> S
push! : (S,%) -> S
top : % -> S
```

These exports come from (p277) BagAggregate(S:Type):

```
any? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
bag : List S -> %
coerce : % -> OutputForm if S has SETCAT
copy : % -> %
count : (S,%) -> NonNegativeInteger
         if S has SETCAT and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
```

```
 empty : () -> %
 empty? : % -> Boolean
 eq? : (%,%) -> Boolean
 eval : (%,List S,List S) -> %
          if S has EVALAB S and S has SETCAT
 eval : (%,S,S) -> %
          if S has EVALAB S and S has SETCAT
 eval : (%,Equation S) -> %
          if S has EVALAB S and S has SETCAT
 eval : (%,List Equation S) -> %
          if S has EVALAB S and S has SETCAT
 every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
 extract! : % -> S
 hash : % -> SingleInteger if S has SETCAT
 insert! : (S,%) -> %
 inspect : % -> S
 latex : % -> String if S has SETCAT
 less? : (%,NonNegativeInteger) -> Boolean
 map : ((S -> S),%) -> %
 map! : ((S -> S),%) -> % if $ has shallowlyMutable
 member? : (S,%) -> Boolean
          if S has SETCAT and $ has finiteAggregate
 members : % -> List S if $ has finiteAggregate
 more? : (%,NonNegativeInteger) -> Boolean
 parts : % -> List S if $ has finiteAggregate
 sample : () -> %
 size? : (%,NonNegativeInteger) -> Boolean
 #? : % -> NonNegativeInteger if $ has finiteAggregate
 ?=? : (%,%) -> Boolean if S has SETCAT
 ?~=? : (%,%) -> Boolean if S has SETCAT
```

— category SKAGG StackAggregate —

```
)abbrev category SKAGG StackAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A stack is a bag where the last item inserted is the first item extracted.

StackAggregate(S:Type): Category == BagAggregate S with
   finiteAggregate
   push_!: (S,%) -> S
```

```
    ++ push!(x,s) pushes x onto stack s, i.e. destructively changing s
    ++ so as to have a new first (top) element x.
    ++ Afterwards, pop!(s) produces x and pop!(s) produces the original s.
    ++
    ++X a:Stack INT:= stack [1,2,3,4,5]
    ++X push! a
    ++X a
  pop_!: % -> S
    ++ pop!(s) returns the top element x, destructively removing x from s.
    ++ Note that Use \axiom{top(s)} to obtain x without removing it from s.
    ++ Error: if s is empty.
    ++
    ++X a:Stack INT:= stack [1,2,3,4,5]
    ++X pop! a
    ++X a
  top: % -> S
    ++ top(s) returns the top element x from s; s remains unchanged.
    ++ Note that Use \axiom{pop!(s)} to obtain x and remove it from s.
    ++
    ++X a:Stack INT:= stack [1,2,3,4,5]
    ++X top a
  depth: % -> NonNegativeInteger
    ++ depth(s) returns the number of elements of stack s.
    ++ Note that \axiom{depth(s) = #s}.
    ++
    ++X a:Stack INT:= stack [1,2,3,4,5]
    ++X depth a
```

———-

— SKAGG.dotabb —

```
"SKAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=SKAGG"];
"SKAGG" -> "BGAGG"
```

———-

— SKAGG.dotfull —

```
"StackAggregate(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SKAGG"];
"StackAggregate(a:Type)" -> "BagAggregate(a:Type)"

"StackAggregate(a:SetCategory)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=SKAGG"];
```

```
"StackAggregate(a:SetCategory)" -> "StackAggregate(a:Type)"
```

———-

— SKAGG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"StackAggregate(a:Type)" [color=lightblue];
"StackAggregate(a:Type)" -> "BagAggregate(a:Type)"

"BagAggregate(a:Type)" [color=lightblue];
"BagAggregate(a:Type)" -> "HomogeneousAggregate(a:Type)"

"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "Aggregate()"

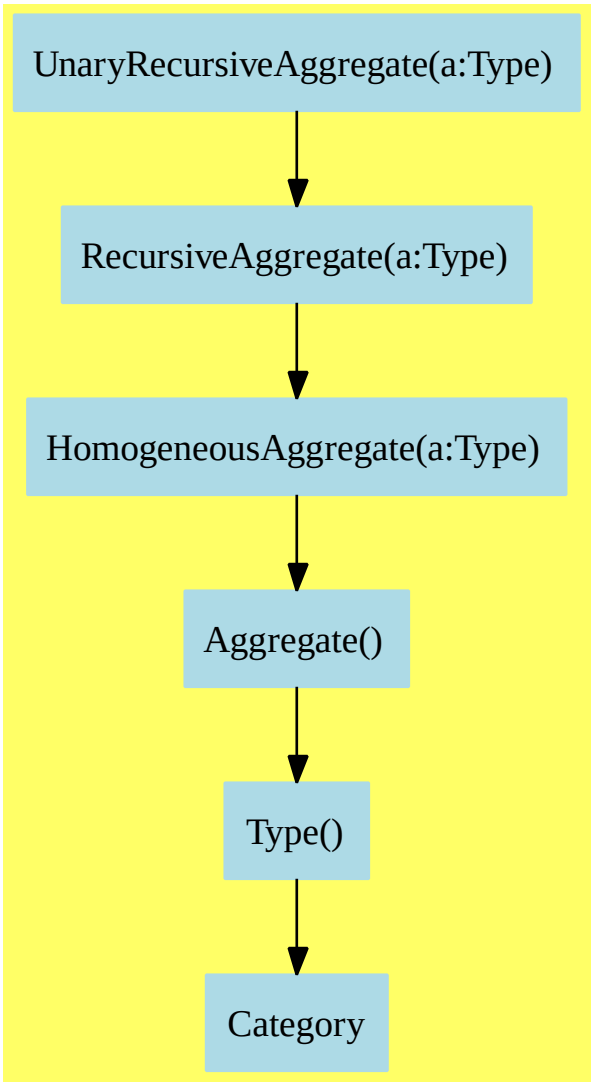
"Aggregate()" [color=lightblue];
"Aggregate()" -> "Type()"

"Type()" [color=lightblue];
"Type()" -> "Category"

"Category" [color=lightblue];
}
```

———-

6.0.96 UnaryRecursiveAggregate (URAGG)



— **UnaryRecursiveAggregate.input** —

```
)set break resume
)sys rm -f UnaryRecursiveAggregate.output
)spool UnaryRecursiveAggregate.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 1
)show UnaryRecursiveAggregate
--R UnaryRecursiveAggregate S: Type  is a category constructor
--R Abbreviation for UnaryRecursiveAggregate is URAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for URAGG
--R
--R------------------------------ Operations --------------------------------
--R children : % -> List %               concat : (S,%) -> %
--R concat : (%,%) -> %                  copy : % -> %
--R cycleEntry : % -> %                  cycleTail : % -> %
--R cyclic? : % -> Boolean               distance : (%,%) -> Integer
--R ?.last : (%,last) -> S               ?.rest : (%,rest) -> %
--R ?.first : (%,first) -> S             ?.value : (%,value) -> S
--R empty : () -> %                      empty? : % -> Boolean
--R eq? : (%,%) -> Boolean               first : % -> S
--R last : % -> S                        leaf? : % -> Boolean
--R leaves : % -> List S                 map : ((S -> S),%) -> %
--R nodes : % -> List %                  rest : % -> %
--R sample : () -> %                     second : % -> S
--R tail : % -> %                        third : % -> S
--R value : % -> S
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R child? : (%,%) -> Boolean if S has SETCAT
--R coerce : % -> OutputForm if S has SETCAT
--R concat! : (%,S) -> % if $ has shallowlyMutable
--R concat! : (%,%) -> % if $ has shallowlyMutable
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R cycleLength : % -> NonNegativeInteger
--R cycleSplit! : % -> % if $ has shallowlyMutable
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R first : (%,NonNegativeInteger) -> %
--R hash : % -> SingleInteger if S has SETCAT
--R last : (%,NonNegativeInteger) -> %
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R node? : (%,%) -> Boolean if S has SETCAT
--R parts : % -> List S if $ has finiteAggregate
```

```
--R rest : (%,NonNegativeInteger) -> %
--R setchildren! : (%,List %) -> % if $ has shallowlyMutable
--R setelt : (%,last,S) -> S if $ has shallowlyMutable
--R setelt : (%,rest,%) -> % if $ has shallowlyMutable
--R setelt : (%,first,S) -> S if $ has shallowlyMutable
--R setelt : (%,value,S) -> S if $ has shallowlyMutable
--R setfirst! : (%,S) -> S if $ has shallowlyMutable
--R setlast! : (%,S) -> S if $ has shallowlyMutable
--R setrest! : (%,%) -> % if $ has shallowlyMutable
--R setvalue! : (%,S) -> S if $ has shallowlyMutable
--R size? : (%,NonNegativeInteger) -> Boolean
--R split! : (%,Integer) -> % if $ has shallowlyMutable
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— **UnaryRecursiveAggregate.help** —

```
==========================================================================
UnaryRecursiveAggregate examples
==========================================================================

See Also:
o )show UnaryRecursiveAggregate
```

———-

See:

⇒ “StreamAggregate” (STAGG) 7.0.107 on page 585
⇐ “RecursiveAggregate” (RCAGG) 5.0.80 on page 351

Exports:

any?	children	child?	coerce	concat
concat!	copy	count	cycleEntry	cycleLength
cycleSplit!	cycleTail	cyclic?	distance	empty
empty?	eq?	eval	every?	first
hash	last	latex	leaf?	leaves
less?	map	map!	member?	members
more?	nodes	node?	parts	rest
sample	second	setchildren!	setelt	setfirst!
setlast!	setrest!	setvalue!	size?	split!
tail	third	value	#?	?.last
?.rest	?.first	?.value	?=?	?~=?

Attributes exported:

- **nil**

Attributes Used:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.

These are directly exported but not implemented:

```
concat : (S,%) -> %
concat! : (%,S) -> % if $ has shallowlyMutable
concat! : (%,%) -> % if $ has shallowlyMutable
first : % -> S
first : (%,NonNegativeInteger) -> %
rest : % -> %
setfirst! : (%,S) -> S if $ has shallowlyMutable
```

These are implemented by this category:

```
children : % -> List %
concat : (%,%) -> %
cycleEntry : % -> %
cycleLength : % -> NonNegativeInteger
cycleSplit! : % -> % if $ has shallowlyMutable
cycleTail : % -> %
cyclic? : % -> Boolean
last : % -> S
last : (%,NonNegativeInteger) -> %
leaf? : % -> Boolean
less? : (%,NonNegativeInteger) -> Boolean
more? : (%,NonNegativeInteger) -> Boolean
nodes : % -> List %
```

```
node? : (%,%) -> Boolean if S has SETCAT
rest : (%,NonNegativeInteger) -> %
second : % -> S
setchildren! : (%,List %) -> % if $ has shallowlyMutable
setelt : (%,first,S) -> S if $ has shallowlyMutable
setelt : (%,last,S) -> S if $ has shallowlyMutable
setelt : (%,rest,%) -> % if $ has shallowlyMutable
setlast! : (%,S) -> S if $ has shallowlyMutable
setvalue! : (%,S) -> S if $ has shallowlyMutable
size? : (%,NonNegativeInteger) -> Boolean
split! : (%,Integer) -> % if $ has shallowlyMutable
tail : % -> %
third : % -> S
value : % -> S
#? : % -> NonNegativeInteger if $ has finiteAggregate
?=? : (%,%) -> Boolean if S has SETCAT
?.first : (%,first) -> S
?.last : (%,last) -> S
?.rest : (%,rest) -> %
```

These exports come from (p351) RecursiveAggregate(S:Type):

```
any? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
child? : (%,%) -> Boolean if S has SETCAT
coerce : % -> OutputForm if S has SETCAT
copy : % -> %
count : (S,%) -> NonNegativeInteger
         if S has SETCAT and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
distance : (%,%) -> Integer
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,S,S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,Equation S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,List Equation S) -> %
         if S has EVALAB S and S has SETCAT
every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
hash : % -> SingleInteger if S has SETCAT
latex : % -> String if S has SETCAT
leaves : % -> List S
map : ((S -> S),%) -> %
map! : ((S -> S),%) -> % if $ has shallowlyMutable
member? : (S,%) -> Boolean
```

```
          if S has SETCAT and $ has finiteAggregate
 members : % -> List S if $ has finiteAggregate
 parts : % -> List S if $ has finiteAggregate
 sample : () -> %
 setelt : (%,value,S) -> S if $ has shallowlyMutable
 setrest! : (%,%) -> % if $ has shallowlyMutable
 ?.value : (%,value) -> S
 ?~=? : (%,%) -> Boolean if S has SETCAT
```

— category URAGG UnaryRecursiveAggregate —

```
)abbrev category URAGG UnaryRecursiveAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A unary-recursive aggregate is a one where nodes may have either
++ 0 or 1 children.
++ This aggregate models, though not precisely, a linked
++ list possibly with a single cycle.
++ A node with one children models a non-empty list, with the
++ \spadfun{value} of the list designating the head, or \spadfun{first},
++ of the list, and the child designating the tail, or \spadfun{rest},
++ of the list. A node with no child then designates the empty list.
++ Since these aggregates are recursive aggregates, they may be cyclic.

UnaryRecursiveAggregate(S:Type): Category == RecursiveAggregate S with
   concat: (%,%) -> %
      ++ concat(u,v) returns an aggregate w consisting of the elements of u
      ++ followed by the elements of v.
      ++ Note that \axiom{v = rest(w,#a)}.
   concat: (S,%) -> %
      ++ concat(x,u) returns aggregate consisting of x followed by
      ++ the elements of u.
      ++ Note that if \axiom{v = concat(x,u)} then \axiom{x = first v}
      ++ and \axiom{u = rest v}.
   first: % -> S
      ++ first(u) returns the first element of u
      ++ (equivalently, the value at the current node).
   elt: (%,"first") -> S
      ++ elt(u,"first") (also written: \axiom{u . first})
      ++ is equivalent to first u.
   first: (%,NonNegativeInteger) -> %
```

```
      ++ first(u,n) returns a copy of the first n (\axiom{n >= 0})
      ++ elements of u.
   rest: % -> %
      ++ rest(u) returns an aggregate consisting of all but the first
      ++ element of u
      ++ (equivalently, the next node of u).
   elt: (%,"rest") -> %
      ++ elt(%,"rest") (also written: \axiom{u.rest}) is
      ++ equivalent to \axiom{rest u}.
   rest: (%,NonNegativeInteger) -> %
      ++ rest(u,n) returns the \axiom{n}th (n >= 0) node of u.
      ++ Note that \axiom{rest(u,0) = u}.
   last: % -> S
      ++ last(u) resturn the last element of u.
      ++ Note that for lists, \axiom{last(u)=u . (maxIndex u)=u . (# u - 1)}.
   elt: (%,"last") -> S
      ++ elt(u,"last") (also written: \axiom{u . last}) is equivalent
      ++ to last u.
   last: (%,NonNegativeInteger) -> %
      ++ last(u,n) returns a copy of the last n (\axiom{n >= 0}) nodes of u.
      ++ Note that \axiom{last(u,n)} is a list of n elements.
   tail: % -> %
      ++ tail(u) returns the last node of u.
      ++ Note that if u is \axiom{shallowlyMutable},
      ++ \axiom{setrest(tail(u),v) = concat(u,v)}.
   second: % -> S
      ++ second(u) returns the second element of u.
      ++ Note that \axiom{second(u) = first(rest(u))}.
   third: % -> S
      ++ third(u) returns the third element of u.
      ++ Note that \axiom{third(u) = first(rest(rest(u)))}.
   cycleEntry: % -> %
      ++ cycleEntry(u) returns the head of a top-level cycle contained in
      ++ aggregate u, or \axiom{empty()} if none exists.
   cycleLength: % -> NonNegativeInteger
      ++ cycleLength(u) returns the length of a top-level cycle
      ++ contained  in aggregate u, or 0 is u has no such cycle.
   cycleTail: % -> %
      ++ cycleTail(u) returns the last node in the cycle, or
      ++ empty if none exists.
   if % has shallowlyMutable then
      concat_!: (%,%) -> %
        ++ concat!(u,v) destructively concatenates v to the end of u.
        ++ Note that \axiom{concat!(u,v) = setlast_!(u,v)}.
      concat_!: (%,S) -> %
        ++ concat!(u,x) destructively adds element x to the end of u.
        ++ Note that \axiom{concat!(a,x) = setlast!(a,[x])}.
      cycleSplit_!: % -> %
        ++ cycleSplit!(u) splits the aggregate by dropping off the cycle.
        ++ The value returned is the cycle entry, or nil if none exists.
```

```
      ++ For example, if \axiom{w = concat(u,v)} is the cyclic list where
      ++ v is the head of the cycle, \axiom{cycleSplit!(w)} will drop v
      ++ off w thus destructively changing w to u, and returning v.
    setfirst_!: (%,S) -> S
      ++ setfirst!(u,x) destructively changes the first element of a to x.
    setelt: (%,"first",S) -> S
      ++ setelt(u,"first",x) (also written: \axiom{u.first := x}) is
      ++ equivalent to \axiom{setfirst!(u,x)}.
    setrest_!: (%,%) -> %
      ++ setrest!(u,v) destructively changes the rest of u to v.
    setelt: (%,"rest",%) -> %
      ++ setelt(u,"rest",v) (also written: \axiom{u.rest := v}) is
      ++ equivalent to \axiom{setrest!(u,v)}.
    setlast_!: (%,S) -> S
      ++ setlast!(u,x) destructively changes the last element of u to x.
    setelt: (%,"last",S) -> S
      ++ setelt(u,"last",x) (also written: \axiom{u.last := b})
      ++ is equivalent to \axiom{setlast!(u,v)}.
    split_!: (%,Integer) -> %
       ++ split!(u,n) splits u into two aggregates: \axiom{v = rest(u,n)}
       ++ and \axiom{w = first(u,n)}, returning \axiom{v}.
       ++ Note that afterwards \axiom{rest(u,n)} returns \axiom{empty()}.
add
 cycleMax ==> 1000

 findCycle: % -> %

 elt(x, "first") == first x
 elt(x,  "last") == last x
 elt(x,  "rest") == rest x
 second x          == first rest x
 third x          == first rest rest x
 cyclic? x          == not empty? x and not empty? findCycle x
 last x          == first tail x

 nodes x ==
   l := empty()$List(%)
   while not empty? x repeat
     l := concat(x, l)
     x := rest x
   reverse_! l

 children x ==
   l := empty()$List(%)
   empty? x => l
   concat(rest x,l)

 leaf? x == empty? x

 value x ==
```

```
    empty? x => error "value of empty object"
    first x

  less?(l, n) ==
    i := n::Integer
    while i > 0 and not empty? l repeat (l := rest l; i := i - 1)
    i > 0

  more?(l, n) ==
    i := n::Integer
    while i > 0 and not empty? l repeat (l := rest l; i := i - 1)
    zero?(i) and not empty? l

  size?(l, n) ==
    i := n::Integer
    while not empty? l and i > 0 repeat (l := rest l; i := i - 1)
    empty? l and zero? i

  #x ==
    for k in 0.. while not empty? x repeat
      k = cycleMax and cyclic? x => error "cyclic list"
      x := rest x
    k

  tail x ==
    empty? x => error "empty list"
    y := rest x
    for k in 0.. while not empty? y repeat
      k = cycleMax and cyclic? x => error "cyclic list"
      y := rest(x := y)
    x

  findCycle x ==
    y := rest x
    while not empty? y repeat
      if eq?(x, y) then return x
      x := rest x
      y := rest y
      if empty? y then return y
      if eq?(x, y) then return y
      y := rest y
    y

  cycleTail x ==
    empty?(y := x := cycleEntry x) => x
    z := rest x
    while not eq?(x,z) repeat (y := z; z := rest z)
    y

  cycleEntry x ==
```

```
  empty? x => x
  empty?(y := findCycle x) => y
  z := rest y
  for l in 1.. while not eq?(y,z) repeat z := rest z
  y := x
  for k in 1..l repeat y := rest y
  while not eq?(x,y) repeat (x := rest x; y := rest y)
  x

cycleLength x ==
  empty? x => 0
  empty?(x := findCycle x) => 0
  y := rest x
  for k in 1.. while not eq?(x,y) repeat y := rest y
  k

rest(x, n) ==
  for i in 1..n repeat
    empty? x => error "Index out of range"
    x := rest x
  x

if % has finiteAggregate then
  last(x, n) ==
    n > (m := #x) => error "index out of range"
    copy rest(x, (m - n)::NonNegativeInteger)

if S has SetCategory then
  x = y ==
    eq?(x, y) => true
    for k in 0.. while not empty? x and not empty? y repeat
      k = cycleMax and cyclic? x => error "cyclic list"
      first x ^= first y => return false
      x := rest x
      y := rest y
    empty? x and empty? y

  node?(u, v) ==
    for k in 0.. while not empty? v repeat
      u = v => return true
      k = cycleMax and cyclic? v => error "cyclic list"
      v := rest v
    u=v

if % has shallowlyMutable then
  setelt(x, "first", a) == setfirst_!(x, a)
  setelt(x,  "last", a) == setlast_!(x, a)
  setelt(x,  "rest", a) == setrest_!(x, a)
  concat(x:%, y:%)          == concat_!(copy x, y)
```

```
    setlast_!(x, s) ==
      empty? x => error "setlast: empty list"
      setfirst_!(tail x, s)
      s

    setchildren_!(u,lv) ==
      #lv=1 => setrest_!(u, first lv)
      error "wrong number of children specified"

    setvalue_!(u,s) == setfirst_!(u,s)

    split_!(p, n) ==
      n < 1 => error "index out of range"
      p := rest(p, (n - 1)::NonNegativeInteger)
      q := rest p
      setrest_!(p, empty())
      q

    cycleSplit_! x ==
      empty?(y := cycleEntry x) or eq?(x, y) => y
      z := rest x
      while not eq?(z, y) repeat (x := z; z := rest z)
      setrest_!(x, empty())
      y
```

———-

— URAGG.dotabb —

```
"URAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=URAGG"];
"URAGG" -> "RCAGG"
```

———-

— URAGG.dotfull —

```
"UnaryRecursiveAggregate(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=URAGG"];
"UnaryRecursiveAggregate(a:Type)" -> "RecursiveAggregate(a:Type)"
```

———-

— URAGG.dotpic —

```
digraph pic {
```

```
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"UnaryRecursiveAggregate(a:Type)" [color=lightblue];
"UnaryRecursiveAggregate(a:Type)" -> "RecursiveAggregate(a:Type)"

"RecursiveAggregate(a:Type)" [color=lightblue];
"RecursiveAggregate(a:Type)" -> "HomogeneousAggregate(a:Type)"

"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "Aggregate()"

"Aggregate()" [color=lightblue];
"Aggregate()" -> "Type()"

"Type()" [color=lightblue];
"Type()" -> "Category"

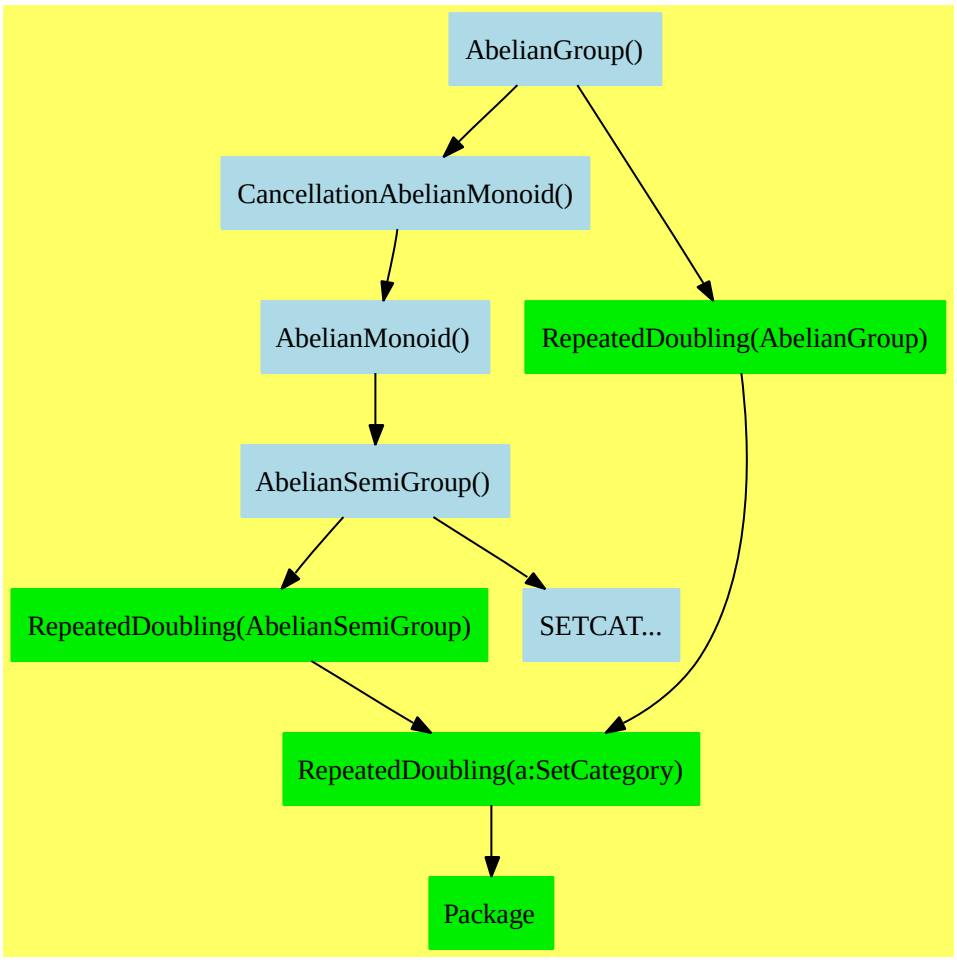
"Category" [color=lightblue];
}
```

———-

Chapter 7

Category Layer 6

7.0.97 AbelianGroup (ABELGRP)



— **AbelianGroup.input** —

```
)set break resume
)sys rm -f AbelianGroup.output
)spool AbelianGroup.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show AbelianGroup
--R AbelianGroup  is a category constructor
--R Abbreviation for AbelianGroup is ABELGRP
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for ABELGRP
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (Integer,%) -> %                ?*? : (PositiveInteger,%) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 0 : () -> %                           coerce : % -> OutputForm
--R hash : % -> SingleInteger             latex : % -> String
--R sample : () -> %                      zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **AbelianGroup.help** —

```
==========================================================================
AbelianGroup examples
==========================================================================

See Also:
o )show AbelianGroup
```

———-

See:

⇒ “FiniteDivisorCategory” (FDIVCAT) 8.0.109 on page 616
⇒ “FunctionSpace” (FS) 17.0.195 on page 1336
⇒ “LeftModule” (LMODULE) 8.0.113 on page 655
⇒ “NonAssociativeRng” (NARNG) 8.0.116 on page 679
⇒ “OrderedAbelianGroup” (OAGROUP) 9.0.126 on page 755
⇒ “RightModule” (RMODULE) 8.0.120 on page 715
⇒ “Rng” (RNG) 8.0.121 on page 719
⇐ “CancellationAbelianMonoid” (CABMON) 6.0.83 on page 380

Exports:

0	coerce	hash	latex	sample
subtractIfCan	zero?	?~=?	?*?	?+?
?-?	-?	?=?		

These are directly exported but not implemented:

```
-? : % -> %
```

These are implemented by this category:

```
subtractIfCan : (%,%) -> Union(%,"failed")
?*? : (Integer,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
```

These exports come from (p380) CancellationAbelianMonoid():

```
0 : () -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
sample : () -> %
zero? : % -> Boolean
?~=? : (%,%) -> Boolean
?*? : (PositiveInteger,%) -> %
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
```

— category ABELGRP AbelianGroup —

```
)abbrev category ABELGRP AbelianGroup
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
```

```
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The class of abelian groups, i.e. additive monoids where
++ each element has an additive inverse.
++
++ Axioms\br
++ \tab{5}\spad{-(-x) = x}\br
++ \tab{5}\spad{x+(-x) = 0}
-- following domain must be compiled with subsumption disabled

AbelianGroup(): Category == CancellationAbelianMonoid with
      "-": % -> %
        ++ -x is the additive inverse of x.
      "-": (%,%) -> %
        ++ x-y is the difference of x and y
        ++ i.e. \spad{x + (-y)}.
        -- subsumes the partial subtraction from previous
      "*": (Integer,%) -> %
        ++ n*x is the product of x by the integer n.
    add
      (x:% - y:%):% == x+(-y)
      subtractIfCan(x:%, y:%):Union(%, "failed") == (x-y)::Union(%,"failed")
      n:NonNegativeInteger * x:% == (n::Integer) * x
      import RepeatedDoubling(%)
      if not (% has Ring) then
        n:Integer * x:% ==
          zero? n => 0
          n>0 => double(n pretend PositiveInteger,x)
          double((-n) pretend PositiveInteger,-x)
```

———-

— ABELGRP.dotabb —

```
"ABELGRP"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ABELGRP"];
"ABELGRP" -> "CABMON"
```

———-

— ABELGRP.dotfull —

```
"AbelianGroup()"
```

```
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ABELGRP"];
"AbelianGroup()" -> "CancellationAbelianMonoid()"
"AbelianGroup()" -> "RepeatedDoubling(AbelianGroup)"
```

———-

— ABELGRP.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"AbelianGroup()" [color=lightblue];
"AbelianGroup()" -> "CancellationAbelianMonoid()"
"AbelianGroup()" -> "RepeatedDoubling(AbelianGroup)"

"RepeatedDoubling(AbelianGroup)" [color="#00EE00"];
"RepeatedDoubling(AbelianGroup)" -> "RepeatedDoubling(a:SetCategory)"

"RepeatedDoubling(AbelianSemiGroup)" [color="#00EE00"];
"RepeatedDoubling(AbelianSemiGroup)" -> "RepeatedDoubling(a:SetCategory)"

"RepeatedDoubling(a:SetCategory)" [color="#00EE00"];
"RepeatedDoubling(a:SetCategory)" -> "Package"

"Package" [color="#00EE00"];

"CancellationAbelianMonoid()" [color=lightblue];
"CancellationAbelianMonoid()" -> "AbelianMonoid()"

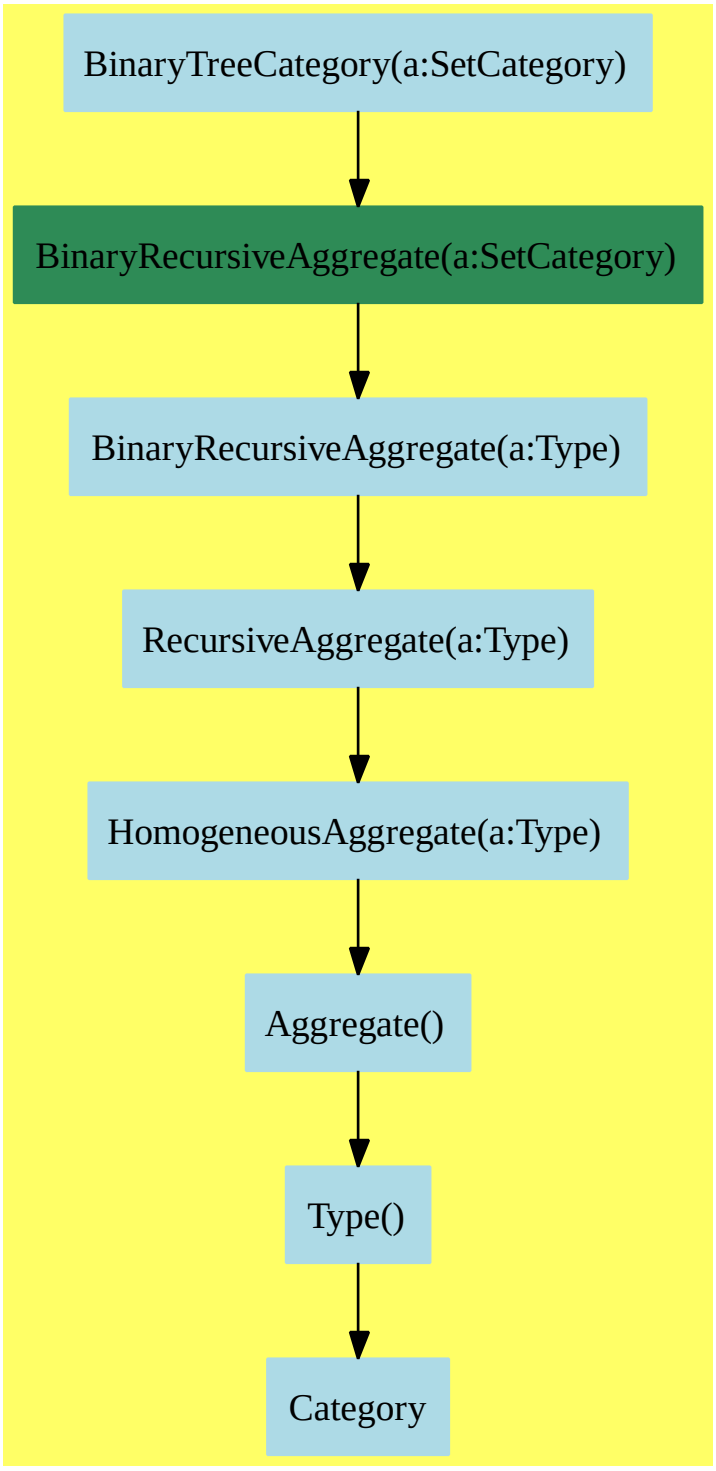
"AbelianMonoid()" [color=lightblue];
"AbelianMonoid()" -> "AbelianSemiGroup()"

"AbelianSemiGroup()" [color=lightblue];
"AbelianSemiGroup()" -> "SETCAT..."
"AbelianSemiGroup()" -> "RepeatedDoubling(AbelianSemiGroup)"

"SETCAT..." [color=lightblue];
}
```

———-

7.0.98 BinaryTreeCategory (BTCAT)



— BinaryTreeCategory.input —

```
)set break resume
)sys rm -f BinaryTreeCategory.output
)spool BinaryTreeCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show BinaryTreeCategory
--R BinaryTreeCategory S: SetCategory  is a category constructor
--R Abbreviation for BinaryTreeCategory is BTCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for BTCAT
--R
--R------------------------------ Operations --------------------------------
--R children : % -> List %               copy : % -> %
--R cyclic? : % -> Boolean               distance : (%,%) -> Integer
--R ?.right : (%,right) -> %             ?.left : (%,left) -> %
--R ?.value : (%,value) -> S             empty : () -> %
--R empty? : % -> Boolean                eq? : (%,%) -> Boolean
--R leaf? : % -> Boolean                 leaves : % -> List S
--R left : % -> %                        map : ((S -> S),%) -> %
--R node : (%,S,%) -> %                  nodes : % -> List %
--R right : % -> %                       sample : () -> %
--R value : % -> S
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R child? : (%,%) -> Boolean if S has SETCAT
--R coerce : % -> OutputForm if S has SETCAT
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R hash : % -> SingleInteger if S has SETCAT
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R node? : (%,%) -> Boolean if S has SETCAT
```

```
--R parts : % -> List S if $ has finiteAggregate
--R setchildren! : (%,List %) -> % if $ has shallowlyMutable
--R setelt : (%,right,%) -> % if $ has shallowlyMutable
--R setelt : (%,left,%) -> % if $ has shallowlyMutable
--R setelt : (%,value,S) -> S if $ has shallowlyMutable
--R setleft! : (%,%) -> % if $ has shallowlyMutable
--R setright! : (%,%) -> % if $ has shallowlyMutable
--R setvalue! : (%,S) -> S if $ has shallowlyMutable
--R size? : (%,NonNegativeInteger) -> Boolean
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— BinaryTreeCategory.help —

```
=====================================================================
BinaryTreeCategory examples
=====================================================================

See Also:
o )show BinaryTreeCategory
```

———-

See:

⇐ "BinaryRecursiveAggregate" (BRAGG) 6.0.82 on page 372

Exports:

any?	child?	children	coerce	copy
count	cyclic?	distance	empty	empty?
eq?	eval	every?	hash	latex
leaf?	leaves	less?	left	map
map!	member?	members	more?	node
node?	nodes	parts	right	sample
setchildren!	setelt	setleft!	setright!	setvalue!
size?	value	#?	?=?	?~=?
?.right	?.left	?.value		

Attributes Exported:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **nil**

These are directly exported but not implemented:

```
node : (%,S,%) -> %
```

These are implemented by this category:

```
copy : % -> %
map! : ((S -> S),%) -> % if $ has shallowlyMutable
#? : % -> NonNegativeInteger if $ has finiteAggregate
```

These exports come from (p372) BinaryRecursiveAggregate(S:SetCategory):

```
any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
child? : (%,%) -> Boolean if S has SETCAT
children : % -> List %
coerce : % -> OutputForm if S has SETCAT
count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
cyclic? : % -> Boolean
distance : (%,%) -> Integer
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
leaf? : % -> Boolean
leaves : % -> List S
left : % -> %
every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
hash : % -> SingleInteger if S has SETCAT
latex : % -> String if S has SETCAT
less? : (%,NonNegativeInteger) -> Boolean
map : ((S -> S),%) -> %
member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
members : % -> List S if $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
parts : % -> List S if $ has finiteAggregate
right : % -> %
sample : () -> %
setchildren! : (%,List %) -> % if $ has shallowlyMutable
setelt : (%,value,S) -> S if $ has shallowlyMutable
```

```
 setelt : (%,right,%) -> % if $ has shallowlyMutable
 setelt : (%,left,%) -> % if $ has shallowlyMutable
 setleft! : (%,%) -> % if $ has shallowlyMutable
 setright! : (%,%) -> % if $ has shallowlyMutable
 setvalue! : (%,S) -> S if $ has shallowlyMutable
 size? : (%,NonNegativeInteger) -> Boolean
 value : % -> S
 ?~=? : (%,%) -> Boolean if S has SETCAT
 ?.value : (%,value) -> S
 ?=? : (%,%) -> Boolean if S has SETCAT
 ?.right : (%,right) -> %
 ?.left : (%,left) -> %
```

— category BTCAT BinaryTreeCategory —

```
)abbrev category BTCAT BinaryTreeCategory
++ Author:W. H. Burge
++ Date Created:17 Feb 1992
++ Date Last Updated:
++ Basic Operations:
++ Related Domains:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ Examples:
++ References:
++ Description:
++ \spadtype{BinaryTreeCategory(S)} is the category of
++ binary trees: a tree which is either empty or else is a
++ \spadfun{node} consisting of a value and a \spadfun{left} and
++ \spadfun{right}, both binary trees.

BinaryTreeCategory(S: SetCategory): Category == _
      BinaryRecursiveAggregate(S) with
   shallowlyMutable
    ++ Binary trees have updateable components
   finiteAggregate
    ++ Binary trees have a finite number of components
   node: (%,S,%) -> %
    ++ node(left,v,right) creates a binary tree with value \spad{v}, a binary
    ++ tree \spad{left}, and a binary tree \spad{right}.
    ++
 add
     cycleTreeMax ==> 5

     copy t ==
       empty? t => empty()
       node(copy left t, value t, copy right t)
```

```
    if % has shallowlyMutable then
      map_!(f,t) ==
        empty? t => t
        t.value := f(t.value)
        map_!(f,left t)
        map_!(f,right t)
        t

    if % has finiteAggregate then
      treeCount : (%, NonNegativeInteger) -> NonNegativeInteger

      #t == treeCount(t,0)

      treeCount(t,k) ==
        empty? t => k
        k := k + 1
        k = cycleTreeMax and cyclic? t => error "cyclic binary tree"
        k := treeCount(left t,k)
        treeCount(right t,k)
```

———-

— **BTCAT.dotabb** —

```
"BTCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=BTCAT"];
"BTCAT" -> "BRAGG"
```

———-

— **BTCAT.dotfull** —

```
"BinaryTreeCategory(a:SetCategory)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=BTCAT"];
"BinaryTreeCategory(a:SetCategory)" ->
   "BinaryRecursiveAggregate(a:SetCategory)"
```

———-

— **BTCAT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];
```

```
"BinaryTreeCategory(a:SetCategory)" [color=lightblue];
"BinaryTreeCategory(a:SetCategory)" ->
   "BinaryRecursiveAggregate(a:SetCategory)"

"BinaryRecursiveAggregate(a:SetCategory)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=BRAGG"];
"BinaryRecursiveAggregate(a:SetCategory)" ->
   "BinaryRecursiveAggregate(a:Type)"

"BinaryRecursiveAggregate(a:Type)" [color=lightblue];
"BinaryRecursiveAggregate(a:Type)" -> "RecursiveAggregate(a:Type)"

"RecursiveAggregate(a:Type)" [color=lightblue];
"RecursiveAggregate(a:Type)" -> "HomogeneousAggregate(a:Type)"

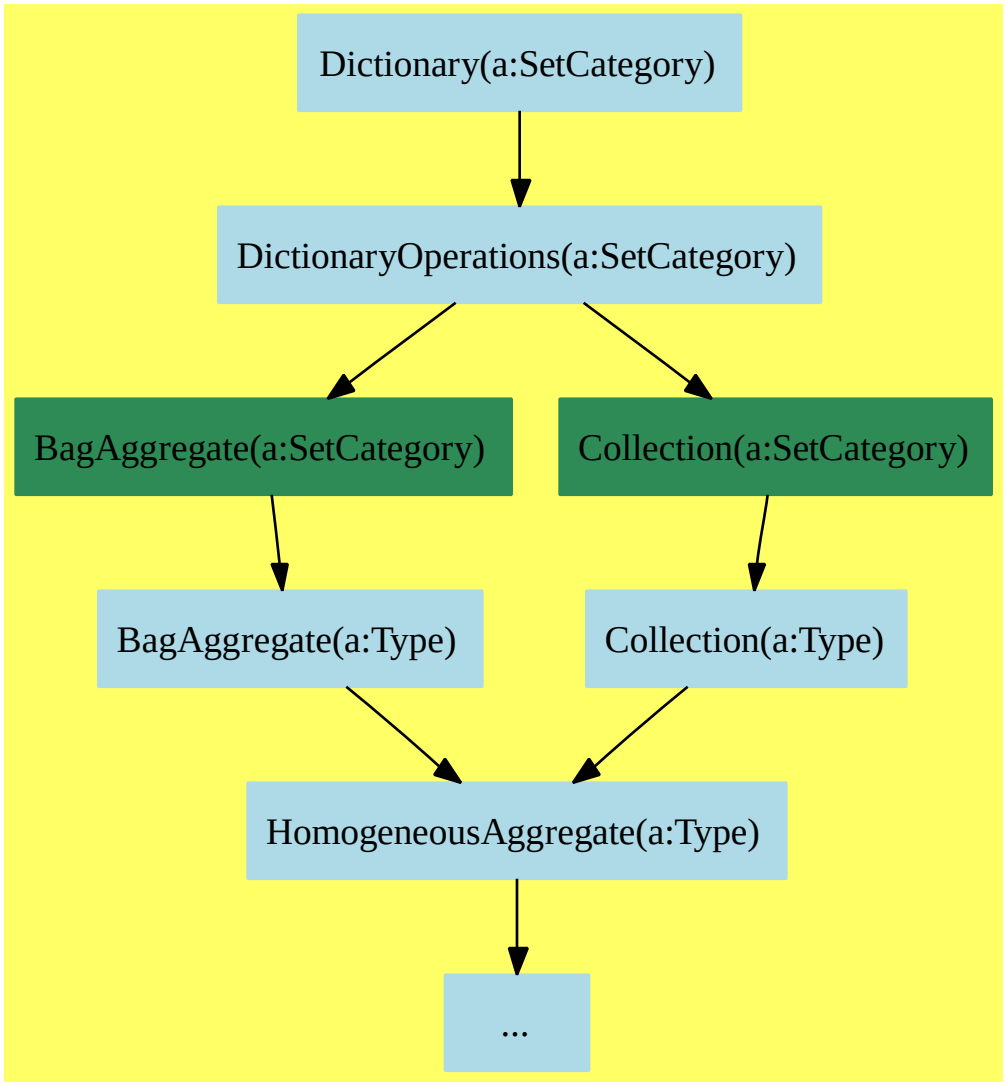
"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "Aggregate()"

"Aggregate()" [color=lightblue];
"Aggregate()" -> "Type()"

"Type()" [color=lightblue];
"Type()" -> "Category"

"Category" [color=lightblue];
}
```

7.0.99 Dictionary (DIAGG)



— **Dictionary.input** —

```
)set break resume
)sys rm -f Dictionary.output
)spool Dictionary.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 1
)show Dictionary
--R Dictionary S: SetCategory  is a category constructor
--R Abbreviation for Dictionary is DIAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for DIAGG
--R
--R------------------------------ Operations --------------------------------
--R bag : List S -> %                    construct : List S -> %
--R copy : % -> %                        dictionary : List S -> %
--R dictionary : () -> %                 empty : () -> %
--R empty? : % -> Boolean                eq? : (%,%) -> Boolean
--R extract! : % -> S                    insert! : (S,%) -> %
--R inspect : % -> S                     map : ((S -> S),%) -> %
--R sample : () -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R coerce : % -> OutputForm if S has SETCAT
--R convert : % -> InputForm if S has KONVERT INFORM
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R find : ((S -> Boolean),%) -> Union(S,"failed")
--R hash : % -> SingleInteger if S has SETCAT
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S,S) -> S if S has SETCAT and $ has finiteAggregate
--R remove : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (S,%) -> % if S has SETCAT and $ has finiteAggregate
--R remove! : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R remove! : (S,%) -> % if $ has finiteAggregate
--R removeDuplicates : % -> % if S has SETCAT and $ has finiteAggregate
--R select : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R select! : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R size? : (%,NonNegativeInteger) -> Boolean
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
```

```
--E 1

)spool
)lisp (bye)
```

———-

— **Dictionary.help** —

```
====================================================================
Dictionary examples
====================================================================

See Also:
o )show Dictionary
```

———-

See:

⇒ “FiniteSetAggregate” (FSAGG) 8.0.110 on page 621
⇒ “KeyedDictionary” (KDAGG) 8.0.111 on page 630
⇐ “DictionaryOperations” (DIOPS) 6.0.84 on page 385

Exports:

any?	bag	coerce	construct	convert
copy	count	dictionary	empty	empty?
eq?	eval	every?	extract!	find
hash	insert!	inspect	latex	less?
map	map!	member?	members	more?
parts	reduce	remove	remove!	removeDuplicates
sample	select	select!	size?	#?
?~=?	?=?			

Attributes exported:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **nil**

Attributes Used:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.

These are implemented by this category:

```
dictionary : List S -> %
select! : ((S -> Boolean),%) -> % if $ has finiteAggregate
?=? : (%,%) -> Boolean if S has SETCAT
remove! : ((S -> Boolean),%) -> % if $ has finiteAggregate
```

These exports come from (p385) DictionaryOperations(S:SetCategory):

```
any? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
bag : List S -> %
coerce : % -> OutputForm if S has SETCAT
construct : List S -> %
convert : % -> InputForm if S has KONVERT INFORM
copy : % -> %
count : (S,%) -> NonNegativeInteger
         if S has SETCAT and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
dictionary : () -> %
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,S,S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,Equation S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,List Equation S) -> %
         if S has EVALAB S and S has SETCAT
every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
extract! : % -> S
find : ((S -> Boolean),%) -> Union(S,"failed")
hash : % -> SingleInteger if S has SETCAT
insert! : (S,%) -> %
inspect : % -> S
latex : % -> String if S has SETCAT
less? : (%,NonNegativeInteger) -> Boolean
map : ((S -> S),%) -> %
map! : ((S -> S),%) -> % if $ has shallowlyMutable
member? : (S,%) -> Boolean
         if S has SETCAT and $ has finiteAggregate
members : % -> List S if $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
parts : % -> List S if $ has finiteAggregate
reduce : (((S,S) -> S),%) -> S if $ has finiteAggregate
reduce : (((S,S) -> S),%,S) -> S if $ has finiteAggregate
reduce : (((S,S) -> S),%,S,S) -> S
       if S has SETCAT and $ has finiteAggregate
remove : ((S -> Boolean),%) -> % if $ has finiteAggregate
```

```
 remove : (S,%) -> % if S has SETCAT and $ has finiteAggregate
 remove! : (S,%) -> % if $ has finiteAggregate
 removeDuplicates : % -> %
          if S has SETCAT and $ has finiteAggregate
 sample : () -> %
 select : ((S -> Boolean),%) -> % if $ has finiteAggregate
 size? : (%,NonNegativeInteger) -> Boolean
 #? : % -> NonNegativeInteger if $ has finiteAggregate
 ?~=? : (%,%) -> Boolean if S has SETCAT
```

— category DIAGG Dictionary —

```
)abbrev category DIAGG Dictionary
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A dictionary is an aggregate in which entries can be inserted,
++ searched for and removed. Duplicates are thrown away on insertion.
++ This category models the usual notion of dictionary which involves
++ large amounts of data where copying is impractical.
++ Principal operations are thus destructive (non-copying) ones.

Dictionary(S:SetCategory): Category ==
 DictionaryOperations S add
   dictionary l ==
     d := dictionary()
     for x in l repeat insert_!(x, d)
     d

   if % has finiteAggregate then
    -- remove(f:S->Boolean,t:%)  == remove_!(f, copy t)
    -- select(f, t)           == select_!(f, copy t)
     select_!(f, t)         == remove_!((x:S):Boolean +-> not f(x), t)

     --extract_! d ==
     --         empty? d => error "empty dictionary"
     --         remove_!(x := first parts d, d, 1)
     --         x

     s = t ==
       eq?(s,t) => true
       #s ^= #t => false
```

```
      _and/[member?(x, t) for x in parts s]

    remove_!(f:S->Boolean, t:%) ==
      for m in parts t repeat if f m then remove_!(m, t)
      t
```

———-

— DIAGG.dotabb —

```
"DIAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=DIAGG"];
"DIAGG" -> "DIOPS"
```

———-

— DIAGG.dotfull —

```
"Dictionary(a:SetCategory)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=DIAGG"];
"Dictionary(a:SetCategory)" -> "DictionaryOperations(a:SetCategory)"

"Dictionary(Record(a:SetCategory,b:SetCategory))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=DIAGG"];
"Dictionary(Record(a:SetCategory,b:SetCategory))" ->
    "Dictionary(a:SetCategory)"
```

———-

— DIAGG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"Dictionary(a:SetCategory)" [color=lightblue];
"Dictionary(a:SetCategory)" -> "DictionaryOperations(a:SetCategory)"

"DictionaryOperations(a:SetCategory)" [color=lightblue];
"DictionaryOperations(a:SetCategory)" -> "BagAggregate(a:SetCategory)"
"DictionaryOperations(a:SetCategory)" -> "Collection(a:SetCategory)"

"BagAggregate(a:SetCategory)" [color=seagreen];
"BagAggregate(a:SetCategory)" -> "BagAggregate(a:Type)"
```

```
"BagAggregate(a:Type)" [color=lightblue];
"BagAggregate(a:Type)" -> "HomogeneousAggregate(a:Type)"

"Collection(a:SetCategory)" [color=seagreen];
"Collection(a:SetCategory)" -> "Collection(a:Type)"

"Collection(a:Type)" [color=lightblue];
"Collection(a:Type)" -> "HomogeneousAggregate(a:Type)"

"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "Aggregate()"

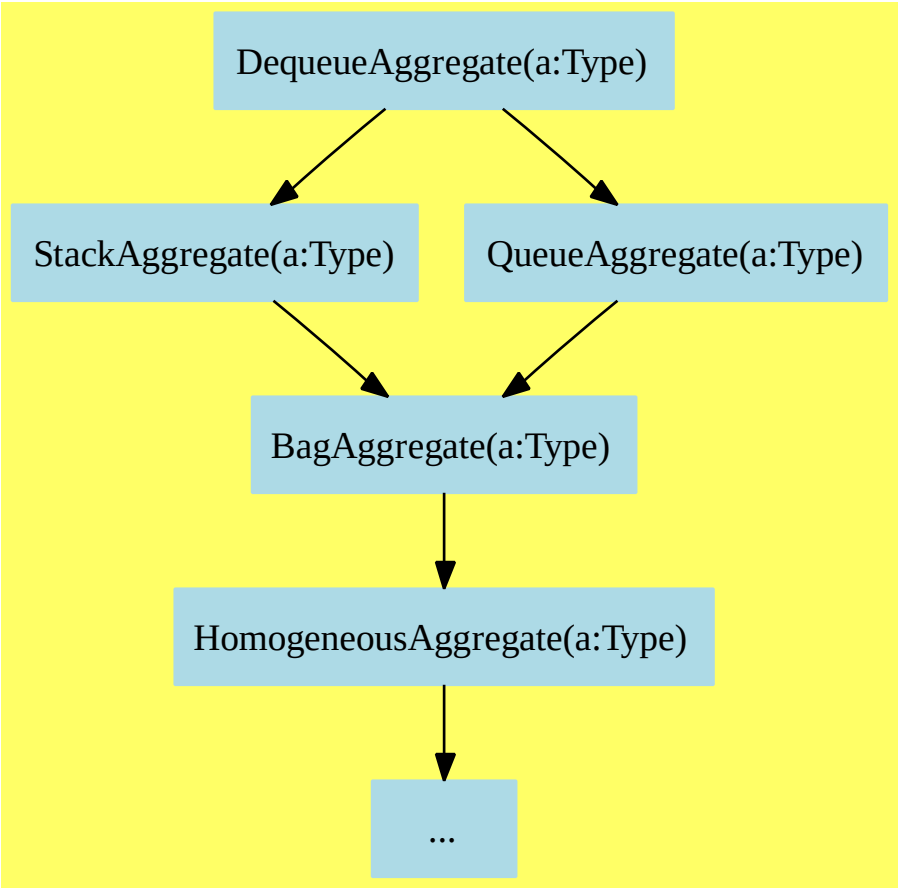
"Aggregate()" [color=lightblue];
"Aggregate()" -> "Type()"

"Type()" [color=lightblue];
"Type()" -> "Category"

"Category" [color=lightblue];
}
```

———-

7.0.100 DequeueAggregate (DQAGG)



— DequeueAggregate.input —

```
)set break resume
)sys rm -f DequeueAggregate.output
)spool DequeueAggregate.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show DequeueAggregate
--R DequeueAggregate S: Type  is a category constructor
--R Abbreviation for DequeueAggregate is DQAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for DQAGG
```

```
--R
--R------------------------------ Operations ------------------------------
--R back : % -> S                          bag : List S -> %
--R bottom! : % -> S                       copy : % -> %
--R depth : % -> NonNegativeInteger        dequeue : List S -> %
--R dequeue : () -> %                      dequeue! : % -> S
--R empty : () -> %                        empty? : % -> Boolean
--R enqueue! : (S,%) -> S                  eq? : (%,%) -> Boolean
--R extract! : % -> S                      extractBottom! : % -> S
--R extractTop! : % -> S                   front : % -> S
--R height : % -> NonNegativeInteger       insert! : (S,%) -> %
--R insertBottom! : (S,%) -> S             insertTop! : (S,%) -> S
--R inspect : % -> S                       length : % -> NonNegativeInteger
--R map : ((S -> S),%) -> %                pop! : % -> S
--R push! : (S,%) -> S                     reverse! : % -> %
--R rotate! : % -> %                       sample : () -> %
--R top : % -> S                           top! : % -> S
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R coerce : % -> OutputForm if S has SETCAT
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R hash : % -> SingleInteger if S has SETCAT
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List S if $ has finiteAggregate
--R size? : (%,NonNegativeInteger) -> Boolean
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— DequeueAggregate.help —

```
====================================================================
```

```
DequeueAggregate examples
==========================================================================

See Also:
o )show DequeueAggregate
```

———-

See:

⇐ “QueueAggregate” (QUAGG) 6.0.93 on page 486
⇐ “StackAggregate” (SKAGG) 6.0.95 on page 500

Exports:

any?	back	bag	bottom!	coerce
copy	count	depth	dequeue	dequeue!
empty	empty?	enqueue!	eq?	eval
every?	extract!	extractBottom!	extractTop!	front
hash	height	insert!	insertBottom!	insertTop!
inspect	latex	length	less?	map
map!	members	member?	more?	parts
pop!	push!	reverse!	rotate!	sample
size?	top	top!	#?	?=?
?~=?				

Attributes exported:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **nil**

These are directly exported but not implemented:

```
bottom! : % -> S
dequeue : () -> %
dequeue : List S -> %
extractBottom! : % -> S
extractTop! : % -> S
height : % -> NonNegativeInteger
insertBottom! : (S,%) -> S
insertTop! : (S,%) -> S
reverse! : % -> %
top! : % -> S
```

These exports come from (p500) StackAggregate(S:Type):

```
any? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
bag : List S -> %
coerce : % -> OutputForm if S has SETCAT
copy : % -> %
count : (S,%) -> NonNegativeInteger
         if S has SETCAT and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
depth : % -> NonNegativeInteger
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,S,S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,Equation S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,List Equation S) -> %
         if S has EVALAB S and S has SETCAT
every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
extract! : % -> S
hash : % -> SingleInteger if S has SETCAT
insert! : (S,%) -> %
inspect : % -> S
latex : % -> String if S has SETCAT
less? : (%,NonNegativeInteger) -> Boolean
map : ((S -> S),%) -> %
map! : ((S -> S),%) -> % if $ has shallowlyMutable
members : % -> List S if $ has finiteAggregate
member? : (S,%) -> Boolean
         if S has SETCAT and $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
parts : % -> List S if $ has finiteAggregate
pop! : % -> S
push! : (S,%) -> S
sample : () -> %
size? : (%,NonNegativeInteger) -> Boolean
top : % -> S
#? : % -> NonNegativeInteger if $ has finiteAggregate
?=? : (%,%) -> Boolean if S has SETCAT
?~=? : (%,%) -> Boolean if S has SETCAT
```

These exports come from (p486) QueueAggregate(S:Type):

```
back : % -> S
dequeue! : % -> S
```

```
 enqueue! : (S,%) -> S
 front : % -> S
 length : % -> NonNegativeInteger
 rotate! : % -> %
```

— category DQAGG DequeueAggregate —

```
)abbrev category DQAGG DequeueAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A dequeue is a doubly ended stack, that is, a bag where first items
++ inserted are the first items extracted, at either the front or
++ the back end of the data structure.

DequeueAggregate(S:Type):
 Category == Join(StackAggregate S,QueueAggregate S) with
   dequeue: () -> %
     ++ dequeue()$D creates an empty dequeue of type D.
   dequeue: List S -> %
     ++ dequeue([x,y,...,z]) creates a dequeue with first (top or front)
     ++ element x, second element y,...,and last (bottom or back) element z.
   height: % -> NonNegativeInteger
     ++ height(d) returns the number of elements in dequeue d.
     ++ Note that \axiom{height(d) = # d}.
   top_!: % -> S
     ++ top!(d) returns the element at the top (front) of the dequeue.
   bottom_!: % -> S
     ++ bottom!(d) returns the element at the bottom (back) of the dequeue.
   insertTop_!: (S,%) -> S
     ++ insertTop!(x,d) destructively inserts x into the dequeue d, that is,
     ++ at the top (front) of the dequeue.
     ++ The element previously at the top of the dequeue becomes the
     ++ second in the dequeue, and so on.
   insertBottom_!: (S,%) -> S
     ++ insertBottom!(x,d) destructively inserts x into the dequeue d
     ++ at the bottom (back) of the dequeue.
   extractTop_!: % -> S
     ++ extractTop!(d) destructively extracts the top (front) element
     ++ from the dequeue d.
     ++ Error: if d is empty.
   extractBottom_!: % -> S
```

```
    ++ extractBottom!(d) destructively extracts the bottom (back) element
    ++ from the dequeue d.
    ++ Error: if d is empty.
  reverse_!: % -> %
    ++ reverse!(d) destructively replaces d by its reverse dequeue, i.e.
    ++ the top (front) element is now the bottom (back) element, and so on.
```

———-

— **DQAGG.dotabb** —

```
"DQAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=DQAGG"];
"DQAGG" -> "SKAGG"
"DQAGG" -> "QUAGG"
```

———-

— **DQAGG.dotfull** —

```
"DequeueAggregate(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=DQAGG"];
"DequeueAggregate(a:Type)" -> "StackAggregate(a:Type)"
"DequeueAggregate(a:Type)" -> "QueueAggregate(a:Type)"

"DequeueAggregate(a:SetCategory)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=DQAGG"];
"DequeueAggregate(a:SetCategory)" -> "DequeueAggregate(a:Type)"
```

———-

— **DQAGG.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"DequeueAggregate(a:Type)" [color=lightblue];
"DequeueAggregate(a:Type)" -> "StackAggregate(a:Type)"
"DequeueAggregate(a:Type)" -> "QueueAggregate(a:Type)"

"StackAggregate(a:Type)" [color=lightblue];
"StackAggregate(a:Type)" -> "BagAggregate(a:Type)"

"QueueAggregate(a:Type)" [color=lightblue];
```

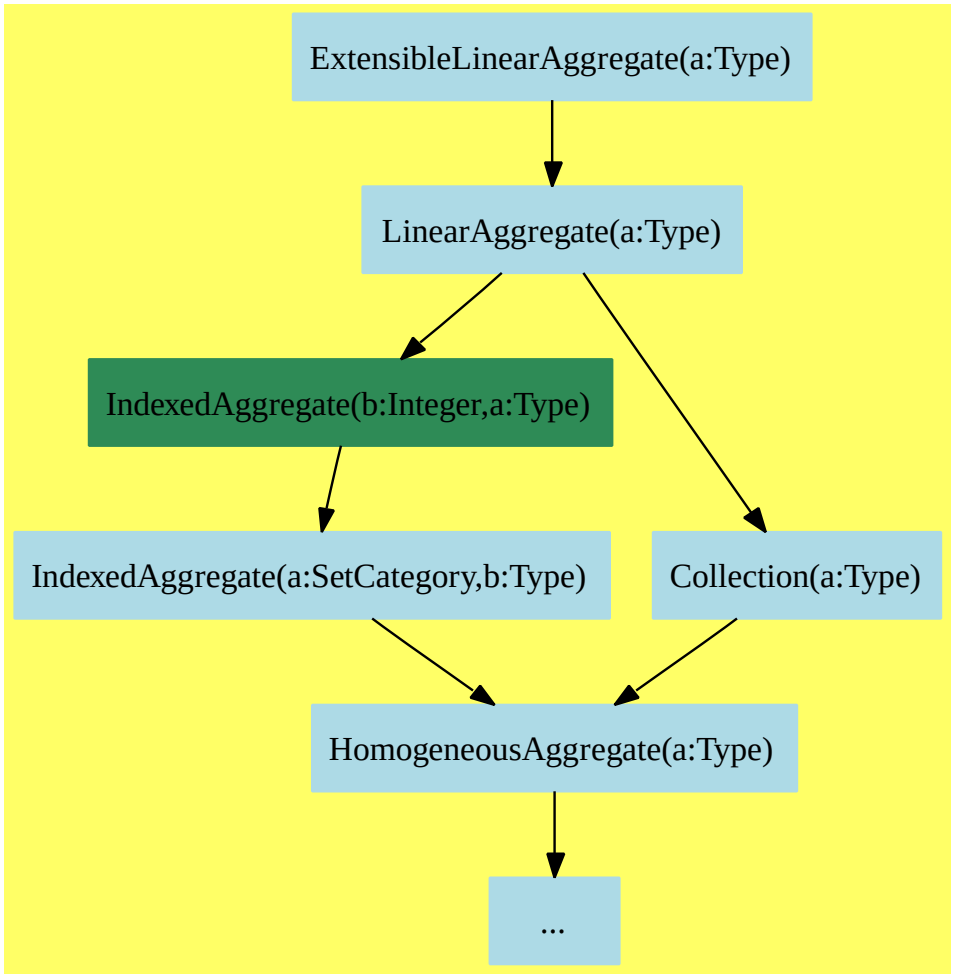
```
"QueueAggregate(a:Type)" -> "BagAggregate(a:Type)"

"BagAggregate(a:Type)" [color=lightblue];
"BagAggregate(a:Type)" -> "HomogeneousAggregate(a:Type)"

"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "..."

"..." [color=lightblue];
}
```

7.0.101 ExtensibleLinearAggregate (ELAGG)



— **ExtensibleLinearAggregate.input** —

```
)set break resume
)sys rm -f ExtensibleLinearAggregate.output
)spool ExtensibleLinearAggregate.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show ExtensibleLinearAggregate
--R ExtensibleLinearAggregate S: Type  is a category constructor
```

```
--R Abbreviation for ExtensibleLinearAggregate is ELAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for ELAGG
--R
--R------------------------------ Operations --------------------------------
--R concat : List % -> %                  concat : (%,%) -> %
--R concat : (S,%) -> %                   concat : (%,S) -> %
--R concat! : (%,%) -> %                  concat! : (%,S) -> %
--R construct : List S -> %               copy : % -> %
--R delete : (%,Integer) -> %             delete! : (%,Integer) -> %
--R ?.? : (%,Integer) -> S                elt : (%,Integer,S) -> S
--R empty : () -> %                       empty? : % -> Boolean
--R entries : % -> List S                 eq? : (%,%) -> Boolean
--R index? : (Integer,%) -> Boolean       indices : % -> List Integer
--R insert : (%,%,Integer) -> %           insert : (S,%,Integer) -> %
--R insert! : (%,%,Integer) -> %          insert! : (S,%,Integer) -> %
--R map : (((S,S) -> S),%,%) -> %         map : ((S -> S),%) -> %
--R new : (NonNegativeInteger,S) -> %     qelt : (%,Integer) -> S
--R sample : () -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R coerce : % -> OutputForm if S has SETCAT
--R convert : % -> InputForm if S has KONVERT INFORM
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R delete : (%,UniversalSegment Integer) -> %
--R delete! : (%,UniversalSegment Integer) -> %
--R ?.? : (%,UniversalSegment Integer) -> %
--R entry? : (S,%) -> Boolean if $ has finiteAggregate and S has SETCAT
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R fill! : (%,S) -> % if $ has shallowlyMutable
--R find : ((S -> Boolean),%) -> Union(S,"failed")
--R first : % -> S if Integer has ORDSET
--R hash : % -> SingleInteger if S has SETCAT
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R maxIndex : % -> Integer if Integer has ORDSET
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R merge! : (%,%) -> % if S has ORDSET
--R merge! : (((S,S) -> Boolean),%,%) -> %
--R minIndex : % -> Integer if Integer has ORDSET
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List S if $ has finiteAggregate
```

```
--R qsetelt! : (%,Integer,S) -> S if $ has shallowlyMutable
--R reduce : (((S,S) -> S),%) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S,S) -> S if S has SETCAT and $ has finiteAggregate
--R remove : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (S,%) -> % if S has SETCAT and $ has finiteAggregate
--R remove! : (S,%) -> % if S has SETCAT
--R remove! : ((S -> Boolean),%) -> %
--R removeDuplicates : % -> % if S has SETCAT and $ has finiteAggregate
--R removeDuplicates! : % -> % if S has SETCAT
--R select : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R select! : ((S -> Boolean),%) -> %
--R setelt : (%,UniversalSegment Integer,S) -> S if $ has shallowlyMutable
--R setelt : (%,Integer,S) -> S if $ has shallowlyMutable
--R size? : (%,NonNegativeInteger) -> Boolean
--R swap! : (%,Integer,Integer) -> Void if $ has shallowlyMutable
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— **ExtensibleLinearAggregate.help** —

```
=====================================================================
ExtensibleLinearAggregate examples
=====================================================================

See Also:
o )show ExtensibleLinearAggregate
```

———-

See:

⇒ “ListAggregate” (LSAGG) 8.0.114 on page 659
⇐ “LinearAggregate” (LNAGG) 6.0.87 on page 404

Exports:

any?	coerce	concat	concat!	construct
copy	convert	count	delete	delete!
elt	empty	empty?	entries	entry?
eval	every?	eq?	fill!	find
first	hash	index?	indices	insert
insert!	latex	less?	map	map!
maxIndex	member?	members	merge!	minIndex
more?	new	parts	qelt	qsetelt!
reduce	remove	remove!	removeDuplicates	removeDuplicates!
sample	select	select!	setelt	size?
swap!	#?	?=?	?.?	?~=?

Attributes Exported:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **nil**

These are directly exported but not implemented:

```
concat! : (%,S) -> %
delete! : (%,Integer) -> %
delete! : (%,UniversalSegment Integer) -> %
insert! : (%,%,Integer) -> %
insert! : (S,%,Integer) -> %
merge! : (((S,S) -> Boolean),%,%) -> %
remove! : ((S -> Boolean),%) -> %
removeDuplicates! : % -> % if S has SETCAT
select! : ((S -> Boolean),%) -> %
```

These are implemented by this category:

```
concat : (%,%) -> %
concat : (%,S) -> %
concat! : (%,%) -> %
delete : (%,Integer) -> %
delete : (%,UniversalSegment Integer) -> %
insert : (%,%,Integer) -> %
insert : (S,%,Integer) -> %
merge! : (%,%) -> % if S has ORDSET
remove : ((S -> Boolean),%) -> % if $ has finiteAggregate
remove : (S,%) -> % if S has SETCAT and $ has finiteAggregate
remove! : (S,%) -> % if S has SETCAT
removeDuplicates : % -> %
         if S has SETCAT and $ has finiteAggregate
select : ((S -> Boolean),%) -> % if $ has finiteAggregate
```

These exports come from (p404) LinearAggregate(S:Type):

```
any? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
coerce : % -> OutputForm if S has SETCAT
concat : List % -> %
concat : (S,%) -> %
construct : List S -> %
copy : % -> %
count : (S,%) -> NonNegativeInteger
         if S has SETCAT and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
convert : % -> InputForm if S has KONVERT INFORM
elt : (%,Integer,S) -> S
empty : () -> %
empty? : % -> Boolean
entries : % -> List S
entry? : (S,%) -> Boolean
         if $ has finiteAggregate and S has SETCAT
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,S,S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,Equation S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,List Equation S) -> %
         if S has EVALAB S and S has SETCAT
every? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
index? : (Integer,%) -> Boolean
indices : % -> List Integer
fill! : (%,S) -> % if $ has shallowlyMutable
find : ((S -> Boolean),%) -> Union(S,"failed")
first : % -> S if Integer has ORDSET
hash : % -> SingleInteger if S has SETCAT
latex : % -> String if S has SETCAT
less? : (%,NonNegativeInteger) -> Boolean
map : (((S,S) -> S),%,%) -> %
map : ((S -> S),%) -> %
map! : ((S -> S),%) -> % if $ has shallowlyMutable
member? : (S,%) -> Boolean
         if S has SETCAT and $ has finiteAggregate
maxIndex : % -> Integer if Integer has ORDSET
new : (NonNegativeInteger,S) -> %
members : % -> List S if $ has finiteAggregate
minIndex : % -> Integer if Integer has ORDSET
more? : (%,NonNegativeInteger) -> Boolean
parts : % -> List S if $ has finiteAggregate
qelt : (%,Integer) -> S
qsetelt! : (%,Integer,S) -> S
```

```
          if $ has shallowlyMutable
 reduce : (((S,S) -> S),%) -> S
          if $ has finiteAggregate
 reduce : (((S,S) -> S),%,S) -> S
          if $ has finiteAggregate
 reduce : (((S,S) -> S),%,S,S) -> S
          if S has SETCAT and $ has finiteAggregate
 sample : () -> %
 setelt : (%,Integer,S) -> S if $ has shallowlyMutable
 setelt : (%,UniversalSegment Integer,S) -> S
          if $ has shallowlyMutable
 size? : (%,NonNegativeInteger) -> Boolean
 swap! : (%,Integer,Integer) -> Void
          if $ has shallowlyMutable
 #? : % -> NonNegativeInteger if $ has finiteAggregate
 ?=? : (%,%) -> Boolean if S has SETCAT
 ?~=? : (%,%) -> Boolean if S has SETCAT
 ?.? : (%,Integer) -> S
 ?.? : (%,UniversalSegment Integer) -> %
```

— category ELAGG ExtensibleLinearAggregate —

```
)abbrev category ELAGG ExtensibleLinearAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ An extensible aggregate is one which allows insertion and deletion of
++ entries. These aggregates are models of lists and streams which are
++ represented by linked structures so as to make insertion, deletion, and
++ concatenation efficient. However, access to elements of these
++ extensible aggregates is generally slow since access is made from the end.
++ See \spadtype{FlexibleArray} for an exception.

ExtensibleLinearAggregate(S:Type):Category == LinearAggregate S with
   shallowlyMutable
   concat_!: (%,S) -> %
     ++ concat!(u,x) destructively adds element x to the end of u.
   concat_!: (%,%) -> %
     ++ concat!(u,v) destructively appends v to the end of u.
     ++ v is unchanged
   delete_!: (%,Integer) -> %
     ++ delete!(u,i) destructively deletes the \axiom{i}th element of u.
```

```
    ++
    ++E Data:=Record(age:Integer,gender:String)
    ++E a1:AssociationList(String,Data):=table()
    ++E a1."tim":=[55,"male"]$Data
    ++E delete!(a1,1)

  delete_!: (%,UniversalSegment(Integer)) -> %
    ++ delete!(u,i..j) destructively deletes elements u.i through u.j.
  remove_!: (S->Boolean,%) -> %
    ++ remove!(p,u) destructively removes all elements x of
    ++ u such that \axiom{p(x)} is true.
  insert_!: (S,%,Integer) -> %
    ++ insert!(x,u,i) destructively inserts x into u at position i.
  insert_!: (%,%,Integer) -> %
    ++ insert!(v,u,i) destructively inserts aggregate v into u
    ++ at position i.
  merge_!: ((S,S)->Boolean,%,%) -> %
    ++ merge!(p,u,v) destructively merges u and v using predicate p.
  select_!: (S->Boolean,%) -> %
    ++ select!(p,u) destructively changes u by keeping only values
    ++ x such that \axiom{p(x)}.
  if S has SetCategory then
    remove_!: (S,%) -> %
      ++ remove!(x,u) destructively removes all values x from u.
    removeDuplicates_!: % -> %
      ++ removeDuplicates!(u) destructively removes duplicates from u.
  if S has OrderedSet then merge_!: (%,%) -> %
      ++ merge!(u,v) destructively merges u and v in ascending order.
add
  delete(x:%, i:Integer)           == delete_!(copy x, i)
  delete(x:%, i:UniversalSegment(Integer))           == delete_!(copy x, i)
  remove(f:S -> Boolean, x:%)   == remove_!(f, copy x)
  insert(s:S, x:%, i:Integer)   == insert_!(s, copy x, i)
  insert(w:%, x:%, i:Integer)   == insert_!(copy w, copy x, i)
  select(f, x)                   == select_!(f, copy x)
  concat(x:%, y:%)           == concat_!(copy x, y)
  concat(x:%, y:S)           == concat_!(copy x, new(1, y))
  concat_!(x:%, y:S)           == concat_!(x, new(1, y))
  if S has SetCategory then
    remove(s:S, x:%)             == remove_!(s, copy x)
    remove_!(s:S, x:%)             == remove_!(y +-> y = s, x)
    removeDuplicates(x:%)   == removeDuplicates_!(copy x)

  if S has OrderedSet then
    merge_!(x, y) == merge_!(_<$S, x, y)
```

———-

— **ELAGG.dotabb** —

```
"ELAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=ELAGG"];
"ELAGG" -> "LNAGG"
```

———-

— **ELAGG.dotfull** —

```
"ExtensibleLinearAggregate(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ELAGG"];
"ExtensibleLinearAggregate(a:Type)" -> "LinearAggregate(a:Type)"
```

———-

— **ELAGG.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"ExtensibleLinearAggregate(a:Type)" [color=lightblue];
"ExtensibleLinearAggregate(a:Type)" -> "LinearAggregate(a:Type)"

"LinearAggregate(a:Type)" [color=lightblue];
"LinearAggregate(a:Type)" -> "IndexedAggregate(b:Integer,a:Type)"
"LinearAggregate(a:Type)" -> "Collection(a:Type)"

"IndexedAggregate(b:Integer,a:Type)" [color=seagreen];
"IndexedAggregate(b:Integer,a:Type)" ->
    "IndexedAggregate(a:SetCategory,b:Type)"

"IndexedAggregate(a:SetCategory,b:Type)" [color=lightblue];
"IndexedAggregate(a:SetCategory,b:Type)" ->
    "HomogeneousAggregate(a:Type)"

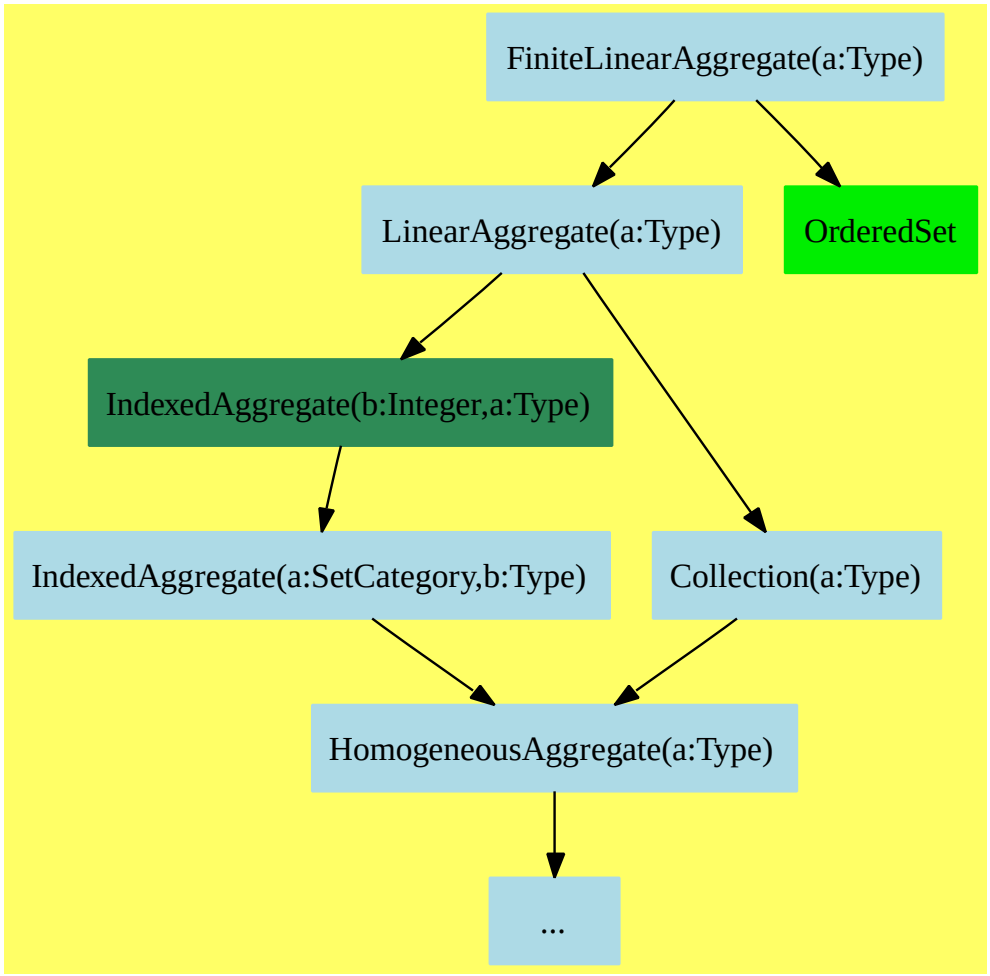
"Collection(a:Type)" [color=lightblue];
"Collection(a:Type)" -> "HomogeneousAggregate(a:Type)"

"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "..."

"..." [color=lightblue];
}
```

———-

7.0.102 FiniteLinearAggregate (FLAGG)



— **FiniteLinearAggregate.input** —

```
)set break resume
)sys rm -f FiniteLinearAggregate.output
)spool FiniteLinearAggregate.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 1
)show FiniteLinearAggregate
--R FiniteLinearAggregate S: Type  is a category constructor
--R Abbreviation for FiniteLinearAggregate is FLAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FLAGG
--R
--R------------------------------ Operations --------------------------------
--R concat : List % -> %                  concat : (%,%) -> %
--R concat : (S,%) -> %                   concat : (%,S) -> %
--R construct : List S -> %               copy : % -> %
--R delete : (%,Integer) -> %             ?.? : (%,Integer) -> S
--R elt : (%,Integer,S) -> S              empty : () -> %
--R empty? : % -> Boolean                 entries : % -> List S
--R eq? : (%,%) -> Boolean                index? : (Integer,%) -> Boolean
--R indices : % -> List Integer           insert : (%,%,Integer) -> %
--R insert : (S,%,Integer) -> %           map : (((S,S) -> S),%,%) -> %
--R map : ((S -> S),%) -> %               new : (NonNegativeInteger,S) -> %
--R qelt : (%,Integer) -> S               reverse : % -> %
--R sample : () -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?<? : (%,%) -> Boolean if S has ORDSET
--R ?<=? : (%,%) -> Boolean if S has ORDSET
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R ?>? : (%,%) -> Boolean if S has ORDSET
--R ?>=? : (%,%) -> Boolean if S has ORDSET
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R coerce : % -> OutputForm if S has SETCAT
--R convert : % -> InputForm if S has KONVERT INFORM
--R copyInto! : (%,%,Integer) -> % if $ has shallowlyMutable
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R delete : (%,UniversalSegment Integer) -> %
--R ?.? : (%,UniversalSegment Integer) -> %
--R entry? : (S,%) -> Boolean if $ has finiteAggregate and S has SETCAT
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R fill! : (%,S) -> % if $ has shallowlyMutable
--R find : ((S -> Boolean),%) -> Union(S,"failed")
--R first : % -> S if Integer has ORDSET
--R hash : % -> SingleInteger if S has SETCAT
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R max : (%,%) -> % if S has ORDSET
--R maxIndex : % -> Integer if Integer has ORDSET
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
```

```
--R members : % -> List S if $ has finiteAggregate
--R merge : (%,%) -> % if S has ORDSET
--R merge : (((S,S) -> Boolean),%,%) -> %
--R min : (%,%) -> % if S has ORDSET
--R minIndex : % -> Integer if Integer has ORDSET
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List S if $ has finiteAggregate
--R position : (S,%,Integer) -> Integer if S has SETCAT
--R position : (S,%) -> Integer if S has SETCAT
--R position : ((S -> Boolean),%) -> Integer
--R qsetelt! : (%,Integer,S) -> S if $ has shallowlyMutable
--R reduce : (((S,S) -> S),%) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S,S) -> S if S has SETCAT and $ has finiteAggregate
--R remove : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (S,%) -> % if S has SETCAT and $ has finiteAggregate
--R removeDuplicates : % -> % if S has SETCAT and $ has finiteAggregate
--R reverse! : % -> % if $ has shallowlyMutable
--R select : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R setelt : (%,UniversalSegment Integer,S) -> S if $ has shallowlyMutable
--R setelt : (%,Integer,S) -> S if $ has shallowlyMutable
--R size? : (%,NonNegativeInteger) -> Boolean
--R sort : % -> % if S has ORDSET
--R sort : (((S,S) -> Boolean),%) -> %
--R sort! : % -> % if S has ORDSET and $ has shallowlyMutable
--R sort! : (((S,S) -> Boolean),%) -> % if $ has shallowlyMutable
--R sorted? : % -> Boolean if S has ORDSET
--R sorted? : (((S,S) -> Boolean),%) -> Boolean
--R swap! : (%,Integer,Integer) -> Void if $ has shallowlyMutable
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— FiniteLinearAggregate.help —

```
=====================================================================
FiniteLinearAggregate examples
=====================================================================

See Also:
o )show FiniteLinearAggregate
```

———-

See:

⇒ "OneDimensionalArrayAggregate" (A1AGG) 8.0.117 on page 684
⇒ "ListAggregate" (LSAGG) 8.0.114 on page 659
⇐ "LinearAggregate" (LNAGG) 6.0.87 on page 404
⇐ "OrderedSet" (ORDSET) 4.0.56 on page 216

Exports:

any?	coerce	concat	construct	convert
copy	copyInto!	count	delete	elt
empty	empty?	entries	entry?	eq?
eval	every?	fill!	find	first
hash	index?	indices	insert	latex
less?	map	map!	max	maxIndex
member?	members	merge	min	minIndex
more?	new	parts	position	qelt
qsetelt!	reduce	remove	removeDuplicates	reverse
reverse!	sample	select	setelt	size?
sort	sort!	sorted?	swap!	#
?.?	?<?	?<=?	?=?	?>?
?>=?	?~=?			

Attributes Exported:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **nil**

Attributes Used:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.

These are directly exported but not implemented:

```
copyInto! : (%,%,Integer) -> %
         if $ has shallowlyMutable
merge : (((S,S) -> Boolean),%,%) -> %
sorted? : (((S,S) -> Boolean),%) -> Boolean
position : (S,%,Integer) -> Integer if S has SETCAT
position : ((S -> Boolean),%) -> Integer
reverse! : % -> % if $ has shallowlyMutable
sort! : (((S,S) -> Boolean),%) -> %
         if $ has shallowlyMutable
```

These are implemented by this category:

```
merge : (%,%) -> % if S has ORDSET
position : (S,%) -> Integer if S has SETCAT
reverse : % -> %
sort : % -> % if S has ORDSET
sort : (((S,S) -> Boolean),%) -> %
sorted? : % -> Boolean if S has ORDSET
sort! : % -> %
         if S has ORDSET and $ has shallowlyMutable
```

These exports come from (p404) LinearAggregate(S:Type):

```
any? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
coerce : % -> OutputForm if S has SETCAT
concat : List % -> %
concat : (%,%) -> %
concat : (S,%) -> %
concat : (%,S) -> %
construct : List S -> %
convert : % -> InputForm if S has KONVERT INFORM
copy : % -> %
count : (S,%) -> NonNegativeInteger
         if S has SETCAT and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
delete : (%,UniversalSegment Integer) -> %
delete : (%,Integer) -> %
elt : (%,Integer,S) -> S
empty : () -> %
empty? : % -> Boolean
entries : % -> List S
entry? : (S,%) -> Boolean
         if $ has finiteAggregate and S has SETCAT
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,S,S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,Equation S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,List Equation S) -> %
         if S has EVALAB S and S has SETCAT
every? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
fill! : (%,S) -> % if $ has shallowlyMutable
find : ((S -> Boolean),%) -> Union(S,"failed")
first : % -> S if Integer has ORDSET
hash : % -> SingleInteger if S has SETCAT
index? : (Integer,%) -> Boolean
indices : % -> List Integer
```

```
insert : (S,%,Integer) -> %
insert : (%,%,Integer) -> %
latex : % -> String if S has SETCAT
less? : (%,NonNegativeInteger) -> Boolean
map : (((S,S) -> S),%,%) -> %
map : ((S -> S),%) -> %
map! : ((S -> S),%) -> % if $ has shallowlyMutable
maxIndex : % -> Integer if Integer has ORDSET
member? : (S,%) -> Boolean
         if S has SETCAT and $ has finiteAggregate
members : % -> List S if $ has finiteAggregate
minIndex : % -> Integer if Integer has ORDSET
more? : (%,NonNegativeInteger) -> Boolean
new : (NonNegativeInteger,S) -> %
parts : % -> List S if $ has finiteAggregate
qelt : (%,Integer) -> S
qsetelt! : (%,Integer,S) -> S
         if $ has shallowlyMutable
reduce : (((S,S) -> S),%) -> S
         if $ has finiteAggregate
reduce : (((S,S) -> S),%,S) -> S
         if $ has finiteAggregate
reduce : (((S,S) -> S),%,S,S) -> S
         if S has SETCAT and $ has finiteAggregate
remove : ((S -> Boolean),%) -> %
         if $ has finiteAggregate
remove : (S,%) -> %
         if S has SETCAT and $ has finiteAggregate
removeDuplicates : % -> %
         if S has SETCAT and $ has finiteAggregate
sample : () -> %
select : ((S -> Boolean),%) -> %
         if $ has finiteAggregate
setelt : (%,Integer,S) -> S
         if $ has shallowlyMutable
setelt : (%,UniversalSegment Integer,S) -> S
         if $ has shallowlyMutable
size? : (%,NonNegativeInteger) -> Boolean
swap! : (%,Integer,Integer) -> Void
         if $ has shallowlyMutable
#? : % -> NonNegativeInteger
         if $ has finiteAggregate
?.? : (%,Integer) -> S
?.? : (%,UniversalSegment Integer) -> %
?~=? : (%,%) -> Boolean if S has SETCAT
?=? : (%,%) -> Boolean if S has SETCAT
```

These exports come from (p216) OrderedSet:

```
max : (%,%) -> % if S has ORDSET
```

```
 min : (%,%) -> % if S has ORDSET
 ?<? : (%,%) -> Boolean if S has ORDSET
 ?<=? : (%,%) -> Boolean if S has ORDSET
 ?>? : (%,%) -> Boolean if S has ORDSET
 ?>=? : (%,%) -> Boolean if S has ORDSET
```

— category FLAGG FiniteLinearAggregate —

```
)abbrev category FLAGG FiniteLinearAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A finite linear aggregate is a linear aggregate of finite length.
++ The finite property of the aggregate adds several exports to the
++ list of exports from \spadtype{LinearAggregate} such as
++ \spadfun{reverse}, \spadfun{sort}, and so on.

FiniteLinearAggregate(S:Type): Category == LinearAggregate S with
   finiteAggregate
   merge: ((S,S)->Boolean,%,%) -> %
      ++ merge(p,a,b) returns an aggregate c which merges \axiom{a} and b.
      ++ The result is produced by examining each element x of \axiom{a}
      ++ and y of b successively. If \axiom{p(x,y)} is true, then x is
      ++ inserted into the result; otherwise y is inserted. If x is
      ++ chosen, the next element of \axiom{a} is examined, and so on.
      ++ When all the elements of one aggregate are examined, the
      ++ remaining elements of the other are appended.
      ++ For example, \axiom{merge(<,[1,3],[2,7,5])} returns
      ++ \axiom{[1,2,3,7,5]}.
   reverse: % -> %
      ++ reverse(a) returns a copy of \axiom{a} with elements
      ++ in reverse order.
   sort: ((S,S)->Boolean,%) -> %
      ++ sort(p,a) returns a copy of \axiom{a} sorted using total ordering
      ++ predicate p.
   sorted?: ((S,S)->Boolean,%) -> Boolean
      ++ sorted?(p,a) tests if \axiom{a} is sorted according to predicate p.
   position: (S->Boolean, %) -> Integer
      ++ position(p,a) returns the index i of the first x in \axiom{a}
      ++ such that \axiom{p(x)} is true, and \axiom{minIndex(a) - 1}
      ++ if there is no such x.
   if S has SetCategory then
```

```
      position: (S, %)        -> Integer
        ++ position(x,a) returns the index i of the first occurrence of
        ++ x in a, and \axiom{minIndex(a) - 1} if there is no such x.
      position: (S,%,Integer) -> Integer
        ++ position(x,a,n) returns the index i of the first occurrence of
        ++ x in \axiom{a} where \axiom{i >= n}, and \axiom{minIndex(a) - 1}
        ++ if no such x is found.
   if S has OrderedSet then
      OrderedSet
      merge: (%,%) -> %
        ++ merge(u,v) merges u and v in ascending order.
        ++ Note that \axiom{merge(u,v) = merge(<=,u,v)}.
      sort: % -> %
        ++ sort(u) returns an u with elements in ascending order.
        ++ Note that \axiom{sort(u) = sort(<=,u)}.
      sorted?: % -> Boolean
        ++ sorted?(u) tests if the elements of u are in ascending order.
   if % has shallowlyMutable then
      copyInto_!: (%,%,Integer) -> %
        ++ copyInto!(u,v,i) returns aggregate u containing a copy of
        ++ v inserted at element i.
      reverse_!: % -> %
        ++ reverse!(u) returns u with its elements in reverse order.
      sort_!: ((S,S)->Boolean,%) -> %
        ++ sort!(p,u) returns u with its elements ordered by p.
      if S has OrderedSet then sort_!: % -> %
        ++ sort!(u) returns u with its elements in ascending order.
 add
    if S has SetCategory then
      position(x:S, t:%) == position(x, t, minIndex t)

    if S has OrderedSet then
--    sorted? l          == sorted?(_<$S, l)
      sorted? l          == sorted?((x,y) +-> x < y or x = y, l)
      merge(x, y) == merge(_<$S, x, y)
      sort l          == sort(_<$S, l)

    if % has shallowlyMutable then
      reverse x         == reverse_! copy x
      sort(f, l) == sort_!(f, copy l)

      if S has OrderedSet then
        sort_! l == sort_!(_<$S, l)
```

———-

— FLAGG.dotabb —

```
"FLAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=FLAGG"];
"FLAGG" -> "LNAGG"
```

———-

— FLAGG.dotfull —

```
"FiniteLinearAggregate(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FLAGG"];
"FiniteLinearAggregate(a:Type)" -> "LinearAggregate(a:Type)"
```

———-

— FLAGG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FiniteLinearAggregate(a:Type)" [color=lightblue];
"FiniteLinearAggregate(a:Type)" -> "LinearAggregate(a:Type)"
"FiniteLinearAggregate(a:Type)" -> "OrderedSet"

"OrderedSet" [color="#00EE00"];

"LinearAggregate(a:Type)" [color=lightblue];
"LinearAggregate(a:Type)" -> "IndexedAggregate(b:Integer,a:Type)"
"LinearAggregate(a:Type)" -> "Collection(a:Type)"

"IndexedAggregate(b:Integer,a:Type)" [color=seagreen];
"IndexedAggregate(b:Integer,a:Type)" ->
    "IndexedAggregate(a:SetCategory,b:Type)"

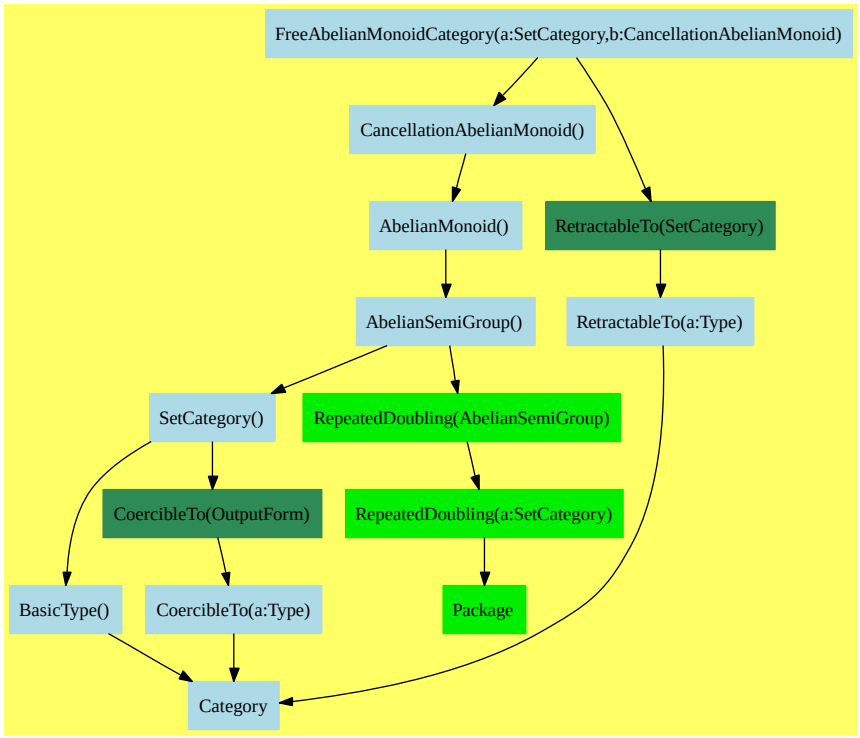
"IndexedAggregate(a:SetCategory,b:Type)" [color=lightblue];
"IndexedAggregate(a:SetCategory,b:Type)" ->
    "HomogeneousAggregate(a:Type)"

"Collection(a:Type)" [color=lightblue];
"Collection(a:Type)" -> "HomogeneousAggregate(a:Type)"

"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "..."

"..." [color=lightblue];
}
```

7.0.103 FreeAbelianMonoidCategory (FAMONC)



— **FreeAbelianMonoidCategory.input** —

```
)set break resume
)sys rm -f FreeAbelianMonoidCategory.output
)spool FreeAbelianMonoidCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FreeAbelianMonoidCategory
--R FreeAbelianMonoidCategory(S: SetCategory,E: CancellationAbelianMonoid)  is a category constr
--R Abbreviation for FreeAbelianMonoidCategory is FAMONC
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FAMONC
```

```
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (E,S) -> %                      ?*? : (PositiveInteger,%) -> %
--R ?+? : (S,%) -> %                      ?+? : (%,%) -> %
--R ?=? : (%,%) -> Boolean                0 : () -> %
--R coefficient : (S,%) -> E              coerce : S -> %
--R coerce : % -> OutputForm              hash : % -> SingleInteger
--R latex : % -> String                   mapCoef : ((E -> E),%) -> %
--R mapGen : ((S -> S),%) -> %            nthCoef : (%,Integer) -> E
--R nthFactor : (%,Integer) -> S          retract : % -> S
--R sample : () -> %                      size : % -> NonNegativeInteger
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R highCommonTerms : (%,%) -> % if E has OAMON
--R retractIfCan : % -> Union(S,"failed")
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R terms : % -> List Record(gen: S,exp: E)
--R
--E 1

)spool
)lisp (bye)
```

———-

— **FreeAbelianMonoidCategory.help** —

```
======================================================================
FreeAbelianMonoidCategory examples
======================================================================

See Also:
o )show FreeAbelianMonoidCategory
```

———-

See:

⇐ “CancellationAbelianMonoid” (CABMON) 6.0.83 on page 380
⇐ “RetractableTo” (RETRACT) 2.0.18 on page 55

Exports:

0	coefficient	coerce	hash	highCommonTerms
latex	mapCoef	mapGen	nthCoef	nthFactor
retract	retractIfCan	sample	size	subtractIfCan
terms	zero?	?*?	?+?	?=?
?~=?				

These are directly exported but not implemented:

```
coefficient : (S,%) -> E
highCommonTerms : (%,%) -> % if E has OAMON
mapCoef : ((E -> E),%) -> %
mapGen : ((S -> S),%) -> %
nthCoef : (%,Integer) -> E
nthFactor : (%,Integer) -> S
size : % -> NonNegativeInteger
terms : % -> List Record(gen: S,exp: E)
?+? : (S,%) -> %
?*? : (E,S) -> %
```

These exports come from (p380) CancellationAbelianMonoid():

```
0 : () -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?*? : (PositiveInteger,%) -> %
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
```

These exports come from (p55) RetractableTo(SetCategory):

```
coerce : S -> %
retract : % -> S
retractIfCan : % -> Union(S,"failed")
```

— category FAMONC FreeAbelianMonoidCategory —

```
)abbrev category FAMONC FreeAbelianMonoidCategory
++ Category for free abelian monoid on any set of generators
++ Author: Manuel Bronstein
++ Date Created: November 1989
++ Date Last Updated: 6 June 1991
++ Description:
++ A free abelian monoid on a set S is the monoid of finite sums of
++ the form \spad{reduce(+,[ni * si])} where the si's are in S, and the ni's
++ are in a given abelian monoid. The operation is commutative.

FreeAbelianMonoidCategory(S: SetCategory, E:CancellationAbelianMonoid): _
  Category == Join(CancellationAbelianMonoid, RetractableTo S) with
```

```
        "+"        : (S, $) -> $
          ++ s + x returns the sum of s and x.
        "*"        : (E, S) -> $
          ++ e * s returns e times s.
        size       : $ -> NonNegativeInteger
          ++ size(x) returns the number of terms in x.
          ++ mapGen(f, a1\^e1 ... an\^en) returns
          ++\spad{f(a1)\^e1 ... f(an)\^en}.
        terms      : $ -> List Record(gen: S, exp: E)
          ++ terms(e1 a1 + ... + en an) returns \spad{[[a1, e1],...,[an, en]]}.
        nthCoef    : ($, Integer) -> E
          ++ nthCoef(x, n) returns the coefficient of the n^th term of x.
        nthFactor  : ($, Integer) -> S
          ++ nthFactor(x, n) returns the factor of the n^th term of x.
        coefficient: (S, $) -> E
          ++ coefficient(s, e1 a1 + ... + en an) returns ei such that
          ++ ai = s, or 0 if s is not one of the ai's.
        mapCoef    : (E -> E, $) -> $
          ++ mapCoef(f, e1 a1 +...+ en an) returns
          ++ \spad{f(e1) a1 +...+ f(en) an}.
        mapGen     : (S -> S, $) -> $
          ++ mapGen(f, e1 a1 +...+ en an) returns
          ++ \spad{e1 f(a1) +...+ en f(an)}.
        if E has OrderedAbelianMonoid then
          highCommonTerms: ($, $) -> $
            ++ highCommonTerms(e1 a1 + ... + en an, f1 b1 + ... + fm bm)
            ++ returns \spad{reduce(+,[max(ei, fi) ci])}
            ++ where ci ranges in the intersection
            ++ of \spad{{a1,...,an}} and \spad{{b1,...,bm}}.
```

———-

— FAMONC.dotabb —

```
"FAMONC"
  [color=lightblue,href="bookvol10.2.pdf#nameddest=FAMONC"];
"FAMONC" -> "CABMON"
"FAMONC" -> "RETRACT"
```

———-

— FAMONC.dotfull —

```
"FreeAbelianMonoidCategory(a:SetCategory,b:CancellationAbelianMonoid)"
  [color=lightblue,href="bookvol10.2.pdf#nameddest=FAMONC"];
"FreeAbelianMonoidCategory(a:SetCategory,b:CancellationAbelianMonoid)" ->
```

```
   "CancellationAbelianMonoid()"
"FreeAbelianMonoidCategory(a:SetCategory,b:CancellationAbelianMonoid)" ->
   "RetractableTo(SetCategory)"
```

———-

— FAMONC.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FreeAbelianMonoidCategory(a:SetCategory,b:CancellationAbelianMonoid)"
  [color=lightblue];
"FreeAbelianMonoidCategory(a:SetCategory,b:CancellationAbelianMonoid)" ->
   "CancellationAbelianMonoid()"
"FreeAbelianMonoidCategory(a:SetCategory,b:CancellationAbelianMonoid)" ->
   "RetractableTo(SetCategory)"

"RetractableTo(SetCategory)" [color=seagreen];
"RetractableTo(SetCategory)" -> "RetractableTo(a:Type)"

"RetractableTo(a:Type)" [color=lightblue];
"RetractableTo(a:Type)" -> "Category"

"CancellationAbelianMonoid()" [color=lightblue];
"CancellationAbelianMonoid()" -> "AbelianMonoid()"

"AbelianMonoid()" [color=lightblue];
"AbelianMonoid()" -> "AbelianSemiGroup()"

"AbelianSemiGroup()" [color=lightblue];
"AbelianSemiGroup()" -> "SetCategory()"
"AbelianSemiGroup()" -> "RepeatedDoubling(AbelianSemiGroup)"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" ->
    "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
```

```
"CoercibleTo(a:Type)" -> "Category"

"RepeatedDoubling(AbelianSemiGroup)" [color="#00EE00"];
"RepeatedDoubling(AbelianSemiGroup)" -> "RepeatedDoubling(a:SetCategory)"

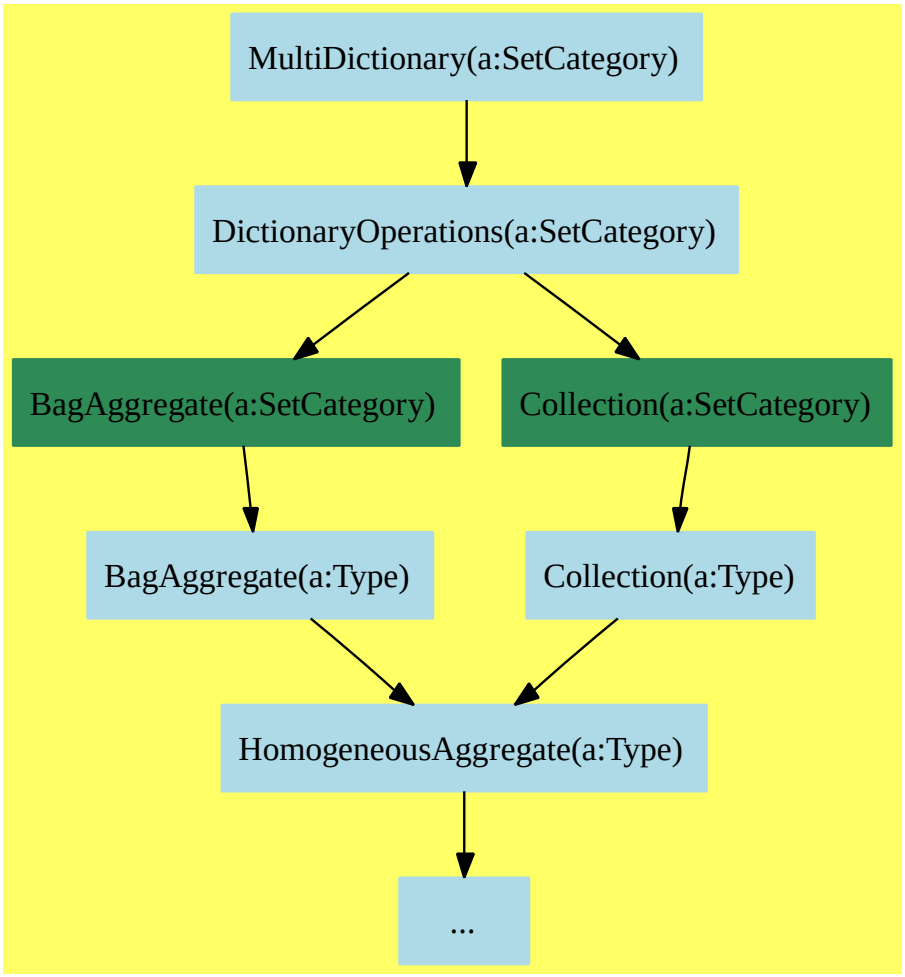
"RepeatedDoubling(a:SetCategory)" [color="#00EE00"];
"RepeatedDoubling(a:SetCategory)" -> "Package"

"Package" [color="#00EE00"];

"Category" [color=lightblue];
}
```

———————-

7.0.104 MultiDictionary (MDAGG)



— **MultiDictionary.input** —

```
)set break resume
)sys rm -f MultiDictionary.output
)spool MultiDictionary.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show MultiDictionary
--R MultiDictionary S: SetCategory  is a category constructor
```

```
--R Abbreviation for MultiDictionary is MDAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for MDAGG
--R
--R------------------------------ Operations --------------------------------
--R bag : List S -> %                    construct : List S -> %
--R copy : % -> %                        dictionary : List S -> %
--R dictionary : () -> %                 empty : () -> %
--R empty? : % -> Boolean                eq? : (%,%) -> Boolean
--R extract! : % -> S                    insert! : (S,%) -> %
--R inspect : % -> S                     map : ((S -> S),%) -> %
--R removeDuplicates! : % -> %           sample : () -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R coerce : % -> OutputForm if S has SETCAT
--R convert : % -> InputForm if S has KONVERT INFORM
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R duplicates : % -> List Record(entry: S,count: NonNegativeInteger)
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R find : ((S -> Boolean),%) -> Union(S,"failed")
--R hash : % -> SingleInteger if S has SETCAT
--R insert! : (S,%,NonNegativeInteger) -> %
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S,S) -> S if S has SETCAT and $ has finiteAggregate
--R remove : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (S,%) -> % if S has SETCAT and $ has finiteAggregate
--R remove! : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R remove! : (S,%) -> % if $ has finiteAggregate
--R removeDuplicates : % -> % if S has SETCAT and $ has finiteAggregate
--R select : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R select! : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R size? : (%,NonNegativeInteger) -> Boolean
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1
```

```
)spool
)lisp (bye)
```

———-

— **MultiDictionary.help** —

```
====================================================================
MultiDictionary examples
====================================================================

See Also:
o )show MultiDictionary
```

———-

See:

⇒ "MultisetAggregate" (MSETAGG) 8.0.115 on page 673
⇐ "DictionaryOperations" (DIOPS) 6.0.84 on page 385

Exports:

any?	bag	coerce	construct	convert
copy	count	dictionary	duplicates	empty
empty?	eq?	eval	every?	extract!
find	hash	insert!	inspect	latex
less?	map	map!	member?	members
more?	parts	reduce	remove	remove!
removeDuplicates	removeDuplicates!	sample	select	select!
size?	#?	?=?	?~=?	

Attributes exported:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **nil**

These are directly exported but not implemented:

```
insert! : (S,%,NonNegativeInteger) -> %
removeDuplicates! : % -> %
duplicates : % -> List Record(entry:S,count:NonNegativeInteger)
```

These exports come from (p385) DictionaryOperations(S:SetCategory):

```
any? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
bag : List S -> %
coerce : % -> OutputForm if S has SETCAT
construct : List S -> %
convert : % -> InputForm if S has KONVERT INFORM
copy : % -> %
count : (S,%) -> NonNegativeInteger
         if S has SETCAT and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
dictionary : () -> %
dictionary : List S -> %
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,S,S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,Equation S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,List Equation S) -> %
         if S has EVALAB S and S has SETCAT
every? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
extract! : % -> S
find : ((S -> Boolean),%) -> Union(S,"failed")
hash : % -> SingleInteger if S has SETCAT
insert! : (S,%) -> %
inspect : % -> S
latex : % -> String if S has SETCAT
less? : (%,NonNegativeInteger) -> Boolean
map : ((S -> S),%) -> %
map! : ((S -> S),%) -> % if $ has shallowlyMutable
member? : (S,%) -> Boolean
         if S has SETCAT and $ has finiteAggregate
members : % -> List S if $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
parts : % -> List S if $ has finiteAggregate
reduce : (((S,S) -> S),%) -> S if $ has finiteAggregate
reduce : (((S,S) -> S),%,S) -> S if $ has finiteAggregate
reduce : (((S,S) -> S),%,S,S) -> S
         if S has SETCAT and $ has finiteAggregate
remove : ((S -> Boolean),%) -> % if $ has finiteAggregate
remove : (S,%) -> %
         if S has SETCAT and $ has finiteAggregate
removeDuplicates : % -> %
         if S has SETCAT and $ has finiteAggregate
remove! : ((S -> Boolean),%) -> % if $ has finiteAggregate
```

```
 remove! : (S,%) -> % if $ has finiteAggregate
 select : ((S -> Boolean),%) -> % if $ has finiteAggregate
 sample : () -> %
 select! : ((S -> Boolean),%) -> % if $ has finiteAggregate
 size? : (%,NonNegativeInteger) -> Boolean
 #? : % -> NonNegativeInteger
          if $ has finiteAggregate
 ?=? : (%,%) -> Boolean if S has SETCAT
 ?~=? : (%,%) -> Boolean if S has SETCAT
```

— category MDAGG MultiDictionary —

```
)abbrev category MDAGG MultiDictionary
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A multi-dictionary is a dictionary which may contain duplicates.
++ As for any dictionary, its size is assumed large so that
++ copying (non-destructive) operations are generally to be avoided.

MultiDictionary(S:SetCategory): Category == DictionaryOperations S with
-- count: (S,%) -> NonNegativeInteger              ++ multiplicity count
   insert_!: (S,%,NonNegativeInteger) -> %
     ++ insert!(x,d,n) destructively inserts n copies of x into dictionary d.
-- remove_!: (S,%,NonNegativeInteger) -> %
--   ++ remove!(x,d,n) destructively removes (up to) n copies of x from
--   ++ dictionary d.
   removeDuplicates_!: % -> %
     ++ removeDuplicates!(d) destructively removes any duplicate values
     ++ in dictionary d.
   duplicates: % -> List Record(entry:S,count:NonNegativeInteger)
     ++ duplicates(d) returns a list of values which have duplicates in d
--   ++ duplicates(d) returns a list of            ++ duplicates iterator
-- to become duplicates: % -> Iterator(D,D)
```

— MDAGG.dotabb —

```
"MDAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=MDAGG"];
```

```
"MDAGG" -> "DIOPS"
```

———-

— **MDAGG.dotfull** —

```
"MultiDictionary(a:SetCategory)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=MDAGG"];
"MultiDictionary(a:SetCategory)" -> "DictionaryOperations(a:SetCategory)"
```

———-

— **MDAGG.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"MultiDictionary(a:SetCategory)" [color=lightblue];
"MultiDictionary(a:SetCategory)" -> "DictionaryOperations(a:SetCategory)"

"DictionaryOperations(a:SetCategory)" [color=lightblue];
"DictionaryOperations(a:SetCategory)" -> "BagAggregate(a:SetCategory)"
"DictionaryOperations(a:SetCategory)" -> "Collection(a:SetCategory)"

"BagAggregate(a:SetCategory)" [color=seagreen];
"BagAggregate(a:SetCategory)" -> "BagAggregate(a:Type)"

"BagAggregate(a:Type)" [color=lightblue];
"BagAggregate(a:Type)" -> "HomogeneousAggregate(a:Type)"

"Collection(a:SetCategory)" [color=seagreen];
"Collection(a:SetCategory)" -> "Collection(a:Type)"

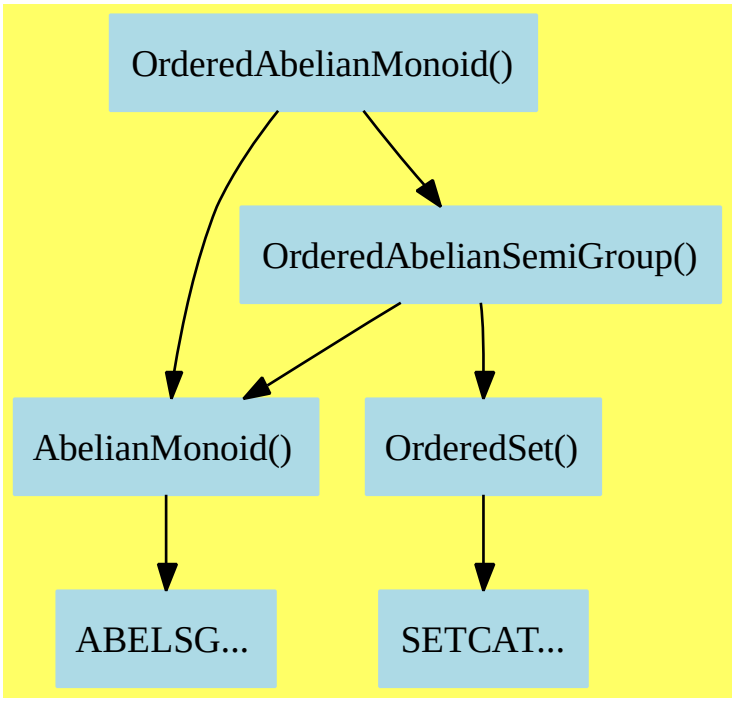
"Collection(a:Type)" [color=lightblue];
"Collection(a:Type)" -> "HomogeneousAggregate(a:Type)"

"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "..."

"..." [color=lightblue];
}
```

———-

7.0.105 OrderedAbelianMonoid (OAMON)



— OrderedAbelianMonoid.input —

```
)set break resume
)sys rm -f OrderedAbelianMonoid.output
)spool OrderedAbelianMonoid.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show OrderedAbelianMonoid
--R OrderedAbelianMonoid  is a category constructor
--R Abbreviation for OrderedAbelianMonoid is OAMON
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for OAMON
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (PositiveInteger,%) -> %        ?+? : (%,%) -> %
--R ?<? : (%,%) -> Boolean                ?<=? : (%,%) -> Boolean
--R ?=? : (%,%) -> Boolean                ?>? : (%,%) -> Boolean
--R ?>=? : (%,%) -> Boolean               0 : () -> %
```

```
--R coerce : % -> OutputForm              hash : % -> SingleInteger
--R latex : % -> String                   max : (%,%) -> %
--R min : (%,%) -> %                      sample : () -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **OrderedAbelianMonoid.help** —

```
=====================================================================
OrderedAbelianMonoid examples
=====================================================================

See Also:
o )show OrderedAbelianMonoid
```

———-

See:

⇒ “OrderedCancellationAbelianMonoid” (OCAMON) 8.0.118 on page 696
⇐ “AbelianMonoid” (ABELMON) 5.0.66 on page 267
⇐ “OrderedAbelianSemiGroup” (OASGP) 6.0.89 on page 456

Exports:

0	coerce	hash	latex	max
min	sample	zero?	?*?	?+?
?<?	?<=?	?=?	?>?	?>=?
?~=?				

These exports come from (p456) OrderedAbelianSemiGroup():

```
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
max : (%,%) -> %
min : (%,%) -> %
?<? : (%,%) -> Boolean
?<=? : (%,%) -> Boolean
?=? : (%,%) -> Boolean
?>? : (%,%) -> Boolean
```

```
 ?>=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
```

These exports come from (p267) AbelianMonoid():

```
 0 : () -> %
 sample : () -> %
 zero? : % -> Boolean
 ?*? : (PositiveInteger,%) -> %
 ?+? : (%,%) -> %
 ?*? : (NonNegativeInteger,%) -> %
```

— category OAMON OrderedAbelianMonoid —

```
)abbrev category OAMON OrderedAbelianMonoid
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ Ordered sets which are also abelian monoids, such that the addition
++ preserves the ordering.

OrderedAbelianMonoid(): Category ==
        Join(OrderedAbelianSemiGroup, AbelianMonoid)
```

———-

— OAMON.dotabb —

```
"OAMON" [color=lightblue,href="bookvol10.2.pdf#nameddest=OAMON"];
"OAMON" -> "OASGP"
"OAMON" -> "ABELMON"
```

———-

— OAMON.dotfull —

```
"OrderedAbelianMonoid()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OAMON"];
```

```
"OrderedAbelianMonoid()" -> "OrderedAbelianSemiGroup()"
"OrderedAbelianMonoid()" -> "AbelianMonoid()"
```

———-

— OAMON.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"OrderedAbelianMonoid()" [color=lightblue];
"OrderedAbelianMonoid()" -> "OrderedAbelianSemiGroup()"
"OrderedAbelianMonoid()" -> "AbelianMonoid()"

"OrderedAbelianSemiGroup()" [color=lightblue];
"OrderedAbelianSemiGroup()" -> "OrderedSet()"
"OrderedAbelianSemiGroup()" -> "AbelianMonoid()"

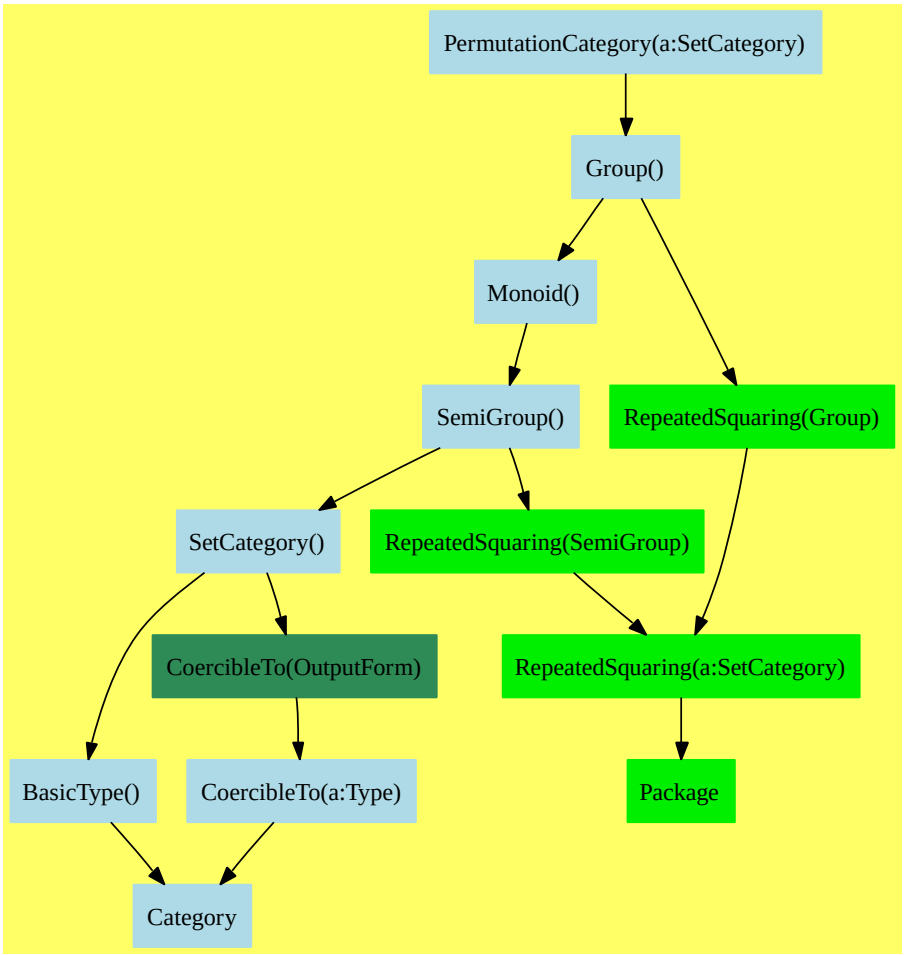
"OrderedSet()" [color=lightblue];
"OrderedSet()" -> "SETCAT..."

"AbelianMonoid()" [color=lightblue];
"AbelianMonoid()" -> "ABELSG..."

"SETCAT..." [color=lightblue];
"ABELSG..." [color=lightblue];
}
```

———-

7.0.106 PermutationCategory (PERMCAT)



— PermutationCategory.input —

```
)set break resume
)sys rm -f PermutationCategory.output
)spool PermutationCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show PermutationCategory
--R PermutationCategory S: SetCategory  is a category constructor
```

```
--R Abbreviation for PermutationCategory is PERMCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PERMCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,%) -> %                      ?**? : (%,Integer) -> %
--R ?**? : (%,PositiveInteger) -> %       ?/? : (%,%) -> %
--R ?<? : (%,%) -> Boolean                ?=? : (%,%) -> Boolean
--R 1 : () -> %                           ?^? : (%,Integer) -> %
--R ?^? : (%,PositiveInteger) -> %        coerce : % -> OutputForm
--R commutator : (%,%) -> %               conjugate : (%,%) -> %
--R cycle : List S -> %                   cycles : List List S -> %
--R ?.? : (%,S) -> S                      eval : (%,S) -> S
--R hash : % -> SingleInteger             inv : % -> %
--R latex : % -> String                   one? : % -> Boolean
--R orbit : (%,S) -> Set S                recip : % -> Union(%,"failed")
--R sample : () -> %                      ?~=? : (%,%) -> Boolean
--R ?**? : (%,NonNegativeInteger) -> %
--R ?<=? : (%,%) -> Boolean if S has ORDSET or S has FINITE
--R ?>? : (%,%) -> Boolean if S has ORDSET or S has FINITE
--R ?>=? : (%,%) -> Boolean if S has ORDSET or S has FINITE
--R ?^? : (%,NonNegativeInteger) -> %
--R max : (%,%) -> % if S has ORDSET or S has FINITE
--R min : (%,%) -> % if S has ORDSET or S has FINITE
--R
--E 1

)spool
)lisp (bye)
```

———-

— PermutationCategory.help —

```
=====================================================================
PermutationCategory examples
=====================================================================

See Also:
o )show PermutationCategory
```

———-

See:

⇐ “Group” (GROUP) 6.0.86 on page 398

Exports:

1	coerce	commutator	conjugate	cycle
cycles	eval	hash	inv	latex
max	min	one?	orbit	recip
sample	?^?	?.?	?~=?	?**?
?<?	?<=?	?>?	?>=?	?*?
?/?	?=?			

Attributes Exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.

These are directly exported but not implemented:

```
cycle : List S -> %
cycles : List List S -> %
eval : (%,S) -> S
orbit : (%,S) -> Set S
?<? : (%,%) -> Boolean
?.? : (%,S) -> S
```

These exports come from (p398) Group():

```
1 : () -> %
coerce : % -> OutputForm
commutator : (%,%) -> %
conjugate : (%,%) -> %
hash : % -> SingleInteger
inv : % -> %
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
?/? : (%,%) -> %
?^? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,Integer) -> %
?*? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?**? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,Integer) -> %
```

These exports come from (p216) OrderedSet():

```
max : (%,%) -> % if S has ORDSET or S has FINITE
min : (%,%) -> % if S has ORDSET or S has FINITE
?>? : (%,%) -> Boolean if S has ORDSET or S has FINITE
?<=? : (%,%) -> Boolean if S has ORDSET or S has FINITE
?>=? : (%,%) -> Boolean if S has ORDSET or S has FINITE
```

— category PERMCAT PermutationCategory —

```
)abbrev category PERMCAT PermutationCategory
++ Authors:  Holger Gollan, Johannes Grabmeier, Gerhard Schneider
++ Date Created: 27 July 1989
++ Date Last Updated: 29 March 1990
++ Basic Operations: cycle, cycles, eval, orbit
++ Related Constructors: PermutationGroup, PermutationGroupExamples
++ Also See: RepresentationTheoryPackage1
++ AMS Classifications:
++ Keywords: permutation, symmetric group
++ References:
++ Description:
++ PermutationCategory provides a categorial environment
++ for subgroups of bijections of a set (i.e. permutations)

PermutationCategory(S:SetCategory): Category  ==  Group with
    cycle   :  List S       ->  %
      ++ cycle(ls) coerces a cycle ls, i.e. a list with not
      ++ repetitions to a permutation, which maps ls.i to
      ++ ls.i+1, indices modulo the length of the list.
      ++ Error: if repetitions occur.
    cycles  :  List List S  ->  %
      ++ cycles(lls) coerces a list list of cycles lls
      ++ to a permutation, each cycle being a list with not
      ++ repetitions, is coerced to the permutation, which maps
      ++ ls.i to ls.i+1, indices modulo the length of the list,
      ++ then these permutations are mutiplied.
      ++ Error: if repetitions occur in one cycle.
    eval  :  (%,S)          ->  S
      ++ eval(p, el) returns the image of el under the
      ++ permutation p.
    elt  :  (%,S)          ->  S
      ++ elt(p, el) returns the image of el under the
      ++ permutation p.
    orbit :  (%,S)          ->  Set S
      ++ orbit(p, el) returns the orbit of el under the
      ++ permutation p, i.e. the set which is given by applications of
      ++ the powers of p to el.
    "<" : (%,%)             ->  Boolean
      ++ p < q is an order relation on permutations.
      ++ Note that this order is only total if and only if S is totally ordered
      ++ or S is finite.
    if S has OrderedSet then OrderedSet
    if S has Finite then OrderedSet
```

———-

— **PERMCAT.dotabb** —

```
"PERMCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PERMCAT"];
"PERMCAT" -> "GROUP"
```

———-

— **PERMCAT.dotfull** —

```
"PermutationCategory(a:SetCategory)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PERMCAT"];
"PermutationCategory(a:SetCategory)" -> "Group()"
```

———-

— **PERMCAT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"PermutationCategory(a:SetCategory)" [color=lightblue];
"PermutationCategory(a:SetCategory)" -> "Group()"

"Group()" [color=lightblue];
"Group()" -> "Monoid()"
"Group()" -> "RepeatedSquaring(Group)"

"Monoid()" [color=lightblue];
"Monoid()" -> "SemiGroup()"

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SetCategory()"
"SemiGroup()" -> "RepeatedSquaring(SemiGroup)"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
```

```
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"RepeatedSquaring(Group)" [color="#00EE00"];
"RepeatedSquaring(Group)" -> "RepeatedSquaring(a:SetCategory)"

"RepeatedSquaring(SemiGroup)" [color="#00EE00"];
"RepeatedSquaring(SemiGroup)" -> "RepeatedSquaring(a:SetCategory)"

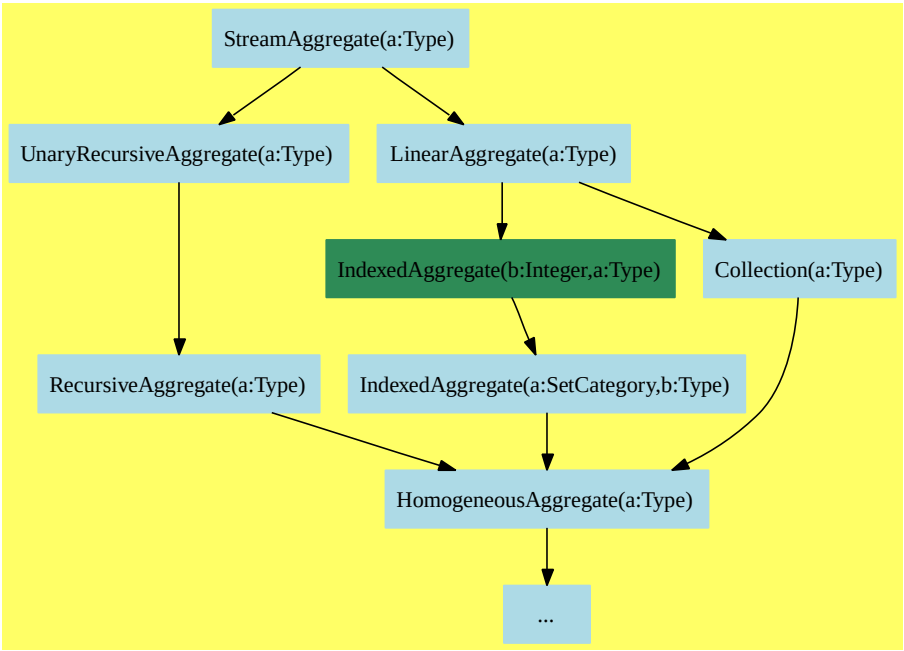
"RepeatedSquaring(a:SetCategory)" [color="#00EE00"];
"RepeatedSquaring(a:SetCategory)" -> "Package"

"Package" [color="#00EE00"];

"Category" [color=lightblue];

}
```

7.0.107 StreamAggregate (STAGG)



— StreamAggregate.input —

```
)set break resume
)sys rm -f StreamAggregate.output
)spool StreamAggregate.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show StreamAggregate
--R StreamAggregate S: Type  is a category constructor
--R Abbreviation for StreamAggregate is STAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for STAGG
--R
--R------------------------------ Operations --------------------------------
--R children : % -> List %                concat : (%,S) -> %
--R concat : List % -> %                  concat : (S,%) -> %
--R concat : (%,%) -> %                   construct : List S -> %
--R copy : % -> %                         cycleEntry : % -> %
--R cycleTail : % -> %                    cyclic? : % -> Boolean
--R delete : (%,Integer) -> %             distance : (%,%) -> Integer
--R elt : (%,Integer,S) -> S              ?.? : (%,Integer) -> S
--R ?.last : (%,last) -> S                ?.rest : (%,rest) -> %
--R ?.first : (%,first) -> S              ?.value : (%,value) -> S
--R empty : () -> %                       empty? : % -> Boolean
--R entries : % -> List S                 eq? : (%,%) -> Boolean
--R explicitlyFinite? : % -> Boolean      first : % -> S
--R index? : (Integer,%) -> Boolean       indices : % -> List Integer
--R insert : (S,%,Integer) -> %           insert : (%,%,Integer) -> %
--R last : % -> S                         leaf? : % -> Boolean
--R leaves : % -> List S                  map : (((S,S) -> S),%,%) -> %
--R map : ((S -> S),%) -> %               new : (NonNegativeInteger,S) -> %
--R nodes : % -> List %                   possiblyInfinite? : % -> Boolean
--R qelt : (%,Integer) -> S               rest : % -> %
--R sample : () -> %                      second : % -> S
--R tail : % -> %                         third : % -> S
--R value : % -> S
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R child? : (%,%) -> Boolean if S has SETCAT
--R coerce : % -> OutputForm if S has SETCAT
--R concat! : (%,S) -> % if $ has shallowlyMutable
--R concat! : (%,%) -> % if $ has shallowlyMutable
--R convert : % -> InputForm if S has KONVERT INFORM
```

```
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R cycleLength : % -> NonNegativeInteger
--R cycleSplit! : % -> % if $ has shallowlyMutable
--R delete : (%,UniversalSegment Integer) -> %
--R ?.? : (%,UniversalSegment Integer) -> %
--R entry? : (S,%) -> Boolean if $ has finiteAggregate and S has SETCAT
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R fill! : (%,S) -> % if $ has shallowlyMutable
--R find : ((S -> Boolean),%) -> Union(S,"failed")
--R first : (%,NonNegativeInteger) -> %
--R hash : % -> SingleInteger if S has SETCAT
--R last : (%,NonNegativeInteger) -> %
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R maxIndex : % -> Integer if Integer has ORDSET
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R minIndex : % -> Integer if Integer has ORDSET
--R more? : (%,NonNegativeInteger) -> Boolean
--R node? : (%,%) -> Boolean if S has SETCAT
--R parts : % -> List S if $ has finiteAggregate
--R qsetelt! : (%,Integer,S) -> S if $ has shallowlyMutable
--R reduce : (((S,S) -> S),%,S,S) -> S if S has SETCAT and $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%) -> S if $ has finiteAggregate
--R remove : (S,%) -> % if S has SETCAT and $ has finiteAggregate
--R remove : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R removeDuplicates : % -> % if S has SETCAT and $ has finiteAggregate
--R rest : (%,NonNegativeInteger) -> %
--R select : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R setchildren! : (%,List %) -> % if $ has shallowlyMutable
--R setelt : (%,Integer,S) -> S if $ has shallowlyMutable
--R setelt : (%,UniversalSegment Integer,S) -> S if $ has shallowlyMutable
--R setelt : (%,last,S) -> S if $ has shallowlyMutable
--R setelt : (%,rest,%) -> % if $ has shallowlyMutable
--R setelt : (%,first,S) -> S if $ has shallowlyMutable
--R setelt : (%,value,S) -> S if $ has shallowlyMutable
--R setfirst! : (%,S) -> S if $ has shallowlyMutable
--R setlast! : (%,S) -> S if $ has shallowlyMutable
--R setrest! : (%,%) -> % if $ has shallowlyMutable
--R setvalue! : (%,S) -> S if $ has shallowlyMutable
--R size? : (%,NonNegativeInteger) -> Boolean
--R split! : (%,Integer) -> % if $ has shallowlyMutable
--R swap! : (%,Integer,Integer) -> Void if $ has shallowlyMutable
```

```
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— **StreamAggregate.help** —

```
=====================================================================
StreamAggregate examples
=====================================================================

See Also:
o )show StreamAggregate
```

———-

See:

⇒ “LazyStreamAggregate” (LZSTAGG) 8.0.112 on page 637
⇐ “LinearAggregate” (LNAGG) 6.0.87 on page 404
⇐ “UnaryRecursiveAggregate” (URAGG) 6.0.96 on page 506

Exports:

any?	children	child?	coerce	concat
concat!	construct	convert	copy	count
cycleEntry	cycleLength	cycleSplit!	cycleTail	cyclic?
delete	distance	elt	empty	empty?
entries	entry?	eq?	eval	every?
explicitlyFinite?	fill!	find	first	hash
index?	indices	insert	last	latex
leaf?	leaves	less?	map	map!
maxIndex	member?	members	minIndex	more?
new	nodes	node?	parts	possiblyInfinite?
qelt	qsetelt!	reduce	remove	removeDuplicates
rest	sample	second	select	setchildren!
setelt	setfirst!	setlast!	setrest!	setvalue!
size?	split!	swap!	tail	third
value	#?	?=?	?.?	?.first
?.last	?.rest	?.value	?~=?	

Attributes exported:

- **nil**

Attributes Used:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.

These are implemented by this category:

```
concat : (%,%) -> %
concat : List % -> %
concat! : (%,%) -> % if $ has shallowlyMutable
fill! : (%,S) -> % if $ has shallowlyMutable
first : (%,NonNegativeInteger) -> %
explicitlyFinite? : % -> Boolean
map! : ((S -> S),%) -> % if $ has shallowlyMutable
possiblyInfinite? : % -> Boolean
setelt : (%,Integer,S) -> S if $ has shallowlyMutable
setelt : (%,UniversalSegment Integer,S) -> S
         if $ has shallowlyMutable
?.? : (%,Integer) -> S
?.? : (%,UniversalSegment Integer) -> %
```

These exports come from (p506) UnaryRecursiveAggregate(S:Type):

```
any? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
children : % -> List %
child? : (%,%) -> Boolean if S has SETCAT
coerce : % -> OutputForm if S has SETCAT
concat : (S,%) -> %
concat : (%,S) -> %
concat! : (%,S) -> % if $ has shallowlyMutable
copy : % -> %
count : (S,%) -> NonNegativeInteger
         if S has SETCAT
         and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
cycleEntry : % -> %
cycleLength : % -> NonNegativeInteger
cycleSplit! : % -> % if $ has shallowlyMutable
cycleTail : % -> %
cyclic? : % -> Boolean
distance : (%,%) -> Integer
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> %
         if S has EVALAB S
```

```
         and S has SETCAT
eval : (%,S,S) -> %
         if S has EVALAB S
         and S has SETCAT
eval : (%,Equation S) -> %
         if S has EVALAB S
         and S has SETCAT
eval : (%,List Equation S) -> %
         if S has EVALAB S
         and S has SETCAT
every? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
first : % -> S
hash : % -> SingleInteger if S has SETCAT
latex : % -> String if S has SETCAT
last : % -> S
last : (%,NonNegativeInteger) -> %
leaf? : % -> Boolean
leaves : % -> List S
less? : (%,NonNegativeInteger) -> Boolean
map : ((S -> S),%) -> %
member? : (S,%) -> Boolean
         if S has SETCAT
         and $ has finiteAggregate
members : % -> List S if $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
nodes : % -> List %
node? : (%,%) -> Boolean if S has SETCAT
parts : % -> List S if $ has finiteAggregate
rest : % -> %
rest : (%,NonNegativeInteger) -> %
sample : () -> %
second : % -> S
setchildren! : (%,List %) -> % if $ has shallowlyMutable
setelt : (%,first,S) -> S if $ has shallowlyMutable
setelt : (%,last,S) -> S if $ has shallowlyMutable
setelt : (%,rest,%) -> % if $ has shallowlyMutable
setelt : (%,value,S) -> S if $ has shallowlyMutable
setfirst! : (%,S) -> S if $ has shallowlyMutable
setlast! : (%,S) -> S if $ has shallowlyMutable
setrest! : (%,%) -> % if $ has shallowlyMutable
setvalue! : (%,S) -> S if $ has shallowlyMutable
size? : (%,NonNegativeInteger) -> Boolean
split! : (%,Integer) -> % if $ has shallowlyMutable
tail : % -> %
third : % -> S
value : % -> S
#? : % -> NonNegativeInteger if $ has finiteAggregate
?=? : (%,%) -> Boolean if S has SETCAT
?.last : (%,last) -> S
```

```
?.rest : (%,rest) -> %
?.first : (%,first) -> S
?~=? : (%,%) -> Boolean if S has SETCAT
?.value : (%,value) -> S
```

These exports come from (p404) LinearAggregate(S:Type):

```
construct : List S -> %
convert : % -> InputForm if S has KONVERT INFORM
delete : (%,Integer) -> %
delete : (%,UniversalSegment Integer) -> %
elt : (%,Integer,S) -> S
entry? : (S,%) -> Boolean
         if $ has finiteAggregate
         and S has SETCAT
entries : % -> List S
find : ((S -> Boolean),%) -> Union(S,"failed")
index? : (Integer,%) -> Boolean
indices : % -> List Integer
insert : (S,%,Integer) -> %
insert : (%,%,Integer) -> %
maxIndex : % -> Integer if Integer has ORDSET
map : (((S,S) -> S),%,%) -> %
minIndex : % -> Integer if Integer has ORDSET
new : (NonNegativeInteger,S) -> %
qelt : (%,Integer) -> S
qsetelt! : (%,Integer,S) -> S
         if $ has shallowlyMutable
reduce : (((S,S) -> S),%,S,S) -> S
         if S has SETCAT
         and $ has finiteAggregate
reduce : (((S,S) -> S),%,S) -> S
         if $ has finiteAggregate
reduce : (((S,S) -> S),%) -> S
         if $ has finiteAggregate
remove : (S,%) -> %
         if S has SETCAT
         and $ has finiteAggregate
remove : ((S -> Boolean),%) -> %
         if $ has finiteAggregate
removeDuplicates : % -> %
         if S has SETCAT
         and $ has finiteAggregate
select : ((S -> Boolean),%) -> %
         if $ has finiteAggregate
swap! : (%,Integer,Integer) -> Void
         if $ has shallowlyMutable
```

— category STAGG StreamAggregate —

```
)abbrev category STAGG StreamAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A stream aggregate is a linear aggregate which possibly has an infinite
++ number of elements. A basic domain constructor which builds stream
++ aggregates is \spadtype{Stream}. From streams, a number of infinite
++ structures such power series can be built. A stream aggregate may
++ also be infinite since it may be cyclic.
++ For example, see \spadtype{DecimalExpansion}.

StreamAggregate(S:Type): Category ==
   Join(UnaryRecursiveAggregate S, LinearAggregate S) with
      explicitlyFinite?: % -> Boolean
        ++ explicitlyFinite?(s) tests if the stream has a finite
        ++ number of elements, and false otherwise.
        ++ Note that for many datatypes,
        ++ \axiom{explicitlyFinite?(s) = not possiblyInfinite?(s)}.
      possiblyInfinite?: % -> Boolean
        ++ possiblyInfinite?(s) tests if the stream s could possibly
        ++ have an infinite number of elements.
        ++ Note that for many datatypes,
        ++ \axiom{possiblyInfinite?(s) = not explictlyFinite?(s)}.
 add
   c2: (%, %) -> S

   explicitlyFinite? x == not cyclic? x
   possiblyInfinite? x == cyclic? x
   first(x, n)               == construct [c2(x, x := rest x) for i in 1..n]

   c2(x, r) ==
     empty? x => error "Index out of range"
     first x

   elt(x:%, i:Integer) ==
     i := i - minIndex x
     (i < 0) or empty?(x := rest(x, i::NonNegativeInteger)) => _
         error "index out of range"
     first x

   elt(x:%, i:UniversalSegment(Integer)) ==
     l := lo(i) - minIndex x
     l < 0 => error "index out of range"
```

```
    not hasHi i => copy(rest(x, l::NonNegativeInteger))
    (h := hi(i) - minIndex x) < l => empty()
    first(rest(x, l::NonNegativeInteger), (h - l + 1)::NonNegativeInteger)

  if % has shallowlyMutable then
    concat(x:%, y:%) == concat_!(copy x, y)

    concat l ==
      empty? l => empty()
      concat_!(copy first l, concat rest l)

    map_!(f, l) ==
      y := l
      while not empty? l repeat
        setfirst_!(l, f first l)
        l := rest l
      y

    fill_!(x, s) ==
      y := x
      while not empty? y repeat (setfirst_!(y, s); y := rest y)
      x

    setelt(x:%, i:Integer, s:S) ==
      i := i - minIndex x
      (i < 0) or empty?(x := rest(x,i::NonNegativeInteger)) => _
          error "index out of range"
      setfirst_!(x, s)

    setelt(x:%, i:UniversalSegment(Integer), s:S) ==
      (l := lo(i) - minIndex x) < 0 => error "index out of range"
      h := if hasHi i then hi(i) - minIndex x else maxIndex x
      h < l => s
      y := rest(x, l::NonNegativeInteger)
      z := rest(y, (h - l + 1)::NonNegativeInteger)
      while not eq?(y, z) repeat (setfirst_!(y, s); y := rest y)
      s

    concat_!(x:%, y:%) ==
      empty? x => y
      setrest_!(tail x, y)
      x
```

———-

— STAGG.dotabb —

```
"STAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=STAGG"];
```

```
"STAGG" -> "RCAGG"
"STAGG" -> "LNAGG"
```

———-

— STAGG.dotfull —

```
"StreamAggregate(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=STAGG"];
"StreamAggregate(a:Type)" -> "UnaryRecursiveAggregate(a:Type)"
"StreamAggregate(a:Type)" -> "LinearAggregate(a:Type)"
```

———-

— STAGG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"StreamAggregate(a:Type)" [color=lightblue];
"StreamAggregate(a:Type)" -> "UnaryRecursiveAggregate(a:Type)"
"StreamAggregate(a:Type)" -> "LinearAggregate(a:Type)"

"UnaryRecursiveAggregate(a:Type)" [color=lightblue];
"UnaryRecursiveAggregate(a:Type)" -> "RecursiveAggregate(a:Type)"

"RecursiveAggregate(a:Type)" [color=lightblue];
"RecursiveAggregate(a:Type)" -> "HomogeneousAggregate(a:Type)"

"LinearAggregate(a:Type)" [color=lightblue];
"LinearAggregate(a:Type)" -> "IndexedAggregate(b:Integer,a:Type)"
"LinearAggregate(a:Type)" -> "Collection(a:Type)"

"IndexedAggregate(b:Integer,a:Type)" [color=seagreen];
"IndexedAggregate(b:Integer,a:Type)" ->
    "IndexedAggregate(a:SetCategory,b:Type)"

"IndexedAggregate(a:SetCategory,b:Type)" [color=lightblue];
"IndexedAggregate(a:SetCategory,b:Type)" ->
    "HomogeneousAggregate(a:Type)"

"Collection(a:Type)" [color=lightblue];
"Collection(a:Type)" -> "HomogeneousAggregate(a:Type)"
```

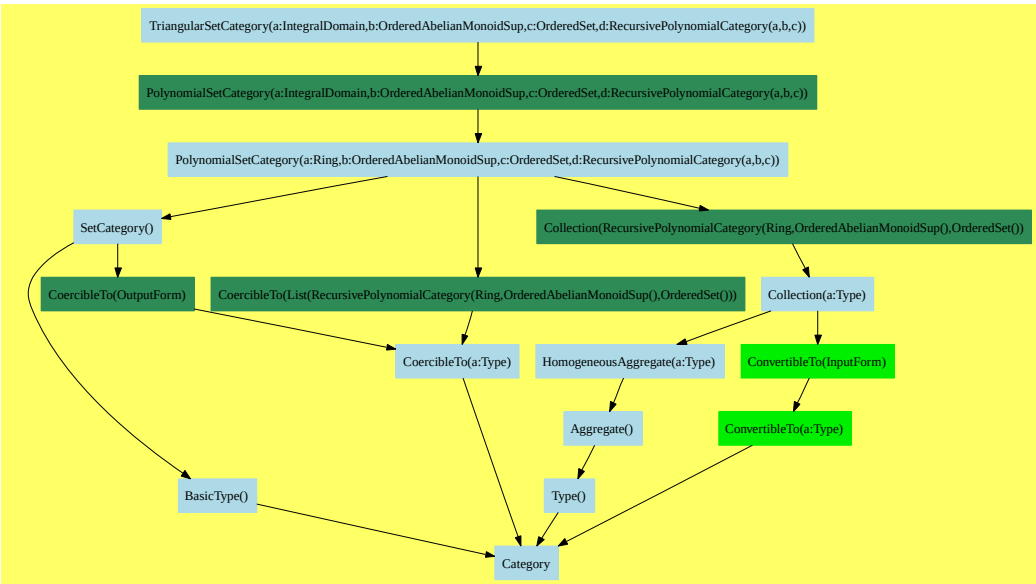
```
"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "..."

"..." [color=lightblue];

}
```

———-

7.0.108 TriangularSetCategory (TSETCAT)



— TriangularSetCategory.input —

```
)set break resume
)sys rm -f TriangularSetCategory.output
)spool TriangularSetCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show TriangularSetCategory
--R TriangularSetCategory(R: IntegralDomain,E: OrderedAbelianMonoidSup,V: OrderedSet,P: RecursivePolynomial
--R Abbreviation for TriangularSetCategory is TSETCAT
```

```
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for TSETCAT
--R
--R------------------------------ Operations ------------------------------
--R ?=? : (%,%) -> Boolean                algebraic? : (V,%) -> Boolean
--R algebraicVariables : % -> List V      coerce : % -> List P
--R coerce : % -> OutputForm              collect : (%,V) -> %
--R collectQuasiMonic : % -> %            collectUnder : (%,V) -> %
--R collectUpper : (%,V) -> %             construct : List P -> %
--R copy : % -> %                         degree : % -> NonNegativeInteger
--R empty : () -> %                       empty? : % -> Boolean
--R eq? : (%,%) -> Boolean                extend : (%,P) -> %
--R first : % -> Union(P,"failed")        hash : % -> SingleInteger
--R headReduce : (P,%) -> P               headReduced? : % -> Boolean
--R headReduced? : (P,%) -> Boolean       infRittWu? : (%,%) -> Boolean
--R initiallyReduce : (P,%) -> P          initiallyReduced? : % -> Boolean
--R initials : % -> List P                last : % -> Union(P,"failed")
--R latex : % -> String                   mainVariable? : (V,%) -> Boolean
--R mainVariables : % -> List V           map : ((P -> P),%) -> %
--R mvar : % -> V                         normalized? : % -> Boolean
--R normalized? : (P,%) -> Boolean        reduceByQuasiMonic : (P,%) -> P
--R removeZero : (P,%) -> P               rest : % -> Union(%,"failed")
--R retract : List P -> %                 sample : () -> %
--R stronglyReduce : (P,%) -> P           stronglyReduced? : % -> Boolean
--R trivialIdeal? : % -> Boolean          variables : % -> List V
--R zeroSetSplit : List P -> List %       ?~=? : (%,%) -> Boolean
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R any? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
--R autoReduced? : (%,((P,List P) -> Boolean)) -> Boolean
--R basicSet : (List P,(P -> Boolean),((P,P) -> Boolean)) -> Union(Record(bas: %,top: List P),"f
--R basicSet : (List P,((P,P) -> Boolean)) -> Union(Record(bas: %,top: List P),"failed")
--R coHeight : % -> NonNegativeInteger if V has FINITE
--R convert : % -> InputForm if P has KONVERT INFORM
--R count : ((P -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R count : (P,%) -> NonNegativeInteger if P has SETCAT and $ has finiteAggregate
--R eval : (%,List Equation P) -> % if P has EVALAB P and P has SETCAT
--R eval : (%,Equation P) -> % if P has EVALAB P and P has SETCAT
--R eval : (%,P,P) -> % if P has EVALAB P and P has SETCAT
--R eval : (%,List P,List P) -> % if P has EVALAB P and P has SETCAT
--R every? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
--R extendIfCan : (%,P) -> Union(%,"failed")
--R find : ((P -> Boolean),%) -> Union(P,"failed")
--R headRemainder : (P,%) -> Record(num: P,den: R) if R has INTDOM
--R initiallyReduced? : (P,%) -> Boolean
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((P -> P),%) -> % if $ has shallowlyMutable
--R member? : (P,%) -> Boolean if P has SETCAT and $ has finiteAggregate
--R members : % -> List P if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List P if $ has finiteAggregate
```

```
--R quasiComponent : % -> Record(close: List P,open: List P)
--R reduce : (P,%,((P,P) -> P),((P,P) -> Boolean)) -> P
--R reduce : (((P,P) -> P),%) -> P if $ has finiteAggregate
--R reduce : (((P,P) -> P),%,P) -> P if $ has finiteAggregate
--R reduce : (((P,P) -> P),%,P,P) -> P if P has SETCAT and $ has finiteAggregate
--R reduced? : (P,%,((P,P) -> Boolean)) -> Boolean
--R remainder : (P,%) -> Record(rnum: R,polnum: P,den: R) if R has INTDOM
--R remove : ((P -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (P,%) -> % if P has SETCAT and $ has finiteAggregate
--R removeDuplicates : % -> % if P has SETCAT and $ has finiteAggregate
--R retractIfCan : List P -> Union(%,"failed")
--R rewriteIdealWithHeadRemainder : (List P,%) -> List P if R has INTDOM
--R rewriteIdealWithRemainder : (List P,%) -> List P if R has INTDOM
--R rewriteSetWithReduction : (List P,%,((P,P) -> P),((P,P) -> Boolean)) -> List P
--R roughBase? : % -> Boolean if R has INTDOM
--R roughEqualIdeals? : (%,%) -> Boolean if R has INTDOM
--R roughSubIdeal? : (%,%) -> Boolean if R has INTDOM
--R roughUnitIdeal? : % -> Boolean if R has INTDOM
--R select : (%,V) -> Union(P,"failed")
--R select : ((P -> Boolean),%) -> % if $ has finiteAggregate
--R size? : (%,NonNegativeInteger) -> Boolean
--R sort : (%,V) -> Record(under: %,floor: %,upper: %)
--R stronglyReduced? : (P,%) -> Boolean
--R triangular? : % -> Boolean if R has INTDOM
--R zeroSetSplitIntoTriangularSystems : List P -> List Record(close: %,open: List P)
--R
--E 1

)spool
)lisp (bye)
```

———-

— TriangularSetCategory.help —

```
=====================================================================
TriangularSetCategory examples
=====================================================================

See Also:
o )show TriangularSetCategory
```

———-

See:

⇒ “RegularTriangularSetCategory” (RSETCAT) 8.0.119 on page 700
⇐ “PolynomialSetCategory” (PSETCAT) 6.0.91 on page 465

Exports:

algebraic?	algebraicVariables
any?	autoReduced?
basicSet	coerce
coHeight	collect
collectQuasiMonic	collectUnder
collectUpper	construct
convert	copy
count	degree
empty	empty?
eq?	eval
every?	extend
extendIfCan	find
first	hash
headReduce	headReduced?
headRemainder	infRittWu?
initiallyReduce	initiallyReduced?
initials	last
latex	less?
mainVariable?	mainVariables
map	map!
member?	members
more?	mvar
normalized?	parts
quasiComponent	reduce
reduced?	reduceByQuasiMonic
remainder	remove
removeDuplicates	removeZero
rest	retract
retractIfCan	rewriteIdealWithHeadRemainder
rewriteIdealWithRemainder	rewriteSetWithReduction
roughBase?	roughEqualIdeals?
roughSubIdeal?	roughUnitIdeal?
sample	select
size?	sort
stronglyReduce	stronglyReduced?
triangular?	trivialIdeal?
variables	zeroSetSplit
zeroSetSplitIntoTriangularSystems	#?
?=?	?~=?

Attributes Exported:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to

the shallowlyMutable proper.

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **nil**

These are directly exported but not implemented:

```
extendIfCan : (%,P) -> Union(%,"failed")
zeroSetSplit : List P -> List %
zeroSetSplitIntoTriangularSystems :
  List P -> List Record(close: %,open: List P)
```

These are implemented by this category:

```
algebraic? : (V,%) -> Boolean
algebraicVariables : % -> List V
autoReduced? : (%,((P,List P) -> Boolean)) -> Boolean
basicSet :
  (List P,(P -> Boolean),((P,P) -> Boolean)) ->
    Union(Record(bas: %,top: List P),"failed")
basicSet : (List P,((P,P) -> Boolean)) ->
  Union(Record(bas: %,top: List P),"failed")
coerce : % -> List P
coHeight : % -> NonNegativeInteger if V has FINITE
collectQuasiMonic : % -> %
collectUnder : (%,V) -> %
collectUpper : (%,V) -> %
construct : List P -> %
convert : % -> InputForm if P has KONVERT INFORM
degree : % -> NonNegativeInteger
extend : (%,P) -> %
first : % -> Union(P,"failed")
headReduce : (P,%) -> P
headReduced? : % -> Boolean
headReduced? : (P,%) -> Boolean
infRittWu? : (%,%) -> Boolean
initiallyReduce : (P,%) -> P
initiallyReduced? : % -> Boolean
initiallyReduced? : (P,%) -> Boolean
initials : % -> List P
last : % -> Union(P,"failed")
mvar : % -> V
normalized? : % -> Boolean
normalized? : (P,%) -> Boolean
quasiComponent : % -> Record(close: List P,open: List P)
reduce : (P,%,((P,P) -> P),((P,P) -> Boolean)) -> P
reduceByQuasiMonic : (P,%) -> P
reduced? : (P,%,((P,P) -> Boolean)) -> Boolean
removeZero : (P,%) -> P
```

```
rest : % -> Union(%,"failed")
retractIfCan : List P -> Union(%,"failed")
rewriteSetWithReduction :
   (List P,%,((P,P) -> P),((P,P) -> Boolean)) -> List P
select : (%,V) -> Union(P,"failed")
stronglyReduce : (P,%) -> P
stronglyReduced? : % -> Boolean
stronglyReduced? : (P,%) -> Boolean
?=? : (%,%) -> Boolean
```

These exports come from (p465) PolynomialSetCategory(R,E,V,P) where R:IntegralDomain, E:OrderedAbelianMonoidSup, V:OrderedSet, P:RecursivePolynomialCategory(R,E,V)):

```
any? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
coerce : % -> OutputForm
collect : (%,V) -> %
copy : % -> %
count : ((P -> Boolean),%) -> NonNegativeInteger
  if $ has finiteAggregate
count : (P,%) -> NonNegativeInteger
  if P has SETCAT and $ has finiteAggregate
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List Equation P) -> % if P has EVALAB P and P has SETCAT
eval : (%,Equation P) -> % if P has EVALAB P and P has SETCAT
eval : (%,P,P) -> % if P has EVALAB P and P has SETCAT
eval : (%,List P,List P) -> % if P has EVALAB P and P has SETCAT
every? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
find : ((P -> Boolean),%) -> Union(P,"failed")
hash : % -> SingleInteger
headRemainder : (P,%) -> Record(num: P,den: R) if R has INTDOM
latex : % -> String
less? : (%,NonNegativeInteger) -> Boolean
mainVariable? : (V,%) -> Boolean
mainVariables : % -> List V
map : ((P -> P),%) -> %
map! : ((P -> P),%) -> % if $ has shallowlyMutable
member? : (P,%) -> Boolean
  if P has SETCAT and $ has finiteAggregate
members : % -> List P if $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
parts : % -> List P if $ has finiteAggregate
reduce : (((P,P) -> P),%) -> P if $ has finiteAggregate
reduce : (((P,P) -> P),%,P) -> P if $ has finiteAggregate
reduce : (((P,P) -> P),%,P,P) -> P
  if P has SETCAT
  and $ has finiteAggregate
remainder : (P,%) -> Record(rnum: R,polnum: P,den: R)
```

```
  if R has INTDOM
 remove : ((P -> Boolean),%) -> % if $ has finiteAggregate
 remove : (P,%) -> % if P has SETCAT and $ has finiteAggregate
 removeDuplicates : % -> %
  if P has SETCAT
  and $ has finiteAggregate
 retract : List P -> %
 rewriteIdealWithHeadRemainder : (List P,%) -> List P
  if R has INTDOM
 rewriteIdealWithRemainder : (List P,%) -> List P
  if R has INTDOM
 roughBase? : % -> Boolean if R has INTDOM
 roughEqualIdeals? : (%,%) -> Boolean if R has INTDOM
 roughSubIdeal? : (%,%) -> Boolean if R has INTDOM
 roughUnitIdeal? : % -> Boolean if R has INTDOM
 sample : () -> %
 select : ((P -> Boolean),%) -> % if $ has finiteAggregate
 size? : (%,NonNegativeInteger) -> Boolean
 sort : (%,V) -> Record(under: %,floor: %,upper: %)
 triangular? : % -> Boolean if R has INTDOM
 trivialIdeal? : % -> Boolean
 variables : % -> List V
 #? : % -> NonNegativeInteger if $ has finiteAggregate
 ?~=? : (%,%) -> Boolean
```

— category TSETCAT TriangularSetCategory —

```
)abbrev category TSETCAT TriangularSetCategory
++ Author: Marc Moreno Maza (marc@nag.co.uk)
++ Date Created: 04/26/1994
++ Date Last Updated: 12/15/1998
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords: polynomial, multivariate, ordered variables set
++ References :
++  [1] P. AUBRY, D. LAZARD and M. MORENO MAZA "On the Theories
++      of Triangular Sets" Journal of Symbol. Comp. (to appear)
++ Description:
++ The category of triangular sets of multivariate polynomials
++ with coefficients in an integral domain.
++ Let \axiom{R} be an integral domain and \axiom{V} a finite ordered set of
++ variables, say \axiom{X1 < X2 < ... < Xn}.
++ A set \axiom{S} of polynomials in \axiom{R[X1,X2,...,Xn]} is triangular
++ if no elements of \axiom{S} lies in \axiom{R}, and if two distinct
++ elements of \axiom{S} have distinct main variables.
++ Note that the empty set is a triangular set. A triangular set is not
++ necessarily a (lexicographical) Groebner basis and the notion of
```

```
++ reduction related to triangular sets is based on the recursive view
++ of polynomials. We recall this notion here and refer to [1] for more
++ details.
++ A polynomial \axiom{P} is reduced w.r.t a non-constant polynomial
++ \axiom{Q} if the degree of \axiom{P} in the main variable of \axiom{Q}
++ is less than the main degree of \axiom{Q}.
++ A polynomial \axiom{P} is reduced w.r.t a triangular set \axiom{T}
++ if it is reduced w.r.t. every polynomial of \axiom{T}.

TriangularSetCategory(R:IntegralDomain,E:OrderedAbelianMonoidSup,_
 V:OrderedSet,P:RecursivePolynomialCategory(R,E,V)):
         Category ==
   PolynomialSetCategory(R,E,V,P) with
     finiteAggregate
     shallowlyMutable

     infRittWu? : ($,$) -> Boolean
       ++ \axiom{infRittWu?(ts1,ts2)} returns true iff \axiom{ts2} has
       ++ higher rank than \axiom{ts1} in Wu Wen Tsun sense.
     basicSet : (List P,((P,P)->Boolean)) -> _
      Union(Record(bas:$,top:List P),"failed")
       ++ \axiom{basicSet(ps,redOp?)} returns \axiom{[bs,ts]} where
       ++ \axiom{concat(bs,ts)} is \axiom{ps} and \axiom{bs}
       ++ is a basic set in Wu Wen Tsun sense of \axiom{ps} w.r.t
       ++ the reduction-test \axiom{redOp?}, if no non-zero constant
       ++ polynomial lie in \axiom{ps}, otherwise \axiom{"failed"} is returned.
     basicSet : (List P,(P->Boolean),((P,P)->Boolean)) -> _
      Union(Record(bas:$,top:List P),"failed")
       ++ \axiom{basicSet(ps,pred?,redOp?)} returns the same as
       ++ \axiom{basicSet(qs,redOp?)}
       ++ where \axiom{qs} consists of the polynomials of \axiom{ps}
       ++ satisfying property \axiom{pred?}.
     initials : $ -> List P
       ++ \axiom{initials(ts)} returns the list of the non-constant initials
       ++ of the members of \axiom{ts}.
     degree : $ -> NonNegativeInteger
       ++ \axiom{degree(ts)} returns the product of main degrees of the
       ++ members of \axiom{ts}.
     quasiComponent : $ -> Record(close:List P,open:List P)
       ++ \axiom{quasiComponent(ts)} returns \axiom{[lp,lq]} where \axiom{lp}
       ++ is the list of the members of \axiom{ts} and \axiom{lq}is
       ++ \axiom{initials(ts)}.
     normalized? : (P,$) -> Boolean
       ++ \axiom{normalized?(p,ts)} returns true iff \axiom{p} and all
       ++ its iterated initials have degree zero w.r.t. the main variables
       ++ of the polynomials of \axiom{ts}
     normalized? : $  -> Boolean
       ++ \axiom{normalized?(ts)} returns true iff for every axiom{p} in
       ++ \axiom{ts} we have \axiom{normalized?(p,us)} where \axiom{us}
       ++ is \axiom{collectUnder(ts,mvar(p))}.
```

```
reduced? : (P,$,((P,P) -> Boolean)) -> Boolean
  ++ \axiom{reduced?(p,ts,redOp?)} returns true iff \axiom{p} is reduced
  ++ w.r.t.in the sense of the operation \axiom{redOp?}, that is if for
  ++ every \axiom{t} in \axiom{ts} \axiom{redOp?(p,t)} holds.
stronglyReduced? : (P,$) -> Boolean
  ++ \axiom{stronglyReduced?(p,ts)} returns true iff \axiom{p}
  ++ is reduced w.r.t. \axiom{ts}.
headReduced? : (P,$) -> Boolean
  ++ \axiom{headReduced?(p,ts)} returns true iff the head of \axiom{p} is
  ++ reduced w.r.t. \axiom{ts}.
initiallyReduced? : (P,$) -> Boolean
  ++ \axiom{initiallyReduced?(p,ts)} returns true iff \axiom{p} and all
  ++ its iterated initials are reduced w.r.t. to the elements of
  ++ \axiom{ts} with the same main variable.
autoReduced? : ($,((P,List(P)) -> Boolean)) -> Boolean
  ++ \axiom{autoReduced?(ts,redOp?)} returns true iff every element of
  ++ \axiom{ts} is reduced w.r.t to every other in the sense of
  ++ \axiom{redOp?}
stronglyReduced? : $ -> Boolean
  ++ \axiom{stronglyReduced?(ts)} returns true iff every element of
  ++ \axiom{ts} is reduced w.r.t to any other element of \axiom{ts}.
headReduced? : $ -> Boolean
  ++ headReduced?(ts) returns true iff the head of every element of
  ++ \axiom{ts} is reduced w.r.t to any other element of \axiom{ts}.
initiallyReduced? : $ -> Boolean
  ++ initiallyReduced?(ts) returns true iff for every element \axiom{p}
  ++ of \axiom{ts}. \axiom{p} and all its iterated initials are reduced
  ++ w.r.t. to the other elements of \axiom{ts} with the same main
  ++ variable.
reduce : (P,$,((P,P) -> P),((P,P) -> Boolean) ) -> P
  ++ \axiom{reduce(p,ts,redOp,redOp?)} returns a polynomial \axiom{r}
  ++ such that \axiom{redOp?(r,p)} holds for every \axiom{p} of
  ++ \axiom{ts} and there exists some product \axiom{h} of the initials
  ++ of the members of \axiom{ts} such that \axiom{h*p - r} lies in the
  ++ ideal generated by \axiom{ts}. The operation \axiom{redOp} must
  ++ satisfy the following conditions. For every \axiom{p} and \axiom{q}
  ++ we have  \axiom{redOp?(redOp(p,q),q)} and there exists an integer
  ++ \axiom{e} and a polynomial \axiom{f} such that
  ++ \axiom{init(q)^e*p = f*q + redOp(p,q)}.
rewriteSetWithReduction : (List P,$,((P,P) -> P),((P,P) -> Boolean) ) ->_
 List P
  ++ \axiom{rewriteSetWithReduction(lp,ts,redOp,redOp?)} returns a list
  ++ \axiom{lq} of polynomials such that
  ++ \axiom{[reduce(p,ts,redOp,redOp?) for p in lp]} and \axiom{lp}
  ++ have the same zeros inside the regular zero set of \axiom{ts}.
  ++ Moreover, for every polynomial \axiom{q} in \axiom{lq} and every
  ++ polynomial \axiom{t} in \axiom{ts}
  ++ \axiom{redOp?(q,t)} holds and there exists a polynomial \axiom{p}
  ++ in the ideal generated by \axiom{lp} and a product \axiom{h} of
  ++ \axiom{initials(ts)} such that \axiom{h*p - r} lies in the ideal
```

```
  ++ generated by \axiom{ts}.
  ++ The operation \axiom{redOp} must satisfy the following conditions.
  ++ For every \axiom{p} and \axiom{q} we have
  ++ \axiom{redOp?(redOp(p,q),q)}
  ++ and there exists an integer \axiom{e} and a polynomial \axiom{f}
  ++ such that \axiom{init(q)^e*p = f*q + redOp(p,q)}.
stronglyReduce : (P,$) -> P
  ++ \axiom{stronglyReduce(p,ts)} returns a polynomial \axiom{r} such that
  ++ \axiom{stronglyReduced?(r,ts)} holds and there exists some product
  ++ \axiom{h} of \axiom{initials(ts)}
  ++ such that \axiom{h*p - r} lies in the ideal generated by \axiom{ts}.
headReduce : (P,$) -> P
  ++ \axiom{headReduce(p,ts)} returns a polynomial \axiom{r}  such that
  ++ \axiom{headReduce?(r,ts)} holds and there exists some product
  ++ \axiom{h} of \axiom{initials(ts)} such that \axiom{h*p - r} lies
  ++ in the ideal generated by \axiom{ts}.
initiallyReduce : (P,$) -> P
  ++ \axiom{initiallyReduce(p,ts)} returns a polynomial \axiom{r}
  ++ such that  \axiom{initiallyReduced?(r,ts)}
  ++ holds and there exists some product \axiom{h} of \axiom{initials(ts)}
  ++ such that \axiom{h*p - r} lies in the ideal generated by \axiom{ts}.
removeZero: (P, $) -> P
  ++ \axiom{removeZero(p,ts)} returns \axiom{0} if \axiom{p} reduces
  ++ to \axiom{0} by pseudo-division w.r.t \axiom{ts} otherwise
  ++ returns a polynomial \axiom{q} computed from \axiom{p}
  ++ by removing any coefficient in \axiom{p} reducing to \axiom{0}.
collectQuasiMonic: $ -> $
  ++ \axiom{collectQuasiMonic(ts)} returns the subset of \axiom{ts}
  ++ consisting of the polynomials with initial in \axiom{R}.
reduceByQuasiMonic: (P, $) -> P
  ++ \axiom{reduceByQuasiMonic(p,ts)} returns the same as
  ++ \axiom{remainder(p,collectQuasiMonic(ts)).polnum}.
zeroSetSplit : List P -> List $
  ++ \axiom{zeroSetSplit(lp)} returns a list \axiom{lts} of triangular
  ++ sets such that the zero set of \axiom{lp} is the union of the
  ++ closures of the regular zero sets  of the members of \axiom{lts}.
zeroSetSplitIntoTriangularSystems : List P -> _
 List Record(close:$,open:List P)
  ++ \axiom{zeroSetSplitIntoTriangularSystems(lp)} returns a list of
  ++ triangular systems \axiom{[[ts1,qs1],...,[tsn,qsn]]} such that the
  ++ zero set of \axiom{lp} is the union of the closures of the
  ++ \axiom{W_i} where \axiom{W_i} consists of the zeros of \axiom{ts}
  ++ which do not cancel any polynomial in \axiom{qsi}.
first : $ -> Union(P,"failed")
  ++ \axiom{first(ts)} returns the polynomial of \axiom{ts} with
  ++ greatest main variable if \axiom{ts} is not empty, otherwise
  ++ returns \axiom{"failed"}.
last : $ -> Union(P,"failed")
  ++ \axiom{last(ts)} returns the polynomial of \axiom{ts} with
  ++ smallest main variable if \axiom{ts} is not empty, otherwise
```

```
      ++ returns \axiom{"failed"}.
    rest : $ -> Union($,"failed")
      ++ \axiom{rest(ts)} returns the polynomials of \axiom{ts} with smaller
      ++ main variable than \axiom{mvar(ts)} if \axiom{ts} is not empty,
      ++ otherwise returns "failed"
    algebraicVariables : $ -> List(V)
      ++ \axiom{algebraicVariables(ts)} returns the decreasingly sorted
      ++ list of the main variables of the polynomials of \axiom{ts}.
    algebraic? : (V,$) -> Boolean
      ++ \axiom{algebraic?(v,ts)} returns true iff \axiom{v} is the
      ++ main variable of some polynomial in \axiom{ts}.
    select : ($,V) -> Union(P,"failed")
      ++ \axiom{select(ts,v)} returns the polynomial of \axiom{ts} with
      ++ \axiom{v} as main variable, if any.
    extendIfCan : ($,P) -> Union($,"failed")
      ++ \axiom{extendIfCan(ts,p)} returns a triangular set which encodes
      ++ the simple extension by \axiom{p} of the extension of the base
      ++ field defined by \axiom{ts}, according
      ++ to the properties of triangular sets of the current domain.
      ++ If the required properties do not hold then "failed" is returned.
      ++ This operation encodes in some sense the properties of the
      ++ triangular sets of the current category. Is is used to implement
      ++ the \axiom{construct} operation to guarantee that every triangular
      ++ set build from a list of polynomials has the required properties.
    extend : ($,P) -> $
      ++ \axiom{extend(ts,p)} returns a triangular set which encodes the
      ++ simple extension by \axiom{p} of the extension of the base field
      ++ defined by \axiom{ts}, according to the properties of triangular
      ++ sets of the current category. If the required properties do not
      ++ hold an error is returned.
    if V has Finite
    then
      coHeight : $ -> NonNegativeInteger
        ++ \axiom{coHeight(ts)} returns \axiom{size()\$V} minus \axiom{\#ts}.
add

    GPS ==> GeneralPolynomialSet(R,E,V,P)
    B ==> Boolean
    RBT ==> Record(bas:$,top:List P)

    ts:$ = us:$ ==
      empty?(ts)$$ => empty?(us)$$
      empty?(us)$$ => false
      first(ts)::P =$P first(us)::P => rest(ts)::$ =$$ rest(us)::$
      false

    infRittWu?(ts,us) ==
      empty?(us)$$ => not empty?(ts)$$
      empty?(ts)$$ => false
      p : P := (last(ts))::P
```

```
    q : P := (last(us))::P
    infRittWu?(p,q)$P => true
    supRittWu?(p,q)$P => false
    v : V := mvar(p)
    infRittWu?(collectUpper(ts,v),collectUpper(us,v))$$

  reduced?(p,ts,redOp?) ==
    lp : List P := members(ts)
    while (not empty? lp) and (redOp?(p,first(lp))) repeat
      lp := rest lp
    empty? lp

  basicSet(ps,redOp?) ==
    ps := remove(zero?,ps)
    any?(ground?,ps) => "failed"::Union(RBT,"failed")
    ps := sort(infRittWu?,ps)
    p,b : P
    bs := empty()$$
    ts : List P := []
    while not empty? ps repeat
      b := first(ps)
      bs := extend(bs,b)$$
      ps := rest ps
      while (not empty? ps) and _
            (not reduced?((p := first(ps)),bs,redOp?)) repeat
        ts := cons(p,ts)
        ps := rest ps
    ([bs,ts]$RBT)::Union(RBT,"failed")

  basicSet(ps,pred?,redOp?) ==
    ps := remove(zero?,ps)
    any?(ground?,ps) => "failed"::Union(RBT,"failed")
    gps : List P := []
    bps : List P := []
    while not empty? ps repeat
      p := first ps
      ps := rest ps
      if pred?(p)
        then
          gps := cons(p,gps)
        else
          bps := cons(p,bps)
    gps := sort(infRittWu?,gps)
    p,b : P
    bs := empty()$$
    ts : List P := []
    while not empty? gps repeat
      b := first(gps)
      bs := extend(bs,b)$$
      gps := rest gps
```

```
    while (not empty? gps) and _
          (not reduced?((p := first(gps)),bs,redOp?)) repeat
      ts := cons(p,ts)
      gps := rest gps
  ts := sort(infRittWu?,concat(ts,bps))
  ([bs,ts]$RBT)::Union(RBT,"failed")

initials ts ==
  lip : List P := []
  empty? ts => lip
  lp := members(ts)
  while not empty? lp repeat
     p := first(lp)
     if not ground?((ip := init(p)))
       then
         lip := cons(primPartElseUnitCanonical(ip),lip)
     lp := rest lp
  removeDuplicates lip

degree ts ==
  empty? ts => 0$NonNegativeInteger
  lp := members ts
  d : NonNegativeInteger := mdeg(first lp)
  while not empty? (lp := rest lp) repeat
    d := d * mdeg(first lp)
  d

quasiComponent ts ==
  [members(ts),initials(ts)]

normalized?(p,ts) ==
  normalized?(p,members(ts))$P

stronglyReduced? (p,ts) ==
  reduced?(p,members(ts))$P

headReduced? (p,ts) ==
  stronglyReduced?(head(p),ts)

initiallyReduced? (p,ts) ==
  lp : List (P) := members(ts)
  red : Boolean := true
  while (not empty? lp) and (not ground?(p)$P) and red repeat
    while (not empty? lp) and (mvar(first(lp)) > mvar(p)) repeat
      lp := rest lp
    if (not empty? lp)
      then
        if  (mvar(first(lp)) = mvar(p))
          then
            if reduced?(p,first(lp))
```

```
              then
                lp := rest lp
                p := init(p)
              else
                red := false
          else
            p := init(p)
  red

reduce(p,ts,redOp,redOp?) ==
  (empty? ts) or (ground? p) => p
  ts0 := ts
  while (not empty? ts) and (not ground? p) repeat
     reductor := (first ts)::P
     ts := (rest ts)::$
     if not redOp?(p,reductor)
       then
         p := redOp(p,reductor)
         ts := ts0
  p

rewriteSetWithReduction(lp,ts,redOp,redOp?) ==
  trivialIdeal? ts => lp
  lp := remove(zero?,lp)
  empty? lp => lp
  any?(ground?,lp) => [1$P]
  rs : List P := []
  while not empty? lp repeat
    p := first lp
    lp := rest lp
    p := primPartElseUnitCanonical reduce(p,ts,redOp,redOp?)
    if not zero? p
      then
        if ground? p
          then
            lp := []
            rs := [1$P]
          else
            rs := cons(p,rs)
  removeDuplicates rs

stronglyReduce(p,ts) ==
  reduce (p,ts,lazyPrem,reduced?)

headReduce(p,ts) ==
  reduce (p,ts,headReduce,headReduced?)

initiallyReduce(p,ts) ==
  reduce (p,ts,initiallyReduce,initiallyReduced?)
```

```
removeZero(p,ts) ==
  (ground? p) or (empty? ts) => p
  v := mvar(p)
  ts_v_- := collectUnder(ts,v)
  if algebraic?(v,ts)
    then
      q := lazyPrem(p,select(ts,v)::P)
      zero? q => return q
      zero? removeZero(q,ts_v_-) => return 0
  empty? ts_v_- => p
  q: P := 0
  while positive? degree(p,v) repeat
     q := removeZero(init(p),ts_v_-) * mainMonomial(p) + q
     p := tail(p)
  q + removeZero(p,ts_v_-)

reduceByQuasiMonic(p, ts) ==
  (ground? p) or (empty? ts) => p
  remainder(p,collectQuasiMonic(ts)).polnum

autoReduced?(ts : $,redOp? : ((P,List(P)) -> Boolean)) ==
  empty? ts => true
  lp : List (P) := members(ts)
  p : P := first(lp)
  lp := rest lp
  while (not empty? lp) and redOp?(p,lp) repeat
     p := first lp
     lp := rest lp
  empty? lp

stronglyReduced? ts ==
  autoReduced? (ts, reduced?)

normalized? ts ==
  autoReduced? (ts,normalized?)

headReduced? ts ==
  autoReduced? (ts,headReduced?)

initiallyReduced?  ts ==
  autoReduced? (ts,initiallyReduced?)

mvar ts ==
  empty? ts => error"Error from TSETCAT in mvar : #1 is empty"
  mvar((first(ts))::P)$P

first ts ==
  empty? ts => "failed"::Union(P,"failed")
  lp : List(P) := sort(supRittWu?,members(ts))$(List P)
  first(lp)::Union(P,"failed")
```

```
last ts ==
  empty? ts => "failed"::Union(P,"failed")
  lp : List(P) := sort(infRittWu?,members(ts))$(List P)
  first(lp)::Union(P,"failed")

rest ts ==
  empty? ts => "failed"::Union($,"failed")
  lp : List(P) := sort(supRittWu?,members(ts))$(List P)
  construct(rest(lp))::Union($,"failed")

coerce (ts:$) : List(P) ==
  sort(supRittWu?,members(ts))$(List P)

algebraicVariables ts ==
  [mvar(p) for p in members(ts)]

algebraic? (v,ts) ==
  member?(v,algebraicVariables(ts))

select  (ts,v) ==
  lp : List (P) := sort(supRittWu?,members(ts))$(List P)
  while (not empty? lp) and (not (v = mvar(first lp))) repeat
    lp := rest lp
  empty? lp => "failed"::Union(P,"failed")
  (first lp)::Union(P,"failed")

collectQuasiMonic ts ==
  lp: List(P) := members(ts)
  newlp: List(P) := []
  while (not empty? lp) repeat
    if ground? init(first(lp)) then newlp := cons(first(lp),newlp)
    lp := rest lp
  construct(newlp)

collectUnder (ts,v) ==
  lp : List (P) := sort(supRittWu?,members(ts))$(List P)
  while (not empty? lp) and (not (v > mvar(first lp))) repeat
    lp := rest lp
  construct(lp)

collectUpper  (ts,v) ==
  lp1 : List(P) := sort(supRittWu?,members(ts))$(List P)
  lp2 : List(P) := []
  while (not empty? lp1) and  (mvar(first lp1) > v) repeat
    lp2 := cons(first(lp1),lp2)
    lp1 := rest lp1
  construct(reverse lp2)

construct(lp:List(P)) ==
```

```
      rif := retractIfCan(lp)@Union($,"failed")
      not (rif case $) => error"in construct : LP -> $ from TSETCAT : bad arg"
      rif::$

    retractIfCan(lp:List(P)) ==
      empty? lp => (empty()$$)::Union($,"failed")
      lp := sort(supRittWu?,lp)
      rif := retractIfCan(rest(lp))@Union($,"failed")
      not (rif case $) => _
       error "in retractIfCan : LP -> ... from TSETCAT : bad arg"
      extendIfCan(rif::$,first(lp))@Union($,"failed")

    extend(ts:$,p:P):$ ==
      eif := extendIfCan(ts,p)@Union($,"failed")
      not (eif case $) => error"in extend : ($,P) -> $ from TSETCAT : bad ars"
      eif::$

    if V has Finite
    then

      coHeight ts ==
        n := size()$V
        m := #(members ts)
        subtractIfCan(n,m)$NonNegativeInteger::NonNegativeInteger
```

———-

— TSETCAT.dotabb —

```
"TSETCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=TSETCAT"];
"TSETCAT" -> "PSETCAT"
```

———-

— TSETCAT.dotfull —

```
"TriangularSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCategory
 [color=lightblue,href="bookvol10.2.pdf#nameddest=TSETCAT"];
"TriangularSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCategory
  -> "PolynomialSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCat

"TriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCategory(a,b
 [color=seagreen,href="bookvol10.2.pdf#nameddest=TSETCAT"];
"TriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCategory(a,b
 ->
```

```
"TriangularSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
```

———-

— TSETCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"TriangularSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
 [color=lightblue];
"TriangularSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
  -> "PolynomialSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursiveP

"PolynomialSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
 [color=seagreen];
"PolynomialSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
  -> "PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialC

"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCatego
 [color=lightblue];
"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCatego
 -> "SetCategory()"
"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCatego
 -> "Collection(RecursivePolynomialCategory(Ring,OrderedAbelianMonoidSup(),OrderedSet())"
"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCatego
 -> "CoercibleTo(List(RecursivePolynomialCategory(Ring,OrderedAbelianMonoidSup(),OrderedSet()))"

"CoercibleTo(List(RecursivePolynomialCategory(Ring,OrderedAbelianMonoidSup(),OrderedSet()))"
 [color=seagreen];
"CoercibleTo(List(RecursivePolynomialCategory(Ring,OrderedAbelianMonoidSup(),OrderedSet()))"
  -> "CoercibleTo(a:Type)"

"Collection(RecursivePolynomialCategory(Ring,OrderedAbelianMonoidSup(),OrderedSet())" [color=sea
"Collection(RecursivePolynomialCategory(Ring,OrderedAbelianMonoidSup(),OrderedSet())"
  -> "Collection(a:Type)"

"Collection(a:Type)" [color=lightblue];
"Collection(a:Type)" -> "HomogeneousAggregate(a:Type)"
"Collection(a:Type)" -> "ConvertibleTo(InputForm)"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
```

```
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"HomogeneousAggregate(a:Type)" [color=lightblue];
"HomogeneousAggregate(a:Type)" -> "Aggregate()"

"Aggregate()" [color=lightblue];
"Aggregate()" -> "Type()"

"Type()" [color=lightblue];
"Type()" -> "Category"

"ConvertibleTo(InputForm)" [color="#00EE00"];
"ConvertibleTo(InputForm)" -> "ConvertibleTo(a:Type)"

"ConvertibleTo(a:Type)" [color="#00EE00"];
"ConvertibleTo(a:Type)" -> "Category"

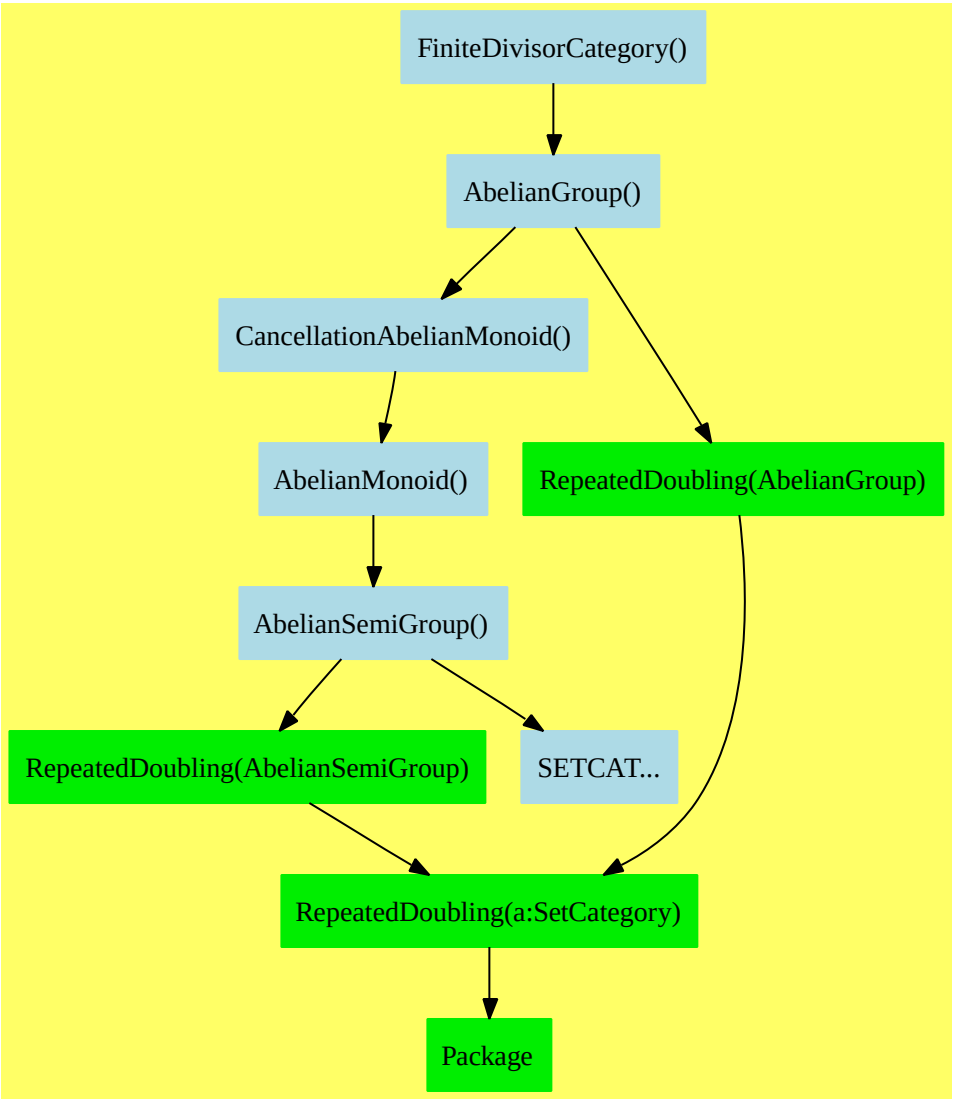
"Category" [color=lightblue];
}
```

———————-

Chapter 8

Category Layer 7

8.0.109 FiniteDivisorCategory (FDIVCAT)



— **FiniteDivisorCategory.input** —

```
)set break resume
)sys rm -f FiniteDivisorCategory.output
)spool FiniteDivisorCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FiniteDivisorCategory
--R FiniteDivisorCategory(F: Field,UP: UnivariatePolynomialCategory t#1,UPUP: UnivariatePolynomialCategory
--R Abbreviation for FiniteDivisorCategory is FDIVCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FDIVCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (Integer,%) -> %               ?*? : (PositiveInteger,%) -> %
--R ?+? : (%,%) -> %                     ?-? : (%,%) -> %
--R -? : % -> %                          ?=? : (%,%) -> Boolean
--R 0 : () -> %                          coerce : % -> OutputForm
--R divisor : (R,UP,UP,UP,F) -> %        divisor : (F,F,Integer) -> %
--R divisor : (F,F) -> %                 divisor : R -> %
--R hash : % -> SingleInteger            latex : % -> String
--R principal? : % -> Boolean            reduce : % -> %
--R sample : () -> %                     zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R decompose : % -> Record(id: FractionalIdeal(UP,Fraction UP,UPUP,R),principalPart: R)
--R divisor : FractionalIdeal(UP,Fraction UP,UPUP,R) -> %
--R generator : % -> Union(R,"failed")
--R ideal : % -> FractionalIdeal(UP,Fraction UP,UPUP,R)
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **FiniteDivisorCategory.help** —

```
=====================================================================
FiniteDivisorCategory examples
=====================================================================
```

```
See Also:
o )show FiniteDivisorCategory
```

See:

⇐ "AbelianGroup" (ABELGRP) 7.0.97 on page 520

Exports:

0	coerce	decompose	divisor	generator
hash	ideal	latex	principal?	reduce
sample	subtractIfCan	zero?	?~=?	?*?
?+?	?-?	-?	?=?	

These are directly exported but not implemented:

```
decompose : % ->
  Record(id: FractionalIdeal(UP,Fraction UP,UPUP,R),principalPart: R)
divisor : R -> %
divisor : FractionalIdeal(UP,Fraction UP,UPUP,R) -> %
divisor : (F,F) -> %
divisor : (F,F,Integer) -> %
divisor : (R,UP,UP,UP,F) -> %
generator : % -> Union(R,"failed")
ideal : % -> FractionalIdeal(UP,Fraction UP,UPUP,R)
reduce : % -> %
```

These are implemented by this category:

```
principal? : % -> Boolean
```

These exports come from (p520) AbelianGroup():

```
0 : () -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
-? : % -> %
?-? : (%,%) -> %
?~=? : (%,%) -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?*? : (PositiveInteger,%) -> %
?*? : (Integer,%) -> %
?*? : (NonNegativeInteger,%) -> %
```

— category FDIVCAT FiniteDivisorCategory —

```
)abbrev category FDIVCAT FiniteDivisorCategory
++ Category for finite rational divisors on a curve
++ Author: Manuel Bronstein
++ Date Created: 19 May 1993
++ Date Last Updated: 19 May 1993
++ Keywords: divisor, algebraic, curve.
++ Description:
++ This category describes finite rational divisors on a curve, that
++ is finite formal sums SUM(n * P) where the n's are integers and the
++ P's are finite rational points on the curve.

FiniteDivisorCategory(F, UP, UPUP, R): Category == Result where
  F   : Field
  UP  : UnivariatePolynomialCategory F
  UPUP: UnivariatePolynomialCategory Fraction UP
  R   : FunctionFieldCategory(F, UP, UPUP)

  ID  ==> FractionalIdeal(UP, Fraction UP, UPUP, R)

  Result ==> AbelianGroup with
    ideal      : % -> ID
      ++ ideal(D) returns the ideal corresponding to a divisor D.
    divisor    : ID -> %
      ++ divisor(I) makes a divisor D from an ideal I.
    divisor    : R -> %
      ++ divisor(g) returns the divisor of the function g.
    divisor    : (F, F) -> %
      ++ divisor(a, b) makes the divisor P: \spad{(x = a, y = b)}.
      ++ Error: if P is singular.
    divisor    : (F, F, Integer) -> %
      ++ divisor(a, b, n) makes the divisor
      ++ \spad{nP} where P: \spad{(x = a, y = b)}.
      ++ P is allowed to be singular if n is a multiple of the rank.
    decompose  : % -> Record(id:ID, principalPart: R)
      ++ decompose(d) returns \spad{[id, f]} where \spad{d = (id) + div(f)}.
    reduce     : % -> %
      ++ reduce(D) converts D to some reduced form (the reduced forms can
      ++ be differents in different implementations).
    principal? : % -> Boolean
      ++ principal?(D) tests if the argument is the divisor of a function.
    generator  : % -> Union(R, "failed")
      ++ generator(d) returns f if \spad{(f) = d},
      ++ "failed" if d is not principal.
    divisor    : (R, UP, UP, UP, F) -> %
      ++ divisor(h, d, d', g, r) returns the sum of all the finite points
      ++ where \spad{h/d} has residue \spad{r}.
      ++ \spad{h} must be integral.
```

```
      ++ \spad{d} must be squarefree.
      ++ \spad{d'} is some derivative of \spad{d} (not necessarily dd/dx).
      ++ \spad{g = gcd(d,discriminant)} contains the ramified zeros of \spad{d}
   add
    principal? d == generator(d) case R
```

———-

— **FDIVCAT.dotabb** —

```
"FDIVCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FDIVCAT"];
"FDIVCAT" -> "ABELGRP"
```

———-

— **FDIVCAT.dotfull** —

```
"FiniteDivisorCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FDIVCAT"];
"FiniteDivisorCategory()" -> "AbelianGroup()"
```

———-

— **FDIVCAT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FiniteDivisorCategory()" [color=lightblue];
"FiniteDivisorCategory()" -> "AbelianGroup()"

"AbelianGroup()" [color=lightblue];
"AbelianGroup()" -> "CancellationAbelianMonoid()"
"AbelianGroup()" -> "RepeatedDoubling(AbelianGroup)"

"RepeatedDoubling(AbelianGroup)" [color="#00EE00"];
"RepeatedDoubling(AbelianGroup)" -> "RepeatedDoubling(a:SetCategory)"

"RepeatedDoubling(AbelianSemiGroup)" [color="#00EE00"];
"RepeatedDoubling(AbelianSemiGroup)" -> "RepeatedDoubling(a:SetCategory)"

"RepeatedDoubling(a:SetCategory)" [color="#00EE00"];
```

```
"RepeatedDoubling(a:SetCategory)" -> "Package"

"Package" [color="#00EE00"];

"CancellationAbelianMonoid()" [color=lightblue];
"CancellationAbelianMonoid()" -> "AbelianMonoid()"

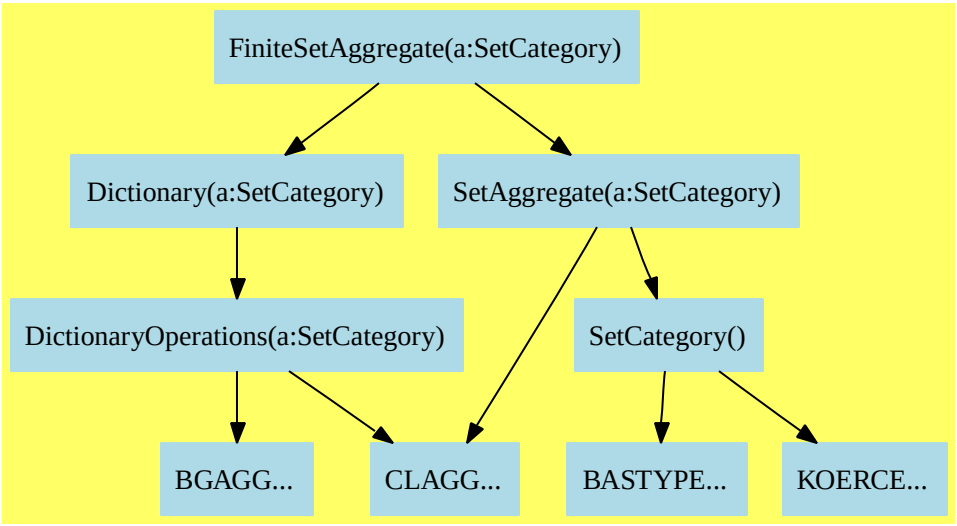
"AbelianMonoid()" [color=lightblue];
"AbelianMonoid()" -> "AbelianSemiGroup()"

"AbelianSemiGroup()" [color=lightblue];
"AbelianSemiGroup()" -> "SETCAT..."
"AbelianSemiGroup()" -> "RepeatedDoubling(AbelianSemiGroup)"

"SETCAT..." [color=lightblue];
}
```

———-

8.0.110 FiniteSetAggregate (FSAGG)



— **FiniteSetAggregate.input** —

```
)set break resume
)sys rm -f FiniteSetAggregate.output
)spool FiniteSetAggregate.output
```

```
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FiniteSetAggregate
--R FiniteSetAggregate S: SetCategory  is a category constructor
--R Abbreviation for FiniteSetAggregate is FSAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FSAGG
--R
--R------------------------------ Operations --------------------------------
--R ?<? : (%,%) -> Boolean                ?=? : (%,%) -> Boolean
--R bag : List S -> %                     brace : () -> %
--R brace : List S -> %                   coerce : % -> OutputForm
--R construct : List S -> %               copy : % -> %
--R dictionary : List S -> %              dictionary : () -> %
--R difference : (%,%) -> %               difference : (%,S) -> %
--R empty : () -> %                       empty? : % -> Boolean
--R eq? : (%,%) -> Boolean                extract! : % -> S
--R hash : % -> SingleInteger             insert! : (S,%) -> %
--R inspect : % -> S                      intersect : (%,%) -> %
--R latex : % -> String                   map : ((S -> S),%) -> %
--R max : % -> S if S has ORDSET          min : % -> S if S has ORDSET
--R sample : () -> %                      set : () -> %
--R set : List S -> %                     subset? : (%,%) -> Boolean
--R union : (%,%) -> %                    union : (%,S) -> %
--R union : (S,%) -> %                    ?~=? : (%,%) -> Boolean
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R cardinality : % -> NonNegativeInteger
--R complement : % -> % if S has FINITE
--R convert : % -> InputForm if S has KONVERT INFORM
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R find : ((S -> Boolean),%) -> Union(S,"failed")
--R index : PositiveInteger -> % if S has FINITE
--R less? : (%,NonNegativeInteger) -> Boolean
--R lookup : % -> PositiveInteger if S has FINITE
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List S if $ has finiteAggregate
--R random : () -> % if S has FINITE
```

```
--R reduce : (((S,S) -> S),%) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S,S) -> S if S has SETCAT and $ has finiteAggregate
--R remove : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (S,%) -> % if S has SETCAT and $ has finiteAggregate
--R remove! : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R remove! : (S,%) -> % if $ has finiteAggregate
--R removeDuplicates : % -> % if S has SETCAT and $ has finiteAggregate
--R select : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R select! : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R size : () -> NonNegativeInteger if S has FINITE
--R size? : (%,NonNegativeInteger) -> Boolean
--R symmetricDifference : (%,%) -> %
--R universe : () -> % if S has FINITE
--R
--E 1

)spool
)lisp (bye)
```

———-

— **FiniteSetAggregate.help** —

```
=====================================================================
FiniteSetAggregate examples
=====================================================================

See Also:
o )show FiniteSetAggregate
```

———-

See:

⇐ “Dictionary” (DIAGG) 7.0.99 on page 533
⇐ “SetAggregate” (SETAGG) 6.0.94 on page 492

Exports:

any?	bag	brace	cardinality	coerce
complement	construct	convert	copy	count
dictionary	difference	empty	empty?	eq?
eval	every?	extract!	find	hash
index	insert!	inspect	intersect	latex
less?	lookup	map	map!	max
member?	members	min	more?	parts
random	reduce	remove	remove!	removeDuplicates
sample	select	select!	set	size
size?	subset?	symmetricDifference	union	universe
#?	?<?	?=?	?~=?	

Attributes Exported:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **partiallyOrderedSet** is true if a set with $<$ which is transitive, but not($a < b$ or $a = b$) does not necessarily imply $b < a$.
- **nil**

These are implemented by this category:

```
brace : List S -> %
cardinality : % -> NonNegativeInteger
coerce : % -> OutputForm
complement : % -> % if S has FINITE
construct : List S -> %
count : (S,%) -> NonNegativeInteger
    if S has SETCAT and $ has finiteAggregate
difference : (%,%) -> %
index : PositiveInteger -> % if S has FINITE
intersect : (%,%) -> %
lookup : % -> PositiveInteger if S has FINITE
max : % -> S if S has ORDSET
min : % -> S if S has ORDSET
random : () -> % if S has FINITE
set : List S -> %
size : () -> NonNegativeInteger if S has FINITE
subset? : (%,%) -> Boolean
symmetricDifference : (%,%) -> %
union : (%,%) -> %
universe : () -> % if S has FINITE
?<? : (%,%) -> Boolean
?=? : (%,%) -> Boolean
```

These exports come from (p533) Dictionary(S:SetCategory):

```
any? : ((S -> Boolean),%) -> Boolean
    if $ has finiteAggregate
bag : List S -> %
convert : % -> InputForm if S has KONVERT INFORM
copy : % -> %
count : ((S -> Boolean),%) -> NonNegativeInteger
    if $ has finiteAggregate
dictionary : () -> %
dictionary : List S -> %
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> %
    if S has EVALAB S
    and S has SETCAT
eval : (%,S,S) -> %
    if S has EVALAB S
    and S has SETCAT
eval : (%,Equation S) -> %
    if S has EVALAB S
    and S has SETCAT
eval : (%,List Equation S) -> %
    if S has EVALAB S
    and S has SETCAT
every? : ((S -> Boolean),%) -> Boolean
    if $ has finiteAggregate
extract! : % -> S
find : ((S -> Boolean),%) -> Union(S,"failed")
hash : % -> SingleInteger
insert! : (S,%) -> %
inspect : % -> S
latex : % -> String
less? : (%,NonNegativeInteger) -> Boolean
select! : ((S -> Boolean),%) -> %
    if $ has finiteAggregate
map : ((S -> S),%) -> %
map! : ((S -> S),%) -> %
    if $ has shallowlyMutable
member? : (S,%) -> Boolean
    if S has SETCAT
    and $ has finiteAggregate
members : % -> List S if $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
parts : % -> List S if $ has finiteAggregate
reduce : (((S,S) -> S),%) -> S
    if $ has finiteAggregate
reduce : (((S,S) -> S),%,S) -> S
    if $ has finiteAggregate
```

```
 reduce : (((S,S) -> S),%,S,S) -> S
     if S has SETCAT
     and $ has finiteAggregate
 remove : ((S -> Boolean),%) -> %
     if $ has finiteAggregate
 remove : (S,%) -> %
     if S has SETCAT
     and $ has finiteAggregate
 remove! : ((S -> Boolean),%) -> %
     if $ has finiteAggregate
 remove! : (S,%) -> % if $ has finiteAggregate
 removeDuplicates : % -> %
     if S has SETCAT
     and $ has finiteAggregate
 sample : () -> %
 select : ((S -> Boolean),%) -> %
     if $ has finiteAggregate
 size? : (%,NonNegativeInteger) -> Boolean
 #? : % -> NonNegativeInteger if $ has finiteAggregate
 ?~=? : (%,%) -> Boolean
```

These exports come from (p492) SetAggregate(S:SetCategory):

```
 brace : () -> %
 difference : (%,S) -> %
 set : () -> %
 union : (%,S) -> %
 union : (S,%) -> %
```

— category FSAGG FiniteSetAggregate —

```
)abbrev category FSAGG FiniteSetAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: 14 Oct, 1993 by RSS
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A finite-set aggregate models the notion of a finite set, that is,
++ a collection of elements characterized by membership, but not
++ by order or multiplicity.
++ See \spadtype{Set} for an example.

FiniteSetAggregate(S:SetCategory): Category ==
  Join(Dictionary S, SetAggregate S) with
```

```
   finiteAggregate
   cardinality: % -> NonNegativeInteger
     ++ cardinality(u) returns the number of elements of u.
     ++ Note that \axiom{cardinality(u) = #u}.
   if S has Finite then
     Finite
     complement: % -> %
       ++ complement(u) returns the complement of the set u,
       ++ i.e. the set of all values not in u.
     universe: () -> %
       ++ universe()$D returns the universal set for finite set aggregate D.
   if S has OrderedSet then
     max: % -> S
       ++ max(u) returns the largest element of aggregate u.
     min: % -> S
       ++ min(u) returns the smallest element of aggregate u.

add
  s < t           == #s < #t and s = intersect(s,t)
  s = t           == #s = #t and empty? difference(s,t)
  brace l           == construct l
  set         l           == construct l
  cardinality s   == #s
  construct l           == (s := set(); for x in l repeat insert_!(x,s); s)
  count(x:S, s:%) == (member?(x, s) => 1; 0)
  subset?(s, t)   == #s < #t and _and/[member?(x, t) for x in parts s]

  coerce(s:%):OutputForm ==
    brace [x::OutputForm for x in parts s]$List(OutputForm)

  intersect(s, t) ==
    i := {}
    for x in parts s | member?(x, t) repeat insert_!(x, i)
    i

  difference(s:%, t:%) ==
    m := copy s
    for x in parts t repeat remove_!(x, m)
    m

  symmetricDifference(s, t) ==
    d := copy s
    for x in parts t repeat
      if member?(x, s) then remove_!(x, d) else insert_!(x, d)
    d

  union(s:%, t:%) ==
     u := copy s
     for x in parts t repeat insert_!(x, u)
     u
```

```
  if S has Finite then
    universe()          == {index(i::PositiveInteger) for i in 1..size()$S}
    complement s == difference(universe(), s )
    size()  == 2 ** size()$S
    index i ==
      {index(j::PositiveInteger)$S for j in 1..size()$S | bit?(i-1,j-1)}
    random()  ==
      index((random()$Integer rem (size()$% + 1))::PositiveInteger)

    lookup s ==
      n:PositiveInteger := 1
      for x in parts s repeat _
        n := n + 2 ** ((lookup(x) - 1)::NonNegativeInteger)
      n

  if S has OrderedSet then
    max s ==
      empty?(l := parts s) => error "Empty set"
      reduce("max", l)

    min s ==
      empty?(l := parts s) => error "Empty set"
      reduce("min", l)
```

———-

— FSAGG.dotabb —

```
"FSAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=FSAGG"];
"FSAGG" -> "DIAGG"
"FSAGG" -> "SETAGG"
```

———-

— FSAGG.dotfull —

```
"FiniteSetAggregate(a:SetCategory)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FSAGG"];
"FiniteSetAggregate(a:SetCategory)" -> "Dictionary(a:SetCategory)"
"FiniteSetAggregate(a:SetCategory)" -> "SetAggregate(a:SetCategory)"
```

———-

— FSAGG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FiniteSetAggregate(a:SetCategory)" [color=lightblue];
"FiniteSetAggregate(a:SetCategory)" -> "Dictionary(a:SetCategory)"
"FiniteSetAggregate(a:SetCategory)" -> "SetAggregate(a:SetCategory)"

"SetAggregate(a:SetCategory)" [color=lightblue];
"SetAggregate(a:SetCategory)" -> "SetCategory()"
"SetAggregate(a:SetCategory)" -> "CLAGG..."

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BASTYPE..."
"SetCategory()" -> "KOERCE..."

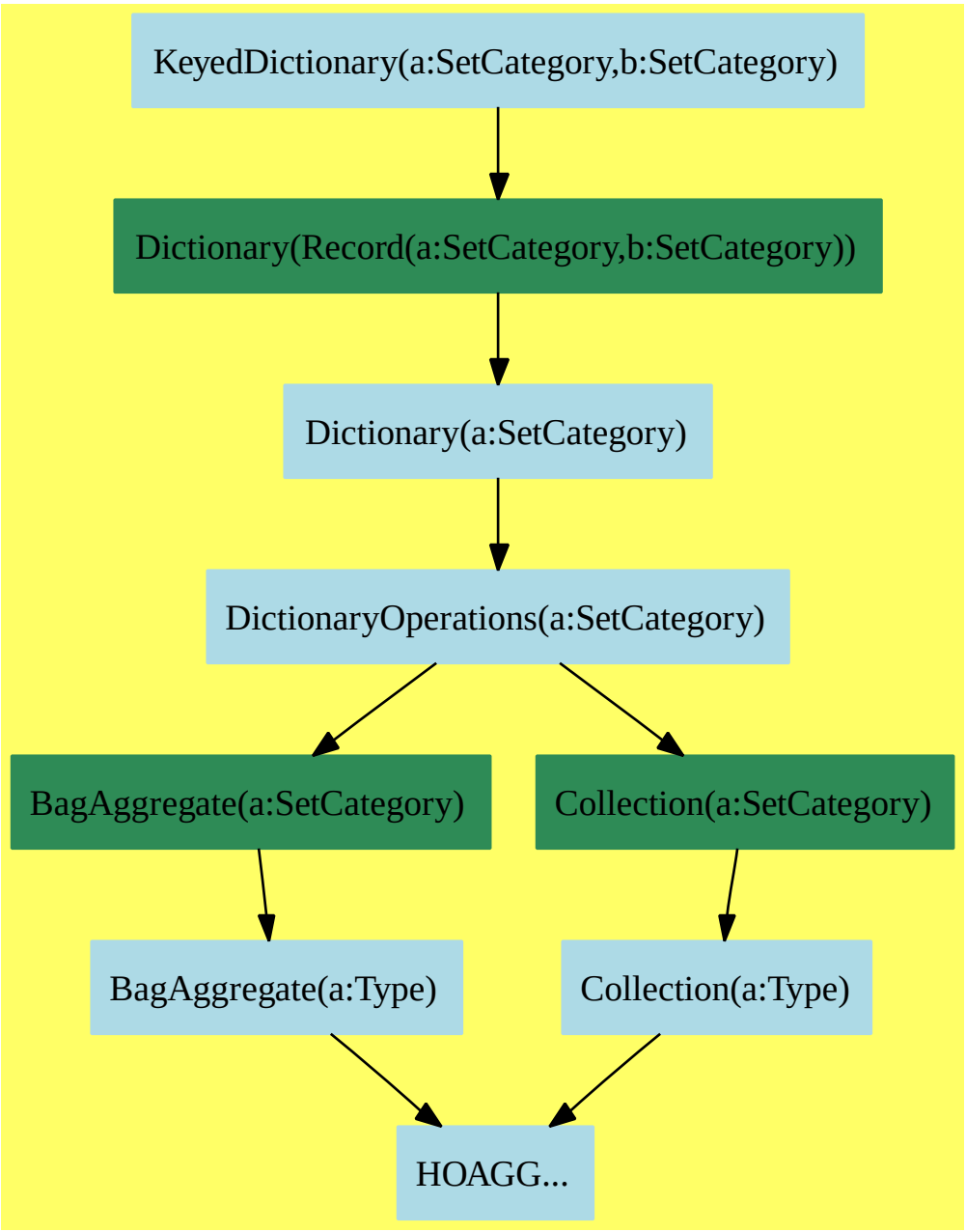
"Dictionary(a:SetCategory)" [color=lightblue];
"Dictionary(a:SetCategory)" -> "DictionaryOperations(a:SetCategory)"

"DictionaryOperations(a:SetCategory)" [color=lightblue];
"DictionaryOperations(a:SetCategory)" -> "BGAGG..."
"DictionaryOperations(a:SetCategory)" -> "CLAGG..."

"BGAGG..." [color=lightblue];
"CLAGG..." [color=lightblue];
"BASTYPE..." [color=lightblue];
"KOERCE..." [color=lightblue];
}
```

———————-

8.0.111 KeyedDictionary (KDAGG)



— **KeyedDictionary.input** —

```
)set break resume
)sys rm -f KeyedDictionary.output
)spool KeyedDictionary.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show KeyedDictionary
--R KeyedDictionary(Key: SetCategory,Entry: SetCategory)  is a category constructor
--R Abbreviation for KeyedDictionary is KDAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for KDAGG
--R
--R------------------------------ Operations --------------------------------
--R copy : % -> %                         dictionary : () -> %
--R empty : () -> %                       empty? : % -> Boolean
--R eq? : (%,%) -> Boolean                key? : (Key,%) -> Boolean
--R keys : % -> List Key                  sample : () -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if Record(key: Key,entry: Entry) has SETCAT
--R any? : ((Record(key: Key,entry: Entry) -> Boolean),%) -> Boolean if $ has finiteAggregate
--R bag : List Record(key: Key,entry: Entry) -> %
--R coerce : % -> OutputForm if Record(key: Key,entry: Entry) has SETCAT
--R construct : List Record(key: Key,entry: Entry) -> %
--R convert : % -> InputForm if Record(key: Key,entry: Entry) has KONVERT INFORM
--R count : (Record(key: Key,entry: Entry),%) -> NonNegativeInteger if Record(key: Key,entry: Entry) has SET
--R count : ((Record(key: Key,entry: Entry) -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R dictionary : List Record(key: Key,entry: Entry) -> %
--R eval : (%,List Record(key: Key,entry: Entry),List Record(key: Key,entry: Entry)) -> % if Record(key: Key
--R eval : (%,Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> % if Record(key: Key,entry: En
--R eval : (%,Equation Record(key: Key,entry: Entry)) -> % if Record(key: Key,entry: Entry) has EVALAB Recor
--R eval : (%,List Equation Record(key: Key,entry: Entry)) -> % if Record(key: Key,entry: Entry) has EVALAB
--R every? : ((Record(key: Key,entry: Entry) -> Boolean),%) -> Boolean if $ has finiteAggregate
--R extract! : % -> Record(key: Key,entry: Entry)
--R find : ((Record(key: Key,entry: Entry) -> Boolean),%) -> Union(Record(key: Key,entry: Entry),"failed")
--R hash : % -> SingleInteger if Record(key: Key,entry: Entry) has SETCAT
--R insert! : (Record(key: Key,entry: Entry),%) -> %
--R inspect : % -> Record(key: Key,entry: Entry)
--R latex : % -> String if Record(key: Key,entry: Entry) has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map : ((Record(key: Key,entry: Entry) -> Record(key: Key,entry: Entry)),%) -> %
--R map! : ((Record(key: Key,entry: Entry) -> Record(key: Key,entry: Entry)),%) -> % if $ has shallowlyMutabl
--R member? : (Record(key: Key,entry: Entry),%) -> Boolean if Record(key: Key,entry: Entry) has SETCAT and $
--R members : % -> List Record(key: Key,entry: Entry) if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List Record(key: Key,entry: Entry) if $ has finiteAggregate
--R reduce : (((Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> Record(key: Key,entry: Entry
--R reduce : (((Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> Record(key: Key,entry: Entry
--R reduce : (((Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> Record(key: Key,entry: Entry
```

```
--R remove : ((Record(key: Key,entry: Entry) -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (Record(key: Key,entry: Entry),%) -> % if Record(key: Key,entry: Entry) has SETCAT
--R remove! : (Key,%) -> Union(Entry,"failed")
--R remove! : ((Record(key: Key,entry: Entry) -> Boolean),%) -> % if $ has finiteAggregate
--R remove! : (Record(key: Key,entry: Entry),%) -> % if $ has finiteAggregate
--R removeDuplicates : % -> % if Record(key: Key,entry: Entry) has SETCAT and $ has finiteAggreg
--R search : (Key,%) -> Union(Entry,"failed")
--R select : ((Record(key: Key,entry: Entry) -> Boolean),%) -> % if $ has finiteAggregate
--R select! : ((Record(key: Key,entry: Entry) -> Boolean),%) -> % if $ has finiteAggregate
--R size? : (%,NonNegativeInteger) -> Boolean
--R ?~=? : (%,%) -> Boolean if Record(key: Key,entry: Entry) has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— **KeyedDictionary.help** —

```
====================================================================
KeyedDictionary examples
====================================================================

See Also:
o )show KeyedDictionary
```

———-

See:

⇒ “TableAggregate” (TBAGG) 9.0.132 on page 797
⇐ “Dictionary” (DIAGG) 7.0.99 on page 533

Exports:

any?	bag	coerce	construct	convert
copy	count	dictionary	empty	empty?
eq?	eval	every?	extract!	find
hash	insert!	inspect	key?	keys
latex	less?	map	map!	member?
members	more?	parts	reduce	remove
remove!	removeDuplicates	sample	search	select
select!	size?	#?	?=?	?~=?

Attributes exported:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **nil**

Attributes Used:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.

These are directly exported but not implemented:

```
remove! : (Key,%) -> Union(Entry,"failed")
search : (Key,%) -> Union(Entry,"failed")
```

These are implemented by this category:

```
key? : (Key,%) -> Boolean
member? : (Record(key: Key,entry: Entry),%) -> Boolean
    if Record(key: Key,entry: Entry) has SETCAT
    and $ has finiteAggregate
keys : % -> List Key
```

These exports come from (p533) Dictionary(R)
where R=Record(a:SetCategory,b:SetCategory))
and S=Record(key: Key,entry: Entry)

```
any? : ((S) -> Boolean),%) -> Boolean
    if $ has finiteAggregate
bag : List S -> %
coerce : % -> OutputForm if S has SETCAT
construct : List S -> %
convert : % -> InputForm if S has KONVERT INFORM
copy : % -> %
count : (S,%) -> NonNegativeInteger
    if S has SETCAT and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
    if $ has finiteAggregate
dictionary : () -> %
dictionary : List S -> %
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> %
    if S has EVALAB S and S has SETCAT
eval : (%,S,S) -> %
    if S has EVALAB S and S has SETCAT
eval : (%,Equation S) -> %
    if S has EVALAB S and S has SETCAT
```

```
eval : (%,List Equation S) -> %
    if S has EVALAB S and S has SETCAT
every? : ((S -> Boolean),%) -> Boolean
    if $ has finiteAggregate
extract! : % -> S
find : ((S -> Boolean),%) -> Union(S,"failed")
hash : % -> SingleInteger if S has SETCAT
insert! : (S,%) -> %
inspect : % -> S
latex : % -> String if S has SETCAT
less? : (%,NonNegativeInteger) -> Boolean
map : ((S -> S),%) -> %
map! : ((S -> S),%) -> %
    if $ has shallowlyMutable
members : % -> List S if $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
parts : % -> List S if $ has finiteAggregate
reduce : (((S,S) -> S),%) -> S
    if $ has finiteAggregate
reduce : (((S,S) -> S),%,S) -> S
    if $ has finiteAggregate
reduce : (((S,S) -> S),%,S,S) -> S
    if S has SETCAT and $ has finiteAggregate
remove : ((S -> Boolean),%) -> %
    if $ has finiteAggregate
remove : (S,%) -> %
    if S has SETCAT and $ has finiteAggregate
remove! : ((S -> Boolean),%) -> %
    if $ has finiteAggregate
remove! : (S,%) -> % if $ has finiteAggregate
removeDuplicates : % -> %
    if S has SETCAT and $ has finiteAggregate
sample : () -> %
select : ((S -> Boolean),%) -> %
    if $ has finiteAggregate
select! : ((S -> Boolean),%) -> %
    if $ has finiteAggregate
size? : (%,NonNegativeInteger) -> Boolean
#? : % -> NonNegativeInteger
    if $ has finiteAggregate
?=? : (%,%) -> Boolean if S has SETCAT
?~=? : (%,%) -> Boolean if S has SETCAT
```

— category KDAGG KeyedDictionary —

```
)abbrev category KDAGG KeyedDictionary
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
```

```
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A keyed dictionary is a dictionary of key-entry pairs for which there is
++ a unique entry for each key.

KeyedDictionary(Key:SetCategory, Entry:SetCategory): Category ==
  Dictionary Record(key:Key,entry:Entry) with
   key?: (Key, %) -> Boolean
     ++ key?(k,t) tests if k is a key in table t.
   keys: % -> List Key
     ++ keys(t) returns the list the keys in table t.
   -- to become keys: % -> Key* and keys: % -> Iterator(Entry,Entry)
   remove_!: (Key, %) -> Union(Entry,"failed")
     ++ remove!(k,t) searches the table t for the key k removing
     ++ (and return) the entry if there.
     ++ If t has no such key, \axiom{remove!(k,t)} returns "failed".
   search: (Key, %) -> Union(Entry,"failed")
     ++ search(k,t) searches the table t for the key k,
     ++ returning the entry stored in t for key k.
     ++ If t has no such key, \axiom{search(k,t)} returns "failed".
 add
   key?(k, t) == search(k, t) case Entry

   member?(p, t) ==
     r := search(p.key, t)
     r case Entry and r::Entry = p.entry

   if % has finiteAggregate then
     keys t == [x.key for x in parts t]
```

———-

— KDAGG.dotabb —

```
"KDAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=KDAGG"];
"KDAGG" -> "DIAGG"
```

———-

— KDAGG.dotfull —

```
"KeyedDictionary(a:SetCategory,b:SetCategory)"
```

```
 [color=lightblue,href="bookvol10.2.pdf#nameddest=KDAGG"];
"KeyedDictionary(a:SetCategory,b:SetCategory)" ->
    "Dictionary(Record(a:SetCategory,b:SetCategory))"
```

———-

— KDAGG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"KeyedDictionary(a:SetCategory,b:SetCategory)" [color=lightblue];
"KeyedDictionary(a:SetCategory,b:SetCategory)" ->
    "Dictionary(Record(a:SetCategory,b:SetCategory))"

"Dictionary(Record(a:SetCategory,b:SetCategory))" [color=seagreen];
"Dictionary(Record(a:SetCategory,b:SetCategory))" ->
    "Dictionary(a:SetCategory)"

"Dictionary(a:SetCategory)" [color=lightblue];
"Dictionary(a:SetCategory)" -> "DictionaryOperations(a:SetCategory)"

"DictionaryOperations(a:SetCategory)" [color=lightblue];
"DictionaryOperations(a:SetCategory)" -> "BagAggregate(a:SetCategory)"
"DictionaryOperations(a:SetCategory)" -> "Collection(a:SetCategory)"

"BagAggregate(a:SetCategory)" [color=seagreen];
"BagAggregate(a:SetCategory)" -> "BagAggregate(a:Type)"

"BagAggregate(a:Type)" [color=lightblue];
"BagAggregate(a:Type)" -> "HOAGG..."

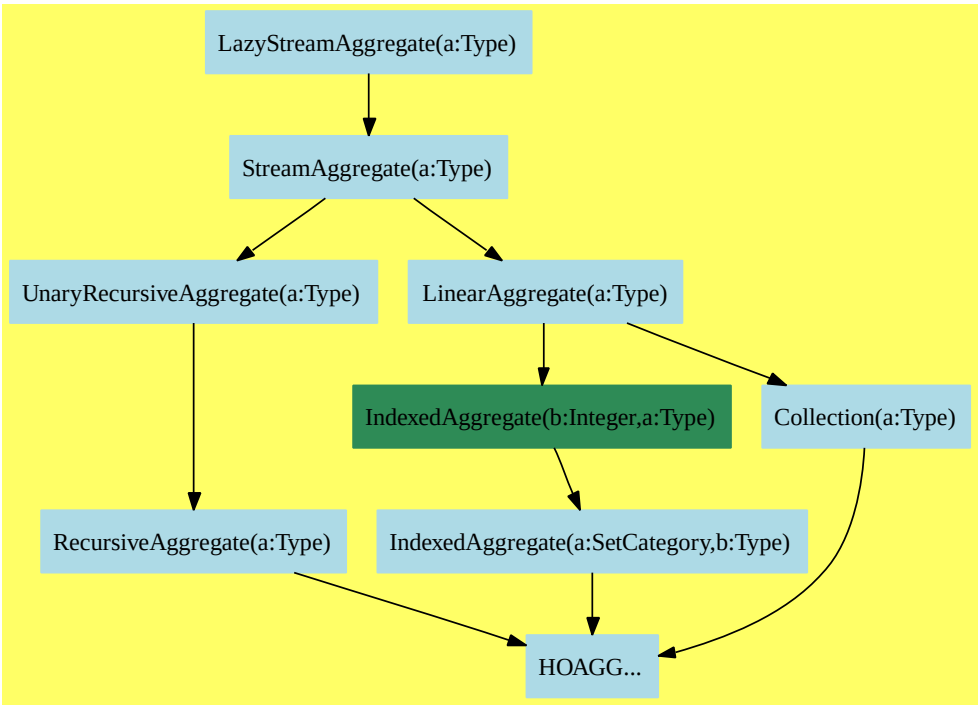
"Collection(a:SetCategory)" [color=seagreen];
"Collection(a:SetCategory)" -> "Collection(a:Type)"

"Collection(a:Type)" [color=lightblue];
"Collection(a:Type)" -> "HOAGG..."

"HOAGG..." [color=lightblue];
}
```

———-

8.0.112 LazyStreamAggregate (LZSTAGG)



— LazyStreamAggregate.input —

```
)set break resume
)sys rm -f LazyStreamAggregate.output
)spool LazyStreamAggregate.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show LazyStreamAggregate
--R LazyStreamAggregate S: Type  is a category constructor
--R Abbreviation for LazyStreamAggregate is LZSTAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for LZSTAGG
--R
--R------------------------------ Operations --------------------------------
--R children : % -> List %                complete : % -> %
--R concat : (%,S) -> %                   concat : List % -> %
--R concat : (S,%) -> %                   concat : (%,%) -> %
--R construct : List S -> %               copy : % -> %
```

```
--R cycleEntry : % -> %                  cycleTail : % -> %
--R cyclic? : % -> Boolean               delete : (%,Integer) -> %
--R distance : (%,%) -> Integer          elt : (%,Integer,S) -> S
--R ?.? : (%,Integer) -> S               ?.last : (%,last) -> S
--R ?.rest : (%,rest) -> %               ?.first : (%,first) -> S
--R ?.value : (%,value) -> S             empty : () -> %
--R empty? : % -> Boolean                entries : % -> List S
--R eq? : (%,%) -> Boolean               explicitEntries? : % -> Boolean
--R explicitlyEmpty? : % -> Boolean      explicitlyFinite? : % -> Boolean
--R extend : (%,Integer) -> %            first : % -> S
--R frst : % -> S                        index? : (Integer,%) -> Boolean
--R indices : % -> List Integer          insert : (S,%,Integer) -> %
--R insert : (%,%,Integer) -> %          last : % -> S
--R lazy? : % -> Boolean                 lazyEvaluate : % -> %
--R leaf? : % -> Boolean                 leaves : % -> List S
--R map : (((S,S) -> S),%,%) -> %        map : ((S -> S),%) -> %
--R new : (NonNegativeInteger,S) -> %    nodes : % -> List %
--R possiblyInfinite? : % -> Boolean     qelt : (%,Integer) -> S
--R remove : ((S -> Boolean),%) -> %     rest : % -> %
--R rst : % -> %                         sample : () -> %
--R second : % -> S                      select : ((S -> Boolean),%) -> %
--R tail : % -> %                        third : % -> S
--R value : % -> S
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R child? : (%,%) -> Boolean if S has SETCAT
--R coerce : % -> OutputForm if S has SETCAT
--R concat! : (%,S) -> % if $ has shallowlyMutable
--R concat! : (%,%) -> % if $ has shallowlyMutable
--R convert : % -> InputForm if S has KONVERT INFORM
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R cycleLength : % -> NonNegativeInteger
--R cycleSplit! : % -> % if $ has shallowlyMutable
--R delete : (%,UniversalSegment Integer) -> %
--R ?.? : (%,UniversalSegment Integer) -> %
--R entry? : (S,%) -> Boolean if $ has finiteAggregate and S has SETCAT
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R fill! : (%,S) -> % if $ has shallowlyMutable
--R find : ((S -> Boolean),%) -> Union(S,"failed")
--R first : (%,NonNegativeInteger) -> %
--R hash : % -> SingleInteger if S has SETCAT
--R last : (%,NonNegativeInteger) -> %
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
```

```
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R maxIndex : % -> Integer if Integer has ORDSET
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R minIndex : % -> Integer if Integer has ORDSET
--R more? : (%,NonNegativeInteger) -> Boolean
--R node? : (%,%) -> Boolean if S has SETCAT
--R numberOfComputedEntries : % -> NonNegativeInteger
--R parts : % -> List S if $ has finiteAggregate
--R qsetelt! : (%,Integer,S) -> S if $ has shallowlyMutable
--R reduce : (((S,S) -> S),%,S,S) -> S if S has SETCAT and $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%) -> S if $ has finiteAggregate
--R remove : (S,%) -> % if S has SETCAT and $ has finiteAggregate
--R removeDuplicates : % -> % if S has SETCAT and $ has finiteAggregate
--R rest : (%,NonNegativeInteger) -> %
--R setchildren! : (%,List %) -> % if $ has shallowlyMutable
--R setelt : (%,Integer,S) -> S if $ has shallowlyMutable
--R setelt : (%,UniversalSegment Integer,S) -> S if $ has shallowlyMutable
--R setelt : (%,last,S) -> S if $ has shallowlyMutable
--R setelt : (%,rest,%) -> % if $ has shallowlyMutable
--R setelt : (%,first,S) -> S if $ has shallowlyMutable
--R setelt : (%,value,S) -> S if $ has shallowlyMutable
--R setfirst! : (%,S) -> S if $ has shallowlyMutable
--R setlast! : (%,S) -> S if $ has shallowlyMutable
--R setrest! : (%,%) -> % if $ has shallowlyMutable
--R setvalue! : (%,S) -> S if $ has shallowlyMutable
--R size? : (%,NonNegativeInteger) -> Boolean
--R split! : (%,Integer) -> % if $ has shallowlyMutable
--R swap! : (%,Integer,Integer) -> Void if $ has shallowlyMutable
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— LazyStreamAggregate.help —

```
=====================================================================
LazyStreamAggregate examples
=====================================================================

See Also:
o )show LazyStreamAggregate
```

———-

See:

⇐ "StreamAggregate" (STAGG) 7.0.107 on page 585

Exports:

any?	child?	children	coerce
complete	concat	concat!	construct
copy	convert	count	cycleEntry
cycleLength	cycleSplit!	cycleTail	cyclic?
delete	distance	elt	empty
empty?	entry?	entries	eq?
explicitEntries?	explicitlyEmpty?	explicitlyFinite?	extend
eval	every?	fill!	find
first	frst	hash	index?
indices	insert	last	latex
lazy?	lazyEvaluate	leaf?	leaves
less?	map	map!	maxIndex
member?	members	minIndex	more?
new	node?	nodes	numberOfComputedEntries
parts	possiblyInfinite?	qelt	qsetelt!
reduce	remove	removeDuplicates	rest
rst	sample	second	select
setchildren!	setelt	setfirst!	setlast!
setrest!	setvalue!	size?	split!
swap!	tail	third	value
#?	?=?	?.?	?.last
?.rest	?.first	?.value	?~=?

Attributes exported:

- **nil**

These are directly exported but not implemented:

```
explicitEntries? : % -> Boolean
explicitlyEmpty? : % -> Boolean
frst : % -> S
lazy? : % -> Boolean
lazyEvaluate : % -> %
numberOfComputedEntries : % -> NonNegativeInteger
remove : ((S -> Boolean),%) -> %
rst : % -> %
select : ((S -> Boolean),%) -> %
```

These are implemented by this category:

```
any? : ((S -> Boolean),%) -> Boolean
```

```
    if $ has finiteAggregate
child? : (%,%) -> Boolean if S has SETCAT
children : % -> List %
complete : % -> %
construct : List S -> %
cycleEntry : % -> %
cycleLength : % -> NonNegativeInteger
cycleTail : % -> %
cyclic? : % -> Boolean
delete : (%,Integer) -> %
delete : (%,UniversalSegment Integer) -> %
distance : (%,%) -> Integer
elt : (%,Integer,S) -> S
entries : % -> List S
every? : ((S -> Boolean),%) -> Boolean
    if $ has finiteAggregate
explicitlyFinite? : % -> Boolean
extend : (%,Integer) -> %
first : (%,NonNegativeInteger) -> %
index? : (Integer,%) -> Boolean
indices : % -> List Integer
insert : (S,%,Integer) -> %
insert : (%,%,Integer) -> %
last : % -> S
last : (%,NonNegativeInteger) -> %
leaf? : % -> Boolean
less? : (%,NonNegativeInteger) -> Boolean
maxIndex : % -> Integer if Integer has ORDSET
minIndex : % -> Integer if Integer has ORDSET
more? : (%,NonNegativeInteger) -> Boolean
node? : (%,%) -> Boolean if S has SETCAT
nodes : % -> List %
possiblyInfinite? : % -> Boolean
rest : % -> %
rest : (%,NonNegativeInteger) -> %
value : % -> S
size? : (%,NonNegativeInteger) -> Boolean
tail : % -> %
#? : % -> NonNegativeInteger if $ has finiteAggregate
?=? : (%,%) -> Boolean if S has SETCAT
?.? : (%,UniversalSegment Integer) -> %
?.? : (%,Integer) -> S
?.first : (%,first) -> S
?.last : (%,last) -> S
?.rest : (%,rest) -> %
```

These exports come from (p585) StreamAggregate(S:Type):

```
coerce : % -> OutputForm if S has SETCAT
concat : (%,%) -> %
```

```
concat : (%,S) -> %
concat : (S,%) -> %
concat : List % -> %
concat! : (%,%) -> % if $ has shallowlyMutable
concat! : (%,S) -> % if $ has shallowlyMutable
convert : % -> InputForm if S has KONVERT INFORM
copy : % -> %
count : (S,%) -> NonNegativeInteger
    if S has SETCAT
    and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
    if $ has finiteAggregate
cycleSplit! : % -> % if $ has shallowlyMutable
empty : () -> %
empty? : % -> Boolean
entry? : (S,%) -> Boolean
    if $ has finiteAggregate
    and S has SETCAT
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> %
    if S has EVALAB S
    and S has SETCAT
eval : (%,S,S) -> %
    if S has EVALAB S
    and S has SETCAT
eval : (%,Equation S) -> %
    if S has EVALAB S
    and S has SETCAT
eval : (%,List Equation S) -> %
    if S has EVALAB S
    and S has SETCAT
fill! : (%,S) -> % if $ has shallowlyMutable
find : ((S -> Boolean),%) -> Union(S,"failed")
first : % -> S
hash : % -> SingleInteger if S has SETCAT
latex : % -> String if S has SETCAT
leaves : % -> List S
map : (((S,S) -> S),%,%) -> %
map : ((S -> S),%) -> %
map! : ((S -> S),%) -> % if $ has shallowlyMutable
member? : (S,%) -> Boolean
    if S has SETCAT
    and $ has finiteAggregate
members : % -> List S if $ has finiteAggregate
new : (NonNegativeInteger,S) -> %
parts : % -> List S if $ has finiteAggregate
qelt : (%,Integer) -> S
qsetelt! : (%,Integer,S) -> S if $ has shallowlyMutable
reduce : (((S,S) -> S),%,S,S) -> S
    if S has SETCAT
```

```
    and $ has finiteAggregate
 reduce : (((S,S) -> S),%,S) -> S if $ has finiteAggregate
 reduce : (((S,S) -> S),%) -> S if $ has finiteAggregate
 remove : (S,%) -> % if S has SETCAT and $ has finiteAggregate
 removeDuplicates : % -> %
    if S has SETCAT
    and $ has finiteAggregate
 sample : () -> %
 second : % -> S
 setchildren! : (%,List %) -> % if $ has shallowlyMutable
 setelt : (%,Integer,S) -> S if $ has shallowlyMutable
 setelt : (%,UniversalSegment Integer,S) -> S
    if $ has shallowlyMutable
 setelt : (%,last,S) -> S if $ has shallowlyMutable
 setelt : (%,rest,%) -> % if $ has shallowlyMutable
 setelt : (%,first,S) -> S if $ has shallowlyMutable
 setelt : (%,value,S) -> S if $ has shallowlyMutable
 setfirst! : (%,S) -> S if $ has shallowlyMutable
 setlast! : (%,S) -> S if $ has shallowlyMutable
 setrest! : (%,%) -> % if $ has shallowlyMutable
 setvalue! : (%,S) -> S if $ has shallowlyMutable
 split! : (%,Integer) -> % if $ has shallowlyMutable
 swap! : (%,Integer,Integer) -> Void if $ has shallowlyMutable
 third : % -> S
 ?.value : (%,value) -> S
 ?~=? : (%,%) -> Boolean if S has SETCAT
```

— category LZSTAGG LazyStreamAggregate —

```
)abbrev category LZSTAGG LazyStreamAggregate
++ Category of streams with lazy evaluation
++ Author: Clifton J. Williamson
++ Date Created: 22 November 1989
++ Date Last Updated: 20 July 1990
++ Keywords: stream, infinite list, infinite sequence
++ Description:
++ LazyStreamAggregate is the category of streams with lazy
++ evaluation.  It is understood that the function 'empty?' will
++ cause lazy evaluation if necessary to determine if there are
++ entries.  Functions which call 'empty?', e.g. 'first' and 'rest',
++ will also cause lazy evaluation if necessary.

LazyStreamAggregate(S:Type): Category == StreamAggregate(S) with
  remove: (S -> Boolean,%) -> %
    ++ remove(f,st) returns a stream consisting of those elements of stream
    ++ st which do not satisfy the predicate f.
    ++ Note that \spad{remove(f,st) = [x for x in st | not f(x)]}.
    ++
    ++X m:=[i for i in 1..]
```

```
  ++X f(i:PositiveInteger):Boolean == even? i
  ++X remove(f,m)

select: (S -> Boolean,%) -> %
  ++ select(f,st) returns a stream consisting of those elements of stream
  ++ st satisfying the predicate f.
  ++ Note that \spad{select(f,st) = [x for x in st | f(x)]}.
  ++
  ++X m:=[i for i in 0..]
  ++X select(x+->prime? x,m)

explicitEntries?: % -> Boolean
  ++ explicitEntries?(s) returns true if the stream s has
  ++ explicitly computed entries, and false otherwise.
  ++
  ++X m:=[i for i in 0..]
  ++X explicitEntries? m

explicitlyEmpty?: % -> Boolean
  ++ explicitlyEmpty?(s) returns true if the stream is an
  ++ (explicitly) empty stream.
  ++ Note that this is a null test which will not cause lazy evaluation.
  ++
  ++X m:=[i for i in 0..]
  ++X explicitlyEmpty? m

lazy?: % -> Boolean
  ++ lazy?(s) returns true if the first node of the stream s
  ++ is a lazy evaluation mechanism which could produce an
  ++ additional entry to s.
  ++
  ++X m:=[i for i in 0..]
  ++X lazy? m

lazyEvaluate: % -> %
  ++ lazyEvaluate(s) causes one lazy evaluation of stream s.
  ++ Caution: the first node must be a lazy evaluation mechanism
  ++ (satisfies \spad{lazy?(s) = true}) as there is no error check.
  ++ Note that a call to this function may
  ++ or may not produce an explicit first entry
frst: % -> S
  ++ frst(s) returns the first element of stream s.
  ++ Caution: this function should only be called after a \spad{empty?}
  ++ test has been made since there no error check.
  ++
  ++X m:=[i for i in 0..]
  ++X frst m

rst: % -> %
  ++ rst(s) returns a pointer to the next node of stream s.
```

```
  ++ Caution: this function should only be called after a \spad{empty?}
  ++ test has been made since there no error check.
  ++
  ++X m:=[i for i in 0..]
  ++X rst m

 numberOfComputedEntries: % -> NonNegativeInteger
  ++ numberOfComputedEntries(st) returns the number of explicitly
  ++ computed entries of stream st which exist immediately prior to the
  ++ time this function is called.
  ++
  ++X m:=[i for i in 0..]
  ++X numberOfComputedEntries m

 extend: (%,Integer) -> %
  ++ extend(st,n) causes entries to be computed, if necessary,
  ++ so that 'st' will have at least 'n' explicit entries or so
  ++ that all entries of 'st' will be computed if 'st' is finite
  ++ with length <= n.
  ++
  ++X m:=[i for i in 0..]
  ++X numberOfComputedEntries m
  ++X extend(m,20)
  ++X numberOfComputedEntries m

 complete: % -> %
  ++ complete(st) causes all entries of 'st' to be computed.
  ++ this function should only be called on streams which are
  ++ known to be finite.
  ++
  ++X m:=[i for i in 1..]
  ++X n:=filterUntil(i+->i>100,m)
  ++X numberOfComputedEntries n
  ++X complete n
  ++X numberOfComputedEntries n

add

 MIN ==> 1  -- minimal stream index

 I   ==> Integer
 NNI ==> NonNegativeInteger
 L   ==> List
 U   ==> UniversalSegment Integer

 indexx?            : (Integer,%) -> Boolean
 cycleElt           : % -> Union(%,"failed")
 computeCycleLength : % -> NNI
 computeCycleEntry  : (%,%) -> %
```

```
--% SETCAT functions

  if S has SetCategory then

    x = y ==
      eq?(x,y) => true
      explicitlyFinite? x and explicitlyFinite? y =>
        entries x = entries y
      explicitEntries? x and explicitEntries? y =>
        frst x = frst y and EQ(rst x, rst y)$Lisp
      -- treat cyclic streams
      false

--% HOAGG functions

  --null x == empty? x

  less?(x,n) ==
    n = 0    => false
    empty? x => true
    less?(rst x,(n-1) :: NNI)

  more?(x,n) ==
    empty? x => false
    n = 0    => true
    more?(rst x,(n-1) :: NNI)

  size?(x,n) ==
    empty? x => n = 0
    size?(rst x,(n-1) :: NNI)

  # x ==
    -- error if stream is not finite
    y := x
    for i in 0.. repeat
      explicitlyEmpty? y  => return i
      lazy? y => error "#: infinite stream"
      y := rst y
      if odd? i then x := rst x
      eq?(x,y) => error "#: infinite stream"

--% CLAGG functions

  any?(f,x) ==
    -- error message only when x is a stream with lazy
    -- evaluation and f(s) = false for all stream elements
    -- 's' which have been computed when the function is
    -- called
    y := x
    for i in 0.. repeat
```

```
      explicitlyEmpty? y  => return false
      lazy? y => error "any?: infinite stream"
      f frst y => return true
      y := rst y
      if odd? i then x := rst x
      eq?(x,y) => return false

  every?(f,x) ==
    -- error message only when x is a stream with lazy
    -- evaluation and f(s) = true for all stream elements
    -- 's' which have been computed when the function is
    -- called
    y := x
    for i in 0.. repeat
      explicitlyEmpty? y => return true
      lazy? y => error "every?: infinite stream"
      not f frst y => return false
      y := rst y
      if odd? i then x := rst x
      eq?(x,y) => return true

-- following ops count and member? are only exported if $ has finiteAggregate

--  count(f:S -> Boolean,x:%) ==
--    -- error if stream is not finite
--    count : NNI := 0
--    y := x
--    for i in 0.. repeat
--      explicitlyEmpty? y  => return count
--      lazy? y => error "count: infinite stream"
--      if f frst y then count := count + 1
--      y := rst y
--      if odd? i then x := rst x
--      eq?(x,y) => error "count: infinite stream"


--  if S has SetCategory then

--      count(s:S,x:%) == count(#1 = s,x)
--        -- error if stream is not finite

--    member?(s,x) ==
--      -- error message only when x is a stream with lazy
--      -- evaluation and 's' is not among the stream elements
--      -- which have been computed when the function is called
--      y := x
--      for i in 0.. repeat
--        explicitlyEmpty? y  => return false
--        lazy? y => error "member?: infinite stream"
--        frst y = s => return true
```

```
--        y := rst y
--        if odd? i then x := rst x
--        eq?(x,y) => return false

  entries x ==
    -- returns a list of elements which have been computed
    -- error if infinite
    y := x
    l : L S := empty()
    for i in 0.. repeat
      explicitlyEmpty? y  => return reverse_! l
      lazy? y => error "infinite stream"
      l := concat(frst y,l)
      y := rst y
      if odd? i then x := rst x
      eq?(x,y) => error "infinite stream"

--% CNAGG functions

  construct l ==
    empty? l => empty()
    concat(first l, construct rest l)

  --entries x ==
    -- returns a list of the stream elements
    -- error if the stream is not finite
    --members x

--% ELTAGG functions

  elt(x:%,n:I) ==
    n < MIN or empty? x => error "elt: no such element"
    n = MIN => frst x
    elt(rst x,n - 1)

  elt(x:%,n:I,s:S) ==
    n < MIN or empty? x => s
    n = MIN => frst x
    elt(rst x,n - 1)

--% IXAGG functions

-- following assumes % has finiteAggregate and S has SetCategory
--  entry?(s,x) ==
--    -- error message only when x is a stream with lazy
--    -- evaluation and 's' is not among the stream elements
--    -- which have been computed when the function is called
--    member?(s,x)

  --entries x ==
```

```
    -- error if the stream is not finite
    --members x

  indexx?(n,x) ==
    empty? x => false
    n = MIN => true
    indexx?(n-1,rst x)

  index?(n,x) ==
    -- returns 'true' iff 'n' is the index of an entry which
    -- may or may not have been computed when the function is
    -- called
    -- additional entries are computed if necessary
    n < MIN => false
    indexx?(n,x)

  indices x ==
    -- error if stream is not finite
    y := x
    l : L I := empty()
    for i in MIN.. repeat
      explicitlyEmpty? y  => return reverse_! l
      lazy? y => error "indices: infinite stream"
      l := concat(i,l)
      y := rst y
      if odd? i then x := rst x
      eq?(x,y) => error "indices: infinite stream"

  maxIndex x ==
    -- error if stream is not finite
    empty? x =>
      error "maxIndex: no maximal index for empty stream"
    y := rst x
    for i in MIN.. repeat
      explicitlyEmpty? y  => return i
      lazy? y => error "maxIndex: infinite stream"
      y := rst y
      if odd? i then x := rst x
      eq?(x,y) => error "maxIndex: infinite stream"

  minIndex x ==
    empty? x => error "minIndex: no minimal index for empty stream"
    MIN

--% LNAGG functions

  delete(x:%,n:I) ==
  -- non-destructive
    not index?(n,x) => error "delete: index out of range"
    concat(first(x,(n - MIN) :: NNI), rest(x,(n - MIN + 1) :: NNI))
```

```
  delete(x:%,seg:U) ==
    low := lo seg
    hasHi seg =>
      high := hi seg
      high < low => copy x
      (not index?(low,x)) or (not index?(high,x)) =>
        error "delete: index out of range"
      concat(first(x,(low - MIN) :: NNI),rest(x,(high - MIN + 1) :: NNI))
    not index?(low,x) => error "delete: index out of range"
    first(x,(low - MIN) :: NNI)

  elt(x:%,seg:U) ==
    low := lo seg
    hasHi seg =>
      high := hi seg
      high < low => empty()
      (not index?(low,x)) or (not index?(high,x)) =>
        error "elt: index out of range"
      first(rest(x,(low - MIN) :: NNI),(high - low + 1) :: NNI)
    not index?(low,x) => error "elt: index out of range"
    rest(x,(low - MIN) :: NNI)

  insert(s:S,x:%,n:I) ==
    not index?(n,x) => error "insert: index out of range"
    nn := (n - MIN) :: NNI
    concat([first(x,nn), concat(s, empty()), rest(x,nn)])

  insert(y:%,x:%,n:I) ==
    not index?(n,x) => error "insert: index out of range"
    nn := (n - MIN) :: NNI
    concat([first(x,nn), y, rest(x,nn)])

--% RCAGG functions

  cycleElt x == cycleElt(x)$CyclicStreamTools(S,%)

  cyclic? x ==
    cycleElt(x) case "failed" => false
    true

  if S has SetCategory then
    child?(x,y) ==
      empty? y => error "child: no children"
      x = rst y

  children x ==
    empty? x => error "children: no children"
    [rst x]
```

```
  distance(x,z) ==
    y := x
    for i in 0.. repeat
      eq?(y,z) => return i
      (explicitlyEmpty? y) or (lazy? y) =>
        error "distance: 2nd arg not a descendent of the 1st"
      y := rst y
      if odd? i then x := rst x
      eq?(x,y) =>
        error "distance: 2nd arg not a descendent of the 1st"

  if S has SetCategory then
    node?(z,x) ==
      -- error message only when x is a stream with lazy
      -- evaluation and 'y' is not a node of 'x'
      -- which has been computed when the function is called
      y := x
      for i in 0.. repeat
        z = y => return true
        explicitlyEmpty? y => return false
        lazy? y => error "node?: infinite stream"
        y := rst y
        if odd? i then x := rst x
        eq?(x,y) => return false

  nodes x ==
    y := x
    l : L % := []
    for i in 0.. repeat
      explicitlyEmpty? y => return reverse_! l
      lazy? y => error "nodes: infinite stream"
      l := concat(y,l)
      y := rst y
      if odd? i then x := rst x
      eq?(x,y) => error "nodes: infinite stream"
    l -- @#$%^& compiler

  leaf? x == empty? rest x

  value x == first x

--% URAGG functions

  computeCycleLength cycElt ==
    computeCycleLength(cycElt)$CyclicStreamTools(S,%)

  computeCycleEntry(x,cycElt) ==
    computeCycleEntry(x,cycElt)$CyclicStreamTools(S,%)

  cycleEntry x ==
```

```
    cycElt := cycleElt x
    cycElt case "failed" =>
      error "cycleEntry: non-cyclic stream"
    computeCycleEntry(x,cycElt::%)

  cycleLength x ==
    cycElt := cycleElt x
    cycElt case "failed" =>
      error "cycleLength: non-cyclic stream"
    computeCycleLength(cycElt::%)

  cycleTail x ==
    cycElt := cycleElt x
    cycElt case "failed" =>
      error "cycleTail: non-cyclic stream"
    y := x := computeCycleEntry(x,cycElt::%)
    z := rst x
    repeat
      eq?(x,z) => return y
      y := z ; z := rst z

  elt(x,"first") == first x

  first(x,n) ==
  -- former name: take
    n = 0 or empty? x => empty()
    concat(frst x, first(rst x,(n-1) :: NNI))

  rest x ==
    empty? x => error "Can't take the rest of an empty stream."
    rst x

  elt(x,"rest") == rest x

  rest(x,n) ==
  -- former name: drop
    n = 0 or empty? x => x
    rest(rst x,(n-1) :: NNI)

  last x ==
    -- error if stream is not finite
    empty? x => error "last: empty stream"
    y1 := x
    y2 := rst x
    for i in 0.. repeat
      explicitlyEmpty? y2 => return frst y1
      lazy? y2 => error "last: infinite stream"
      y1 := y2
      y2 := rst y2
      if odd? i then x := rst x
```

```
      eq?(x,y2) => error "last: infinite stream"

  if % has finiteAggregate then -- # is only defined for finiteAggregates
    last(x,n) ==
      possiblyInfinite? x => error "last: infinite stream"
      m := # x
      m < n => error "last: index out of range"
      copy rest(x,(m-n)::NNI)

  elt(x,"last") == last x

  tail x ==
    -- error if stream is not finite
    empty? x => error "tail: empty stream"
    y1 := x
    y2 := rst x
    for i in 0.. repeat
      explicitlyEmpty? y2 => return y1
      lazy? y2 => error "tail: infinite stream"
      y1 := y2
      y2 := rst y2
      if odd? i then x := rst x
      eq?(x,y2) => error "tail: infinite stream"

--% STAGG functions

  possiblyInfinite? x ==
    y := x
    for i in 0.. repeat
      explicitlyEmpty? y  => return false
      lazy? y => return true
      if odd? i then x := rst x
      y := rst y
      eq?(x,y) => return true

  explicitlyFinite? x == not possiblyInfinite? x

--% LZSTAGG functions

  extend(x,n) ==
    y := x
    for i in 1..n while not empty? y repeat y := rst y
    x

  complete x ==
    y := x
    while not empty? y repeat y := rst y
    x
```

———-

— **LZSTAGG.dotabb** —

```
"LZSTAGG"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=LZSTAGG"];
"LZSTAGG" -> "STAGG"
```

———-

— **LZSTAGG.dotfull** —

```
"LazyStreamAggregate(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=LZSTAGG"];
"LazyStreamAggregate(a:Type)" -> "StreamAggregate(a:Type)"
```

———-

— **LZSTAGG.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"LazyStreamAggregate(a:Type)" [color=lightblue];
"LazyStreamAggregate(a:Type)" -> "StreamAggregate(a:Type)"

"StreamAggregate(a:Type)" [color=lightblue];
"StreamAggregate(a:Type)" -> "UnaryRecursiveAggregate(a:Type)"
"StreamAggregate(a:Type)" -> "LinearAggregate(a:Type)"

"UnaryRecursiveAggregate(a:Type)" [color=lightblue];
"UnaryRecursiveAggregate(a:Type)" -> "RecursiveAggregate(a:Type)"

"RecursiveAggregate(a:Type)" [color=lightblue];
"RecursiveAggregate(a:Type)" -> "HOAGG..."

"LinearAggregate(a:Type)" [color=lightblue];
"LinearAggregate(a:Type)" -> "IndexedAggregate(b:Integer,a:Type)"
"LinearAggregate(a:Type)" -> "Collection(a:Type)"

"IndexedAggregate(b:Integer,a:Type)" [color=seagreen];
"IndexedAggregate(b:Integer,a:Type)" ->
    "IndexedAggregate(a:SetCategory,b:Type)"
```

```
"IndexedAggregate(a:SetCategory,b:Type)" [color=lightblue];
"IndexedAggregate(a:SetCategory,b:Type)" -> "HOAGG..."

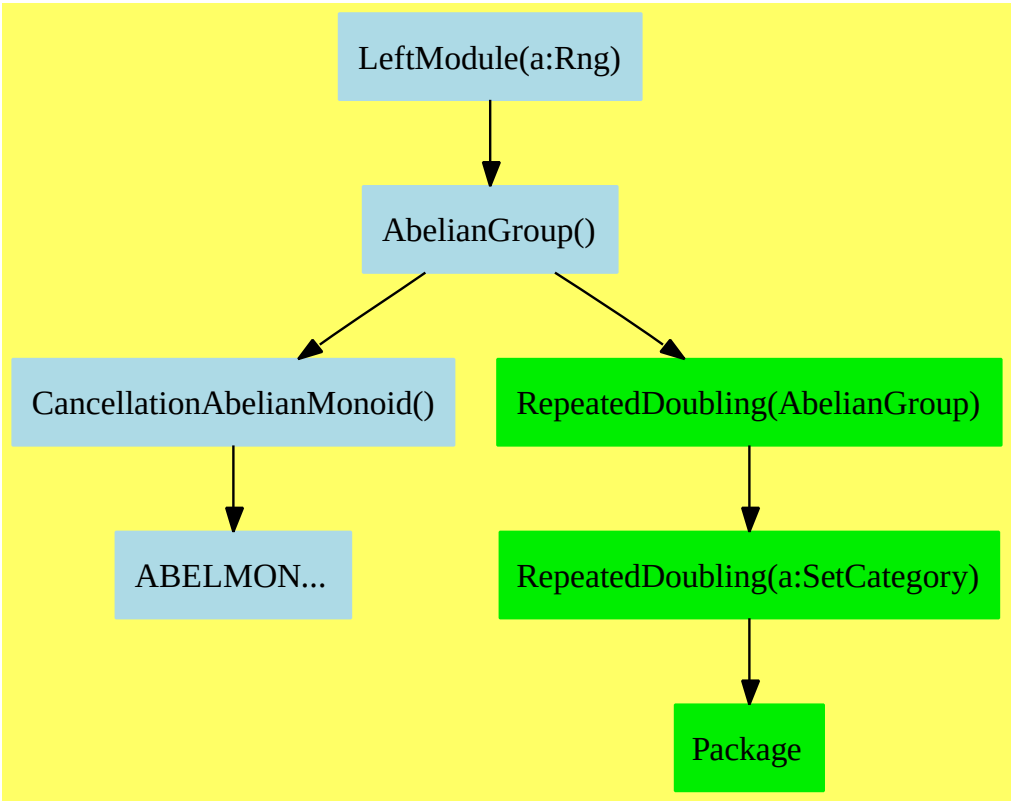
"Collection(a:Type)" [color=lightblue];
"Collection(a:Type)" -> "HOAGG..."

"HOAGG..." [color=lightblue];

}
```

———-

8.0.113 LeftModule (LMODULE)



— **LeftModule.input** —

```
)set break resume
)sys rm -f LeftModule.output
)spool LeftModule.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show LeftModule
--R LeftModule R: Rng  is a category constructor
--R Abbreviation for LeftModule is LMODULE
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for LMODULE
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (R,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?+? : (%,%) -> %
--R ?-? : (%,%) -> %                      -? : % -> %
--R ?=? : (%,%) -> Boolean                0 : () -> %
--R coerce : % -> OutputForm              hash : % -> SingleInteger
--R latex : % -> String                   sample : () -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— LeftModule.help —

```
=====================================================================
LeftModule examples
=====================================================================

See Also:
o )show LeftModule
```

———-

See:

⇒ “BiModule” (BMODULE) 9.0.122 on page 726
⇒ “LeftAlgebra” (LALG) 10.0.141 on page 864
⇒ “Ring” (RING) 9.0.129 on page 770

⇐ "AbelianGroup" (ABELGRP) 7.0.97 on page 520

Exports:

0	coerce	hash	latex	sample
subtractIfCan	zero?	?~=?	?*?	?+?
?-?	-?	?=?		

These are directly exported but not implemented:

```
?*? : (R,%) -> %
```

These exports come from (p520) AbelianGroup():

```
0 : () -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?~=? : (%,%) -> Boolean
?=? : (%,%) -> Boolean
?+? : (%,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?*? : (Integer,%) -> %
?-? : (%,%) -> %
-? : % -> %
```

— category LMODULE LeftModule —

```
)abbrev category LMODULE LeftModule
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The category of left modules over an rng (ring not necessarily with unit).
++ This is an abelian group which supports left multiplation by elements of
++ the rng.
++
++ Axioms\br
++ \tab{5}\spad{ (a*b)*x = a*(b*x) }\br
```

```
++ \tab{5}\spad{ (a+b)*x = (a*x)+(b*x) }\br
++ \tab{5}\spad{ a*(x+y) = (a*x)+(a*y) }

LeftModule(R:Rng):Category == AbelianGroup with
   "*": (R,%) -> %
     ++ r*x returns the left multiplication of the module element x
     ++ by the ring element r.
```

———-

— **LMODULE.dotabb** —

```
"LMODULE"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=LMODULE"];
"LMODULE" -> "ABELGRP"
```

———-

— **LMODULE.dotfull** —

```
"LeftModule(a:Rng)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=LMODULE"];
"LeftModule(a:Rng)" -> "AbelianGroup()"

"LeftModule(a:Ring)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=LMODULE"];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"
```

———-

— **LMODULE.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "AbelianGroup()"

"AbelianGroup()" [color=lightblue];
"AbelianGroup()" -> "CancellationAbelianMonoid()"
"AbelianGroup()" -> "RepeatedDoubling(AbelianGroup)"

"RepeatedDoubling(AbelianGroup)" [color="#00EE00"];
```

```
"RepeatedDoubling(AbelianGroup)" -> "RepeatedDoubling(a:SetCategory)"

"RepeatedDoubling(a:SetCategory)" [color="#00EE00"];
"RepeatedDoubling(a:SetCategory)" -> "Package"

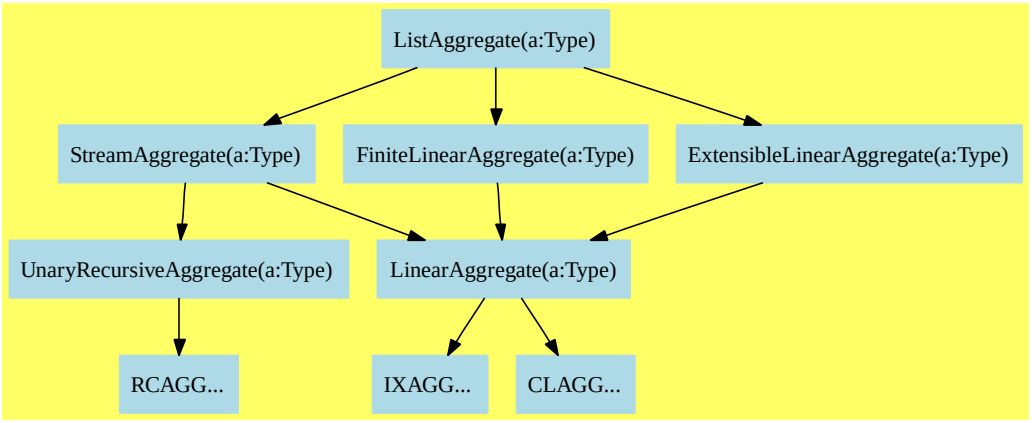
"Package" [color="#00EE00"];

"CancellationAbelianMonoid()" [color=lightblue];
"CancellationAbelianMonoid()" -> "ABELMON..."

"ABELMON..." [color=lightblue];
}
```

———-

8.0.114 ListAggregate (LSAGG)



— **ListAggregate.input** —

```
)set break resume
)sys rm -f ListAggregate.output
)spool ListAggregate.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show ListAggregate
--R ListAggregate S: Type  is a category constructor
```

```
--R Abbreviation for ListAggregate is LSAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for LSAGG
--R
--R------------------------------ Operations --------------------------------
--R children : % -> List %               concat : (%,S) -> %
--R concat : List % -> %                 concat : (S,%) -> %
--R concat : (%,%) -> %                  concat! : (%,S) -> %
--R concat! : (%,%) -> %                 construct : List S -> %
--R copy : % -> %                        cycleEntry : % -> %
--R cycleTail : % -> %                   cyclic? : % -> Boolean
--R delete : (%,Integer) -> %            delete! : (%,Integer) -> %
--R distance : (%,%) -> Integer          elt : (%,Integer,S) -> S
--R ?.? : (%,Integer) -> S               ?.last : (%,last) -> S
--R ?.rest : (%,rest) -> %               ?.first : (%,first) -> S
--R ?.value : (%,value) -> S             empty : () -> %
--R empty? : % -> Boolean                entries : % -> List S
--R eq? : (%,%) -> Boolean               explicitlyFinite? : % -> Boolean
--R first : % -> S                       index? : (Integer,%) -> Boolean
--R indices : % -> List Integer          insert : (S,%,Integer) -> %
--R insert : (%,%,Integer) -> %          insert! : (S,%,Integer) -> %
--R insert! : (%,%,Integer) -> %         last : % -> S
--R leaf? : % -> Boolean                 leaves : % -> List S
--R list : S -> %                        map : (((S,S) -> S),%,%) -> %
--R map : ((S -> S),%) -> %              new : (NonNegativeInteger,S) -> %
--R nodes : % -> List %                  possiblyInfinite? : % -> Boolean
--R qelt : (%,Integer) -> S              rest : % -> %
--R reverse : % -> %                     sample : () -> %
--R second : % -> S                      tail : % -> %
--R third : % -> S                       value : % -> S
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?<? : (%,%) -> Boolean if S has ORDSET
--R ?<=? : (%,%) -> Boolean if S has ORDSET
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R ?>? : (%,%) -> Boolean if S has ORDSET
--R ?>=? : (%,%) -> Boolean if S has ORDSET
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R child? : (%,%) -> Boolean if S has SETCAT
--R coerce : % -> OutputForm if S has SETCAT
--R convert : % -> InputForm if S has KONVERT INFORM
--R copyInto! : (%,%,Integer) -> % if $ has shallowlyMutable
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R cycleLength : % -> NonNegativeInteger
--R cycleSplit! : % -> % if $ has shallowlyMutable
--R delete : (%,UniversalSegment Integer) -> %
--R delete! : (%,UniversalSegment Integer) -> %
--R ?.? : (%,UniversalSegment Integer) -> %
--R entry? : (S,%) -> Boolean if $ has finiteAggregate and S has SETCAT
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
```

```
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R fill! : (%,S) -> % if $ has shallowlyMutable
--R find : ((S -> Boolean),%) -> Union(S,"failed")
--R first : (%,NonNegativeInteger) -> %
--R hash : % -> SingleInteger if S has SETCAT
--R last : (%,NonNegativeInteger) -> %
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R max : (%,%) -> % if S has ORDSET
--R maxIndex : % -> Integer if Integer has ORDSET
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R merge : (((S,S) -> Boolean),%,%) -> %
--R merge : (%,%) -> % if S has ORDSET
--R merge! : (((S,S) -> Boolean),%,%) -> %
--R merge! : (%,%) -> % if S has ORDSET
--R min : (%,%) -> % if S has ORDSET
--R minIndex : % -> Integer if Integer has ORDSET
--R more? : (%,NonNegativeInteger) -> Boolean
--R node? : (%,%) -> Boolean if S has SETCAT
--R parts : % -> List S if $ has finiteAggregate
--R position : ((S -> Boolean),%) -> Integer
--R position : (S,%) -> Integer if S has SETCAT
--R position : (S,%,Integer) -> Integer if S has SETCAT
--R qsetelt! : (%,Integer,S) -> S if $ has shallowlyMutable
--R reduce : (((S,S) -> S),%,S,S) -> S if S has SETCAT and $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%) -> S if $ has finiteAggregate
--R remove : (S,%) -> % if S has SETCAT and $ has finiteAggregate
--R remove : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R remove! : ((S -> Boolean),%) -> %
--R remove! : (S,%) -> % if S has SETCAT
--R removeDuplicates : % -> % if S has SETCAT and $ has finiteAggregate
--R removeDuplicates! : % -> % if S has SETCAT
--R rest : (%,NonNegativeInteger) -> %
--R reverse! : % -> % if $ has shallowlyMutable
--R select : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R select! : ((S -> Boolean),%) -> %
--R setchildren! : (%,List %) -> % if $ has shallowlyMutable
--R setelt : (%,Integer,S) -> S if $ has shallowlyMutable
--R setelt : (%,UniversalSegment Integer,S) -> S if $ has shallowlyMutable
--R setelt : (%,last,S) -> S if $ has shallowlyMutable
--R setelt : (%,rest,%) -> % if $ has shallowlyMutable
--R setelt : (%,first,S) -> S if $ has shallowlyMutable
--R setelt : (%,value,S) -> S if $ has shallowlyMutable
--R setfirst! : (%,S) -> S if $ has shallowlyMutable
```

```
--R setlast! : (%,S) -> S if $ has shallowlyMutable
--R setrest! : (%,%) -> % if $ has shallowlyMutable
--R setvalue! : (%,S) -> S if $ has shallowlyMutable
--R size? : (%,NonNegativeInteger) -> Boolean
--R sort : (((S,S) -> Boolean),%) -> %
--R sort : % -> % if S has ORDSET
--R sort! : (((S,S) -> Boolean),%) -> % if $ has shallowlyMutable
--R sort! : % -> % if S has ORDSET and $ has shallowlyMutable
--R sorted? : (((S,S) -> Boolean),%) -> Boolean
--R sorted? : % -> Boolean if S has ORDSET
--R split! : (%,Integer) -> % if $ has shallowlyMutable
--R swap! : (%,Integer,Integer) -> Void if $ has shallowlyMutable
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— **ListAggregate.help** —

```
=====================================================================
ListAggregate examples
=====================================================================

See Also:
o )show ListAggregate
```

———-

See:

⇒ “AssociationListAggregate” (ALAGG) 10.0.134 on page 819
⇐ “ExtensibleLinearAggregate” (ELAGG) 7.0.101 on page 547
⇐ “FiniteLinearAggregate” (FLAGG) 7.0.102 on page 555

Exports:

any?	children	child?	coerce
concat	concat!	construct	convert
copy	copyInto!	count	cycleEntry
cycleLength	cycleSplit!	cycleTail	cyclic?
delete	delete!	distance	elt
empty	empty?	entries	entry?
eq?	eval	every?	explicitlyFinite?
fill!	find	first	hash
index?	indices	insert	insert!
last	latex	leaf?	leaves
less?	list	map	map!
max	maxIndex	member?	members
merge	merge!	min	minIndex
more?	new	nodes	node?
parts	position	possiblyInfinite?	qelt
qsetelt!	reduce	remove	remove!
removeDuplicates	removeDuplicates!	rest	reverse
reverse!	sample	second	select
select!	setchildren!	setelt	setfirst!
setlast!	setrest!	setvalue!	size?
sort	sort!	sorted?	split!
swap!	tail	third	value
#?	?.?	?.last	?.rest
?.first	?.value	?<?	?<=?
?=?	?>?	?>=?	?~=?

Attributes exported:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **nil**

These are implemented by this category:

```
copy : % -> %
copyInto! : (%,%,Integer) -> % if $ has shallowlyMutable
delete! : (%,Integer) -> %
delete! : (%,UniversalSegment Integer) -> %
find : ((S -> Boolean),%) -> Union(S,"failed")
insert! : (S,%,Integer) -> %
insert! : (%,%,Integer) -> %
list : S -> %
map : (((S,S) -> S),%,%) -> %
merge : (((S,S) -> Boolean),%,%) -> %
```

```
merge! : (((S,S) -> Boolean),%,%) -> %
new : (NonNegativeInteger,S) -> %
position : ((S -> Boolean),%) -> Integer
position : (S,%,Integer) -> Integer if S has SETCAT
reduce : (((S,S) -> S),%,S) -> S
         if $ has finiteAggregate
reduce : (((S,S) -> S),%,S,S) -> S
         if S has SETCAT
         and $ has finiteAggregate
reduce : (((S,S) -> S),%) -> S
         if $ has finiteAggregate
remove! : ((S -> Boolean),%) -> %
removeDuplicates! : % -> % if S has SETCAT
reverse! : % -> % if $ has shallowlyMutable
select : ((S -> Boolean),%) -> %
         if $ has finiteAggregate
sort! : (((S,S) -> Boolean),%) -> %
         if $ has shallowlyMutable
sorted? : (((S,S) -> Boolean),%) -> Boolean
?<? : (%,%) -> Boolean if S has ORDSET
```

These exports come from (p585) StreamAggregate(S:Type):

```
any? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
children : % -> List %
child? : (%,%) -> Boolean if S has SETCAT
coerce : % -> OutputForm if S has SETCAT
concat : (%,S) -> %
concat : List % -> %
concat : (S,%) -> %
concat : (%,%) -> %
concat! : (%,S) -> %
concat! : (%,%) -> %
construct : List S -> %
convert : % -> InputForm if S has KONVERT INFORM
count : (S,%) -> NonNegativeInteger
         if S has SETCAT
         and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
cycleEntry : % -> %
cycleLength : % -> NonNegativeInteger
cycleSplit! : % -> % if $ has shallowlyMutable
cycleTail : % -> %
cyclic? : % -> Boolean
delete : (%,UniversalSegment Integer) -> %
delete : (%,Integer) -> %
distance : (%,%) -> Integer
elt : (%,Integer,S) -> S
```

```
empty : () -> %
empty? : % -> Boolean
entry? : (S,%) -> Boolean
         if $ has finiteAggregate
         and S has SETCAT
entries : % -> List S
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> %
         if S has EVALAB S
         and S has SETCAT
eval : (%,S,S) -> %
         if S has EVALAB S
         and S has SETCAT
eval : (%,Equation S) -> %
         if S has EVALAB S
         and S has SETCAT
eval : (%,List Equation S) -> %
         if S has EVALAB S
         and S has SETCAT
every? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
explicitlyFinite? : % -> Boolean
fill! : (%,S) -> % if $ has shallowlyMutable
first : % -> S
first : (%,NonNegativeInteger) -> %
hash : % -> SingleInteger if S has SETCAT
index? : (Integer,%) -> Boolean
indices : % -> List Integer
insert : (S,%,Integer) -> %
insert : (%,%,Integer) -> %
last : % -> S
last : (%,NonNegativeInteger) -> %
latex : % -> String if S has SETCAT
leaf? : % -> Boolean
leaves : % -> List S
less? : (%,NonNegativeInteger) -> Boolean
map : ((S -> S),%) -> %
map! : ((S -> S),%) -> % if $ has shallowlyMutable
maxIndex : % -> Integer if Integer has ORDSET
member? : (S,%) -> Boolean
         if S has SETCAT
         and $ has finiteAggregate
members : % -> List S if $ has finiteAggregate
minIndex : % -> Integer if Integer has ORDSET
more? : (%,NonNegativeInteger) -> Boolean
nodes : % -> List %
node? : (%,%) -> Boolean if S has SETCAT
parts : % -> List S if $ has finiteAggregate
possiblyInfinite? : % -> Boolean
qelt : (%,Integer) -> S
```

```
qsetelt! : (%,Integer,S) -> S if $ has shallowlyMutable
remove : (S,%) -> %
         if S has SETCAT
         and $ has finiteAggregate
remove : ((S -> Boolean),%) -> % if $ has finiteAggregate
removeDuplicates : % -> %
         if S has SETCAT
         and $ has finiteAggregate
rest : % -> %
rest : (%,NonNegativeInteger) -> %
sample : () -> %
second : % -> S
setchildren! : (%,List %) -> % if $ has shallowlyMutable
setelt : (%,Integer,S) -> S if $ has shallowlyMutable
setelt : (%,UniversalSegment Integer,S) -> S
         if $ has shallowlyMutable
setelt : (%,last,S) -> S if $ has shallowlyMutable
setelt : (%,rest,%) -> % if $ has shallowlyMutable
setelt : (%,first,S) -> S if $ has shallowlyMutable
setelt : (%,value,S) -> S if $ has shallowlyMutable
setfirst! : (%,S) -> S if $ has shallowlyMutable
setlast! : (%,S) -> S if $ has shallowlyMutable
setrest! : (%,%) -> % if $ has shallowlyMutable
setvalue! : (%,S) -> S if $ has shallowlyMutable
size? : (%,NonNegativeInteger) -> Boolean
split! : (%,Integer) -> % if $ has shallowlyMutable
swap! : (%,Integer,Integer) -> Void if $ has shallowlyMutable
tail : % -> %
third : % -> S
value : % -> S
#? : % -> NonNegativeInteger if $ has finiteAggregate
?.last : (%,last) -> S
?.rest : (%,rest) -> %
?.first : (%,first) -> S
?.value : (%,value) -> S
?.? : (%,Integer) -> S
?.? : (%,UniversalSegment Integer) -> %
?=? : (%,%) -> Boolean if S has SETCAT
?~=? : (%,%) -> Boolean if S has SETCAT
```

These exports come from (p555) FiniteLinearAggregate(S:Type)

```
max : (%,%) -> % if S has ORDSET
merge : (%,%) -> % if S has ORDSET
min : (%,%) -> % if S has ORDSET
position : (S,%) -> Integer if S has SETCAT
reverse : % -> %
sort : (((S,S) -> Boolean),%) -> %
sort : % -> % if S has ORDSET
sort! : % -> % if S has ORDSET and $ has shallowlyMutable
```

```
 sorted? : % -> Boolean if S has ORDSET
 ?<=? : (%,%) -> Boolean if S has ORDSET
 ?>? : (%,%) -> Boolean if S has ORDSET
 ?>=? : (%,%) -> Boolean if S has ORDSET
```

These exports come from (p547) ExtensibleLinearAggregate(S:Type):

```
 merge! : (%,%) -> % if S has ORDSET
 remove! : (S,%) -> % if S has SETCAT
 select! : ((S -> Boolean),%) -> %
```

— category LSAGG ListAggregate —

```
)abbrev category LSAGG ListAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A list aggregate is a model for a linked list data structure.
++ A linked list is a versatile
++ data structure. Insertion and deletion are efficient and
++ searching is a linear operation.

ListAggregate(S:Type): Category == Join(StreamAggregate S,
   FiniteLinearAggregate S, ExtensibleLinearAggregate S) with
      list: S -> %
        ++ list(x) returns the list of one element x.
 add
   cycleMax ==> 1000

   mergeSort: ((S, S) -> Boolean, %, Integer) -> %

   sort_!(f, l)              == mergeSort(f, l, #l)
   list x                    == concat(x, empty())
   reduce(f, x)                    ==
     empty? x => _
       error "reducing over an empty list needs the 3 argument form"
     reduce(f, rest x, first x)
   merge(f, p, q)            == merge_!(f, copy p, copy q)

   select_!(f, x) ==
     while not empty? x and not f first x repeat x := rest x
     empty? x => x
```

```
    y := x
    z := rest y
    while not empty? z repeat
      if f first z then (y := z; z := rest z)
                   else (z := rest z; setrest_!(y, z))
    x

  merge_!(f, p, q) ==
    empty? p => q
    empty? q => p
    eq?(p, q) => error "cannot merge a list into itself"
    if f(first p, first q)
      then (r := t := p; p := rest p)
      else (r := t := q; q := rest q)
    while not empty? p and not empty? q repeat
      if f(first p, first q)
        then (setrest_!(t, p); t := p; p := rest p)
        else (setrest_!(t, q); t := q; q := rest q)
    setrest_!(t, if empty? p then q else p)
    r

  insert_!(s:S, x:%, i:Integer) ==
    i < (m := minIndex x) => error "index out of range"
    i = m => concat(s, x)
    y := rest(x, (i - 1 - m)::NonNegativeInteger)
    z := rest y
    setrest_!(y, concat(s, z))
    x

  insert_!(w:%, x:%, i:Integer) ==
    i < (m := minIndex x) => error "index out of range"
    i = m => concat_!(w, x)
    y := rest(x, (i - 1 - m)::NonNegativeInteger)
    z := rest y
    setrest_!(y, w)
    concat_!(y, z)
    x

  remove_!(f:S -> Boolean, x:%) ==
    while not empty? x and f first x repeat x := rest x
    empty? x => x
    p := x
    q := rest x
    while not empty? q repeat
      if f first q then q := setrest_!(p, rest q)
                   else (p := q; q := rest q)
    x

  delete_!(x:%, i:Integer) ==
    i < (m := minIndex x) => error "index out of range"
```

```
  i = m => rest x
  y := rest(x, (i - 1 - m)::NonNegativeInteger)
  setrest_!(y, rest(y, 2))
  x

delete_!(x:%, i:UniversalSegment(Integer)) ==
  (l := lo i) < (m := minIndex x) => error "index out of range"
  h := if hasHi i then hi i else maxIndex x
  h < l => x
  l = m => rest(x, (h + 1 - m)::NonNegativeInteger)
  t := rest(x, (l - 1 - m)::NonNegativeInteger)
  setrest_!(t, rest(t, (h - l + 2)::NonNegativeInteger))
  x

find(f, x) ==
  while not empty? x and not f first x repeat x := rest x
  empty? x => "failed"
  first x

position(f:S -> Boolean, x:%) ==
  for k in minIndex(x).. while not empty? x and not f first x repeat
    x := rest x
  empty? x => minIndex(x) - 1
  k

mergeSort(f, p, n) ==
  if n = 2 and f(first rest p, first p) then p := reverse_! p
  n < 3 => p
  l := (n quo 2)::NonNegativeInteger
  q := split_!(p, l)
  p := mergeSort(f, p, l)
  q := mergeSort(f, q, n - l)
  merge_!(f, p, q)

sorted?(f, l) ==
  empty? l => true
  p := rest l
  while not empty? p repeat
    not f(first l, first p) => return false
    p := rest(l := p)
  true

reduce(f, x, i) ==
  r := i
  while not empty? x repeat (r := f(r, first x); x := rest x)
  r

if S has SetCategory then
   reduce(f, x, i,a) ==
     r := i
```

```
        while not empty? x and r ^= a repeat
          r := f(r, first x)
          x := rest x
        r

   new(n, s) ==
     l := empty()
     for k in 1..n repeat l := concat(s, l)
     l

   map(f, x, y) ==
     z := empty()
     while not empty? x and not empty? y repeat
       z := concat(f(first x, first y), z)
       x := rest x
       y := rest y
     reverse_! z

-- map(f, x, y, d) ==
--   z := empty()
--   while not empty? x and not empty? y repeat
--     z := concat(f(first x, first y), z)
--     x := rest x
--     y := rest y
--   z := reverseInPlace z
--   if not empty? x then
--        z := concat_!(z, map(f(#1, d), x))
--   if not empty? y then
--        z := concat_!(z, map(f(d, #1), y))
--   z

   reverse_! x ==
     empty? x => x
     empty?(y := rest x) => x
     setrest_!(x, empty())
     while not empty? y repeat
       z := rest y
       setrest_!(y, x)
       x := y
       y := z
     x

   copy x ==
     y := empty()
     for k in 0.. while not empty? x repeat
       k = cycleMax and cyclic? x => error "cyclic list"
       y := concat(first x, y)
       x := rest x
     reverse_! y
```

```
  copyInto_!(y, x, s) ==
    s < (m := minIndex y) => error "index out of range"
    z := rest(y, (s - m)::NonNegativeInteger)
    while not empty? z and not empty? x repeat
      setfirst_!(z, first x)
      x := rest x
      z := rest z
    y

  if S has SetCategory then
    position(w, x, s) ==
      s < (m := minIndex x) => error "index out of range"
      x := rest(x, (s - m)::NonNegativeInteger)
      for k in s.. while not empty? x and w ^= first x repeat
        x := rest x
      empty? x => minIndex x - 1
      k

    removeDuplicates_! l ==
      p := l
      while not empty? p repeat
        p := setrest_!(p, remove_!((x:S):Boolean +-> x = first p, rest p))
      l

  if S has OrderedSet then
    x < y ==
       while not empty? x and not empty? y repeat
         first x ^= first y => return(first x < first y)
         x := rest x
         y := rest y
       empty? x => not empty? y
       false
```

———-

— LSAGG.dotabb —

```
"LSAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=LSAGG"];
"LSAGG" -> "FLAGG"
"LSAGG" -> "ELAGG"
```

———-

— LSAGG.dotfull —

```
"ListAggregate(a:Type)"
```

```
 [color=lightblue,href="bookvol10.2.pdf#nameddest=LSAGG"];
"ListAggregate(a:Type)" -> "StreamAggregate(a:Type)"
"ListAggregate(a:Type)" -> "FiniteLinearAggregate(a:Type)"
"ListAggregate(a:Type)" -> "ExtensibleLinearAggregate(a:Type)"

"ListAggregate(Record(a:SetCategory,b:SetCategory))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=LSAGG"];
"ListAggregate(Record(a:SetCategory,b:SetCategory))" ->
    "ListAggregate(a:Type)"
```

———-

— LSAGG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"ListAggregate(a:Type)" [color=lightblue];
"ListAggregate(a:Type)" -> "StreamAggregate(a:Type)"
"ListAggregate(a:Type)" -> "FiniteLinearAggregate(a:Type)"
"ListAggregate(a:Type)" -> "ExtensibleLinearAggregate(a:Type)"

"StreamAggregate(a:Type)" [color=lightblue];
"StreamAggregate(a:Type)" -> "UnaryRecursiveAggregate(a:Type)"
"StreamAggregate(a:Type)" -> "LinearAggregate(a:Type)"

"FiniteLinearAggregate(a:Type)" [color=lightblue];
"FiniteLinearAggregate(a:Type)" -> "LinearAggregate(a:Type)"

"ExtensibleLinearAggregate(a:Type)" [color=lightblue];
"ExtensibleLinearAggregate(a:Type)" -> "LinearAggregate(a:Type)"

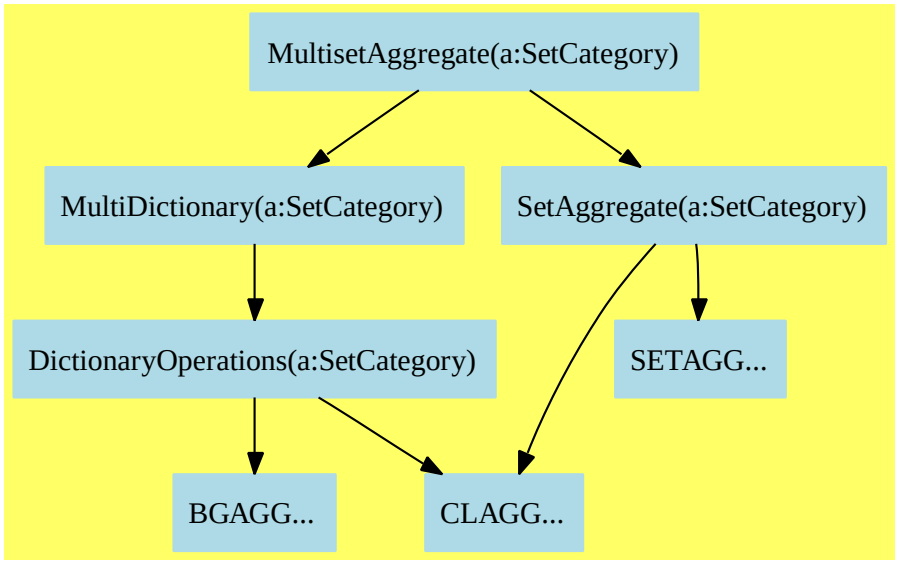
"UnaryRecursiveAggregate(a:Type)" [color=lightblue];
"UnaryRecursiveAggregate(a:Type)" -> "RCAGG..."

"LinearAggregate(a:Type)" [color=lightblue];
"LinearAggregate(a:Type)" -> "IXAGG..."
"LinearAggregate(a:Type)" -> "CLAGG..."

"CLAGG..." [color=lightblue];
"IXAGG..." [color=lightblue];
"RCAGG..." [color=lightblue];
}
```

———-

8.0.115 MultisetAggregate (MSETAGG)



— **MultisetAggregate.input** —

```
)set break resume
)sys rm -f MultisetAggregate.output
)spool MultisetAggregate.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show MultisetAggregate
--R MultisetAggregate S: SetCategory  is a category constructor
--R Abbreviation for MultisetAggregate is MSETAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for MSETAGG
--R
--R------------------------------ Operations --------------------------------
--R ?<? : (%,%) -> Boolean                ?=? : (%,%) -> Boolean
--R bag : List S -> %                     brace : () -> %
--R brace : List S -> %                   coerce : % -> OutputForm
--R construct : List S -> %               copy : % -> %
--R dictionary : List S -> %              dictionary : () -> %
--R difference : (%,%) -> %               difference : (%,S) -> %
```

```
--R empty : () -> %                        empty? : % -> Boolean
--R eq? : (%,%) -> Boolean                 extract! : % -> S
--R hash : % -> SingleInteger              insert! : (S,%) -> %
--R inspect : % -> S                       intersect : (%,%) -> %
--R latex : % -> String                    map : ((S -> S),%) -> %
--R removeDuplicates! : % -> %             sample : () -> %
--R set : () -> %                          set : List S -> %
--R subset? : (%,%) -> Boolean             union : (%,%) -> %
--R union : (%,S) -> %                     union : (S,%) -> %
--R ?~=? : (%,%) -> Boolean
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R convert : % -> InputForm if S has KONVERT INFORM
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R duplicates : % -> List Record(entry: S,count: NonNegativeInteger)
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R find : ((S -> Boolean),%) -> Union(S,"failed")
--R insert! : (S,%,NonNegativeInteger) -> %
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S,S) -> S if S has SETCAT and $ has finiteAggregate
--R remove : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (S,%) -> % if S has SETCAT and $ has finiteAggregate
--R remove! : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R remove! : (S,%) -> % if $ has finiteAggregate
--R removeDuplicates : % -> % if S has SETCAT and $ has finiteAggregate
--R select : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R select! : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R size? : (%,NonNegativeInteger) -> Boolean
--R symmetricDifference : (%,%) -> %
--R
--E 1

)spool
)lisp (bye)
```

— **MultisetAggregate.help** —

```
====================================================================
MultisetAggregate examples
====================================================================

See Also:
o )show MultisetAggregate
```

———-

See:

⇒ "OrderedMultisetAggregate" (OMSAGG) 9.0.128 on page 763
⇐ "MultiDictionary" (MDAGG) 7.0.104 on page 570
⇐ "SetAggregate" (SETAGG) 6.0.94 on page 492

Exports:

any?	bag	brace	coerce
construct	convert	copy	count
dictionary	difference	duplicates	empty
empty?	eq?	eval	every?
extract!	find	hash	insert!
inspect	intersect	latex	less?
map	map!	member?	members
more?	parts	reduce	remove
remove!	removeDuplicates	removeDuplicates!	sample
select	select!	set	size?
subset?	symmetricDifference	union	# ?
?~=?	?<?	?=?	

Attributes exported:

- **partiallyOrderedSet** is true if a set with $<$ which is transitive, but not($a < b$ or $a = b$) does not necessarily imply $b < a$.
- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **nil**

These exports come from (p570) MultiDictionary(S:SetCategory):

```
any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
bag : List S -> %
coerce : % -> OutputForm
```

```
construct : List S -> %
convert : % -> InputForm if S has KONVERT INFORM
copy : % -> %
count : (S,%) -> NonNegativeInteger
    if S has SETCAT and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
    if $ has finiteAggregate
dictionary : List S -> %
dictionary : () -> %
duplicates : % ->
    List Record(entry: S,count: NonNegativeInteger)
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
extract! : % -> S
find : ((S -> Boolean),%) -> Union(S,"failed")
hash : % -> SingleInteger
insert! : (S,%) -> %
insert! : (S,%,NonNegativeInteger) -> %
inspect : % -> S
latex : % -> String
less? : (%,NonNegativeInteger) -> Boolean
map : ((S -> S),%) -> %
map! : ((S -> S),%) -> % if $ has shallowlyMutable
member? : (S,%) -> Boolean
    if S has SETCAT and $ has finiteAggregate
members : % -> List S if $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
parts : % -> List S if $ has finiteAggregate
reduce : (((S,S) -> S),%) -> S
    if $ has finiteAggregate
reduce : (((S,S) -> S),%,S) -> S
    if $ has finiteAggregate
reduce : (((S,S) -> S),%,S,S) -> S
    if S has SETCAT and $ has finiteAggregate
remove : ((S -> Boolean),%) -> %
    if $ has finiteAggregate
remove : (S,%) -> %
    if S has SETCAT and $ has finiteAggregate
removeDuplicates : % -> %
    if S has SETCAT and $ has finiteAggregate
removeDuplicates! : % -> %
remove! : ((S -> Boolean),%) -> %
    if $ has finiteAggregate
remove! : (S,%) -> % if $ has finiteAggregate
```

```
 select : ((S -> Boolean),%) -> %
     if $ has finiteAggregate
 select! : ((S -> Boolean),%) -> %
     if $ has finiteAggregate
 size? : (%,NonNegativeInteger) -> Boolean
 #? : % -> NonNegativeInteger if $ has finiteAggregate
 ?~=? : (%,%) -> Boolean
 ?=? : (%,%) -> Boolean
```

These exports come from (p492) SetAggregate(S:SetCategory):

```
 brace : () -> %
 brace : List S -> %
 difference : (%,S) -> %
 intersect : (%,%) -> %
 sample : () -> %
 set : () -> %
 set : List S -> %
 subset? : (%,%) -> Boolean
 symmetricDifference : (%,%) -> %
 union : (%,S) -> %
 union : (S,%) -> %
 union : (%,%) -> %
 ?<? : (%,%) -> Boolean
```

— category MSETAGG MultisetAggregate —

```
)abbrev category MSETAGG MultisetAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A multi-set aggregate is a set which keeps track of the multiplicity
++ of its elements.

MultisetAggregate(S:SetCategory):
 Category == Join(MultiDictionary S, SetAggregate S)
```

———-

— MSETAGG.dotabb —

```
"MSETAGG"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=MSETAGG"];
"MSETAGG" -> "MDAGG"
"MSETAGG" -> "SETAGG"
```

———-

— **MSETAGG.dotfull** —

```
"MultisetAggregate(a:SetCategory)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=MSETAGG"];
"MultisetAggregate(a:SetCategory)" -> "MultiDictionary(a:SetCategory)"
"MultisetAggregate(a:SetCategory)" -> "SetAggregate(a:SetCategory)"
```

———-

— **MSETAGG.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"MultisetAggregate(a:SetCategory)" [color=lightblue];
"MultisetAggregate(a:SetCategory)" -> "MultiDictionary(a:SetCategory)"
"MultisetAggregate(a:SetCategory)" -> "SetAggregate(a:SetCategory)"

"MultiDictionary(a:SetCategory)" [color=lightblue];
"MultiDictionary(a:SetCategory)" -> "DictionaryOperations(a:SetCategory)"

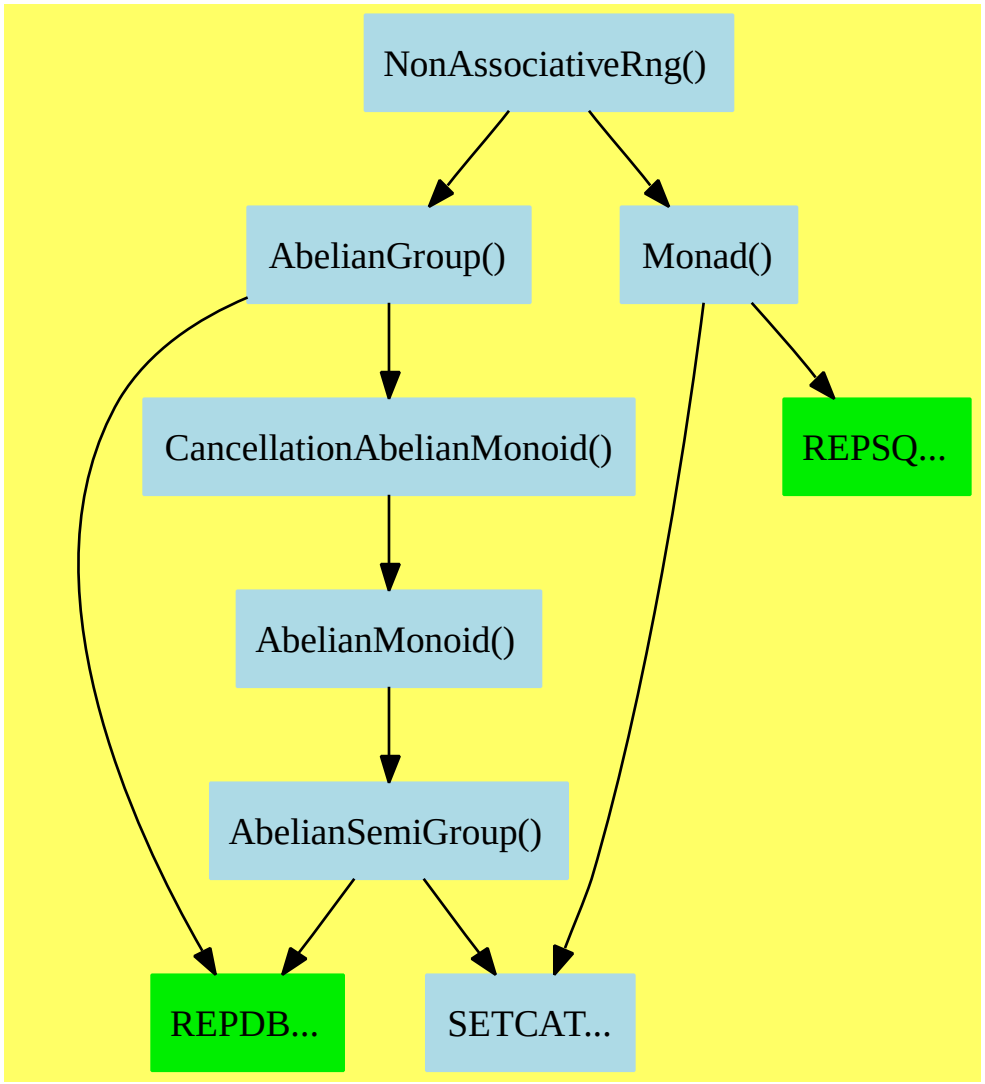
"SetAggregate(a:SetCategory)" [color=lightblue];
"SetAggregate(a:SetCategory)" -> "SETAGG..."
"SetAggregate(a:SetCategory)" -> "CLAGG..."

"DictionaryOperations(a:SetCategory)" [color=lightblue];
"DictionaryOperations(a:SetCategory)" -> "BGAGG..."
"DictionaryOperations(a:SetCategory)" -> "CLAGG..."

"BGAGG..." [color=lightblue];
"CLAGG..." [color=lightblue];
"SETAGG..." [color=lightblue];
}
```

———-

8.0.116 NonAssociativeRng (NARNG)



— **NonAssociativeRng.input** —

```
)set break resume
)sys rm -f NonAssociativeRng.output
)spool NonAssociativeRng.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 1
)show NonAssociativeRng
--R NonAssociativeRng  is a category constructor
--R Abbreviation for NonAssociativeRng is NARNG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for NARNG
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 0 : () -> %                           antiCommutator : (%,%) -> %
--R associator : (%,%,%) -> %             coerce : % -> OutputForm
--R commutator : (%,%) -> %               hash : % -> SingleInteger
--R latex : % -> String                   sample : () -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R leftPower : (%,PositiveInteger) -> %
--R rightPower : (%,PositiveInteger) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— NonAssociativeRng.help —

```
=====================================================================
NonAssociativeRng examples
=====================================================================

See Also:
o )show NonAssociativeRng
```

———-

See:

⇒ “NonAssociativeAlgebra” (NAALG) 11.0.157 on page 980
⇒ “NonAssociativeRing” (NASRING) 9.0.124 on page 740
⇐ “AbelianGroup” (ABELGRP) 7.0.97 on page 520
⇐ “Monad” (MONAD) 4.0.52 on page 197

Exports:

0	antiCommutator	associator	coerce	commutator
hash	latex	leftPower	rightPower	sample
subtractIfCan	zero?	?*?	?**?	?+?
?-?	-?	?=?	?~=?	

These are implemented by this category:

```
antiCommutator : (%,%) -> %
associator : (%,%,%) -> %
commutator : (%,%) -> %
```

These exports come from (p520) AbelianGroup():

```
0 : () -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?~=? : (%,%) -> Boolean
?*? : (PositiveInteger,%) -> %
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?*? : (Integer,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
```

These exports come from (p197) Monad():

```
leftPower : (%,PositiveInteger) -> %
rightPower : (%,PositiveInteger) -> %
?*? : (%,%) -> %
?**? : (%,PositiveInteger) -> %
```

— category NARNG NonAssociativeRng —

```
)abbrev category NARNG NonAssociativeRng
++ Author: J. Grabmeier, R. Wisbauer
++ Date Created: 01 March 1991
++ Date Last Updated: 03 July 1991
++ Basic Operations: +, *, -, **
++ Related Constructors: Rng, Ring, NonAssociativeRing
++ Also See:
++ AMS Classifications:
```

```
++ Keywords: not associative ring
++ Reference:
++  R.D. Schafer: An Introduction to Nonassociative Algebras
++  Academic Press, New York, 1966
++ Description:
++ NonAssociativeRng is a basic ring-type structure, not necessarily
++ commutative or associative, and not necessarily with unit.\br
++ Axioms\br
++ \tab{5}x*(y+z) = x*y + x*z\br
++ \tab{5}(x+y)*z = x*z + y*z\br
++
++ Common Additional Axioms\br
++ \tab{5}noZeroDivisors\tab{5} ab = 0 => a=0 or b=0

NonAssociativeRng(): Category == Join(AbelianGroup,Monad)  with
    associator: (%,%,%) -> %
      ++ associator(a,b,c) returns \spad{(a*b)*c-a*(b*c)}.
    commutator: (%,%) -> %
      ++ commutator(a,b) returns \spad{a*b-b*a}.
    antiCommutator: (%,%) -> %
      ++ antiCommutator(a,b) returns \spad{a*b+b*a}.
  add
    associator(x,y,z) == (x*y)*z - x*(y*z)
    commutator(x,y) == x*y - y*x
    antiCommutator(x,y) == x*y + y*x
```

———-

— NARNG.dotabb —

```
"NARNG"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=NARNG"];
"NARNG" -> "ABELGRP"
"NARNG" -> "MONAD"
```

———-

— NARNG.dotfull —

```
"NonAssociativeRng()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=NARNG"];
"NonAssociativeRng()" -> "AbelianGroup()"
"NonAssociativeRng()" -> "Monad()"
```

———-

— NARNG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"NonAssociativeRng()" [color=lightblue];
"NonAssociativeRng()" -> "AbelianGroup()"
"NonAssociativeRng()" -> "Monad()"

"Monad()" [color=lightblue];
"Monad()" -> "SETCAT..."
"Monad()" -> "REPSQ..."

"AbelianGroup()" [color=lightblue];
"AbelianGroup()" -> "CancellationAbelianMonoid()"
"AbelianGroup()" -> "REPDB..."

"CancellationAbelianMonoid()" [color=lightblue];
"CancellationAbelianMonoid()" -> "AbelianMonoid()"

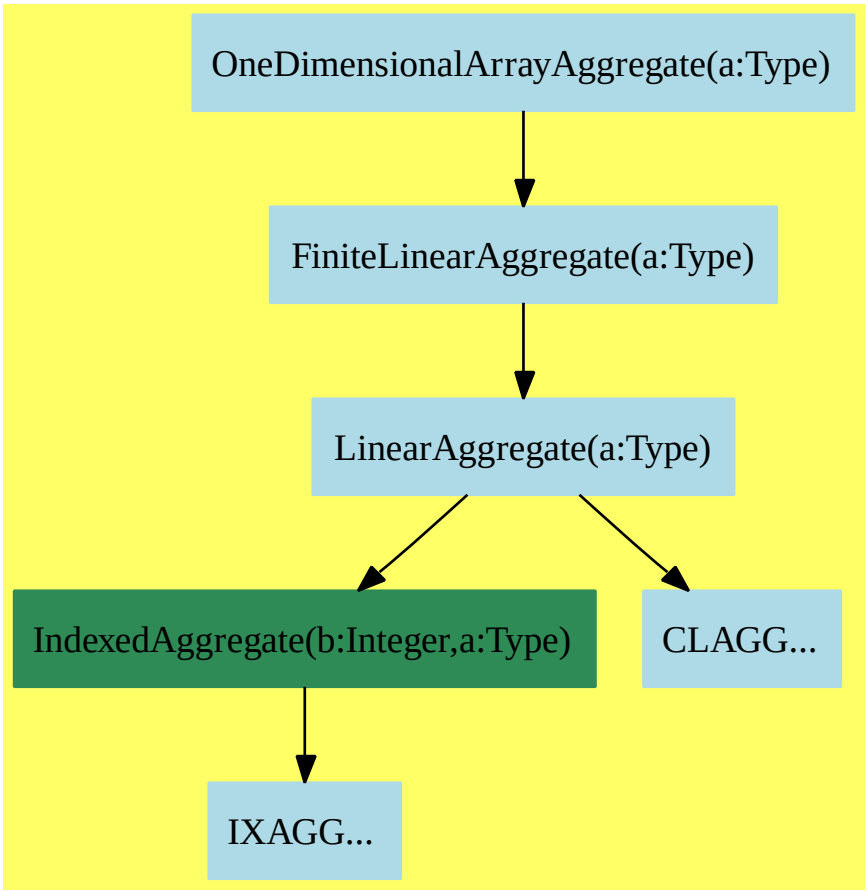
"AbelianMonoid()" [color=lightblue];
"AbelianMonoid()" -> "AbelianSemiGroup()"

"AbelianSemiGroup()" [color=lightblue];
"AbelianSemiGroup()" -> "SETCAT..."
"AbelianSemiGroup()" -> "REPDB..."

"REPDB..." [color="#00EE00"];
"REPSQ..." [color="#00EE00"];
"SETCAT..." [color=lightblue];
}
```

———-

8.0.117 OneDimensionalArrayAggregate (A1AGG)



— OneDimensionalArrayAggregate.input —

```
)set break resume
)sys rm -f OneDimensionalArrayAggregate.output
)spool OneDimensionalArrayAggregate.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show OneDimensionalArrayAggregate
--R OneDimensionalArrayAggregate S: Type  is a category constructor
--R Abbreviation for OneDimensionalArrayAggregate is A1AGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for A1AGG
```

```
--R
--R------------------------------ Operations --------------------------------
--R concat : List % -> %                  concat : (%,%) -> %
--R concat : (S,%) -> %                   concat : (%,S) -> %
--R construct : List S -> %               copy : % -> %
--R delete : (%,Integer) -> %             ?.? : (%,Integer) -> S
--R elt : (%,Integer,S) -> S              empty : () -> %
--R empty? : % -> Boolean                 entries : % -> List S
--R eq? : (%,%) -> Boolean                index? : (Integer,%) -> Boolean
--R indices : % -> List Integer           insert : (%,%,Integer) -> %
--R insert : (S,%,Integer) -> %           map : (((S,S) -> S),%,%) -> %
--R map : ((S -> S),%) -> %               new : (NonNegativeInteger,S) -> %
--R qelt : (%,Integer) -> S               reverse : % -> %
--R sample : () -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?<? : (%,%) -> Boolean if S has ORDSET
--R ?<=? : (%,%) -> Boolean if S has ORDSET
--R ?=? : (%,%) -> Boolean if S has SETCAT
--R ?>? : (%,%) -> Boolean if S has ORDSET
--R ?>=? : (%,%) -> Boolean if S has ORDSET
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R coerce : % -> OutputForm if S has SETCAT
--R convert : % -> InputForm if S has KONVERT INFORM
--R copyInto! : (%,%,Integer) -> % if $ has shallowlyMutable
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R delete : (%,UniversalSegment Integer) -> %
--R ?.? : (%,UniversalSegment Integer) -> %
--R entry? : (S,%) -> Boolean if $ has finiteAggregate and S has SETCAT
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R fill! : (%,S) -> % if $ has shallowlyMutable
--R find : ((S -> Boolean),%) -> Union(S,"failed")
--R first : % -> S if Integer has ORDSET
--R hash : % -> SingleInteger if S has SETCAT
--R latex : % -> String if S has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R max : (%,%) -> % if S has ORDSET
--R maxIndex : % -> Integer if Integer has ORDSET
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R merge : (%,%) -> % if S has ORDSET
--R merge : (((S,S) -> Boolean),%,%) -> %
--R min : (%,%) -> % if S has ORDSET
--R minIndex : % -> Integer if Integer has ORDSET
--R more? : (%,NonNegativeInteger) -> Boolean
```

```
--R parts : % -> List S if $ has finiteAggregate
--R position : (S,%,Integer) -> Integer if S has SETCAT
--R position : (S,%) -> Integer if S has SETCAT
--R position : ((S -> Boolean),%) -> Integer
--R qsetelt! : (%,Integer,S) -> S if $ has shallowlyMutable
--R reduce : (((S,S) -> S),%) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S,S) -> S if S has SETCAT and $ has finiteAggregate
--R remove : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (S,%) -> % if S has SETCAT and $ has finiteAggregate
--R removeDuplicates : % -> % if S has SETCAT and $ has finiteAggregate
--R reverse! : % -> % if $ has shallowlyMutable
--R select : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R setelt : (%,UniversalSegment Integer,S) -> S if $ has shallowlyMutable
--R setelt : (%,Integer,S) -> S if $ has shallowlyMutable
--R size? : (%,NonNegativeInteger) -> Boolean
--R sort : % -> % if S has ORDSET
--R sort : (((S,S) -> Boolean),%) -> %
--R sort! : % -> % if S has ORDSET and $ has shallowlyMutable
--R sort! : (((S,S) -> Boolean),%) -> % if $ has shallowlyMutable
--R sorted? : % -> Boolean if S has ORDSET
--R sorted? : (((S,S) -> Boolean),%) -> Boolean
--R swap! : (%,Integer,Integer) -> Void if $ has shallowlyMutable
--R ?~=? : (%,%) -> Boolean if S has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— OneDimensionalArrayAggregate.help —

```
=====================================================================
OneDimensionalArrayAggregate examples
=====================================================================

See Also:
o )show OneDimensionalArrayAggregate
```

———-

See:

⇒ “BitAggregate” (BTAGG) 9.0.123 on page 731
⇒ “StringAggregate” (SRAGG) 9.0.131 on page 786
⇒ “VectorCategory” (VECTCAT) 9.0.133 on page 808

⇐ “FiniteLinearAggregate” (FLAGG) 7.0.102 on page 555

Exports:

any?	coerce	concat	construct	convert
copy	copyInto!	count	delete	elt
empty	empty?	entries	entry?	eq?
eval	every?	fill!	find	first
hash	index?	indices	insert	latex
less?	map	map!	max	maxIndex
member?	members	merge	min	minIndex
more?	new	parts	position	qelt
qsetelt!	reduce	remove	removeDuplicates	reverse
reverse!	sample	select	setelt	size?
sort	sort!	sorted?	swap!	#?
?.?	?<?	?<=?	?~=?	?=?
?>?	?>=?			

Attributes Exported:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **nil**

These are implemented by this category:

```
any? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
coerce : % -> OutputForm if S has SETCAT
concat : (%,%) -> %
concat : List % -> %
construct : List S -> %
copy : % -> %
copyInto! : (%,%,Integer) -> % if $ has shallowlyMutable
count : ((S -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
delete : (%,UniversalSegment Integer) -> %
delete : (%,Integer) -> %
every? : ((S -> Boolean),%) -> Boolean
         if $ has finiteAggregate
find : ((S -> Boolean),%) -> Union(S,"failed")
insert : (%,%,Integer) -> %
map : (((S,S) -> S),%,%) -> %
map! : ((S -> S),%) -> % if $ has shallowlyMutable
merge : (((S,S) -> Boolean),%,%) -> %
```

```
parts : % -> List S if $ has finiteAggregate
position : ((S -> Boolean),%) -> Integer
position : (S,%,Integer) -> Integer
         if S has SETCAT
reduce : (((S,S) -> S),%) -> S
         if $ has finiteAggregate
reduce : (((S,S) -> S),%,S) -> S
         if $ has finiteAggregate
reduce : (((S,S) -> S),%,S,S) -> S
         if S has SETCAT and $ has finiteAggregate
reverse! : % -> % if $ has shallowlyMutable
setelt : (%,UniversalSegment Integer,S) -> S
         if $ has shallowlyMutable
sort! : (((S,S) -> Boolean),%) -> %
         if $ has shallowlyMutable
sorted? : (((S,S) -> Boolean),%) -> Boolean
?.? : (%,UniversalSegment Integer) -> %
?=? : (%,%) -> Boolean if S has SETCAT
?<? : (%,%) -> Boolean if S has ORDSET
```

These exports come from (p555) FiniteLinearAggregate(S:Type):

```
concat : (S,%) -> %
concat : (%,S) -> %
convert : % -> InputForm if S has KONVERT INFORM
count : (S,%) -> NonNegativeInteger
         if S has SETCAT and $ has finiteAggregate
elt : (%,Integer,S) -> S
empty : () -> %
empty? : % -> Boolean
entries : % -> List S
entry? : (S,%) -> Boolean
         if $ has finiteAggregate and S has SETCAT
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,S,S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,Equation S) -> %
         if S has EVALAB S and S has SETCAT
eval : (%,List Equation S) -> %
         if S has EVALAB S and S has SETCAT
fill! : (%,S) -> % if $ has shallowlyMutable
first : % -> S if Integer has ORDSET
hash : % -> SingleInteger if S has SETCAT
index? : (Integer,%) -> Boolean
indices : % -> List Integer
insert : (S,%,Integer) -> %
latex : % -> String if S has SETCAT
less? : (%,NonNegativeInteger) -> Boolean
```

```
map : ((S -> S),%) -> %
max : (%,%) -> % if S has ORDSET
maxIndex : % -> Integer if Integer has ORDSET
member? : (S,%) -> Boolean
         if S has SETCAT and $ has finiteAggregate
members : % -> List S if $ has finiteAggregate
merge : (%,%) -> % if S has ORDSET
min : (%,%) -> % if S has ORDSET
minIndex : % -> Integer if Integer has ORDSET
more? : (%,NonNegativeInteger) -> Boolean
new : (NonNegativeInteger,S) -> %
position : (S,%) -> Integer if S has SETCAT
qelt : (%,Integer) -> S
qsetelt! : (%,Integer,S) -> S
         if $ has shallowlyMutable
remove : ((S -> Boolean),%) -> %
         if $ has finiteAggregate
remove : (S,%) -> %
         if S has SETCAT and $ has finiteAggregate
removeDuplicates : % -> %
         if S has SETCAT and $ has finiteAggregate
reverse : % -> %
sample : () -> %
select : ((S -> Boolean),%) -> %
         if $ has finiteAggregate
setelt : (%,Integer,S) -> S if $ has shallowlyMutable
size? : (%,NonNegativeInteger) -> Boolean
sort : % -> % if S has ORDSET
sort : (((S,S) -> Boolean),%) -> %
sort! : % -> %
         if S has ORDSET and $ has shallowlyMutable
sorted? : % -> Boolean if S has ORDSET
swap! : (%,Integer,Integer) -> Void
         if $ has shallowlyMutable
#? : % -> NonNegativeInteger if $ has finiteAggregate
?.? : (%,Integer) -> S
?~=? : (%,%) -> Boolean if S has SETCAT
?>? : (%,%) -> Boolean if S has ORDSET
?>=? : (%,%) -> Boolean if S has ORDSET
?<=? : (%,%) -> Boolean if S has ORDSET
```

— category A1AGG OneDimensionalArrayAggregate —

```
)abbrev category A1AGG OneDimensionalArrayAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
```

```
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ One-dimensional-array aggregates serves as models for one-dimensional
++ arrays. Categorically, these aggregates are finite linear aggregates
++ with the \spadatt{shallowlyMutable} property, that is, any component of
++ the array may be changed without affecting the
++ identity of the overall array.
++ Array data structures are typically represented by a fixed area in
++ storage and cannot efficiently grow or shrink on demand as can list
++ structures (see however \spadtype{FlexibleArray} for a data structure
++ which is a cross between a list and an array).
++ Iteration over, and access to, elements of arrays is extremely fast
++ (and often can be optimized to open-code).
++ Insertion and deletion however is generally slow since an entirely new
++ data structure must be created for the result.

OneDimensionalArrayAggregate(S:Type): Category ==
    FiniteLinearAggregate S with shallowlyMutable
  add
    parts x            == [qelt(x, i) for i in minIndex x .. maxIndex x]
    sort_!(f, a) == quickSort(f, a)$FiniteLinearAggregateSort(S, %)

    any?(f, a) ==
      for i in minIndex a .. maxIndex a repeat
        f qelt(a, i) => return true
      false

    every?(f, a) ==
      for i in minIndex a .. maxIndex a repeat
        not(f qelt(a, i)) => return false
      true

    position(f:S -> Boolean, a:%) ==
      for i in minIndex a .. maxIndex a repeat
        f qelt(a, i) => return i
      minIndex(a) - 1

    find(f, a) ==
      for i in minIndex a .. maxIndex a repeat
        f qelt(a, i) => return qelt(a, i)
      "failed"

    count(f:S->Boolean, a:%) ==
      n:NonNegativeInteger := 0
      for i in minIndex a .. maxIndex a repeat
        if f(qelt(a, i)) then n := n+1
      n
```

```
    map_!(f, a) ==
      for i in minIndex a .. maxIndex a repeat
        qsetelt_!(a, i, f qelt(a, i))
      a

    setelt(a:%, s:UniversalSegment(Integer), x:S) ==
      l := lo s; h := if hasHi s then hi s else maxIndex a
      l < minIndex a or h > maxIndex a => error "index out of range"
      for k in l..h repeat qsetelt_!(a, k, x)
      x

    reduce(f, a) ==
      empty? a => error "cannot reduce an empty aggregate"
      r := qelt(a, m := minIndex a)
      for k in m+1 .. maxIndex a repeat r := f(r, qelt(a, k))
      r

    reduce(f, a, identity) ==
      for k in minIndex a .. maxIndex a repeat
        identity := f(identity, qelt(a, k))
      identity

    if S has SetCategory then
       reduce(f, a, identity,absorber) ==
         for k in minIndex a .. maxIndex a while identity ^= absorber
                repeat identity := f(identity, qelt(a, k))
         identity

-- this is necessary since new has disappeared.
    stupidnew: (NonNegativeInteger, %, %) -> %
    stupidget: List % -> S
-- a and b are not both empty if n > 0
    stupidnew(n, a, b) ==
      zero? n => empty()
      new(n, (empty? a => qelt(b, minIndex b); qelt(a, minIndex a)))
-- at least one element of l must be non-empty
    stupidget l ==
      for a in l repeat
        not empty? a => return first a
      error "Should not happen"

    map(f, a, b) ==
      m := max(minIndex a, minIndex b)
      n := min(maxIndex a, maxIndex b)
      l := max(0, n - m + 1)::NonNegativeInteger
      c := stupidnew(l, a, b)
      for i in minIndex(c).. for j in m..n repeat
        qsetelt_!(c, i, f(qelt(a, j), qelt(b, j)))
      c
```

```
--  map(f, a, b, x) ==
--    m := min(minIndex a, minIndex b)
--    n := max(maxIndex a, maxIndex b)
--    l := (n - m + 1)::NonNegativeInteger
--    c := new l
--    for i in minIndex(c).. for j in m..n repeat
--        qsetelt_!(c, i, f(a(j, x), b(j, x)))
--    c

    merge(f, a, b) ==
      r := stupidnew(#a + #b, a, b)
      i := minIndex a
      m := maxIndex a
      j := minIndex b
      n := maxIndex b
      for k in minIndex(r).. while i <= m and j <= n repeat
        if f(qelt(a, i), qelt(b, j)) then
          qsetelt_!(r, k, qelt(a, i))
          i := i+1
        else
          qsetelt_!(r, k, qelt(b, j))
          j := j+1
      for k in k.. for i in i..m repeat qsetelt_!(r, k, elt(a, i))
      for k in k.. for j in j..n repeat qsetelt_!(r, k, elt(b, j))
      r

    elt(a:%, s:UniversalSegment(Integer)) ==
      l := lo s
      h := if hasHi s then hi s else maxIndex a
      l < minIndex a or h > maxIndex a => error "index out of range"
      r := stupidnew(max(0, h - l + 1)::NonNegativeInteger, a, a)
      for k in minIndex r.. for i in l..h repeat
        qsetelt_!(r, k, qelt(a, i))
      r

    insert(a:%, b:%, i:Integer) ==
      m := minIndex b
      n := maxIndex b
      i < m or i > n => error "index out of range"
      y := stupidnew(#a + #b, a, b)
      for k in minIndex y.. for j in m..i-1 repeat
        qsetelt_!(y, k, qelt(b, j))
      for k in k.. for j in minIndex a .. maxIndex a repeat
        qsetelt_!(y, k, qelt(a, j))
      for k in k.. for j in i..n repeat qsetelt_!(y, k, qelt(b, j))
      y

    copy x ==
      y := stupidnew(#x, x, x)
```

```
      for i in minIndex x .. maxIndex x for j in minIndex y .. repeat
        qsetelt_!(y, j, qelt(x, i))
      y

    copyInto_!(y, x, s) ==
      s < minIndex y or s + #x > maxIndex y + 1 =>
                                              error "index out of range"
      for i in minIndex x .. maxIndex x for j in s.. repeat
        qsetelt_!(y, j, qelt(x, i))
      y

    construct l ==
--    a := new(#l)
      empty? l => empty()
      a := new(#l, first l)
      for i in minIndex(a).. for x in l repeat qsetelt_!(a, i, x)
      a

    delete(a:%, s:UniversalSegment(Integer)) ==
      l := lo s; h := if hasHi s then hi s else maxIndex a
      l < minIndex a or h > maxIndex a => error "index out of range"
      h < l => copy a
      r := stupidnew((#a - h + l - 1)::NonNegativeInteger, a, a)
      for k in minIndex(r).. for i in minIndex a..l-1 repeat
        qsetelt_!(r, k, qelt(a, i))
      for k in k.. for i in h+1 .. maxIndex a repeat
        qsetelt_!(r, k, qelt(a, i))
      r

    delete(x:%, i:Integer) ==
      i < minIndex x or i > maxIndex x => error "index out of range"
      y := stupidnew((#x - 1)::NonNegativeInteger, x, x)
      for i in minIndex(y).. for j in minIndex x..i-1 repeat
        qsetelt_!(y, i, qelt(x, j))
      for i in i .. for j in i+1 .. maxIndex x repeat
        qsetelt_!(y, i, qelt(x, j))
      y

    reverse_! x ==
      m := minIndex x
      n := maxIndex x
      for i in 0..((n-m) quo 2) repeat swap_!(x, m+i, n-i)
      x

    concat l ==
      empty? l => empty()
      n := _+/[#a for a in l]
      i := minIndex(r := new(n, stupidget l))
      for a in l repeat
        copyInto_!(r, a, i)
```

```
      i := i + #a
    r

  sorted?(f, a) ==
    for i in minIndex(a)..maxIndex(a)-1 repeat
      not f(qelt(a, i), qelt(a, i + 1)) => return false
    true

  concat(x:%, y:%) ==
    z := stupidnew(#x + #y, x, y)
    copyInto_!(z, x, i := minIndex z)
    copyInto_!(z, y, i + #x)
    z

  if S has SetCategory then
    x = y ==
      #x ^= #y => false
      for i in minIndex x .. maxIndex x repeat
        not(qelt(x, i) = qelt(y, i)) => return false
      true

    coerce(r:%):OutputForm ==
      bracket commaSeparate
            [qelt(r, k)::OutputForm for k in minIndex r .. maxIndex r]

    position(x:S, t:%, s:Integer) ==
      n := maxIndex t
      s < minIndex t or s > n => error "index out of range"
      for k in s..n repeat
        qelt(t, k) = x => return k
      minIndex(t) - 1

  if S has OrderedSet then
    a < b ==
      for i in minIndex a .. maxIndex a
        for j in minIndex b .. maxIndex b repeat
          qelt(a, i) ^= qelt(b, j) => return a.i < b.j
      #a < #b
```

———-

— A1AGG.dotabb —

```
"A1AGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=A1AGG"];
"A1AGG" -> "FLAGG"
```

———-

— **A1AGG.dotfull** —

```
"OneDimensionalArrayAggregate(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=A1AGG"];
"OneDimensionalArrayAggregate(a:Type)" ->
    "FiniteLinearAggregate(a:Type)"

"OneDimensionalArrayAggregate(Character)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=A1AGG"];
"OneDimensionalArrayAggregate(Character)" ->
    "OneDimensionalArrayAggregate(a:Type)"

"OneDimensionalArrayAggregate(Boolean)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=A1AGG"];
"OneDimensionalArrayAggregate(Boolean)" ->
    "OneDimensionalArrayAggregate(a:Type)"
```

———-

— **A1AGG.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"OneDimensionalArrayAggregate(a:Type)" [color=lightblue];
"OneDimensionalArrayAggregate(a:Type)" ->
    "FiniteLinearAggregate(a:Type)"

"FiniteLinearAggregate(a:Type)" [color=lightblue];
"FiniteLinearAggregate(a:Type)" -> "LinearAggregate(a:Type)"

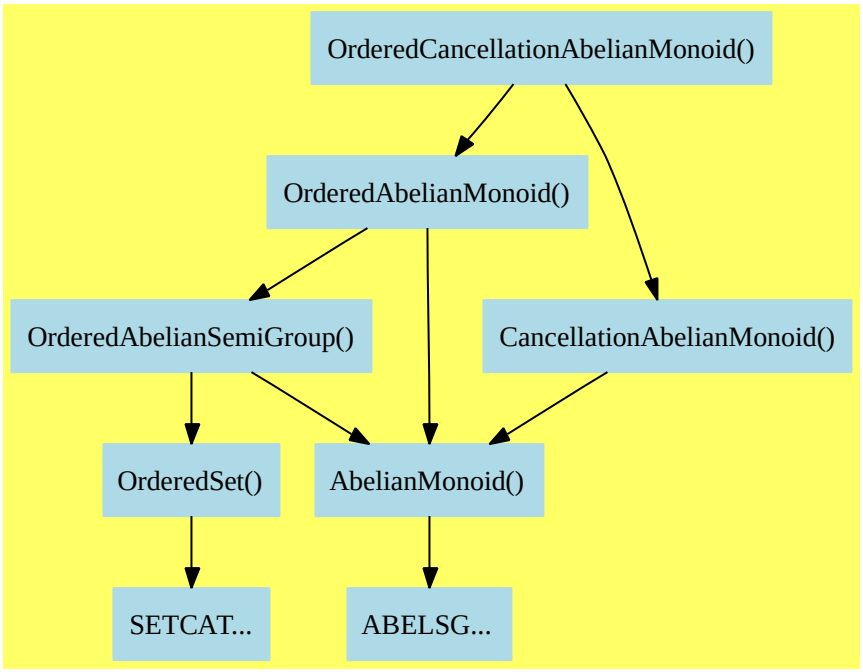
"LinearAggregate(a:Type)" [color=lightblue];
"LinearAggregate(a:Type)" -> "IndexedAggregate(b:Integer,a:Type)"
"LinearAggregate(a:Type)" -> "CLAGG..."

"IndexedAggregate(b:Integer,a:Type)" [color=seagreen];
"IndexedAggregate(b:Integer,a:Type)" -> "IXAGG..."

"CLAGG..." [color=lightblue];
"IXAGG..." [color=lightblue];
}
```

———-

8.0.118 OrderedCancellationAbelianMonoid (OCAMON)



— OrderedCancellationAbelianMonoid.input —

```
)set break resume
)sys rm -f OrderedCancellationAbelianMonoid.output
)spool OrderedCancellationAbelianMonoid.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show OrderedCancellationAbelianMonoid
--R OrderedCancellationAbelianMonoid  is a category constructor
--R Abbreviation for OrderedCancellationAbelianMonoid is OCAMON
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for OCAMON
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (PositiveInteger,%) -> %        ?+? : (%,%) -> %
--R ?<? : (%,%) -> Boolean                ?<=? : (%,%) -> Boolean
--R ?=? : (%,%) -> Boolean                ?>? : (%,%) -> Boolean
--R ?>=? : (%,%) -> Boolean               0 : () -> %
--R coerce : % -> OutputForm              hash : % -> SingleInteger
```

```
--R latex : % -> String                    max : (%,%) -> %
--R min : (%,%) -> %                       sample : () -> %
--R zero? : % -> Boolean                   ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **OrderedCancellationAbelianMonoid.help** —

```
====================================================================
OrderedCancellationAbelianMonoid examples
====================================================================

See Also:
o )show OrderedCancellationAbelianMonoid
```

———-

See:

⇒ “OrderedAbelianGroup” (OAGROUP) 9.0.126 on page 755
⇒ “OrderedAbelianMonoidSup” (OAMONS) 9.0.127 on page 759
⇐ “CancellationAbelianMonoid” (CABMON) 6.0.83 on page 380
⇐ “OrderedAbelianMonoid” (OAMON) 7.0.105 on page 576

Exports:

0	coerce	hash	latex	max
min	sample	subtractIfCan	zero?	?~=?
?*?	?+?	?<?	?<=?	?=?
?>?	?>=?			

These exports come from (p576) OrderedAbelianMonoid():

```
0 : () -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
max : (%,%) -> %
min : (%,%) -> %
sample : () -> %
zero? : % -> Boolean
```

```
 ?<? : (%,%) -> Boolean
 ?<=? : (%,%) -> Boolean
 ?=? : (%,%) -> Boolean
 ?>? : (%,%) -> Boolean
 ?>=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
 ?*? : (NonNegativeInteger,%) -> %
 ?*? : (PositiveInteger,%) -> %
 ?+? : (%,%) -> %
```

These exports come from (p380) CancellationAbelianMonoid():

```
 subtractIfCan : (%,%) -> Union(%,"failed")
```

— category OCAMON OrderedCancellationAbelianMonoid —

```
)abbrev category OCAMON OrderedCancellationAbelianMonoid
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ Ordered sets which are also abelian cancellation monoids,
++ such that the addition preserves the ordering.

OrderedCancellationAbelianMonoid(): Category ==
        Join(OrderedAbelianMonoid, CancellationAbelianMonoid)
```

———-

— OCAMON.dotabb —

```
"OCAMON"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OCAMON"];
"OCAMON" -> "OAMON"
"OCAMON" -> "CABMON"
```

———-

— OCAMON.dotfull —

```
"OrderedCancellationAbelianMonoid()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OCAMON"];
"OrderedCancellationAbelianMonoid()" -> "OrderedAbelianMonoid()"
"OrderedCancellationAbelianMonoid()" -> "CancellationAbelianMonoid()"
```

———-

— OCAMON.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"OrderedCancellationAbelianMonoid()" [color=lightblue];
"OrderedCancellationAbelianMonoid()" -> "OrderedAbelianMonoid()"
"OrderedCancellationAbelianMonoid()" -> "CancellationAbelianMonoid()"

"OrderedAbelianMonoid()" [color=lightblue];
"OrderedAbelianMonoid()" -> "OrderedAbelianSemiGroup()"
"OrderedAbelianMonoid()" -> "AbelianMonoid()"

"OrderedAbelianSemiGroup()" [color=lightblue];
"OrderedAbelianSemiGroup()" -> "OrderedSet()"
"OrderedAbelianSemiGroup()" -> "AbelianMonoid()"

"OrderedSet()" [color=lightblue];
"OrderedSet()" -> "SETCAT..."

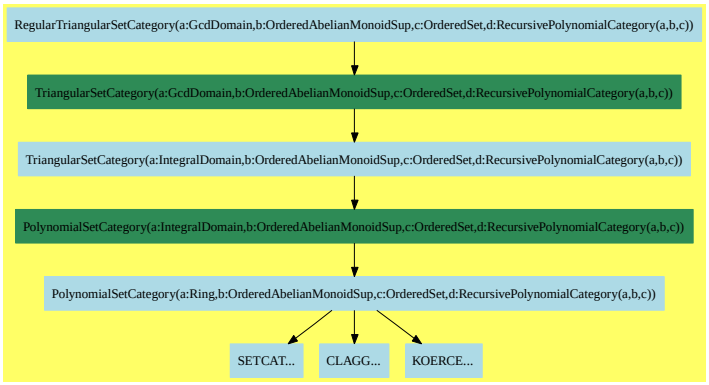
"CancellationAbelianMonoid()" [color=lightblue];
"CancellationAbelianMonoid()" -> "AbelianMonoid()"

"AbelianMonoid()" [color=lightblue];
"AbelianMonoid()" -> "ABELSG..."

"SETCAT..." [color=lightblue];
"ABELSG..." [color=lightblue];
}
```

———-

8.0.119 RegularTriangularSetCategory (RSETCAT)



— RegularTriangularSetCategory.input —

```
)set break resume
)sys rm -f RegularTriangularSetCategory.output
)spool RegularTriangularSetCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show RegularTriangularSetCategory
--R RegularTriangularSetCategory(R: GcdDomain,E: OrderedAbelianMonoidSup,V: OrderedSet,P: Recurs
--R Abbreviation for RegularTriangularSetCategory is RSETCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for RSETCAT
--R
--R------------------------------ Operations ------------------------------
--R ?=? : (%,%) -> Boolean                algebraic? : (V,%) -> Boolean
--R algebraicVariables : % -> List V      augment : (List P,%) -> List %
--R augment : (P,List %) -> List %        augment : (P,%) -> List %
--R coerce : % -> List P                  coerce : % -> OutputForm
--R collect : (%,V) -> %                  collectQuasiMonic : % -> %
--R collectUnder : (%,V) -> %             collectUpper : (%,V) -> %
--R construct : List P -> %               copy : % -> %
--R degree : % -> NonNegativeInteger      empty : () -> %
--R empty? : % -> Boolean                 eq? : (%,%) -> Boolean
--R extend : (List P,%) -> List %         extend : (P,List %) -> List %
--R extend : (P,%) -> List %              extend : (%,P) -> %
--R first : % -> Union(P,"failed")        hash : % -> SingleInteger
--R headReduce : (P,%) -> P               headReduced? : % -> Boolean
--R headReduced? : (P,%) -> Boolean       infRittWu? : (%,%) -> Boolean
--R initiallyReduce : (P,%) -> P          initiallyReduced? : % -> Boolean
```

```
--R initials : % -> List P                internalAugment : (P,%) -> %
--R intersect : (P,List %) -> List %      intersect : (List P,%) -> List %
--R intersect : (P,%) -> List %           invertible? : (P,%) -> Boolean
--R invertibleSet : (P,%) -> List %       last : % -> Union(P,"failed")
--R latex : % -> String                   mainVariable? : (V,%) -> Boolean
--R mainVariables : % -> List V           map : ((P -> P),%) -> %
--R mvar : % -> V                         normalized? : % -> Boolean
--R normalized? : (P,%) -> Boolean        purelyAlgebraic? : % -> Boolean
--R reduceByQuasiMonic : (P,%) -> P       removeZero : (P,%) -> P
--R rest : % -> Union(%,"failed")         retract : List P -> %
--R sample : () -> %                      stronglyReduce : (P,%) -> P
--R stronglyReduced? : % -> Boolean       trivialIdeal? : % -> Boolean
--R variables : % -> List V               zeroSetSplit : List P -> List %
--R ?~=? : (%,%) -> Boolean
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R algebraicCoefficients? : (P,%) -> Boolean
--R any? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
--R augment : (List P,List %) -> List %
--R autoReduced? : (%,((P,List P) -> Boolean)) -> Boolean
--R basicSet : (List P,(P -> Boolean),((P,P) -> Boolean)) -> Union(Record(bas: %,top: List P),"failed")
--R basicSet : (List P,((P,P) -> Boolean)) -> Union(Record(bas: %,top: List P),"failed")
--R coHeight : % -> NonNegativeInteger if V has FINITE
--R convert : % -> InputForm if P has KONVERT INFORM
--R count : ((P -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R count : (P,%) -> NonNegativeInteger if P has SETCAT and $ has finiteAggregate
--R eval : (%,List Equation P) -> % if P has EVALAB P and P has SETCAT
--R eval : (%,Equation P) -> % if P has EVALAB P and P has SETCAT
--R eval : (%,P,P) -> % if P has EVALAB P and P has SETCAT
--R eval : (%,List P,List P) -> % if P has EVALAB P and P has SETCAT
--R every? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
--R extend : (List P,List %) -> List %
--R extendIfCan : (%,P) -> Union(%,"failed")
--R find : ((P -> Boolean),%) -> Union(P,"failed")
--R headRemainder : (P,%) -> Record(num: P,den: R) if R has INTDOM
--R initiallyReduced? : (P,%) -> Boolean
--R internalAugment : (List P,%) -> %
--R intersect : (List P,List %) -> List %
--R invertible? : (P,%) -> List Record(val: Boolean,tower: %)
--R invertibleElseSplit? : (P,%) -> Union(Boolean,List %)
--R lastSubResultant : (P,P,%) -> List Record(val: P,tower: %)
--R lastSubResultantElseSplit : (P,P,%) -> Union(P,List %)
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((P -> P),%) -> % if $ has shallowlyMutable
--R member? : (P,%) -> Boolean if P has SETCAT and $ has finiteAggregate
--R members : % -> List P if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List P if $ has finiteAggregate
--R purelyAlgebraic? : (P,%) -> Boolean
--R purelyAlgebraicLeadingMonomial? : (P,%) -> Boolean
--R purelyTranscendental? : (P,%) -> Boolean
```

```
--R quasiComponent : % -> Record(close: List P,open: List P)
--R reduce : (P,%,((P,P) -> P),((P,P) -> Boolean)) -> P
--R reduce : (((P,P) -> P),%) -> P if $ has finiteAggregate
--R reduce : (((P,P) -> P),%,P) -> P if $ has finiteAggregate
--R reduce : (((P,P) -> P),%,P,P) -> P if P has SETCAT and $ has finiteAggregate
--R reduced? : (P,%,((P,P) -> Boolean)) -> Boolean
--R remainder : (P,%) -> Record(rnum: R,polnum: P,den: R) if R has INTDOM
--R remove : ((P -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (P,%) -> % if P has SETCAT and $ has finiteAggregate
--R removeDuplicates : % -> % if P has SETCAT and $ has finiteAggregate
--R retractIfCan : List P -> Union(%,"failed")
--R rewriteIdealWithHeadRemainder : (List P,%) -> List P if R has INTDOM
--R rewriteIdealWithRemainder : (List P,%) -> List P if R has INTDOM
--R rewriteSetWithReduction : (List P,%,((P,P) -> P),((P,P) -> Boolean)) -> List P
--R roughBase? : % -> Boolean if R has INTDOM
--R roughEqualIdeals? : (%,%) -> Boolean if R has INTDOM
--R roughSubIdeal? : (%,%) -> Boolean if R has INTDOM
--R roughUnitIdeal? : % -> Boolean if R has INTDOM
--R select : (%,V) -> Union(P,"failed")
--R select : ((P -> Boolean),%) -> % if $ has finiteAggregate
--R size? : (%,NonNegativeInteger) -> Boolean
--R sort : (%,V) -> Record(under: %,floor: %,upper: %)
--R squareFreePart : (P,%) -> List Record(val: P,tower: %)
--R stronglyReduced? : (P,%) -> Boolean
--R triangular? : % -> Boolean if R has INTDOM
--R zeroSetSplit : (List P,Boolean) -> List %
--R zeroSetSplitIntoTriangularSystems : List P -> List Record(close: %,open: List P)
--R
--E 1

)spool
)lisp (bye)
```

———-

— RegularTriangularSetCategory.help —

```
=====================================================================
RegularTriangularSetCategory examples
=====================================================================

See Also:
o )show RegularTriangularSetCategory
```

———-

See:

⇒ “NormalizedTriangularSetCategory” (NTSCAT) 9.0.125 on page 745

⇒ "SquareFreeRegularTriangularSetCategory" (SFRTCAT) 9.0.130 on page 775
⇐ "TriangularSetCategory" (TSETCAT) 7.0.108 on page 595

Exports:

algebraicCoefficients?	algebraicVariables	any?
augment	autoReduced?	basicSet
coerce	coHeight	collect
collectQuasiMonic	collectUnder	collectUpper
construct	convert	copy
count	degree	empty
empty?	eq?	eval
every?	extend	extendIfCan
find	first	hash
headReduce	headReduced?	headRemainder
infRittWu?	initiallyReduce	initiallyReduced?
initials	internalAugment	intersect
invertible?	invertibleElseSplit?	invertibleSet
last	lastSubResultant	lastSubResultantElseSplit
less?	latex	mainVariable?
mainVariables	map	map!
member?	members	more?
mvar	normalized?	parts
purelyAlgebraic?	purelyAlgebraicLeadingMonomial?	purelyTranscendental?
quasiComponent	reduce	reduceByQuasiMonic
reduced?	remainder	remove
removeDuplicates	removeZero	rest
retract	retractIfCan	rewriteIdealWithHeadRemainder
rewriteIdealWithRemainder	rewriteSetWithReduction	roughBase?
roughEqualIdeals?	roughSubIdeal?	roughUnitIdeal?
sample	select	size?
sort	squareFreePart	stronglyReduce
stronglyReduced?	triangular?	trivialIdeal?
variables	zeroSetSplit	zeroSetSplitIntoTriangularSystems
#?	?=?	?~=?

Attributes Exported:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **nil**

These are directly exported but not implemented:

```
augment : (P,%) -> List %
extend : (P,%) -> List %
internalAugment : (P,%) -> %
internalAugment : (List P,%) -> %
intersect : (P,%) -> List %
invertibleElseSplit? : (P,%) -> Union(Boolean,List %)
invertible? : (P,%) -> List Record(val: Boolean,tower: %)
invertible? : (P,%) -> Boolean
invertibleSet : (P,%) -> List %
lastSubResultant : (P,P,%) -> List Record(val: P,tower: %)
lastSubResultantElseSplit : (P,P,%) -> Union(P,List %)
squareFreePart : (P,%) -> List Record(val: P,tower: %)
zeroSetSplit : (List P,Boolean) -> List %
```

These are implemented by this category:

```
algebraicCoefficients? : (P,%) -> Boolean
augment : (P,List %) -> List %
augment : (List P,%) -> List %
augment : (List P,List %) -> List %
extend : (P,List %) -> List %
extend : (List P,%) -> List %
extend : (List P,List %) -> List %
intersect : (List P,List %) -> List %
intersect : (List P,%) -> List %
intersect : (P,List %) -> List %
purelyAlgebraic? : % -> Boolean
purelyAlgebraic? : (P,%) -> Boolean
purelyAlgebraicLeadingMonomial? : (P,%) -> Boolean
purelyTranscendental? : (P,%) -> Boolean
```

These exports come from (p595) TriangularSetCategory(R,E,V,P) where R:GcdDomain, E:OrderedAbelianMonoidSup, V:OrderedSet, P:RecursivePolynomialCategory(R,E,V)):

```
algebraic? : (V,%) -> Boolean
algebraicVariables : % -> List V
any? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
autoReduced? : (%,((P,List P) -> Boolean)) -> Boolean
basicSet :
  (List P,((P,P) -> Boolean)) ->
    Union(Record(bas: %,top: List P),"failed")
basicSet :
  (List P,(P -> Boolean),((P,P) -> Boolean)) ->
     Union(Record(bas: %,top: List P),"failed")
coerce : % -> List P
coerce : % -> OutputForm
coHeight : % -> NonNegativeInteger if V has FINITE
collect : (%,V) -> %
```

```
collectQuasiMonic : % -> %
collectUnder : (%,V) -> %
collectUpper : (%,V) -> %
construct : List P -> %
convert : % -> InputForm if P has KONVERT INFORM
copy : % -> %
count : ((P -> Boolean),%) -> NonNegativeInteger
  if $ has finiteAggregate
count : (P,%) -> NonNegativeInteger
  if P has SETCAT
  and $ has finiteAggregate
degree : % -> NonNegativeInteger
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List Equation P) -> % if P has EVALAB P and P has SETCAT
eval : (%,Equation P) -> % if P has EVALAB P and P has SETCAT
eval : (%,P,P) -> % if P has EVALAB P and P has SETCAT
eval : (%,List P,List P) -> % if P has EVALAB P and P has SETCAT
every? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
extend : (%,P) -> %
extendIfCan : (%,P) -> Union(%,"failed")
find : ((P -> Boolean),%) -> Union(P,"failed")
first : % -> Union(P,"failed")
hash : % -> SingleInteger
headReduce : (P,%) -> P
headReduced? : % -> Boolean
headReduced? : (P,%) -> Boolean
headRemainder : (P,%) -> Record(num: P,den: R) if R has INTDOM
infRittWu? : (%,%) -> Boolean
initiallyReduce : (P,%) -> P
initiallyReduced? : % -> Boolean
initiallyReduced? : (P,%) -> Boolean
initials : % -> List P
last : % -> Union(P,"failed")
latex : % -> String
less? : (%,NonNegativeInteger) -> Boolean
mainVariable? : (V,%) -> Boolean
mainVariables : % -> List V
map : ((P -> P),%) -> %
map! : ((P -> P),%) -> % if $ has shallowlyMutable
member? : (P,%) -> Boolean if P has SETCAT and $ has finiteAggregate
members : % -> List P if $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
mvar : % -> V
normalized? : % -> Boolean
normalized? : (P,%) -> Boolean
parts : % -> List P if $ has finiteAggregate
quasiComponent : % -> Record(close: List P,open: List P)
reduce : (((P,P) -> P),%) -> P if $ has finiteAggregate
```

```
reduce : (((P,P) -> P),%,P) -> P if $ has finiteAggregate
reduce : (((P,P) -> P),%,P,P) -> P
  if P has SETCAT
  and $ has finiteAggregate
reduce : (P,%,((P,P) -> P),((P,P) -> Boolean)) -> P
reduceByQuasiMonic : (P,%) -> P
reduced? : (P,%,((P,P) -> Boolean)) -> Boolean
remainder : (P,%) -> Record(rnum: R,polnum: P,den: R) if R has INTDOM
remove : ((P -> Boolean),%) -> % if $ has finiteAggregate
remove : (P,%) -> % if P has SETCAT and $ has finiteAggregate
removeDuplicates : % -> % if P has SETCAT and $ has finiteAggregate
removeZero : (P,%) -> P
rest : % -> Union(%,"failed")
retract : List P -> %
rewriteIdealWithHeadRemainder : (List P,%) -> List P if R has INTDOM
rewriteIdealWithRemainder : (List P,%) -> List P if R has INTDOM
retractIfCan : List P -> Union(%,"failed")
rewriteSetWithReduction :
   (List P,%,((P,P) -> P),((P,P) -> Boolean)) -> List P
roughBase? : % -> Boolean if R has INTDOM
roughEqualIdeals? : (%,%) -> Boolean if R has INTDOM
roughSubIdeal? : (%,%) -> Boolean if R has INTDOM
roughUnitIdeal? : % -> Boolean if R has INTDOM
sample : () -> %
select : ((P -> Boolean),%) -> % if $ has finiteAggregate
select : (%,V) -> Union(P,"failed")
size? : (%,NonNegativeInteger) -> Boolean
sort : (%,V) -> Record(under: %,floor: %,upper: %)
stronglyReduce : (P,%) -> P
stronglyReduced? : (P,%) -> Boolean
stronglyReduced? : % -> Boolean
triangular? : % -> Boolean if R has INTDOM
trivialIdeal? : % -> Boolean
variables : % -> List V
zeroSetSplit : List P -> List %
zeroSetSplitIntoTriangularSystems :
   List P -> List Record(close: %,open: List P)
#? : % -> NonNegativeInteger if $ has finiteAggregate
?~=? : (%,%) -> Boolean
?=? : (%,%) -> Boolean
```

— category RSETCAT RegularTriangularSetCategory —

```
)abbrev category RSETCAT RegularTriangularSetCategory
++ Author: Marc Moreno Maza
++ Date Created: 09/03/1998
++ Date Last Updated: 12/15/1998
++ Basic Functions:
++ Related Constructors:
```

```
++ Also See: essai Graphisme
++ AMS Classifications:
++ Keywords: polynomial, multivariate, ordered variables set
++ References :
++  [1] M. KALKBRENER "Three contributions to elimination theory"
++      Phd Thesis, University of Linz, Austria, 1991.
++  [2] M. KALKBRENER "Algorithmic properties of polynomial rings"
++      Journal of Symbol. Comp. 1998
++  [3] P. AUBRY, D. LAZARD and M. MORENO MAZA "On the Theories
++      of Triangular Sets" Journal of Symbol. Comp. (to appear)
++  [4] M. MORENO MAZA "A new algorithm for computing triangular
++      decomposition of algebraic varieties" NAG Tech. Rep. 4/98.
++ Description:
++ The category of regular triangular sets, introduced under
++ the name regular chains in [1] (and other papers).
++ In [3] it is proved that regular triangular sets and towers of simple
++ extensions of a field are equivalent notions.
++ In the following definitions, all polynomials and ideals
++ are taken from the polynomial ring \spad{k[x1,...,xn]} where \spad{k}
++ is the fraction field of \spad{R}.
++ The triangular set \spad{[t1,...,tm]} is regular
++ iff for every \spad{i} the initial of \spad{ti+1} is invertible
++ in the tower of simple extensions associated with \spad{[t1,...,ti]}.
++ A family \spad{[T1,...,Ts]} of regular triangular sets
++ is a split of Kalkbrener of a given ideal \spad{I}
++ iff the radical of \spad{I} is equal to the intersection
++ of the radical ideals generated by the saturated ideals
++ of the \spad{[T1,...,Ti]}.
++ A family \spad{[T1,...,Ts]} of regular triangular sets
++ is a split of Kalkbrener of a given triangular set \spad{T}
++ iff it is a split of Kalkbrener of the saturated ideal of \spad{T}.
++ Let \spad{K} be an algebraic closure of \spad{k}.
++ Assume that \spad{V} is finite with cardinality
++ \spad{n} and let \spad{A} be the affine space \spad{K^n}.
++ For a regular triangular set \spad{T} let denote by \spad{W(T)} the
++ set of regular zeros of \spad{T}.
++ A family \spad{[T1,...,Ts]} of regular triangular sets
++ is a split of Lazard of a given subset \spad{S} of \spad{A}
++ iff the union of the \spad{W(Ti)} contains \spad{S} and
++ is contained in the closure of \spad{S} (w.r.t. Zariski topology).
++ A family \spad{[T1,...,Ts]} of regular triangular sets
++ is a split of Lazard of a given triangular set \spad{T}
++ if it is a split of Lazard of \spad{W(T)}.
++ Note that if \spad{[T1,...,Ts]} is a split of Lazard of
++ \spad{T} then it is also a split of Kalkbrener of \spad{T}.
++ The converse is false.
++ This category provides operations related to both kinds of
++ splits, the former being related to ideals decomposition whereas
++ the latter deals with varieties decomposition.
++ See the example illustrating the RegularTriangularSet constructor for more
```

```
++ explanations about decompositions by means of regular triangular sets.

RegularTriangularSetCategory(R:GcdDomain, E:OrderedAbelianMonoidSup,_
  V:OrderedSet,P:RecursivePolynomialCategory(R,E,V)):
         Category ==
   TriangularSetCategory(R,E,V,P) with
    purelyAlgebraic?: (P,$) -> Boolean
      ++ \spad{purelyAlgebraic?(p,ts)} returns \spad{true} iff every
      ++ variable of \spad{p} is algebraic w.r.t. \spad{ts}.
    purelyTranscendental? : (P,$) -> Boolean
      ++ \spad{purelyTranscendental?(p,ts)} returns \spad{true} iff every
      ++ variable of \spad{p} is not algebraic w.r.t. \spad{ts}
    algebraicCoefficients?  : (P,$) -> Boolean
      ++ \spad{algebraicCoefficients?(p,ts)} returns \spad{true} iff every
      ++ variable of \spad{p} which is not the main one of \spad{p}
      ++ is algebraic w.r.t. \spad{ts}.
    purelyAlgebraic?: $ -> Boolean
      ++ \spad{purelyAlgebraic?(ts)} returns true iff for every algebraic
      ++ variable \spad{v} of \spad{ts} we have
      ++ \spad{algebraicCoefficients?(t_v,ts_v_-)} where \spad{ts_v}
      ++ is select from TriangularSetCategory(ts,v) and
      ++ \spad{ts_v_-} is
      ++ collectUnder from TriangularSetCategory(ts,v).
    purelyAlgebraicLeadingMonomial?: (P, $) -> Boolean
      ++ \spad{purelyAlgebraicLeadingMonomial?(p,ts)} returns true iff
      ++ the main variable of any non-constant iterarted initial
      ++ of \spad{p} is algebraic w.r.t. \spad{ts}.
    invertibleElseSplit? : (P,$) -> Union(Boolean,List $)
      ++ \spad{invertibleElseSplit?(p,ts)} returns \spad{true} (resp.
      ++ \spad{false}) if \spad{p} is invertible in the tower
      ++ associated with \spad{ts} or returns a split of Kalkbrener
      ++ of \spad{ts}.
    invertible? : (P,$) -> List Record(val : Boolean, tower : $)
      ++ \spad{invertible?(p,ts)} returns \spad{lbwt} where \spad{lbwt.i}
      ++ is the result of \spad{invertibleElseSplit?(p,lbwt.i.tower)} and
      ++ the list of the \spad{(lqrwt.i).tower} is a split of Kalkbrener of
      ++ \spad{ts}.
    invertible?: (P,$) -> Boolean
      ++ \spad{invertible?(p,ts)} returns true iff \spad{p} is invertible
      ++ in the tower associated with \spad{ts}.
    invertibleSet: (P,$) -> List $
      ++ \spad{invertibleSet(p,ts)} returns a split of Kalkbrener of the
      ++ quotient ideal of the ideal \axiom{I}  by \spad{p} where \spad{I} is
      ++ the radical of saturated of \spad{ts}.
    lastSubResultantElseSplit: (P, P, $) -> Union(P,List $)
      ++ \spad{lastSubResultantElseSplit(p1,p2,ts)} returns either
      ++ \spad{g} a quasi-monic gcd of \spad{p1} and \spad{p2} w.r.t.
      ++ the \spad{ts} or a split of Kalkbrener of \spad{ts}.
      ++ This assumes that \spad{p1} and \spad{p2} have the same maim
      ++ variable and that this variable is greater that any variable
```

```
  ++ occurring in \spad{ts}.
lastSubResultant: (P, P, $) -> List Record(val : P, tower : $)
  ++ \spad{lastSubResultant(p1,p2,ts)} returns \spad{lpwt} such that
  ++ \spad{lpwt.i.val} is a quasi-monic gcd of \spad{p1} and \spad{p2}
  ++ w.r.t. \spad{lpwt.i.tower}, for every \spad{i}, and such
  ++ that the list of the \spad{lpwt.i.tower} is a split of Kalkbrener of
  ++ \spad{ts}. Moreover, if \spad{p1} and \spad{p2} do not
  ++ have a non-trivial gcd w.r.t. \spad{lpwt.i.tower} then
  ++ \spad{lpwt.i.val} is the resultant of these polynomials w.r.t.
  ++ \spad{lpwt.i.tower}. This assumes that \spad{p1} and \spad{p2} have
  ++ the same main variable and that this variable is greater that any
  ++ variable occurring in \spad{ts}.
squareFreePart: (P,$) -> List Record(val : P, tower : $)
  ++ \spad{squareFreePart(p,ts)} returns \spad{lpwt} such that
  ++ \spad{lpwt.i.val} is a square-free polynomial
  ++ w.r.t. \spad{lpwt.i.tower}, this polynomial being associated with
  ++ \spad{p} modulo \spad{lpwt.i.tower}, for every \spad{i}. Moreover,
  ++ the list of the \spad{lpwt.i.tower} is a split
  ++ of Kalkbrener of \spad{ts}.
  ++ WARNING: This assumes that \spad{p} is a non-constant polynomial
  ++ such that if \spad{p} is added to \spad{ts}, then the resulting set
  ++ is a regular triangular set.
intersect: (P,$) -> List $
  ++ \spad{intersect(p,ts)} returns the same as
  ++ \spad{intersect([p],ts)}
intersect: (List P, $) -> List $
  ++ \spad{intersect(lp,ts)} returns \spad{lts} a split of Lazard
  ++ of the intersection of the affine variety associated
  ++ with \spad{lp} and the regular zero set of \spad{ts}.
intersect: (List P, List $) -> List $
  ++ \spad{intersect(lp,lts)} returns the same as
  ++ \spad{concat([intersect(lp,ts) for ts in lts])|}
intersect: (P, List $) -> List $
  ++ \spad{intersect(p,lts)} returns the same as
  ++ \spad{intersect([p],lts)}
augment: (P,$) -> List $
  ++ \spad{augment(p,ts)} assumes that \spad{p} is a non-constant
  ++ polynomial whose main variable is greater than any variable
  ++ of \spad{ts}. This operation assumes also that if \spad{p} is
  ++ added to \spad{ts} the resulting set, say \spad{ts+p}, is a
  ++ regular triangular set. Then it returns a split of Kalkbrener
  ++ of \spad{ts+p}. This may not be \spad{ts+p} itself, if for
  ++ instance \spad{ts+p} is required to be square-free.
augment: (P,List $) -> List $
  ++ \spad{augment(p,lts)} returns the same as
  ++ \spad{concat([augment(p,ts) for ts in lts])}
augment: (List P,$) -> List $
  ++ \spad{augment(lp,ts)} returns \spad{ts} if \spad{empty? lp},
  ++ \spad{augment(p,ts)} if \spad{lp = [p]}, otherwise
  ++ \spad{augment(first lp, augment(rest lp, ts))}
```

```
    augment: (List P,List $) -> List $
      ++ \spad{augment(lp,lts)} returns the same as
      ++ \spad{concat([augment(lp,ts) for ts in lts])}
    internalAugment: (P, $) -> $
      ++ \spad{internalAugment(p,ts)} assumes that \spad{augment(p,ts)}
      ++ returns a singleton and returns it.
    internalAugment: (List P, $) -> $
      ++ \spad{internalAugment(lp,ts)} returns \spad{ts} if \spad{lp}
      ++ is empty otherwise returns
      ++ \spad{internalAugment(rest lp, internalAugment(first lp, ts))}
    extend: (P,$) -> List $
      ++ \spad{extend(p,ts)} assumes that \spad{p} is a non-constant
      ++ polynomial whose main variable is greater than any variable
      ++ of \spad{ts}. Then it returns a split of Kalkbrener
      ++ of \spad{ts+p}. This may not be \spad{ts+p} itself, if for
      ++ instance \spad{ts+p} is not a regular triangular set.
    extend: (P, List $)  -> List $
      ++ \spad{extend(p,lts)} returns the same as
      ++ \spad{concat([extend(p,ts) for ts in lts])|}
    extend: (List P,$) -> List $
      ++ \spad{extend(lp,ts)} returns \spad{ts} if \spad{empty? lp}
      ++ \spad{extend(p,ts)} if \spad{lp = [p]} else
      ++ \spad{extend(first lp, extend(rest lp, ts))}
    extend: (List P,List $) -> List $
      ++ \spad{extend(lp,lts)} returns the same as
      ++ \spad{concat([extend(lp,ts) for ts in lts])|}
    zeroSetSplit: (List P, Boolean) -> List $
      ++ \spad{zeroSetSplit(lp,clos?)} returns \spad{lts} a split of
      ++ Kalkbrener of the radical ideal associated with \spad{lp}.
      ++ If \spad{clos?} is false, it is also a decomposition of the
      ++ variety associated with \spad{lp} into the regular zero set of the
      ++ \spad{ts} in \spad{lts} (or, in other words, a split of Lazard of
      ++ this variety). See the example illustrating the
      ++ \spadtype{RegularTriangularSet} constructor for more explanations
      ++ about decompositions by means of regular triangular sets.

  add

    NNI ==> NonNegativeInteger
    INT ==> Integer
    LP ==> List P
    PWT ==> Record(val : P, tower : $)
    LpWT ==> Record(val : (List P), tower : $)
    Split ==> List $
    pack ==> PolynomialSetUtilitiesPackage(R,E,V,P)

    purelyAlgebraic?(p: P, ts: $): Boolean ==
      ground? p => true
      not algebraic?(mvar(p),ts) => false
      algebraicCoefficients?(p,ts)
```

```
purelyTranscendental?(p:P,ts:$): Boolean  ==
  empty? ts => true
  lv : List V := variables(p)$P
  while (not empty? lv) and (not algebraic?(first(lv),ts)) repeat _
    lv := rest lv
  empty? lv

purelyAlgebraicLeadingMonomial?(p: P, ts: $): Boolean  ==
  ground? p => true
  algebraic?(mvar(p),ts) and purelyAlgebraicLeadingMonomial?(init(p), ts)

algebraicCoefficients?(p:P,ts:$): Boolean  ==
  ground? p => true
  (not ground? init(p)) and not (algebraic?(mvar(init(p)),ts)) => false
  algebraicCoefficients?(init(p),ts) =>
    ground? tail(p) => true
    mvar(tail(p)) = mvar(p) =>
      algebraicCoefficients?(tail(p),ts)
    algebraic?(mvar(tail(p)),ts) =>
      algebraicCoefficients?(tail(p),ts)
    false
  false

if V has Finite
then
 purelyAlgebraic?(ts: $): Boolean ==
    empty? ts => true
    size()$V = #ts => true
    lp: LP := sort(infRittWu?,members(ts))
    i: NonNegativeInteger := size()$V
    for p in lp repeat
      v: V := mvar(p)
      (i = (lookup(v)$V)::NNI) =>
        i := subtractIfCan(i,1)::NNI
      univariate?(p)$pack =>
        i := subtractIfCan(i,1)::NNI
      not algebraicCoefficients?(p,collectUnder(ts,v)) =>
        return false
      i := subtractIfCan(i,1)::NNI
    true

else

  purelyAlgebraic?(ts: $): Boolean ==
    empty? ts => true
    v: V := mvar(ts)
    p: P := select(ts,v)::P
    ts := collectUnder(ts,v)
    empty? ts => univariate?(p)$pack
```

```
    not purelyAlgebraic?(ts) => false
    algebraicCoefficients?(p,ts)

augment(p:P,lts:List $) ==
  toSave: Split := []
  while not empty? lts repeat
    ts := first lts
    lts := rest lts
    toSave := concat(augment(p,ts),toSave)
  toSave

augment(lp:LP,ts:$) ==
  toSave: Split := [ts]
  empty? lp => toSave
  lp := sort(infRittWu?,lp)
  while not empty? lp repeat
    p := first lp
    lp := rest lp
    toSave := augment(p,toSave)
  toSave

augment(lp:LP,lts:List $) ==
  empty? lp => lts
  toSave: Split := []
  while not empty? lts repeat
    ts := first lts
    lts := rest lts
    toSave := concat(augment(lp,ts),toSave)
  toSave

extend(p:P,lts:List $) ==
  toSave : Split := []
  while not empty? lts repeat
    ts := first lts
    lts := rest lts
    toSave := concat(extend(p,ts),toSave)
  toSave

extend(lp:LP,ts:$) ==
  toSave: Split := [ts]
  empty? lp => toSave
  lp := sort(infRittWu?,lp)
  while not empty? lp repeat
    p := first lp
    lp := rest lp
    toSave := extend(p,toSave)
  toSave

extend(lp:LP,lts:List $) ==
  empty? lp => lts
```

```
      toSave: Split := []
      while not empty? lts repeat
        ts := first lts
        lts := rest lts
        toSave := concat(extend(lp,ts),toSave)
      toSave

    intersect(lp:LP,lts:List $): List $  ==
      -- A VERY GENERAL default algorithm
      (empty? lp) or (empty? lts) => lts
      lp := [primitivePart(p) for p in lp]
      lp := removeDuplicates lp
      lp := remove(zero?,lp)
      any?(ground?,lp) => []
      toSee: List LpWT := [[lp,ts]$LpWT for ts in lts]
      toSave: List $ := []
      lp: LP
      p: P
      ts: $
      lus: List $
      while (not empty? toSee) repeat
        lpwt := first toSee; toSee := rest toSee
        lp := lpwt.val; ts := lpwt.tower
        empty? lp => toSave := cons(ts, toSave)
        p := first lp;  lp := rest lp
        lus := intersect(p,ts)
        toSee := concat([[lp,us]$LpWT for us in lus], toSee)
      toSave

    intersect(lp: LP,ts: $): List $  ==
      intersect(lp,[ts])

    intersect(p: P,lts: List $): List $  ==
      intersect([p],lts)
```

———-

— **RSETCAT.dotabb** —

```
"RSETCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RSETCAT"];
"RSETCAT" -> "TSETCAT"
```

———-

— **RSETCAT.dotfull** —

```
"RegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePoly
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RSETCAT"];
"RegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePoly
  ->
"TriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialC
```

———-

— RSETCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"RegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePoly
 [color=lightblue];
"RegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePoly
  ->
"TriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialC

"TriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialC
 [color=seagreen];
"TriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialC
 ->
"TriangularSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
"TriangularSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
 [color=lightblue];
"TriangularSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
  -> "PolynomialSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursiveP

"PolynomialSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
 [color=seagreen];
"PolynomialSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
  -> "PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialC

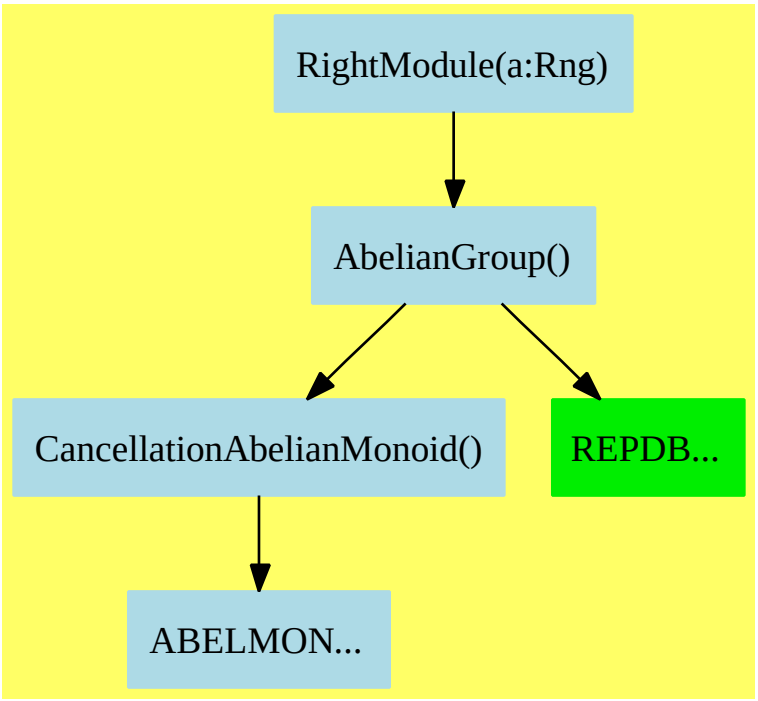
"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCatego
 [color=lightblue];
"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCatego
 -> "SETCAT..."
"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCatego
 -> "CLAGG..."
"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCatego
 -> "KOERCE..."

"SETCAT..." [color=lightblue];
"KOERCE..." [color=lightblue];
"CLAGG..." [color=lightblue];
```

```
}
```

———-

8.0.120 RightModule (RMODULE)



— **RightModule.input** —

```
)set break resume
)sys rm -f RightModule.output
)spool RightModule.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show RightModule
--R RightModule R: Rng  is a category constructor
--R Abbreviation for RightModule is RMODULE
--R This constructor is exposed in this frame.
```

```
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for RMODULE
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (%,R) -> %                    ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %      ?+? : (%,%) -> %
--R ?-? : (%,%) -> %                    -? : % -> %
--R ?=? : (%,%) -> Boolean              0 : () -> %
--R coerce : % -> OutputForm            hash : % -> SingleInteger
--R latex : % -> String                 sample : () -> %
--R zero? : % -> Boolean                ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **RightModule.help** —

```
=====================================================================
RightModule examples
=====================================================================

See Also:
o )show RightModule
```

———-

See:

⇒ “BiModule” (BMODULE) 9.0.122 on page 726
⇐ “AbelianGroup” (ABELGRP) 7.0.97 on page 520

Exports:

0	coerce	hash	latex	sample
subtractIfCan	zero?	?~=?	?*?	?+?
?-?	-?	?=?		

These are directly exported but not implemented:

```
?*? : (%,R) -> %
```

These exports come from (p520) AbelianGroup():

```
0 : () -> %
```

```
 coerce : % -> OutputForm
 hash : % -> SingleInteger
 latex : % -> String
 sample : () -> %
 subtractIfCan : (%,%) -> Union(%,"failed")
 zero? : % -> Boolean
 ?~=? : (%,%) -> Boolean
 ?*? : (PositiveInteger,%) -> %
 ?+? : (%,%) -> %
 ?=? : (%,%) -> Boolean
 ?*? : (NonNegativeInteger,%) -> %
 ?*? : (Integer,%) -> %
 ?-? : (%,%) -> %
 -? : % -> %
```

— category RMODULE RightModule —

```
)abbrev category RMODULE RightModule
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The category of right modules over an rng (ring not necessarily
++ with unit). This is an abelian group which supports right
++ multiplication by elements of the rng.
++
++ Axioms\br
++ \tab{5}\spad{ x*(a*b) = (x*a)*b }\br
++ \tab{5}\spad{ x*(a+b) = (x*a)+(x*b) }\br
++ \tab{5}\spad{ (x+y)*x = (x*a)+(y*a) }

RightModule(R:Rng):Category == AbelianGroup with
   "*": (%,R) -> %
     ++ x*r returns the right multiplication of the module element x
     ++ by the ring element r.
```

———-

— RMODULE.dotabb —

```
"RMODULE"
```

```
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RMODULE"];
"RMODULE" -> "ABELGRP"
```

———-

— **RMODULE.dotfull** —

```
"RightModule(a:Rng)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RMODULE"];
"RightModule(a:Rng)" -> "AbelianGroup()"

"RightModule(a:Ring)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=RMODULE"];
"RightModule(a:Ring)" -> "RightModule(a:Rng)"
```

———-

— **RMODULE.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"RightModule(a:Rng)" [color=lightblue];
"RightModule(a:Rng)" -> "AbelianGroup()"

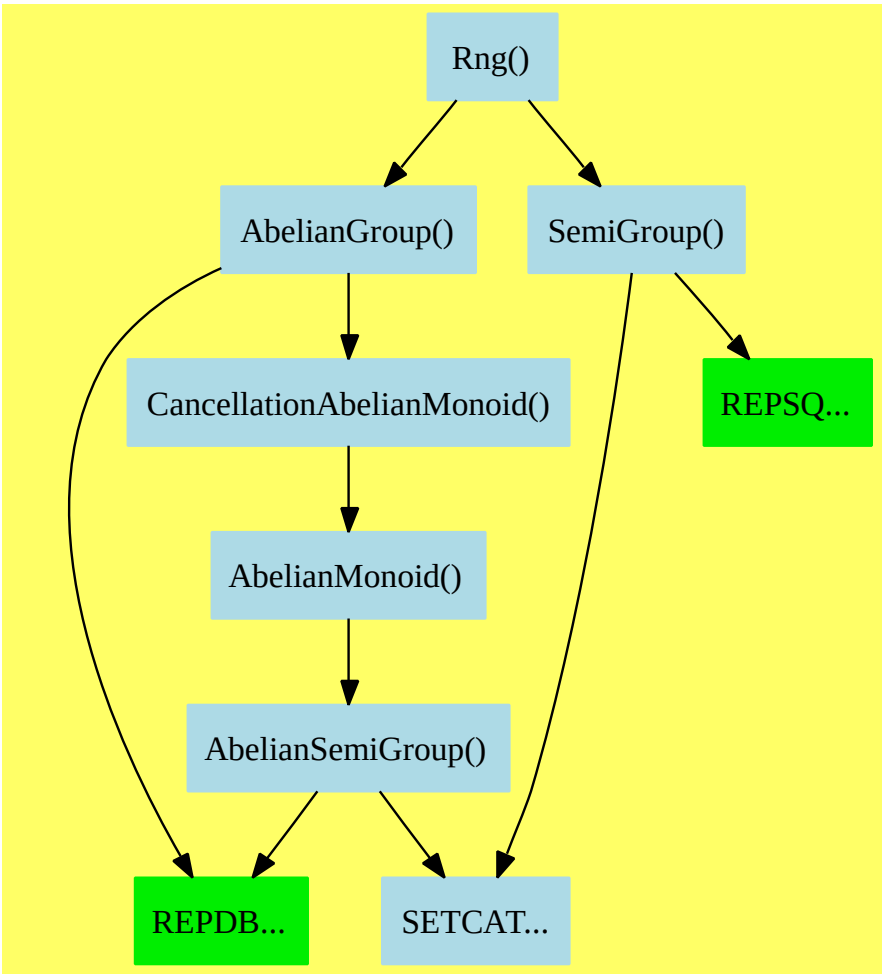
"AbelianGroup()" [color=lightblue];
"AbelianGroup()" -> "CancellationAbelianMonoid()"
"AbelianGroup()" -> "REPDB..."

"CancellationAbelianMonoid()" [color=lightblue];
"CancellationAbelianMonoid()" -> "ABELMON..."

"ABELMON..." [color=lightblue];
"REPDB..." [color="#00EE00"];
}
```

———-

8.0.121 Rng (RNG)



Rng is a Ring that does not necessarily have a unit.

— Rng.input —

```
)set break resume
)sys rm -f Rng.output
)spool Rng.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
```

```
)show Rng
--R Rng  is a category constructor
--R Abbreviation for Rng is RNG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for RNG
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 0 : () -> %                           ?^? : (%,PositiveInteger) -> %
--R coerce : % -> OutputForm              hash : % -> SingleInteger
--R latex : % -> String                   sample : () -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **Rng.help** —

```
====================================================================
Rng examples
====================================================================

See Also:
o )show Rng
```

———-

See:

⇒ “Ring” (RING) 9.0.129 on page 770
⇐ “AbelianGroup” (ABELGRP) 7.0.97 on page 520
⇐ “SemiGroup” (SGROUP) 4.0.61 on page 239

Exports:

0	coerce	hash	latex	sample
zero?	subtractIfCan	?*?	?**?	?+?
?-?	-?	?=?	?~=?	?^?

These exports come from (p520) AbelianGroup():

```
0 : () -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (Integer,%) -> %
?-? : (%,%) -> %
-? : % -> %
```

These exports come from (p239) SemiGroup():

```
?*? : (%,%) -> %
?**? : (%,PositiveInteger) -> %
?^? : (%,PositiveInteger) -> %
```

— category RNG Rng —

```
)abbrev category RNG Rng
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The category of associative rings, not necessarily commutative, and not
++ necessarily with a 1. This is a combination of an abelian group
++ and a semigroup, with multiplication distributing over addition.
++
++ Axioms\br
++ \tab{5}\spad{ x*(y+z) = x*y + x*z}\br
++ \tab{5}\spad{ (x+y)*z = x*z + y*z }
++
++ Conditional attributes\br
++ \tab{5}noZeroDivisors\tab{5}\spad{  ab = 0 => a=0 or b=0}
```

```
Rng(): Category == Join(AbelianGroup,SemiGroup)
```

———-

— RNG.dotabb —

```
"RNG" [color=lightblue,href="bookvol10.2.pdf#nameddest=RNG"];
"RNG" -> "ABELGRP"
"RNG" -> "SGROUP"
```

———-

— RNG.dotfull —

```
"Rng()" [color=lightblue,href="bookvol10.2.pdf#nameddest=RNG"];
"Rng()" -> "AbelianGroup()"
"Rng()" -> "SemiGroup()"
```

———-

— RNG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"Rng()" [color=lightblue];
"Rng()" -> "AbelianGroup()"
"Rng()" -> "SemiGroup()"

"AbelianGroup()" [color=lightblue];
"AbelianGroup()" -> "CancellationAbelianMonoid()"
"AbelianGroup()" -> "REPDB..."

"CancellationAbelianMonoid()" [color=lightblue];
"CancellationAbelianMonoid()" -> "AbelianMonoid()"

"AbelianMonoid()" [color=lightblue];
"AbelianMonoid()" -> "AbelianSemiGroup()"

"AbelianSemiGroup()" [color=lightblue];
"AbelianSemiGroup()" -> "SETCAT..."
"AbelianSemiGroup()" -> "REPDB..."
```

```
"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SETCAT..."
"SemiGroup()" -> "REPSQ..."

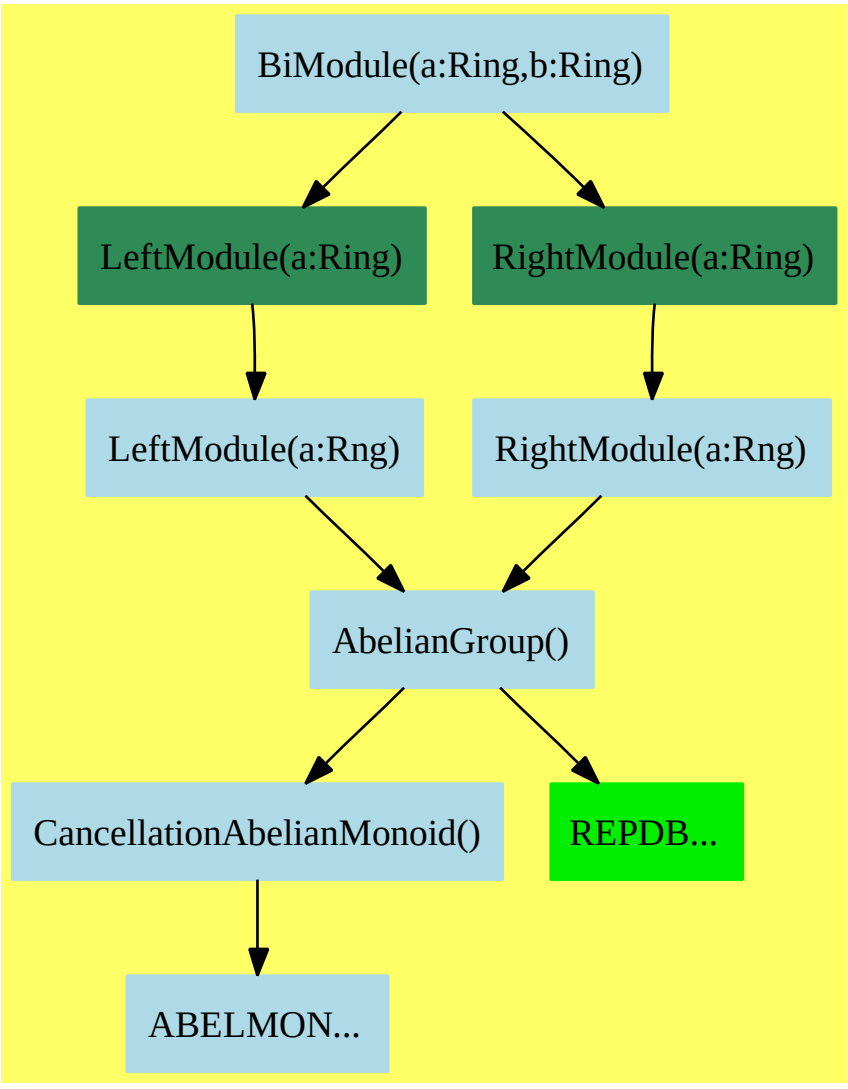
"REPDB..." [color="#00EE00"];
"REPSQ..." [color="#00EE00"];
"SETCAT..." [color=lightblue];
}
```

———————-

Chapter 9

Category Layer 8

9.0.122 BiModule (BMODULE)



— **BiModule.input** —

```
)set break resume
)sys rm -f BiModule.output
)spool BiModule.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show BiModule
--R BiModule(R: Ring,S: Ring)  is a category constructor
--R Abbreviation for BiModule is BMODULE
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for BMODULE
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,S) -> %                      ?*? : (R,%) -> %
--R ?*? : (Integer,%) -> %                ?*? : (PositiveInteger,%) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 0 : () -> %                           coerce : % -> OutputForm
--R hash : % -> SingleInteger             latex : % -> String
--R sample : () -> %                      zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **BiModule.help** —

```
=====================================================================
BiModule examples
=====================================================================

See Also:
o )show BiModule
```

———-

See:

⇒ "AbelianMonoidRing" (AMR) 13.0.170 on page 1101
⇒ "CommutativeRing" (COMRING) 10.0.137 on page 843
⇒ "DirectProductCategory" (DIRPCAT) 12.0.160 on page 1001
⇒ "EntireRing" (ENTIRER) 10.0.139 on page 853
⇒ "FreeModuleCat" (FMCAT) 10.0.140 on page 858
⇒ "Module" (MODULE) 10.0.143 on page 874
⇒ "MonogenicLinearOperator" (MLO) 12.0.165 on page 1053
⇒ "RectangularMatrixCategory" (RMATCAT) 10.0.147 on page 901
⇒ "SquareMatrixCategory" (SMATCAT) 12.0.168 on page 1082
⇒ "UnivariateSkewPolynomialCategory" (OREPCAT) 10.0.150 on page 929
⇒ "XAlgebra" (XALG) 10.0.151 on page 940
⇐ "LeftModule" (LMODULE) 8.0.113 on page 655
⇐ "RightModule" (RMODULE) 8.0.120 on page 715

Exports:

0	coerce	hash	latex	sample
subtractIfCan	zero?	?~=?	?*?	?+?
?-?	-?	?=?		

Attributes Exported:

- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These exports come from (p655) LeftModule(R:Ring):

```
0 : () -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?~=? : (%,%) -> Boolean
?*? : (R,%) -> %
?=? : (%,%) -> Boolean
?+? : (%,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?*? : (Integer,%) -> %
?-? : (%,%) -> %
-? : % -> %
```

These exports come from (p715) RightModule(S:Ring):

```
 ?*? : (%,S) -> %
```

— **category BMODULE BiModule** —

```
)abbrev category BMODULE BiModule
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A \spadtype{BiModule} is both a left and right module with respect
++ to potentially different rings.
++
++ Axiom\br
++ \tab{5}\spad{ r*(x*s) = (r*x)*s }

BiModule(R:Ring,S:Ring):Category ==
  Join(LeftModule(R),RightModule(S)) with
     leftUnitary
       ++ \spad{1 * x = x}
     rightUnitary
       ++ \spad{x * 1 = x}
```

———-

— **BMODULE.dotabb** —

```
"BMODULE"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=BMODULE"];
"BMODULE" -> "LMODULE"
"BMODULE" -> "RMODULE"
```

———-

— **BMODULE.dotfull** —

```
"BiModule(a:Ring,b:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=BMODULE"];
"BiModule(a:Ring,b:Ring)" -> "LeftModule(a:Ring)"
"BiModule(a:Ring,b:Ring)" -> "RightModule(a:Ring)"
```

```
"BiModule(a:CommutativeRing,b:CommutativeRing)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=BMODULE"];
"BiModule(a:CommutativeRing,b:CommutativeRing)" -> "BiModule(a:Ring,b:Ring)"

"BiModule(a:Ring,b:OrderedAbelianMonoid)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=BMODULE"];
"BiModule(a:Ring,b:OrderedAbelianMonoid)" -> "BiModule(a:Ring,b:Ring)"
```

———-

— BMODULE.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"BiModule(a:Ring,b:Ring)" [color=lightblue];
"BiModule(a:Ring,b:Ring)" -> "LeftModule(a:Ring)"
"BiModule(a:Ring,b:Ring)" -> "RightModule(a:Ring)"

"RightModule(a:Ring)" [color=seagreen];
"RightModule(a:Ring)" -> "RightModule(a:Rng)"

"RightModule(a:Rng)" [color=lightblue];
"RightModule(a:Rng)" -> "AbelianGroup()"

"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "AbelianGroup()"

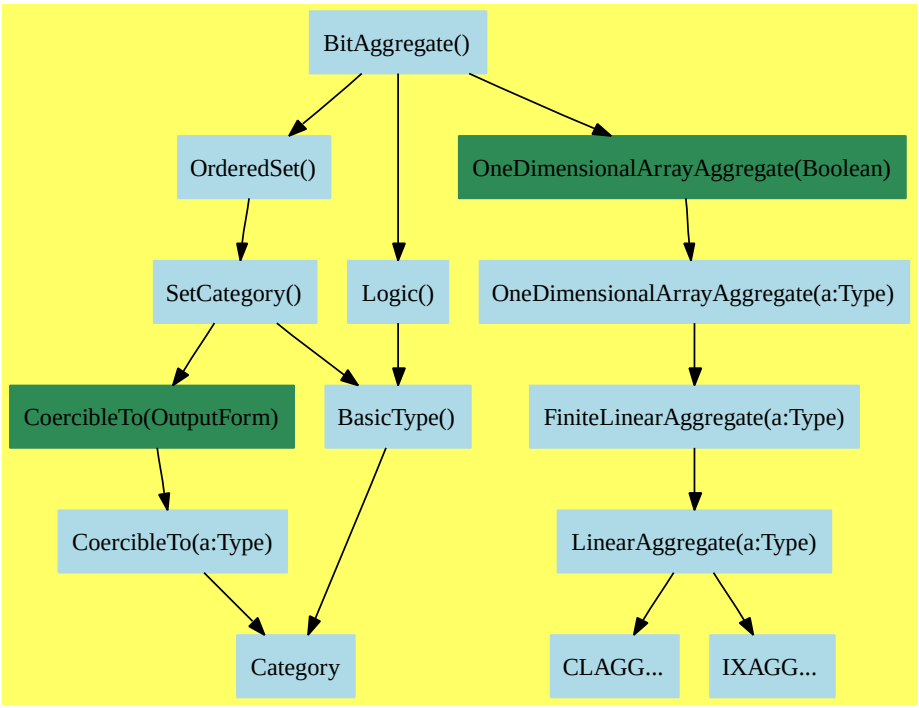
"AbelianGroup()" [color=lightblue];
"AbelianGroup()" -> "CancellationAbelianMonoid()"
"AbelianGroup()" -> "REPDB..."

"CancellationAbelianMonoid()" [color=lightblue];
"CancellationAbelianMonoid()" -> "ABELMON..."

"ABELMON..." [color=lightblue];
"REPDB..." [color="#00EE00"];
}
```

———-

9.0.123 BitAggregate (BTAGG)



— BitAggregate.input —

```
)set break resume
)sys rm -f BitAggregate.output
)spool BitAggregate.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show BitAggregate
--R BitAggregate  is a category constructor
--R Abbreviation for BitAggregate is BTAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for BTAGG
--R
--R------------------------------ Operations --------------------------------
--R ?/\? : (%,%) -> %                    ?<? : (%,%) -> Boolean
--R ?<=? : (%,%) -> Boolean              ?=? : (%,%) -> Boolean
--R ?>? : (%,%) -> Boolean               ?>=? : (%,%) -> Boolean
--R ?\/? : (%,%) -> %                    ^? : % -> %
```

```
--R ?and? : (%,%) -> %                    coerce : % -> OutputForm
--R concat : (%,Boolean) -> %             concat : (Boolean,%) -> %
--R concat : (%,%) -> %                   concat : List % -> %
--R construct : List Boolean -> %         copy : % -> %
--R delete : (%,Integer) -> %             ?.? : (%,Integer) -> Boolean
--R empty : () -> %                       empty? : % -> Boolean
--R entries : % -> List Boolean           eq? : (%,%) -> Boolean
--R hash : % -> SingleInteger             index? : (Integer,%) -> Boolean
--R indices : % -> List Integer           insert : (%,%,Integer) -> %
--R latex : % -> String                   max : (%,%) -> %
--R min : (%,%) -> %                      nand : (%,%) -> %
--R nor : (%,%) -> %                      not? : % -> %
--R ?or? : (%,%) -> %                     qelt : (%,Integer) -> Boolean
--R reverse : % -> %                      sample : () -> %
--R xor : (%,%) -> %                      ~? : % -> %
--R ?~=? : (%,%) -> Boolean
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R any? : ((Boolean -> Boolean),%) -> Boolean if $ has finiteAggregate
--R convert : % -> InputForm if Boolean has KONVERT INFORM
--R copyInto! : (%,%,Integer) -> % if $ has shallowlyMutable
--R count : ((Boolean -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R count : (Boolean,%) -> NonNegativeInteger if Boolean has SETCAT and $ has finiteAggregate
--R delete : (%,UniversalSegment Integer) -> %
--R elt : (%,Integer,Boolean) -> Boolean
--R ?.? : (%,UniversalSegment Integer) -> %
--R entry? : (Boolean,%) -> Boolean if $ has finiteAggregate and Boolean has SETCAT
--R eval : (%,List Equation Boolean) -> % if Boolean has EVALAB BOOLEAN and Boolean has SETCAT
--R eval : (%,Equation Boolean) -> % if Boolean has EVALAB BOOLEAN and Boolean has SETCAT
--R eval : (%,Boolean,Boolean) -> % if Boolean has EVALAB BOOLEAN and Boolean has SETCAT
--R eval : (%,List Boolean,List Boolean) -> % if Boolean has EVALAB BOOLEAN and Boolean has SETC
--R every? : ((Boolean -> Boolean),%) -> Boolean if $ has finiteAggregate
--R fill! : (%,Boolean) -> % if $ has shallowlyMutable
--R find : ((Boolean -> Boolean),%) -> Union(Boolean,"failed")
--R first : % -> Boolean if Integer has ORDSET
--R insert : (Boolean,%,Integer) -> %
--R less? : (%,NonNegativeInteger) -> Boolean
--R map : ((Boolean -> Boolean),%) -> %
--R map : (((Boolean,Boolean) -> Boolean),%,%) -> %
--R map! : ((Boolean -> Boolean),%) -> % if $ has shallowlyMutable
--R maxIndex : % -> Integer if Integer has ORDSET
--R member? : (Boolean,%) -> Boolean if Boolean has SETCAT and $ has finiteAggregate
--R members : % -> List Boolean if $ has finiteAggregate
--R merge : (((Boolean,Boolean) -> Boolean),%,%) -> %
--R merge : (%,%) -> % if Boolean has ORDSET
--R minIndex : % -> Integer if Integer has ORDSET
--R more? : (%,NonNegativeInteger) -> Boolean
--R new : (NonNegativeInteger,Boolean) -> %
--R parts : % -> List Boolean if $ has finiteAggregate
--R position : ((Boolean -> Boolean),%) -> Integer
--R position : (Boolean,%) -> Integer if Boolean has SETCAT
```

```
--R position : (Boolean,%,Integer) -> Integer if Boolean has SETCAT
--R qsetelt! : (%,Integer,Boolean) -> Boolean if $ has shallowlyMutable
--R reduce : (((Boolean,Boolean) -> Boolean),%,Boolean,Boolean) -> Boolean if Boolean has SETCAT and $ has
--R reduce : (((Boolean,Boolean) -> Boolean),%,Boolean) -> Boolean if $ has finiteAggregate
--R reduce : (((Boolean,Boolean) -> Boolean),%) -> Boolean if $ has finiteAggregate
--R remove : (Boolean,%) -> % if Boolean has SETCAT and $ has finiteAggregate
--R remove : ((Boolean -> Boolean),%) -> % if $ has finiteAggregate
--R removeDuplicates : % -> % if Boolean has SETCAT and $ has finiteAggregate
--R reverse! : % -> % if $ has shallowlyMutable
--R select : ((Boolean -> Boolean),%) -> % if $ has finiteAggregate
--R setelt : (%,Integer,Boolean) -> Boolean if $ has shallowlyMutable
--R setelt : (%,UniversalSegment Integer,Boolean) -> Boolean if $ has shallowlyMutable
--R size? : (%,NonNegativeInteger) -> Boolean
--R sort : (((Boolean,Boolean) -> Boolean),%) -> %
--R sort : % -> % if Boolean has ORDSET
--R sort! : (((Boolean,Boolean) -> Boolean),%) -> % if $ has shallowlyMutable
--R sort! : % -> % if Boolean has ORDSET and $ has shallowlyMutable
--R sorted? : (((Boolean,Boolean) -> Boolean),%) -> Boolean
--R sorted? : % -> Boolean if Boolean has ORDSET
--R swap! : (%,Integer,Integer) -> Void if $ has shallowlyMutable
--R
--E 1

)spool
)lisp (bye)
```

———-

— **BitAggregate.help** —

```
=====================================================================
BitAggregate examples
=====================================================================

See Also:
o )show BitAggregate
```

———-

See:

⇐ “Logic” (LOGIC) 3.0.29 on page 95
⇐ “OneDimensionalArrayAggregate” (A1AGG) 8.0.117 on page 684
⇐ “OrderedSet” (ORDSET) 4.0.56 on page 216

Exports:

any?	coerce	concat	construct	convert
copy	copyInto!	count	delete	elt
empty	empty?	entry?	entries	eq?
eval	every?	fill!	find	first
hash	index?	indices	insert	latex
less?	map	map!	max	maxIndex
member?	members	merge	min	minIndex
more?	nand	new	nor	not?
parts	position	qelt	qsetelt!	reduce
remove	removeDuplicates	reverse	reverse!	sample
select	setelt	size?	sort	sort!
sorted?	swap!	xor	#?	?/\?
?<?	?<=?	?=?	?>?	?>=?
?\/?	^?	?and?	?.?	?or?
~?	?~=?			

Attributes exported:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **nil**

These are directly exported but not implemented:

```
?and? : (%,%) -> %
?or? : (%,%) -> %
xor : (%,%) -> %
```

These are implemented by this category:

```
not? : % -> %
^? : % -> %
~? : % -> %
?/\? : (%,%) -> %
?\/? : (%,%) -> %
nand : (%,%) -> %
nor : (%,%) -> %
```

These exports come from (p216) OrderedSet():

```
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
max : (%,%) -> %
min : (%,%) -> %
```

```
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?<? : (%,%) -> Boolean
?>? : (%,%) -> Boolean
?<=? : (%,%) -> Boolean
?>=? : (%,%) -> Boolean
```

TPDHERE: Note that none of the exports of Logic are needed. Perhaps this can be eliminated.

These exports come from (p684) OneDimensionalArrayAggregate(Boolean):

```
any? : ((Boolean -> Boolean),%) -> Boolean
    if $ has finiteAggregate
concat : (%,%) -> %
concat : List % -> %
concat : (%,Boolean) -> %
concat : (Boolean,%) -> %
construct : List Boolean -> %
convert : % -> InputForm
    if Boolean has KONVERT INFORM
copy : % -> %
copyInto! : (%,%,Integer) -> %
    if $ has shallowlyMutable
count : (Boolean,%) -> NonNegativeInteger
    if Boolean has SETCAT and $ has finiteAggregate
count : ((Boolean -> Boolean),%) -> NonNegativeInteger
    if $ has finiteAggregate
delete : (%,UniversalSegment Integer) -> %
delete : (%,Integer) -> %
elt : (%,Integer,Boolean) -> Boolean
empty : () -> %
empty? : % -> Boolean
entry? : (Boolean,%) -> Boolean
    if $ has finiteAggregate and Boolean has SETCAT
entries : % -> List Boolean
eq? : (%,%) -> Boolean
eval : (%,List Equation Boolean) -> %
    if Boolean has EVALAB BOOLEAN and Boolean has SETCAT
eval : (%,Equation Boolean) -> %
    if Boolean has EVALAB BOOLEAN and Boolean has SETCAT
eval : (%,Boolean,Boolean) -> %
    if Boolean has EVALAB BOOLEAN and Boolean has SETCAT
eval : (%,List Boolean,List Boolean) -> %
    if Boolean has EVALAB BOOLEAN and Boolean has SETCAT
every? : ((Boolean -> Boolean),%) -> Boolean
    if $ has finiteAggregate
fill! : (%,Boolean) -> %
    if $ has shallowlyMutable
find : ((Boolean -> Boolean),%) -> Union(Boolean,"failed")
```

```
first : % -> Boolean if Integer has ORDSET
index? : (Integer,%) -> Boolean
indices : % -> List Integer
insert : (Boolean,%,Integer) -> %
insert : (%,%,Integer) -> %
less? : (%,NonNegativeInteger) -> Boolean
map : ((Boolean -> Boolean),%) -> %
map : (((Boolean,Boolean) -> Boolean),%,%) -> %
map! : ((Boolean -> Boolean),%) -> %
    if $ has shallowlyMutable
maxIndex : % -> Integer if Integer has ORDSET
member? : (Boolean,%) -> Boolean
    if Boolean has SETCAT and $ has finiteAggregate
members : % -> List Boolean if $ has finiteAggregate
merge : (%,%) -> % if Boolean has ORDSET
minIndex : % -> Integer if Integer has ORDSET
more? : (%,NonNegativeInteger) -> Boolean
new : (NonNegativeInteger,Boolean) -> %
merge : (((Boolean,Boolean) -> Boolean),%,%) -> %
parts : % -> List Boolean if $ has finiteAggregate
position : ((Boolean -> Boolean),%) -> Integer
position : (Boolean,%,Integer) -> Integer
    if Boolean has SETCAT
position : (Boolean,%) -> Integer
    if Boolean has SETCAT
qelt : (%,Integer) -> Boolean
qsetelt! : (%,Integer,Boolean) -> Boolean
    if $ has shallowlyMutable
reverse : % -> %
reduce : (((Boolean,Boolean) -> Boolean),%) -> Boolean
    if $ has finiteAggregate
reduce : (((Boolean,Boolean) -> Boolean),%,Boolean) -> Boolean
    if $ has finiteAggregate
reduce :
 (((Boolean,Boolean) -> Boolean),%,Boolean,Boolean) -> Boolean
    if Boolean has SETCAT and $ has finiteAggregate
remove : (Boolean,%) -> %
    if Boolean has SETCAT and $ has finiteAggregate
remove : ((Boolean -> Boolean),%) -> %
    if $ has finiteAggregate
removeDuplicates : % -> %
    if Boolean has SETCAT and $ has finiteAggregate
reverse! : % -> % if $ has shallowlyMutable
sample : () -> %
setelt : (%,UniversalSegment Integer,Boolean) -> Boolean
    if $ has shallowlyMutable
select : ((Boolean -> Boolean),%) -> %
    if $ has finiteAggregate
setelt : (%,Integer,Boolean) -> Boolean
    if $ has shallowlyMutable
```

```
 size? : (%,NonNegativeInteger) -> Boolean
 sort : (((Boolean,Boolean) -> Boolean),%) -> %
 sort : % -> % if Boolean has ORDSET
 sort! : % -> %
     if Boolean has ORDSET and $ has shallowlyMutable
 sort! : (((Boolean,Boolean) -> Boolean),%) -> %
     if $ has shallowlyMutable
 sorted? : % -> Boolean if Boolean has ORDSET
 sorted? : (((Boolean,Boolean) -> Boolean),%) -> Boolean
 swap! : (%,Integer,Integer) -> Void if $ has shallowlyMutable
 ?.? : (%,UniversalSegment Integer) -> %
 #? : % -> NonNegativeInteger if $ has finiteAggregate
 ?.? : (%,Integer) -> Boolean
```

— category BTAGG BitAggregate —

```
)abbrev category BTAGG BitAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The bit aggregate category models aggregates representing large
++ quantities of Boolean data.

BitAggregate(): Category ==
  Join(OrderedSet, Logic, OneDimensionalArrayAggregate Boolean) with
    "not": % -> %
      ++ not(b) returns the logical not of bit aggregate
      ++ \axiom{b}.
    "^"  : % -> %
      ++ ^ b returns the logical not of bit aggregate
      ++ \axiom{b}.
    nand : (%, %) -> %
      ++ nand(a,b) returns the logical nand of bit aggregates
      ++ \axiom{a} and \axiom{b}.
    nor         : (%, %) -> %
      ++ nor(a,b) returns the logical nor of bit aggregates
      ++ \axiom{a} and \axiom{b}.
    _and : (%, %) -> %
      ++ a and b returns the logical and of bit aggregates
      ++ \axiom{a} and \axiom{b}.
    _or         : (%, %) -> %
      ++ a or b returns the logical or of bit aggregates
```

```
      ++ \axiom{a} and \axiom{b}.
    xor         : (%, %) -> %
      ++ xor(a,b) returns the logical exclusive-or of bit aggregates
      ++ \axiom{a} and \axiom{b}.

 add
   not v      == map(_not, v)
   _^ v              == map(_not, v)
   _~(v)      == map(_~, v)
   _/_\(v, u) == map(_/_\, v, u)
   _\_/(v, u) == map(_\_/, v, u)
   nand(v, u) == map(nand, v, u)
   nor(v, u)  == map(nor, v, u)
```

———-

— BTAGG.dotabb —

```
"BTAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=BTAGG"];
"BTAGG" -> "ORDSET"
"BTAGG" -> "LOGIC"
"BTAGG" -> "A1AGG"
```

———-

— BTAGG.dotfull —

```
"BitAggregate()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=BTAGG"];
"BitAggregate()" -> "OrderedSet()"
"BitAggregate()" -> "Logic()"
"BitAggregate()" -> "OneDimensionalArrayAggregate(Boolean)"
```

———-

— BTAGG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"BitAggregate()" [color=lightblue];
"BitAggregate()" -> "OrderedSet()"
"BitAggregate()" -> "Logic()"
```

```
"BitAggregate()" -> "OneDimensionalArrayAggregate(Boolean)"

"OneDimensionalArrayAggregate(Boolean)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=A1AGG"];
"OneDimensionalArrayAggregate(Boolean)" ->
    "OneDimensionalArrayAggregate(a:Type)"

"OneDimensionalArrayAggregate(a:Type)" [color=lightblue];
"OneDimensionalArrayAggregate(a:Type)" ->
    "FiniteLinearAggregate(a:Type)"

"FiniteLinearAggregate(a:Type)" [color=lightblue];
"FiniteLinearAggregate(a:Type)" -> "LinearAggregate(a:Type)"

"LinearAggregate(a:Type)" [color=lightblue];
"LinearAggregate(a:Type)" -> "IXAGG..."
"LinearAggregate(a:Type)" -> "CLAGG..."

"OrderedSet()" [color=lightblue];
"OrderedSet()" -> "SetCategory()"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Logic()" [color=lightblue];
"Logic()" -> "BasicType()"

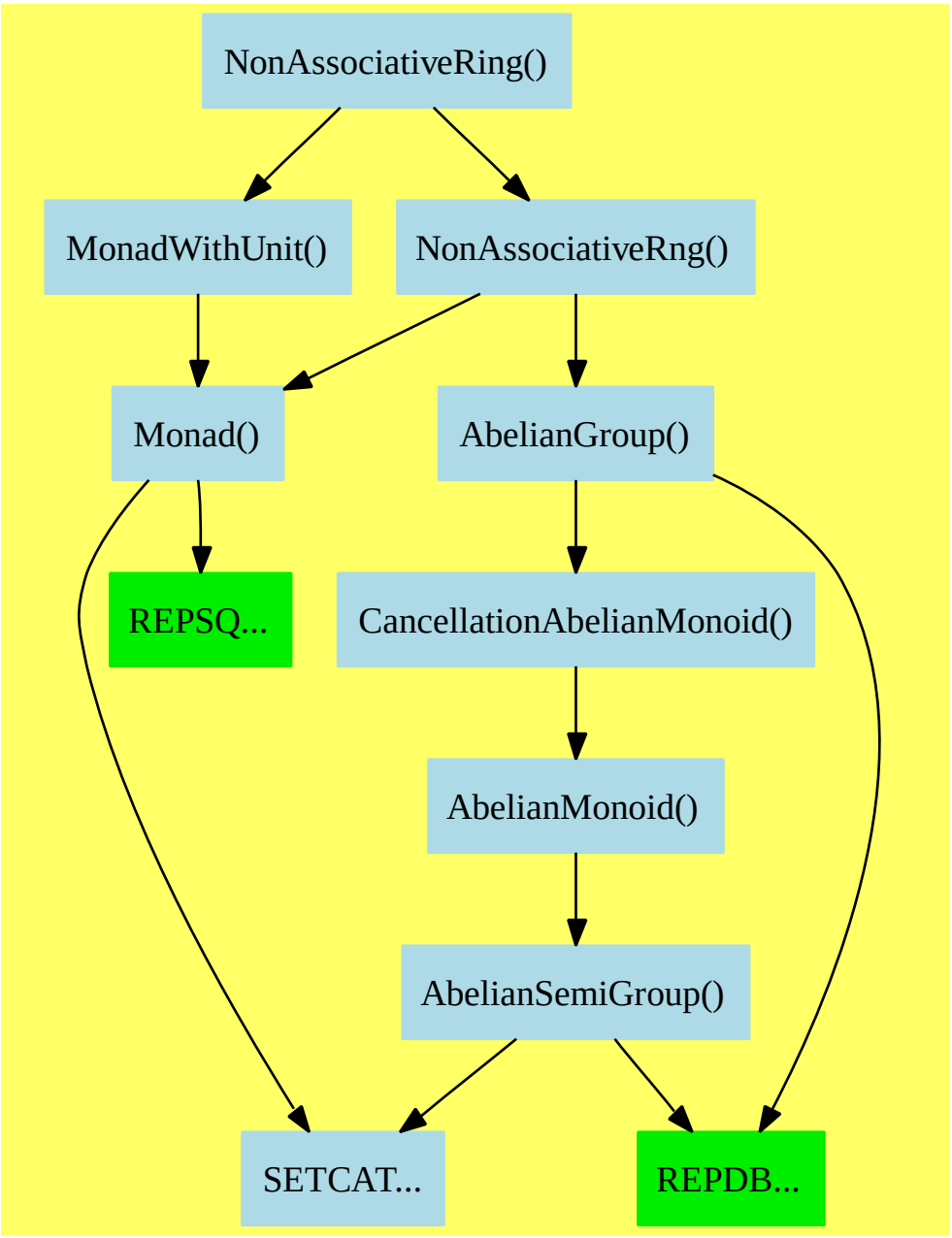
"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"Category" [color=lightblue];

"CLAGG..." [color=lightblue];
"IXAGG..." [color=lightblue];
}
```

———-

9.0.124 NonAssociativeRing (NASRING)



— **NonAssociativeRing.input** —

```
)set break resume
)sys rm -f NonAssociativeRing.output
)spool NonAssociativeRing.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show NonAssociativeRing
--R NonAssociativeRing  is a category constructor
--R Abbreviation for NonAssociativeRing is NASRING
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for NASRING
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R antiCommutator : (%,%) -> %           associator : (%,%,%) -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R commutator : (%,%) -> %               hash : % -> SingleInteger
--R latex : % -> String                   one? : % -> Boolean
--R recip : % -> Union(%,"failed")        sample : () -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R leftPower : (%,NonNegativeInteger) -> %
--R leftPower : (%,PositiveInteger) -> %
--R leftRecip : % -> Union(%,"failed")
--R rightPower : (%,NonNegativeInteger) -> %
--R rightPower : (%,PositiveInteger) -> %
--R rightRecip : % -> Union(%,"failed")
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— NonAssociativeRing.help —

```
====================================================================
NonAssociativeRing examples
====================================================================
```

```
See Also:
o )show NonAssociativeRing
```

———-

See:

⇐ “MonadWithUnit” (MONADWU) 5.0.75 on page 325
⇐ “NonAssociativeRng” (NARNG) 8.0.116 on page 679

Exports:

0	1	antiCommutator	associator	characteristic
coerce	commutator	hash	latex	leftPower
leftRecip	one?	recip	rightPower	rightRecip
sample	subtractIfCan	zero?	?*?	?~=?
?**?	?+?	?-?	-?	?=?

These are directly exported but not implemented:

```
characteristic : () -> NonNegativeInteger
```

These are implemented by this category:

```
coerce : Integer -> %
```

These exports come from (p679) NonAssociativeRng():

```
0 : () -> %
antiCommutator : (%,%) -> %
associator : (%,%,%) -> %
coerce : % -> OutputForm
commutator : (%,%) -> %
hash : % -> SingleInteger
latex : % -> String
leftPower : (%,PositiveInteger) -> %
rightPower : (%,PositiveInteger) -> %
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?~=? : (%,%) -> Boolean
?*? : (PositiveInteger,%) -> %
?*? : (%,%) -> %
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?*? : (Integer,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
```

```
 -? : % -> %
 ?**? : (%,PositiveInteger) -> %
```

These exports come from (p325) MonadWithUnit():

```
 1 : () -> %
 leftPower : (%,NonNegativeInteger) -> %
 leftRecip : % -> Union(%,"failed")
 one? : % -> Boolean
 recip : % -> Union(%,"failed")
 rightPower : (%,NonNegativeInteger) -> %
 rightRecip : % -> Union(%,"failed")
 ?**? : (%,NonNegativeInteger) -> %
```

— category NASRING NonAssociativeRing —

```
)abbrev category NASRING NonAssociativeRing
++ Author: J. Grabmeier, R. Wisbauer
++ Date Created: 01 March 1991
++ Date Last Updated: 11 June 1991
++ Basic Operations: +, *, -, **
++ Related Constructors: NonAssociativeRng, Rng, Ring
++ Also See:
++ AMS Classifications:
++ Keywords: non-associative ring with unit
++ Reference:
++  R.D. Schafer: An Introduction to Nonassociative Algebras
++  Academic Press, New York, 1966
++ Description:
++ A NonAssociativeRing is a non associative rng which has a unit,
++ the multiplication is not necessarily commutative or associative.

NonAssociativeRing(): Category == Join(NonAssociativeRng,MonadWithUnit) with
      characteristic: -> NonNegativeInteger
        ++ characteristic() returns the characteristic of the ring.
        --we can not make this a constant, since some domains are mutable
      coerce: Integer -> %
        ++ coerce(n) coerces the integer n to an element of the ring.
   add
      n:Integer
      coerce(n) == n * 1$%
```

———-

— NASRING.dotabb —

```
"NASRING"
```

```
 [color=lightblue,href="bookvol10.2.pdf#nameddest=NASRING"];
"NASRING" -> "MONADWU"
"NASRING" -> "NARNG"
```

———-

— **NASRING.dotfull** —

```
"NonAssociativeRing()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=NASRING"];
"NonAssociativeRing()" -> "NonAssociativeRng()"
"NonAssociativeRing()" -> "MonadWithUnit()"
```

———-

— **NASRING.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"NonAssociativeRing()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=NASRING"];
"NonAssociativeRing()" -> "NonAssociativeRng()"
"NonAssociativeRing()" -> "MonadWithUnit()"

"MonadWithUnit()" [color=lightblue];
"MonadWithUnit()" -> "Monad()"

"NonAssociativeRng()" [color=lightblue];
"NonAssociativeRng()" -> "AbelianGroup()"
"NonAssociativeRng()" -> "Monad()"

"Monad()" [color=lightblue];
"Monad()" -> "SETCAT..."
"Monad()" -> "REPSQ..."

"AbelianGroup()" [color=lightblue];
"AbelianGroup()" -> "CancellationAbelianMonoid()"
"AbelianGroup()" -> "REPDB..."

"CancellationAbelianMonoid()" [color=lightblue];
"CancellationAbelianMonoid()" -> "AbelianMonoid()"

"AbelianMonoid()" [color=lightblue];
```

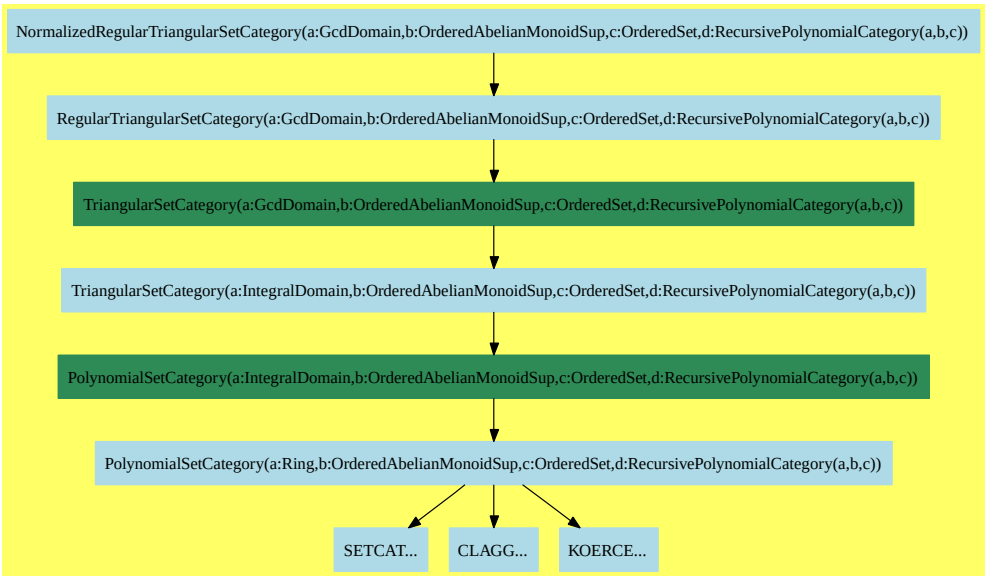
```
"AbelianMonoid()" -> "AbelianSemiGroup()"

"AbelianSemiGroup()" [color=lightblue];
"AbelianSemiGroup()" -> "SETCAT..."
"AbelianSemiGroup()" -> "REPDB..."

"REPDB..." [color="#00EE00"];
"REPSQ..." [color="#00EE00"];
"SETCAT..." [color=lightblue];
}
```

———-

9.0.125 NormalizedTriangularSetCategory (NTSCAT)



— NormalizedTriangularSetCategory.input —

```
)set break resume
)sys rm -f NormalizedTriangularSetCategory.output
)spool NormalizedTriangularSetCategory.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 1
)show NormalizedTriangularSetCategory
--R NormalizedTriangularSetCategory(R: GcdDomain,E: OrderedAbelianMonoidSup,V: OrderedSet,P: Rec
--R Abbreviation for NormalizedTriangularSetCategory is NTSCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for NTSCAT
--R
--R------------------------------ Operations --------------------------------
--R ?=? : (%,%) -> Boolean               algebraic? : (V,%) -> Boolean
--R algebraicVariables : % -> List V     augment : (List P,%) -> List %
--R augment : (P,List %) -> List %       augment : (P,%) -> List %
--R coerce : % -> List P                 coerce : % -> OutputForm
--R collect : (%,V) -> %                 collectQuasiMonic : % -> %
--R collectUnder : (%,V) -> %            collectUpper : (%,V) -> %
--R construct : List P -> %              copy : % -> %
--R degree : % -> NonNegativeInteger     empty : () -> %
--R empty? : % -> Boolean                eq? : (%,%) -> Boolean
--R extend : (List P,%) -> List %        extend : (P,List %) -> List %
--R extend : (P,%) -> List %             extend : (%,P) -> %
--R first : % -> Union(P,"failed")       hash : % -> SingleInteger
--R headReduce : (P,%) -> P              headReduced? : % -> Boolean
--R headReduced? : (P,%) -> Boolean      infRittWu? : (%,%) -> Boolean
--R initiallyReduce : (P,%) -> P         initiallyReduced? : % -> Boolean
--R initials : % -> List P               internalAugment : (P,%) -> %
--R intersect : (P,List %) -> List %     intersect : (List P,%) -> List %
--R intersect : (P,%) -> List %          invertible? : (P,%) -> Boolean
--R invertibleSet : (P,%) -> List %      last : % -> Union(P,"failed")
--R latex : % -> String                  mainVariable? : (V,%) -> Boolean
--R mainVariables : % -> List V          map : ((P -> P),%) -> %
--R mvar : % -> V                        normalized? : % -> Boolean
--R normalized? : (P,%) -> Boolean       purelyAlgebraic? : % -> Boolean
--R reduceByQuasiMonic : (P,%) -> P      removeZero : (P,%) -> P
--R rest : % -> Union(%,"failed")        retract : List P -> %
--R sample : () -> %                     stronglyReduce : (P,%) -> P
--R stronglyReduced? : % -> Boolean      trivialIdeal? : % -> Boolean
--R variables : % -> List V              zeroSetSplit : List P -> List %
--R ?~=? : (%,%) -> Boolean
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R algebraicCoefficients? : (P,%) -> Boolean
--R any? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
--R augment : (List P,List %) -> List %
--R autoReduced? : (%,((P,List P) -> Boolean)) -> Boolean
--R basicSet : (List P,(P -> Boolean),((P,P) -> Boolean)) -> Union(Record(bas: %,top: List P),"f
--R basicSet : (List P,((P,P) -> Boolean)) -> Union(Record(bas: %,top: List P),"failed")
--R coHeight : % -> NonNegativeInteger if V has FINITE
--R convert : % -> InputForm if P has KONVERT INFORM
--R count : ((P -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R count : (P,%) -> NonNegativeInteger if P has SETCAT and $ has finiteAggregate
--R eval : (%,List Equation P) -> % if P has EVALAB P and P has SETCAT
--R eval : (%,Equation P) -> % if P has EVALAB P and P has SETCAT
```

```
--R eval : (%,P,P) -> % if P has EVALAB P and P has SETCAT
--R eval : (%,List P,List P) -> % if P has EVALAB P and P has SETCAT
--R every? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
--R extend : (List P,List %) -> List %
--R extendIfCan : (%,P) -> Union(%,"failed")
--R find : ((P -> Boolean),%) -> Union(P,"failed")
--R headRemainder : (P,%) -> Record(num: P,den: R) if R has INTDOM
--R initiallyReduced? : (P,%) -> Boolean
--R internalAugment : (List P,%) -> %
--R intersect : (List P,List %) -> List %
--R invertible? : (P,%) -> List Record(val: Boolean,tower: %)
--R invertibleElseSplit? : (P,%) -> Union(Boolean,List %)
--R lastSubResultant : (P,P,%) -> List Record(val: P,tower: %)
--R lastSubResultantElseSplit : (P,P,%) -> Union(P,List %)
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((P -> P),%) -> % if $ has shallowlyMutable
--R member? : (P,%) -> Boolean if P has SETCAT and $ has finiteAggregate
--R members : % -> List P if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List P if $ has finiteAggregate
--R purelyAlgebraic? : (P,%) -> Boolean
--R purelyAlgebraicLeadingMonomial? : (P,%) -> Boolean
--R purelyTranscendental? : (P,%) -> Boolean
--R quasiComponent : % -> Record(close: List P,open: List P)
--R reduce : (P,%,((P,P) -> P),((P,P) -> Boolean)) -> P
--R reduce : (((P,P) -> P),%) -> P if $ has finiteAggregate
--R reduce : (((P,P) -> P),%,P) -> P if $ has finiteAggregate
--R reduce : (((P,P) -> P),%,P,P) -> P if P has SETCAT and $ has finiteAggregate
--R reduced? : (P,%,((P,P) -> Boolean)) -> Boolean
--R remainder : (P,%) -> Record(rnum: R,polnum: P,den: R) if R has INTDOM
--R remove : ((P -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (P,%) -> % if P has SETCAT and $ has finiteAggregate
--R removeDuplicates : % -> % if P has SETCAT and $ has finiteAggregate
--R retractIfCan : List P -> Union(%,"failed")
--R rewriteIdealWithHeadRemainder : (List P,%) -> List P if R has INTDOM
--R rewriteIdealWithRemainder : (List P,%) -> List P if R has INTDOM
--R rewriteSetWithReduction : (List P,%,((P,P) -> P),((P,P) -> Boolean)) -> List P
--R roughBase? : % -> Boolean if R has INTDOM
--R roughEqualIdeals? : (%,%) -> Boolean if R has INTDOM
--R roughSubIdeal? : (%,%) -> Boolean if R has INTDOM
--R roughUnitIdeal? : % -> Boolean if R has INTDOM
--R select : (%,V) -> Union(P,"failed")
--R select : ((P -> Boolean),%) -> % if $ has finiteAggregate
--R size? : (%,NonNegativeInteger) -> Boolean
--R sort : (%,V) -> Record(under: %,floor: %,upper: %)
--R squareFreePart : (P,%) -> List Record(val: P,tower: %)
--R stronglyReduced? : (P,%) -> Boolean
--R triangular? : % -> Boolean if R has INTDOM
--R zeroSetSplit : (List P,Boolean) -> List %
--R zeroSetSplitIntoTriangularSystems : List P -> List Record(close: %,open: List P)
```

```
--R
--E 1

)spool
)lisp (bye)
```

———-

— **NormalizedTriangularSetCategory.help** —

```
==================================================================
NormalizedTriangularSetCategory examples
==================================================================

See Also:
o )show NormalizedTriangularSetCategory
```

———-

See:

⇒ “SquareFreeNormalizedTriangularSetCategory” (SNTSCAT) 10.0.148 on page 910
⇐ “RegularTriangularSetCategory” (RSETCAT) 8.0.119 on page 700

Exports:

algebraic?
algebraicCoefficients?
algebraicVariables
any?
augment
autoReduced?
basicSet
coerce
coHeight
collect
collectQuasiMonic
collectUnder
collectUpper
construct
copy
convert
count
degree
empty
empty?
eq?
eval
every?
extend
extendIfCan
find
first
hash
headReduce
headReduced?
headRemainder
infRittWu?
initiallyReduce
initiallyReduced?
initials
internalAugment
intersect
invertible?
invertibleElseSplit?
invertibleSet
last
lastSubResultant
lastSubResultantElseSplit
latex
less?
mainVariable?
mainVariables
map
map!
member?
members
more?
mvar
normalized?
parts
purelyAlgebraic?
purelyAlgebraicLeadingMonomial?
purelyTranscendental?
quasiComponent
reduce
reduceByQuasiMonic
reduced?
remainder
remove
removeDuplicates
removeZero
rest
retract
retractIfCan
rewriteIdealWithHeadRemainder
rewriteIdealWithRemainder
rewriteSetWithReduction
roughBase?
roughEqualIdeals?
roughSubIdeal?
roughUnitIdeal?
sample
select
size?
sort
squareFreePart
stronglyReduce
stronglyReduced?
triangular?
trivialIdeal?
variables
zeroSetSplit
zeroSetSplitIntoTriangularSystems
#?
?=?
?~=?

Attributes Exported:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **nil**

These exports come from (p700) RegularTriangularSetCategory(R,E,V,P) where R:GcdDomain, E:OrderedAbelianMonoidSup, V:OrderedSet, P:RecursivePolynomialCategory(R,E,V)):

```
algebraic? : (V,%) -> Boolean
algebraicCoefficients? : (P,%) -> Boolean
algebraicVariables : % -> List V
any? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
augment : (List P,List %) -> List %
augment : (List P,%) -> List %
augment : (P,List %) -> List %
augment : (P,%) -> List %
autoReduced? : (%,((P,List P) -> Boolean)) -> Boolean
basicSet :
 (List P,(P -> Boolean),((P,P) -> Boolean)) ->
   Union(Record(bas: %,top: List P),"failed")
basicSet :
 (List P,((P,P) -> Boolean)) ->
   Union(Record(bas: %,top: List P),"failed")
coerce : % -> List P
coerce : % -> OutputForm
coHeight : % -> NonNegativeInteger if V has FINITE
collect : (%,V) -> %
collectQuasiMonic : % -> %
collectUnder : (%,V) -> %
collectUpper : (%,V) -> %
construct : List P -> %
copy : % -> %
convert : % -> InputForm if P has KONVERT INFORM
count : ((P -> Boolean),%) -> NonNegativeInteger
  if $ has finiteAggregate
count : (P,%) -> NonNegativeInteger
  if P has SETCAT
  and $ has finiteAggregate
degree : % -> NonNegativeInteger
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List Equation P) -> % if P has EVALAB P and P has SETCAT
eval : (%,Equation P) -> % if P has EVALAB P and P has SETCAT
```

```
eval : (%,P,P) -> % if P has EVALAB P and P has SETCAT
eval : (%,List P,List P) -> % if P has EVALAB P and P has SETCAT
every? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
extend : (List P,List %) -> List %
extend : (List P,%) -> List %
extend : (P,List %) -> List %
extend : (P,%) -> List %
extend : (%,P) -> %
extendIfCan : (%,P) -> Union(%,"failed")
find : ((P -> Boolean),%) -> Union(P,"failed")
first : % -> Union(P,"failed")
hash : % -> SingleInteger
headReduce : (P,%) -> P
headReduced? : % -> Boolean
headReduced? : (P,%) -> Boolean
headRemainder : (P,%) -> Record(num: P,den: R) if R has INTDOM
infRittWu? : (%,%) -> Boolean
initiallyReduce : (P,%) -> P
initiallyReduced? : % -> Boolean
initiallyReduced? : (P,%) -> Boolean
initials : % -> List P
internalAugment : (P,%) -> %
internalAugment : (List P,%) -> %
intersect : (P,List %) -> List %
intersect : (List P,%) -> List %
intersect : (P,%) -> List %
intersect : (List P,List %) -> List %
invertible? : (P,%) -> Boolean
invertible? : (P,%) -> List Record(val: Boolean,tower: %)
invertibleElseSplit? : (P,%) -> Union(Boolean,List %)
invertibleSet : (P,%) -> List %
last : % -> Union(P,"failed")
lastSubResultant : (P,P,%) -> List Record(val: P,tower: %)
lastSubResultantElseSplit : (P,P,%) -> Union(P,List %)
latex : % -> String
less? : (%,NonNegativeInteger) -> Boolean
mainVariable? : (V,%) -> Boolean
mainVariables : % -> List V
map : ((P -> P),%) -> %
map! : ((P -> P),%) -> % if $ has shallowlyMutable
member? : (P,%) -> Boolean if P has SETCAT and $ has finiteAggregate
members : % -> List P if $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
mvar : % -> V
normalized? : % -> Boolean
normalized? : (P,%) -> Boolean
parts : % -> List P if $ has finiteAggregate
purelyAlgebraic? : (P,%) -> Boolean
purelyAlgebraic? : % -> Boolean
purelyAlgebraicLeadingMonomial? : (P,%) -> Boolean
```

```
purelyTranscendental? : (P,%) -> Boolean
quasiComponent : % -> Record(close: List P,open: List P)
reduce : (P,%,((P,P) -> P),((P,P) -> Boolean)) -> P
reduce : (((P,P) -> P),%) -> P if $ has finiteAggregate
reduce : (((P,P) -> P),%,P) -> P if $ has finiteAggregate
reduce : (((P,P) -> P),%,P,P) -> P
  if P has SETCAT
  and $ has finiteAggregate
reduceByQuasiMonic : (P,%) -> P
reduced? : (P,%,((P,P) -> Boolean)) -> Boolean
remainder : (P,%) -> Record(rnum: R,polnum: P,den: R)
  if R has INTDOM
remove : ((P -> Boolean),%) -> % if $ has finiteAggregate
remove : (P,%) -> % if P has SETCAT and $ has finiteAggregate
removeDuplicates : % -> % if P has SETCAT and $ has finiteAggregate
removeZero : (P,%) -> P
rest : % -> Union(%,"failed")
retract : List P -> %
retractIfCan : List P -> Union(%,"failed")
rewriteIdealWithHeadRemainder : (List P,%) -> List P if R has INTDOM
rewriteIdealWithRemainder : (List P,%) -> List P if R has INTDOM
rewriteSetWithReduction :
   (List P,%,((P,P) -> P),((P,P) -> Boolean)) -> List P
roughBase? : % -> Boolean if R has INTDOM
roughEqualIdeals? : (%,%) -> Boolean if R has INTDOM
roughSubIdeal? : (%,%) -> Boolean if R has INTDOM
roughUnitIdeal? : % -> Boolean if R has INTDOM
sample : () -> %
select : (%,V) -> Union(P,"failed")
select : ((P -> Boolean),%) -> % if $ has finiteAggregate
size? : (%,NonNegativeInteger) -> Boolean
sort : (%,V) -> Record(under: %,floor: %,upper: %)
squareFreePart : (P,%) -> List Record(val: P,tower: %)
stronglyReduce : (P,%) -> P
stronglyReduced? : (P,%) -> Boolean
stronglyReduced? : % -> Boolean
triangular? : % -> Boolean if R has INTDOM
trivialIdeal? : % -> Boolean
variables : % -> List V
zeroSetSplit : List P -> List %
zeroSetSplit : (List P,Boolean) -> List %
zeroSetSplitIntoTriangularSystems :
  List P -> List Record(close: %,open: List P)
#? : % -> NonNegativeInteger if $ has finiteAggregate
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
```

— category NTSCAT NormalizedTriangularSetCategory —

```
)abbrev category NTSCAT NormalizedTriangularSetCategory
++ Author: Marc Moreno Maza
++ Date Created: 10/07/1998
++ Date Last Updated: 12/12/1998
++ Basic Functions:
++ Related Constructors:
++ Also See: essai Graphisme
++ AMS Classifications:
++ Keywords: polynomial, multivariate, ordered variables set
++ References :
++  [1] D. LAZARD "A new method for solving algebraic systems of
++      positive dimension" Discr. App. Math. 33:147-160,1991
++  [2] P. AUBRY, D. LAZARD and M. MORENO MAZA "On the Theories
++      of Triangular Sets" Journal of Symbol. Comp. (to appear)
++  [3] M. MORENO MAZA and R. RIOBOO "Computations of gcd over
++      algebraic towers of simple extensions" In proceedings of AAECC11
++      Paris, 1995.
++  [4] M. MORENO MAZA "Calculs de pgcd au-dessus des tours
++      d'extensions simples et resolution des systemes d'equations
++      algebriques" These, Universite P.etM. Curie, Paris, 1997.
++ Description:
++ The category of normalized triangular sets. A triangular
++ set ts is said normalized if for every algebraic
++ variable v of ts the polynomial select(ts,v)
++ is normalized w.r.t. every polynomial in collectUnder(ts,v).
++ A polynomial p is said normalized w.r.t. a non-constant
++ polynomial q if p is constant or degree(p,mdeg(q)) = 0
++ and init(p) is normalized w.r.t. q. One of the important
++ features of normalized triangular sets is that they are regular sets.

NormalizedTriangularSetCategory(R:GcdDomain,E:OrderedAbelianMonoidSup,_
 V:OrderedSet,P:RecursivePolynomialCategory(R,E,V)):
         Category ==  RegularTriangularSetCategory(R,E,V,P)
```

———-

— NTSCAT.dotabb —

```
"NTSCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=NTSCAT"];
"NTSCAT" -> "RSETCAT"
```

———-

— NTSCAT.dotfull —

```
"NormalizedRegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyn
```

```
 [color=lightblue,href="bookvol10.2.pdf#nameddest=NTSCAT"];
"NormalizedRegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:Rec
 ->
"RegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePoly
```

———-

— NTSCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"NormalizedRegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:Rec
 [color=lightblue];
"NormalizedRegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:Rec
 ->
"RegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePoly

"RegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePoly
 [color=lightblue];
"RegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePoly
  ->
"TriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialC

"TriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialC
 [color=seagreen];
"TriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialC
 ->
"TriangularSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
"TriangularSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
 [color=lightblue];
"TriangularSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
  -> "PolynomialSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursiveP

"PolynomialSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
 [color=seagreen];
"PolynomialSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
  -> "PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialC

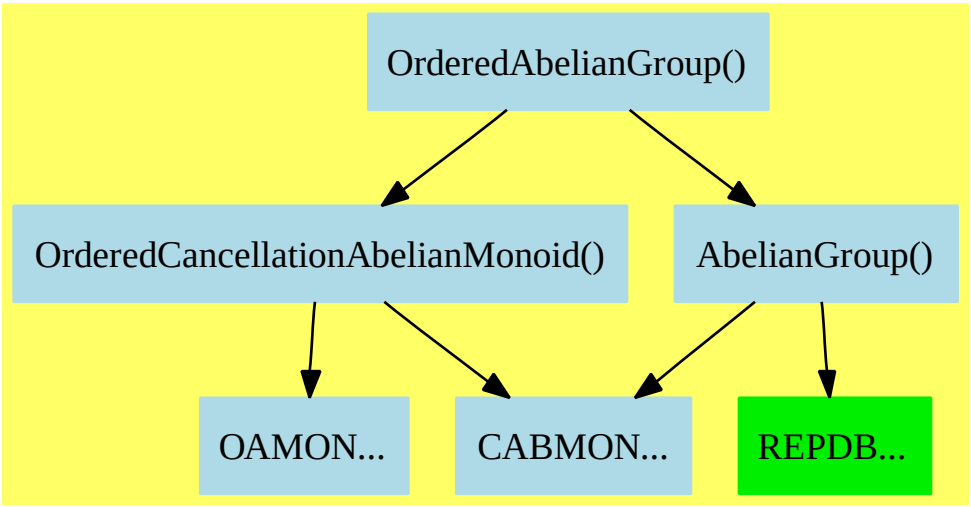
"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCatego
 [color=lightblue];
"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCatego
 -> "SETCAT..."
"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCatego
 -> "CLAGG..."
"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCatego
```

```
 -> "KOERCE..."

"SETCAT..." [color=lightblue];
"KOERCE..." [color=lightblue];
"CLAGG..." [color=lightblue];
}
```

———-

9.0.126 OrderedAbelianGroup (OAGROUP)



— **OrderedAbelianGroup.input** —

```
)set break resume
)sys rm -f OrderedAbelianGroup.output
)spool OrderedAbelianGroup.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show OrderedAbelianGroup
--R OrderedAbelianGroup  is a category constructor
--R Abbreviation for OrderedAbelianGroup is OAGROUP
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for OAGROUP
```

```
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (Integer,%) -> %                ?*? : (PositiveInteger,%) -> %
--R ?+? : (%,%) -> %                      -? : % -> %
--R ?-? : (%,%) -> %                      ?<? : (%,%) -> Boolean
--R ?<=? : (%,%) -> Boolean               ?=? : (%,%) -> Boolean
--R ?>? : (%,%) -> Boolean                ?>=? : (%,%) -> Boolean
--R 0 : () -> %                           coerce : % -> OutputForm
--R hash : % -> SingleInteger             latex : % -> String
--R max : (%,%) -> %                      min : (%,%) -> %
--R sample : () -> %                      zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— OrderedAbelianGroup.help —

```
====================================================================
OrderedAbelianGroup examples
====================================================================

See Also:
o )show OrderedAbelianGroup
```

———-

See:

⇒ “OrderedRing” (ORDRING) 10.0.144 on page 879
⇐ “AbelianGroup” (ABELGRP) 7.0.97 on page 520
⇐ “OrderedCancellationAbelianMonoid” (OCAMON) 8.0.118 on page 696

Exports:

0	coerce	hash	latex	max
min	sample	subtractIfCan	zero?	?~=?
?*?	?+?	-?	?-?	?<?
?<=?	?=?	?>?	?>=?	

These exports come from (p696) OrderedCancellationAbelianMonoid():

```
0 : () -> %
```

```
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
max : (%,%) -> %
min : (%,%) -> %
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?<? : (%,%) -> Boolean
?<=? : (%,%) -> Boolean
?=? : (%,%) -> Boolean
?>? : (%,%) -> Boolean
?>=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?+? : (%,%) -> %
```

These exports come from (p520) AbelianGroup():

```
-? : % -> %
?*? : (Integer,%) -> %
?-? : (%,%) -> %
```

— category OAGROUP OrderedAbelianGroup —

```
)abbrev category OAGROUP OrderedAbelianGroup
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ Ordered sets which are also abelian groups, such that the
++ addition preserves the ordering.

OrderedAbelianGroup(): Category ==
        Join(OrderedCancellationAbelianMonoid, AbelianGroup)
```

———-

— OAGROUP.dotabb —

```
"OAGROUP"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OAGROUP"];
"OAGROUP" -> "OCAMON"
"OAGROUP" -> "ABELGRP"
```

———-

— OAGROUP.dotfull —

```
"OrderedAbelianGroup()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OAGROUP"];
"OrderedAbelianGroup()" -> "OrderedCancellationAbelianMonoid()"
"OrderedAbelianGroup()" -> "AbelianGroup()"
```

———-

— OAGROUP.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"OrderedAbelianGroup()" [color=lightblue];
"OrderedAbelianGroup()" -> "OrderedCancellationAbelianMonoid()"
"OrderedAbelianGroup()" -> "AbelianGroup()"

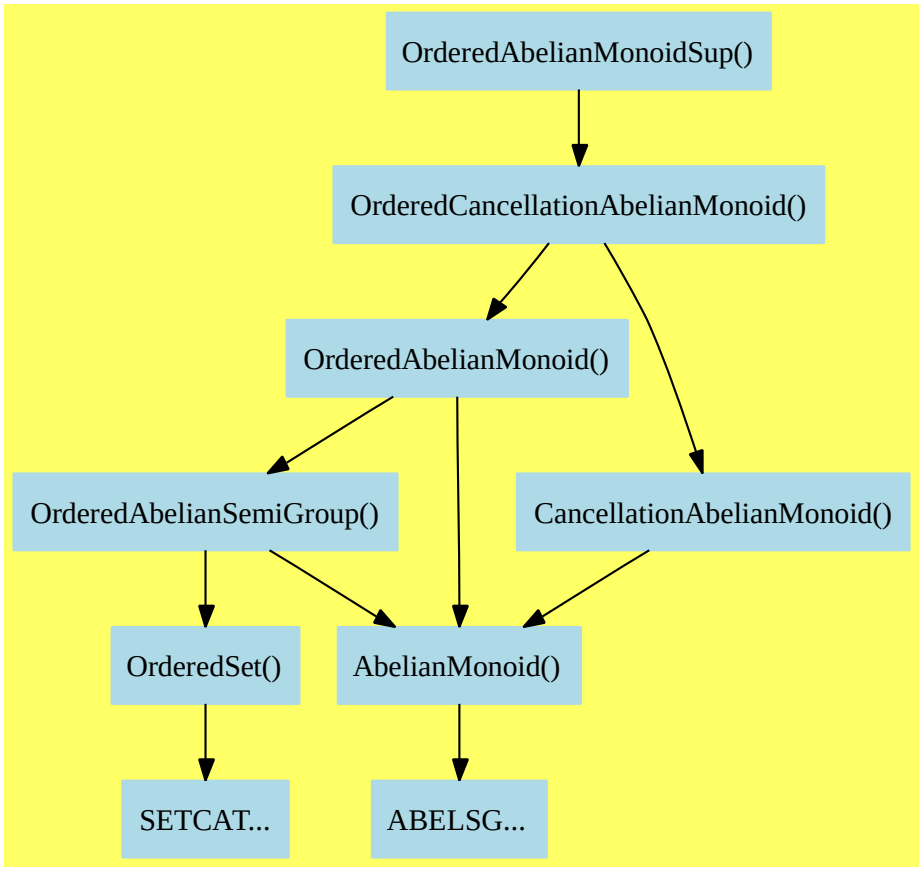
"OrderedCancellationAbelianMonoid()" [color=lightblue];
"OrderedCancellationAbelianMonoid()" -> "OAMON..."
"OrderedCancellationAbelianMonoid()" -> "CABMON..."

"AbelianGroup()" [color=lightblue];
"AbelianGroup()" -> "CABMON..."
"AbelianGroup()" -> "REPDB..."

"REPDB..." [color="#00EE00"];
"OAMON..." [color=lightblue];
"CABMON..." [color=lightblue];
}
```

———-

9.0.127 OrderedAbelianMonoidSup (OAMONS)



— OrderedAbelianMonoidSup.input —

```
)set break resume
)sys rm -f OrderedAbelianMonoidSup.output
)spool OrderedAbelianMonoidSup.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show OrderedAbelianMonoidSup
--R OrderedAbelianMonoidSup  is a category constructor
--R Abbreviation for OrderedAbelianMonoidSup is OAMONS
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for OAMONS
```

```
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (PositiveInteger,%) -> %        ?+? : (%,%) -> %
--R ?<? : (%,%) -> Boolean               ?<=? : (%,%) -> Boolean
--R ?=? : (%,%) -> Boolean               ?>? : (%,%) -> Boolean
--R ?>=? : (%,%) -> Boolean              0 : () -> %
--R coerce : % -> OutputForm             hash : % -> SingleInteger
--R latex : % -> String                  max : (%,%) -> %
--R min : (%,%) -> %                     sample : () -> %
--R sup : (%,%) -> %                     zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— OrderedAbelianMonoidSup.help —

```
=====================================================================
OrderedAbelianMonoidSup examples
=====================================================================

See Also:
o )show OrderedAbelianMonoidSup
```

———-

See:

⇐ “DirectProductCategory” (DIRPCAT) 12.0.160 on page 1001
⇐ “OrderedCancellationAbelianMonoid” (OCAMON) 8.0.118 on page 696

Exports:

0	coerce	hash	latex	max
min	sample	subtractIfCan	sup	zero?
?~=?	?*?	?<=?	?+?	?<?
?=?	?>?	?>=?		

These are directly exported but not implemented:

```
sup : (%,%) -> %
```

These exports come from (p696) OrderedCancellationAbelianMonoid():

```
 0 : () -> %
 coerce : % -> OutputForm
 hash : % -> SingleInteger
 latex : % -> String
 max : (%,%) -> %
 min : (%,%) -> %
 sample : () -> %
 subtractIfCan : (%,%) -> Union(%,"failed")
 zero? : % -> Boolean
 ?~=? : (%,%) -> Boolean
 ?*? : (NonNegativeInteger,%) -> %
 ?*? : (PositiveInteger,%) -> %
 ?+? : (%,%) -> %
 ?<? : (%,%) -> Boolean
 ?<=? : (%,%) -> Boolean
 ?=? : (%,%) -> Boolean
 ?>? : (%,%) -> Boolean
 ?>=? : (%,%) -> Boolean
```

— category OAMONS OrderedAbelianMonoidSup —

```
)abbrev category OAMONS OrderedAbelianMonoidSup
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ This domain is an OrderedAbelianMonoid with a sup
++ operation added. The purpose of the sup operator
++ in this domain is to act as a supremum with respect to the
++ partial order imposed by `-`, rather than with respect to
++ the total $>$ order (since that is "max").
++
++ Axioms\br
++ \tab{5}sup(a,b)-a \~~= "failed"\br
++ \tab{5}sup(a,b)-b \~~= "failed"\br
++ \tab{5}x-a \~~= "failed" and x-b \~~= "failed" => x >= sup(a,b)\br

OrderedAbelianMonoidSup(): Category == OrderedCancellationAbelianMonoid with
  sup: (%,%) -> %
    ++ sup(x,y) returns the least element from which both
    ++ x and y can be subtracted.
```

———-

— **OAMONS.dotabb** —

```
"OAMONS"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OAMONS"];
"OAMONS" -> "OCAMON"
```

———-

— **OAMONS.dotfull** —

```
"OrderedAbelianMonoidSup()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OAMONS"];
"OrderedAbelianMonoidSup()" -> "OrderedCancellationAbelianMonoid()"
```

———-

— **OAMONS.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"OrderedAbelianMonoidSup()" [color=lightblue];
"OrderedAbelianMonoidSup()" -> "OrderedCancellationAbelianMonoid()"

"OrderedCancellationAbelianMonoid()" [color=lightblue];
"OrderedCancellationAbelianMonoid()" -> "OrderedAbelianMonoid()"
"OrderedCancellationAbelianMonoid()" -> "CancellationAbelianMonoid()"

"OrderedAbelianMonoid()" [color=lightblue];
"OrderedAbelianMonoid()" -> "OrderedAbelianSemiGroup()"
"OrderedAbelianMonoid()" -> "AbelianMonoid()"

"OrderedAbelianSemiGroup()" [color=lightblue];
"OrderedAbelianSemiGroup()" -> "OrderedSet()"
"OrderedAbelianSemiGroup()" -> "AbelianMonoid()"

"OrderedSet()" [color=lightblue];
"OrderedSet()" -> "SETCAT..."

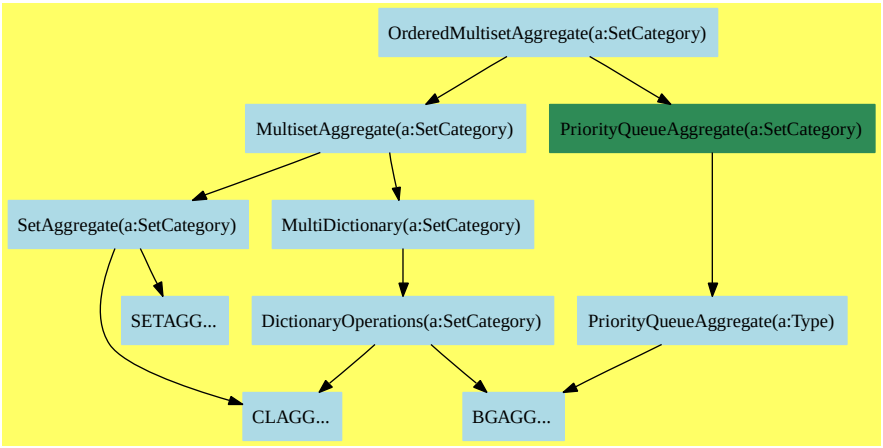
"CancellationAbelianMonoid()" [color=lightblue];
"CancellationAbelianMonoid()" -> "AbelianMonoid()"
```

```
"AbelianMonoid()" [color=lightblue];
"AbelianMonoid()" -> "ABELSG..."

"SETCAT..." [color=lightblue];
"ABELSG..." [color=lightblue];
}
```

———-

9.0.128 OrderedMultisetAggregate (OMSAGG)



— OrderedMultisetAggregate.input —

```
)set break resume
)sys rm -f OrderedMultisetAggregate.output
)spool OrderedMultisetAggregate.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show OrderedMultisetAggregate
--R OrderedMultisetAggregate S: OrderedSet  is a category constructor
--R Abbreviation for OrderedMultisetAggregate is OMSAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for OMSAGG
--R
--R------------------------------ Operations --------------------------------
```

```
--R ?<? : (%,%) -> Boolean                ?=? : (%,%) -> Boolean
--R bag : List S -> %                     brace : () -> %
--R brace : List S -> %                   coerce : % -> OutputForm
--R construct : List S -> %               copy : % -> %
--R dictionary : List S -> %              dictionary : () -> %
--R difference : (%,%) -> %               difference : (%,S) -> %
--R empty : () -> %                       empty? : % -> Boolean
--R eq? : (%,%) -> Boolean                extract! : % -> S
--R hash : % -> SingleInteger             insert! : (S,%) -> %
--R inspect : % -> S                      intersect : (%,%) -> %
--R latex : % -> String                   map : ((S -> S),%) -> %
--R max : % -> S                          merge : (%,%) -> %
--R merge! : (%,%) -> %                   min : % -> S
--R removeDuplicates! : % -> %            sample : () -> %
--R set : () -> %                         set : List S -> %
--R subset? : (%,%) -> Boolean            union : (%,%) -> %
--R union : (%,S) -> %                    union : (S,%) -> %
--R ?~=? : (%,%) -> Boolean
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R convert : % -> InputForm if S has KONVERT INFORM
--R count : (S,%) -> NonNegativeInteger if S has SETCAT and $ has finiteAggregate
--R count : ((S -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R duplicates : % -> List Record(entry: S,count: NonNegativeInteger)
--R eval : (%,List S,List S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,S,S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,Equation S) -> % if S has EVALAB S and S has SETCAT
--R eval : (%,List Equation S) -> % if S has EVALAB S and S has SETCAT
--R every? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
--R find : ((S -> Boolean),%) -> Union(S,"failed")
--R insert! : (S,%,NonNegativeInteger) -> %
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((S -> S),%) -> % if $ has shallowlyMutable
--R member? : (S,%) -> Boolean if S has SETCAT and $ has finiteAggregate
--R members : % -> List S if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S) -> S if $ has finiteAggregate
--R reduce : (((S,S) -> S),%,S,S) -> S if S has SETCAT and $ has finiteAggregate
--R remove : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (S,%) -> % if S has SETCAT and $ has finiteAggregate
--R remove! : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R remove! : (S,%) -> % if $ has finiteAggregate
--R removeDuplicates : % -> % if S has SETCAT and $ has finiteAggregate
--R select : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R select! : ((S -> Boolean),%) -> % if $ has finiteAggregate
--R size? : (%,NonNegativeInteger) -> Boolean
--R symmetricDifference : (%,%) -> %
--R
```

```
--E 1

)spool
)lisp (bye)
```

———-

— **OrderedMultisetAggregate.help** —

```
====================================================================
OrderedMultisetAggregate examples
====================================================================

See Also:
o )show OrderedMultisetAggregate
```

———-

See:

⇐ "MultisetAggregate" (MSETAGG) 8.0.115 on page 673
⇐ "PriorityQueueAggregate" (PRQAGG) 6.0.92 on page 480

Exports:

any?	bag	brace	coerce
construct	convert	copy	count
dictionary	difference	empty	empty?
eq?	duplicates	eval	every?
extract!	find	hash	insert!
inspect	intersect	latex	less?
map	map!	max	member?
members	merge	merge!	min
more?	parts	reduce	remove
remove!	removeDuplicates	removeDuplicates!	sample
select	select!	set	size?
subset?	symmetricDifference	union	# ?
?<?	?=?	?~?	

Attributes exported:

- **partiallyOrderedSet** is true if a set with $<$ which is transitive, but not($a < b$ or $a = b$) does not necessarily imply $b < a$.
- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **nil**

These are directly exported but not implemented:

```
min : % -> S
```

These exports come from (p673) MultisetAggregate(S:OrderedSet):

```
any? : ((S -> Boolean),%) -> Boolean if $ has finiteAggregate
bag : List S -> %
brace : () -> %
brace : List S -> %
coerce : % -> OutputForm
construct : List S -> %
convert : % -> InputForm if S has KONVERT INFORM
copy : % -> %
count : (S,%) -> NonNegativeInteger
    if S has SETCAT and $ has finiteAggregate
count : ((S -> Boolean),%) -> NonNegativeInteger
    if $ has finiteAggregate
dictionary : List S -> %
dictionary : () -> %
difference : (%,S) -> %
difference : (%,%) -> %
duplicates : % ->
    List Record(entry: S,count: NonNegativeInteger)
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List S,List S) -> %
    if S has EVALAB S and S has SETCAT
eval : (%,S,S) -> %
    if S has EVALAB S and S has SETCAT
eval : (%,Equation S) -> %
    if S has EVALAB S and S has SETCAT
eval : (%,List Equation S) -> %
    if S has EVALAB S and S has SETCAT
every? : ((S -> Boolean),%) -> Boolean
    if $ has finiteAggregate
extract! : % -> S
find : ((S -> Boolean),%) -> Union(S,"failed")
hash : % -> SingleInteger
insert! : (S,%) -> %
insert! : (S,%,NonNegativeInteger) -> %
inspect : % -> S
intersect : (%,%) -> %
latex : % -> String
less? : (%,NonNegativeInteger) -> Boolean
```

```
map : ((S -> S),%) -> %
map! : ((S -> S),%) -> % if $ has shallowlyMutable
member? : (S,%) -> Boolean
    if S has SETCAT and $ has finiteAggregate
members : % -> List S if $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
parts : % -> List S if $ has finiteAggregate
reduce : (((S,S) -> S),%) -> S
    if $ has finiteAggregate
reduce : (((S,S) -> S),%,S) -> S
    if $ has finiteAggregate
reduce : (((S,S) -> S),%,S,S) -> S
    if S has SETCAT and $ has finiteAggregate
remove : ((S -> Boolean),%) -> %
    if $ has finiteAggregate
remove : (S,%) -> %
    if S has SETCAT and $ has finiteAggregate
remove! : ((S -> Boolean),%) -> %
    if $ has finiteAggregate
remove! : (S,%) -> %
    if $ has finiteAggregate
removeDuplicates : % -> %
    if S has SETCAT and $ has finiteAggregate
removeDuplicates! : % -> %
sample : () -> %
select : ((S -> Boolean),%) -> % if $ has finiteAggregate
select! : ((S -> Boolean),%) -> % if $ has finiteAggregate
set : () -> %
set : List S -> %
size? : (%,NonNegativeInteger) -> Boolean
subset? : (%,%) -> Boolean
symmetricDifference : (%,%) -> %
union : (%,%) -> %
union : (%,S) -> %
union : (S,%) -> %
#? : % -> NonNegativeInteger if $ has finiteAggregate
?~=? : (%,%) -> Boolean
?=? : (%,%) -> Boolean
?<? : (%,%) -> Boolean
```

These exports come from (p480) PriorityQueueAggregate(S:OrderedSet):

```
max : % -> S
merge : (%,%) -> %
merge! : (%,%) -> %
```

— category OMSAGG OrderedMultisetAggregate —

```
)abbrev category OMSAGG OrderedMultisetAggregate
```

```
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ An ordered-multiset aggregate is a multiset built over an ordered set S
++ so that the relative sizes of its entries can be assessed.
++ These aggregates serve as models for priority queues.

OrderedMultisetAggregate(S:OrderedSet): Category ==
   Join(MultisetAggregate S,PriorityQueueAggregate S) with
   -- max: % -> S                       ++ smallest entry in the set
   -- duplicates: % -> List Record(entry:S,count:NonNegativeInteger)
        ++ to become an in order iterator
   -- parts: % -> List S              ++ in order iterator
      min: % -> S
        ++ min(u) returns the smallest entry in the multiset aggregate u.
```

———-

— OMSAGG.dotabb —

```
"OMSAGG"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OMSAGG"];
"OMSAGG" -> "MSETAGG"
"OMSAGG" -> "PRQAGG"
```

———-

— OMSAGG.dotfull —

```
"OrderedMultisetAggregate(a:SetCategory)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OMSAGG"];
"OrderedMultisetAggregate(a:SetCategory)" ->
    "MultisetAggregate(a:SetCategory)"
"OrderedMultisetAggregate(a:SetCategory)" ->
   "PriorityQueueAggregate(a:SetCategory)"
```

———-

— OMSAGG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"OrderedMultisetAggregate(a:SetCategory)" [color=lightblue];
"OrderedMultisetAggregate(a:SetCategory)" ->
    "MultisetAggregate(a:SetCategory)"
"OrderedMultisetAggregate(a:SetCategory)" ->
   "PriorityQueueAggregate(a:SetCategory)"

"MultisetAggregate(a:SetCategory)" [color=lightblue];
"MultisetAggregate(a:SetCategory)" -> "MultiDictionary(a:SetCategory)"
"MultisetAggregate(a:SetCategory)" -> "SetAggregate(a:SetCategory)"

"MultiDictionary(a:SetCategory)" [color=lightblue];
"MultiDictionary(a:SetCategory)" -> "DictionaryOperations(a:SetCategory)"

"SetAggregate(a:SetCategory)" [color=lightblue];
"SetAggregate(a:SetCategory)" -> "SETAGG..."
"SetAggregate(a:SetCategory)" -> "CLAGG..."

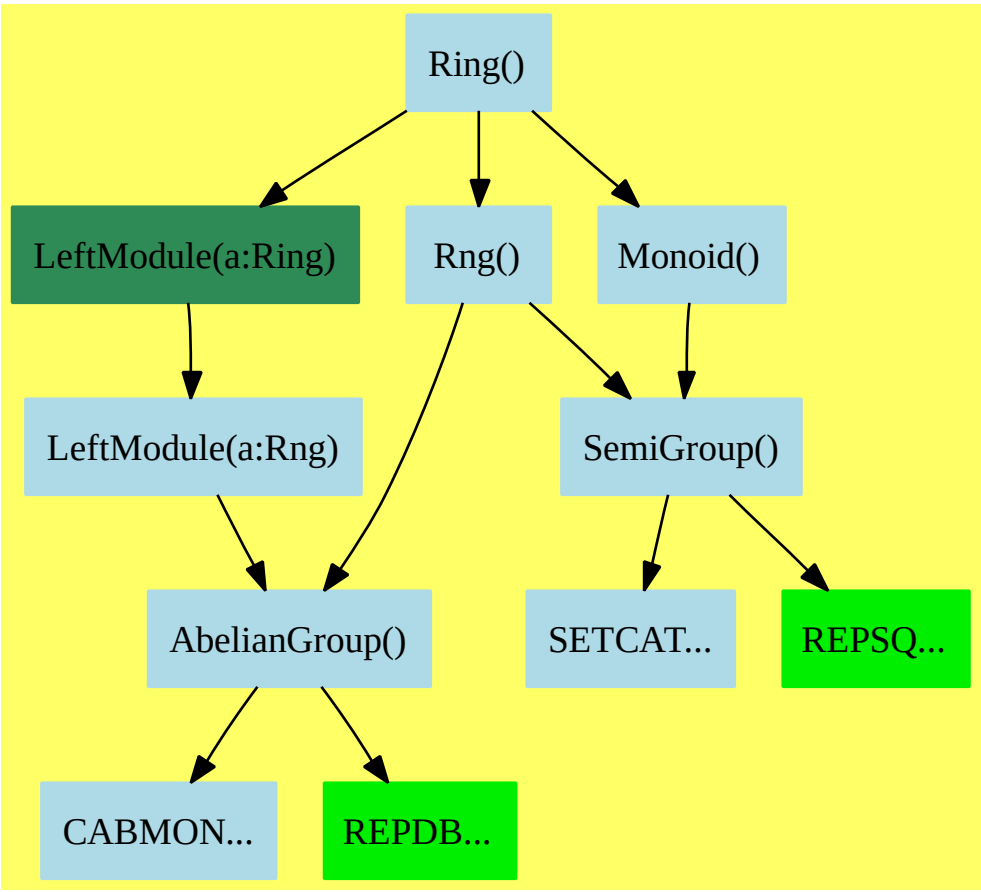
"DictionaryOperations(a:SetCategory)" [color=lightblue];
"DictionaryOperations(a:SetCategory)" -> "BGAGG..."
"DictionaryOperations(a:SetCategory)" -> "CLAGG..."

"PriorityQueueAggregate(a:SetCategory)" [color=seagreen];
"PriorityQueueAggregate(a:SetCategory)" -> "PriorityQueueAggregate(a:Type)"

"PriorityQueueAggregate(a:Type)" [color=lightblue];
"PriorityQueueAggregate(a:Type)" -> "BGAGG..."

"BGAGG..." [color=lightblue];
"CLAGG..." [color=lightblue];
"SETAGG..." [color=lightblue];
}
```

9.0.129 Ring (RING)



— **Ring.input** —

```
)set break resume
)sys rm -f Ring.output
)spool Ring.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show Ring
--R Ring  is a category constructor
--R Abbreviation for Ring is RING
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for RING
```

```
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (%,%) -> %                       ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %         ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                       ?-? : (%,%) -> %
--R -? : % -> %                            ?=? : (%,%) -> Boolean
--R 1 : () -> %                            0 : () -> %
--R ?^? : (%,PositiveInteger) -> %         coerce : Integer -> %
--R coerce : % -> OutputForm               hash : % -> SingleInteger
--R latex : % -> String                    one? : % -> Boolean
--R recip : % -> Union(%,"failed")         sample : () -> %
--R zero? : % -> Boolean                   ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **Ring.help** —

```
=====================================================================
Ring examples
=====================================================================

See Also:
o )show Ring
```

———-

See:

⇒ “Algebra” (ALGEBRA) 11.0.152 on page 947
⇒ “AbelianMonoidRing” (AMR) 13.0.170 on page 1101
⇒ “CharacteristicNonZero” (CHARNZ) 10.0.135 on page 833
⇒ “CharacteristicZero” (CHARZ) 10.0.136 on page 838
⇒ “CommutativeRing” (COMRING) 10.0.137 on page 843
⇒ “DifferentialExtension” (DIFEXT) 11.0.153 on page 953
⇒ “DifferentialRing” (DIFRING) 10.0.138 on page 848
⇒ “EntireRing” (ENTIRER) 10.0.139 on page 853
⇒ “FunctionSpace” (FS) 17.0.195 on page 1336

⇒ "LeftAlgebra" (LALG) 10.0.141 on page 864
⇒ "LinearlyExplicitRingOver" (LINEXP) 10.0.142 on page 869
⇒ "MonogenicLinearOperator" (MLO) 12.0.165 on page 1053
⇒ "OrderedRing" (ORDRING) 10.0.144 on page 879
⇒ "PartialDifferentialRing" (PDRING) 10.0.145 on page 885
⇒ "UnivariateSkewPolynomialCategory" (OREPCAT) 10.0.150 on page 929
⇒ "XAlgebra" (XALG) 10.0.151 on page 940
⇒ "XFreeAlgebra" (XFALG) 11.0.159 on page 992
⇐ "LeftModule" (LMODULE) 8.0.113 on page 655
⇐ "Monoid" (MONOID) 5.0.76 on page 331
⇐ "Rng" (RNG) 8.0.121 on page 719

Exports:

1	0	characteristic	coerce	hash
latex	one?	recip	sample	subtractIfCan
zero?	?~=?	?*?	?**?	?^?
?+?	?-?	-?	?=?	

Attributes Exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.

These are directly exported but not implemented:

```
characteristic : () -> NonNegativeInteger
```

These are implemented by this category:

```
coerce : Integer -> %
```

These exports come from (p719) Rng():

```
0 : () -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (Integer,%) -> %
?*? : (%,%) -> %
?-? : (%,%) -> %
```

```
 -? : % -> %
 ?**? : (%,PositiveInteger) -> %
 ?^? : (%,PositiveInteger) -> %
```

These exports come from (p331) Monoid():

```
 1 : () -> %
 one? : % -> Boolean
 recip : % -> Union(%,"failed")
 ?**? : (%,NonNegativeInteger) -> %
 ?^? : (%,NonNegativeInteger) -> %
```

— category RING Ring —

```
)abbrev category RING Ring
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The category of rings with unity, always associative, but
++ not necessarily commutative.

Ring(): Category == Join(Rng,Monoid,LeftModule(%)) with
      characteristic: () -> NonNegativeInteger
        ++ characteristic() returns the characteristic of the ring
        ++ this is the smallest positive integer n such that
        ++ \spad{n*x=0} for all x in the ring, or zero if no such n
        ++ exists.
        --We can not make this a constant, since some domains are mutable
      coerce: Integer -> %
        ++ coerce(i) converts the integer i to a member of the given domain.
--    recip: % -> Union(%,"failed") -- inherited from Monoid
      unitsKnown
        ++ recip truly yields
        ++ reciprocal or "failed" if not a unit.
        ++ Note that \spad{recip(0) = "failed"}.
   add
      n:Integer
      coerce(n) == n * 1$%
```

———-

— **RING.dotabb** —

```
"RING"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RING"];
"RING" -> "RNG"
"RING" -> "MONOID"
"RING" -> "LMODULE"
```

———-

— **RING.dotfull** —

```
"Ring()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RING"];
"Ring()" -> "Rng()"
"Ring()" -> "Monoid()"
"Ring()" -> "LeftModule(a:Ring)"
```

———-

— **RING.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"Ring()" [color=lightblue];
"Ring()" -> "Rng()"
"Ring()" -> "Monoid()"
"Ring()" -> "LeftModule(a:Ring)"

"Rng()" [color=lightblue];
"Rng()" -> "AbelianGroup()"
"Rng()" -> "SemiGroup()"

"Monoid()" [color=lightblue];
"Monoid()" -> "SemiGroup()"

"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "AbelianGroup()"
```

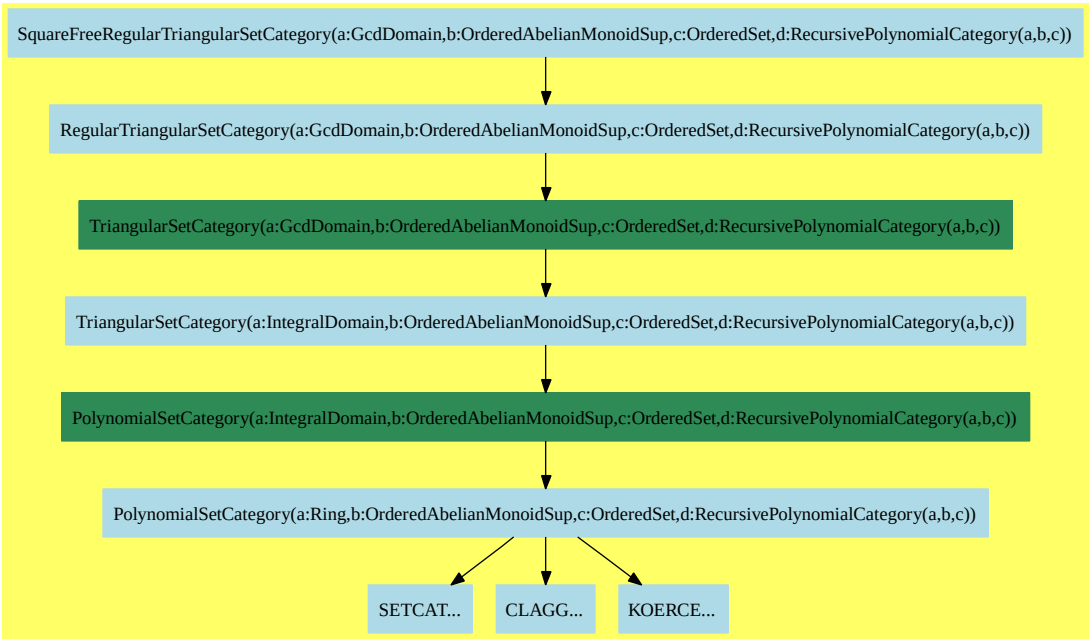
```
"AbelianGroup()" [color=lightblue];
"AbelianGroup()" -> "CABMON..."
"AbelianGroup()" -> "REPDB..."

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SETCAT..."
"SemiGroup()" -> "REPSQ..."

"REPDB..." [color="#00EE00"];
"REPSQ..." [color="#00EE00"];
"SETCAT..." [color=lightblue];
"CABMON..." [color=lightblue];
}
```

———-

9.0.130 SquareFreeRegularTriangularSetCategory (SFRTCAT)



— SquareFreeRegularTriangularSetCategory.input —

```
)set break resume
)sys rm -f SquareFreeRegularTriangularSetCategory.output
```

```
)spool SquareFreeRegularTriangularSetCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show SquareFreeRegularTriangularSetCategory
--R SquareFreeRegularTriangularSetCategory(R: GcdDomain,E: OrderedAbelianMonoidSup,V: OrderedSet
--R Abbreviation for SquareFreeRegularTriangularSetCategory is SFRTCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for SFRTCAT
--R
--R------------------------------ Operations ------------------------------
--R ?=? : (%,%) -> Boolean                algebraic? : (V,%) -> Boolean
--R algebraicVariables : % -> List V      augment : (List P,%) -> List %
--R augment : (P,List %) -> List %        augment : (P,%) -> List %
--R coerce : % -> List P                  coerce : % -> OutputForm
--R collect : (%,V) -> %                  collectQuasiMonic : % -> %
--R collectUnder : (%,V) -> %             collectUpper : (%,V) -> %
--R construct : List P -> %               copy : % -> %
--R degree : % -> NonNegativeInteger      empty : () -> %
--R empty? : % -> Boolean                 eq? : (%,%) -> Boolean
--R extend : (List P,%) -> List %         extend : (P,List %) -> List %
--R extend : (P,%) -> List %              extend : (%,P) -> %
--R first : % -> Union(P,"failed")        hash : % -> SingleInteger
--R headReduce : (P,%) -> P               headReduced? : % -> Boolean
--R headReduced? : (P,%) -> Boolean       infRittWu? : (%,%) -> Boolean
--R initiallyReduce : (P,%) -> P          initiallyReduced? : % -> Boolean
--R initials : % -> List P                internalAugment : (P,%) -> %
--R intersect : (P,List %) -> List %      intersect : (List P,%) -> List %
--R intersect : (P,%) -> List %           invertible? : (P,%) -> Boolean
--R invertibleSet : (P,%) -> List %       last : % -> Union(P,"failed")
--R latex : % -> String                   mainVariable? : (V,%) -> Boolean
--R mainVariables : % -> List V           map : ((P -> P),%) -> %
--R mvar : % -> V                         normalized? : % -> Boolean
--R normalized? : (P,%) -> Boolean        purelyAlgebraic? : % -> Boolean
--R reduceByQuasiMonic : (P,%) -> P       removeZero : (P,%) -> P
--R rest : % -> Union(%,"failed")         retract : List P -> %
--R sample : () -> %                      stronglyReduce : (P,%) -> P
--R stronglyReduced? : % -> Boolean       trivialIdeal? : % -> Boolean
--R variables : % -> List V               zeroSetSplit : List P -> List %
--R ?~=? : (%,%) -> Boolean
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R algebraicCoefficients? : (P,%) -> Boolean
--R any? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
--R augment : (List P,List %) -> List %
--R autoReduced? : (%,((P,List P) -> Boolean)) -> Boolean
--R basicSet : (List P,(P -> Boolean),((P,P) -> Boolean)) -> Union(Record(bas: %,top: List P),"f
--R basicSet : (List P,((P,P) -> Boolean)) -> Union(Record(bas: %,top: List P),"failed")
--R coHeight : % -> NonNegativeInteger if V has FINITE
```

```
--R convert : % -> InputForm if P has KONVERT INFORM
--R count : ((P -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R count : (P,%) -> NonNegativeInteger if P has SETCAT and $ has finiteAggregate
--R eval : (%,List Equation P) -> % if P has EVALAB P and P has SETCAT
--R eval : (%,Equation P) -> % if P has EVALAB P and P has SETCAT
--R eval : (%,P,P) -> % if P has EVALAB P and P has SETCAT
--R eval : (%,List P,List P) -> % if P has EVALAB P and P has SETCAT
--R every? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
--R extend : (List P,List %) -> List %
--R extendIfCan : (%,P) -> Union(%,"failed")
--R find : ((P -> Boolean),%) -> Union(P,"failed")
--R headRemainder : (P,%) -> Record(num: P,den: R) if R has INTDOM
--R initiallyReduced? : (P,%) -> Boolean
--R internalAugment : (List P,%) -> %
--R intersect : (List P,List %) -> List %
--R invertible? : (P,%) -> List Record(val: Boolean,tower: %)
--R invertibleElseSplit? : (P,%) -> Union(Boolean,List %)
--R lastSubResultant : (P,P,%) -> List Record(val: P,tower: %)
--R lastSubResultantElseSplit : (P,P,%) -> Union(P,List %)
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((P -> P),%) -> % if $ has shallowlyMutable
--R member? : (P,%) -> Boolean if P has SETCAT and $ has finiteAggregate
--R members : % -> List P if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List P if $ has finiteAggregate
--R purelyAlgebraic? : (P,%) -> Boolean
--R purelyAlgebraicLeadingMonomial? : (P,%) -> Boolean
--R purelyTranscendental? : (P,%) -> Boolean
--R quasiComponent : % -> Record(close: List P,open: List P)
--R reduce : (P,%,((P,P) -> P),((P,P) -> Boolean)) -> P
--R reduce : (((P,P) -> P),%) -> P if $ has finiteAggregate
--R reduce : (((P,P) -> P),%,P) -> P if $ has finiteAggregate
--R reduce : (((P,P) -> P),%,P,P) -> P if P has SETCAT and $ has finiteAggregate
--R reduced? : (P,%,((P,P) -> Boolean)) -> Boolean
--R remainder : (P,%) -> Record(rnum: R,polnum: P,den: R) if R has INTDOM
--R remove : ((P -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (P,%) -> % if P has SETCAT and $ has finiteAggregate
--R removeDuplicates : % -> % if P has SETCAT and $ has finiteAggregate
--R retractIfCan : List P -> Union(%,"failed")
--R rewriteIdealWithHeadRemainder : (List P,%) -> List P if R has INTDOM
--R rewriteIdealWithRemainder : (List P,%) -> List P if R has INTDOM
--R rewriteSetWithReduction : (List P,%,((P,P) -> P),((P,P) -> Boolean)) -> List P
--R roughBase? : % -> Boolean if R has INTDOM
--R roughEqualIdeals? : (%,%) -> Boolean if R has INTDOM
--R roughSubIdeal? : (%,%) -> Boolean if R has INTDOM
--R roughUnitIdeal? : % -> Boolean if R has INTDOM
--R select : (%,V) -> Union(P,"failed")
--R select : ((P -> Boolean),%) -> % if $ has finiteAggregate
--R size? : (%,NonNegativeInteger) -> Boolean
--R sort : (%,V) -> Record(under: %,floor: %,upper: %)
```

```
--R squareFreePart : (P,%) -> List Record(val: P,tower: %)
--R stronglyReduced? : (P,%) -> Boolean
--R triangular? : % -> Boolean if R has INTDOM
--R zeroSetSplit : (List P,Boolean) -> List %
--R zeroSetSplitIntoTriangularSystems : List P -> List Record(close: %,open: List P)
--R
--E 1

)spool
)lisp (bye)
```

———-

— **SquareFreeRegularTriangularSetCategory.help** —

```
=====================================================================
SquareFreeRegularTriangularSetCategory examples
=====================================================================

See Also:
o )show SquareFreeRegularTriangularSetCategory
```

———-

See:

⇒ “SquareFreeNormalizedTriangularSetCategory” (SNTSCAT) 10.0.148 on page 910
⇐ “RegularTriangularSetCategory” (RSETCAT) 8.0.119 on page 700

Exports:

algebraic?
algebraicCoefficients?
algebraicVariables
any?
augment
autoReduced?
basicSet
coerce
coHeight
collect
collectQuasiMonic
collectUnder
collectUpper
construct
copy
convert
count
degree
empty
empty?
eq?
eval
every?
extend
extendIfCan
find
first
hash
headReduce
headReduced?
headRemainder
infRittWu?
initiallyReduce
initiallyReduced?
initials
internalAugment
intersect
invertible?
invertibleElseSplit?
invertibleSet
last
lastSubResultant
lastSubResultantElseSplit
latex
less?
mainVariable?
mainVariables
map
map!
member?
members
more?
mvar
normalized?
parts
purelyAlgebraic?
purelyAlgebraicLeadingMonomial?
purelyTranscendental?
quasiComponent
reduce
reduceByQuasiMonic
reduced?
remainder
remove
removeDuplicates
removeZero
rest
retract
retractIfCan
rewriteIdealWithHeadRemainder
rewriteIdealWithRemainder
rewriteSetWithReduction
roughBase?
roughEqualIdeals?
roughSubIdeal?
roughUnitIdeal?
sample
select
size?
sort
squareFreePart
stronglyReduce
stronglyReduced?
triangular?
trivialIdeal?
variables
zeroSetSplit
zeroSetSplitIntoTriangularSystems
#?
?=?
?~=?

Attributes Exported:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **nil**

These exports come from (p700) RegularTriangularSetCategory(R,E,V,P) where R:GcdDomain, E:OrderedAbelianMonoidSup, V:OrderedSet, P:RecursivePolynomialCategory(R,E,V)):

```
algebraic? : (V,%) -> Boolean
algebraicCoefficients? : (P,%) -> Boolean
algebraicVariables : % -> List V
any? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
augment : (List P,List %) -> List %
augment : (List P,%) -> List %
augment : (P,List %) -> List %
augment : (P,%) -> List %
autoReduced? : (%,((P,List P) -> Boolean)) -> Boolean
basicSet :
 (List P,(P -> Boolean),((P,P) -> Boolean)) ->
   Union(Record(bas: %,top: List P),"failed")
basicSet :
 (List P,((P,P) -> Boolean)) ->
   Union(Record(bas: %,top: List P),"failed")
coerce : % -> List P
coerce : % -> OutputForm
coHeight : % -> NonNegativeInteger if V has FINITE
collect : (%,V) -> %
collectQuasiMonic : % -> %
collectUnder : (%,V) -> %
collectUpper : (%,V) -> %
construct : List P -> %
convert : % -> InputForm if P has KONVERT INFORM
copy : % -> %
count : ((P -> Boolean),%) -> NonNegativeInteger
  if $ has finiteAggregate
count : (P,%) -> NonNegativeInteger
  if P has SETCAT
  and $ has finiteAggregate
degree : % -> NonNegativeInteger
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List Equation P) -> % if P has EVALAB P and P has SETCAT
eval : (%,Equation P) -> % if P has EVALAB P and P has SETCAT
```

```
eval : (%,P,P) -> % if P has EVALAB P and P has SETCAT
eval : (%,List P,List P) -> % if P has EVALAB P and P has SETCAT
every? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
extend : (List P,List %) -> List %
extend : (List P,%) -> List %
extend : (P,List %) -> List %
extend : (P,%) -> List %
extend : (%,P) -> %
extendIfCan : (%,P) -> Union(%,"failed")
find : ((P -> Boolean),%) -> Union(P,"failed")
first : % -> Union(P,"failed")
hash : % -> SingleInteger
headReduce : (P,%) -> P
headReduced? : % -> Boolean
headReduced? : (P,%) -> Boolean
headRemainder : (P,%) -> Record(num: P,den: R) if R has INTDOM
infRittWu? : (%,%) -> Boolean
initiallyReduce : (P,%) -> P
initiallyReduced? : % -> Boolean
initiallyReduced? : (P,%) -> Boolean
initials : % -> List P
internalAugment : (P,%) -> %
internalAugment : (List P,%) -> %
intersect : (P,List %) -> List %
intersect : (List P,%) -> List %
intersect : (P,%) -> List %
intersect : (List P,List %) -> List %
invertible? : (P,%) -> Boolean
invertible? : (P,%) -> List Record(val: Boolean,tower: %)
invertibleElseSplit? : (P,%) -> Union(Boolean,List %)
invertibleSet : (P,%) -> List %
last : % -> Union(P,"failed")
lastSubResultant : (P,P,%) -> List Record(val: P,tower: %)
lastSubResultantElseSplit : (P,P,%) -> Union(P,List %)
latex : % -> String
less? : (%,NonNegativeInteger) -> Boolean
mainVariable? : (V,%) -> Boolean
mainVariables : % -> List V
map : ((P -> P),%) -> %
map! : ((P -> P),%) -> % if $ has shallowlyMutable
member? : (P,%) -> Boolean if P has SETCAT and $ has finiteAggregate
members : % -> List P if $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
mvar : % -> V
normalized? : % -> Boolean
normalized? : (P,%) -> Boolean
parts : % -> List P if $ has finiteAggregate
purelyAlgebraic? : (P,%) -> Boolean
purelyAlgebraic? : % -> Boolean
purelyAlgebraicLeadingMonomial? : (P,%) -> Boolean
```

```
purelyTranscendental? : (P,%) -> Boolean
quasiComponent : % -> Record(close: List P,open: List P)
reduce : (P,%,((P,P) -> P),((P,P) -> Boolean)) -> P
reduce : (((P,P) -> P),%) -> P if $ has finiteAggregate
reduce : (((P,P) -> P),%,P) -> P if $ has finiteAggregate
reduce : (((P,P) -> P),%,P,P) -> P
  if P has SETCAT
  and $ has finiteAggregate
reduceByQuasiMonic : (P,%) -> P
reduced? : (P,%,((P,P) -> Boolean)) -> Boolean
remainder : (P,%) -> Record(rnum: R,polnum: P,den: R)
  if R has INTDOM
remove : ((P -> Boolean),%) -> % if $ has finiteAggregate
remove : (P,%) -> % if P has SETCAT and $ has finiteAggregate
removeDuplicates : % -> % if P has SETCAT and $ has finiteAggregate
removeZero : (P,%) -> P
rest : % -> Union(%,"failed")
retract : List P -> %
retractIfCan : List P -> Union(%,"failed")
rewriteIdealWithHeadRemainder : (List P,%) -> List P if R has INTDOM
rewriteIdealWithRemainder : (List P,%) -> List P if R has INTDOM
rewriteSetWithReduction :
   (List P,%,((P,P) -> P),((P,P) -> Boolean)) -> List P
roughBase? : % -> Boolean if R has INTDOM
roughEqualIdeals? : (%,%) -> Boolean if R has INTDOM
roughSubIdeal? : (%,%) -> Boolean if R has INTDOM
roughUnitIdeal? : % -> Boolean if R has INTDOM
sample : () -> %
select : (%,V) -> Union(P,"failed")
select : ((P -> Boolean),%) -> % if $ has finiteAggregate
size? : (%,NonNegativeInteger) -> Boolean
sort : (%,V) -> Record(under: %,floor: %,upper: %)
squareFreePart : (P,%) -> List Record(val: P,tower: %)
stronglyReduce : (P,%) -> P
stronglyReduced? : (P,%) -> Boolean
stronglyReduced? : % -> Boolean
triangular? : % -> Boolean if R has INTDOM
trivialIdeal? : % -> Boolean
variables : % -> List V
zeroSetSplit : List P -> List %
zeroSetSplit : (List P,Boolean) -> List %
zeroSetSplitIntoTriangularSystems :
  List P -> List Record(close: %,open: List P)
#? : % -> NonNegativeInteger if $ has finiteAggregate
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
```

— category SFRTCAT SquareFreeRegularTriangularSetCategory —

```
)abbrev category SFRTCAT SquareFreeRegularTriangularSetCategory
++ Author: Marc Moreno Maza
++ Date Created: 09/03/1996
++ Date Last Updated: 09/10/1998
++ Basic Functions:
++ Related Constructors:
++ Also See: essai Graphisme
++ AMS Classifications:
++ Keywords: polynomial, multivariate, ordered variables set
++ References :
++  [1] D. LAZARD "A new method for solving algebraic systems of
++      positive dimension" Discr. App. Math. 33:147-160,1991
++  [2] M. KALKBRENER "Algorithmic properties of polynomial rings"
++      Habilitation Thesis, ETZH, Zurich, 1995.
++  [3] M. MORENO MAZA "A new algorithm for computing triangular
++      decomposition of algebraic varieties" NAG Tech. Rep. 4/98.
++ Description:
++ The category of square-free regular triangular sets.
++ A regular triangular set \spad{ts} is square-free if
++ the gcd of any polynomial \spad{p} in \spad{ts} and
++ differentiate(p,mvar(p)) w.r.t. collectUnder(ts,mvar(p))
++ has degree zero w.r.t. \spad{mvar(p)}. Thus any square-free regular
++ set defines a tower of square-free simple extensions.

SquareFreeRegularTriangularSetCategory(R:GcdDomain,E:OrderedAbelianMonoidSup,_
  V:OrderedSet,P:RecursivePolynomialCategory(R,E,V)):
         Category ==
   RegularTriangularSetCategory(R,E,V,P)
```

———-

— SFRTCAT.dotabb —

```
"SFRTCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SFRTCAT"];
"SFRTCAT" -> "RSETCAT"
```

———-

— SFRTCAT.dotfull —

```
"SquareFreeRegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SFRTCAT"];
"SquareFreeRegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
 ->
"RegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCateg
```

———-

— SFRTCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"SquareFreeRegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:Rec
 [color=lightblue];
"SquareFreeRegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:Rec
 ->
"RegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePoly

"RegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePoly
 [color=lightblue];
"RegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePoly
  ->
"TriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialC

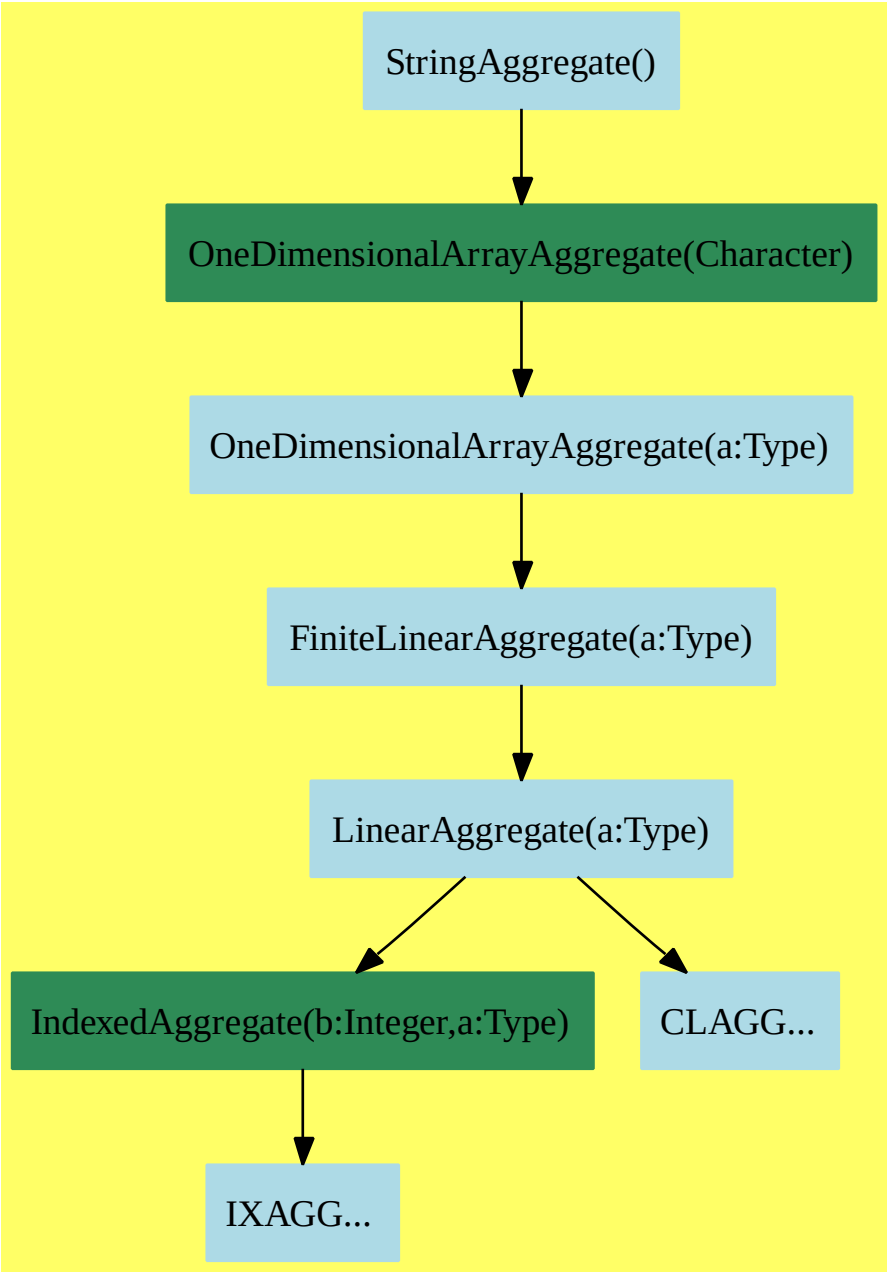
"TriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialC
 [color=seagreen];
"TriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialC
 ->
"TriangularSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
"TriangularSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
 [color=lightblue];
"TriangularSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
  -> "PolynomialSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursiveP

"PolynomialSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
 [color=seagreen];
"PolynomialSetCategory(a:IntegralDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno
  -> "PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialC

"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCatego
 [color=lightblue];
"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCatego
 -> "SETCAT..."
"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCatego
 -> "CLAGG..."
"PolynomialSetCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolynomialCatego
 -> "KOERCE..."

"SETCAT..." [color=lightblue];
"KOERCE..." [color=lightblue];
"CLAGG..." [color=lightblue];
}
```

9.0.131 StringAggregate (SRAGG)



— **StringAggregate.input** —

```
)set break resume
)sys rm -f StringAggregate.output
)spool StringAggregate.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show StringAggregate
--R StringAggregate  is a category constructor
--R Abbreviation for StringAggregate is SRAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for SRAGG
--R
--R------------------------------ Operations --------------------------------
--R coerce : Character -> %               concat : List % -> %
--R concat : (%,%) -> %                   concat : (Character,%) -> %
--R concat : (%,Character) -> %           construct : List Character -> %
--R copy : % -> %                         delete : (%,Integer) -> %
--R ?.? : (%,%) -> %                      ?.? : (%,Integer) -> Character
--R empty : () -> %                       empty? : % -> Boolean
--R entries : % -> List Character         eq? : (%,%) -> Boolean
--R index? : (Integer,%) -> Boolean       indices : % -> List Integer
--R insert : (%,%,Integer) -> %           leftTrim : (%,Character) -> %
--R lowerCase : % -> %                    lowerCase! : % -> %
--R prefix? : (%,%) -> Boolean            qelt : (%,Integer) -> Character
--R reverse : % -> %                      rightTrim : (%,Character) -> %
--R sample : () -> %                      split : (%,Character) -> List %
--R suffix? : (%,%) -> Boolean            trim : (%,CharacterClass) -> %
--R trim : (%,Character) -> %             upperCase : % -> %
--R upperCase! : % -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?<? : (%,%) -> Boolean if Character has ORDSET
--R ?<=? : (%,%) -> Boolean if Character has ORDSET
--R ?=? : (%,%) -> Boolean if Character has SETCAT
--R ?>? : (%,%) -> Boolean if Character has ORDSET
--R ?>=? : (%,%) -> Boolean if Character has ORDSET
--R any? : ((Character -> Boolean),%) -> Boolean if $ has finiteAggregate
--R coerce : % -> OutputForm if Character has SETCAT
--R convert : % -> InputForm if Character has KONVERT INFORM
--R copyInto! : (%,%,Integer) -> % if $ has shallowlyMutable
--R count : (Character,%) -> NonNegativeInteger if Character has SETCAT and $ has finiteAggregate
--R count : ((Character -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R delete : (%,UniversalSegment Integer) -> %
--R ?.? : (%,UniversalSegment Integer) -> %
--R elt : (%,Integer,Character) -> Character
--R entry? : (Character,%) -> Boolean if $ has finiteAggregate and Character has SETCAT
--R eval : (%,List Character,List Character) -> % if Character has EVALAB CHAR and Character has SETCAT
--R eval : (%,Character,Character) -> % if Character has EVALAB CHAR and Character has SETCAT
--R eval : (%,Equation Character) -> % if Character has EVALAB CHAR and Character has SETCAT
```

```
--R eval : (%,List Equation Character) -> % if Character has EVALAB CHAR and Character has SETCA
--R every? : ((Character -> Boolean),%) -> Boolean if $ has finiteAggregate
--R fill! : (%,Character) -> % if $ has shallowlyMutable
--R find : ((Character -> Boolean),%) -> Union(Character,"failed")
--R first : % -> Character if Integer has ORDSET
--R hash : % -> SingleInteger if Character has SETCAT
--R insert : (Character,%,Integer) -> %
--R latex : % -> String if Character has SETCAT
--R leftTrim : (%,CharacterClass) -> %
--R less? : (%,NonNegativeInteger) -> Boolean
--R map : (((Character,Character) -> Character),%,%) -> %
--R map : ((Character -> Character),%) -> %
--R map! : ((Character -> Character),%) -> % if $ has shallowlyMutable
--R match : (%,%,Character) -> NonNegativeInteger
--R match? : (%,%,Character) -> Boolean
--R max : (%,%) -> % if Character has ORDSET
--R maxIndex : % -> Integer if Integer has ORDSET
--R member? : (Character,%) -> Boolean if Character has SETCAT and $ has finiteAggregate
--R members : % -> List Character if $ has finiteAggregate
--R merge : (%,%) -> % if Character has ORDSET
--R merge : (((Character,Character) -> Boolean),%,%) -> %
--R min : (%,%) -> % if Character has ORDSET
--R minIndex : % -> Integer if Integer has ORDSET
--R more? : (%,NonNegativeInteger) -> Boolean
--R new : (NonNegativeInteger,Character) -> %
--R parts : % -> List Character if $ has finiteAggregate
--R position : (CharacterClass,%,Integer) -> Integer
--R position : (%,%,Integer) -> Integer
--R position : (Character,%,Integer) -> Integer if Character has SETCAT
--R position : (Character,%) -> Integer if Character has SETCAT
--R position : ((Character -> Boolean),%) -> Integer
--R qsetelt! : (%,Integer,Character) -> Character if $ has shallowlyMutable
--R reduce : (((Character,Character) -> Character),%) -> Character if $ has finiteAggregate
--R reduce : (((Character,Character) -> Character),%,Character) -> Character if $ has finiteAggr
--R reduce : (((Character,Character) -> Character),%,Character,Character) -> Character if Charac
--R remove : ((Character -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (Character,%) -> % if Character has SETCAT and $ has finiteAggregate
--R removeDuplicates : % -> % if Character has SETCAT and $ has finiteAggregate
--R replace : (%,UniversalSegment Integer,%) -> %
--R reverse! : % -> % if $ has shallowlyMutable
--R rightTrim : (%,CharacterClass) -> %
--R select : ((Character -> Boolean),%) -> % if $ has finiteAggregate
--R setelt : (%,UniversalSegment Integer,Character) -> Character if $ has shallowlyMutable
--R setelt : (%,Integer,Character) -> Character if $ has shallowlyMutable
--R size? : (%,NonNegativeInteger) -> Boolean
--R sort : % -> % if Character has ORDSET
--R sort : (((Character,Character) -> Boolean),%) -> %
--R sort! : % -> % if Character has ORDSET and $ has shallowlyMutable
--R sort! : (((Character,Character) -> Boolean),%) -> % if $ has shallowlyMutable
--R sorted? : % -> Boolean if Character has ORDSET
```

```
--R sorted? : (((Character,Character) -> Boolean),%) -> Boolean
--R split : (%,CharacterClass) -> List %
--R substring? : (%,%,Integer) -> Boolean
--R swap! : (%,Integer,Integer) -> Void if $ has shallowlyMutable
--R ?~=? : (%,%) -> Boolean if Character has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— **StringAggregate.help** —

```
=====================================================================
StringAggregate examples
=====================================================================

See Also:
o )show StringAggregate
```

———-

See:

⇒ “StringCategory” (STRICAT) 10.0.149 on page 920
⇐ “OneDimensionalArrayAggregate” (A1AGG) 8.0.117 on page 684

Exports:

any?	coerce	concat	construct	copy
convert	copyInto!	count	delete	elt
empty	empty?	entries	entry?	eval
every?	eq?	fill!	find	first
hash	index?	indices	insert	latex
leftTrim	less?	lowerCase	lowerCase!	map
map!	match	match?	max	maxIndex
member?	members	merge	min	minIndex
more?	new	parts	position	prefix?
qelt	qsetelt!	reduce	remove	removeDuplicates
replace	reverse	reverse!	rightTrim	sample
setelt	size?	sort	sort!	sorted?
split	substring?	suffix?	swap!	trim
upperCase	upperCase!	#?	?~=?	?.?
?<?	?<=?	?=?	?>?	?>=?

Attributes exported:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **nil**

These are directly exported but not implemented:

```
leftTrim : (%,Character) -> %
leftTrim : (%,CharacterClass) -> %
lowerCase! : % -> %
match : (%,%,Character) -> NonNegativeInteger
match? : (%,%,Character) -> Boolean
position : (CharacterClass,%,Integer) -> Integer
position : (%,%,Integer) -> Integer
replace : (%,UniversalSegment Integer,%) -> %
rightTrim : (%,Character) -> %
rightTrim : (%,CharacterClass) -> %
split : (%,Character) -> List %
split : (%,CharacterClass) -> List %
substring? : (%,%,Integer) -> Boolean
suffix? : (%,%) -> Boolean
upperCase! : % -> %
```

These are implemented by this category:

```
coerce : Character -> %
lowerCase : % -> %
prefix? : (%,%) -> Boolean
trim : (%,CharacterClass) -> %
trim : (%,Character) -> %
upperCase : % -> %
?.? : (%,%) -> %
```

These exports come from (p684) OneDimensionalArrayAggregate(Character):

```
any? : ((Character -> Boolean),%) -> Boolean
         if $ has finiteAggregate
coerce : % -> OutputForm if Character has SETCAT
concat : List % -> %
concat : (%,%) -> %
concat : (Character,%) -> %
concat : (%,Character) -> %
construct : List Character -> %
convert : % -> InputForm
         if Character has KONVERT INFORM
copy : % -> %
```

```
copyInto! : (%,%,Integer) -> %
         if $ has shallowlyMutable
count : (Character,%) -> NonNegativeInteger
         if Character has SETCAT
         and $ has finiteAggregate
count : ((Character -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
delete : (%,Integer) -> %
delete : (%,UniversalSegment Integer) -> %
elt : (%,Integer,Character) -> Character
empty : () -> %
empty? : % -> Boolean
entries : % -> List Character
entry? : (Character,%) -> Boolean
         if $ has finiteAggregate
         and Character has SETCAT
eq? : (%,%) -> Boolean
eval : (%,List Character,List Character) -> %
         if Character has EVALAB CHAR
         and Character has SETCAT
eval : (%,Character,Character) -> %
         if Character has EVALAB CHAR
         and Character has SETCAT
eval : (%,Equation Character) -> %
         if Character has EVALAB CHAR
         and Character has SETCAT
eval : (%,List Equation Character) -> %
         if Character has EVALAB CHAR
         and Character has SETCAT
every? : ((Character -> Boolean),%) -> Boolean
         if $ has finiteAggregate
fill! : (%,Character) -> %
         if $ has shallowlyMutable
find : ((Character -> Boolean),%) -> Union(Character,"failed")
first : % -> Character
         if Integer has ORDSET
hash : % -> SingleInteger
         if Character has SETCAT
index? : (Integer,%) -> Boolean
indices : % -> List Integer
insert : (Character,%,Integer) -> %
insert : (%,%,Integer) -> %
latex : % -> String if Character has SETCAT
less? : (%,NonNegativeInteger) -> Boolean
map : ((Character -> Character),%) -> %
map : (((Character,Character) -> Character),%,%) -> %
map! : ((Character -> Character),%) -> %
         if $ has shallowlyMutable
max : (%,%) -> % if Character has ORDSET
maxIndex : % -> Integer if Integer has ORDSET
```

```
member? : (Character,%) -> Boolean
         if Character has SETCAT
         and $ has finiteAggregate
members : % -> List Character
         if $ has finiteAggregate
merge : (%,%) -> % if Character has ORDSET
merge : (((Character,Character) -> Boolean),%,%) -> %
min : (%,%) -> % if Character has ORDSET
minIndex : % -> Integer if Integer has ORDSET
more? : (%,NonNegativeInteger) -> Boolean
new : (NonNegativeInteger,Character) -> %
parts : % -> List Character if $ has finiteAggregate
position : (Character,%) -> Integer
         if Character has SETCAT
position : ((Character -> Boolean),%) -> Integer
position : (Character,%,Integer) -> Integer
         if Character has SETCAT
qelt : (%,Integer) -> Character
qsetelt! : (%,Integer,Character) -> Character
         if $ has shallowlyMutable
reduce : (((Character,Character) -> Character),%) -> Character
         if $ has finiteAggregate
reduce :
  (((Character,Character) -> Character),%,Character) -> Character
         if $ has finiteAggregate
reduce :
  (((Character,Character) -> Character),%,Character,Character)
      -> Character
         if Character has SETCAT and $ has finiteAggregate
remove : ((Character -> Boolean),%) -> %
         if $ has finiteAggregate
remove : (Character,%) -> %
         if Character has SETCAT
         and $ has finiteAggregate
removeDuplicates : % -> %
         if Character has SETCAT
         and $ has finiteAggregate
reverse : % -> %
reverse! : % -> % if $ has shallowlyMutable
sample : () -> %
select : ((Character -> Boolean),%) -> %
         if $ has finiteAggregate
setelt : (%,UniversalSegment Integer,Character) -> Character
         if $ has shallowlyMutable
sort! : (((Character,Character) -> Boolean),%) -> %
         if $ has shallowlyMutable
sorted? : (((Character,Character) -> Boolean),%) -> Boolean
setelt : (%,Integer,Character) -> Character
         if $ has shallowlyMutable
size? : (%,NonNegativeInteger) -> Boolean
```

```
 sort : % -> % if Character has ORDSET
 sort : (((Character,Character) -> Boolean),%) -> %
 sort! : % -> %
          if Character has ORDSET
          and $ has shallowlyMutable
 sorted? : % -> Boolean if Character has ORDSET
 swap! : (%,Integer,Integer) -> Void
          if $ has shallowlyMutable
 #? : % -> NonNegativeInteger if $ has finiteAggregate
 ?~=? : (%,%) -> Boolean if Character has SETCAT
 ?.? : (%,UniversalSegment Integer) -> %
 ?.? : (%,Integer) -> Character
 ?<? : (%,%) -> Boolean if Character has ORDSET
 ?<=? : (%,%) -> Boolean if Character has ORDSET
 ?=? : (%,%) -> Boolean if Character has SETCAT
 ?>? : (%,%) -> Boolean if Character has ORDSET
 ?>=? : (%,%) -> Boolean if Character has ORDSET
```

— category SRAGG StringAggregate —

```
)abbrev category SRAGG StringAggregate
++ Author: Stephen Watt and Michael Monagan.
++ revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A string aggregate is a category for strings, that is,
++ one dimensional arrays of characters.

StringAggregate: Category == OneDimensionalArrayAggregate Character with
    lowerCase            : % -> %
      ++ lowerCase(s) returns the string with all characters in lower case.
    lowerCase_!: % -> %
      ++ lowerCase!(s) destructively replaces the alphabetic characters
      ++ in s by lower case.
    upperCase            : % -> %
      ++ upperCase(s) returns the string with all characters in upper case.
    upperCase_!: % -> %
      ++ upperCase!(s) destructively replaces the alphabetic characters
      ++ in s by upper case characters.
    prefix?            : (%, %) -> Boolean
      ++ prefix?(s,t) tests if the string s is the initial substring of t.
      ++ Note that \axiom{prefix?(s,t) ==
```

```
    ++   reduce(and,[s.i = t.i for i in 0..maxIndex s])}.
  suffix?            : (%, %) -> Boolean
    ++ suffix?(s,t) tests if the string s is the final substring of t.
    ++ Note that \axiom{suffix?(s,t) ==
    ++  reduce(and,[s.i = t.(n - m + i) for i in 0..maxIndex s])}
    ++ where m and n denote the maxIndex of s and t respectively.
  substring?: (%, %, Integer) -> Boolean
    ++ substring?(s,t,i) tests if s is a substring of t beginning at
    ++ index i.
    ++ Note that \axiom{substring?(s,t,0) = prefix?(s,t)}.
  match: (%, %, Character) -> NonNegativeInteger
    ++ match(p,s,wc) tests if pattern \axiom{p} matches subject \axiom{s}
    ++ where \axiom{wc} is a wild card character. If no match occurs,
    ++ the index \axiom{0} is returned; otheriwse, the value returned
    ++ is the first index of the first character in the subject matching
    ++ the subject (excluding that matched by an initial wild-card).
    ++ For example, \axiom{match("*to*","yorktown","*")} returns \axiom{5}
    ++ indicating a successful match starting at index \axiom{5} of
    ++ \axiom{"yorktown"}.
  match?: (%, %, Character) -> Boolean
    ++ match?(s,t,c) tests if s matches t except perhaps for
    ++ multiple and consecutive occurrences of character c.
    ++ Typically c is the blank character.
  replace            : (%, UniversalSegment(Integer), %) -> %
    ++ replace(s,i..j,t) replaces the substring \axiom{s(i..j)}
    ++ of s by string t.
  position           : (%, %, Integer) -> Integer
    ++ position(s,t,i) returns the position j of the substring s in
    ++ string t, where \axiom{j >= i} is required.
  position           : (CharacterClass, %, Integer) -> Integer
    ++ position(cc,t,i) returns the position \axiom{j >= i} in t of
    ++ the first character belonging to cc.
  coerce             : Character -> %
    ++ coerce(c) returns c as a string s with the character c.
  split: (%, Character) -> List %
    ++ split(s,c) returns a list of substrings delimited by character c.
  split: (%, CharacterClass) -> List %
    ++ split(s,cc) returns a list of substrings delimited by
    ++ characters in cc.
  trim: (%, Character) -> %
    ++ trim(s,c) returns s with all characters c deleted from right
    ++ and left ends.
    ++ For example, \axiom{trim(" abc ", char " ")} returns \axiom{"abc"}.
  trim: (%, CharacterClass) -> %
    ++ trim(s,cc) returns s with all characters in cc deleted from right
    ++ and left ends.
    ++ For example, \axiom{trim("(abc)", charClass "()")}
    ++ returns \axiom{"abc"}.
  leftTrim: (%, Character) -> %
    ++ leftTrim(s,c) returns s with all leading characters c deleted.
```

```
      ++ For example, \axiom{leftTrim("  abc  ", char " ")}
      ++ returns \axiom{"abc  "}.
    leftTrim: (%, CharacterClass) -> %
      ++ leftTrim(s,cc) returns s with all leading characters in cc deleted.
      ++ For example, \axiom{leftTrim("(abc)", charClass "()")}
      ++ returns \axiom{"abc)"}.
    rightTrim: (%, Character) -> %
      ++ rightTrim(s,c) returns s with all trailing occurrences of c deleted.
      ++ For example, \axiom{rightTrim("  abc  ", char " ")}
      ++ returns \axiom{"  abc"}.
    rightTrim: (%, CharacterClass) -> %
      ++ rightTrim(s,cc) returns s with all trailing occurences of
      ++ characters in cc deleted.
      ++ For example, \axiom{rightTrim("(abc)", charClass "()")}
      ++ returns \axiom{"(abc"}.
    elt: (%, %) -> %
      ++ elt(s,t) returns the concatenation of s and t. It is provided to
      ++ allow juxtaposition of strings to work as concatenation.
      ++ For example, \axiom{"smoo" "shed"} returns \axiom{"smooshed"}.
 add
   trim(s: %, c:  Character)          == leftTrim(rightTrim(s, c),        c)
   trim(s: %, cc: CharacterClass) == leftTrim(rightTrim(s, cc), cc)
   lowerCase s                 == lowerCase_! copy s
   upperCase s                 == upperCase_! copy s
   prefix?(s, t)         == substring?(s, t, minIndex t)
   coerce(c:Character):% == new(1, c)
   elt(s:%, t:%): %         == concat(s,t)$%
```

———-

— SRAGG.dotabb —

```
"SRAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=SRAGG"];
"SRAGG" -> "A1AGG"
```

———-

— SRAGG.dotfull —

```
"StringAggregate()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SRAGG"];
"StringAggregate()" -> "OneDimensionalArrayAggregate(Character)"
```

———-

— SRAGG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"StringAggregate()" [color=lightblue];
"StringAggregate()" -> "OneDimensionalArrayAggregate(Character)"

"OneDimensionalArrayAggregate(Character)" [color=seagreen];
"OneDimensionalArrayAggregate(Character)" ->
    "OneDimensionalArrayAggregate(a:Type)"

"OneDimensionalArrayAggregate(a:Type)" [color=lightblue];
"OneDimensionalArrayAggregate(a:Type)" ->
    "FiniteLinearAggregate(a:Type)"

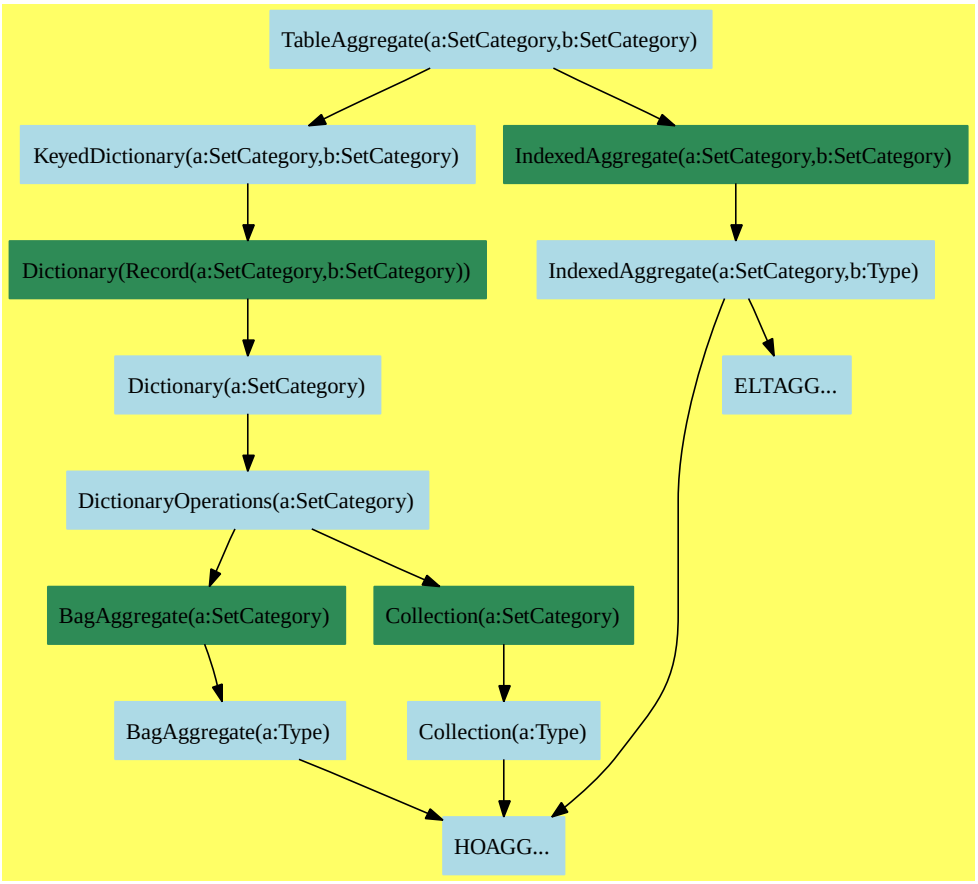
"FiniteLinearAggregate(a:Type)" [color=lightblue];
"FiniteLinearAggregate(a:Type)" -> "LinearAggregate(a:Type)"

"LinearAggregate(a:Type)" [color=lightblue];
"LinearAggregate(a:Type)" -> "IndexedAggregate(b:Integer,a:Type)"
"LinearAggregate(a:Type)" -> "CLAGG..."

"IndexedAggregate(b:Integer,a:Type)" [color=seagreen];
"IndexedAggregate(b:Integer,a:Type)" -> "IXAGG..."

"CLAGG..." [color=lightblue];
"IXAGG..." [color=lightblue];
}
```

9.0.132 TableAggregate (TBAGG)



— TableAggregate.input —

```
)set break resume
)sys rm -f TableAggregate.output
)spool TableAggregate.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show TableAggregate
--R TableAggregate(Key: SetCategory,Entry: SetCategory)  is a category constructor
--R Abbreviation for TableAggregate is TBAGG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for TBAGG
```

```
--R
--R------------------------------ Operations --------------------------------
--R copy : % -> %                          dictionary : () -> %
--R elt : (%,Key,Entry) -> Entry           ?.? : (%,Key) -> Entry
--R empty : () -> %                        empty? : % -> Boolean
--R entries : % -> List Entry              eq? : (%,%) -> Boolean
--R index? : (Key,%) -> Boolean            indices : % -> List Key
--R key? : (Key,%) -> Boolean              keys : % -> List Key
--R map : ((Entry -> Entry),%) -> %        qelt : (%,Key) -> Entry
--R sample : () -> %                       setelt : (%,Key,Entry) -> Entry
--R table : () -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?=? : (%,%) -> Boolean if Entry has SETCAT or Record(key: Key,entry: Entry) has SETCAT
--R any? : ((Entry -> Boolean),%) -> Boolean if $ has finiteAggregate
--R any? : ((Record(key: Key,entry: Entry) -> Boolean),%) -> Boolean if $ has finiteAggregate
--R bag : List Record(key: Key,entry: Entry) -> %
--R coerce : % -> OutputForm if Entry has SETCAT or Record(key: Key,entry: Entry) has SETCAT
--R construct : List Record(key: Key,entry: Entry) -> %
--R convert : % -> InputForm if Record(key: Key,entry: Entry) has KONVERT INFORM
--R count : ((Entry -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R count : (Entry,%) -> NonNegativeInteger if Entry has SETCAT and $ has finiteAggregate
--R count : (Record(key: Key,entry: Entry),%) -> NonNegativeInteger if Record(key: Key,entry: En
--R count : ((Record(key: Key,entry: Entry) -> Boolean),%) -> NonNegativeInteger if $ has finite
--R dictionary : List Record(key: Key,entry: Entry) -> %
--R entry? : (Entry,%) -> Boolean if $ has finiteAggregate and Entry has SETCAT
--R eval : (%,List Equation Entry) -> % if Entry has EVALAB Entry and Entry has SETCAT
--R eval : (%,Equation Entry) -> % if Entry has EVALAB Entry and Entry has SETCAT
--R eval : (%,Entry,Entry) -> % if Entry has EVALAB Entry and Entry has SETCAT
--R eval : (%,List Entry,List Entry) -> % if Entry has EVALAB Entry and Entry has SETCAT
--R eval : (%,List Record(key: Key,entry: Entry),List Record(key: Key,entry: Entry)) -> % if Rec
--R eval : (%,Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> % if Record(key: K
--R eval : (%,Equation Record(key: Key,entry: Entry)) -> % if Record(key: Key,entry: Entry) has
--R eval : (%,List Equation Record(key: Key,entry: Entry)) -> % if Record(key: Key,entry: Entry)
--R every? : ((Entry -> Boolean),%) -> Boolean if $ has finiteAggregate
--R every? : ((Record(key: Key,entry: Entry) -> Boolean),%) -> Boolean if $ has finiteAggregate
--R extract! : % -> Record(key: Key,entry: Entry)
--R fill! : (%,Entry) -> % if $ has shallowlyMutable
--R find : ((Record(key: Key,entry: Entry) -> Boolean),%) -> Union(Record(key: Key,entry: Entry)
--R first : % -> Entry if Key has ORDSET
--R hash : % -> SingleInteger if Entry has SETCAT or Record(key: Key,entry: Entry) has SETCAT
--R insert! : (Record(key: Key,entry: Entry),%) -> %
--R inspect : % -> Record(key: Key,entry: Entry)
--R latex : % -> String if Entry has SETCAT or Record(key: Key,entry: Entry) has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R map : (((Entry,Entry) -> Entry),%,%) -> %
--R map : ((Record(key: Key,entry: Entry) -> Record(key: Key,entry: Entry)),%) -> %
--R map! : ((Entry -> Entry),%) -> % if $ has shallowlyMutable
--R map! : ((Record(key: Key,entry: Entry) -> Record(key: Key,entry: Entry)),%) -> % if $ has sh
--R maxIndex : % -> Key if Key has ORDSET
--R member? : (Entry,%) -> Boolean if Entry has SETCAT and $ has finiteAggregate
```

```
--R member? : (Record(key: Key,entry: Entry),%) -> Boolean if Record(key: Key,entry: Entry) has SETCAT and $
--R members : % -> List Entry if $ has finiteAggregate
--R members : % -> List Record(key: Key,entry: Entry) if $ has finiteAggregate
--R minIndex : % -> Key if Key has ORDSET
--R more? : (%,NonNegativeInteger) -> Boolean
--R parts : % -> List Entry if $ has finiteAggregate
--R parts : % -> List Record(key: Key,entry: Entry) if $ has finiteAggregate
--R qsetelt! : (%,Key,Entry) -> Entry if $ has shallowlyMutable
--R reduce : (((Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> Record(key: Key,entry: Entry
--R reduce : (((Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> Record(key: Key,entry: Entry
--R reduce : (((Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> Record(key: Key,entry: Entry
--R remove : ((Record(key: Key,entry: Entry) -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (Record(key: Key,entry: Entry),%) -> % if Record(key: Key,entry: Entry) has SETCAT and $ has fi
--R remove! : (Key,%) -> Union(Entry,"failed")
--R remove! : ((Record(key: Key,entry: Entry) -> Boolean),%) -> % if $ has finiteAggregate
--R remove! : (Record(key: Key,entry: Entry),%) -> % if $ has finiteAggregate
--R removeDuplicates : % -> % if Record(key: Key,entry: Entry) has SETCAT and $ has finiteAggregate
--R search : (Key,%) -> Union(Entry,"failed")
--R select : ((Record(key: Key,entry: Entry) -> Boolean),%) -> % if $ has finiteAggregate
--R select! : ((Record(key: Key,entry: Entry) -> Boolean),%) -> % if $ has finiteAggregate
--R size? : (%,NonNegativeInteger) -> Boolean
--R swap! : (%,Key,Key) -> Void if $ has shallowlyMutable
--R table : List Record(key: Key,entry: Entry) -> %
--R ?~=? : (%,%) -> Boolean if Entry has SETCAT or Record(key: Key,entry: Entry) has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— TableAggregate.help —

```
=====================================================================
TableAggregate examples
=====================================================================

See Also:
o )show TableAggregate
```

———-

See:

⇒ “AssociationListAggregate” (ALAGG) 10.0.134 on page 819
⇐ “IndexedAggregate” (IXAGG) 5.0.74 on page 318
⇐ “KeyedDictionary” (KDAGG) 8.0.111 on page 630

Exports:

any?	bag	coerce	construct	convert
copy	count	dictionary	elt	empty
empty?	entries	entry?	eq?	eval
every?	extract!	fill!	find	first
hash	index?	indices	insert!	inspect
key?	keys	latex	less?	map
map!	maxIndex	member?	members	minIndex
more?	parts	qelt	qsetelt!	reduce
remove	remove!	removeDuplicates	sample	search
select	select!	setelt	size?	swap!
table	#?	?=?	?.?	?~=?

Attributes exported:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **nil**

Attributes Used:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.

These are directly exported but not implemented:

```
setelt : (%,Key,Entry) -> Entry
```

These are implemented by this category:

```
any? : ((Entry -> Boolean),%) -> Boolean
    if $ has finiteAggregate
coerce : % -> OutputForm
         if Entry has SETCAT
         or Record(key: Key,entry: Entry) has SETCAT
count : ((Entry -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
elt : (%,Key,Entry) -> Entry
entries : % -> List Entry
every? : ((Entry -> Boolean),%) -> Boolean
        if $ has finiteAggregate
extract! : % -> Record(key: Key,entry: Entry)
find :
  ((Record(key: Key,entry: Entry) -> Boolean),%)
     -> Union(Record(key: Key,entry: Entry),"failed")
index? : (Key,%) -> Boolean
```

```
indices : % -> List Key
insert! : (Record(key: Key,entry: Entry),%) -> %
inspect : % -> Record(key: Key,entry: Entry)
map : (((Entry,Entry) -> Entry),%,%) -> %
map : ((Record(key: Key,entry: Entry)
      -> Record(key: Key,entry: Entry)),%) -> %
map! : ((Entry -> Entry),%) -> % if $ has shallowlyMutable
map! :
  ((Record(key: Key,entry: Entry)
      -> Record(key: Key,entry: Entry)),%)
       -> % if $ has shallowlyMutable
parts : % -> List Entry if $ has finiteAggregate
parts : % -> List Record(key: Key,entry: Entry)
         if $ has finiteAggregate
remove! : (Key,%) -> Union(Entry,"failed")
table : () -> %
table : List Record(key: Key,entry: Entry) -> %
?.? : (%,Key) -> Entry
?=? : (%,%) -> Boolean
         if Entry has SETCAT
         or Record(key: Key,entry: Entry) has SETCAT
```

These exports come from (p630) KeyedDictionary(Key,Entry)
where Key:SetCategory and Entry:SetCategory
and RecKE=Record(key: Key,entry: Entry):

```
any? : ((RecKE -> Boolean),%) -> Boolean
         if $ has finiteAggregate
bag : List RecKE -> %
construct : List RecKE -> %
convert : % -> InputForm if RecKE has KONVERT INFORM
copy : % -> %
count : (Entry,%) -> NonNegativeInteger
         if Entry has SETCAT
         and $ has finiteAggregate
count : ((RecKE -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
dictionary : () -> %
dictionary : List RecKE -> %
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List RecKE,List RecKE) -> %
         if RecKE has EVALAB RecKE
         and RecKE has SETCAT
eval : (%,RecKE,RecKE) -> %
         if RecKE has EVALAB RecKE
         and RecKE has SETCAT
eval : (%,Equation RecKE) -> %
         if RecKE has EVALAB RecKE
```

```
         and RecKE has SETCAT
eval : (%,List Equation RecKE) -> %
         if RecKE has EVALAB RecKE
         and RecKE has SETCAT
every? : ((RecKE -> Boolean),%) -> Boolean
         if $ has finiteAggregate
key? : (Key,%) -> Boolean
keys : % -> List Key
hash : % -> SingleInteger
         if Entry has SETCAT
         or RecKE has SETCAT
latex : % -> String
         if Entry has SETCAT
         or RecKE has SETCAT
less? : (%,NonNegativeInteger) -> Boolean
member? : (RecKE,%) -> Boolean
    if RecKE has SETCAT
     and $ has finiteAggregate
members : % -> List RecKE if $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
reduce :
 (((RecKE,RecKE) -> RecKE),%) -> RecKE
         if $ has finiteAggregate
reduce :
 (((RecKE,RecKE)->RecKE),%,RecKE) -> RecKE
         if $ has finiteAggregate
reduce :
 (((RecKE,RecKE)->RecKE),%,RecKE,RecKE)
   -> RecKE
         if RecKE has SETCAT
         and $ has finiteAggregate
remove : ((RecKE -> Boolean),%) -> %
         if $ has finiteAggregate
remove : (RecKE,%) -> %
         if RecKE has SETCAT
         and $ has finiteAggregate
remove! : ((RecKE -> Boolean),%) -> %
         if $ has finiteAggregate
remove! : (RecKE,%) -> % if $ has finiteAggregate
removeDuplicates : % -> %
         if RecKE has SETCAT
         and $ has finiteAggregate
sample : () -> %
search : (Key,%) -> Union(Entry,"failed")
select : ((RecKE -> Boolean),%) -> %
         if $ has finiteAggregate
select! : ((RecKE -> Boolean),%) -> %
         if $ has finiteAggregate
#? : % -> NonNegativeInteger if $ has finiteAggregate
?~=? : (%,%) -> Boolean
```

```
          if Entry has SETCAT
          or RecKE has SETCAT
```

These exports come from (p318) IndexedAggregate(Key,Entry))
where Key:SetCategory and Entry:SetCategory
and RecKE=Record(key: Key,entry: Entry):

```
 count : (RecKE,%) -> NonNegativeInteger
          if RecKE has SETCAT
          and $ has finiteAggregate
 entry? : (Entry,%) -> Boolean
          if $ has finiteAggregate
          and Entry has SETCAT
 eval : (%,List Equation Entry) -> %
          if Entry has EVALAB Entry
          and Entry has SETCAT
 eval : (%,Equation Entry) -> %
          if Entry has EVALAB Entry
          and Entry has SETCAT
 eval : (%,Entry,Entry) -> %
          if Entry has EVALAB Entry
          and Entry has SETCAT
 eval : (%,List Entry,List Entry) -> %
          if Entry has EVALAB Entry
          and Entry has SETCAT
 fill! : (%,Entry) -> % if $ has shallowlyMutable
 first : % -> Entry if Key has ORDSET
 map : ((Entry -> Entry),%) -> %
 maxIndex : % -> Key if Key has ORDSET
 member? : (Entry,%) -> Boolean
          if Entry has SETCAT
          and $ has finiteAggregate
 members : % -> List Entry if $ has finiteAggregate
 minIndex : % -> Key if Key has ORDSET
 qelt : (%,Key) -> Entry
 qsetelt! : (%,Key,Entry) -> Entry if $ has shallowlyMutable
 size? : (%,NonNegativeInteger) -> Boolean
 swap! : (%,Key,Key) -> Void if $ has shallowlyMutable
```

— category TBAGG TableAggregate —

```
)abbrev category TBAGG TableAggregate
++ Author: Michael Monagan, Stephen Watt;
++ revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
```

```
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A table aggregate is a model of a table, i.e. a discrete many-to-one
++ mapping from keys to entries.

TableAggregate(Key:SetCategory, Entry:SetCategory): Category ==
  Join(KeyedDictionary(Key,Entry),IndexedAggregate(Key,Entry)) with
   setelt: (%,Key,Entry) -> Entry           -- setelt_! later
     ++ setelt(t,k,e) (also written \axiom{t.k := e}) is equivalent
     ++ to \axiom{(insert([k,e],t); e)}.
   table: () -> %
     ++ table()$T creates an empty table of type T.
     ++
     ++E Data:=Record(age:Integer,gender:String)
     ++E a1:AssociationList(String,Data):=table()
     ++E a1."tim":=[55,"male"]$Data
   table: List Record(key:Key,entry:Entry) -> %
     ++ table([x,y,...,z]) creates a table consisting of entries
     ++ \axiom{x,y,...,z}.
   -- to become table: Record(key:Key,entry:Entry)* -> %
   map: ((Entry, Entry) -> Entry, %, %) -> %
     ++ map(fn,t1,t2) creates a new table t from given tables t1 and t2 with
     ++ elements fn(x,y) where x and y are corresponding elements from t1
     ++ and t2 respectively.
 add
   table()               == empty()
   table l               == dictionary l
-- empty()               == dictionary()

   insert_!(p, t)      == (t(p.key) := p.entry; t)
   indices t              == keys t

   coerce(t:%):OutputForm ==
     prefix("table"::OutputForm,
                   [k::OutputForm = (t.k)::OutputForm for k in keys t])

   elt(t, k) ==
      (r := search(k, t)) case Entry => r::Entry
      error "key not in table"

   elt(t, k, e) ==
      (r := search(k, t)) case Entry => r::Entry
      e

   map_!(f, t) ==
      for k in keys t repeat t.k := f t.k
      t
```

```
   map(f:(Entry, Entry) -> Entry, s:%, t:%) ==
      z := table()
      for k in keys s | key?(k, t) repeat z.k := f(s.k, t.k)
      z

-- map(f, s, t, x) ==
--    z := table()
--    for k in keys s repeat z.k := f(s.k, t(k, x))
--    for k in keys t | not key?(k, s) repeat z.k := f(t.k, x)
--    z

   if % has finiteAggregate then
     parts(t:%):List Record(key:Key,entry:Entry) ==
         [[k, t.k] for k in keys t]
     parts(t:%):List Entry   == [t.k for k in keys t]
     entries(t:%):List Entry == parts(t)

     s:% = t:% ==
       eq?(s,t) => true
       #s ^= #t => false
       for k in keys s repeat
         (e := search(k, t)) _
           case "failed" or (e::Entry) ^= s.k => return false
       true

     map(f: Record(key:Key,entry:Entry)->Record(key:Key,entry:Entry),t:%):%==
       z := table()
       for k in keys t repeat
         ke: Record(key:Key,entry:Entry) := f [k, t.k]
         z ke.key := ke.entry
       z
     map_!(f:Record(key:Key,entry:Entry)->Record(key:Key,entry:Entry),t:%):%_
      ==
       lke: List Record(key:Key,entry:Entry) := nil()
       for k in keys t repeat
         lke := cons(f [k, remove_!(k,t)::Entry], lke)
       for ke in lke repeat
         t ke.key := ke.entry
       t

     inspect(t: %): Record(key:Key,entry:Entry) ==
       ks := keys t
       empty? ks => error "Cannot extract from an empty aggregate"
       [first ks, t first ks]

     find(f: Record(key:Key,entry:Entry)->Boolean, t:%):_
           Union(Record(key:Key,entry:Entry), "failed") ==
       for ke in parts(t)@List(Record(key:Key,entry:Entry)) _
          repeat if f ke then return ke
```

```
      "failed"

    index?(k: Key, t: %): Boolean ==
      search(k,t) case Entry

    remove_!(x:Record(key:Key,entry:Entry), t:%) ==
      if member?(x, t) then remove_!(x.key, t)
      t
    extract_!(t: %): Record(key:Key,entry:Entry) ==
      k: Record(key:Key,entry:Entry) := inspect t
      remove_!(k.key, t)
      k

    any?(f: Entry->Boolean, t: %): Boolean ==
      for k in keys t | f t k repeat return true
      false
    every?(f: Entry->Boolean, t: %): Boolean ==
      for k in keys t | not f t k repeat return false
      true
    count(f: Entry->Boolean, t: %): NonNegativeInteger ==
      tally: NonNegativeInteger := 0
      for k in keys t | f t k repeat tally := tally + 1
      tally
```

———-

— TBAGG.dotabb —

```
"TBAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=TBAGG"];
"TBAGG" -> "KDAGG"
"TBAGG" -> "IXAGG"
```

———-

— TBAGG.dotfull —

```
"TableAggregate(a:SetCategory,b:SetCategory)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=TBAGG"];
"TableAggregate(a:SetCategory,b:SetCategory)" ->
    "KeyedDictionary(a:SetCategory,b:SetCategory)"
"TableAggregate(a:SetCategory,b:SetCategory)" ->
    "IndexedAggregate(a:SetCategory,b:SetCategory)"
```

———-

— TBAGG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"TableAggregate(a:SetCategory,b:SetCategory)" [color=lightblue];
"TableAggregate(a:SetCategory,b:SetCategory)" ->
    "KeyedDictionary(a:SetCategory,b:SetCategory)"
"TableAggregate(a:SetCategory,b:SetCategory)" ->
    "IndexedAggregate(a:SetCategory,b:SetCategory)"

"IndexedAggregate(a:SetCategory,b:SetCategory)" [color=seagreen];
"IndexedAggregate(a:SetCategory,b:SetCategory)" ->
    "IndexedAggregate(a:SetCategory,b:Type)"

"IndexedAggregate(a:SetCategory,b:Type)" [color=lightblue];
"IndexedAggregate(a:SetCategory,b:Type)" -> "HOAGG..."
"IndexedAggregate(a:SetCategory,b:Type)" -> "ELTAGG..."

"KeyedDictionary(a:SetCategory,b:SetCategory)" [color=lightblue];
"KeyedDictionary(a:SetCategory,b:SetCategory)" ->
    "Dictionary(Record(a:SetCategory,b:SetCategory))"

"Dictionary(Record(a:SetCategory,b:SetCategory))" [color=seagreen];
"Dictionary(Record(a:SetCategory,b:SetCategory))" ->
    "Dictionary(a:SetCategory)"

"Dictionary(a:SetCategory)" [color=lightblue];
"Dictionary(a:SetCategory)" -> "DictionaryOperations(a:SetCategory)"

"DictionaryOperations(a:SetCategory)" [color=lightblue];
"DictionaryOperations(a:SetCategory)" -> "BagAggregate(a:SetCategory)"
"DictionaryOperations(a:SetCategory)" -> "Collection(a:SetCategory)"

"BagAggregate(a:SetCategory)" [color=seagreen];
"BagAggregate(a:SetCategory)" -> "BagAggregate(a:Type)"

"BagAggregate(a:Type)" [color=lightblue];
"BagAggregate(a:Type)" -> "HOAGG..."

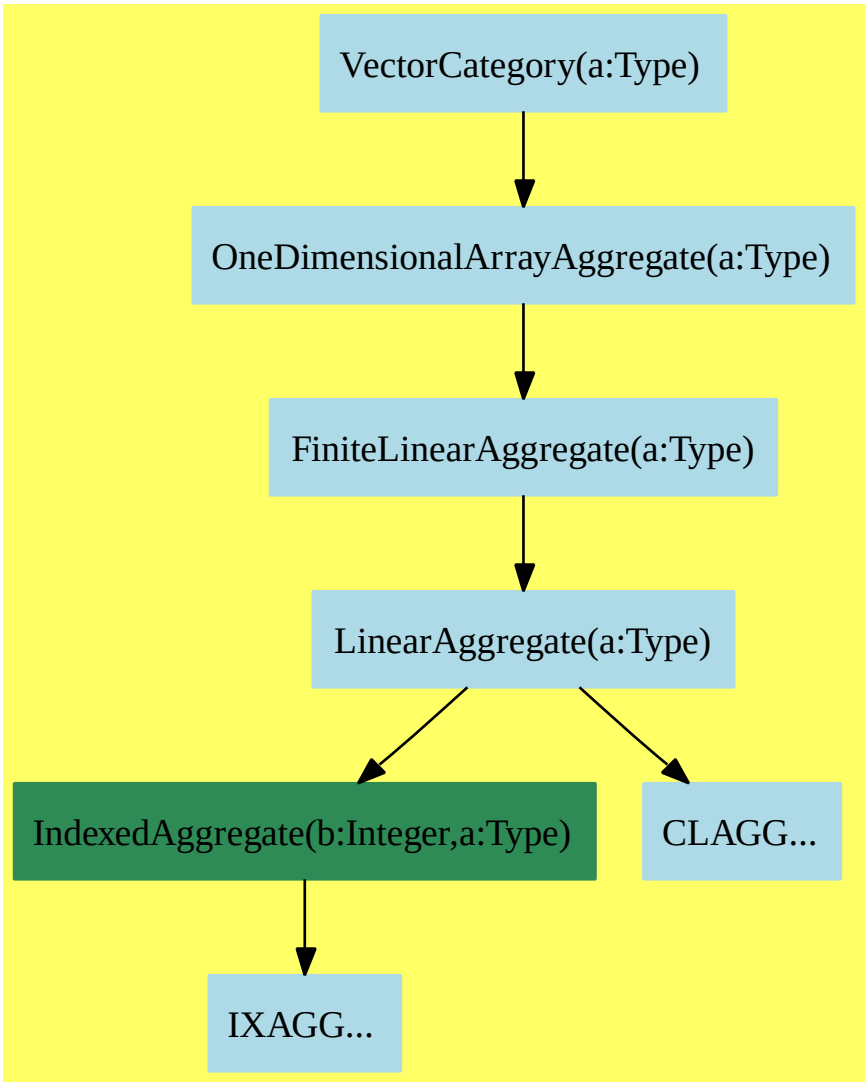
"Collection(a:SetCategory)" [color=seagreen];
"Collection(a:SetCategory)" -> "Collection(a:Type)"

"Collection(a:Type)" [color=lightblue];
"Collection(a:Type)" -> "HOAGG..."

"ELTAGG..." [color=lightblue];
"HOAGG..." [color=lightblue];
}
```

———-

9.0.133 VectorCategory (VECTCAT)



— **VectorCategory.input** —

```
)set break resume
)sys rm -f VectorCategory.output
```

```
)spool VectorCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show VectorCategory
--R VectorCategory R: Type  is a category constructor
--R Abbreviation for VectorCategory is VECTCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for VECTCAT
--R
--R------------------------------ Operations --------------------------------
--R -? : % -> % if R has ABELGRP          concat : List % -> %
--R concat : (%,%) -> %                   concat : (R,%) -> %
--R concat : (%,R) -> %                   construct : List R -> %
--R copy : % -> %                         delete : (%,Integer) -> %
--R ?.? : (%,Integer) -> R                elt : (%,Integer,R) -> R
--R empty : () -> %                       empty? : % -> Boolean
--R entries : % -> List R                 eq? : (%,%) -> Boolean
--R index? : (Integer,%) -> Boolean       indices : % -> List Integer
--R insert : (%,%,Integer) -> %           insert : (R,%,Integer) -> %
--R map : (((R,R) -> R),%,%) -> %         map : ((R -> R),%) -> %
--R new : (NonNegativeInteger,R) -> %     qelt : (%,Integer) -> R
--R reverse : % -> %                      sample : () -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?*? : (%,R) -> % if R has MONOID
--R ?*? : (R,%) -> % if R has MONOID
--R ?*? : (Integer,%) -> % if R has ABELGRP
--R ?+? : (%,%) -> % if R has ABELSG
--R ?-? : (%,%) -> % if R has ABELGRP
--R ?<? : (%,%) -> Boolean if R has ORDSET
--R ?<=? : (%,%) -> Boolean if R has ORDSET
--R ?=? : (%,%) -> Boolean if R has SETCAT
--R ?>? : (%,%) -> Boolean if R has ORDSET
--R ?>=? : (%,%) -> Boolean if R has ORDSET
--R any? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
--R coerce : % -> OutputForm if R has SETCAT
--R convert : % -> InputForm if R has KONVERT INFORM
--R copyInto! : (%,%,Integer) -> % if $ has shallowlyMutable
--R count : (R,%) -> NonNegativeInteger if R has SETCAT and $ has finiteAggregate
--R count : ((R -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R cross : (%,%) -> % if R has RING
--R delete : (%,UniversalSegment Integer) -> %
--R dot : (%,%) -> R if R has RING
--R ?.? : (%,UniversalSegment Integer) -> %
--R entry? : (R,%) -> Boolean if $ has finiteAggregate and R has SETCAT
--R eval : (%,List R,List R) -> % if R has EVALAB R and R has SETCAT
--R eval : (%,R,R) -> % if R has EVALAB R and R has SETCAT
--R eval : (%,Equation R) -> % if R has EVALAB R and R has SETCAT
```

```
--R eval : (%,List Equation R) -> % if R has EVALAB R and R has SETCAT
--R every? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
--R fill! : (%,R) -> % if $ has shallowlyMutable
--R find : ((R -> Boolean),%) -> Union(R,"failed")
--R first : % -> R if Integer has ORDSET
--R hash : % -> SingleInteger if R has SETCAT
--R latex : % -> String if R has SETCAT
--R length : % -> R if R has RING and R has RADCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R magnitude : % -> R if R has RING and R has RADCAT
--R map! : ((R -> R),%) -> % if $ has shallowlyMutable
--R max : (%,%) -> % if R has ORDSET
--R maxIndex : % -> Integer if Integer has ORDSET
--R member? : (R,%) -> Boolean if R has SETCAT and $ has finiteAggregate
--R members : % -> List R if $ has finiteAggregate
--R merge : (%,%) -> % if R has ORDSET
--R merge : (((R,R) -> Boolean),%,%) -> %
--R min : (%,%) -> % if R has ORDSET
--R minIndex : % -> Integer if Integer has ORDSET
--R more? : (%,NonNegativeInteger) -> Boolean
--R outerProduct : (%,%) -> Matrix R if R has RING
--R parts : % -> List R if $ has finiteAggregate
--R position : (R,%,Integer) -> Integer if R has SETCAT
--R position : (R,%) -> Integer if R has SETCAT
--R position : ((R -> Boolean),%) -> Integer
--R qsetelt! : (%,Integer,R) -> R if $ has shallowlyMutable
--R reduce : (((R,R) -> R),%) -> R if $ has finiteAggregate
--R reduce : (((R,R) -> R),%,R) -> R if $ has finiteAggregate
--R reduce : (((R,R) -> R),%,R,R) -> R if R has SETCAT and $ has finiteAggregate
--R remove : ((R -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (R,%) -> % if R has SETCAT and $ has finiteAggregate
--R removeDuplicates : % -> % if R has SETCAT and $ has finiteAggregate
--R reverse! : % -> % if $ has shallowlyMutable
--R select : ((R -> Boolean),%) -> % if $ has finiteAggregate
--R setelt : (%,UniversalSegment Integer,R) -> R if $ has shallowlyMutable
--R setelt : (%,Integer,R) -> R if $ has shallowlyMutable
--R size? : (%,NonNegativeInteger) -> Boolean
--R sort : % -> % if R has ORDSET
--R sort : (((R,R) -> Boolean),%) -> %
--R sort! : % -> % if R has ORDSET and $ has shallowlyMutable
--R sort! : (((R,R) -> Boolean),%) -> % if $ has shallowlyMutable
--R sorted? : % -> Boolean if R has ORDSET
--R sorted? : (((R,R) -> Boolean),%) -> Boolean
--R swap! : (%,Integer,Integer) -> Void if $ has shallowlyMutable
--R zero : NonNegativeInteger -> % if R has ABELMON
--R ?~=? : (%,%) -> Boolean if R has SETCAT
--R
--E 1

)spool
```

```
)lisp (bye)
```

———-

— **VectorCategory.help** —

```
====================================================================
VectorCategory examples
====================================================================

See Also:
o )show VectorCategory
```

———-

See:

⇒ “PointCategory” (PTCAT) 10.0.146 on page 893
⇐ “OneDimensionalArrayAggregate” (A1AGG) 8.0.117 on page 684

Exports:

any?	coerce	concat	construct	convert
copy	copyInto!	count	cross	delete
dot	entry?	elt	empty	empty?
entries	eq?	eval	every?	fill!
find	first	hash	index?	indices
insert	latex	length	less?	magnitude
map	map!	max	maxIndex	member?
members	merge	min	minIndex	more?
new	outerProduct	parts	position	qelt
qsetelt!	reduce	remove	removeDuplicates	reverse
reverse!	sample	select	setelt	size?
sort	sort!	sorted?	swap!	zero
#?	?*?	?+?	?-?	?<?
?<=?	?=?	?>?	?>=?	-?
?.?	?~=?			

Attributes Exported:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **finiteAggregate** is true if it is an aggregate with a finite number of elements.

These are implemented by this category:

```
cross : (%,%) -> % if R has RING
dot : (%,%) -> R if R has RING
length : % -> R if R has RING and R has RADCAT
magnitude : % -> R if R has RING and R has RADCAT
outerProduct : (%,%) -> Matrix R if R has RING
zero : NonNegativeInteger -> % if R has ABELMON
?*? : (Integer,%) -> % if R has ABELGRP
?*? : (%,R) -> % if R has MONOID
?*? : (R,%) -> % if R has MONOID
?-? : (%,%) -> % if R has ABELGRP
-? : % -> % if R has ABELGRP
?+? : (%,%) -> % if R has ABELSG
```

These exports come from (p684) OneDimensionalArrayAggregate(R:Type):

```
any? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
coerce : % -> OutputForm if R has SETCAT
concat : (R,%) -> %
concat : (%,R) -> %
concat : List % -> %
concat : (%,%) -> %
construct : List R -> %
convert : % -> InputForm if R has KONVERT INFORM
copy : % -> %
copyInto! : (%,%,Integer) -> % if $ has shallowlyMutable
count : ((R -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
count : (R,%) -> NonNegativeInteger if R has SETCAT and $ has finiteAggregate
delete : (%,Integer) -> %
delete : (%,UniversalSegment Integer) -> %
elt : (%,Integer,R) -> R
empty : () -> %
empty? : % -> Boolean
entries : % -> List R
entry? : (R,%) -> Boolean if $ has finiteAggregate and R has SETCAT
eq? : (%,%) -> Boolean
eval : (%,List R,List R) -> % if R has EVALAB R and R has SETCAT
eval : (%,R,R) -> % if R has EVALAB R and R has SETCAT
eval : (%,Equation R) -> % if R has EVALAB R and R has SETCAT
eval : (%,List Equation R) -> % if R has EVALAB R and R has SETCAT
every? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
fill! : (%,R) -> % if $ has shallowlyMutable
find : ((R -> Boolean),%) -> Union(R,"failed")
first : % -> R if Integer has ORDSET
hash : % -> SingleInteger if R has SETCAT
index? : (Integer,%) -> Boolean
indices : % -> List Integer
insert : (%,%,Integer) -> %
insert : (R,%,Integer) -> %
latex : % -> String if R has SETCAT
less? : (%,NonNegativeInteger) -> Boolean
```

```
map : ((R -> R),%) -> %
map : (((R,R) -> R),%,%) -> %
map! : ((R -> R),%) -> % if $ has shallowlyMutable
max : (%,%) -> % if R has ORDSET
maxIndex : % -> Integer if Integer has ORDSET
member? : (R,%) -> Boolean if R has SETCAT and $ has finiteAggregate
members : % -> List R if $ has finiteAggregate
merge : (%,%) -> % if R has ORDSET
merge : (((R,R) -> Boolean),%,%) -> %
min : (%,%) -> % if R has ORDSET
minIndex : % -> Integer if Integer has ORDSET
more? : (%,NonNegativeInteger) -> Boolean
new : (NonNegativeInteger,R) -> %
parts : % -> List R if $ has finiteAggregate
position : (R,%) -> Integer if R has SETCAT
position : ((R -> Boolean),%) -> Integer
position : (R,%,Integer) -> Integer if R has SETCAT
qelt : (%,Integer) -> R
qsetelt! : (%,Integer,R) -> R if $ has shallowlyMutable
reduce : (((R,R) -> R),%) -> R if $ has finiteAggregate
reduce : (((R,R) -> R),%,R) -> R if $ has finiteAggregate
reduce : (((R,R) -> R),%,R,R) -> R if R has SETCAT and $ has finiteAggregate
remove : ((R -> Boolean),%) -> % if $ has finiteAggregate
remove : (R,%) -> % if R has SETCAT and $ has finiteAggregate
removeDuplicates : % -> % if R has SETCAT and $ has finiteAggregate
reverse : % -> %
reverse! : % -> % if $ has shallowlyMutable
sample : () -> %
select : ((R -> Boolean),%) -> % if $ has finiteAggregate
setelt : (%,Integer,R) -> R if $ has shallowlyMutable
setelt : (%,UniversalSegment Integer,R) -> R if $ has shallowlyMutable
size? : (%,NonNegativeInteger) -> Boolean
sort : % -> % if R has ORDSET
sort : (((R,R) -> Boolean),%) -> %
sort! : % -> % if R has ORDSET and $ has shallowlyMutable
sort! : (((R,R) -> Boolean),%) -> % if $ has shallowlyMutable
sorted? : % -> Boolean if R has ORDSET
sorted? : (((R,R) -> Boolean),%) -> Boolean
swap! : (%,Integer,Integer) -> Void if $ has shallowlyMutable
#? : % -> NonNegativeInteger if $ has finiteAggregate
?.? : (%,Integer) -> R
?.? : (%,UniversalSegment Integer) -> %
?=? : (%,%) -> Boolean if R has SETCAT
?<? : (%,%) -> Boolean if R has ORDSET
?<=? : (%,%) -> Boolean if R has ORDSET
?>? : (%,%) -> Boolean if R has ORDSET
?>=? : (%,%) -> Boolean if R has ORDSET
?~=? : (%,%) -> Boolean if R has SETCAT
```

— category VECTCAT VectorCategory —

```
)abbrev category VECTCAT VectorCategory
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors: DirectProductCategory, Vector, IndexedVector
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ \spadtype{VectorCategory} represents the type of vector like objects,
++ i.e. finite sequences indexed by some finite segment of the
++ integers. The operations available on vectors depend on the structure
++ of the underlying components. Many operations from the component domain
++ are defined for vectors componentwise. It can by assumed that extraction or
++ updating components can be done in constant time.

VectorCategory(R:Type): Category == OneDimensionalArrayAggregate R with
    if R has AbelianSemiGroup then
      _+ : (%, %) -> %
        ++ x + y returns the component-wise sum of the vectors x and y.
        ++ Error: if x and y are not of the same length.
    if R has AbelianMonoid then
      zero: NonNegativeInteger -> %
        ++ zero(n) creates a zero vector of length n.
    if R has AbelianGroup then
      _- : % -> %
        ++ -x negates all components of the vector x.
      _- : (%, %) -> %
        ++ x - y returns the component-wise difference of the vectors x and y.
        ++ Error: if x and y are not of the same length.
      _* : (Integer, %) -> %
        ++ n * y multiplies each component of the vector y by the integer n.
    if R has Monoid then
      _* : (R, %) -> %
        ++ r * y multiplies the element r times each component of the vector y.
      _* : (%, R) -> %
        ++ y * r multiplies each component of the vector y by the element r.
    if R has Ring then
      dot: (%, %) -> R
        ++ dot(x,y) computes the inner product of the two vectors x and y.
        ++ Error: if x and y are not of the same length.
      outerProduct: (%, %) -> Matrix R
        ++ outerProduct(u,v) constructs the matrix whose (i,j)'th element is
        ++ u(i)*v(j).
      cross: (%, %) -> %
        ++ vectorProduct(u,v) constructs the cross product of u and v.
```

```
      ++ Error: if u and v are not of length 3.
   if R has RadicalCategory and R has Ring then
     length: % -> R
       ++ length(v) computes the sqrt(dot(v,v)), i.e. the magnitude
     magnitude: % -> R
       ++ magnitude(v) computes the sqrt(dot(v,v)), i.e. the length
add
   if R has AbelianSemiGroup then
     u + v ==
       (n := #u) ^= #v => error "Vectors must be of the same length"
       map(_+ , u, v)

   if R has AbelianMonoid then
     zero n == new(n, 0)

   if R has AbelianGroup then
     - u             == map(x +-> -x, u)

     n:Integer * u:% == map(x +-> n * x, u)

     u - v           == u + (-v)

   if R has Monoid then
     u:% * r:R       == map(x +-> x * r, u)

     r:R * u:%       == map(x +-> r * x, u)

   if R has Ring then
     dot(u, v) ==
       #u ^= #v => error "Vectors must be of the same length"
       _+/[qelt(u, i) * qelt(v, i) for i in minIndex u .. maxIndex u]

     outerProduct(u, v) ==
       matrix [[qelt(u, i) * qelt(v,j) for i in minIndex u .. maxIndex u] _
               for j in minIndex v .. maxIndex v]

     cross(u, v) ==
       #u ^= 3 or #v ^= 3 => error "Vectors must be of length 3"
       construct [qelt(u, 2)*qelt(v, 3) - qelt(u, 3)*qelt(v, 2) , _
                  qelt(u, 3)*qelt(v, 1) - qelt(u, 1)*qelt(v, 3) , _
                  qelt(u, 1)*qelt(v, 2) - qelt(u, 2)*qelt(v, 1) ]

   if R has RadicalCategory and R has Ring then
     length p ==
        sqrt(dot(p,p))

     magnitude p ==
        sqrt(dot(p,p))
```

———-

— **VECTCAT.dotabb** —

```
"VECTCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=VECTCAT"];
"VECTCAT" -> "A1AGG"
```

———-

— **VECTCAT.dotfull** —

```
"VectorCategory(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=VECTCAT"];
"VectorCategory(a:Type)" -> "OneDimensionalArrayAggregate(a:Type)"

"VectorCategory(a:Ring)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=VECTCAT"];
"VectorCategory(a:Ring)" -> "VectorCategory(a:Type)"
```

———-

— **VECTCAT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"VectorCategory(a:Type)" [color=lightblue];
"VectorCategory(a:Type)" -> "OneDimensionalArrayAggregate(a:Type)"

"OneDimensionalArrayAggregate(a:Type)" [color=lightblue];
"OneDimensionalArrayAggregate(a:Type)" ->
    "FiniteLinearAggregate(a:Type)"

"FiniteLinearAggregate(a:Type)" [color=lightblue];
"FiniteLinearAggregate(a:Type)" -> "LinearAggregate(a:Type)"

"LinearAggregate(a:Type)" [color=lightblue];
"LinearAggregate(a:Type)" -> "IndexedAggregate(b:Integer,a:Type)"
"LinearAggregate(a:Type)" -> "CLAGG..."

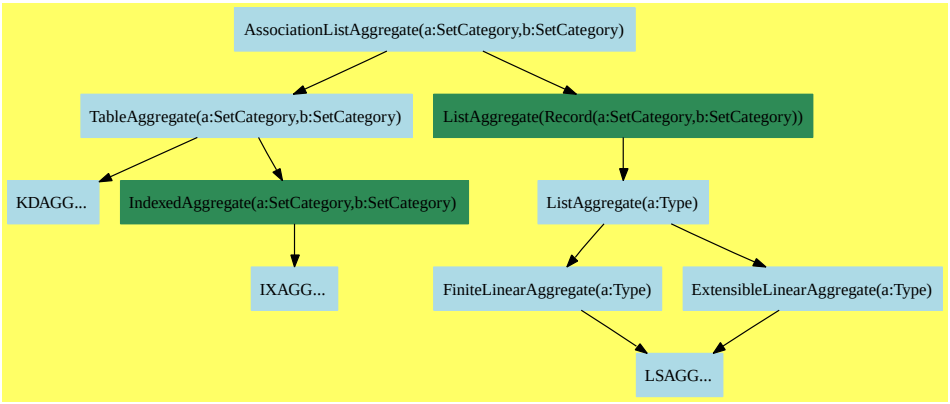
"IndexedAggregate(b:Integer,a:Type)" [color=seagreen];
"IndexedAggregate(b:Integer,a:Type)" -> "IXAGG..."
```

```
"CLAGG..." [color=lightblue];
"IXAGG..." [color=lightblue];
}
```

Chapter 10

Category Layer 9

10.0.134 AssociationListAggregate (ALAGG)



— **AssociationListAggregate.input** —

```
)set break resume
)sys rm -f AssociationListAggregate.output
)spool AssociationListAggregate.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show AssociationListAggregate
--R AssociationListAggregate(Key: SetCategory,Entry: SetCategory)  is a category constructor
--R Abbreviation for AssociationListAggregate is ALAGG
--R This constructor is exposed in this frame.
```

```
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for ALAGG
--R
--R------------------------------ Operations --------------------------------
--R children : % -> List %                concat : (%,%) -> %
--R concat : List % -> %                  concat! : (%,%) -> %
--R copy : % -> %                         cycleEntry : % -> %
--R cycleTail : % -> %                    cyclic? : % -> Boolean
--R delete : (%,Integer) -> %             delete! : (%,Integer) -> %
--R dictionary : () -> %                  distance : (%,%) -> Integer
--R ?.rest : (%,rest) -> %                elt : (%,Key,Entry) -> Entry
--R ?.? : (%,Key) -> Entry                empty : () -> %
--R empty? : % -> Boolean                 entries : % -> List Entry
--R eq? : (%,%) -> Boolean                explicitlyFinite? : % -> Boolean
--R index? : (Integer,%) -> Boolean       index? : (Key,%) -> Boolean
--R indices : % -> List Integer           indices : % -> List Key
--R insert : (%,%,Integer) -> %           insert! : (%,%,Integer) -> %
--R key? : (Key,%) -> Boolean             keys : % -> List Key
--R leaf? : % -> Boolean                  map : ((Entry -> Entry),%) -> %
--R nodes : % -> List %                   possiblyInfinite? : % -> Boolean
--R qelt : (%,Key) -> Entry               rest : % -> %
--R reverse : % -> %                      sample : () -> %
--R setelt : (%,Key,Entry) -> Entry       table : () -> %
--R tail : % -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?<? : (%,%) -> Boolean if Record(key: Key,entry: Entry) has ORDSET
--R ?<=? : (%,%) -> Boolean if Record(key: Key,entry: Entry) has ORDSET
--R ?=? : (%,%) -> Boolean if Record(key: Key,entry: Entry) has SETCAT or Entry has SETCAT or Re
--R ?>? : (%,%) -> Boolean if Record(key: Key,entry: Entry) has ORDSET
--R ?>=? : (%,%) -> Boolean if Record(key: Key,entry: Entry) has ORDSET
--R any? : ((Record(key: Key,entry: Entry) -> Boolean),%) -> Boolean if $ has finiteAggregate
--R any? : ((Entry -> Boolean),%) -> Boolean if $ has finiteAggregate
--R any? : ((Record(key: Key,entry: Entry) -> Boolean),%) -> Boolean if $ has finiteAggregate
--R assoc : (Key,%) -> Union(Record(key: Key,entry: Entry),"failed")
--R bag : List Record(key: Key,entry: Entry) -> %
--R child? : (%,%) -> Boolean if Record(key: Key,entry: Entry) has SETCAT
--R coerce : % -> OutputForm if Record(key: Key,entry: Entry) has SETCAT or Entry has SETCAT or
--R concat : (Record(key: Key,entry: Entry),%) -> %
--R concat : (%,Record(key: Key,entry: Entry)) -> %
--R concat! : (%,Record(key: Key,entry: Entry)) -> %
--R construct : List Record(key: Key,entry: Entry) -> %
--R construct : List Record(key: Key,entry: Entry) -> %
--R convert : % -> InputForm if Record(key: Key,entry: Entry) has KONVERT INFORM or Record(key:
--R copyInto! : (%,%,Integer) -> % if $ has shallowlyMutable
--R count : ((Record(key: Key,entry: Entry) -> Boolean),%) -> NonNegativeInteger if $ has finite
--R count : (Record(key: Key,entry: Entry),%) -> NonNegativeInteger if Record(key: Key,entry: En
--R count : ((Entry -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R count : (Entry,%) -> NonNegativeInteger if Entry has SETCAT and $ has finiteAggregate
--R count : (Record(key: Key,entry: Entry),%) -> NonNegativeInteger if Record(key: Key,entry: En
--R count : ((Record(key: Key,entry: Entry) -> Boolean),%) -> NonNegativeInteger if $ has finite
--R cycleLength : % -> NonNegativeInteger
```

```
--R cycleSplit! : % -> % if $ has shallowlyMutable
--R delete : (%,UniversalSegment Integer) -> %
--R delete! : (%,UniversalSegment Integer) -> %
--R dictionary : List Record(key: Key,entry: Entry) -> %
--R ?.value : (%,value) -> Record(key: Key,entry: Entry)
--R ?.first : (%,first) -> Record(key: Key,entry: Entry)
--R ?.last : (%,last) -> Record(key: Key,entry: Entry)
--R ?.? : (%,UniversalSegment Integer) -> %
--R ?.? : (%,Integer) -> Record(key: Key,entry: Entry)
--R elt : (%,Integer,Record(key: Key,entry: Entry)) -> Record(key: Key,entry: Entry)
--R entries : % -> List Record(key: Key,entry: Entry)
--R entry? : (Record(key: Key,entry: Entry),%) -> Boolean if $ has finiteAggregate and Record(key: Key,entry
--R entry? : (Entry,%) -> Boolean if $ has finiteAggregate and Entry has SETCAT
--R eval : (%,List Equation Record(key: Key,entry: Entry)) -> % if Record(key: Key,entry: Entry) has EVALAB
--R eval : (%,Equation Record(key: Key,entry: Entry)) -> % if Record(key: Key,entry: Entry) has EVALAB Recor
--R eval : (%,Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> % if Record(key: Key,entry: En
--R eval : (%,List Record(key: Key,entry: Entry),List Record(key: Key,entry: Entry)) -> % if Record(key: Key
--R eval : (%,List Equation Entry) -> % if Entry has EVALAB Entry and Entry has SETCAT
--R eval : (%,Equation Entry) -> % if Entry has EVALAB Entry and Entry has SETCAT
--R eval : (%,Entry,Entry) -> % if Entry has EVALAB Entry and Entry has SETCAT
--R eval : (%,List Entry,List Entry) -> % if Entry has EVALAB Entry and Entry has SETCAT
--R eval : (%,List Record(key: Key,entry: Entry),List Record(key: Key,entry: Entry)) -> % if Record(key: Key
--R eval : (%,Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> % if Record(key: Key,entry: En
--R eval : (%,Equation Record(key: Key,entry: Entry)) -> % if Record(key: Key,entry: Entry) has EVALAB Recor
--R eval : (%,List Equation Record(key: Key,entry: Entry)) -> % if Record(key: Key,entry: Entry) has EVALAB
--R every? : ((Record(key: Key,entry: Entry) -> Boolean),%) -> Boolean if $ has finiteAggregate
--R every? : ((Entry -> Boolean),%) -> Boolean if $ has finiteAggregate
--R every? : ((Record(key: Key,entry: Entry) -> Boolean),%) -> Boolean if $ has finiteAggregate
--R extract! : % -> Record(key: Key,entry: Entry)
--R fill! : (%,Record(key: Key,entry: Entry)) -> % if $ has shallowlyMutable
--R fill! : (%,Entry) -> % if $ has shallowlyMutable
--R find : ((Record(key: Key,entry: Entry) -> Boolean),%) -> Union(Record(key: Key,entry: Entry),"failed")
--R find : ((Record(key: Key,entry: Entry) -> Boolean),%) -> Union(Record(key: Key,entry: Entry),"failed")
--R first : % -> Record(key: Key,entry: Entry)
--R first : (%,NonNegativeInteger) -> %
--R first : % -> Entry if Key has ORDSET
--R hash : % -> SingleInteger if Record(key: Key,entry: Entry) has SETCAT or Entry has SETCAT or Record(key
--R insert : (Record(key: Key,entry: Entry),%,Integer) -> %
--R insert! : (Record(key: Key,entry: Entry),%,Integer) -> %
--R insert! : (Record(key: Key,entry: Entry),%) -> %
--R inspect : % -> Record(key: Key,entry: Entry)
--R last : % -> Record(key: Key,entry: Entry)
--R last : (%,NonNegativeInteger) -> %
--R latex : % -> String if Record(key: Key,entry: Entry) has SETCAT or Entry has SETCAT or Record(key: Key,
--R leaves : % -> List Record(key: Key,entry: Entry)
--R less? : (%,NonNegativeInteger) -> Boolean
--R list : Record(key: Key,entry: Entry) -> %
--R map : ((Record(key: Key,entry: Entry) -> Record(key: Key,entry: Entry)),%) -> %
--R map : (((Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> Record(key: Key,entry: Entry))
--R map : (((Entry,Entry) -> Entry),%,%) -> %
```

```
--R map : ((Record(key: Key,entry: Entry) -> Record(key: Key,entry: Entry)),%) -> %
--R map! : ((Record(key: Key,entry: Entry) -> Record(key: Key,entry: Entry)),%) -> % if $ has sh
--R map! : ((Entry -> Entry),%) -> % if $ has shallowlyMutable
--R map! : ((Record(key: Key,entry: Entry) -> Record(key: Key,entry: Entry)),%) -> % if $ has sh
--R max : (%,%) -> % if Record(key: Key,entry: Entry) has ORDSET
--R maxIndex : % -> Integer if Integer has ORDSET
--R maxIndex : % -> Key if Key has ORDSET
--R member? : (Record(key: Key,entry: Entry),%) -> Boolean if Record(key: Key,entry: Entry) has
--R member? : (Entry,%) -> Boolean if Entry has SETCAT and $ has finiteAggregate
--R member? : (Record(key: Key,entry: Entry),%) -> Boolean if Record(key: Key,entry: Entry) has
--R members : % -> List Record(key: Key,entry: Entry) if $ has finiteAggregate
--R members : % -> List Entry if $ has finiteAggregate
--R members : % -> List Record(key: Key,entry: Entry) if $ has finiteAggregate
--R merge : (%,%) -> % if Record(key: Key,entry: Entry) has ORDSET
--R merge : (((Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> Boolean),%,%) ->
--R merge! : (%,%) -> % if Record(key: Key,entry: Entry) has ORDSET
--R merge! : (((Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> Boolean),%,%) ->
--R min : (%,%) -> % if Record(key: Key,entry: Entry) has ORDSET
--R minIndex : % -> Integer if Integer has ORDSET
--R minIndex : % -> Key if Key has ORDSET
--R more? : (%,NonNegativeInteger) -> Boolean
--R new : (NonNegativeInteger,Record(key: Key,entry: Entry)) -> %
--R node? : (%,%) -> Boolean if Record(key: Key,entry: Entry) has SETCAT
--R parts : % -> List Record(key: Key,entry: Entry) if $ has finiteAggregate
--R parts : % -> List Entry if $ has finiteAggregate
--R parts : % -> List Record(key: Key,entry: Entry) if $ has finiteAggregate
--R position : (Record(key: Key,entry: Entry),%,Integer) -> Integer if Record(key: Key,entry: En
--R position : (Record(key: Key,entry: Entry),%) -> Integer if Record(key: Key,entry: Entry) has
--R position : ((Record(key: Key,entry: Entry) -> Boolean),%) -> Integer
--R qelt : (%,Integer) -> Record(key: Key,entry: Entry)
--R qsetelt! : (%,Integer,Record(key: Key,entry: Entry)) -> Record(key: Key,entry: Entry) if $ h
--R qsetelt! : (%,Key,Entry) -> Entry if $ has shallowlyMutable
--R reduce : (((Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> Record(key: Key,
--R reduce : (((Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> Record(key: Key,
--R reduce : (((Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> Record(key: Key,
--R reduce : (((Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> Record(key: Key,
--R reduce : (((Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> Record(key: Key,
--R reduce : (((Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> Record(key: Key,
--R remove : ((Record(key: Key,entry: Entry) -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (Record(key: Key,entry: Entry),%) -> % if Record(key: Key,entry: Entry) has SETCAT
--R remove : ((Record(key: Key,entry: Entry) -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (Record(key: Key,entry: Entry),%) -> % if Record(key: Key,entry: Entry) has SETCAT
--R remove! : (Record(key: Key,entry: Entry),%) -> % if Record(key: Key,entry: Entry) has SETCAT
--R remove! : ((Record(key: Key,entry: Entry) -> Boolean),%) -> %
--R remove! : (Key,%) -> Union(Entry,"failed")
--R remove! : ((Record(key: Key,entry: Entry) -> Boolean),%) -> % if $ has finiteAggregate
--R remove! : (Record(key: Key,entry: Entry),%) -> % if $ has finiteAggregate
--R removeDuplicates : % -> % if Record(key: Key,entry: Entry) has SETCAT and $ has finiteAggreg
--R removeDuplicates! : % -> % if Record(key: Key,entry: Entry) has SETCAT
--R rest : (%,NonNegativeInteger) -> %
```

```
--R reverse! : % -> % if $ has shallowlyMutable
--R search : (Key,%) -> Union(Entry,"failed")
--R second : % -> Record(key: Key,entry: Entry)
--R select : ((Record(key: Key,entry: Entry) -> Boolean),%) -> % if $ has finiteAggregate
--R select : ((Record(key: Key,entry: Entry) -> Boolean),%) -> % if $ has finiteAggregate
--R select! : ((Record(key: Key,entry: Entry) -> Boolean),%) -> %
--R select! : ((Record(key: Key,entry: Entry) -> Boolean),%) -> % if $ has finiteAggregate
--R setchildren! : (%,List %) -> % if $ has shallowlyMutable
--R setelt : (%,value,Record(key: Key,entry: Entry)) -> Record(key: Key,entry: Entry) if $ has shallowlyMuta
--R setelt : (%,first,Record(key: Key,entry: Entry)) -> Record(key: Key,entry: Entry) if $ has shallowlyMuta
--R setelt : (%,rest,%) -> % if $ has shallowlyMutable
--R setelt : (%,last,Record(key: Key,entry: Entry)) -> Record(key: Key,entry: Entry) if $ has shallowlyMutab
--R setelt : (%,UniversalSegment Integer,Record(key: Key,entry: Entry)) -> Record(key: Key,entry: Entry) if
--R setelt : (%,Integer,Record(key: Key,entry: Entry)) -> Record(key: Key,entry: Entry) if $ has shallowlyMu
--R setfirst! : (%,Record(key: Key,entry: Entry)) -> Record(key: Key,entry: Entry) if $ has shallowlyMutable
--R setlast! : (%,Record(key: Key,entry: Entry)) -> Record(key: Key,entry: Entry) if $ has shallowlyMutable
--R setrest! : (%,%) -> % if $ has shallowlyMutable
--R setvalue! : (%,Record(key: Key,entry: Entry)) -> Record(key: Key,entry: Entry) if $ has shallowlyMutable
--R size? : (%,NonNegativeInteger) -> Boolean
--R sort : % -> % if Record(key: Key,entry: Entry) has ORDSET
--R sort : (((Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> Boolean),%) -> %
--R sort! : % -> % if Record(key: Key,entry: Entry) has ORDSET and $ has shallowlyMutable
--R sort! : (((Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> Boolean),%) -> % if $ has sha
--R sorted? : % -> Boolean if Record(key: Key,entry: Entry) has ORDSET
--R sorted? : (((Record(key: Key,entry: Entry),Record(key: Key,entry: Entry)) -> Boolean),%) -> Boolean
--R split! : (%,Integer) -> % if $ has shallowlyMutable
--R swap! : (%,Integer,Integer) -> Void if $ has shallowlyMutable
--R swap! : (%,Key,Key) -> Void if $ has shallowlyMutable
--R table : List Record(key: Key,entry: Entry) -> %
--R third : % -> Record(key: Key,entry: Entry)
--R value : % -> Record(key: Key,entry: Entry)
--R ?~=? : (%,%) -> Boolean if Record(key: Key,entry: Entry) has SETCAT or Entry has SETCAT or Record(key: K
--R
--E 1

)spool
)lisp (bye)
```

———-

— AssociationListAggregate.help —

```
=====================================================================
AssociationListAggregate examples
=====================================================================

See Also:
o )show AssociationListAggregate
```

———-

See:

⇐ "ListAggregate" (LSAGG) 8.0.114 on page 659
⇐ "TableAggregate" (TBAGG) 9.0.132 on page 797

Exports:

any?	assoc	bag	children
child?	coerce	concat	concat!
construct	convert	copy	copyInto!
count	cycleEntry	cycleLength	cycleSplit!
cycleTail	cyclic?	delete	delete!
dictionary	distance	elt	empty
empty?	entries	entry?	eq?
eval	every?	explicitlyFinite?	extract!
fill!	find	first	hash
index?	indices	insert	insert!
inspect	key?	keys	last
latex	leaf?	leaves	less?
list	map	map!	max
maxIndex	member?	members	merge
merge!	min	minIndex	more?
new	nodes	node?	parts
position	possiblyInfinite?	qelt	qsetelt!
reduce	remove	remove!	removeDuplicates
removeDuplicates!	rest	reverse	reverse!
sample	search	second	select
select!	setchildren!	setelt	setfirst!
setlast!	setrest!	setvalue!	size?
sort	sort!	sorted?	split!
swap!	table	tail	third
value	#?	?<?	?<=?
?=?	?>?	?>=?	?~=?
?.rest	?.value	?.first	?.last
?.?			

Attributes exported:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **nil**

These are directly exported but not implemented:

```
assoc : (Key,%) -> Union(Record(key: Key,entry: Entry),"failed")
```

These exports come from (p797) TableAggregate(Key, Entry)
where Key:SetCategory and Entry:SetCategory
and RecKE = Record(key: Key,entry: Entry)

```
any? : ((RecKE -> Boolean),%) -> Boolean
         if $ has finiteAggregate
any? : ((Entry -> Boolean),%) -> Boolean
         if $ has finiteAggregate
any? : ((RecKE -> Boolean),%) -> Boolean
         if $ has finiteAggregate
bag : List RecKE -> %
construct : List RecKE -> %
convert : % -> InputForm
         if RecKE has KONVERT INFORM
         or RecKE has KONVERT INFORM
copy : % -> %
count :
 ((RecKE -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
count : (RecKE,%) -> NonNegativeInteger
         if RecKE has SETCAT
         and $ has finiteAggregate
count : ((Entry -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
count : (Entry,%) -> NonNegativeInteger
         if Entry has SETCAT
         and $ has finiteAggregate
count : (RecKE,%) -> NonNegativeInteger
         if RecKE has SETCAT
         and $ has finiteAggregate
count :
 ((RecKE -> Boolean),%) -> NonNegativeInteger
         if $ has finiteAggregate
dictionary : () -> %
dictionary : List RecKE -> %
elt : (%,Key,Entry) -> Entry
elt : (%,Integer,RecKE) -> RecKE
empty : () -> %
empty? : % -> Boolean
entries : % -> List Entry
entry? : (Entry,%) -> Boolean
         if $ has finiteAggregate
         and Entry has SETCAT
eq? : (%,%) -> Boolean
eval : (%,List Equation RecKE) -> %
         if RecKE has EVALAB RecKE
         and RecKE has SETCAT
eval : (%,Equation RecKE) -> %
```

```
          if RecKE has EVALAB RecKE
          and RecKE has SETCAT
eval : (%,RecKE,RecKE) -> %
          if RecKE has EVALAB RecKE
          and RecKE has SETCAT
eval : (%,List RecKE,List RecKE) -> %
          if RecKE has EVALAB RecKE
          and RecKE has SETCAT
eval : (%,List Equation Entry) -> %
          if Entry has EVALAB Entry
          and Entry has SETCAT
eval : (%,Equation Entry) -> %
          if Entry has EVALAB Entry
          and Entry has SETCAT
eval : (%,Entry,Entry) -> %
          if Entry has EVALAB Entry
          and Entry has SETCAT
eval : (%,List Entry,List Entry) -> %
          if Entry has EVALAB Entry
          and Entry has SETCAT
eval : (%,List RecKE,List RecKE) -> %
          if RecKE has EVALAB RecKE
          and RecKE has SETCAT
eval : (%,RecKE,RecKE) -> %
          if RecKE has EVALAB RecKE
          and RecKE has SETCAT
eval : (%,Equation RecKE) -> %
          if RecKE has EVALAB RecKE
          and RecKE has SETCAT
eval : (%,List Equation RecKE) -> %
          if RecKE has EVALAB RecKE
          and RecKE has SETCAT
every? : ((RecKE -> Boolean),%) -> Boolean
          if $ has finiteAggregate
every? : ((Entry -> Boolean),%) -> Boolean
          if $ has finiteAggregate
extract! : % -> RecKE
fill! : (%,Entry) -> % if $ has shallowlyMutable
find : ((RecKE -> Boolean),%) -> Union(RecKE,"failed")
first : % -> Entry if Key has ORDSET
hash : % -> SingleInteger
          if RecKE has SETCAT
          or Entry has SETCAT
          or RecKE has SETCAT
index? : (Key,%) -> Boolean
indices : % -> List Key
insert! : (RecKE,%) -> %
inspect : % -> RecKE
key? : (Key,%) -> Boolean
keys : % -> List Key
```

```
latex : % -> String
         if RecKE has SETCAT
         or Entry has SETCAT
         or RecKE has SETCAT
less? : (%,NonNegativeInteger) -> Boolean
map : ((Entry -> Entry),%) -> %
map : ((RecKE -> RecKE),%) -> %
map : (((Entry,Entry) -> Entry),%,%) -> %
map! : ((RecKE -> RecKE),%) -> %
         if $ has shallowlyMutable
map! : ((Entry -> Entry),%) -> % if $ has shallowlyMutable
map! : ((RecKE -> RecKE),%) -> %
         if $ has shallowlyMutable
maxIndex : % -> Key if Key has ORDSET
member? : (RecKE,%) -> Boolean
         if RecKE has SETCAT
         and $ has finiteAggregate
member? : (Entry,%) -> Boolean
         if Entry has SETCAT
         and $ has finiteAggregate
members : % -> List RecKE if $ has finiteAggregate
members : % -> List Entry if $ has finiteAggregate
members : % -> List RecKE if $ has finiteAggregate
minIndex : % -> Key if Key has ORDSET
more? : (%,NonNegativeInteger) -> Boolean
parts : % -> List Entry if $ has finiteAggregate
parts : % -> List RecKE if $ has finiteAggregate
qelt : (%,Key) -> Entry
qsetelt! : (%,Key,Entry) -> Entry if $ has shallowlyMutable
reduce :
 (((RecKE,RecKE) -> RecKE),%)
   -> RecKE
         if $ has finiteAggregate
reduce :
 (((RecKE,RecKE) -> RecKE),%,RecKE)
   -> RecKE
         if $ has finiteAggregate
reduce :
 (((RecKE,RecKE) -> RecKE),%,RecKE,RecKE)
   -> RecKE
         if RecKE has SETCAT
         and $ has finiteAggregate
remove : ((RecKE -> Boolean),%) -> % if $ has finiteAggregate
remove : (RecKE,%) -> %
         if RecKE has SETCAT
         and $ has finiteAggregate
remove! : (Key,%) -> Union(Entry,"failed")
remove! : (RecKE,%) -> % if RecKE has SETCAT
remove! : (RecKE,%) -> % if $ has finiteAggregate
removeDuplicates : % -> %
```

```
          if RecKE has SETCAT
          and $ has finiteAggregate
          or RecKE has SETCAT
          and $ has finiteAggregate
 sample : () -> %
 search : (Key,%) -> Union(Entry,"failed")
 select : ((RecKE -> Boolean),%) -> %
          if $ has finiteAggregate
 select! : ((RecKE -> Boolean),%) -> %
          if $ has finiteAggregate
 setelt : (%,Key,Entry) -> Entry
 size? : (%,NonNegativeInteger) -> Boolean
 swap! : (%,Key,Key) -> Void if $ has shallowlyMutable
 table : () -> %
 table : List RecKE -> %
 ?~=? : (%,%) -> Boolean
          if RecKE has SETCAT
          or Entry has SETCAT
          or RecKE has SETCAT
 ?.? : (%,Key) -> Entry
```

These exports come from (p659) ListAggregate(a)
where a is Record(key:Key,entry:Entry)
and RecKE=Record(key: Key,entry: Entry)

```
 children : % -> List %
 child? : (%,%) -> Boolean if RecKE has SETCAT
 coerce : % -> OutputForm
          if RecKE has SETCAT
          or Entry has SETCAT
          or RecKE has SETCAT
 concat : (%,%) -> %
 concat : List % -> %
 concat : (RecKE,%) -> %
 concat : (%,RecKE) -> %
 concat! : (%,%) -> %
 concat! : (%,RecKE) -> %
 copyInto! : (%,%,Integer) -> % if $ has shallowlyMutable
 cycleEntry : % -> %
 cycleLength : % -> NonNegativeInteger
 cycleSplit! : % -> % if $ has shallowlyMutable
 cycleTail : % -> %
 cyclic? : % -> Boolean
 delete : (%,Integer) -> %
 delete! : (%,Integer) -> %
 delete : (%,UniversalSegment Integer) -> %
 delete! : (%,UniversalSegment Integer) -> %
 distance : (%,%) -> Integer
 entries : % -> List RecKE
 entry? : (RecKE,%) -> Boolean
```

```
         if $ has finiteAggregate
         and RecKE has SETCAT
explicitlyFinite? : % -> Boolean
fill! : (%,RecKE) -> % if $ has shallowlyMutable
first : % -> RecKE
first : (%,NonNegativeInteger) -> %
index? : (Integer,%) -> Boolean
indices : % -> List Integer
insert : (%,%,Integer) -> %
insert : (RecKE,%,Integer) -> %
insert! : (%,%,Integer) -> %
insert! : (RecKE,%,Integer) -> %
last : % -> RecKE
last : (%,NonNegativeInteger) -> %
leaf? : % -> Boolean
leaves : % -> List RecKE
list : RecKE -> %
map : (((RecKE,RecKE) -> RecKE),%,%) -> %
max : (%,%) -> % if RecKE has ORDSET
maxIndex : % -> Integer if Integer has ORDSET
merge : (%,%) -> % if RecKE has ORDSET
merge : (((RecKE,RecKE) -> Boolean),%,%) -> %
merge! : (%,%) -> % if RecKE has ORDSET
merge! : (((RecKE,RecKE) -> Boolean),%,%) -> %
min : (%,%) -> % if RecKE has ORDSET
minIndex : % -> Integer if Integer has ORDSET
new : (NonNegativeInteger,RecKE) -> %
nodes : % -> List %
node? : (%,%) -> Boolean if RecKE has SETCAT
position : (RecKE,%,Integer) -> Integer
         if RecKE has SETCAT
position : (RecKE,%) -> Integer
         if RecKE has SETCAT
position : ((RecKE -> Boolean),%) -> Integer
possiblyInfinite? : % -> Boolean
qelt : (%,Integer) -> RecKE
qsetelt! : (%,Integer,RecKE) -> RecKE
         if $ has shallowlyMutable
remove! : ((RecKE -> Boolean),%) -> %
remove! : ((RecKE -> Boolean),%) -> %
         if $ has finiteAggregate
removeDuplicates! : % -> % if RecKE has SETCAT
rest : % -> %
rest : (%,NonNegativeInteger) -> %
reverse : % -> %
reverse! : % -> % if $ has shallowlyMutable
second : % -> RecKE
select! : ((RecKE -> Boolean),%) -> %
setchildren! : (%,List %) -> % if $ has shallowlyMutable
setelt : (%,value,RecKE) -> RecKE
```

```
         if $ has shallowlyMutable
setelt : (%,first,RecKE) -> RecKE
         if $ has shallowlyMutable
setelt : (%,rest,%) -> % if $ has shallowlyMutable
setelt : (%,last,RecKE) -> RecKE
         if $ has shallowlyMutable
setelt : (%,UniversalSegment Integer,RecKE) -> RecKE
         if $ has shallowlyMutable
setelt : (%,Integer,RecKE) -> RecKE
         if $ has shallowlyMutable
setfirst! : (%,RecKE) -> RecKE
         if $ has shallowlyMutable
setlast! : (%,RecKE) -> RecKE
         if $ has shallowlyMutable
setrest! : (%,%) -> % if $ has shallowlyMutable
setvalue! : (%,RecKE) -> RecKE
         if $ has shallowlyMutable
sort : % -> % if RecKE has ORDSET
sort : (((RecKE,RecKE) -> Boolean),%) -> %
sort! : % -> %
         if RecKE has ORDSET
         and $ has shallowlyMutable
sort! : (((RecKE,RecKE) -> Boolean),%) -> %
         if $ has shallowlyMutable
sorted? : % -> Boolean if RecKE has ORDSET
sorted? : (((RecKE,RecKE) -> Boolean),%) -> Boolean
split! : (%,Integer) -> % if $ has shallowlyMutable
swap! : (%,Integer,Integer) -> Void
         if $ has shallowlyMutable
tail : % -> %
third : % -> RecKE
value : % -> RecKE
#? : % -> NonNegativeInteger if $ has finiteAggregate
?<? : (%,%) -> Boolean if RecKE has ORDSET
?<=? : (%,%) -> Boolean if RecKE has ORDSET
?=? : (%,%) -> Boolean
         if RecKE has SETCAT
         or Entry has SETCAT
         or RecKE has SETCAT
?>? : (%,%) -> Boolean if RecKE has ORDSET
?>=? : (%,%) -> Boolean if RecKE has ORDSET
?.value : (%,value) -> RecKE
?.first : (%,first) -> RecKE
?.last : (%,last) -> RecKE
?.rest : (%,rest) -> %
?.? : (%,UniversalSegment Integer) -> %
?.? : (%,Integer) -> RecKE
```

— category ALAGG AssociationListAggregate —

```
)abbrev category ALAGG AssociationListAggregate
++ Author: Michael Monagan; revised by Manuel Bronstein and Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: April 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ An association list is a list of key entry pairs which may be viewed
++ as a table.        It is a poor mans version of a table:
++ searching for a key is a linear operation.

AssociationListAggregate(Key:SetCategory,Entry:SetCategory): Category ==
   Join(TableAggregate(Key, Entry), _
         ListAggregate Record(key:Key,entry:Entry)) with
      assoc: (Key, %) -> Union(Record(key:Key,entry:Entry), "failed")
        ++ assoc(k,u) returns the element x in association list u stored
        ++ with key k, or "failed" if u has no key k.
```

———-

— ALAGG.dotabb —

```
"ALAGG" [color=lightblue,href="bookvol10.2.pdf#nameddest=ALAGG"];
"ALAGG" -> "TBAGG"
"ALAGG" -> "LSAGG"
```

———-

— ALAGG.dotfull —

```
"AssociationListAggregate(a:SetCategory,b:SetCategory)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ALAGG"];
"AssociationListAggregate(a:SetCategory,b:SetCategory)" ->
    "TableAggregate(a:SetCategory,b:SetCategory)"
"AssociationListAggregate(a:SetCategory,b:SetCategory)" ->
    "ListAggregate(Record(a:SetCategory,b:SetCategory))"
```

———-

— ALAGG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"AssociationListAggregate(a:SetCategory,b:SetCategory)" [color=lightblue];
"AssociationListAggregate(a:SetCategory,b:SetCategory)" ->
    "TableAggregate(a:SetCategory,b:SetCategory)"
"AssociationListAggregate(a:SetCategory,b:SetCategory)" ->
    "ListAggregate(Record(a:SetCategory,b:SetCategory))"

"TableAggregate(a:SetCategory,b:SetCategory)" [color=lightblue];
"TableAggregate(a:SetCategory,b:SetCategory)" -> "KDAGG..."
"TableAggregate(a:SetCategory,b:SetCategory)" ->
    "IndexedAggregate(a:SetCategory,b:SetCategory)"

"IndexedAggregate(a:SetCategory,b:SetCategory)" [color=seagreen];
"IndexedAggregate(a:SetCategory,b:SetCategory)" -> "IXAGG..."

"ListAggregate(Record(a:SetCategory,b:SetCategory))" [color=seagreen];
"ListAggregate(Record(a:SetCategory,b:SetCategory))" ->
    "ListAggregate(a:Type)"

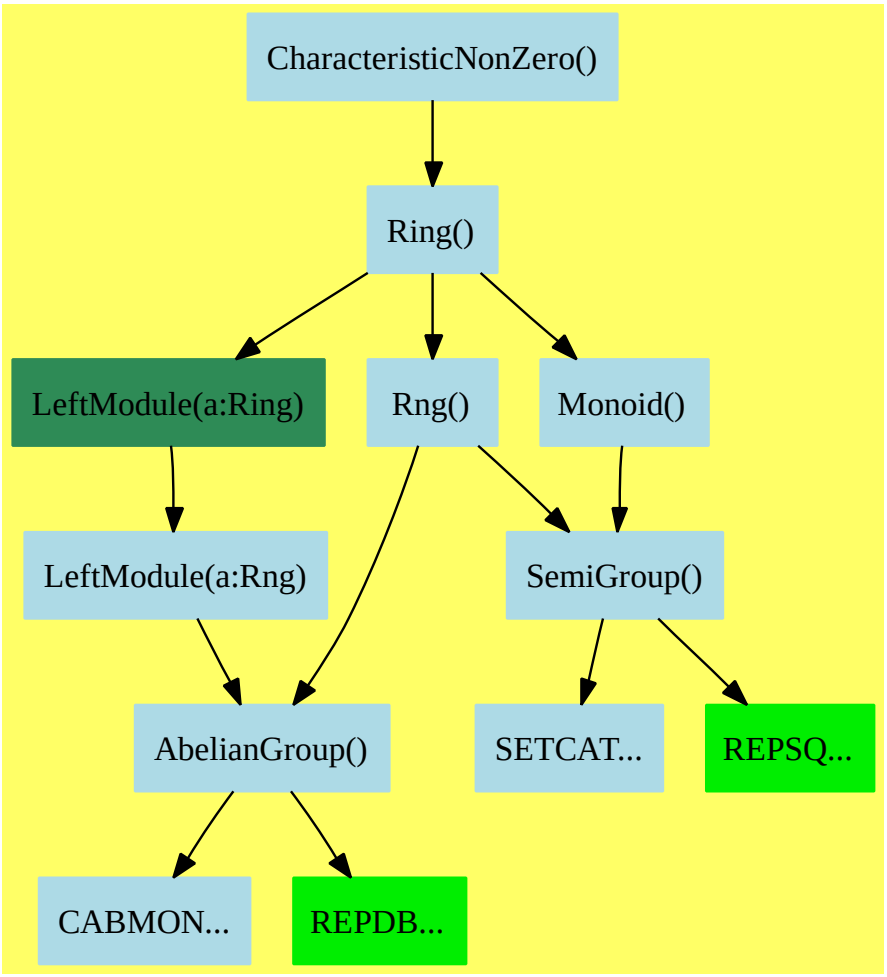
"ListAggregate(a:Type)" [color=lightblue];
"ListAggregate(a:Type)" -> "FiniteLinearAggregate(a:Type)"
"ListAggregate(a:Type)" -> "ExtensibleLinearAggregate(a:Type)"

"FiniteLinearAggregate(a:Type)" [color=lightblue];
"FiniteLinearAggregate(a:Type)" -> "LSAGG..."

"ExtensibleLinearAggregate(a:Type)" [color=lightblue];
"ExtensibleLinearAggregate(a:Type)" -> "LSAGG..."

"KDAGG..." [color=lightblue];
"IXAGG..." [color=lightblue];
"LSAGG..." [color=lightblue];
}
```

10.0.135 CharacteristicNonZero (CHARNZ)



— CharacteristicNonZero.input —

```
)set break resume
)sys rm -f CharacteristicNonZero.output
)spool CharacteristicNonZero.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show CharacteristicNonZero
--R CharacteristicNonZero  is a category constructor
```

```
--R Abbreviation for CharacteristicNonZero is CHARNZ
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for CHARNZ
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        coerce : Integer -> %
--R coerce : % -> OutputForm              hash : % -> SingleInteger
--R latex : % -> String                   one? : % -> Boolean
--R recip : % -> Union(%,"failed")        sample : () -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed")
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— CharacteristicNonZero.help —

```
=====================================================================
CharacteristicNonZero examples
=====================================================================

See Also:
o )show CharacteristicNonZero
```

———-

See:

⇒ “FieldOfPrimeCharacteristic” (FPC) 17.0.193 on page 1323
⇒ “FiniteRankAlgebra” (FINRALG) 17.0.194 on page 1329
⇒ “FunctionSpace” (FS) 17.0.195 on page 1336
⇒ “QuotientFieldCategory” (QFCAT) 17.0.198 on page 1376
⇐ “Ring” (RING) 9.0.129 on page 770

Exports:

1	0	coerce	hash	latex
one?	recip	sample	zero?	characteristic
charthRoot	subtractIfCan	?~=?	?^?	?*?
?**?	?+?	?-?	-?	?=?

Attributes exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.

These are directly exported but not implemented:

```
charthRoot : % -> Union(%,"failed")
```

These exports come from (p770) Ring():

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : Integer -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?~=? : (%,%) -> Boolean
?^? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?*? : (%,%) -> %
?*? : (NonNegativeInteger,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?**? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
?+? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
?=? : (%,%) -> Boolean
```

— category CHARNZ CharacteristicNonZero —

```
)abbrev category CHARNZ CharacteristicNonZero
++ Author:
```

```
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ Rings of Characteristic Non Zero

CharacteristicNonZero():Category == Ring with
    charthRoot: % -> Union(%,"failed")
       ++ charthRoot(x) returns the pth root of x
       ++ where p is the characteristic of the ring.
```

———-

— CHARNZ.dotabb —

```
"CHARNZ"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=CHARNZ"];
"CHARNZ" -> "RING"
```

———-

— CHARNZ.dotfull —

```
"CharacteristicNonZero()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=CHARNZ"];
"CharacteristicNonZero()" -> "Ring()"
```

———-

— CHARNZ.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"CharacteristicNonZero()" [color=lightblue];
"CharacteristicNonZero()" -> "Ring()"

"Ring()" [color=lightblue];
```

```
"Ring()" -> "Rng()"
"Ring()" -> "Monoid()"
"Ring()" -> "LeftModule(a:Ring)"

"Rng()" [color=lightblue];
"Rng()" -> "AbelianGroup()"
"Rng()" -> "SemiGroup()"

"Monoid()" [color=lightblue];
"Monoid()" -> "SemiGroup()"

"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "AbelianGroup()"

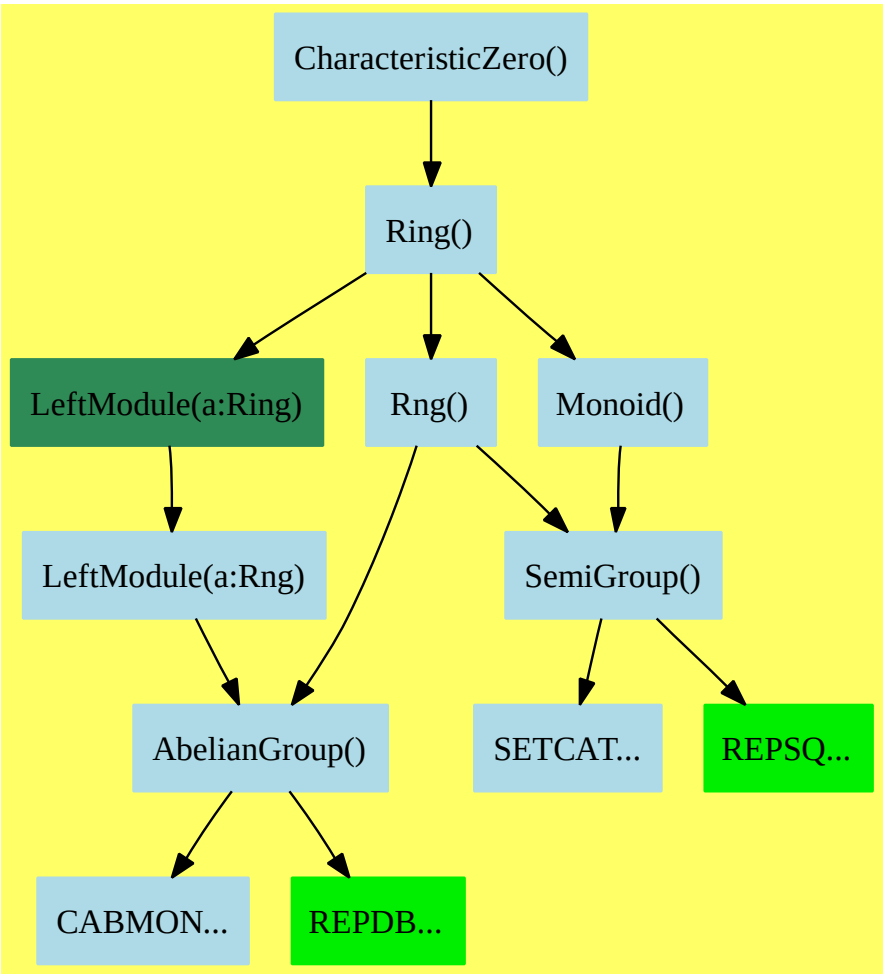
"AbelianGroup()" [color=lightblue];
"AbelianGroup()" -> "CABMON..."
"AbelianGroup()" -> "REPDB..."

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SETCAT..."
"SemiGroup()" -> "REPSQ..."

"REPDB..." [color="#00EE00"];
"REPSQ..." [color="#00EE00"];
"SETCAT..." [color=lightblue];
"CABMON..." [color=lightblue];
}
```

———————-

10.0.136 CharacteristicZero (CHARZ)



— **CharacteristicZero.input** —

```
)set break resume
)sys rm -f CharacteristicZero.output
)spool CharacteristicZero.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show CharacteristicZero
--R CharacteristicZero  is a category constructor
```

```
--R Abbreviation for CharacteristicZero is CHARZ
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for CHARZ
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        coerce : Integer -> %
--R coerce : % -> OutputForm              hash : % -> SingleInteger
--R latex : % -> String                   one? : % -> Boolean
--R recip : % -> Union(%,"failed")        sample : () -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— CharacteristicZero.help —

```
=====================================================================
CharacteristicZero examples
=====================================================================

See Also:
o )show CharacteristicZero
```

———-

See:

⇒ “FiniteRankAlgebra” (FINRALG) 17.0.194 on page 1329
⇒ “FunctionSpace” (FS) 17.0.195 on page 1336
⇒ “IntegerNumberSystem” (INS) 16.0.186 on page 1228
⇒ “PAdicIntegerCategory” (PADICCT) 16.0.188 on page 1249
⇒ “QuotientFieldCategory” (QFCAT) 17.0.198 on page 1376
⇒ “RealClosedField” (RCFIELD) 17.0.199 on page 1390

⇒ "RealNumberSystem" (RNS) 17.0.200 on page 1400
⇐ "Ring" (RING) 9.0.129 on page 770

Exports:

1	0	coerce	hash	latex
one?	recip	sample	zero?	characteristic
subtractIfCan	?~=?	?^?	?*?	
?**?	?+?	?-?	-?	?=?

Attributes exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.

These exports come from (p770) Ring():

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : Integer -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?~=? : (%,%) -> Boolean
?^? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?*? : (%,%) -> %
?*? : (NonNegativeInteger,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?**? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
?+? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
?=? : (%,%) -> Boolean
```

— category CHARZ CharacteristicZero —

```
)abbrev category CHARZ CharacteristicZero
++ Author:
++ Date Created:
++ Date Last Updated:
```

```
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ Rings of Characteristic Zero.

CharacteristicZero():Category == Ring
```

———-

— CHARZ.dotabb —

```
"CHARZ"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=CHARZ"];
"CHARZ" -> "RING"
```

———-

— CHARZ.dotfull —

```
"CharacteristicZero()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=CHARZ"];
"CharacteristicZero()" -> "Ring()"
```

———-

— CHARZ.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"CharacteristicZero()" [color=lightblue];
"CharacteristicZero()" -> "Ring()"

"Ring()" [color=lightblue];
"Ring()" -> "Rng()"
"Ring()" -> "Monoid()"
"Ring()" -> "LeftModule(a:Ring)"

"Rng()" [color=lightblue];
```

```
"Rng()" -> "AbelianGroup()"
"Rng()" -> "SemiGroup()"

"Monoid()" [color=lightblue];
"Monoid()" -> "SemiGroup()"

"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "AbelianGroup()"

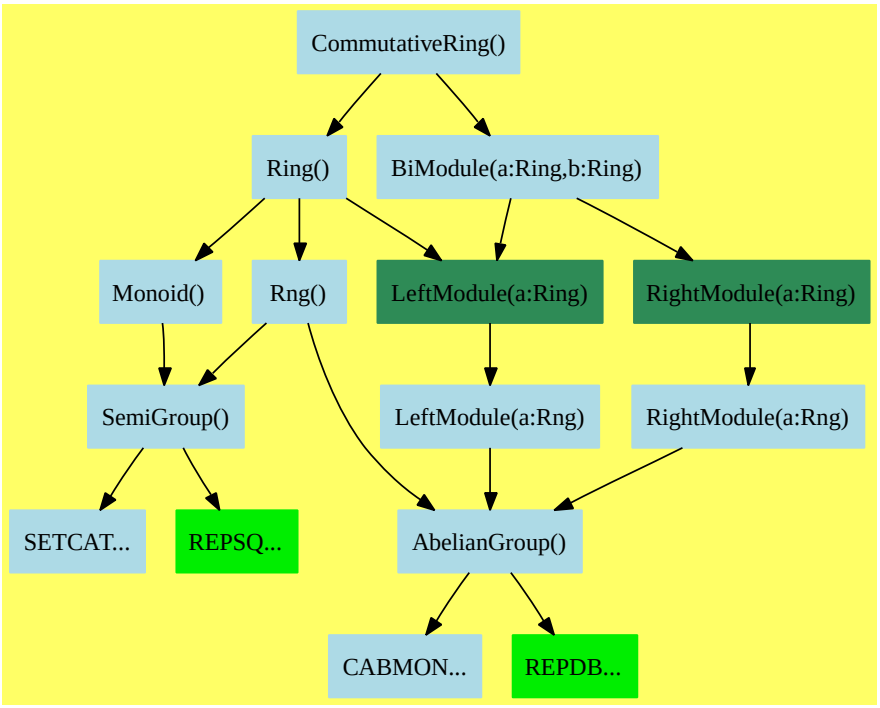
"AbelianGroup()" [color=lightblue];
"AbelianGroup()" -> "CABMON..."
"AbelianGroup()" -> "REPDB..."

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SETCAT..."
"SemiGroup()" -> "REPSQ..."

"REPDB..." [color="#00EE00"];
"REPSQ..." [color="#00EE00"];
"SETCAT..." [color=lightblue];
"CABMON..." [color=lightblue];
}
```

———-

10.0.137 CommutativeRing (COMRING)



— CommutativeRing.input —

```
)set break resume
)sys rm -f CommutativeRing.output
)spool CommutativeRing.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show CommutativeRing
--R CommutativeRing  is a category constructor
--R Abbreviation for CommutativeRing is COMRING
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for COMRING
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
```

```
--R 1 : () -> %                          0 : () -> %
--R ?^? : (%,PositiveInteger) -> %       coerce : Integer -> %
--R coerce : % -> OutputForm             hash : % -> SingleInteger
--R latex : % -> String                  one? : % -> Boolean
--R recip : % -> Union(%,"failed")       sample : () -> %
--R zero? : % -> Boolean                 ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **CommutativeRing.help** —

```
====================================================================
CommutativeRing examples
====================================================================

See Also:
o )show CommutativeRing
```

———-

Commutative Rings are a subset of IntegralDomains.
⇒ “IntegralDomain” (INTDOM) 12.0.164 on page 1047.

See:

⇒ “ComplexCategory” (COMPCAT) 20.0.216 on page 1619
⇒ “IntegralDomain” (INTDOM) 12.0.164 on page 1047
⇒ “FunctionSpace” (FS) 17.0.195 on page 1336
⇒ “MonogenicAlgebra” (MONOGEN) 19.0.214 on page 1598
⇒ “RealClosedField” (RCFIELD) 17.0.199 on page 1390
⇒ “UnivariatePolynomialCategory” (UPOLYC) 17.0.204 on page 1471
⇐ “BiModule” (BMODULE) 9.0.122 on page 726
⇐ “Ring” (RING) 9.0.129 on page 770

Exports:

1	0	coerce	hash	latex
one?	recip	sample	zero?	characteristic
subtractIfCan	?~=?	?^?	?*?	?**?
?+?	?-?	-?	?=?	

Attributes exported:

- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These exports come from (p770) Ring():

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : Integer -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
```

— category COMRING CommutativeRing —

```
)abbrev category COMRING CommutativeRing
++ Author:
```

```
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The category of commutative rings with unity, i.e. rings where
++ \spadop{*} is commutative, and which have a multiplicative identity
++ element.

--CommutativeRing():Category == Join(Ring,BiModule(%:Ring,%:Ring)) with
CommutativeRing():Category == Join(Ring,BiModule(%,%)) with
    commutative("*")  ++ multiplication is commutative.
```

— COMRING.dotabb —

```
"COMRING"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=COMRING"];
"COMRING" -> "RING"
"COMRING" -> "BMODULE"
```

— COMRING.dotfull —

```
"CommutativeRing()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=COMRING"];
"CommutativeRing()" -> "Ring()"
"CommutativeRing()" -> "BiModule(a:Ring,b:Ring)"
```

— COMRING.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"CommutativeRing()" [color=lightblue];
```

```
"CommutativeRing()" -> "Ring()"
"CommutativeRing()" -> "BiModule(a:Ring,b:Ring)"

"Ring()" [color=lightblue];
"Ring()" -> "Rng()"
"Ring()" -> "Monoid()"
"Ring()" -> "LeftModule(a:Ring)"

"BiModule(a:Ring,b:Ring)" [color=lightblue];
"BiModule(a:Ring,b:Ring)" -> "LeftModule(a:Ring)"
"BiModule(a:Ring,b:Ring)" -> "RightModule(a:Ring)"

"RightModule(a:Ring)" [color=seagreen];
"RightModule(a:Ring)" -> "RightModule(a:Rng)"

"RightModule(a:Rng)" [color=lightblue];
"RightModule(a:Rng)" -> "AbelianGroup()"

"Rng()" [color=lightblue];
"Rng()" -> "AbelianGroup()"
"Rng()" -> "SemiGroup()"

"Monoid()" [color=lightblue];
"Monoid()" -> "SemiGroup()"

"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "AbelianGroup()"

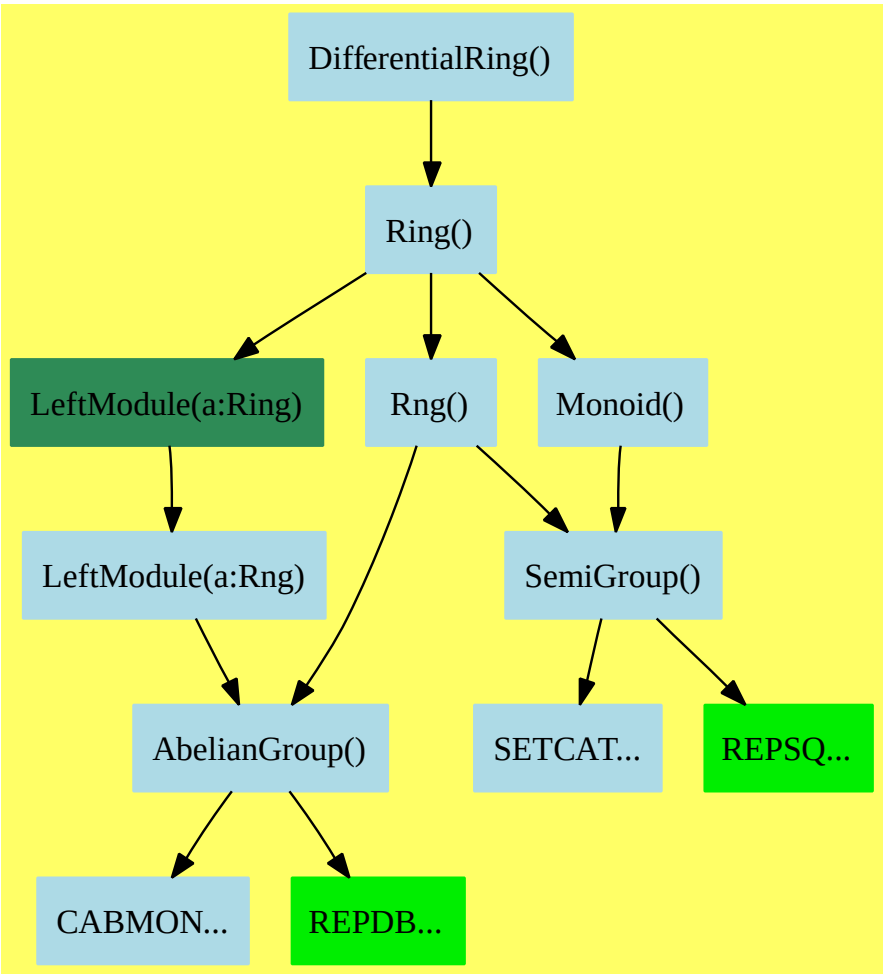
"AbelianGroup()" [color=lightblue];
"AbelianGroup()" -> "CABMON..."
"AbelianGroup()" -> "REPDB..."

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SETCAT..."
"SemiGroup()" -> "REPSQ..."

"REPDB..." [color="#00EE00"];
"REPSQ..." [color="#00EE00"];
"SETCAT..." [color=lightblue];
"CABMON..." [color=lightblue];
}
```

———-

10.0.138 DifferentialRing (DIFRING)



— **DifferentialRing.input** —

```
)set break resume
)sys rm -f DifferentialRing.output
)spool DifferentialRing.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show DifferentialRing
--R DifferentialRing  is a category constructor
```

```
--R Abbreviation for DifferentialRing is DIFRING
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for DIFRING
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R D : (%,NonNegativeInteger) -> %       D : % -> %
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        coerce : Integer -> %
--R coerce : % -> OutputForm              differentiate : % -> %
--R hash : % -> SingleInteger             latex : % -> String
--R one? : % -> Boolean                   recip : % -> Union(%,"failed")
--R sample : () -> %                      zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R differentiate : (%,NonNegativeInteger) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— DifferentialRing.help —

```
=====================================================================
DifferentialRing examples
=====================================================================

See Also:
o )show DifferentialRing
```

———-

See:

⇒ “IntegerNumberSystem” (INS) 16.0.186 on page 1228
⇒ “FiniteFieldCategory” (FFIELDC) 18.0.207 on page 1518
⇒ “UnivariatePolynomialCategory” (UPOLYC) 17.0.204 on page 1471
⇐ “Ring” (RING) 9.0.129 on page 770

Exports:

1	0	characteristic	coerce	D
differentiate	hash	latex	one?	recip
sample	subtractIfCan	zero?	?~=?	?**?
?^?	?*?	?+?	?-?	-?
?=?				

Attributes exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.

These are directly exported but not implemented:

```
differentiate : % -> %
```

These are implemented by this category:

```
D : % -> %
D : (%,NonNegativeInteger) -> %
differentiate : (%,NonNegativeInteger) -> %
```

These exports come from (p770) Ring():

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : Integer -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?~=? : (%,%) -> Boolean
?**? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?*? : (%,%) -> %
?*? : (NonNegativeInteger,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?+? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
?=? : (%,%) -> Boolean
```

— category DIFRING DifferentialRing —

```
)abbrev category DIFRING DifferentialRing
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ An ordinary differential ring, that is, a ring with an operation
++ \spadfun{differentiate}.
++
++ Axioms\br
++ \tab{5}\spad{differentiate(x+y) = differentiate(x)+differentiate(y)}\br
++ \tab{5}\spad{differentiate(x*y) = x*differentiate(y) + differentiate(x)*y}

DifferentialRing(): Category == Ring with
    differentiate: % -> %
         ++ differentiate(x) returns the derivative of x.
         ++ This function is a simple differential operator
         ++ where no variable needs to be specified.
    D: % -> %
         ++ D(x) returns the derivative of x.
         ++ This function is a simple differential operator
         ++ where no variable needs to be specified.
    differentiate: (%, NonNegativeInteger) -> %
         ++ differentiate(x, n) returns the n-th derivative of x.
    D: (%, NonNegativeInteger) -> %
         ++ D(x, n) returns the n-th derivative of x.
  add
    D r == differentiate r
    differentiate(r, n) ==
      for i in 1..n repeat r := differentiate r
      r
    D(r,n) == differentiate(r,n)
```

———-

— DIFRING.dotabb —

```
"DIFRING"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=DIFRING"];
"DIFRING" -> "RING"
```

———-

— DIFRING.dotfull —

```
"DifferentialRing()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=DIFRING"];
"DifferentialRing()" -> "Ring()"
```

———-

— DIFRING.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"DifferentialRing()" [color=lightblue];
"DifferentialRing()" -> "Ring()"

"Ring()" [color=lightblue];
"Ring()" -> "Rng()"
"Ring()" -> "Monoid()"
"Ring()" -> "LeftModule(a:Ring)"

"Rng()" [color=lightblue];
"Rng()" -> "AbelianGroup()"
"Rng()" -> "SemiGroup()"

"Monoid()" [color=lightblue];
"Monoid()" -> "SemiGroup()"

"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "AbelianGroup()"

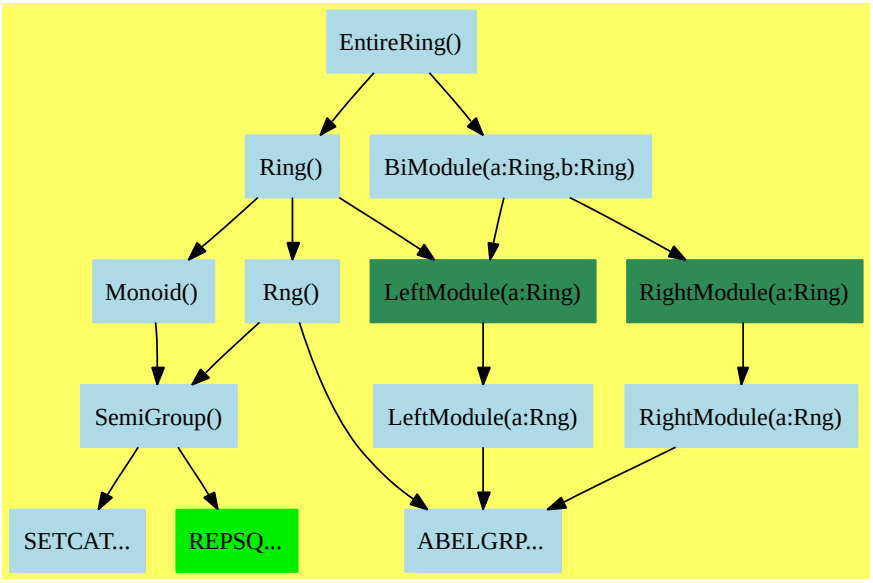
"AbelianGroup()" [color=lightblue];
"AbelianGroup()" -> "CABMON..."
"AbelianGroup()" -> "REPDB..."

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SETCAT..."
"SemiGroup()" -> "REPSQ..."
```

```
"REPDB..." [color="#00EE00"];
"REPSQ..." [color="#00EE00"];
"SETCAT..." [color=lightblue];
"CABMON..." [color=lightblue];
}
```

———-

10.0.139 EntireRing (ENTIRER)



— **EntireRing.input** —

```
)set break resume
)sys rm -f EntireRing.output
)spool EntireRing.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show EntireRing
--R EntireRing  is a category constructor
--R Abbreviation for EntireRing is ENTIRER
```

```
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for ENTIRER
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        coerce : Integer -> %
--R coerce : % -> OutputForm              hash : % -> SingleInteger
--R latex : % -> String                   one? : % -> Boolean
--R recip : % -> Union(%,"failed")        sample : () -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **EntireRing.help** —

```
=====================================================================
EntireRing examples
=====================================================================

See Also:
o )show EntireRing
```

———-

See:

⇒ “DivisionRing” (DIVRING) 12.0.161 on page 1013
⇒ “IntegralDomain” (INTDOM) 12.0.164 on page 1047
⇐ “BiModule” (BMODULE) 9.0.122 on page 726
⇐ “Ring” (RING) 9.0.129 on page 770

Exports:

1	0	characteristic	coerce	hash
latex	one?	recip	sample	subtractIfCan
zero?	?^?	?~=?	?*?	?**?
?+?	?-?	-?	?=?	

Attributes Exported:

- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These exports come from (p770) Ring():

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : Integer -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
```

— category ENTIRER EntireRing —

```
)abbrev category ENTIRER EntireRing
++ Author:
++ Date Created:
```

```
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ Entire Rings (non-commutative Integral Domains), i.e. a ring
++ not necessarily commutative which has no zero divisors.
++
++ Axioms\br
++ \tab{5}\spad{ab=0 => a=0 or b=0}  -- known as noZeroDivisors\br
++ \tab{5}\spad{not(1=0)}

--EntireRing():Category == Join(Ring,BiModule(%:Ring,%:Ring)) with
EntireRing():Category == Join(Ring,BiModule(%,%)) with
      noZeroDivisors  ++ if a product is zero then one of the factors
                      ++ must be zero.
```

———-

— ENTIRER.dotabb —

```
"ENTIRER"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ENTIRER"];
"ENTIRER" -> "RING"
"ENTIRER" -> "BMODULE"
```

———-

— ENTIRER.dotfull —

```
"EntireRing()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ENTIRER"];
"EntireRing()" -> "Ring()"
"EntireRing()" -> "BiModule(a:Ring,b:Ring)"
```

———-

— ENTIRER.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
```

```
 node [shape=box, color=white, style=filled];

"EntireRing()" [color=lightblue];
"EntireRing()" -> "Ring()"
"EntireRing()" -> "BiModule(a:Ring,b:Ring)"

"Ring()" [color=lightblue];
"Ring()" -> "Rng()"
"Ring()" -> "Monoid()"
"Ring()" -> "LeftModule(a:Ring)"

"Rng()" [color=lightblue];
"Rng()" -> "ABELGRP..."
"Rng()" -> "SemiGroup()"

"Monoid()" [color=lightblue];
"Monoid()" -> "SemiGroup()"

"BiModule(a:Ring,b:Ring)" [color=lightblue];
"BiModule(a:Ring,b:Ring)" -> "LeftModule(a:Ring)"
"BiModule(a:Ring,b:Ring)" -> "RightModule(a:Ring)"

"RightModule(a:Ring)" [color=seagreen];
"RightModule(a:Ring)" -> "RightModule(a:Rng)"

"RightModule(a:Rng)" [color=lightblue];
"RightModule(a:Rng)" -> "ABELGRP..."

"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

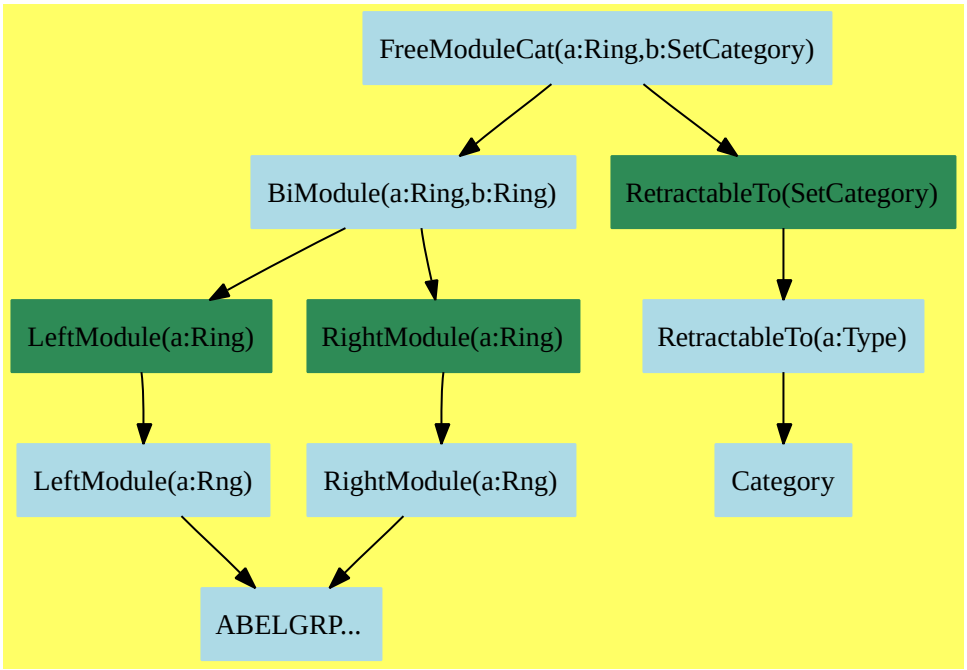
"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "ABELGRP..."

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SETCAT..."
"SemiGroup()" -> "REPSQ..."

"REPSQ..." [color="#00EE00"];
"SETCAT..." [color=lightblue];
"ABELGRP..." [color=lightblue];
}
```

———-

10.0.140 FreeModuleCat (FMCAT)



— FreeModuleCat.input —

```
)set break resume
)sys rm -f FreeModuleCat.output
)spool FreeModuleCat.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FreeModuleCat
--R FreeModuleCat(R: Ring,Basis: SetCategory)  is a category constructor
--R Abbreviation for FreeModuleCat is FMCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FMCAT
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (R,Basis) -> %                 ?*? : (%,R) -> %
--R ?*? : (R,%) -> %                     ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %       ?+? : (%,%) -> %
--R ?-? : (%,%) -> %                     -? : % -> %
--R ?=? : (%,%) -> Boolean               0 : () -> %
```

```
--R coefficient : (%,Basis) -> R          coefficients : % -> List R
--R coerce : Basis -> %                   coerce : % -> OutputForm
--R hash : % -> SingleInteger             latex : % -> String
--R leadingCoefficient : % -> R           leadingMonomial : % -> Basis
--R map : ((R -> R),%) -> %               monom : (Basis,R) -> %
--R monomial? : % -> Boolean              monomials : % -> List %
--R reductum : % -> %                     retract : % -> Basis
--R sample : () -> %                      zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R leadingTerm : % -> Record(k: Basis,c: R)
--R listOfTerms : % -> List Record(k: Basis,c: R)
--R numberOfMonomials : % -> NonNegativeInteger
--R retractIfCan : % -> Union(Basis,"failed")
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **FreeModuleCat.help** —

```
=====================================================================
FreeModuleCat examples
=====================================================================

See Also:
o )show FreeModuleCat
```

———-

See:

⇐ “BiModule” (BMODULE) 9.0.122 on page 726
⇐ “RetractableTo” (RETRACT) 2.0.18 on page 55

Exports:

0	coefficient	coefficients	coerce
hash	latex	leadingCoefficient	leadingMonomial
leadingTerm	listOfTerms	map	monom
monomial?	monomials	numberOfMonomials	reductum
retract	retractIfCan	sample	subtractIfCan
zero?	?~=?	?*?	?+?
?-?	-?	?=?	

Attributes Exported:

- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
coefficient : (%,Basis) -> R
coefficients : % -> List R
leadingCoefficient : % -> R
leadingMonomial : % -> Basis
leadingTerm : % -> Record(k: Basis,c: R)
listOfTerms : % -> List Record(k: Basis,c: R)
map : ((R -> R),%) -> %
monom : (Basis,R) -> %
monomial? : % -> Boolean
monomials : % -> List %
numberOfMonomials : % -> NonNegativeInteger
reductum : % -> %
?*? : (R,Basis) -> %
```

These exports come from (p726) BiModule(R:Ring,R:Ring):

```
0 : () -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?~=? : (%,%) -> Boolean
?*? : (R,%) -> %
?=? : (%,%) -> Boolean
?+? : (%,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?*? : (Integer,%) -> %
?-? : (%,%) -> %
-? : % -> %
?*? : (%,R) -> %
```

These exports come from (p55) RetractableTo(Basis:SetCategory):

```
coerce : Basis -> %
retract : % -> Basis
retractIfCan : % -> Union(Basis,"failed")
```

— category FMCAT FreeModuleCat —

```
)abbrev category FMCAT FreeModuleCat
++ Author: Michel Petitot petitot@lifl.fr
++ Date Created: 91
++ Date Last Updated: 7 Juillet 92
++ Fix History: compilation v 2.1 le 13 dec 98
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A domain of this category
++ implements formal linear combinations
++ of elements from a domain \spad{Basis} with coefficients
++ in a domain \spad{R}. The domain \spad{Basis} needs only
++ to belong to the category \spadtype{SetCategory} and \spad{R}
++ to the category \spadtype{Ring}. Thus the coefficient ring
++ may be non-commutative.
++ See the \spadtype{XDistributedPolynomial} constructor
++ for examples of domains built with the \spadtype{FreeModuleCat}
++ category constructor.

FreeModuleCat(R, Basis):Category == Exports where
   R: Ring
   Basis: SetCategory
   TERM ==> Record(k: Basis, c: R)

   Exports == Join(BiModule(R,R), RetractableTo Basis) with
     "*"                : (R, Basis) -> %
       ++ \spad{r*b} returns the product of \spad{r} by \spad{b}.
     coefficient        : (%, Basis) -> R
       ++ \spad{coefficient(x,b)} returns the coefficient
       ++ of \spad{b} in \spad{x}.
     map                : (R -> R, %) -> %
       ++ \spad{map(fn,u)} maps function \spad{fn} onto the coefficients
       ++  of the non-zero monomials of \spad{u}.
     monom              : (Basis, R) -> %
       ++ \spad{monom(b,r)} returns the element with the single monomial
       ++  \spad{b} and coefficient \spad{r}.
     monomial?          : % -> Boolean
       ++ \spad{monomial?(x)} returns true if \spad{x} contains a single
       ++ monomial.
     listOfTerms        : % -> List TERM
       ++ \spad{listOfTerms(x)} returns a list \spad{lt} of terms with type
       ++ \spad{Record(k: Basis, c: R)} such that \spad{x} equals
       ++ \spad{reduce(+, map(x +-> monom(x.k, x.c), lt))}.
     coefficients       : % -> List R
       ++ \spad{coefficients(x)} returns the list of coefficients of \spad{x}
     monomials          : % -> List %
```

```
      ++ \spad{monomials(x)} returns the list of \spad{r_i*b_i}
      ++ whose sum is \spad{x}.
    numberOfMonomials  : % -> NonNegativeInteger
      ++ \spad{numberOfMonomials(x)} returns the number of monomials
      ++ of \spad{x}.
    leadingMonomial    : % -> Basis
      ++ \spad{leadingMonomial(x)} returns the first element from
      ++ \spad{Basis} which appears in \spad{listOfTerms(x)}.
    leadingCoefficient : % -> R
      ++ \spad{leadingCoefficient(x)} returns the first coefficient
      ++ which appears in \spad{listOfTerms(x)}.
    leadingTerm        : % -> TERM
      ++ \spad{leadingTerm(x)} returns the first term which
      ++ appears in \spad{listOfTerms(x)}.
    reductum           : % -> %
      ++ \spad{reductum(x)} returns \spad{x} minus its leading term.

     -- attributs
    if R has CommutativeRing then Module(R)
```

———-

— FMCAT.dotabb —

```
"FMCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FMCAT"];
"FMCAT" -> "BMODULE"
"FMCAT" -> "RETRACT"
```

———-

— FMCAT.dotfull —

```
"FreeModuleCat(a:Ring,b:SetCategory)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FMCAT"];
"FreeModuleCat(a:Ring,b:SetCategory)" -> "BiModule(a:Ring,b:Ring)"
"FreeModuleCat(a:Ring,b:SetCategory)" -> "RetractableTo(SetCategory)"
```

———-

— FMCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
```

```
 node [shape=box, color=white, style=filled];

"FreeModuleCat(a:Ring,b:SetCategory)" [color=lightblue];
"FreeModuleCat(a:Ring,b:SetCategory)" -> "BiModule(a:Ring,b:Ring)"
"FreeModuleCat(a:Ring,b:SetCategory)" -> "RetractableTo(SetCategory)"

"RetractableTo(SetCategory)" [color=seagreen];
"RetractableTo(SetCategory)" -> "RetractableTo(a:Type)"

"RetractableTo(a:Type)" [color=lightblue];
"RetractableTo(a:Type)" -> "Category"

"Category" [color=lightblue];

"BiModule(a:Ring,b:Ring)" [color=lightblue];
"BiModule(a:Ring,b:Ring)" -> "LeftModule(a:Ring)"
"BiModule(a:Ring,b:Ring)" -> "RightModule(a:Ring)"

"RightModule(a:Ring)" [color=seagreen];
"RightModule(a:Ring)" -> "RightModule(a:Rng)"

"RightModule(a:Rng)" [color=lightblue];
"RightModule(a:Rng)" -> "ABELGRP..."

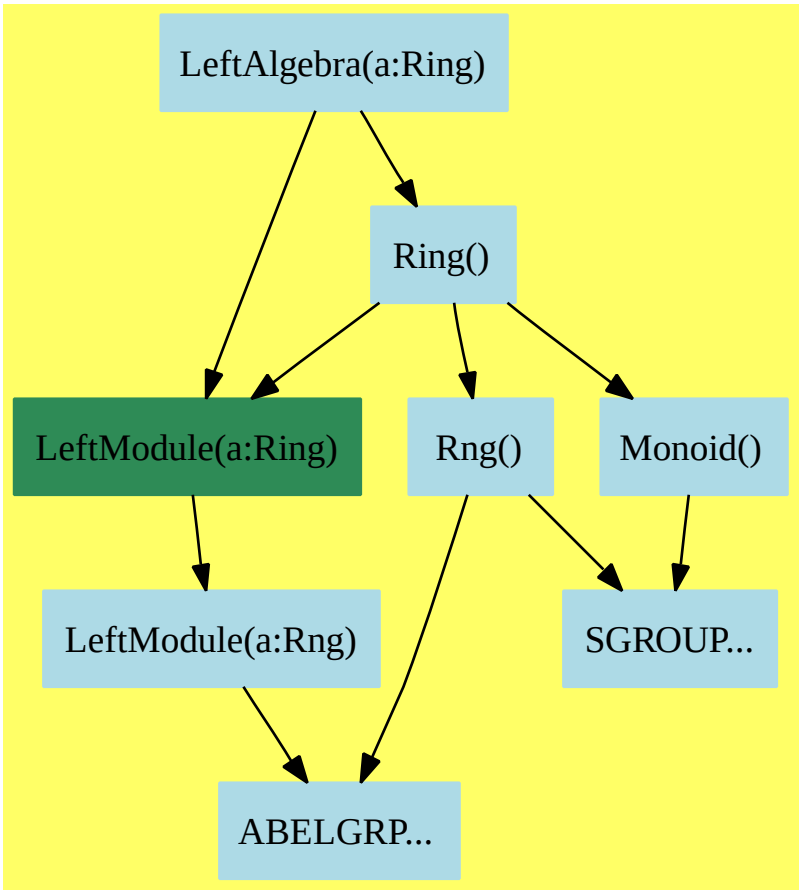
"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "ABELGRP..."

"ABELGRP..." [color=lightblue];
}
```

———-

10.0.141 LeftAlgebra (LALG)



— **LeftAlgebra.input** —

```
)set break resume
)sys rm -f LeftAlgebra.output
)spool LeftAlgebra.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show LeftAlgebra
--R LeftAlgebra R: Ring  is a category constructor
--R Abbreviation for LeftAlgebra is LALG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for LALG
```

```
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (R,%) -> %                       ?*? : (%,%) -> %
--R ?*? : (Integer,%) -> %                 ?*? : (PositiveInteger,%) -> %
--R ?**? : (%,PositiveInteger) -> %        ?+? : (%,%) -> %
--R ?-? : (%,%) -> %                       -? : % -> %
--R ?=? : (%,%) -> Boolean                 1 : () -> %
--R 0 : () -> %                            ?^? : (%,PositiveInteger) -> %
--R coerce : R -> %                        coerce : Integer -> %
--R coerce : % -> OutputForm               hash : % -> SingleInteger
--R latex : % -> String                    one? : % -> Boolean
--R recip : % -> Union(%,"failed")         sample : () -> %
--R zero? : % -> Boolean                   ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **LeftAlgebra.help** —

```
=====================================================================
LeftAlgebra examples
=====================================================================

See Also:
o )show LeftAlgebra
```

———-

See:

⇐ “LeftModule” (LMODULE) 8.0.113 on page 655
⇐ “Ring” (RING) 9.0.129 on page 770

Exports:

0	1	coerce	hash	latex
one?	recip	sample	zero?	characteristic
subtractIfCan	?*?	?+?	?-?	-?
?=?	?~=?	?**?	?^?	

Attributes Exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.

These are implemented by this category:

```
coerce : R -> %
```

These exports come from (p770) Ring():

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : % -> OutputForm
coerce : Integer -> %
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (Integer,%) -> %
?*? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
```

These exports come from (p655) LeftModule(R:Type):

```
?*? : (R,%) -> %
```

— category LALG LeftAlgebra —

```
)abbrev category LALG LeftAlgebra
++ Author: Larry A. Lambe
++ Date  : 03/01/89; revised 03/17/89; revised 12/02/90.
++ Description:
++ The category of all left algebras over an arbitrary ring.
```

```
LeftAlgebra(R:Ring): Category == Join(Ring, LeftModule R) with
      coerce: R -> %
        ++ coerce(r) returns r * 1 where 1 is the identity of the
        ++ left algebra.
    add
      coerce(x:R):% == x * 1$%
```

———-

— **LALG.dotabb** —

```
"LALG"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=LALG"];
"LALG" -> "LMODULE"
"LALG" -> "RING"
```

———-

— **LALG.dotfull** —

```
"LeftAlgebra(a:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=LALG"];
"LeftAlgebra(a:Ring)" -> "LeftModule(a:Ring)"
"LeftAlgebra(a:Ring)" -> "Ring()"
```

———-

— **LALG.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"LeftAlgebra(a:Ring)" [color=lightblue];
"LeftAlgebra(a:Ring)" -> "LeftModule(a:Ring)"
"LeftAlgebra(a:Ring)" -> "Ring()"

"Ring()" [color=lightblue];
"Ring()" -> "Rng()"
"Ring()" -> "Monoid()"
"Ring()" -> "LeftModule(a:Ring)"

"Rng()" [color=lightblue];
```

```
"Rng()" -> "ABELGRP..."
"Rng()" -> "SGROUP..."

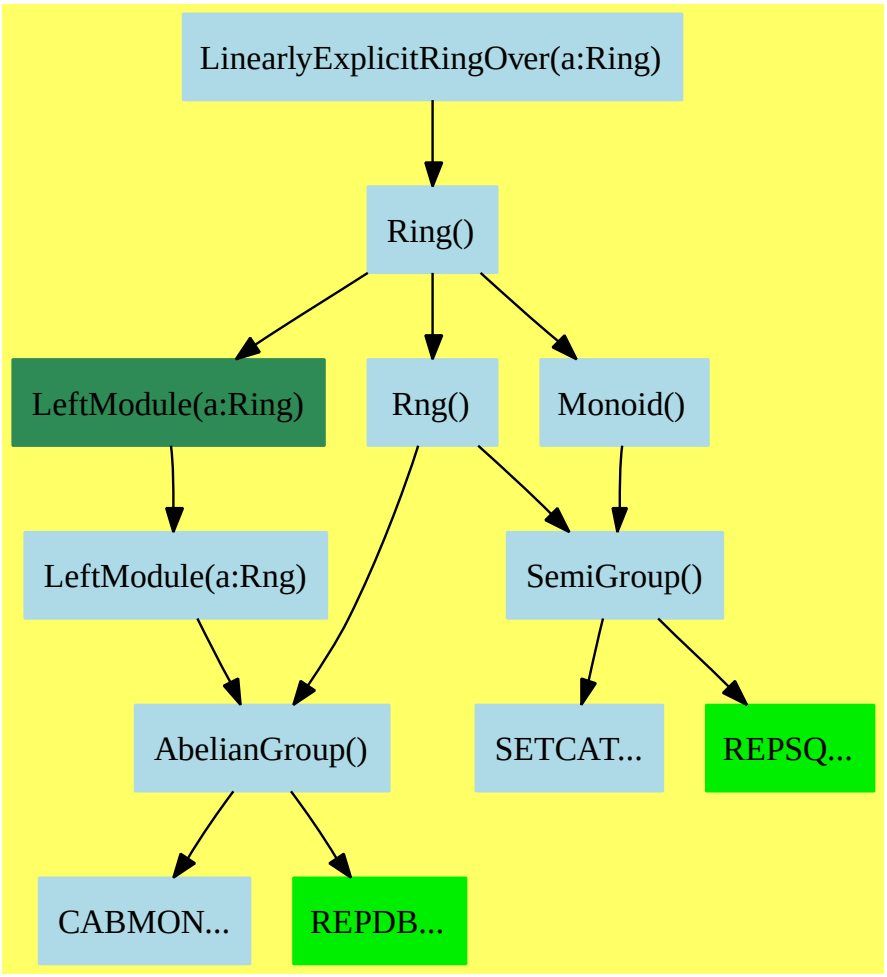
"Monoid()" [color=lightblue];
"Monoid()" -> "SGROUP..."

"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "ABELGRP..."

"SGROUP..." [color=lightblue];
"ABELGRP..." [color=lightblue];
}
```

10.0.142 LinearlyExplicitRingOver (LINEXP)



— **LinearlyExplicitRingOver.input** —

```
)set break resume
)sys rm -f LinearlyExplicitRingOver.output
)spool LinearlyExplicitRingOver.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show LinearlyExplicitRingOver
--R LinearlyExplicitRingOver R: Ring  is a category constructor
```

```
--R Abbreviation for LinearlyExplicitRingOver is LINEXP
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for LINEXP
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        coerce : Integer -> %
--R coerce : % -> OutputForm              hash : % -> SingleInteger
--R latex : % -> String                   one? : % -> Boolean
--R recip : % -> Union(%,"failed")        sample : () -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix R,vec: Vector R)
--R reducedSystem : Matrix % -> Matrix R
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **LinearlyExplicitRingOver.help** —

```
=====================================================================
LinearlyExplicitRingOver examples
=====================================================================

See Also:
o )show LinearlyExplicitRingOver
```

———-

See:

⇒ “FullyLinearlyExplicitRingOver” (FLINEXP) 11.0.154 on page 960
⇒ “IntegerNumberSystem” (INS) 16.0.186 on page 1228
⇒ “MonogenicAlgebra” (MONOGEN) 19.0.214 on page 1598
⇐ “Ring” (RING) 9.0.129 on page 770

Exports:

0	1	characteristic	coerce	hash
latex	one?	recip	reducedSystem	subtractIfCan
sample	zero?	?*?	?**?	?+?
?-?	-?	?=?	?^?	?~=?

Attributes exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return

These are directly exported but not implemented:

```
reducedSystem : (Matrix %,Vector %) ->
    Record(mat: Matrix R,vec: Vector R)
reducedSystem : Matrix % -> Matrix R
```

These exports come from (p770) Ring():

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : Integer -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (Integer,%) -> %
?*? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
```

— category LINEXP LinearlyExplicitRingOver —

```
)abbrev category LINEXP LinearlyExplicitRingOver
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ An extension ring with an explicit linear dependence test.

LinearlyExplicitRingOver(R:Ring): Category == Ring with
  reducedSystem: Matrix % -> Matrix R
    ++ reducedSystem(A) returns a matrix B such that \spad{A x = 0}
    ++ and \spad{B x = 0} have the same solutions in R.
  reducedSystem: (Matrix %,Vector %) -> Record(mat:Matrix R,vec:Vector R)
    ++ reducedSystem(A, v) returns a matrix B and a vector w such that
    ++ \spad{A x = v} and \spad{B x = w} have the same solutions in R.
```

———-

— LINEXP.dotabb —

```
"LINEXP"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=LINEXP"];
"LINEXP" -> "RING"
```

———-

— LINEXP.dotfull —

```
"LinearlyExplicitRingOver(a:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=LINEXP"];
"LinearlyExplicitRingOver(a:Ring)" -> "Ring()"

"LinearlyExplicitRingOver(Integer)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=LINEXP"];
"LinearlyExplicitRingOver(Integer)" -> "LinearlyExplicitRingOver(a:Ring)"

"LinearlyExplicitRingOver(Fraction(Integer))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=LINEXP"];
"LinearlyExplicitRingOver(Fraction(Integer))" ->
    "LinearlyExplicitRingOver(a:Ring)"
```

———-

— LINEXP.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"LinearlyExplicitRingOver(a:Ring)" [color=lightblue];
"LinearlyExplicitRingOver(a:Ring)" -> "Ring()"

"Ring()" [color=lightblue];
"Ring()" -> "Rng()"
"Ring()" -> "Monoid()"
"Ring()" -> "LeftModule(a:Ring)"

"Rng()" [color=lightblue];
"Rng()" -> "AbelianGroup()"
"Rng()" -> "SemiGroup()"

"Monoid()" [color=lightblue];
"Monoid()" -> "SemiGroup()"

"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "AbelianGroup()"

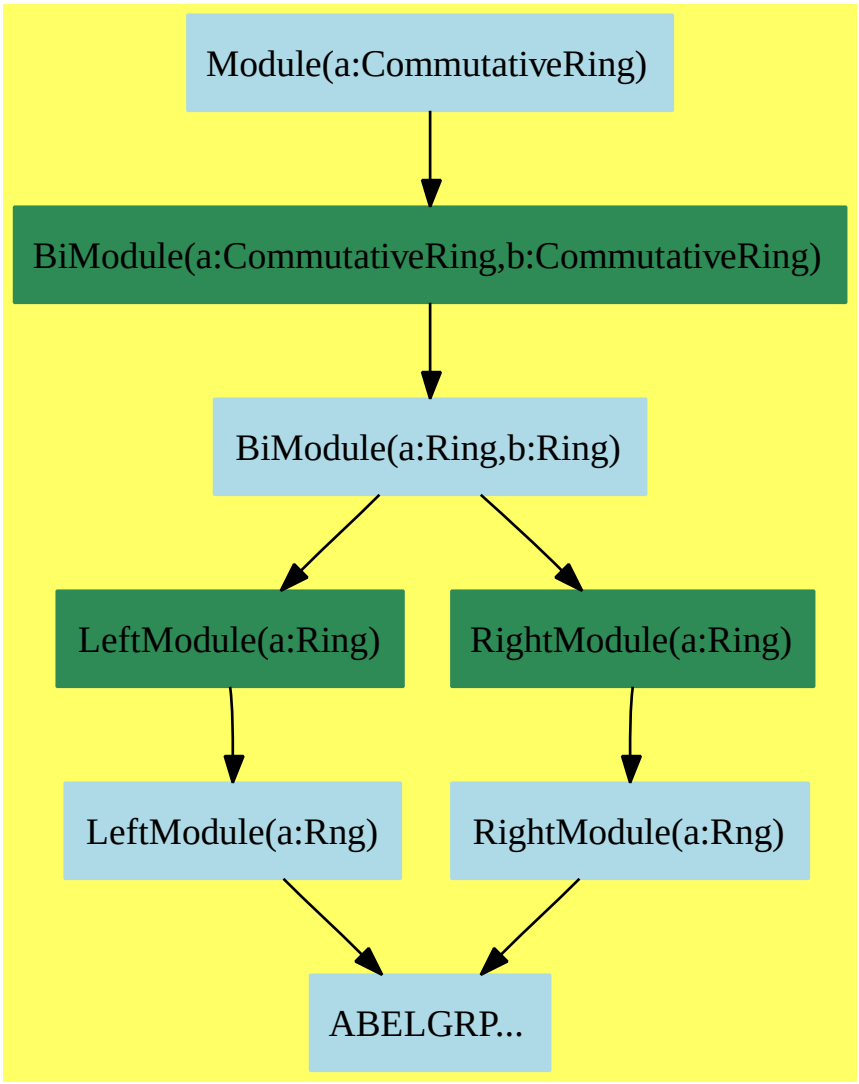
"AbelianGroup()" [color=lightblue];
"AbelianGroup()" -> "CABMON..."
"AbelianGroup()" -> "REPDB..."

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SETCAT..."
"SemiGroup()" -> "REPSQ..."

"REPDB..." [color="#00EE00"];
"REPSQ..." [color="#00EE00"];
"SETCAT..." [color=lightblue];
"CABMON..." [color=lightblue];
}
```

———-

10.0.143 Module (MODULE)



— Module.input —

```
)set break resume
)sys rm -f Module.output
)spool Module.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 1
)show Module
--R Module R: CommutativeRing  is a category constructor
--R Abbreviation for Module is MODULE
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for MODULE
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,R) -> %                      ?*? : (R,%) -> %
--R ?*? : (Integer,%) -> %                ?*? : (PositiveInteger,%) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 0 : () -> %                           coerce : % -> OutputForm
--R hash : % -> SingleInteger             latex : % -> String
--R sample : () -> %                      zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **Module.help** —

```
=====================================================================
Module examples
=====================================================================

See Also:
o )show Module
```

———-

See:

⇒ “Algebra” (ALGEBRA) 11.0.152 on page 947
⇒ “LieAlgebra” (LIECAT) 11.0.155 on page 966
⇒ “NonAssociativeAlgebra” (NAALG) 11.0.157 on page 980
⇒ “RectangularMatrixCategory” (RMATCAT) 10.0.147 on page 901
⇒ “VectorSpace” (VSPACE) 11.0.158 on page 987
⇐ “BiModule” (BMODULE) 9.0.122 on page 726

Exports:

0	coerce	hash	latex	sample
subtractIfCan	zero?	?~=?	?*?	?+?
?-?	-?	?=?		

Attributes exported:

- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are implemented by this category:

```
?*? : (%,R) -> %
```

These exports come from (p726) BiModule(a:Ring,b:Ring):

```
0 : () -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (R,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?+? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
?=? : (%,%) -> Boolean
```

— category MODULE Module —

```
)abbrev category MODULE Module
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The category of modules over a commutative ring.
```

```
++
++ Axioms\br
++ \tab{5}\spad{1*x = x}\br
++ \tab{5}\spad{(a*b)*x = a*(b*x)}\br
++ \tab{5}\spad{(a+b)*x = (a*x)+(b*x)}\br
++ \tab{5}\spad{a*(x+y) = (a*x)+(a*y)}

Module(R:CommutativeRing): Category == BiModule(R,R)
  add
    if not(R is %) then x:%*r:R == r*x
```

———-

— **MODULE.dotabb** —

```
"MODULE"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=MODULE"];
"MODULE" -> "BMODULE"
```

———-

— **MODULE.dotfull** —

```
"Module(a:CommutativeRing)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=MODULE"];
"Module(a:CommutativeRing)" ->
  "BiModule(a:CommutativeRing,b:CommutativeRing)"

"Module(Field)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=MODULE"];
"Module(Field)" -> "Module(a:CommutativeRing)"
```

———-

— **MODULE.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"Module(a:CommutativeRing)" [color=lightblue];
"Module(a:CommutativeRing)" ->
  "BiModule(a:CommutativeRing,b:CommutativeRing)"
```

```
"BiModule(a:CommutativeRing,b:CommutativeRing)" [color=seagreen];
"BiModule(a:CommutativeRing,b:CommutativeRing)" -> "BiModule(a:Ring,b:Ring)"

"BiModule(a:Ring,b:Ring)" [color=lightblue];
"BiModule(a:Ring,b:Ring)" -> "LeftModule(a:Ring)"
"BiModule(a:Ring,b:Ring)" -> "RightModule(a:Ring)"

"RightModule(a:Ring)" [color=seagreen];
"RightModule(a:Ring)" -> "RightModule(a:Rng)"

"RightModule(a:Rng)" [color=lightblue];
"RightModule(a:Rng)" -> "ABELGRP..."

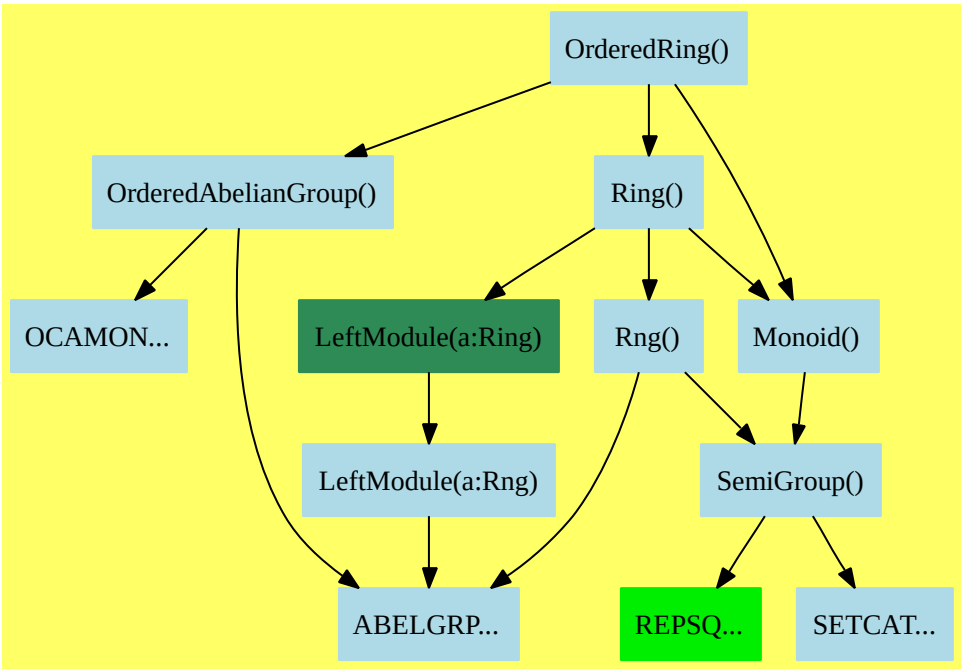
"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "ABELGRP..."

"ABELGRP..." [color=lightblue];
}
```

———-

10.0.144 OrderedRing (ORDRING)



— OrderedRing.input —

```
)set break resume
)sys rm -f OrderedRing.output
)spool OrderedRing.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show OrderedRing
--R OrderedRing  is a category constructor
--R Abbreviation for OrderedRing is ORDRING
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for ORDRING
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,%) -> %                    ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %      ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                    -? : % -> %
--R ?-? : (%,%) -> %                    ?<? : (%,%) -> Boolean
--R ?<=? : (%,%) -> Boolean             ?=? : (%,%) -> Boolean
```

```
--R ?>? : (%,%) -> Boolean                ?>=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        abs : % -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R hash : % -> SingleInteger             latex : % -> String
--R max : (%,%) -> %                      min : (%,%) -> %
--R negative? : % -> Boolean              one? : % -> Boolean
--R positive? : % -> Boolean              recip : % -> Union(%,"failed")
--R sample : () -> %                      sign : % -> Integer
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **OrderedRing.help** —

```
====================================================================
OrderedRing examples
====================================================================

See Also:
o )show OrderedRing
```

———-

See:

⇒ “DirectProductCategory” (DIRPCAT) 12.0.160 on page 1001
⇒ “OrderedIntegralDomain” (OINTDOM) 13.0.174 on page 1135
⇒ “RealClosedField” (RCFIELD) 17.0.199 on page 1390
⇒ “RealNumberSystem” (RNS) 17.0.200 on page 1400
⇐ “Monoid” (MONOID) 5.0.76 on page 331
⇐ “OrderedAbelianGroup” (OAGROUP) 9.0.126 on page 755
⇐ “Ring” (RING) 9.0.129 on page 770

Exports:

1	0	abs	characteristic	coerce
hash	latex	max	min	negative?
one?	positive?	recip	sample	sign
subtractIfCan	zero?	?^?	?~=?	?*?
?**?	?+?	-?	?-?	?<?
?<=?	?=?	?>?	?>=?	

Attributes exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.

These are implemented by this category:

```
abs : % -> %
negative? : % -> Boolean
positive? : % -> Boolean
sign : % -> Integer
```

These exports come from (p755) OrderedAbelianGroup():

```
0 : () -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
max : (%,%) -> %
min : (%,%) -> %
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?<? : (%,%) -> Boolean
?<=? : (%,%) -> Boolean
?=? : (%,%) -> Boolean
?>? : (%,%) -> Boolean
?>=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (Integer,%) -> %
?+? : (%,%) -> %
-? : % -> %
?-? : (%,%) -> %
```

These exports come from (p770) Ring():

```
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : Integer -> %
one? : % -> Boolean
```

```
 recip : % -> Union(%,"failed")
 ?**? : (%,NonNegativeInteger) -> %
 ?**? : (%,PositiveInteger) -> %
 ?*? : (%,%) -> %
 ?^? : (%,NonNegativeInteger) -> %
 ?^? : (%,PositiveInteger) -> %
```

— category ORDRING OrderedRing —

```
)abbrev category ORDRING OrderedRing
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ Ordered sets which are also rings, that is, domains where the ring
++ operations are compatible with the ordering.
++
++ Axiom\br
++ \tab{5}\spad{0<a and b<c => ab< ac}

OrderedRing(): Category == Join(OrderedAbelianGroup,Ring,Monoid) with
     positive?: % -> Boolean
       ++ positive?(x) tests whether x is strictly greater than 0.
     negative?: % -> Boolean
       ++ negative?(x) tests whether x is strictly less than 0.
     sign     : % -> Integer
       ++ sign(x) is 1 if x is positive, -1 if x is negative,
       ++ 0 if x equals 0.
     abs      : % -> %
       ++ abs(x) returns the absolute value of x.
  add
     positive? x == x>0
     negative? x == x<0
     sign x ==
       positive? x => 1
       negative? x => -1
       zero? x => 0
       error "x satisfies neither positive?, negative? or zero?"
     abs x ==
       positive? x => x
       negative? x => -x
       zero? x => 0
       error "x satisfies neither positive?, negative? or zero?"
```

———-

— **ORDRING.dotabb** —

```
"ORDRING"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ORDRING"];
"ORDRING" -> "OAGROUP"
"ORDRING" -> "RING"
"ORDRING" -> "MONOID"
```

———-

— **ORDRING.dotfull** —

```
"OrderedRing()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ORDRING"];
"OrderedRing()" -> "OrderedAbelianGroup()"
"OrderedRing()" -> "Ring()"
"OrderedRing()" -> "Monoid()"
```

———-

— **ORDRING.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"OrderedRing()" [color=lightblue];
"OrderedRing()" -> "OrderedAbelianGroup()"
"OrderedRing()" -> "Ring()"
"OrderedRing()" -> "Monoid()"

"Ring()" [color=lightblue];
"Ring()" -> "Rng()"
"Ring()" -> "Monoid()"
"Ring()" -> "LeftModule(a:Ring)"

"Rng()" [color=lightblue];
"Rng()" -> "ABELGRP..."
"Rng()" -> "SemiGroup()"

"Monoid()" [color=lightblue];
```

```
"Monoid()" -> "SemiGroup()"

"OrderedAbelianGroup()" [color=lightblue];
"OrderedAbelianGroup()" -> "OCAMON..."
"OrderedAbelianGroup()" -> "ABELGRP..."

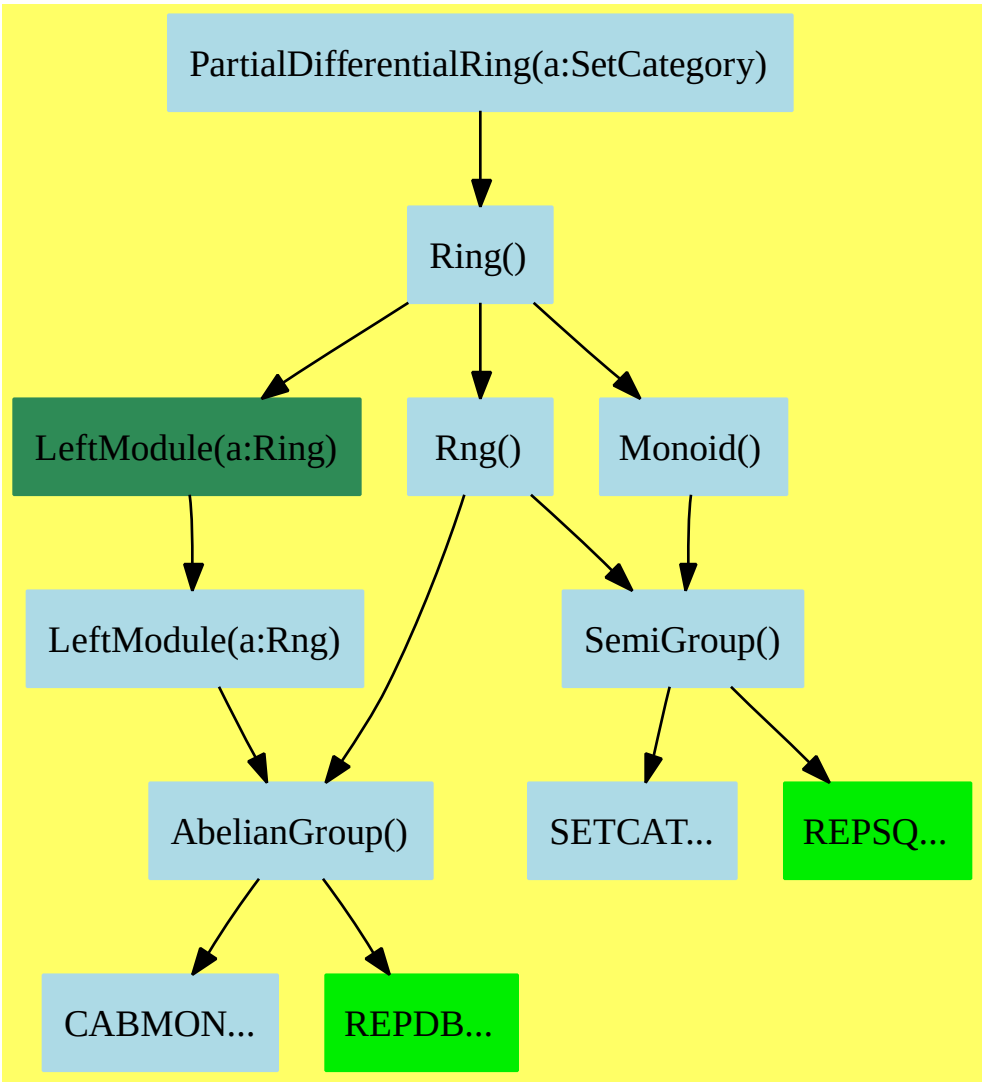
"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "ABELGRP..."

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SETCAT..."
"SemiGroup()" -> "REPSQ..."

"REPSQ..." [color="#00EE00"];
"OCAMON..." [color=lightblue];
"SETCAT..." [color=lightblue];
"ABELGRP..." [color=lightblue];
}
```

10.0.145 PartialDifferentialRing (PDRING)



— **PartialDifferentialRing.input** —

```
)set break resume
)sys rm -f PartialDifferentialRing.output
)spool PartialDifferentialRing.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 1
)show PartialDifferentialRing
--R PartialDifferentialRing S: SetCategory  is a category constructor
--R Abbreviation for PartialDifferentialRing is PDRING
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PDRING
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R D : (%,List S) -> %                   D : (%,S) -> %
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        coerce : Integer -> %
--R coerce : % -> OutputForm              differentiate : (%,List S) -> %
--R differentiate : (%,S) -> %            hash : % -> SingleInteger
--R latex : % -> String                   one? : % -> Boolean
--R recip : % -> Union(%,"failed")        sample : () -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R D : (%,List S,List NonNegativeInteger) -> %
--R D : (%,S,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R differentiate : (%,List S,List NonNegativeInteger) -> %
--R differentiate : (%,S,NonNegativeInteger) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— PartialDifferentialRing.help —

```
=====================================================================
PartialDifferentialRing examples
=====================================================================

See Also:
o )show PartialDifferentialRing
```

———-

See:

⇒ “DifferentialExtension” (DIFEXT) 11.0.153 on page 953
⇒ “FunctionSpace” (FS) 17.0.195 on page 1336
⇒ “MultivariateTaylorSeriesCategory” (MTSCAT) 15.0.182 on page 1193
⇒ “PolynomialCategory” (POLYCAT) 16.0.189 on page 1255
⇐ “Ring” (RING) 9.0.129 on page 770

Exports:

1	0	characteristic	coerce	D
differentiate	hash	latex	one?	recip
sample	subtractIfCan	zero?	?^?	?*?
?~=?	?**?	?+?	?-?	-?
?=?				

Attributes exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return “failed” if its argument is not a unit.

These are directly exported but not implemented:

```
differentiate : (%,S) -> %
```

These are implemented by this category:

```
D : (%,S) -> %
D : (%,List S) -> %
D : (%,S,NonNegativeInteger) -> %
D : (%,List S,List NonNegativeInteger) -> %
differentiate : (%,List S) -> %
differentiate : (%,S,NonNegativeInteger) -> %
differentiate : (%,List S,List NonNegativeInteger) -> %
```

These exports come from (p770) Ring():

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : Integer -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
```

```
 zero? : % -> Boolean
 ?+? : (%,%) -> %
 ?=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
 ?*? : (%,%) -> %
 ?*? : (Integer,%) -> %
 ?*? : (PositiveInteger,%) -> %
 ?*? : (NonNegativeInteger,%) -> %
 ?-? : (%,%) -> %
 -? : % -> %
 ?^? : (%,PositiveInteger) -> %
 ?^? : (%,NonNegativeInteger) -> %
 ?**? : (%,NonNegativeInteger) -> %
 ?**? : (%,PositiveInteger) -> %
```

— category PDRING PartialDifferentialRing —

```
)abbrev category PDRING PartialDifferentialRing
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A partial differential ring with differentiations indexed by a
++ parameter type S.
++
++ Axioms\br
++ \tab{5}\spad{differentiate(x+y,e)=differentiate(x,e)+differentiate(y,e)}\br
++ \tab{5}\spad{differentiate(x*y,e)=x*differentiate(y,e)+differentiate(x,e)*y}

PartialDifferentialRing(S:SetCategory): Category == Ring with
    differentiate: (%, S) -> %
        ++ differentiate(x,v) computes the partial derivative of x
        ++ with respect to v.
    differentiate: (%, List S) -> %
        ++ differentiate(x,[s1,...sn]) computes successive partial
        ++ derivatives,
        ++ i.e. \spad{differentiate(...differentiate(x, s1)..., sn)}.
    differentiate: (%, S, NonNegativeInteger) -> %
        ++ differentiate(x, s, n) computes multiple partial derivatives, i.e.
        ++ n-th derivative of x with respect to s.
    differentiate: (%, List S, List NonNegativeInteger) -> %
        ++ differentiate(x, [s1,...,sn], [n1,...,nn]) computes
        ++ multiple partial derivatives, i.e.
```

```
    D: (%, S) -> %
        ++ D(x,v) computes the partial derivative of x
        ++ with respect to v.
    D: (%, List S) -> %
        ++ D(x,[s1,...sn]) computes successive partial derivatives,
        ++ i.e. \spad{D(...D(x, s1)..., sn)}.
    D: (%, S, NonNegativeInteger) -> %
        ++ D(x, s, n) computes multiple partial derivatives, i.e.
        ++ n-th derivative of x with respect to s.
    D: (%, List S, List NonNegativeInteger) -> %
        ++ D(x, [s1,...,sn], [n1,...,nn]) computes
        ++ multiple partial derivatives, i.e.
        ++ \spad{D(...D(x, s1, n1)..., sn, nn)}.
  add
    differentiate(r:%, l:List S) ==
      for s in l repeat r := differentiate(r, s)
      r

    differentiate(r:%, s:S, n:NonNegativeInteger) ==
      for i in 1..n repeat r := differentiate(r, s)
      r

    differentiate(r:%, ls:List S, ln:List NonNegativeInteger) ==
      for s in ls for n in ln repeat r := differentiate(r, s, n)
      r

    D(r:%, v:S) == differentiate(r,v)
    D(r:%, lv:List S) == differentiate(r,lv)
    D(r:%, v:S, n:NonNegativeInteger) == differentiate(r,v,n)
    D(r:%, lv:List S, ln:List NonNegativeInteger) == differentiate(r, lv, ln)
```

———-

— **PDRING.dotabb** —

```
"PDRING"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PDRING"];
"PDRING" -> "RING"
```

———-

— **PDRING.dotfull** —

```
"PartialDifferentialRing(a:SetCategory)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PDRING"];
"PartialDifferentialRing(a:SetCategory)" -> "Ring()"
```

```
"PartialDifferentialRing(a:OrderedSet)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=PDRING"];
"PartialDifferentialRing(a:OrderedSet)" ->
    "PartialDifferentialRing(a:SetCategory)"

"PartialDifferentialRing(Symbol)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=PDRING"];
"PartialDifferentialRing(Symbol)" ->
    "PartialDifferentialRing(a:SetCategory)"
```

———-

— PDRING.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"PartialDifferentialRing(a:SetCategory)" [color=lightblue];
"PartialDifferentialRing(a:SetCategory)" -> "Ring()"

"Ring()" [color=lightblue];
"Ring()" -> "Rng()"
"Ring()" -> "Monoid()"
"Ring()" -> "LeftModule(a:Ring)"

"Rng()" [color=lightblue];
"Rng()" -> "AbelianGroup()"
"Rng()" -> "SemiGroup()"

"Monoid()" [color=lightblue];
"Monoid()" -> "SemiGroup()"

"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

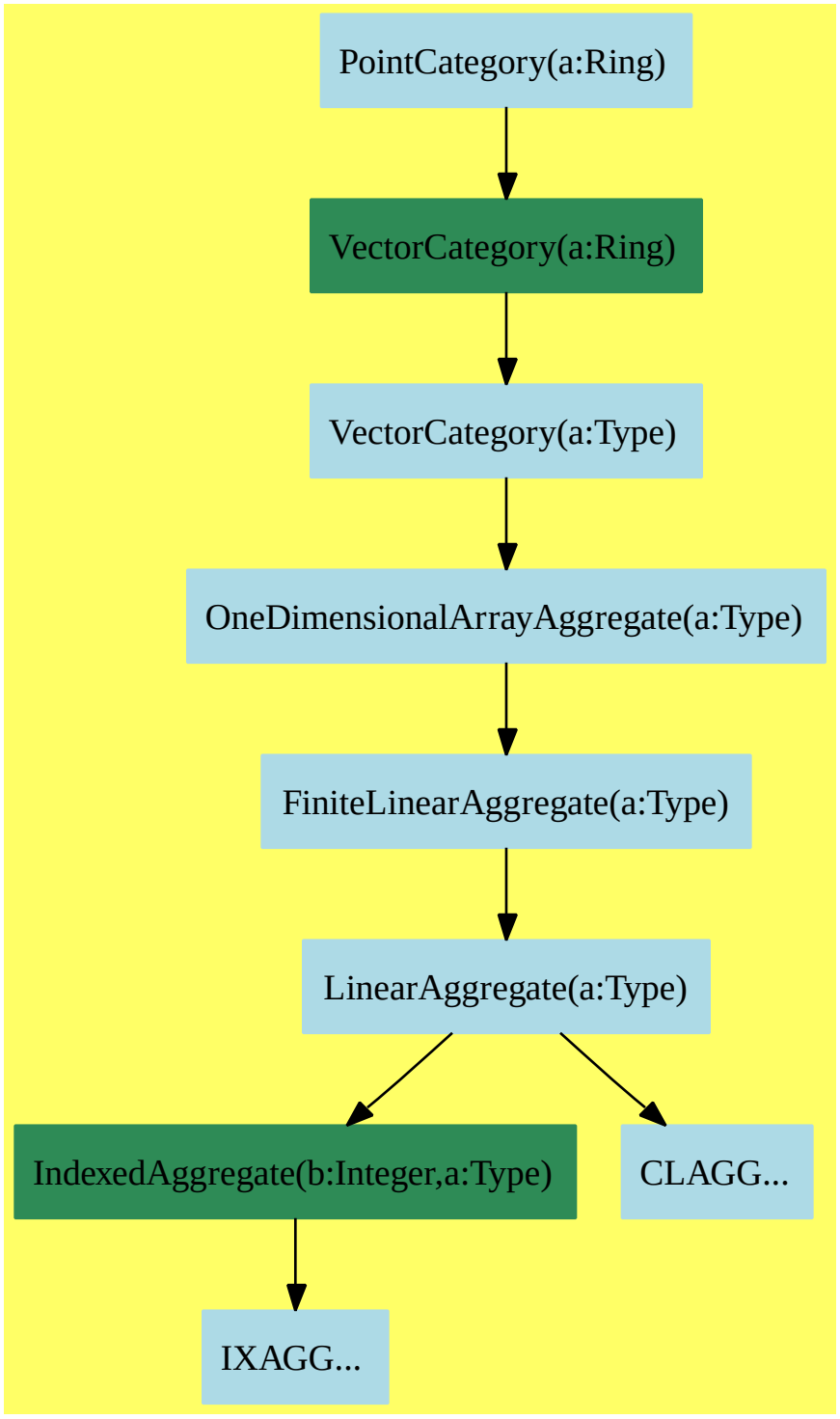
"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "AbelianGroup()"

"AbelianGroup()" [color=lightblue];
"AbelianGroup()" -> "CABMON..."
"AbelianGroup()" -> "REPDB..."

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SETCAT..."
"SemiGroup()" -> "REPSQ..."
```

```
"REPDB..." [color="#00EE00"];
"REPSQ..." [color="#00EE00"];
"SETCAT..." [color=lightblue];
"CABMON..." [color=lightblue];
}
```

10.0.146 PointCategory (PTCAT)



— PointCategory.input —

```
)set break resume
)sys rm -f PointCategory.output
)spool PointCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show PointCategory
--R PointCategory R: Ring  is a category constructor
--R Abbreviation for PointCategory is PTCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PTCAT
--R
--R------------------------------ Operations --------------------------------
--R -? : % -> % if R has ABELGRP          concat : List % -> %
--R concat : (%,%) -> %                   concat : (R,%) -> %
--R concat : (%,R) -> %                   construct : List R -> %
--R convert : List R -> %                 copy : % -> %
--R cross : (%,%) -> %                    delete : (%,Integer) -> %
--R dimension : % -> PositiveInteger      ?.? : (%,Integer) -> R
--R elt : (%,Integer,R) -> R              empty : () -> %
--R empty? : % -> Boolean                 entries : % -> List R
--R eq? : (%,%) -> Boolean                extend : (%,List R) -> %
--R index? : (Integer,%) -> Boolean       indices : % -> List Integer
--R insert : (%,%,Integer) -> %           insert : (R,%,Integer) -> %
--R map : (((R,R) -> R),%,%) -> %         map : ((R -> R),%) -> %
--R new : (NonNegativeInteger,R) -> %     point : List R -> %
--R qelt : (%,Integer) -> R               reverse : % -> %
--R sample : () -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?*? : (%,R) -> % if R has MONOID
--R ?*? : (R,%) -> % if R has MONOID
--R ?*? : (Integer,%) -> % if R has ABELGRP
--R ?+? : (%,%) -> % if R has ABELSG
--R ?-? : (%,%) -> % if R has ABELGRP
--R ?<? : (%,%) -> Boolean if R has ORDSET
--R ?<=? : (%,%) -> Boolean if R has ORDSET
--R ?=? : (%,%) -> Boolean if R has SETCAT
--R ?>? : (%,%) -> Boolean if R has ORDSET
--R ?>=? : (%,%) -> Boolean if R has ORDSET
--R any? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
--R coerce : % -> OutputForm if R has SETCAT
--R convert : % -> InputForm if R has KONVERT INFORM
--R copyInto! : (%,%,Integer) -> % if $ has shallowlyMutable
```

```
--R count : (R,%) -> NonNegativeInteger if R has SETCAT and $ has finiteAggregate
--R count : ((R -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R delete : (%,UniversalSegment Integer) -> %
--R dot : (%,%) -> R if R has RING
--R ?.? : (%,UniversalSegment Integer) -> %
--R entry? : (R,%) -> Boolean if $ has finiteAggregate and R has SETCAT
--R eval : (%,List R,List R) -> % if R has EVALAB R and R has SETCAT
--R eval : (%,R,R) -> % if R has EVALAB R and R has SETCAT
--R eval : (%,Equation R) -> % if R has EVALAB R and R has SETCAT
--R eval : (%,List Equation R) -> % if R has EVALAB R and R has SETCAT
--R every? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
--R fill! : (%,R) -> % if $ has shallowlyMutable
--R find : ((R -> Boolean),%) -> Union(R,"failed")
--R first : % -> R if Integer has ORDSET
--R hash : % -> SingleInteger if R has SETCAT
--R latex : % -> String if R has SETCAT
--R length : % -> R if R has RING and R has RADCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R magnitude : % -> R if R has RING and R has RADCAT
--R map! : ((R -> R),%) -> % if $ has shallowlyMutable
--R max : (%,%) -> % if R has ORDSET
--R maxIndex : % -> Integer if Integer has ORDSET
--R member? : (R,%) -> Boolean if R has SETCAT and $ has finiteAggregate
--R members : % -> List R if $ has finiteAggregate
--R merge : (%,%) -> % if R has ORDSET
--R merge : (((R,R) -> Boolean),%,%) -> %
--R min : (%,%) -> % if R has ORDSET
--R minIndex : % -> Integer if Integer has ORDSET
--R more? : (%,NonNegativeInteger) -> Boolean
--R outerProduct : (%,%) -> Matrix R if R has RING
--R parts : % -> List R if $ has finiteAggregate
--R position : (R,%,Integer) -> Integer if R has SETCAT
--R position : (R,%) -> Integer if R has SETCAT
--R position : ((R -> Boolean),%) -> Integer
--R qsetelt! : (%,Integer,R) -> R if $ has shallowlyMutable
--R reduce : (((R,R) -> R),%) -> R if $ has finiteAggregate
--R reduce : (((R,R) -> R),%,R) -> R if $ has finiteAggregate
--R reduce : (((R,R) -> R),%,R,R) -> R if R has SETCAT and $ has finiteAggregate
--R remove : ((R -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (R,%) -> % if R has SETCAT and $ has finiteAggregate
--R removeDuplicates : % -> % if R has SETCAT and $ has finiteAggregate
--R reverse! : % -> % if $ has shallowlyMutable
--R select : ((R -> Boolean),%) -> % if $ has finiteAggregate
--R setelt : (%,UniversalSegment Integer,R) -> R if $ has shallowlyMutable
--R setelt : (%,Integer,R) -> R if $ has shallowlyMutable
--R size? : (%,NonNegativeInteger) -> Boolean
--R sort : % -> % if R has ORDSET
--R sort : (((R,R) -> Boolean),%) -> %
--R sort! : % -> % if R has ORDSET and $ has shallowlyMutable
--R sort! : (((R,R) -> Boolean),%) -> % if $ has shallowlyMutable
```

```
--R sorted? : % -> Boolean if R has ORDSET
--R sorted? : (((R,R) -> Boolean),%) -> Boolean
--R swap! : (%,Integer,Integer) -> Void if $ has shallowlyMutable
--R zero : NonNegativeInteger -> % if R has ABELMON
--R ?~=? : (%,%) -> Boolean if R has SETCAT
--R
--E 1

)spool
)lisp (bye)
```

———-

— **PointCategory.help** —

```
=====================================================================
PointCategory examples
=====================================================================

See Also:
o )show PointCategory
```

———-

See:

⇐ “VectorCategory” (VECTCAT) 9.0.133 on page 808

Exports:

any?	coerce	concat	construct	convert
copy	copyInto!	cross	count	delete
dimension	dot	elt	empty	empty?
entry?	entries	eq?	eval	every?
extend	fill!	find	first	hash
index?	indices	insert	latex	length
less?	magnitude	map	map!	max
maxIndex	member?	members	merge	min
minIndex	more?	new	outerProduct	parts
point	position	qelt	qsetelt!	reduce
remove	removeDuplicates	reverse	reverse!	sample
select	setelt	size?	sort	sort!
sorted?	swap!	zero	#?	?.?
?~=?	-?	?*?	?+?	?-?
?<?	?<=?	?=?	?>?	?>=?

Attributes Exported:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **nil**

These are directly exported but not implemented:

```
convert : List R -> %
cross : (%,%) -> %
dimension : % -> PositiveInteger
extend : (%,List R) -> %
point : List R -> %
```

These exports come from (p808) VectorCategory(R:Ring):

```
any? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
coerce : % -> OutputForm if R has SETCAT
concat : List % -> %
concat : (%,%) -> %
concat : (R,%) -> %
concat : (%,R) -> %
construct : List R -> %
convert : % -> InputForm if R has KONVERT INFORM
copy : % -> %
copyInto! : (%,%,Integer) -> % if $ has shallowlyMutable
count : (R,%) -> NonNegativeInteger if R has SETCAT and $ has finiteAggregate
count : ((R -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
delete : (%,Integer) -> %
delete : (%,UniversalSegment Integer) -> %
dot : (%,%) -> R if R has RING
elt : (%,Integer,R) -> R
empty : () -> %
empty? : % -> Boolean
entry? : (R,%) -> Boolean if $ has finiteAggregate and R has SETCAT
entries : % -> List R
eq? : (%,%) -> Boolean
eval : (%,List R,List R) -> % if R has EVALAB R and R has SETCAT
eval : (%,R,R) -> % if R has EVALAB R and R has SETCAT
eval : (%,Equation R) -> % if R has EVALAB R and R has SETCAT
eval : (%,List Equation R) -> % if R has EVALAB R and R has SETCAT
every? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
fill! : (%,R) -> % if $ has shallowlyMutable
find : ((R -> Boolean),%) -> Union(R,"failed")
first : % -> R if Integer has ORDSET
hash : % -> SingleInteger if R has SETCAT
index? : (Integer,%) -> Boolean
indices : % -> List Integer
```

```
insert : (%,%,Integer) -> %
insert : (R,%,Integer) -> %
latex : % -> String if R has SETCAT
length : % -> R if R has RING and R has RADCAT
less? : (%,NonNegativeInteger) -> Boolean
magnitude : % -> R if R has RING and R has RADCAT
map : (((R,R) -> R),%,%) -> %
map : ((R -> R),%) -> %
map! : ((R -> R),%) -> % if $ has shallowlyMutable
max : (%,%) -> % if R has ORDSET
maxIndex : % -> Integer if Integer has ORDSET
member? : (R,%) -> Boolean if R has SETCAT and $ has finiteAggregate
members : % -> List R if $ has finiteAggregate
merge : (%,%) -> % if R has ORDSET
merge : (((R,R) -> Boolean),%,%) -> %
min : (%,%) -> % if R has ORDSET
minIndex : % -> Integer if Integer has ORDSET
more? : (%,NonNegativeInteger) -> Boolean
new : (NonNegativeInteger,R) -> %
outerProduct : (%,%) -> Matrix R if R has RING
parts : % -> List R if $ has finiteAggregate
position : (R,%,Integer) -> Integer if R has SETCAT
position : (R,%) -> Integer if R has SETCAT
position : ((R -> Boolean),%) -> Integer
qelt : (%,Integer) -> R
qsetelt! : (%,Integer,R) -> R if $ has shallowlyMutable
reduce : (((R,R) -> R),%) -> R if $ has finiteAggregate
reduce : (((R,R) -> R),%,R) -> R if $ has finiteAggregate
reduce : (((R,R) -> R),%,R,R) -> R if R has SETCAT and $ has finiteAggregate
remove : ((R -> Boolean),%) -> % if $ has finiteAggregate
remove : (R,%) -> % if R has SETCAT and $ has finiteAggregate
removeDuplicates : % -> % if R has SETCAT and $ has finiteAggregate
reverse : % -> %
reverse! : % -> % if $ has shallowlyMutable
sample : () -> %
select : ((R -> Boolean),%) -> % if $ has finiteAggregate
setelt : (%,UniversalSegment Integer,R) -> R if $ has shallowlyMutable
setelt : (%,Integer,R) -> R if $ has shallowlyMutable
size? : (%,NonNegativeInteger) -> Boolean
sort : % -> % if R has ORDSET
sort : (((R,R) -> Boolean),%) -> %
sort! : % -> % if R has ORDSET and $ has shallowlyMutable
sort! : (((R,R) -> Boolean),%) -> % if $ has shallowlyMutable
sorted? : % -> Boolean if R has ORDSET
sorted? : (((R,R) -> Boolean),%) -> Boolean
swap! : (%,Integer,Integer) -> Void if $ has shallowlyMutable
zero : NonNegativeInteger -> % if R has ABELMON
#? : % -> NonNegativeInteger if $ has finiteAggregate
?.? : (%,Integer) -> R
?.? : (%,UniversalSegment Integer) -> %
```

```
 ?~=? : (%,%) -> Boolean if R has SETCAT
 ?<? : (%,%) -> Boolean if R has ORDSET
 ?<=? : (%,%) -> Boolean if R has ORDSET
 ?=? : (%,%) -> Boolean if R has SETCAT
 ?>? : (%,%) -> Boolean if R has ORDSET
 ?>=? : (%,%) -> Boolean if R has ORDSET
 ?*? : (Integer,%) -> % if R has ABELGRP
 ?*? : (%,R) -> % if R has MONOID
 ?*? : (R,%) -> % if R has MONOID
 ?-? : (%,%) -> % if R has ABELGRP
 -? : % -> % if R has ABELGRP
 ?+? : (%,%) -> % if R has ABELSG
```

— category PTCAT PointCategory —

```
)abbrev category PTCAT PointCategory
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Operations: point, elt, setelt, copy, dimension, minIndex, maxIndex,
++ convert
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ PointCategory is the category of points in space which
++ may be plotted via the graphics facilities.  Functions are provided for
++ defining points and handling elements of points.

PointCategory(R:Ring) : Category == VectorCategory(R) with
  point: List R -> %
    ++ point(l) returns a point category defined by a list l of elements from
    ++ the domain R.
  dimension: % -> PositiveInteger
    ++ dimension(s) returns the dimension of the point category s.
  convert: List R -> %
    ++ convert(l) takes a list of elements, l, from the domain Ring and
    ++ returns the form of point category.
  cross: (%,%) -> %
      ++ cross(p,q) computes the cross product of the two points \spad{p}
      ++ and \spad{q}. Error if the p and q are not 3 dimensional
  extend : (%,List R) -> %
        ++ extend(x,l,r) \undocumented
```

———-

— **PTCAT.dotabb** —

```
"PTCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PTCAT"];
"PTCAT" -> "VECTCAT"
```

———-

— **PTCAT.dotfull** —

```
"PointCategory(a:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PTCAT"];
"PointCategory(a:Ring)" -> "VectorCategory(a:Ring)"
```

———-

— **PTCAT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"PointCategory(a:Ring)" [color=lightblue];
"PointCategory(a:Ring)" -> "VectorCategory(a:Ring)"

"VectorCategory(a:Ring)" [color=seagreen];
"VectorCategory(a:Ring)" -> "VectorCategory(a:Type)"

"VectorCategory(a:Type)" [color=lightblue];
"VectorCategory(a:Type)" -> "OneDimensionalArrayAggregate(a:Type)"

"OneDimensionalArrayAggregate(a:Type)" [color=lightblue];
"OneDimensionalArrayAggregate(a:Type)" ->
    "FiniteLinearAggregate(a:Type)"

"FiniteLinearAggregate(a:Type)" [color=lightblue];
"FiniteLinearAggregate(a:Type)" -> "LinearAggregate(a:Type)"

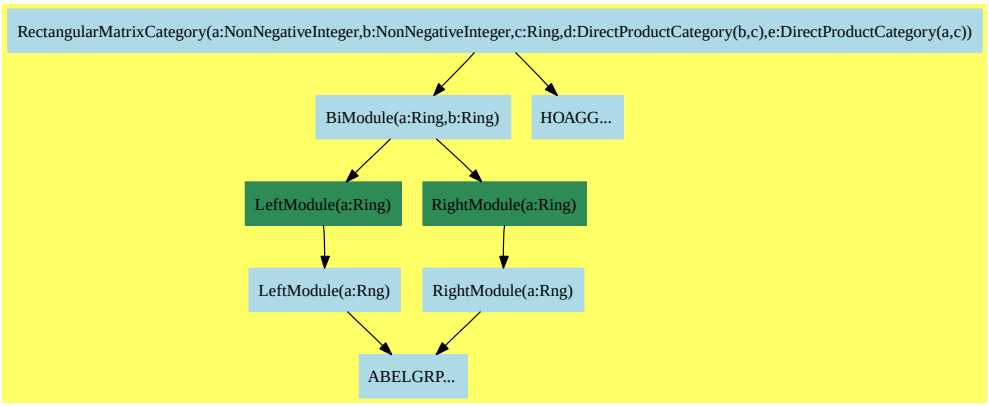
"LinearAggregate(a:Type)" [color=lightblue];
"LinearAggregate(a:Type)" -> "IndexedAggregate(b:Integer,a:Type)"
"LinearAggregate(a:Type)" -> "CLAGG..."

"IndexedAggregate(b:Integer,a:Type)" [color=seagreen];
"IndexedAggregate(b:Integer,a:Type)" -> "IXAGG..."
```

```
"CLAGG..." [color=lightblue];
"IXAGG..." [color=lightblue];
}
```

———-

10.0.147 RectangularMatrixCategory (RMATCAT)



We define three categories for matrices

- MatrixCategory is the category of all matrices
- RectangularMatrixCategory is the category of all matrices of a given dimension
- SquareMatrixCategory inherits from RectangularMatrixCategory

RectangularMatrixCategory does not automatically inherit MatrixCategory. Note that domains in DirectProductCategory(n,R), which are expected as parameters of RectangularMatrixCategory do not satisfy FiniteLinearAggregate(R) as required in MatrixCategory.

The RectangularMatrix domain is matrices of fixed dimension.

— **RectangularMatrixCategory.input** —

```
)set break resume
)sys rm -f RectangularMatrixCategory.output
)spool RectangularMatrixCategory.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 1
)show RectangularMatrixCategory
--R RectangularMatrixCategory(m: NonNegativeInteger,n: NonNegativeInteger,R: Ring,Row: DirectPro
--R Abbreviation for RectangularMatrixCategory is RMATCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for RMATCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,R) -> %                      ?*? : (R,%) -> %
--R ?*? : (Integer,%) -> %                ?*? : (PositiveInteger,%) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 0 : () -> %                           antisymmetric? : % -> Boolean
--R coerce : % -> OutputForm              column : (%,Integer) -> Col
--R copy : % -> %                         diagonal? : % -> Boolean
--R elt : (%,Integer,Integer,R) -> R      elt : (%,Integer,Integer) -> R
--R empty : () -> %                       empty? : % -> Boolean
--R eq? : (%,%) -> Boolean                hash : % -> SingleInteger
--R latex : % -> String                   listOfLists : % -> List List R
--R map : (((R,R) -> R),%,%) -> %         map : ((R -> R),%) -> %
--R matrix : List List R -> %             maxColIndex : % -> Integer
--R maxRowIndex : % -> Integer            minColIndex : % -> Integer
--R minRowIndex : % -> Integer            ncols : % -> NonNegativeInteger
--R nrows : % -> NonNegativeInteger       qelt : (%,Integer,Integer) -> R
--R row : (%,Integer) -> Row              sample : () -> %
--R square? : % -> Boolean                symmetric? : % -> Boolean
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?*? : (NonNegativeInteger,%) -> %
--R ?/? : (%,R) -> % if R has FIELD
--R any? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
--R count : ((R -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R count : (R,%) -> NonNegativeInteger if R has SETCAT and $ has finiteAggregate
--R eval : (%,List Equation R) -> % if R has EVALAB R and R has SETCAT
--R eval : (%,Equation R) -> % if R has EVALAB R and R has SETCAT
--R eval : (%,R,R) -> % if R has EVALAB R and R has SETCAT
--R eval : (%,List R,List R) -> % if R has EVALAB R and R has SETCAT
--R every? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
--R exquo : (%,R) -> Union(%,"failed") if R has INTDOM
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((R -> R),%) -> % if $ has shallowlyMutable
--R member? : (R,%) -> Boolean if R has SETCAT and $ has finiteAggregate
--R members : % -> List R if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
--R nullSpace : % -> List Col if R has INTDOM
--R nullity : % -> NonNegativeInteger if R has INTDOM
--R parts : % -> List R if $ has finiteAggregate
--R rank : % -> NonNegativeInteger if R has INTDOM
--R rowEchelon : % -> % if R has EUCDOM
```

```
--R size? : (%,NonNegativeInteger) -> Boolean
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **RectangularMatrixCategory.help** —

```
=====================================================================
RectangularMatrixCategory examples
=====================================================================

See Also:
o )show RectangularMatrixCategory
```

———-

See:

⇒ “SquareMatrixCategory” (SMATCAT) 12.0.168 on page 1082
⇐ “BiModule” (BMODULE) 9.0.122 on page 726
⇐ “HomogeneousAggregate” (HOAGG) 4.0.49 on page 181
⇐ “MatrixCategory” (MATCAT) 6.0.88 on page 412

Exports:

0	antisymmetric?	any?	coerce	column
copy	count	diagonal?	elt	empty
empty?	eq?	eval	every?	exquo
hash	latex	less?	listOfLists	map
map!	matrix	maxColIndex	maxRowIndex	member?
members	minColIndex	minRowIndex	more?	ncols
nrows	nullSpace	nullity	parts	qelt
rank	row	rowEchelon	sample	size?
square?	subtractIfCan	symmetric?	zero?	#?
?*?	?/?	?+?	?-?	-?
?=?	?~=?			

Attributes Exported:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **leftUnitary** is true if $1 * x = x$ for all x.

- **rightUnitary** is true if $x * 1 = x$ for all x.
- **nil**

These are directly exported but not implemented:

```
column : (%,Integer) -> Col
elt : (%,Integer,Integer) -> R
elt : (%,Integer,Integer,R) -> R
exquo : (%,R) -> Union(%,"failed") if R has INTDOM
listOfLists : % -> List List R
map : (((R,R) -> R),%,%) -> %
map : ((R -> R),%) -> %
maxColIndex : % -> Integer
maxRowIndex : % -> Integer
matrix : List List R -> %
minColIndex : % -> Integer
minRowIndex : % -> Integer
nullity : % -> NonNegativeInteger if R has INTDOM
nullSpace : % -> List Col if R has INTDOM
qelt : (%,Integer,Integer) -> R
rank : % -> NonNegativeInteger if R has INTDOM
row : (%,Integer) -> Row
rowEchelon : % -> % if R has EUCDOM
?/? : (%,R) -> % if R has FIELD
```

These are implemented by this category:

```
antisymmetric? : % -> Boolean
diagonal? : % -> Boolean
ncols : % -> NonNegativeInteger
nrows : % -> NonNegativeInteger
square? : % -> Boolean
symmetric? : % -> Boolean
```

These exports come from (p726) BiModule(a:Ring,b:Ring)

```
0 : () -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?~=? : (%,%) -> Boolean
?*? : (R,%) -> %
?*? : (%,R) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
```

```
?*? : (Integer,%) -> %
?=? : (%,%) -> Boolean
?+? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
```

These exports come from (p181) HomogeneousAggregate(Ring)"

```
any? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
copy : % -> %
count : ((R -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
count : (R,%) -> NonNegativeInteger if R has SETCAT and $ has finiteAggregate
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List Equation R) -> % if R has EVALAB R and R has SETCAT
eval : (%,Equation R) -> % if R has EVALAB R and R has SETCAT
eval : (%,R,R) -> % if R has EVALAB R and R has SETCAT
eval : (%,List R,List R) -> % if R has EVALAB R and R has SETCAT
every? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
less? : (%,NonNegativeInteger) -> Boolean
map! : ((R -> R),%) -> % if $ has shallowlyMutable
members : % -> List R if $ has finiteAggregate
member? : (R,%) -> Boolean if R has SETCAT and $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
parts : % -> List R if $ has finiteAggregate
size? : (%,NonNegativeInteger) -> Boolean
#? : % -> NonNegativeInteger if $ has finiteAggregate
```

— category RMATCAT RectangularMatrixCategory —

```
)abbrev category RMATCAT RectangularMatrixCategory
++ Authors: Grabmeier, Gschnitzer, Williamson
++ Date Created: 1987
++ Date Last Updated: July 1990
++ Basic Operations:
++ Related Domains: RectangularMatrix(m,n,R)
++ Also See:
++ AMS Classifications:
++ Keywords:
++ Examples:
++ References:
++ Description:
++ \spadtype{RectangularMatrixCategory} is a category of matrices of fixed
++ dimensions. The dimensions of the matrix will be parameters of the
++ domain. Domains in this category will be R-modules and will be non-mutable.

RectangularMatrixCategory(m,n,R,Row,Col): Category == Definition where
  m,n : NonNegativeInteger
```

```
  R   : Ring
  Row : DirectProductCategory(n,R)
  Col : DirectProductCategory(m,R)

  Definition ==> Join(BiModule(R,R),HomogeneousAggregate(R)) with

    finiteAggregate
      ++ matrices are finite

    if R has CommutativeRing then Module(R)

--% Matrix creation

    matrix: List List R -> %
      ++ \spad{matrix(l)} converts the list of lists l to a matrix, where the
      ++ list of lists is viewed as a list of the rows of the matrix.

--% Predicates

    square?  : % -> Boolean
      ++ \spad{square?(m)} returns true if m is a square matrix (i.e. if m
      ++ has the same number of rows as columns) and false otherwise.
    diagonal?: % -> Boolean
      ++ \spad{diagonal?(m)} returns true if the matrix m is square and
      ++ diagonal (i.e. all entries of m not on the diagonal are zero) and
      ++ false otherwise.
    symmetric?: % -> Boolean
      ++ \spad{symmetric?(m)} returns true if the matrix m is square and
      ++ symmetric (i.e. \spad{m[i,j] = m[j,i]} for all \spad{i} and j) and
      ++ false otherwise.
    antisymmetric?: % -> Boolean
      ++ \spad{antisymmetric?(m)} returns true if the matrix m is square and
      ++ antisymmetric (i.e. \spad{m[i,j] = -m[j,i]} for all \spad{i} and j)
      ++ and false otherwise.

--% Size inquiries

    minRowIndex : % -> Integer
      ++ \spad{minRowIndex(m)} returns the index of the 'first' row of the
      ++ matrix m.
    maxRowIndex : % -> Integer
      ++ \spad{maxRowIndex(m)} returns the index of the 'last' row of the
      ++ matrix m.
    minColIndex : % -> Integer
      ++ \spad{minColIndex(m)} returns the index of the 'first' column of the
      ++ matrix m.
    maxColIndex : % -> Integer
      ++ \spad{maxColIndex(m)} returns the index of the 'last' column of the
      ++ matrix m.
    nrows : % -> NonNegativeInteger
```

```
      ++ \spad{nrows(m)} returns the number of rows in the matrix m.
    ncols : % -> NonNegativeInteger
      ++ \spad{ncols(m)} returns the number of columns in the matrix m.

--% Part extractions

    listOfLists: % -> List List R
      ++ \spad{listOfLists(m)} returns the rows of the matrix m as a list
      ++ of lists.
    elt: (%,Integer,Integer) -> R
      ++ \spad{elt(m,i,j)} returns the element in the \spad{i}th row and
      ++ \spad{j}th column of the matrix m.
      ++ Error: if indices are outside the proper
      ++ ranges.
    qelt: (%,Integer,Integer) -> R
      ++ \spad{qelt(m,i,j)} returns the element in the \spad{i}th row and
      ++ \spad{j}th column of
      ++ the matrix m. Note that there is NO error check to determine
      ++ if indices are in the proper ranges.
    elt: (%,Integer,Integer,R) -> R
      ++ \spad{elt(m,i,j,r)} returns the element in the \spad{i}th row and
      ++ \spad{j}th column of the matrix m, if m has an \spad{i}th row and a
      ++ \spad{j}th column, and returns r otherwise.
    row: (%,Integer) -> Row
      ++ \spad{row(m,i)} returns the \spad{i}th row of the matrix m.
      ++ Error: if the index is outside the proper range.
    column: (%,Integer) -> Col
      ++ \spad{column(m,j)} returns the \spad{j}th column of the matrix m.
      ++ Error: if the index outside the proper range.

--% Map and Zip

    map: (R -> R,%) -> %
      ++ \spad{map(f,a)} returns b, where \spad{b(i,j) = a(i,j)} for all i, j.
    map:((R,R) -> R,%,%) -> %
      ++ \spad{map(f,a,b)} returns c, where c is such that
      ++ \spad{c(i,j) = f(a(i,j),b(i,j))} for all \spad{i}, j.

--% Arithmetic

    if R has IntegralDomain then
      "exquo": (%,R) -> Union(%,"failed")
        ++ \spad{exquo(m,r)} computes the exact quotient of the elements
        ++ of m by r, returning \axiom{"failed"} if this is not possible.
    if R has Field then
      "/": (%,R) -> %
        ++ \spad{m/r} divides the elements of m by r. Error: if \spad{r = 0}.

--% Linear algebra
```

```
 if R has EuclideanDomain then
   rowEchelon: % -> %
     ++ \spad{rowEchelon(m)} returns the row echelon form of the matrix m.
 if R has IntegralDomain then
   rank: % -> NonNegativeInteger
     ++ \spad{rank(m)} returns the rank of the matrix m.
   nullity: % -> NonNegativeInteger
     ++ \spad{nullity(m)} returns the nullity of the matrix m. This is
     ++ the dimension of the null space of the matrix m.
   nullSpace: % -> List Col
     ++ \spad{nullSpace(m)}+ returns a basis for the null space of
     ++ the matrix m.
add
  nrows x == m

  ncols x == n

  square? x == m = n

  diagonal? x ==
    not square? x => false
    for i in minRowIndex x .. maxRowIndex x repeat
      for j in minColIndex x .. maxColIndex x
        | (j - minColIndex x) ^= (i - minRowIndex x) repeat
          not zero? qelt(x, i, j) => return false
    true

  symmetric? x ==
    m ^= n => false
    mr := minRowIndex x; mc := minColIndex x
    for i in 0..(n - 1) repeat
      for j in (i + 1)..(n - 1) repeat
        qelt(x,mr + i,mc + j) ^= qelt(x,mr + j,mc + i) => return false
    true

  antisymmetric? x ==
    m ^= n => false
    mr := minRowIndex x; mc := minColIndex x
    for i in 0..(n - 1) repeat
      for j in i..(n - 1) repeat
        qelt(x,mr + i,mc + j) ^= -qelt(x,mr + j,mc + i) =>
          return false
    true
```

———-

— **RMATCAT.dotabb** —

```
"RMATCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RMATCAT"];
"RMATCAT" -> "BMODULE"
"RMATCAT" -> "HOAGG"
```

———-

— RMATCAT.dotfull —

```
"RectangularMatrixCategory(a:NonNegativeInteger,b:NonNegativeInteger,c:Ring,d:DirectProductCategory(b,c),e:I
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RMATCAT"];
"RectangularMatrixCategory(a:NonNegativeInteger,b:NonNegativeInteger,c:Ring,d:DirectProductCategory(b,c),e:I
  -> "BiModule(a:Ring,b:Ring)"
"RectangularMatrixCategory(a:NonNegativeInteger,b:NonNegativeInteger,c:Ring,d:DirectProductCategory(b,c),e:I
  -> "HomogeneousAggregate(Ring)"
```

———-

— RMATCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"RectangularMatrixCategory(a:NonNegativeInteger,b:NonNegativeInteger,c:Ring,d:DirectProductCategory(b,c),e:I
 [color=lightblue];
"RectangularMatrixCategory(a:NonNegativeInteger,b:NonNegativeInteger,c:Ring,d:DirectProductCategory(b,c),e:I
  -> "BiModule(a:Ring,b:Ring)"
"RectangularMatrixCategory(a:NonNegativeInteger,b:NonNegativeInteger,c:Ring,d:DirectProductCategory(b,c),e:I
  -> "HOAGG..."

"HOAGG..." [color=lightblue];
"ABELGRP..." [color=lightblue];

"BiModule(a:Ring,b:Ring)" [color=lightblue];
"BiModule(a:Ring,b:Ring)" -> "LeftModule(a:Ring)"
"BiModule(a:Ring,b:Ring)" -> "RightModule(a:Ring)"

"RightModule(a:Ring)" [color=seagreen];
"RightModule(a:Ring)" -> "RightModule(a:Rng)"

"RightModule(a:Rng)" [color=lightblue];
"RightModule(a:Rng)" -> "ABELGRP..."

"LeftModule(a:Ring)" [color=seagreen];
```

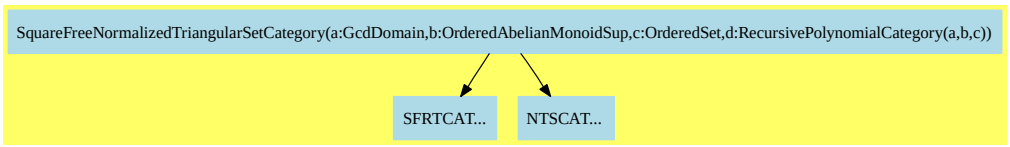
```
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "ABELGRP..."

}
```

———-

10.0.148 SquareFreeNormalizedTriangularSetCategory (SNTSCAT)



— SquareFreeNormalizedTriangularSetCategory.input —

```
)set break resume
)sys rm -f SquareFreeNormalizedTriangularSetCategory.output
)spool SquareFreeNormalizedTriangularSetCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show SquareFreeNormalizedTriangularSetCategory
--R SquareFreeNormalizedTriangularSetCategory(R: GcdDomain,E: OrderedAbelianMonoidSup,V: Ordered
--R Abbreviation for SquareFreeNormalizedTriangularSetCategory is SNTSCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for SNTSCAT
--R
--R------------------------------ Operations --------------------------------
--R ?=? : (%,%) -> Boolean                algebraic? : (V,%) -> Boolean
--R algebraicVariables : % -> List V      augment : (List P,%) -> List %
--R augment : (P,List %) -> List %        augment : (P,%) -> List %
--R coerce : % -> List P                  coerce : % -> OutputForm
--R collect : (%,V) -> %                  collectQuasiMonic : % -> %
--R collectUnder : (%,V) -> %             collectUpper : (%,V) -> %
--R construct : List P -> %               copy : % -> %
--R degree : % -> NonNegativeInteger      empty : () -> %
--R empty? : % -> Boolean                 eq? : (%,%) -> Boolean
--R extend : (List P,%) -> List %         extend : (P,List %) -> List %
--R extend : (P,%) -> List %              extend : (%,P) -> %
```

```
--R first : % -> Union(P,"failed")        hash : % -> SingleInteger
--R headReduce : (P,%) -> P               headReduced? : % -> Boolean
--R headReduced? : (P,%) -> Boolean       infRittWu? : (%,%) -> Boolean
--R initiallyReduce : (P,%) -> P          initiallyReduced? : % -> Boolean
--R initials : % -> List P                internalAugment : (P,%) -> %
--R intersect : (P,List %) -> List %      intersect : (List P,%) -> List %
--R intersect : (P,%) -> List %           invertible? : (P,%) -> Boolean
--R invertibleSet : (P,%) -> List %       last : % -> Union(P,"failed")
--R latex : % -> String                   mainVariable? : (V,%) -> Boolean
--R mainVariables : % -> List V           map : ((P -> P),%) -> %
--R mvar : % -> V                         normalized? : % -> Boolean
--R normalized? : (P,%) -> Boolean        purelyAlgebraic? : % -> Boolean
--R reduceByQuasiMonic : (P,%) -> P       removeZero : (P,%) -> P
--R rest : % -> Union(%,"failed")         retract : List P -> %
--R sample : () -> %                      stronglyReduce : (P,%) -> P
--R stronglyReduced? : % -> Boolean       trivialIdeal? : % -> Boolean
--R variables : % -> List V               zeroSetSplit : List P -> List %
--R ?~=? : (%,%) -> Boolean
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R algebraicCoefficients? : (P,%) -> Boolean
--R any? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
--R augment : (List P,List %) -> List %
--R autoReduced? : (%,((P,List P) -> Boolean)) -> Boolean
--R basicSet : (List P,(P -> Boolean),((P,P) -> Boolean)) -> Union(Record(bas: %,top: List P),"failed")
--R basicSet : (List P,((P,P) -> Boolean)) -> Union(Record(bas: %,top: List P),"failed")
--R coHeight : % -> NonNegativeInteger if V has FINITE
--R convert : % -> InputForm if P has KONVERT INFORM
--R count : ((P -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R count : (P,%) -> NonNegativeInteger if P has SETCAT and $ has finiteAggregate
--R eval : (%,List Equation P) -> % if P has EVALAB P and P has SETCAT
--R eval : (%,Equation P) -> % if P has EVALAB P and P has SETCAT
--R eval : (%,P,P) -> % if P has EVALAB P and P has SETCAT
--R eval : (%,List P,List P) -> % if P has EVALAB P and P has SETCAT
--R every? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
--R extend : (List P,List %) -> List %
--R extendIfCan : (%,P) -> Union(%,"failed")
--R find : ((P -> Boolean),%) -> Union(P,"failed")
--R headRemainder : (P,%) -> Record(num: P,den: R) if R has INTDOM
--R initiallyReduced? : (P,%) -> Boolean
--R internalAugment : (List P,%) -> %
--R intersect : (List P,List %) -> List %
--R invertible? : (P,%) -> List Record(val: Boolean,tower: %)
--R invertibleElseSplit? : (P,%) -> Union(Boolean,List %)
--R lastSubResultant : (P,P,%) -> List Record(val: P,tower: %)
--R lastSubResultantElseSplit : (P,P,%) -> Union(P,List %)
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((P -> P),%) -> % if $ has shallowlyMutable
--R member? : (P,%) -> Boolean if P has SETCAT and $ has finiteAggregate
--R members : % -> List P if $ has finiteAggregate
--R more? : (%,NonNegativeInteger) -> Boolean
```

```
--R parts : % -> List P if $ has finiteAggregate
--R purelyAlgebraic? : (P,%) -> Boolean
--R purelyAlgebraicLeadingMonomial? : (P,%) -> Boolean
--R purelyTranscendental? : (P,%) -> Boolean
--R quasiComponent : % -> Record(close: List P,open: List P)
--R reduce : (P,%,((P,P) -> P),((P,P) -> Boolean)) -> P
--R reduce : (((P,P) -> P),%) -> P if $ has finiteAggregate
--R reduce : (((P,P) -> P),%,P) -> P if $ has finiteAggregate
--R reduce : (((P,P) -> P),%,P,P) -> P if P has SETCAT and $ has finiteAggregate
--R reduced? : (P,%,((P,P) -> Boolean)) -> Boolean
--R remainder : (P,%) -> Record(rnum: R,polnum: P,den: R) if R has INTDOM
--R remove : ((P -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (P,%) -> % if P has SETCAT and $ has finiteAggregate
--R removeDuplicates : % -> % if P has SETCAT and $ has finiteAggregate
--R retractIfCan : List P -> Union(%,"failed")
--R rewriteIdealWithHeadRemainder : (List P,%) -> List P if R has INTDOM
--R rewriteIdealWithRemainder : (List P,%) -> List P if R has INTDOM
--R rewriteSetWithReduction : (List P,%,((P,P) -> P),((P,P) -> Boolean)) -> List P
--R roughBase? : % -> Boolean if R has INTDOM
--R roughEqualIdeals? : (%,%) -> Boolean if R has INTDOM
--R roughSubIdeal? : (%,%) -> Boolean if R has INTDOM
--R roughUnitIdeal? : % -> Boolean if R has INTDOM
--R select : (%,V) -> Union(P,"failed")
--R select : ((P -> Boolean),%) -> % if $ has finiteAggregate
--R size? : (%,NonNegativeInteger) -> Boolean
--R sort : (%,V) -> Record(under: %,floor: %,upper: %)
--R squareFreePart : (P,%) -> List Record(val: P,tower: %)
--R stronglyReduced? : (P,%) -> Boolean
--R triangular? : % -> Boolean if R has INTDOM
--R zeroSetSplit : (List P,Boolean) -> List %
--R zeroSetSplitIntoTriangularSystems : List P -> List Record(close: %,open: List P)
--R
--E 1

)spool
)lisp (bye)
```

———-

— SquareFreeNormalizedTriangularSetCategory.help —

```
====================================================================
SquareFreeNormalizedTriangularSetCategory examples
====================================================================

See Also:
o )show SquareFreeNormalizedTriangularSetCategory
```

———-

See:

⇐ “NormalizedTriangularSetCategory” (NTSCAT) 9.0.125 on page 745
⇐ “SquareFreeRegularTriangularSetCategory” (SFRTCAT) 9.0.130 on page 775

Exports:

algebraic?
algebraicCoefficients?
algebraicVariables
any?
augment
autoReduced?
basicSet
coerce
coHeight
collect
collectQuasiMonic
collectUnder
collectUpper
construct
convert
copy
count
degree
empty
empty?
eq?
eval
every?
extend
extendIfCan
find
first
hash
headReduce
headReduced?
headRemainder
infRittWu?
initiallyReduce
initiallyReduced?
initials
internalAugment
intersect
invertible?
invertibleElseSplit?
invertibleSet
last
lastSubResultant
lastSubResultantElseSplit
latex
less?
mainVariable?
mainVariables
map
map!
member?
members
more?
mvar
normalized?
parts
purelyAlgebraic?
purelyAlgebraicLeadingMonomial?
purelyTranscendental?
quasiComponent
reduce
reduced?
reduceByQuasiMonic
remainder
remove
removeDuplicates
removeZero
rest
retract
retractIfCan
rewriteIdealWithHeadRemainder
rewriteIdealWithRemainder
rewriteSetWithReduction
roughBase?
roughEqualIdeals?
roughSubIdeal?
roughUnitIdeal?
sample
select
size?
sort
squareFreePart
stronglyReduce
stronglyReduced?
triangular?
trivialIdeal?
variables
zeroSetSplit
zeroSetSplitIntoTriangularSystems
#?
?=?
?~=?

Attributes Exported:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **nil**

These exports come from
(p775) SquareFreeRegularTriangularSetCategory(R,E,V,P)
where R:GcdDomain, E:OrderedAbelianMonoidSup, V:OrderedSet,
P:RecursivePolynomialCategory(R,E,V)):

```
algebraic? : (V,%) -> Boolean
algebraicCoefficients? : (P,%) -> Boolean
algebraicVariables : % -> List V
any? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
augment : (List P,List %) -> List %
augment : (List P,%) -> List %
augment : (P,List %) -> List %
augment : (P,%) -> List %
autoReduced? : (%,((P,List P) -> Boolean)) -> Boolean
basicSet :
  (List P,(P -> Boolean),((P,P) -> Boolean)) ->
     Union(Record(bas: %,top: List P),"failed")
basicSet :
  (List P,((P,P) -> Boolean)) ->
     Union(Record(bas: %,top: List P),"failed")
coerce : % -> List P
coerce : % -> OutputForm
coHeight : % -> NonNegativeInteger if V has FINITE
collect : (%,V) -> %
collectQuasiMonic : % -> %
collectUnder : (%,V) -> %
collectUpper : (%,V) -> %
construct : List P -> %
convert : % -> InputForm if P has KONVERT INFORM
copy : % -> %
count : ((P -> Boolean),%) -> NonNegativeInteger
  if $ has finiteAggregate
count : (P,%) -> NonNegativeInteger
  if P has SETCAT
  and $ has finiteAggregate
degree : % -> NonNegativeInteger
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List Equation P) -> % if P has EVALAB P and P has SETCAT
```

```
eval : (%,Equation P) -> % if P has EVALAB P and P has SETCAT
eval : (%,P,P) -> % if P has EVALAB P and P has SETCAT
eval : (%,List P,List P) -> % if P has EVALAB P and P has SETCAT
every? : ((P -> Boolean),%) -> Boolean if $ has finiteAggregate
extend : (List P,List %) -> List %
extend : (List P,%) -> List %
extend : (P,List %) -> List %
extend : (P,%) -> List %
extend : (%,P) -> %
extendIfCan : (%,P) -> Union(%,"failed")
find : ((P -> Boolean),%) -> Union(P,"failed")
first : % -> Union(P,"failed")
hash : % -> SingleInteger
headReduce : (P,%) -> P
headReduced? : % -> Boolean
headReduced? : (P,%) -> Boolean
headRemainder : (P,%) -> Record(num: P,den: R) if R has INTDOM
infRittWu? : (%,%) -> Boolean
initiallyReduce : (P,%) -> P
initiallyReduced? : % -> Boolean
initiallyReduced? : (P,%) -> Boolean
initials : % -> List P
internalAugment : (P,%) -> %
internalAugment : (List P,%) -> %
intersect : (List P,List %) -> List %
intersect : (P,List %) -> List %
intersect : (List P,%) -> List %
intersect : (P,%) -> List %
invertible? : (P,%) -> Boolean
invertible? : (P,%) -> List Record(val: Boolean,tower: %)
invertibleElseSplit? : (P,%) -> Union(Boolean,List %)
invertibleSet : (P,%) -> List %
last : % -> Union(P,"failed")
lastSubResultant : (P,P,%) -> List Record(val: P,tower: %)
lastSubResultantElseSplit : (P,P,%) -> Union(P,List %)
latex : % -> String
less? : (%,NonNegativeInteger) -> Boolean
mainVariable? : (V,%) -> Boolean
mainVariables : % -> List V
map : ((P -> P),%) -> %
map! : ((P -> P),%) -> % if $ has shallowlyMutable
member? : (P,%) -> Boolean if P has SETCAT and $ has finiteAggregate
members : % -> List P if $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
mvar : % -> V
normalized? : % -> Boolean
normalized? : (P,%) -> Boolean
parts : % -> List P if $ has finiteAggregate
purelyAlgebraic? : % -> Boolean
purelyAlgebraic? : (P,%) -> Boolean
```

```
purelyAlgebraicLeadingMonomial? : (P,%) -> Boolean
purelyTranscendental? : (P,%) -> Boolean
quasiComponent : % -> Record(close: List P,open: List P)
reduce : (P,%,((P,P) -> P),((P,P) -> Boolean)) -> P
reduce : (((P,P) -> P),%) -> P if $ has finiteAggregate
reduce : (((P,P) -> P),%,P) -> P if $ has finiteAggregate
reduce : (((P,P) -> P),%,P,P) -> P
  if P has SETCAT
  and $ has finiteAggregate
reduced? : (P,%,((P,P) -> Boolean)) -> Boolean
reduceByQuasiMonic : (P,%) -> P
remainder : (P,%) -> Record(rnum: R,polnum: P,den: R) if R has INTDOM
remove : ((P -> Boolean),%) -> % if $ has finiteAggregate
remove : (P,%) -> % if P has SETCAT and $ has finiteAggregate
removeDuplicates : % -> % if P has SETCAT and $ has finiteAggregate
removeZero : (P,%) -> P
rest : % -> Union(%,"failed")
retract : List P -> %
retractIfCan : List P -> Union(%,"failed")
rewriteIdealWithHeadRemainder : (List P,%) -> List P if R has INTDOM
rewriteIdealWithRemainder : (List P,%) -> List P if R has INTDOM
rewriteSetWithReduction :
   (List P,%,((P,P) -> P),((P,P) -> Boolean)) -> List P
roughBase? : % -> Boolean if R has INTDOM
roughEqualIdeals? : (%,%) -> Boolean if R has INTDOM
roughSubIdeal? : (%,%) -> Boolean if R has INTDOM
roughUnitIdeal? : % -> Boolean if R has INTDOM
sample : () -> %
select : (%,V) -> Union(P,"failed")
select : ((P -> Boolean),%) -> % if $ has finiteAggregate
size? : (%,NonNegativeInteger) -> Boolean
sort : (%,V) -> Record(under: %,floor: %,upper: %)
squareFreePart : (P,%) -> List Record(val: P,tower: %)
stronglyReduce : (P,%) -> P
stronglyReduced? : (P,%) -> Boolean
stronglyReduced? : % -> Boolean
triangular? : % -> Boolean if R has INTDOM
trivialIdeal? : % -> Boolean
variables : % -> List V
zeroSetSplit : List P -> List %
zeroSetSplit : (List P,Boolean) -> List %
zeroSetSplitIntoTriangularSystems :
   List P -> List Record(close: %,open: List P)
#? : % -> NonNegativeInteger if $ has finiteAggregate
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
```

— category SNTSCAT SquareFreeNormalizedTriangularSetCategory

—

```
)abbrev category SNTSCAT SquareFreeNormalizedTriangularSetCategory
++ Author: Marc Moreno Maza
++ Date Created: 10/07/1998
++ Date Last Updated: 12/16/1998
++ Basic Functions:
++ Related Constructors:
++ Also See: essai Graphisme
++ AMS Classifications:
++ Keywords: polynomial, multivariate, ordered variables set
++ References :
++ [1] D. LAZARD "A new method for solving algebraic systems of
++ positive dimension" Discr. App. Math. 33:147-160,1991
++ Description:
++ The category of square-free and normalized triangular sets.
++ Thus, up to the primitivity axiom of [1], these sets are Lazard
++ triangular sets.

SquareFreeNormalizedTriangularSetCategory(R:GcdDomain,_
                                          E:OrderedAbelianMonoidSup,_
                                          V:OrderedSet,_
                                         P:RecursivePolynomialCategory(R,E,V)):
         Category ==
   Join(SquareFreeRegularTriangularSetCategory(R,E,V,P), _
        NormalizedTriangularSetCategory(R,E,V,P))
```

———-

— SNTSCAT.dotabb —

```
"SNTSCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SNTSCAT"];
"SNTSCAT" -> "NTSCAT"
"SNTSCAT" -> "SFRTCAT"
```

———-

— SNTSCAT.dotfull —

```
"SquareFreeNormalizedTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SNTSCAT"];
"SquareFreeNormalizedTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:
 ->
"SquareFreeRegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:Rec

"SquareFreeNormalizedTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:
 ->
```

"NormalizedRegularTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePolyno

———————-

— SNTSCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"SquareFreeNormalizedTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePol
 [color=lightblue];
"SquareFreeNormalizedTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePol
 -> "SFRTCAT..."

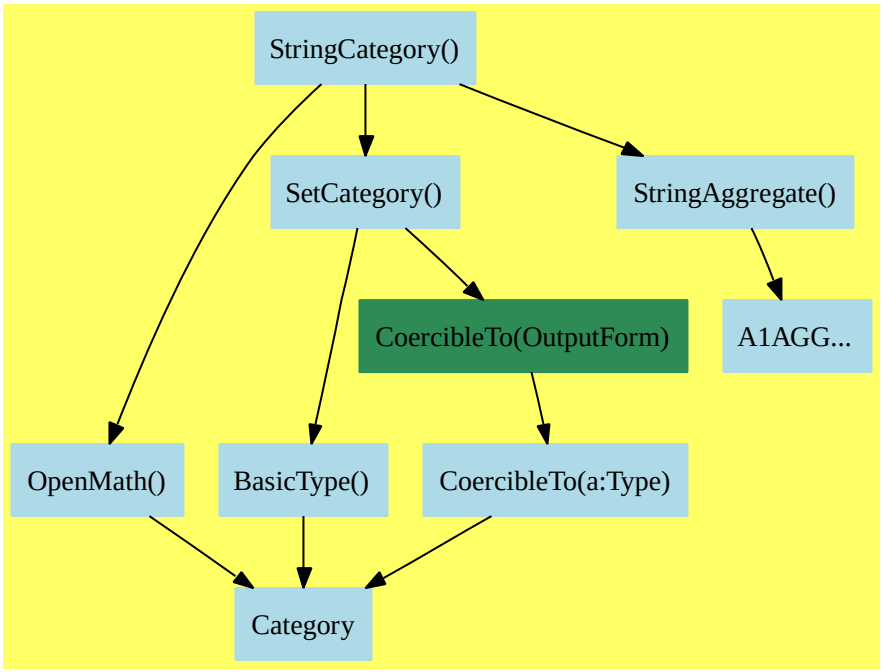
"SquareFreeNormalizedTriangularSetCategory(a:GcdDomain,b:OrderedAbelianMonoidSup,c:OrderedSet,d:RecursivePol
 -> "NTSCAT..."

"SFRTCAT..."  [color=lightblue];
"NTSCAT..."  [color=lightblue];

}
```

———————-

10.0.149 StringCategory (STRICAT)



— **StringCategory.input** —

```
)set break resume
)sys rm -f StringCategory.output
)spool StringCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show StringCategory
--R StringCategory  is a category constructor
--R Abbreviation for StringCategory is STRICAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for STRICAT
--R
--R------------------------------ Operations ------------------------------
--R ?=? : (%,%) -> Boolean                OMwrite : (%,Boolean) -> String
--R OMwrite : % -> String                 coerce : % -> OutputForm
--R coerce : Character -> %               concat : List % -> %
--R concat : (%,%) -> %                   concat : (Character,%) -> %
--R concat : (%,Character) -> %           construct : List Character -> %
```

```
--R copy : % -> %                          delete : (%,Integer) -> %
--R ?.? : (%,%) -> %                       ?.? : (%,Integer) -> Character
--R empty : () -> %                        empty? : % -> Boolean
--R entries : % -> List Character          eq? : (%,%) -> Boolean
--R hash : % -> SingleInteger              index? : (Integer,%) -> Boolean
--R indices : % -> List Integer            insert : (%,%,Integer) -> %
--R latex : % -> String                    leftTrim : (%,Character) -> %
--R lowerCase : % -> %                     lowerCase! : % -> %
--R prefix? : (%,%) -> Boolean             qelt : (%,Integer) -> Character
--R reverse : % -> %                       rightTrim : (%,Character) -> %
--R sample : () -> %                       split : (%,Character) -> List %
--R string : Integer -> %                  suffix? : (%,%) -> Boolean
--R trim : (%,CharacterClass) -> %         trim : (%,Character) -> %
--R upperCase : % -> %                     upperCase! : % -> %
--R ?~=? : (%,%) -> Boolean
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?<? : (%,%) -> Boolean if Character has ORDSET
--R ?<=? : (%,%) -> Boolean if Character has ORDSET
--R ?>? : (%,%) -> Boolean if Character has ORDSET
--R ?>=? : (%,%) -> Boolean if Character has ORDSET
--R OMwrite : (OpenMathDevice,%,Boolean) -> Void
--R OMwrite : (OpenMathDevice,%) -> Void
--R any? : ((Character -> Boolean),%) -> Boolean if $ has finiteAggregate
--R convert : % -> InputForm if Character has KONVERT INFORM
--R copyInto! : (%,%,Integer) -> % if $ has shallowlyMutable
--R count : (Character,%) -> NonNegativeInteger if Character has SETCAT and $ has finiteAggregate
--R count : ((Character -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R delete : (%,UniversalSegment Integer) -> %
--R ?.? : (%,UniversalSegment Integer) -> %
--R elt : (%,Integer,Character) -> Character
--R entry? : (Character,%) -> Boolean if $ has finiteAggregate and Character has SETCAT
--R eval : (%,List Character,List Character) -> % if Character has EVALAB CHAR and Character has SETCAT
--R eval : (%,Character,Character) -> % if Character has EVALAB CHAR and Character has SETCAT
--R eval : (%,Equation Character) -> % if Character has EVALAB CHAR and Character has SETCAT
--R eval : (%,List Equation Character) -> % if Character has EVALAB CHAR and Character has SETCAT
--R every? : ((Character -> Boolean),%) -> Boolean if $ has finiteAggregate
--R fill! : (%,Character) -> % if $ has shallowlyMutable
--R find : ((Character -> Boolean),%) -> Union(Character,"failed")
--R first : % -> Character if Integer has ORDSET
--R insert : (Character,%,Integer) -> %
--R leftTrim : (%,CharacterClass) -> %
--R less? : (%,NonNegativeInteger) -> Boolean
--R map : (((Character,Character) -> Character),%,%) -> %
--R map : ((Character -> Character),%) -> %
--R map! : ((Character -> Character),%) -> % if $ has shallowlyMutable
--R match : (%,%,Character) -> NonNegativeInteger
--R match? : (%,%,Character) -> Boolean
--R max : (%,%) -> % if Character has ORDSET
--R maxIndex : % -> Integer if Integer has ORDSET
--R member? : (Character,%) -> Boolean if Character has SETCAT and $ has finiteAggregate
```

```
--R members : % -> List Character if $ has finiteAggregate
--R merge : (%,%) -> % if Character has ORDSET
--R merge : (((Character,Character) -> Boolean),%,%) -> %
--R min : (%,%) -> % if Character has ORDSET
--R minIndex : % -> Integer if Integer has ORDSET
--R more? : (%,NonNegativeInteger) -> Boolean
--R new : (NonNegativeInteger,Character) -> %
--R parts : % -> List Character if $ has finiteAggregate
--R position : (CharacterClass,%,Integer) -> Integer
--R position : (%,%,Integer) -> Integer
--R position : (Character,%,Integer) -> Integer if Character has SETCAT
--R position : (Character,%) -> Integer if Character has SETCAT
--R position : ((Character -> Boolean),%) -> Integer
--R qsetelt! : (%,Integer,Character) -> Character if $ has shallowlyMutable
--R reduce : (((Character,Character) -> Character),%) -> Character if $ has finiteAggregate
--R reduce : (((Character,Character) -> Character),%,Character) -> Character if $ has finiteAggr
--R reduce : (((Character,Character) -> Character),%,Character,Character) -> Character if Charac
--R remove : ((Character -> Boolean),%) -> % if $ has finiteAggregate
--R remove : (Character,%) -> % if Character has SETCAT and $ has finiteAggregate
--R removeDuplicates : % -> % if Character has SETCAT and $ has finiteAggregate
--R replace : (%,UniversalSegment Integer,%) -> %
--R reverse! : % -> % if $ has shallowlyMutable
--R rightTrim : (%,CharacterClass) -> %
--R select : ((Character -> Boolean),%) -> % if $ has finiteAggregate
--R setelt : (%,UniversalSegment Integer,Character) -> Character if $ has shallowlyMutable
--R setelt : (%,Integer,Character) -> Character if $ has shallowlyMutable
--R size? : (%,NonNegativeInteger) -> Boolean
--R sort : % -> % if Character has ORDSET
--R sort : (((Character,Character) -> Boolean),%) -> %
--R sort! : % -> % if Character has ORDSET and $ has shallowlyMutable
--R sort! : (((Character,Character) -> Boolean),%) -> % if $ has shallowlyMutable
--R sorted? : % -> Boolean if Character has ORDSET
--R sorted? : (((Character,Character) -> Boolean),%) -> Boolean
--R split : (%,CharacterClass) -> List %
--R substring? : (%,%,Integer) -> Boolean
--R swap! : (%,Integer,Integer) -> Void if $ has shallowlyMutable
--R
--E 1

)spool
)lisp (bye)
```

———-

— StringCategory.help —

```
=====================================================================
StringCategory examples
=====================================================================
```

```
See Also:
o )show StringCategory
```

———-

See:

⇐ “OpenMath” (OM) 2.0.13 on page 39
⇐ “SetCategory” (SETCAT) 3.0.34 on page 113
⇐ “StringAggregate” (SRAGG) 9.0.131 on page 786

Exports:

any?	coerce	concat	construct	convert
copy	copyInto!	count	delete	elt
empty	empty?	entry?	entries	eq?
eval	every?	fill!	find	first
hash	index?	indices	insert	latex
leftTrim	less?	lowerCase	lowerCase!	map
map!	match	match?	max	maxIndex
member?	members	merge	min	minIndex
more?	new	OMwrite	parts	position
prefix?	qelt	qsetelt!	reduce	remove
removeDuplicates	replace	reverse	reverse!	rightTrim
sample	select	setelt	size?	sort
sort!	sorted?	split	string	substring?
suffix?	swap!	trim	upperCase	upperCase!
#?	?<?	?<=?	?>?	?>=?
?=?	?.?	?~=?		

Attributes exported:

- **shallowlyMutable** is true if its values have immediate components that are updateable (mutable). Note that the properties of any component domain are irrevelant to the shallowlyMutable proper.
- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **nil**

These are directly exported but not implemented:

```
string : Integer -> %
```

These exports come from (p786) StringAggregate():

```
any? : ((Character -> Boolean),%) -> Boolean
```

```
    if $ has finiteAggregate
coerce : % -> OutputForm
coerce : Character -> %
concat : List % -> %
concat : (%,%) -> %
concat : (Character,%) -> %
concat : (%,Character) -> %
construct : List Character -> %
convert : % -> InputForm
    if Character has KONVERT INFORM
copy : % -> %
copyInto! : (%,%,Integer) -> %
    if $ has shallowlyMutable
count : (Character,%) -> NonNegativeInteger
    if Character has SETCAT
    and $ has finiteAggregate
count : ((Character -> Boolean),%) -> NonNegativeInteger
    if $ has finiteAggregate
delete : (%,UniversalSegment Integer) -> %
delete : (%,Integer) -> %
elt : (%,Integer,Character) -> Character
empty : () -> %
empty? : % -> Boolean
entry? : (Character,%) -> Boolean
    if $ has finiteAggregate
    and Character has SETCAT
entries : % -> List Character
eq? : (%,%) -> Boolean
eval : (%,List Character,List Character) -> %
    if Character has EVALAB CHAR
    and Character has SETCAT
eval : (%,Character,Character) -> %
    if Character has EVALAB CHAR
    and Character has SETCAT
eval : (%,Equation Character) -> %
    if Character has EVALAB CHAR
    and Character has SETCAT
eval : (%,List Equation Character) -> %
    if Character has EVALAB CHAR
    and Character has SETCAT
every? : ((Character -> Boolean),%) -> Boolean
    if $ has finiteAggregate
fill! : (%,Character) -> %
    if $ has shallowlyMutable
find : ((Character -> Boolean),%) -> Union(Character,"failed")
first : % -> Character
    if Integer has ORDSET
hash : % -> SingleInteger
index? : (Integer,%) -> Boolean
indices : % -> List Integer
```

```
insert : (%,%,Integer) -> %
insert : (Character,%,Integer) -> %
latex : % -> String
leftTrim : (%,Character) -> %
leftTrim : (%,CharacterClass) -> %
less? : (%,NonNegativeInteger) -> Boolean
lowerCase : % -> %
lowerCase! : % -> %
map : (((Character,Character) -> Character),%,%) -> %
map : ((Character -> Character),%) -> %
map! : ((Character -> Character),%) -> %
    if $ has shallowlyMutable
match : (%,%,Character) -> NonNegativeInteger
match? : (%,%,Character) -> Boolean
max : (%,%) -> % if Character has ORDSET
maxIndex : % -> Integer if Integer has ORDSET
member? : (Character,%) -> Boolean
    if Character has SETCAT
    and $ has finiteAggregate
members : % -> List Character
    if $ has finiteAggregate
merge : (%,%) -> % if Character has ORDSET
merge : (((Character,Character) -> Boolean),%,%) -> %
min : (%,%) -> % if Character has ORDSET
minIndex : % -> Integer if Integer has ORDSET
more? : (%,NonNegativeInteger) -> Boolean
new : (NonNegativeInteger,Character) -> %
parts : % -> List Character if $ has finiteAggregate
position : (Character,%) -> Integer
    if Character has SETCAT
position : ((Character -> Boolean),%) -> Integer
position : (Character,%,Integer) -> Integer
    if Character has SETCAT
position : (CharacterClass,%,Integer) -> Integer
position : (%,%,Integer) -> Integer
prefix? : (%,%) -> Boolean
qelt : (%,Integer) -> Character
qsetelt! : (%,Integer,Character) -> Character
    if $ has shallowlyMutable
reduce : (((Character,Character) -> Character),%)
   -> Character
    if $ has finiteAggregate
reduce : (((Character,Character) -> Character),%,Character)
   -> Character
    if $ has finiteAggregate
reduce :
 (((Character,Character) -> Character),%,Character,Character)
   -> Character
    if Character has SETCAT
    and $ has finiteAggregate
```

```
remove : ((Character -> Boolean),%) -> %
    if $ has finiteAggregate
remove : (Character,%) -> %
    if Character has SETCAT
    and $ has finiteAggregate
removeDuplicates : % -> %
    if Character has SETCAT
    and $ has finiteAggregate
replace : (%,UniversalSegment Integer,%) -> %
reverse : % -> %
reverse! : % -> % if $ has shallowlyMutable
rightTrim : (%,CharacterClass) -> %
rightTrim : (%,Character) -> %
sample : () -> %
select : ((Character -> Boolean),%) -> %
    if $ has finiteAggregate
setelt :
  (%,UniversalSegment Integer,Character) -> Character
    if $ has shallowlyMutable
setelt : (%,Integer,Character) -> Character
    if $ has shallowlyMutable
size? : (%,NonNegativeInteger) -> Boolean
sort : % -> % if Character has ORDSET
sort : (((Character,Character) -> Boolean),%) -> %
sort! : % -> %
    if Character has ORDSET
    and $ has shallowlyMutable
sort! : (((Character,Character) -> Boolean),%) -> %
    if $ has shallowlyMutable
sorted? : (((Character,Character) -> Boolean),%) -> Boolean
sorted? : % -> Boolean if Character has ORDSET
split : (%,CharacterClass) -> List %
split : (%,Character) -> List %
substring? : (%,%,Integer) -> Boolean
suffix? : (%,%) -> Boolean
swap! : (%,Integer,Integer) -> Void
    if $ has shallowlyMutable
trim : (%,CharacterClass) -> %
trim : (%,Character) -> %
upperCase : % -> %
upperCase! : % -> %
#? : % -> NonNegativeInteger if $ has finiteAggregate
?<? : (%,%) -> Boolean if Character has ORDSET
?<=? : (%,%) -> Boolean if Character has ORDSET
?>? : (%,%) -> Boolean if Character has ORDSET
?>=? : (%,%) -> Boolean if Character has ORDSET
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?.? : (%,UniversalSegment Integer) -> %
?.? : (%,Integer) -> Character
```

```
 ?.? : (%,%) -> %
```

These exports come from (p113) SetCategory():

These exports come from (p39) OpenMath():

```
 OMwrite : (%,Boolean) -> String
 OMwrite : % -> String
 OMwrite : (OpenMathDevice,%,Boolean) -> Void
 OMwrite : (OpenMathDevice,%) -> Void
```

— category STRICAT StringCategory —

```
)abbrev category STRICAT StringCategory
-- Note that StringCategory is built into the old compiler
-- redundant SetCategory added to help A# compiler
++ Description:
++ A category for string-like objects

StringCategory():Category == _
     Join(StringAggregate(), SetCategory, OpenMath) with
  string: Integer -> %
    ++ string(i) returns the decimal representation of i in a string
```

———-

— STRICAT.dotabb —

```
"STRICAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=STRICAT"];
"STRICAT" -> "OM"
"STRICAT" -> "SETCAT"
"STRICAT" -> "SRAGG"
```

———-

— STRICAT.dotfull —

```
"StringCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=STRICAT"];
"StringCategory()" -> "OpenMath()"
"StringCategory()" -> "SetCategory()"
"StringCategory()" -> "StringAggregate()"
```

———-

— STRICAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"StringCategory()" [color=lightblue];
"StringCategory()" -> "OpenMath()"
"StringCategory()" -> "SetCategory()"
"StringCategory()" -> "StringAggregate()"

"OpenMath()" [color=lightblue];
"OpenMath()" -> "Category"

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"StringAggregate()" [color=lightblue];
"StringAggregate()" -> "A1AGG..."

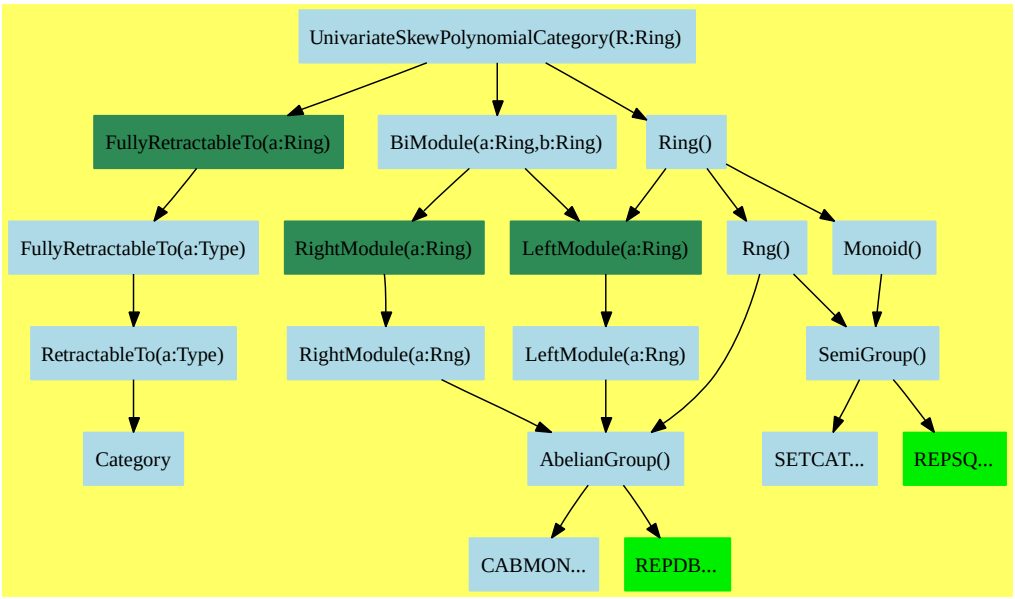
"A1AGG..." [color=lightblue];

"Category" [color=lightblue];

}
```

———-

10.0.150 UnivariateSkewPolynomialCategory (OREPCAT)



— UnivariateSkewPolynomialCategory.input —

```
)set break resume
)sys rm -f UnivariateSkewPolynomialCategory.output
)spool UnivariateSkewPolynomialCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show UnivariateSkewPolynomialCategory
--R UnivariateSkewPolynomialCategory R: Ring  is a category constructor
--R Abbreviation for UnivariateSkewPolynomialCategory is OREPCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for OREPCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (R,%) -> %                      ?*? : (%,R) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        apply : (%,R,R) -> R
--R coefficients : % -> List R            coerce : R -> %
```

```
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R degree : % -> NonNegativeInteger      hash : % -> SingleInteger
--R latex : % -> String                   leadingCoefficient : % -> R
--R one? : % -> Boolean                   recip : % -> Union(%,"failed")
--R reductum : % -> %                     retract : % -> R
--R sample : () -> %                      zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R coefficient : (%,NonNegativeInteger) -> R
--R coerce : Fraction Integer -> % if R has RETRACT FRAC INT
--R content : % -> R if R has GCDDOM
--R exquo : (%,R) -> Union(%,"failed") if R has INTDOM
--R leftDivide : (%,%) -> Record(quotient: %,remainder: %) if R has FIELD
--R leftExactQuotient : (%,%) -> Union(%,"failed") if R has FIELD
--R leftExtendedGcd : (%,%) -> Record(coef1: %,coef2: %,generator: %) if R has FIELD
--R leftGcd : (%,%) -> % if R has FIELD
--R leftLcm : (%,%) -> % if R has FIELD
--R leftQuotient : (%,%) -> % if R has FIELD
--R leftRemainder : (%,%) -> % if R has FIELD
--R minimumDegree : % -> NonNegativeInteger
--R monicLeftDivide : (%,%) -> Record(quotient: %,remainder: %) if R has INTDOM
--R monicRightDivide : (%,%) -> Record(quotient: %,remainder: %) if R has INTDOM
--R monomial : (R,NonNegativeInteger) -> %
--R primitivePart : % -> % if R has GCDDOM
--R retract : % -> Fraction Integer if R has RETRACT FRAC INT
--R retract : % -> Integer if R has RETRACT INT
--R retractIfCan : % -> Union(R,"failed")
--R retractIfCan : % -> Union(Fraction Integer,"failed") if R has RETRACT FRAC INT
--R retractIfCan : % -> Union(Integer,"failed") if R has RETRACT INT
--R rightDivide : (%,%) -> Record(quotient: %,remainder: %) if R has FIELD
--R rightExactQuotient : (%,%) -> Union(%,"failed") if R has FIELD
--R rightExtendedGcd : (%,%) -> Record(coef1: %,coef2: %,generator: %) if R has FIELD
--R rightGcd : (%,%) -> % if R has FIELD
--R rightLcm : (%,%) -> % if R has FIELD
--R rightQuotient : (%,%) -> % if R has FIELD
--R rightRemainder : (%,%) -> % if R has FIELD
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— UnivariateSkewPolynomialCategory.help —

```
=====================================================================
UnivariateSkewPolynomialCategory examples
=====================================================================

See Also:
o )show UnivariateSkewPolynomialCategory
```

———-

See:

⇒ “LinearOrdinaryDifferentialOperatorCategory” (LODOCAT) 11.0.156 on page 971
⇐ “BiModule” (BMODULE) 9.0.122 on page 726
⇐ “FullyRetractableTo” (FRETRCT) 3.0.27 on page 87
⇐ “Ring” (RING) 9.0.129 on page 770

Exports:

0	1	apply	characteristic
coefficient	coefficients	coerce	content
degree	exquo	hash	latex
leadingCoefficient	leftDivide	leftExactQuotient	leftExtendedGcd
leftGcd	leftLcm	leftQuotient	leftRemainder
minimumDegree	monicLeftDivide	monicRightDivide	monomial
one?	primitivePart	recip	reductum
retract	retractIfCan	rightDivide	rightExactQuotient
rightExtendedGcd	rightGcd	rightLcm	rightQuotient
rightRemainder	sample	subtractIfCan	zero?
?*?	?**?	?+?	?-?
-?	?=?	?^?	?~=?

Attributes Exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return “failed” if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
apply : (%,R,R) -> R
coefficient : (%,NonNegativeInteger) -> R
degree : % -> NonNegativeInteger
leadingCoefficient : % -> R
leftDivide : (%,%) -> Record(quotient: %,remainder: %) if R has FIELD
minimumDegree : % -> NonNegativeInteger
```

```
monicLeftDivide : (%,%) -> Record(quotient: %,remainder: %) if R has INTDOM
monicRightDivide : (%,%) -> Record(quotient: %,remainder: %) if R has INTDOM
monomial : (R,NonNegativeInteger) -> %
reductum : % -> %
rightDivide : (%,%) -> Record(quotient: %,remainder: %) if R has FIELD
```

These are implemented by this category:

```
coefficients : % -> List R
coerce : R -> %
content : % -> R if R has GCDDOM
exquo : (%,R) -> Union(%,"failed") if R has INTDOM
leftExactQuotient : (%,%) -> Union(%,"failed") if R has FIELD
leftExtendedGcd : (%,%) -> Record(coef1: %,coef2: %,generator: %) if R has FIELD
leftGcd : (%,%) -> % if R has FIELD
leftLcm : (%,%) -> % if R has FIELD
leftQuotient : (%,%) -> % if R has FIELD
leftRemainder : (%,%) -> % if R has FIELD
primitivePart : % -> % if R has GCDDOM
retractIfCan : % -> Union(R,"failed")
rightExactQuotient : (%,%) -> Union(%,"failed") if R has FIELD
rightExtendedGcd : (%,%) -> Record(coef1: %,coef2: %,generator: %) if R has FIELD
rightGcd : (%,%) -> % if R has FIELD
rightLcm : (%,%) -> % if R has FIELD
rightQuotient : (%,%) -> % if R has FIELD
rightRemainder : (%,%) -> % if R has FIELD
?*? : (R,%) -> %
```

These exports come from (p770) Ring():

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : % -> OutputForm
coerce : Integer -> %
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?**? : (%,NonNegativeInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (Integer,%) -> %
```

```
?*? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?^? : (%,PositiveInteger) -> %
```

These exports come from (p726) BiModule(R:Ring,R:Ring):

```
?*? : (%,R) -> %
```

These exports come from (p87) FullyRetractableTo(R:Ring):

```
coerce : Fraction Integer -> % if R has RETRACT FRAC INT
retract : % -> R
retract : % -> Fraction Integer if R has RETRACT FRAC INT
retract : % -> Integer if R has RETRACT INT
retractIfCan : % -> Union(Fraction Integer,"failed") if R has RETRACT FRAC INT
retractIfCan : % -> Union(Integer,"failed") if R has RETRACT INT
```

— category OREPCAT UnivariateSkewPolynomialCategory —

```
)abbrev category OREPCAT UnivariateSkewPolynomialCategory
++ Author: Manuel Bronstein, Jean Della Dora, Stephen M. Watt
++ Date Created: 19 October 1993
++ Date Last Updated: 1 February 1994
++ Description:
++ This is the category of univariate skew polynomials over an Ore
++ coefficient ring.
++ The multiplication is given by \spad{x a = \sigma(a) x + \delta a}.
++ This category is an evolution of the types
++ MonogenicLinearOperator, OppositeMonogenicLinearOperator, and
++ NonCommutativeOperatorDivision

UnivariateSkewPolynomialCategory(R:Ring):
  Category == Join(Ring, BiModule(R, R), FullyRetractableTo R) with
      degree: $ -> NonNegativeInteger
          ++ degree(l) is \spad{n} if
          ++   \spad{l = sum(monomial(a(i),i), i = 0..n)}.
      minimumDegree: $ -> NonNegativeInteger
          ++ minimumDegree(l) is the smallest \spad{k} such that
          ++ \spad{a(k) ^= 0} if
          ++   \spad{l = sum(monomial(a(i),i), i = 0..n)}.
      leadingCoefficient: $ -> R
          ++ leadingCoefficient(l) is \spad{a(n)} if
          ++   \spad{l = sum(monomial(a(i),i), i = 0..n)}.
      reductum: $ -> $
          ++ reductum(l) is \spad{l - monomial(a(n),n)} if
          ++   \spad{l = sum(monomial(a(i),i), i = 0..n)}.
```

```
coefficient: ($, NonNegativeInteger) -> R
    ++ coefficient(l,k) is \spad{a(k)} if
    ++   \spad{l = sum(monomial(a(i),i), i = 0..n)}.
monomial: (R, NonNegativeInteger) -> $
    ++ monomial(c,k) produces c times the k-th power of
    ++ the generating operator, \spad{monomial(1,1)}.
coefficients: % -> List R
    ++ coefficients(l) returns the list of all the nonzero
    ++ coefficients of l.
apply: (%, R, R) -> R
  ++ apply(p, c, m) returns \spad{p(m)} where the action is
  ++ given by \spad{x m = c sigma(m) + delta(m)}.
if R has CommutativeRing then Algebra R
if R has IntegralDomain then
  "exquo": (%, R) -> Union(%, "failed")
    ++ exquo(l, a) returns the exact quotient of l by a,
    ++ returning \axiom{"failed"} if this is not possible.
  monicLeftDivide:   (%, %) -> Record(quotient: %, remainder: %)
    ++ monicLeftDivide(a,b) returns the pair \spad{[q,r]} such that
    ++ \spad{a = b*q + r} and the degree of \spad{r} is
    ++ less than the degree of \spad{b}.
    ++ \spad{b} must be monic.
    ++ This process is called ``left division''.
  monicRightDivide:   (%, %) -> Record(quotient: %, remainder: %)
    ++ monicRightDivide(a,b) returns the pair \spad{[q,r]} such that
    ++ \spad{a = q*b + r} and the degree of \spad{r} is
    ++ less than the degree of \spad{b}.
    ++ \spad{b} must be monic.
    ++ This process is called ``right division''.
if R has GcdDomain then
  content: % -> R
    ++ content(l) returns the gcd of all the coefficients of l.
  primitivePart: % -> %
    ++ primitivePart(l) returns l0 such that \spad{l = a * l0}
    ++ for some a in R, and \spad{content(l0) = 1}.
if R has Field then
  leftDivide:   (%, %) -> Record(quotient: %, remainder: %)
    ++ leftDivide(a,b) returns the pair \spad{[q,r]} such that
    ++ \spad{a = b*q + r} and the degree of \spad{r} is
    ++ less than the degree of \spad{b}.
    ++ This process is called ``left division''.
  leftQuotient:  (%, %) -> %
    ++ leftQuotient(a,b) computes the pair \spad{[q,r]} such that
    ++ \spad{a = b*q + r} and the degree of \spad{r} is
    ++ less than the degree of \spad{b}.
    ++ The value \spad{q} is returned.
  leftRemainder:  (%, %) -> %
    ++ leftRemainder(a,b) computes the pair \spad{[q,r]} such that
    ++ \spad{a = b*q + r} and the degree of \spad{r} is
    ++ less than the degree of \spad{b}.
```

```
  ++ The value \spad{r} is returned.
leftExactQuotient:(%, %) -> Union(%, "failed")
  ++ leftExactQuotient(a,b) computes the value \spad{q}, if it exists,
  ++  such that \spad{a = b*q}.
leftGcd:   (%, %) -> %
  ++ leftGcd(a,b) computes the value \spad{g} of highest degree
  ++ such that
  ++    \spad{a = g*aa}
  ++    \spad{b = g*bb}
  ++ for some values \spad{aa} and \spad{bb}.
  ++ The value \spad{g} is computed using left-division.
leftExtendedGcd:   (%, %) -> Record(coef1:%, coef2:%, generator:%)
  ++ leftExtendedGcd(a,b) returns \spad{[c,d]} such that
  ++ \spad{g = a * c + b * d = leftGcd(a, b)}.
rightLcm:   (%, %) -> %
  ++ rightLcm(a,b) computes the value \spad{m} of lowest degree
  ++ such that \spad{m = a*aa = b*bb} for some values
  ++ \spad{aa} and \spad{bb}.  The value \spad{m} is
  ++ computed using left-division.
rightDivide:   (%, %) -> Record(quotient: %, remainder: %)
  ++ rightDivide(a,b) returns the pair \spad{[q,r]} such that
  ++ \spad{a = q*b + r} and the degree of \spad{r} is
  ++ less than the degree of \spad{b}.
  ++ This process is called ``right division''.
rightQuotient:  (%, %) -> %
  ++ rightQuotient(a,b) computes the pair \spad{[q,r]} such that
  ++ \spad{a = q*b + r} and the degree of \spad{r} is
  ++ less than the degree of \spad{b}.
  ++ The value \spad{q} is returned.
rightRemainder:  (%, %) -> %
  ++ rightRemainder(a,b) computes the pair \spad{[q,r]} such that
  ++ \spad{a = q*b + r} and the degree of \spad{r} is
  ++ less than the degree of \spad{b}.
  ++ The value \spad{r} is returned.
rightExactQuotient:(%, %) -> Union(%, "failed")
  ++ rightExactQuotient(a,b) computes the value \spad{q}, if it exists
  ++ such that \spad{a = q*b}.
rightGcd:   (%, %) -> %
  ++ rightGcd(a,b) computes the value \spad{g} of highest degree
  ++ such that
  ++    \spad{a = aa*g}
  ++    \spad{b = bb*g}
  ++ for some values \spad{aa} and \spad{bb}.
  ++ The value \spad{g} is computed using right-division.
rightExtendedGcd:   (%, %) -> Record(coef1:%, coef2:%, generator:%)
  ++ rightExtendedGcd(a,b) returns \spad{[c,d]} such that
  ++ \spad{g = c * a + d * b = rightGcd(a, b)}.
leftLcm:   (%, %) -> %
  ++ leftLcm(a,b) computes the value \spad{m} of lowest degree
  ++ such that \spad{m = aa*a = bb*b} for some values
```

```
        ++ \spad{aa} and \spad{bb}.  The value \spad{m} is
        ++ computed using right-division.

   add
      coerce(x:R):% == monomial(x, 0)

      coefficients l ==
        ans:List(R) := empty()
        while l ^= 0 repeat
          ans := concat(leadingCoefficient l, ans)
          l   := reductum l
        ans

      a:R * y:% ==
        z:% := 0
        while y ^= 0 repeat
          z := z + monomial(a * leadingCoefficient y, degree y)
          y := reductum y
        z

      retractIfCan(x:%):Union(R, "failed") ==
        zero? x or zero? degree x => leadingCoefficient x
        "failed"

      if R has IntegralDomain then
        l exquo a ==
          ans:% := 0
          while l ^= 0 repeat
            (u := (leadingCoefficient(l) exquo a)) case "failed" =>
               return "failed"
            ans := ans + monomial(u::R, degree l)
            l   := reductum l
          ans

      if R has GcdDomain then
        content l       == gcd coefficients l

        primitivePart l == (l exquo content l)::%

      if R has Field then
        leftEEA:  (%, %) -> Record(gcd:%, coef1:%, coef2:%, lcm:%)
        rightEEA: (%, %) -> Record(gcd:%, coef1:%, coef2:%, lcm:%)
        ncgcd:    (%, %, (%, %) -> %) -> %
        nclcm:  (%, %, (%, %) -> Record(gcd:%, coef1:%, coef2:%, lcm:%)) -> %
        exactQuotient: Record(quotient:%, remainder:%) -> Union(%, "failed")
        extended: (%, %, (%, %) -> Record(gcd:%, coef1:%, coef2:%, lcm:%)) ->
                                        Record(coef1:%, coef2:%, generator:%)

        leftQuotient(a, b)         == leftDivide(a,b).quotient
```

```
leftRemainder(a, b)        == leftDivide(a,b).remainder

leftExtendedGcd(a, b)      == extended(a, b, leftEEA)

rightLcm(a, b)             == nclcm(a, b, leftEEA)

rightQuotient(a, b)        == rightDivide(a,b).quotient

rightRemainder(a, b)       == rightDivide(a,b).remainder

rightExtendedGcd(a, b)     == extended(a, b, rightEEA)

leftLcm(a, b)              == nclcm(a, b, rightEEA)

leftExactQuotient(a, b)    == exactQuotient leftDivide(a, b)

rightExactQuotient(a, b)   == exactQuotient rightDivide(a, b)

rightGcd(a, b)             == ncgcd(a, b, rightRemainder)

leftGcd(a, b)              == ncgcd(a, b, leftRemainder)

exactQuotient qr  == (zero?(qr.remainder) => qr.quotient; "failed")

-- returns [g = leftGcd(a, b), c, d, l = rightLcm(a, b)]
-- such that g := a c + b d
leftEEA(a, b) ==
  a0 := a
  u0:% := v:% := 1
  v0:% := u:% := 0
  while b ^= 0 repeat
    qr     := leftDivide(a, b)
    (a, b) := (b, qr.remainder)
    (u0, u):= (u, u0 - u * qr.quotient)
    (v0, v):= (v, v0 - v * qr.quotient)
  [a, u0, v0, a0 * u]

ncgcd(a, b, ncrem) ==
  zero? a => b
  zero? b => a
  degree a < degree b => ncgcd(b, a, ncrem)
  while b ^= 0 repeat (a, b) := (b, ncrem(a, b))
  a

extended(a, b, eea) ==
  zero? a => [0, 1, b]
  zero? b => [1, 0, a]
  degree a < degree b =>
    rec := eea(b, a)
    [rec.coef2, rec.coef1, rec.gcd]
```

```
          rec := eea(a, b)
          [rec.coef1, rec.coef2, rec.gcd]

        nclcm(a, b, eea) ==
          zero? a or zero? b => 0
          degree a < degree b => nclcm(b, a, eea)
          rec := eea(a, b)
          rec.lcm

        -- returns [g = rightGcd(a, b), c, d, l = leftLcm(a, b)]
        -- such that g := a c + b d
        rightEEA(a, b) ==
          a0 := a
          u0:% := v:% := 1
          v0:% := u:% := 0
          while b ^= 0 repeat
            qr     := rightDivide(a, b)
            (a, b) := (b, qr.remainder)
            (u0, u):= (u, u0 - qr.quotient * u)
            (v0, v):= (v, v0 - qr.quotient * v)
          [a, u0, v0, u * a0]
```

———-

— OREPCAT.dotabb —

```
"OREPCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OREPCAT"];
"OREPCAT" -> "BMODULE"
"OREPCAT" -> "FRETRCT"
"OREPCAT" -> "RING"
```

———-

— OREPCAT.dotfull —

```
"UnivariateSkewPolynomialCategory(R:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OREPCAT"];
"UnivariateSkewPolynomialCategory(R:Ring)"
  -> "BiModule(a:Ring,b:Ring)"
"UnivariateSkewPolynomialCategory(R:Ring)"
  -> "FullyRetractableTo(a:Ring)"
"UnivariateSkewPolynomialCategory(R:Ring)"
  -> "Ring()"
```

———-

— OREPCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"UnivariateSkewPolynomialCategory(R:Ring)" [color=lightblue];
"UnivariateSkewPolynomialCategory(R:Ring)"
  -> "BiModule(a:Ring,b:Ring)"
"UnivariateSkewPolynomialCategory(R:Ring)"
  -> "FullyRetractableTo(a:Ring)"
"UnivariateSkewPolynomialCategory(R:Ring)"
  -> "Ring()"

"FullyRetractableTo(a:Ring)" [color=seagreen];
"FullyRetractableTo(a:Ring)" -> "FullyRetractableTo(a:Type)"

"FullyRetractableTo(a:Type)" [color=lightblue];
"FullyRetractableTo(a:Type)" -> "RetractableTo(a:Type)"

"RetractableTo(a:Type)" [color=lightblue];
"RetractableTo(a:Type)" -> "Category"

"Category" [color=lightblue];

"Ring()" [color=lightblue];
"Ring()" -> "Rng()"
"Ring()" -> "Monoid()"
"Ring()" -> "LeftModule(a:Ring)"

"Rng()" [color=lightblue];
"Rng()" -> "AbelianGroup()"
"Rng()" -> "SemiGroup()"

"Monoid()" [color=lightblue];
"Monoid()" -> "SemiGroup()"

"BiModule(a:Ring,b:Ring)" [color=lightblue];
"BiModule(a:Ring,b:Ring)" -> "LeftModule(a:Ring)"
"BiModule(a:Ring,b:Ring)" -> "RightModule(a:Ring)"

"RightModule(a:Ring)" [color=seagreen];
"RightModule(a:Ring)" -> "RightModule(a:Rng)"

"RightModule(a:Rng)" [color=lightblue];
"RightModule(a:Rng)" -> "AbelianGroup()"

"LeftModule(a:Ring)" [color=seagreen];
```

```
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

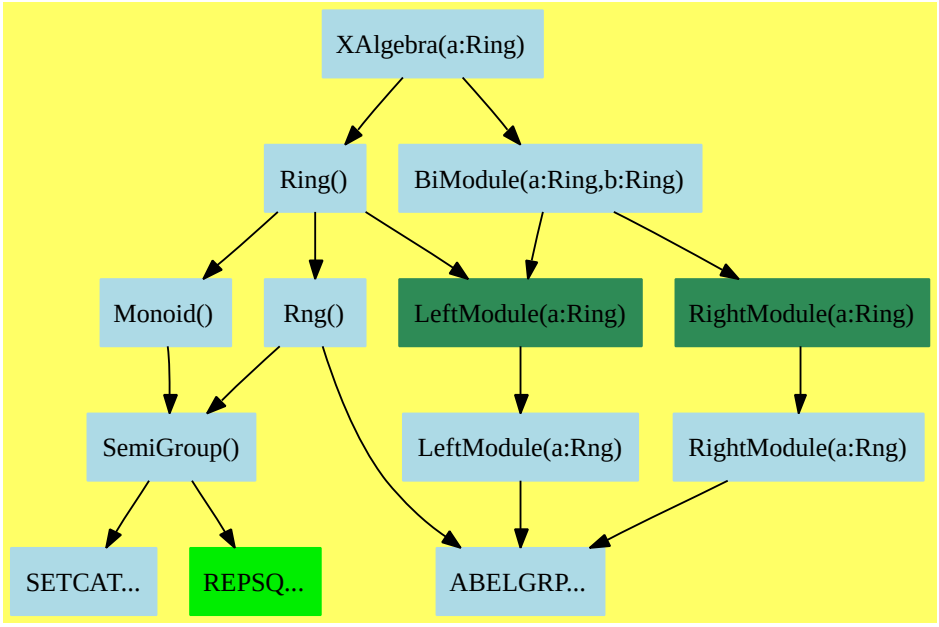
"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "AbelianGroup()"

"AbelianGroup()" [color=lightblue];
"AbelianGroup()" -> "CABMON..."
"AbelianGroup()" -> "REPDB..."

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SETCAT..."
"SemiGroup()" -> "REPSQ..."

"REPDB..." [color="#00EE00"];
"REPSQ..." [color="#00EE00"];
"SETCAT..." [color=lightblue];
"CABMON..." [color=lightblue];
}
```

10.0.151 XAlgebra (XALG)



— XAlgebra.input —

```
)set break resume
)sys rm -f XAlgebra.output
)spool XAlgebra.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show XAlgebra
--R XAlgebra R: Ring  is a category constructor
--R Abbreviation for XAlgebra is XALG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for XALG
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (R,%) -> %                      ?*? : (%,R) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        coerce : R -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R hash : % -> SingleInteger             latex : % -> String
--R one? : % -> Boolean                   recip : % -> Union(%,"failed")
--R sample : () -> %                      zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— XAlgebra.help —

```
=====================================================================
XAlgebra examples
=====================================================================
```

```
See Also:
o )show XAlgebra
```

See:

⇒ "XFreeAlgebra" (XFALG) 11.0.159 on page 992
⇐ "BiModule" (BMODULE) 9.0.122 on page 726
⇐ "Ring" (RING) 9.0.129 on page 770

Exports:

0	1	characteristic	coerce	hash
latex	one?	recip	sample	subtractIfCan
zero?	?^?	?~=?	?*?	?**?
?+?	?-?	-?	?=?	

Attributes Exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
coerce : R -> %
```

These exports come from (p770) Ring():

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : % -> OutputForm
coerce : Integer -> %
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
```

```
 ?*? : (PositiveInteger,%) -> %
 ?*? : (Integer,%) -> %
 ?*? : (%,%) -> %
 ?-? : (%,%) -> %
 -? : % -> %
 ?**? : (%,NonNegativeInteger) -> %
 ?**? : (%,PositiveInteger) -> %
 ?^? : (%,NonNegativeInteger) -> %
 ?^? : (%,PositiveInteger) -> %
```

These exports come from (p726) BiModule(R:Ring,R:Ring):

```
 ?*? : (R,%) -> %
 ?*? : (%,R) -> %
```

— category XALG XAlgebra —

```
)abbrev category XALG XAlgebra
++ Author: Michel Petitot petitot@lifl.fr
++ Date Created: 91
++ Date Last Updated: 7 Juillet 92
++ Fix History: compilation v 2.1 le 13 dec 98
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ This is the category of algebras over non-commutative rings.
++ It is used by constructors of non-commutative algebras such as
++ XPolynomialRing and XFreeAlgebra

XAlgebra(R: Ring): Category ==
  Join(Ring, BiModule(R,R)) with
    coerce: R -> %
      ++ \spad{coerce(r)} equals  \spad{r*1}.
    if R has CommutativeRing then Algebra(R)
```

———-

— XALG.dotabb —

```
"XALG"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=XALG"];
"XALG" -> "BMODULE"
"XALG" -> "RING"
```

———-

— XALG.dotfull —

```
"XAlgebra(a:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=XALG"];
"XAlgebra(a:Ring)" -> "Ring()"
"XAlgebra(a:Ring)" -> "BiModule(a:Ring,b:Ring)"
```

———-

— XALG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"XAlgebra(a:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=XALG"];
"XAlgebra(a:Ring)" -> "Ring()"
"XAlgebra(a:Ring)" -> "BiModule(a:Ring,b:Ring)"

"Ring()" [color=lightblue];
"Ring()" -> "Rng()"
"Ring()" -> "Monoid()"
"Ring()" -> "LeftModule(a:Ring)"

"BiModule(a:Ring,b:Ring)" [color=lightblue];
"BiModule(a:Ring,b:Ring)" -> "LeftModule(a:Ring)"
"BiModule(a:Ring,b:Ring)" -> "RightModule(a:Ring)"

"RightModule(a:Ring)" [color=seagreen];
"RightModule(a:Ring)" -> "RightModule(a:Rng)"

"RightModule(a:Rng)" [color=lightblue];
"RightModule(a:Rng)" -> "ABELGRP..."

"Rng()" [color=lightblue];
"Rng()" -> "ABELGRP..."
"Rng()" -> "SemiGroup()"

"Monoid()" [color=lightblue];
"Monoid()" -> "SemiGroup()"

"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"
```

```
"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "ABELGRP..."

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SETCAT..."
"SemiGroup()" -> "REPSQ..."

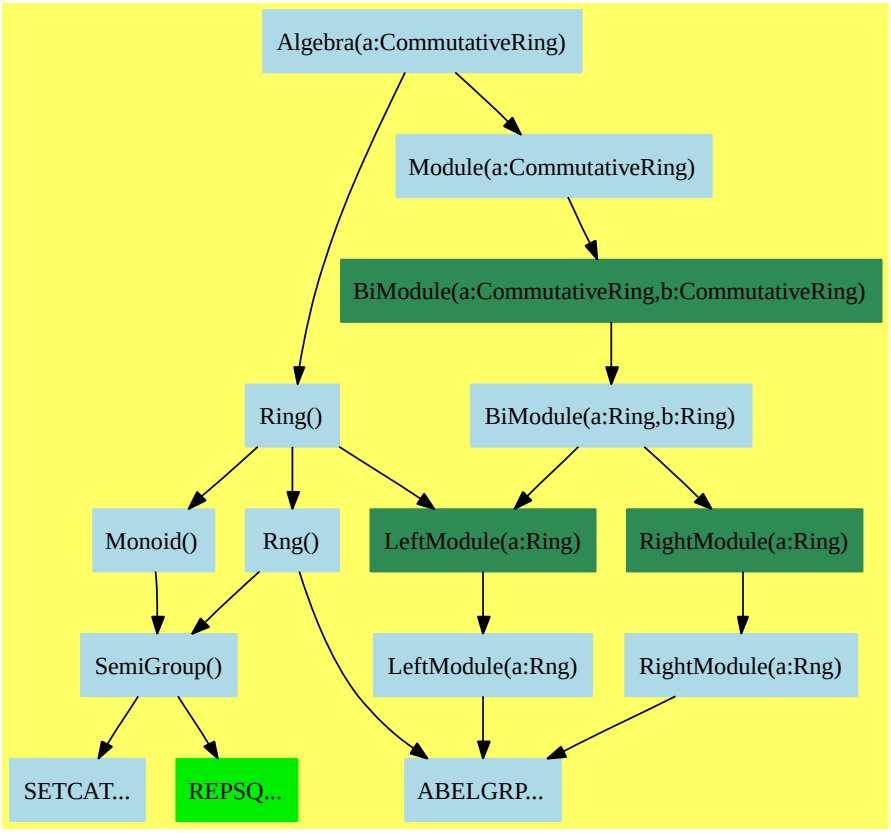
"REPSQ..." [color="#00EE00"];
"SETCAT..." [color=lightblue];
"ABELGRP..." [color=lightblue];
}
```

———————-

Chapter 11

Category Layer 10

11.0.152 Algebra (ALGEBRA)



— Algebra.input —

```
)set break resume
)sys rm -f Algebra.output
)spool Algebra.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show Algebra
--R Algebra R: CommutativeRing  is a category constructor
--R Abbreviation for Algebra is ALGEBRA
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for ALGEBRA
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (R,%) -> %                      ?*? : (%,R) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        coerce : R -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R hash : % -> SingleInteger             latex : % -> String
--R one? : % -> Boolean                   recip : % -> Union(%,"failed")
--R sample : () -> %                      zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— Algebra.help —

```
=====================================================================
Algebra examples
=====================================================================
```

```
See Also:
o )show Algebra
```

———-

See:

⇒ “DivisionRing” (DIVRING) 12.0.161 on page 1013
⇒ “FiniteRankAlgebra” (FINRALG) 17.0.194 on page 1329
⇒ “FunctionSpace” (FS) 17.0.195 on page 1336
⇒ “IntegralDomain” (INTDOM) 12.0.164 on page 1047
⇒ “MonogenicLinearOperator” (MLO) 12.0.165 on page 1053
⇒ “OctonionCategory” (OC) 12.0.166 on page 1059
⇒ “QuotientFieldCategory” (QFCAT) 17.0.198 on page 1376
⇒ “QuaternionCategory” (QUATCAT) 12.0.167 on page 1071
⇒ “RealClosedField” (RCFIELD) 17.0.199 on page 1390
⇐ “Module” (MODULE) 10.0.143 on page 874
⇐ “Ring” (RING) 9.0.129 on page 770

Exports:

1	0	characteristic	coerce	hash
latex	one?	recip	sample	subtractIfCan
zero?	?*?	?+?	?-?	-?
?=?	?~=?	?**?	?^?	

Attributes exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return “failed” if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are implemented by this category:

```
coerce : R -> %
```

These exports come from (p770) Ring():

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : Integer -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
```

```
 one? : % -> Boolean
 recip : % -> Union(%,"failed")
 sample : () -> %
 subtractIfCan : (%,%) -> Union(%,"failed")
 zero? : % -> Boolean
 ?+? : (%,%) -> %
 ?=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
 ?*? : (NonNegativeInteger,%) -> %
 ?*? : (PositiveInteger,%) -> %
 ?*? : (Integer,%) -> %
 ?*? : (%,%) -> %
 ?-? : (%,%) -> %
 -? : % -> %
 ?**? : (%,PositiveInteger) -> %
 ?**? : (%,NonNegativeInteger) -> %
 ?^? : (%,NonNegativeInteger) -> %
 ?^? : (%,PositiveInteger) -> %
```

These exports come from (p874) Module(R:CommutativeRing):

```
 ?*? : (R,%) -> %
 ?*? : (%,R) -> %
```

— category ALGEBRA Algebra —

```
)abbrev category ALGEBRA Algebra
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The category of associative algebras (modules which are themselves rings).
++
++ Axioms\br
++ \tab{5}\spad{(b+c)::% = (b::%) + (c::%)}\br
++ \tab{5}\spad{(b*c)::% = (b::%) * (c::%)}\br
++ \tab{5}\spad{(1::R)::% = 1::%}\br
++ \tab{5}\spad{b*x = (b::%)*x}\br
++ \tab{5}\spad{r*(a*b) = (r*a)*b = a*(r*b)}

Algebra(R:CommutativeRing): Category ==
  Join(Ring, Module R) with
      coerce: R -> %
```

```
        ++ coerce(r) maps the ring element r to a member of the algebra.
 add
  coerce(x:R):% == x * 1$%
```

———-

— **ALGEBRA.dotabb** —

```
"ALGEBRA"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ALGEBRA"];
"ALGEBRA" -> "RING"
"ALGEBRA" -> "MODULE"
```

———-

— **ALGEBRA.dotfull** —

```
"Algebra(a:CommutativeRing)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ALGEBRA"];
"Algebra(a:CommutativeRing)" -> "Ring()"
"Algebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"

"Algebra(a:Field)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=ALGEBRA"];
"Algebra(a:Field)" -> "Algebra(a:CommutativeRing)"

"Algebra(a:CommutativeRing)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=ALGEBRA"];
"Algebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"

"Algebra(Fraction(Integer))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=ALGEBRA"];
"Algebra(Fraction(Integer))" -> "Algebra(a:CommutativeRing)"

"Algebra(Integer)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=ALGEBRA"];
"Algebra(Integer)" -> "Algebra(a:CommutativeRing)"

"Algebra(IntegralDomain)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=ALGEBRA"];
"Algebra(IntegralDomain)" -> "Algebra(a:CommutativeRing)"
```

———-

— **ALGEBRA.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"Algebra(a:CommutativeRing)" [color=lightblue];
"Algebra(a:CommutativeRing)" -> "Ring()"
"Algebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"

"Ring()" [color=lightblue];
"Ring()" -> "Rng()"
"Ring()" -> "Monoid()"
"Ring()" -> "LeftModule(a:Ring)"

"Rng()" [color=lightblue];
"Rng()" -> "ABELGRP..."
"Rng()" -> "SemiGroup()"

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SETCAT..."
"SemiGroup()" -> "REPSQ..."

"Monoid()" [color=lightblue];
"Monoid()" -> "SemiGroup()"

"Module(a:CommutativeRing)" [color=lightblue];
"Module(a:CommutativeRing)" ->
  "BiModule(a:CommutativeRing,b:CommutativeRing)"

"BiModule(a:CommutativeRing,b:CommutativeRing)" [color=seagreen];
"BiModule(a:CommutativeRing,b:CommutativeRing)" -> "BiModule(a:Ring,b:Ring)"

"BiModule(a:Ring,b:Ring)" [color=lightblue];
"BiModule(a:Ring,b:Ring)" -> "LeftModule(a:Ring)"
"BiModule(a:Ring,b:Ring)" -> "RightModule(a:Ring)"

"RightModule(a:Ring)" [color=seagreen];
"RightModule(a:Ring)" -> "RightModule(a:Rng)"

"RightModule(a:Rng)" [color=lightblue];
"RightModule(a:Rng)" -> "ABELGRP..."

"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

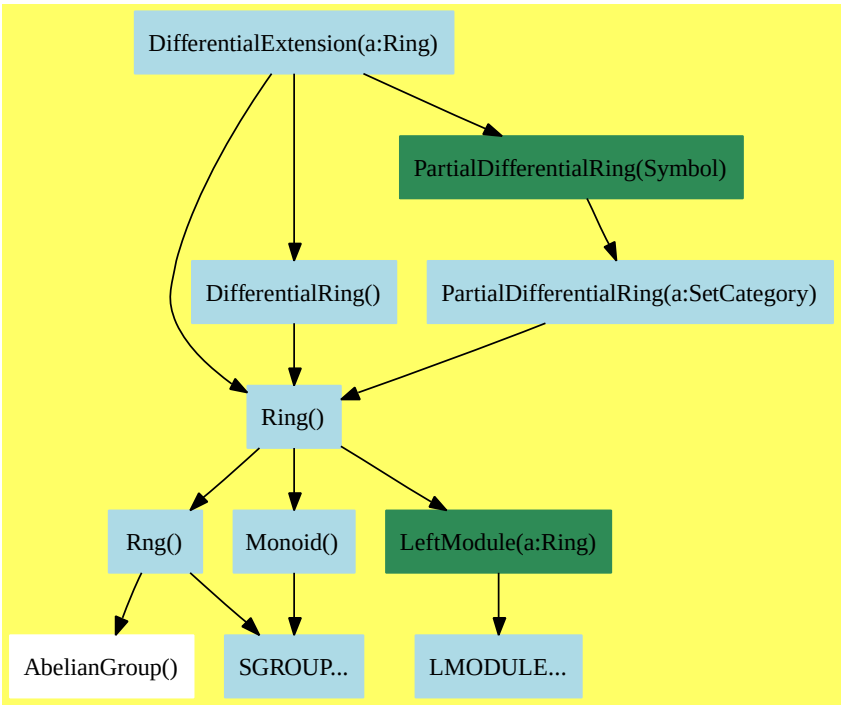
"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "ABELGRP..."

"ABELGRP..." [color=lightblue];
"REPSQ..." [color="#00EE00"];
```

```
"SETCAT..." [color=lightblue];
}
```

———-

11.0.153 DifferentialExtension (DIFEXT)



— DifferentialExtension.input —

```
)set break resume
)sys rm -f DifferentialExtension.output
)spool DifferentialExtension.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show DifferentialExtension
--R DifferentialExtension R: Ring  is a category constructor
```

```
--R Abbreviation for DifferentialExtension is DIFEXT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for DIFEXT
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R D : % -> % if R has DIFRING           D : (%,(R -> R)) -> %
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        coerce : Integer -> %
--R coerce : % -> OutputForm              hash : % -> SingleInteger
--R latex : % -> String                   one? : % -> Boolean
--R recip : % -> Union(%,"failed")        sample : () -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R D : (%,NonNegativeInteger) -> % if R has DIFRING
--R D : (%,Symbol) -> % if R has PDRING SYMBOL
--R D : (%,List Symbol) -> % if R has PDRING SYMBOL
--R D : (%,Symbol,NonNegativeInteger) -> % if R has PDRING SYMBOL
--R D : (%,List Symbol,List NonNegativeInteger) -> % if R has PDRING SYMBOL
--R D : (%,(R -> R),NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R differentiate : % -> % if R has DIFRING
--R differentiate : (%,NonNegativeInteger) -> % if R has DIFRING
--R differentiate : (%,Symbol) -> % if R has PDRING SYMBOL
--R differentiate : (%,List Symbol) -> % if R has PDRING SYMBOL
--R differentiate : (%,Symbol,NonNegativeInteger) -> % if R has PDRING SYMBOL
--R differentiate : (%,List Symbol,List NonNegativeInteger) -> % if R has PDRING SYMBOL
--R differentiate : (%,(R -> R),NonNegativeInteger) -> %
--R differentiate : (%,(R -> R)) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— DifferentialExtension.help —

```
=====================================================================
DifferentialExtension examples
=====================================================================
```

```
See Also:
o )show DifferentialExtension
```

———————-

See:

⇒ “ComplexCategory” (COMPCAT) 20.0.216 on page 1619
⇒ “DifferentialPolynomialCategory” (DPOLCAT) 17.0.192 on page 1306
⇒ “DirectProductCategory” (DIRPCAT) 12.0.160 on page 1001
⇒ “QuaternionCategory” (QUATCAT) 12.0.167 on page 1071
⇒ “QuotientFieldCategory” (QFCAT) 17.0.198 on page 1376
⇒ “SquareMatrixCategory” (SMATCAT) 12.0.168 on page 1082
⇒ “UnivariatePolynomialCategory” (UPOLYC) 17.0.204 on page 1471
⇐ “PartialDifferentialRing” (PDRING) 10.0.145 on page 885
⇐ “Ring” (RING) 9.0.129 on page 770

Exports:

1	0	characteristic	coerce	D
differentiate	hash	latex	one?	recip
sample	subtractIfCan	zero?	?*?	?~=?
?**?	?+?	?-?	-?	?=?
?^?				

Attributes exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return “failed” if its argument is not a unit.

These are directly exported but not implemented:

```
differentiate : (%,(R -> R)) -> %
```

These are implemented by this category:

```
D : (%,(R -> R)) -> %
D : (%,(R -> R),NonNegativeInteger) -> %
differentiate : % -> % if R has DIFRING
differentiate : (%,(R -> R),NonNegativeInteger) -> %
differentiate : (%,Symbol) -> % if R has PDRING SYMBOL
```

These exports come from (p770) Ring():

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
```

```
coerce : Integer -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
```

These exports come from (p848) DifferentialRing():

```
D : % -> % if R has DIFRING
D : (%,NonNegativeInteger) -> % if R has DIFRING
differentiate : (%,NonNegativeInteger) -> %
    if R has DIFRING
```

These exports come from (p885) PartialDifferentialRing(Symbol):

```
D : (%,Symbol) -> % if R has PDRING SYMBOL
D : (%,List Symbol) -> % if R has PDRING SYMBOL
D : (%,Symbol,NonNegativeInteger) -> %
    if R has PDRING SYMBOL
D : (%,List Symbol,List NonNegativeInteger) -> %
    if R has PDRING SYMBOL
differentiate : (%,List Symbol) -> %
    if R has PDRING SYMBOL
differentiate : (%,Symbol,NonNegativeInteger) -> %
    if R has PDRING SYMBOL
differentiate : (%,List Symbol,List NonNegativeInteger) -> %
    if R has PDRING SYMBOL
```

— category DIFEXT DifferentialExtension —

```
)abbrev category DIFEXT DifferentialExtension
```

```
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ Differential extensions of a ring R.
++ Given a differentiation on R, extend it to a differentiation on %.

DifferentialExtension(R:Ring): Category == Ring with
    differentiate: (%, R -> R) -> %
       ++ differentiate(x, deriv) differentiates x extending
       ++ the derivation deriv on R.
    differentiate: (%, R -> R, NonNegativeInteger) -> %
       ++ differentiate(x, deriv, n) differentiate x n times
       ++ using a derivation which extends deriv on R.
    D: (%, R -> R) -> %
       ++ D(x, deriv) differentiates x extending
       ++ the derivation deriv on R.
    D: (%, R -> R, NonNegativeInteger) -> %
       ++ D(x, deriv, n) differentiate x n times
       ++ using a derivation which extends deriv on R.
    if R has DifferentialRing then DifferentialRing
    if R has PartialDifferentialRing(Symbol) then
              PartialDifferentialRing(Symbol)
  add
    differentiate(x:%, derivation: R -> R, n:NonNegativeInteger):% ==
      for i in 1..n repeat x := differentiate(x, derivation)
      x
    D(x:%, derivation: R -> R) == differentiate(x, derivation)
    D(x:%, derivation: R -> R, n:NonNegativeInteger) ==
            differentiate(x, derivation, n)

    if R has DifferentialRing then
      differentiate x == differentiate(x, differentiate$R)

    if R has PartialDifferentialRing Symbol then
      differentiate(x:%, v:Symbol):% ==
        differentiate(x, s +-> differentiate(s, v)$R)
```

———-

— **DIFEXT.dotabb** —

```
"DIFEXT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=DIFEXT"];
"DIFEXT" -> "RING"
"DIFEXT" -> "DIFRING"
"DIFEXT" -> "PDRING"
```

———-

— DIFEXT.dotfull —

```
"DifferentialExtension(a:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=DIFEXT"];
"DifferentialExtension(a:Ring)" -> "Ring()"
"DifferentialExtension(a:Ring)" -> "DifferentialRing()"
"DifferentialExtension(a:Ring)" -> "PartialDifferentialRing(Symbol)"

"DifferentialExtension(IntegralDomain)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=DIFEXT"];
"DifferentialExtension(IntegralDomain)" ->
  "DifferentialExtension(a:Ring)"

"DifferentialExtension(CommutativeRing)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=DIFEXT"];
"DifferentialExtension(CommutativeRing)" ->
  "DifferentialExtension(a:Ring)"
```

———-

— DIFEXT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"DifferentialExtension(a:Ring)" [color=lightblue];
"DifferentialExtension(a:Ring)" -> "Ring()"
"DifferentialExtension(a:Ring)" -> "DifferentialRing()"
"DifferentialExtension(a:Ring)" -> "PartialDifferentialRing(Symbol)"

"PartialDifferentialRing(Symbol)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=PDRING"];
"PartialDifferentialRing(Symbol)" ->
    "PartialDifferentialRing(a:SetCategory)"

"PartialDifferentialRing(a:SetCategory)"
```

```
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PDRING"];
"PartialDifferentialRing(a:SetCategory)" -> "Ring()"

"DifferentialRing()" [color=lightblue];
"DifferentialRing()" -> "Ring()"

"Ring()" [color=lightblue];
"Ring()" -> "Rng()"
"Ring()" -> "Monoid()"
"Ring()" -> "LeftModule(a:Ring)"

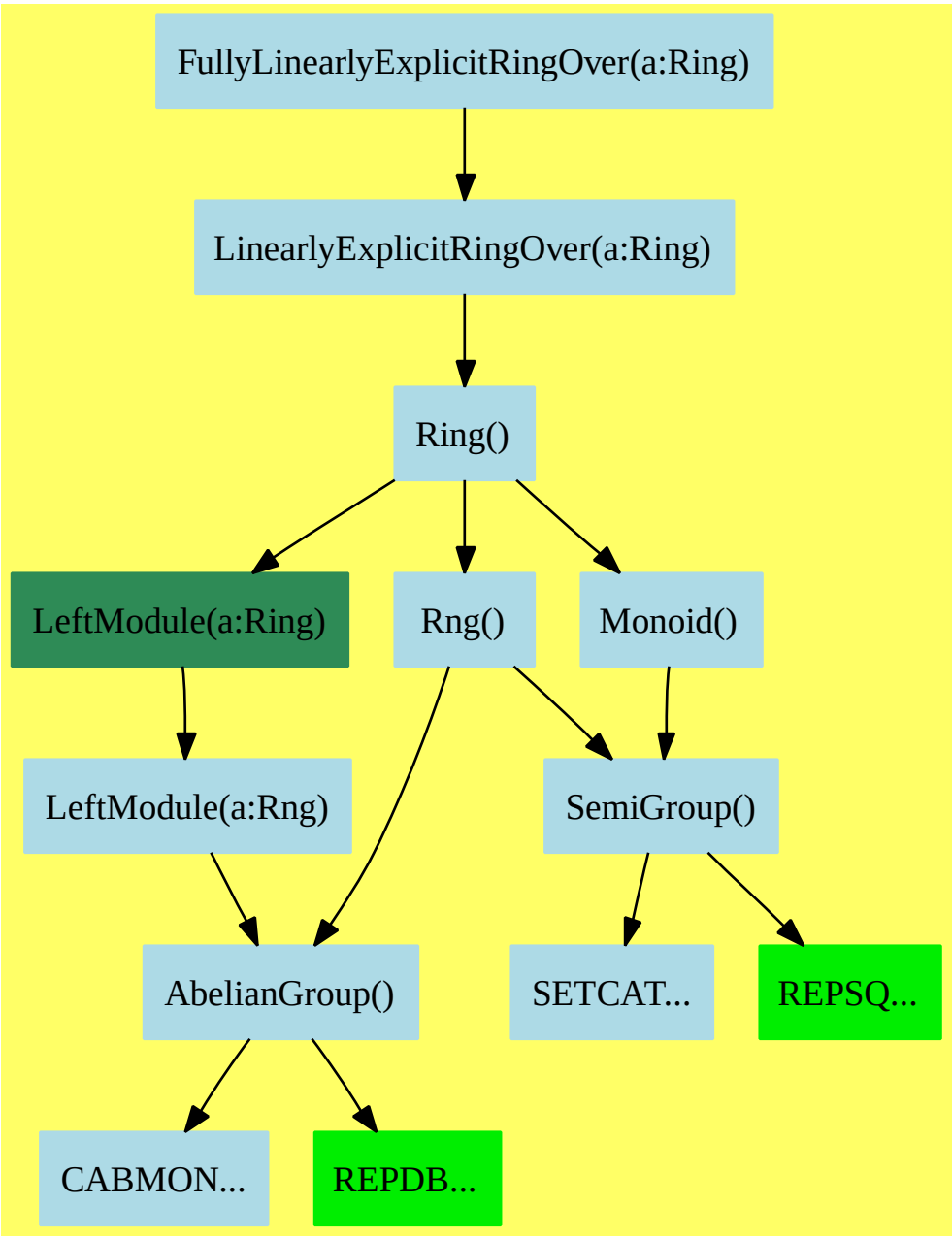
"Rng()" [color=lightblue];
"Rng()" -> "AbelianGroup()"
"Rng()" -> "SGROUP..."

"Monoid()" [color=lightblue];
"Monoid()" -> "SGROUP..."

"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LMODULE..."

"SGROUP..." [color=lightblue];
"LMODULE..." [color=lightblue];
}
```

11.0.154 FullyLinearlyExplicitRingOver (FLINEXP)



— **FullyLinearlyExplicitRingOver.input** —

```
)set break resume
)sys rm -f FullyLinearlyExplicitRingOver.output
)spool FullyLinearlyExplicitRingOver.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FullyLinearlyExplicitRingOver
--R FullyLinearlyExplicitRingOver R: Ring  is a category constructor
--R Abbreviation for FullyLinearlyExplicitRingOver is FLINEXP
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FLINEXP
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        coerce : Integer -> %
--R coerce : % -> OutputForm              hash : % -> SingleInteger
--R latex : % -> String                   one? : % -> Boolean
--R recip : % -> Union(%,"failed")        sample : () -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R reducedSystem : Matrix % -> Matrix Integer if R has LINEXP INT
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer) if R has LINEXP I
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix R,vec: Vector R)
--R reducedSystem : Matrix % -> Matrix R
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— FullyLinearlyExplicitRingOver.help —

```
=====================================================================
FullyLinearlyExplicitRingOver examples
=====================================================================

See Also:
```

```
o )show FullyLinearlyExplicitRingOver
```

See:

⇒ "ComplexCategory" (COMPCAT) 20.0.216 on page 1619
⇒ "DirectProductCategory" (DIRPCAT) 12.0.160 on page 1001
⇒ "FunctionSpace" (FS) 17.0.195 on page 1336
⇒ "MonogenicAlgebra" (MONOGEN) 19.0.214 on page 1598
⇒ "PolynomialCategory" (POLYCAT) 16.0.189 on page 1255
⇒ "QuaternionCategory" (QUATCAT) 12.0.167 on page 1071
⇒ "QuotientFieldCategory" (QFCAT) 17.0.198 on page 1376
⇒ "SquareMatrixCategory" (SMATCAT) 12.0.168 on page 1082
⇐ "LinearlyExplicitRingOver" (LINEXP) 10.0.142 on page 869

Exports:

1	0	characteristic	coerce	hash
latex	one?	recip	reducedSystem	sample
subtractIfCan	zero?	?*?	?**?	?+?
?-?	-?	?=?	?^?	?~=?

Attributes exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.

These are implemented by this category:

```
reducedSystem : Matrix % ->
    Matrix Integer if R has LINEXP INT
reducedSystem : (Matrix %,Vector %) ->
    Record(mat: Matrix Integer,vec: Vector Integer)
         if R has LINEXP INT
```

These exports come from (p869) LinearlyExplicitRingOver(a:Ring):

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : Integer -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
reducedSystem : (Matrix %,Vector %) ->
```

```
    Record(mat: Matrix R,vec: Vector R)
 reducedSystem : Matrix % -> Matrix R
 sample : () -> %
 subtractIfCan : (%,%) -> Union(%,"failed")
 zero? : % -> Boolean
 ?+? : (%,%) -> %
 ?=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
 ?*? : (%,%) -> %
 ?*? : (Integer,%) -> %
 ?*? : (PositiveInteger,%) -> %
 ?*? : (NonNegativeInteger,%) -> %
 ?-? : (%,%) -> %
 -? : % -> %
 ?**? : (%,PositiveInteger) -> %
 ?**? : (%,NonNegativeInteger) -> %
 ?^? : (%,PositiveInteger) -> %
 ?^? : (%,NonNegativeInteger) -> %
```

— category FLINEXP FullyLinearlyExplicitRingOver —

```
)abbrev category FLINEXP FullyLinearlyExplicitRingOver
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ S is \spadtype{FullyLinearlyExplicitRingOver R} means that S is a
++ \spadtype{LinearlyExplicitRingOver R} and, in addition, if R is a
++ \spadtype{LinearlyExplicitRingOver Integer}, then so is S

FullyLinearlyExplicitRingOver(R:Ring):Category ==
  LinearlyExplicitRingOver R with
    if (R has LinearlyExplicitRingOver Integer) then
            LinearlyExplicitRingOver Integer
 add
  if not(R is Integer) then
    if (R has LinearlyExplicitRingOver Integer) then
      reducedSystem(m:Matrix %):Matrix(Integer) ==
        reducedSystem(reducedSystem(m)@Matrix(R))

      reducedSystem(m:Matrix %, v:Vector %):
        Record(mat:Matrix(Integer), vec:Vector(Integer)) ==
          rec := reducedSystem(m, v)@Record(mat:Matrix R, vec:Vector R)
```

```
          reducedSystem(rec.mat, rec.vec)
```

———-

— **FLINEXP.dotabb** —

```
"FLINEXP"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FLINEXP"];
"FLINEXP" -> "LINEXP"
```

———-

— **FLINEXP.dotfull** —

```
"FullyLinearlyExplicitRingOver(a:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FLINEXP"];
"FullyLinearlyExplicitRingOver(a:Ring)" ->
     "LinearlyExplicitRingOver(a:Ring)"

"FullyLinearlyExplicitRingOver(a:CommutativeRing)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=FLINEXP"];
"FullyLinearlyExplicitRingOver(a:CommutativeRing)" ->
    "FullyLinearlyExplicitRingOver(a:Ring)"

"FullyLinearlyExplicitRingOver(IntegralDomain)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=FLINEXP"];
"FullyLinearlyExplicitRingOver(IntegralDomain)" ->
    "FullyLinearlyExplicitRingOver(a:Ring)"
```

———-

— **FLINEXP.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FullyLinearlyExplicitRingOver(a:Ring)" [color=lightblue];
"FullyLinearlyExplicitRingOver(a:Ring)" ->
     "LinearlyExplicitRingOver(a:Ring)"

"LinearlyExplicitRingOver(a:Ring)" [color=lightblue];
"LinearlyExplicitRingOver(a:Ring)" -> "Ring()"
```

```
"Ring()" [color=lightblue];
"Ring()" -> "Rng()"
"Ring()" -> "Monoid()"
"Ring()" -> "LeftModule(a:Ring)"

"Rng()" [color=lightblue];
"Rng()" -> "AbelianGroup()"
"Rng()" -> "SemiGroup()"

"Monoid()" [color=lightblue];
"Monoid()" -> "SemiGroup()"

"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "AbelianGroup()"

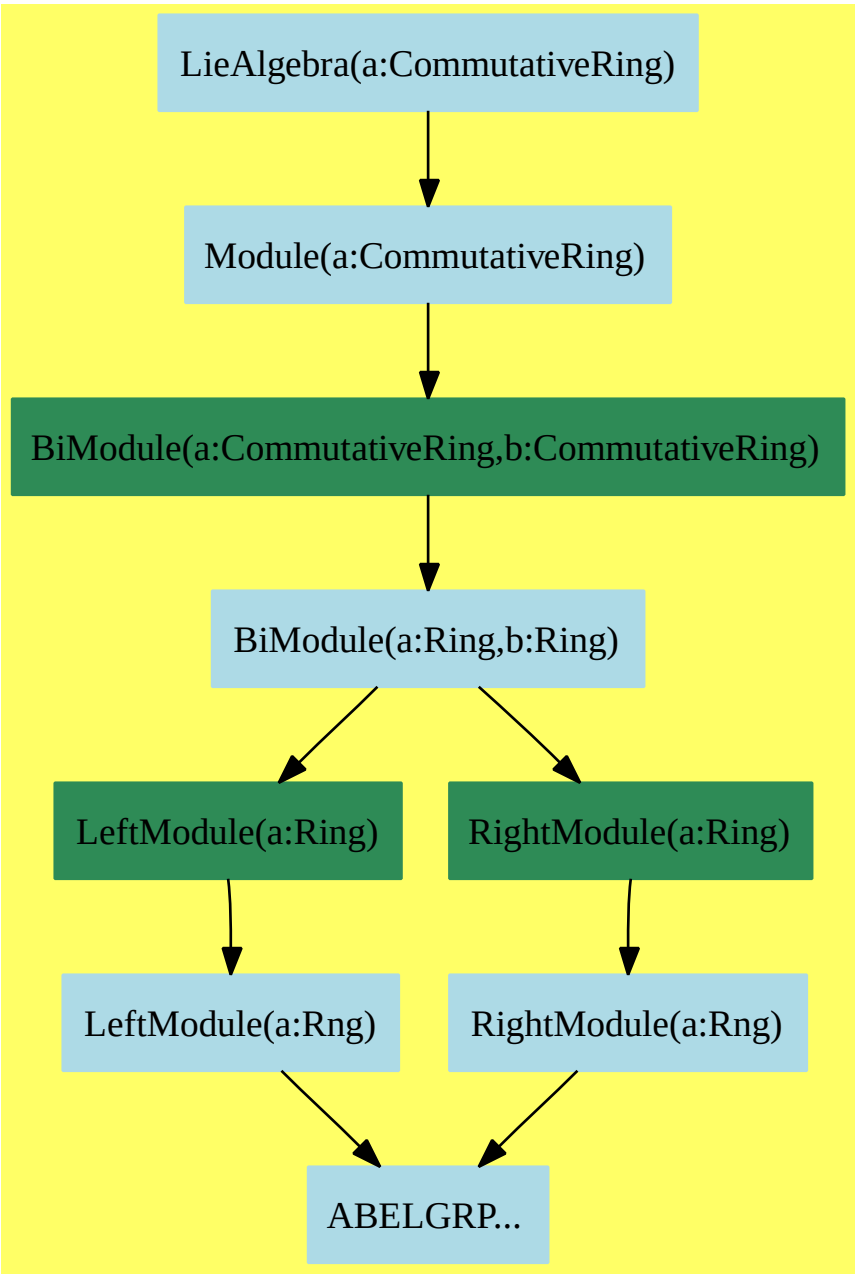
"AbelianGroup()" [color=lightblue];
"AbelianGroup()" -> "CABMON..."
"AbelianGroup()" -> "REPDB..."

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SETCAT..."
"SemiGroup()" -> "REPSQ..."

"REPDB..." [color="#00EE00"];
"REPSQ..." [color="#00EE00"];
"SETCAT..." [color=lightblue];
"CABMON..." [color=lightblue];
}
```

———————-

11.0.155 LieAlgebra (LIECAT)



— **LieAlgebra.input** —

```
)set break resume
)sys rm -f LieAlgebra.output
)spool LieAlgebra.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show LieAlgebra
--R LieAlgebra R: CommutativeRing  is a category constructor
--R Abbreviation for LieAlgebra is LIECAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for LIECAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,R) -> %                    ?*? : (R,%) -> %
--R ?*? : (Integer,%) -> %              ?*? : (PositiveInteger,%) -> %
--R ?+? : (%,%) -> %                    ?-? : (%,%) -> %
--R -? : % -> %                         ?=? : (%,%) -> Boolean
--R 0 : () -> %                         coerce : % -> OutputForm
--R construct : (%,%) -> %              hash : % -> SingleInteger
--R latex : % -> String                 sample : () -> %
--R zero? : % -> Boolean                ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?/? : (%,R) -> % if R has FIELD
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— LieAlgebra.help —

```
=====================================================================
LieAlgebra examples
=====================================================================

See Also:
o )show LieAlgebra
```

———-

See:

⇒ “FreeLieAlgebra” (FLALG) 12.0.163 on page 1041

⇐ "Module" (MODULE) 10.0.143 on page 874

Exports:

0	coerce	construct	hash	latex
sample	subtractIfCan	zero?	?~=?	?/?
?*?	?+?	?-?	-?	?=?

Attributes Exported:

- **NullSquare** means that $[x, x] = 0$ holds. See `LieAlgebra`.
- **JacobiIdentity** means that $[x, [y, z]] + [y, [z, x]] + [z, [x, y]] = 0$ holds. See `LieAlgebra`.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
construct : (%,%) -> %
```

These are implemented by this category:

```
?/? : (%,R) -> % if R has FIELD
```

These exports come from (p874) Module(R:Ring):

```
0 : () -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (%,R) -> %
?*? : (R,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?+? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
?=? : (%,%) -> Boolean
```

— category LIECAT LieAlgebra —

```
)abbrev category LIECAT LieAlgebra
++ Author: Michel Petitot (petitot@lifl.fr).
++ Date Created: 91
++ Date Last Updated: 7 Juillet 92
++ Keywords:
++ References:
++ Description:
++ The category of Lie Algebras.
++ It is used by the domains of non-commutative algebra,
++ LiePolynomial and XPBWPolynomial.

LieAlgebra(R: CommutativeRing): Category ==  Module(R) with
    construct:  ($,$) -> $
      ++ \axiom{construct(x,y)} returns the Lie bracket of \axiom{x}
      ++ and \axiom{y}.
    NullSquare
      ++ \axiom{NullSquare} means that \axiom{[x,x] = 0} holds.
    JacobiIdentity
      ++ \axiom{JacobiIdentity} means that
      ++ \axiom{[x,[y,z]]+[y,[z,x]]+[z,[x,y]] = 0} holds.
    if R has Field then
       "/"   :  ($,R) -> $
         ++ \axiom{x/r} returns the division of \axiom{x} by \axiom{r}.
  add
    if R has Field then x / r == inv(r)$R * x
```

———-

— LIECAT.dotabb —

```
"LIECAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=LIECAT"];
"LIECAT" -> "MODULE"
```

———-

— LIECAT.dotfull —

```
"LieAlgebra(a:CommutativeRing)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=LIECAT"];
"LieAlgebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"
```

———-

— LIECAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"LieAlgebra(a:CommutativeRing)" [color=lightblue];
"LieAlgebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"

"Module(a:CommutativeRing)" [color=lightblue];
"Module(a:CommutativeRing)" ->
  "BiModule(a:CommutativeRing,b:CommutativeRing)"

"BiModule(a:CommutativeRing,b:CommutativeRing)" [color=seagreen];
"BiModule(a:CommutativeRing,b:CommutativeRing)" -> "BiModule(a:Ring,b:Ring)"

"BiModule(a:Ring,b:Ring)" [color=lightblue];
"BiModule(a:Ring,b:Ring)" -> "LeftModule(a:Ring)"
"BiModule(a:Ring,b:Ring)" -> "RightModule(a:Ring)"

"RightModule(a:Ring)" [color=seagreen];
"RightModule(a:Ring)" -> "RightModule(a:Rng)"

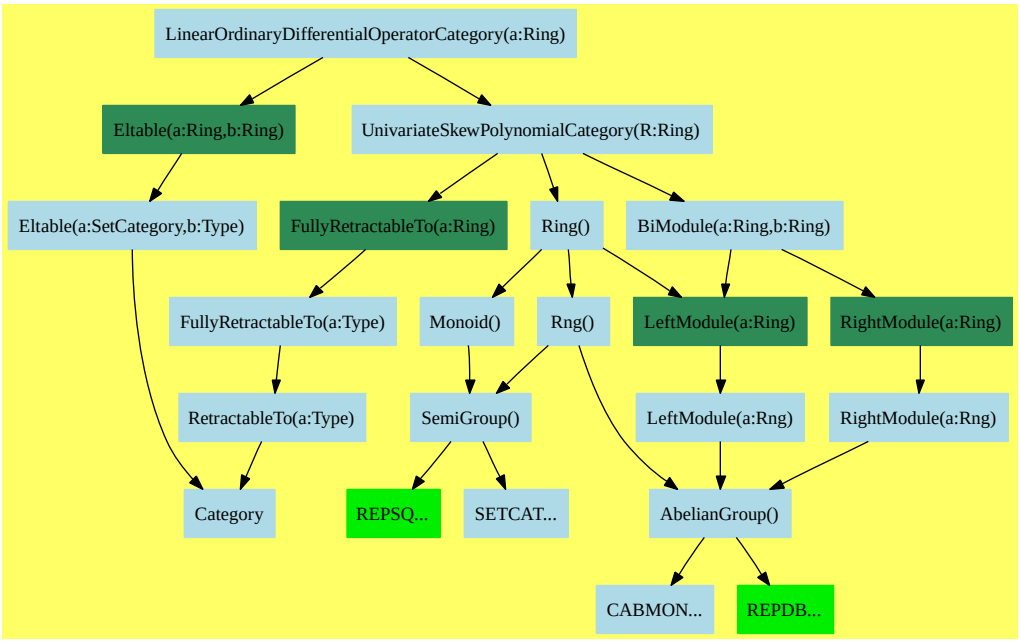
"RightModule(a:Rng)" [color=lightblue];
"RightModule(a:Rng)" -> "ABELGRP..."

"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "ABELGRP..."

"ABELGRP..." [color=lightblue];
}
```

11.0.156 LinearOrdinaryDifferentialOperatorCategory (LODOCAT)



— LinearOrdinaryDifferentialOperatorCategory.input —

```
)set break resume
)sys rm -f LinearOrdinaryDifferentialOperatorCategory.output
)spool LinearOrdinaryDifferentialOperatorCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show LinearOrdinaryDifferentialOperatorCategory
--R LinearOrdinaryDifferentialOperatorCategory A: Ring  is a category constructor
--R Abbreviation for LinearOrdinaryDifferentialOperatorCategory is LODOCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for LODOCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (A,%) -> %                      ?*? : (%,A) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R D : () -> %                           1 : () -> %
```

```
--R 0 : () -> %                          ?^? : (%,PositiveInteger) -> %
--R adjoint : % -> %                     apply : (%,A,A) -> A
--R coefficients : % -> List A           coerce : A -> %
--R coerce : Integer -> %                coerce : % -> OutputForm
--R degree : % -> NonNegativeInteger     ?.? : (%,A) -> A
--R hash : % -> SingleInteger            latex : % -> String
--R leadingCoefficient : % -> A          one? : % -> Boolean
--R recip : % -> Union(%,"failed")       reductum : % -> %
--R retract : % -> A                     sample : () -> %
--R zero? : % -> Boolean                 ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R coefficient : (%,NonNegativeInteger) -> A
--R coerce : Fraction Integer -> % if A has RETRACT FRAC INT
--R content : % -> A if A has GCDDOM
--R directSum : (%,%) -> % if A has FIELD
--R exquo : (%,A) -> Union(%,"failed") if A has INTDOM
--R leftDivide : (%,%) -> Record(quotient: %,remainder: %) if A has FIELD
--R leftExactQuotient : (%,%) -> Union(%,"failed") if A has FIELD
--R leftExtendedGcd : (%,%) -> Record(coef1: %,coef2: %,generator: %) if A has FIELD
--R leftGcd : (%,%) -> % if A has FIELD
--R leftLcm : (%,%) -> % if A has FIELD
--R leftQuotient : (%,%) -> % if A has FIELD
--R leftRemainder : (%,%) -> % if A has FIELD
--R minimumDegree : % -> NonNegativeInteger
--R monicLeftDivide : (%,%) -> Record(quotient: %,remainder: %) if A has INTDOM
--R monicRightDivide : (%,%) -> Record(quotient: %,remainder: %) if A has INTDOM
--R monomial : (A,NonNegativeInteger) -> %
--R primitivePart : % -> % if A has GCDDOM
--R retract : % -> Fraction Integer if A has RETRACT FRAC INT
--R retract : % -> Integer if A has RETRACT INT
--R retractIfCan : % -> Union(A,"failed")
--R retractIfCan : % -> Union(Fraction Integer,"failed") if A has RETRACT FRAC INT
--R retractIfCan : % -> Union(Integer,"failed") if A has RETRACT INT
--R rightDivide : (%,%) -> Record(quotient: %,remainder: %) if A has FIELD
--R rightExactQuotient : (%,%) -> Union(%,"failed") if A has FIELD
--R rightExtendedGcd : (%,%) -> Record(coef1: %,coef2: %,generator: %) if A has FIELD
--R rightGcd : (%,%) -> % if A has FIELD
--R rightLcm : (%,%) -> % if A has FIELD
--R rightQuotient : (%,%) -> % if A has FIELD
--R rightRemainder : (%,%) -> % if A has FIELD
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R symmetricPower : (%,NonNegativeInteger) -> % if A has FIELD
--R symmetricProduct : (%,%) -> % if A has FIELD
--R symmetricSquare : % -> % if A has FIELD
--R
--E 1
```

```
)spool
)lisp (bye)
```

———-

— **LinearOrdinaryDifferentialOperatorCategory.help** —

```
====================================================================
LinearOrdinaryDifferentialOperatorCategory examples
====================================================================

See Also:
o )show LinearOrdinaryDifferentialOperatorCategory
```

———-

See:

⇐ “Eltable” (ELTAB) 2.0.10 on page 29
⇐ “UnivariateSkewPolynomialCategory” (OREPCAT) 10.0.150 on page 929

Exports:

0	1	adjoint
apply	characteristic	coefficient
coefficients	coerce	content
D	degree	directSum
exquo	hash	latex
leadingCoefficient	leftDivide	leftExactQuotient
leftExtendedGcd	leftGcd	leftLcm
leftQuotient	leftRemainder	minimumDegree
monicLeftDivide	monicRightDivide	monomial
one?	primitivePart	recip
reductum	retract	retractIfCan
rightDivide	rightExactQuotient	rightExtendedGcd
rightGcd	rightLcm	rightQuotient
rightRemainder	sample	subtractIfCan
symmetricPower	symmetricProduct	symmetricSquare
zero?	?^?	?.?
?~=?	?*?	?**?
?+?	?-?	-?
?=?		

Attributes Exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return “failed” if its argument is not a unit.

- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
directSum : (%,%) -> % if A has FIELD
symmetricPower : (%,NonNegativeInteger) -> % if A has FIELD
symmetricProduct : (%,%) -> % if A has FIELD
```

These are implemented by this category:

```
adjoint : % -> %
D : () -> %
symmetricSquare : % -> % if A has FIELD
```

These exports come from (p929) UnivariateSkewPolynomialCategory(A:Ring):

```
0 : () -> %
1 : () -> %
apply : (%,A,A) -> A
characteristic : () -> NonNegativeInteger
coefficient : (%,NonNegativeInteger) -> A
coefficients : % -> List A
coerce : % -> OutputForm
coerce : Integer -> %
coerce : A -> %
coerce : Fraction Integer -> % if A has RETRACT FRAC INT
content : % -> A if A has GCDDOM
degree : % -> NonNegativeInteger
exquo : (%,A) -> Union(%,"failed") if A has INTDOM
hash : % -> SingleInteger
latex : % -> String
leadingCoefficient : % -> A
leftDivide : (%,%) -> Record(quotient: %,remainder: %)
   if A has FIELD
leftExactQuotient : (%,%) -> Union(%,"failed") if A has FIELD
leftExtendedGcd : (%,%) -> Record(coef1: %,coef2: %,generator: %)
   if A has FIELD
leftGcd : (%,%) -> % if A has FIELD
leftLcm : (%,%) -> % if A has FIELD
leftQuotient : (%,%) -> % if A has FIELD
leftRemainder : (%,%) -> % if A has FIELD
minimumDegree : % -> NonNegativeInteger
monicLeftDivide : (%,%) -> Record(quotient: %,remainder: %)
   if A has INTDOM
monicRightDivide : (%,%) -> Record(quotient: %,remainder: %)
   if A has INTDOM
monomial : (A,NonNegativeInteger) -> %
```

```
one? : % -> Boolean
primitivePart : % -> % if A has GCDDOM
recip : % -> Union(%,"failed")
reductum : % -> %
retract : % -> A
retract : % -> Fraction Integer if A has RETRACT FRAC INT
retract : % -> Integer if A has RETRACT INT
retractIfCan : % -> Union(Fraction Integer,"failed")
   if A has RETRACT FRAC INT
retractIfCan : % -> Union(Integer,"failed")
   if A has RETRACT INT
retractIfCan : % -> Union(A,"failed")
rightDivide : (%,%) -> Record(quotient: %,remainder: %)
   if A has FIELD
rightExactQuotient : (%,%) -> Union(%,"failed") if A has FIELD
rightExtendedGcd :
  (%,%) -> Record(coef1: %,coef2: %,generator: %) if A has FIELD
rightGcd : (%,%) -> % if A has FIELD
rightLcm : (%,%) -> % if A has FIELD
rightQuotient : (%,%) -> % if A has FIELD
rightRemainder : (%,%) -> % if A has FIELD
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?**? : (%,NonNegativeInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?+? : (%,%) -> %
?*? : (%,A) -> %
?*? : (A,%) -> %
?~=? : (%,%) -> Boolean
?=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (Integer,%) -> %
?*? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?^? : (%,PositiveInteger) -> %
```

These exports come from (p29) Eltable(A:Ring,A:Ring):

```
?.? : (%,A) -> A
```

— category LODOCAT LinearOrdinaryDifferentialOperatorCategory —

```
)abbrev category LODOCAT LinearOrdinaryDifferentialOperatorCategory
```

```
++ Author: Manuel Bronstein
++ Date Created: 9 December 1993
++ Date Last Updated: 15 April 1994
++ Keywords: differential operator
++ Description:
++ LinearOrdinaryDifferentialOperatorCategory is the category
++ of differential operators with coefficients in a ring A with a given
++ derivation.
++
++ Multiplication of operators corresponds to functional composition:\br
++ (L1 * L2).(f) = L1 L2 f

LinearOrdinaryDifferentialOperatorCategory(A:Ring): Category ==
  Join(UnivariateSkewPolynomialCategory A, Eltable(A, A)) with
        D: () -> %
            ++ D() provides the operator corresponding to a derivation
            ++ in the ring \spad{A}.
        adjoint: % -> %
            ++ adjoint(a) returns the adjoint operator of a.
        if A has Field then
          symmetricProduct: (%, %) -> %
            ++ symmetricProduct(a,b) computes an operator \spad{c} of
            ++ minimal order such that the nullspace of \spad{c} is
            ++ generated by all the products of a solution of \spad{a} by
            ++ a solution of \spad{b}.
          symmetricPower  : (%, NonNegativeInteger) -> %
            ++ symmetricPower(a,n) computes an operator \spad{c} of
            ++ minimal order such that the nullspace of \spad{c} is
            ++ generated by all the products of \spad{n} solutions
            ++ of \spad{a}.
          symmetricSquare : % -> %
            ++ symmetricSquare(a) computes \spad{symmetricProduct(a,a)}
            ++ using a more efficient method.
          directSum: (%, %) -> %
            ++ directSum(a,b) computes an operator \spad{c} of
            ++ minimal order such that the nullspace of \spad{c} is
            ++ generated by all the sums of a solution of \spad{a} by
            ++ a solution of \spad{b}.
   add
        m1monom: NonNegativeInteger -> %

        D() == monomial(1, 1)

        m1monom n ==
          a:A := (odd? n => -1; 1)
          monomial(a, n)

        adjoint a ==
          ans:% := 0
          while a ^= 0 repeat
```

```
            ans := ans + m1monom(degree a) * leadingCoefficient(a)::%
            a   := reductum a
          ans

        if A has Field then symmetricSquare l == symmetricPower(l, 2)
```

———-

— **LODOCAT.dotabb** —

```
"LODOCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=LODOCAT"];
"LODOCAT" -> "ELTAB"
"LODOCAT" -> "OREPCAT"
```

———-

— **LODOCAT.dotfull** —

```
"LinearOrdinaryDifferentialOperatorCategory(a:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=LODOCAT"];
"LinearOrdinaryDifferentialOperatorCategory(a:Ring)"
  -> "Eltable(a:Ring,b:Ring)"
"LinearOrdinaryDifferentialOperatorCategory(a:Ring)"
  -> "UnivariateSkewPolynomialCategory(R:Ring)"
```

———-

— **LODOCAT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"LinearOrdinaryDifferentialOperatorCategory(a:Ring)" [color=lightblue];
"LinearOrdinaryDifferentialOperatorCategory(a:Ring)"
  -> "Eltable(a:Ring,b:Ring)"
"LinearOrdinaryDifferentialOperatorCategory(a:Ring)"
  -> "UnivariateSkewPolynomialCategory(R:Ring)"

"Eltable(a:Ring,b:Ring)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=ELTAB"];
"Eltable(a:Ring,b:Ring)" ->
   "Eltable(a:SetCategory,b:Type)"
```

```
"Eltable(a:SetCategory,b:Type)" [color=lightblue];
"Eltable(a:SetCategory,b:Type)" -> "Category"

"UnivariateSkewPolynomialCategory(R:Ring)" [color=lightblue];
"UnivariateSkewPolynomialCategory(R:Ring)"
  -> "BiModule(a:Ring,b:Ring)"
"UnivariateSkewPolynomialCategory(R:Ring)"
  -> "FullyRetractableTo(a:Ring)"
"UnivariateSkewPolynomialCategory(R:Ring)"
  -> "Ring()"

"FullyRetractableTo(a:Ring)" [color=seagreen];
"FullyRetractableTo(a:Ring)" -> "FullyRetractableTo(a:Type)"

"FullyRetractableTo(a:Type)" [color=lightblue];
"FullyRetractableTo(a:Type)" -> "RetractableTo(a:Type)"

"RetractableTo(a:Type)" [color=lightblue];
"RetractableTo(a:Type)" -> "Category"

"Category" [color=lightblue];

"Ring()" [color=lightblue];
"Ring()" -> "Rng()"
"Ring()" -> "Monoid()"
"Ring()" -> "LeftModule(a:Ring)"

"Rng()" [color=lightblue];
"Rng()" -> "AbelianGroup()"
"Rng()" -> "SemiGroup()"

"Monoid()" [color=lightblue];
"Monoid()" -> "SemiGroup()"

"BiModule(a:Ring,b:Ring)" [color=lightblue];
"BiModule(a:Ring,b:Ring)" -> "LeftModule(a:Ring)"
"BiModule(a:Ring,b:Ring)" -> "RightModule(a:Ring)"

"RightModule(a:Ring)" [color=seagreen];
"RightModule(a:Ring)" -> "RightModule(a:Rng)"

"RightModule(a:Rng)" [color=lightblue];
"RightModule(a:Rng)" -> "AbelianGroup()"

"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

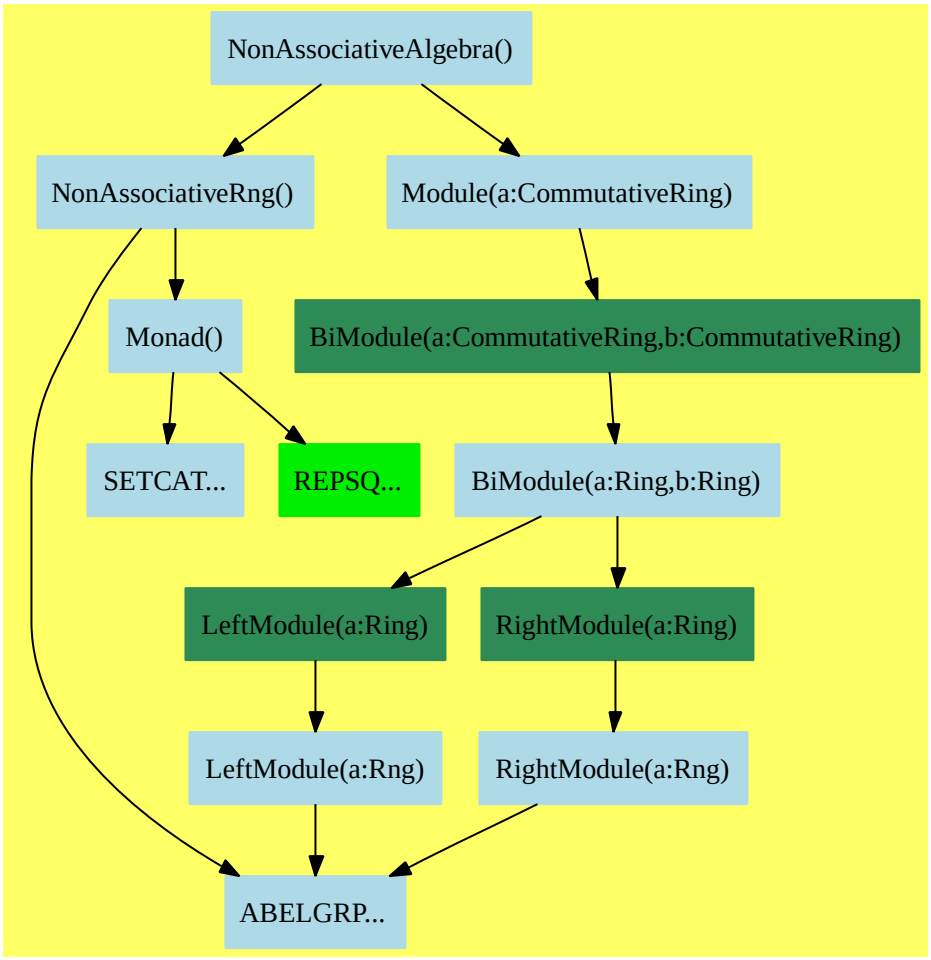
"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "AbelianGroup()"
```

```
"AbelianGroup()" [color=lightblue];
"AbelianGroup()" -> "CABMON..."
"AbelianGroup()" -> "REPDB..."

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SETCAT..."
"SemiGroup()" -> "REPSQ..."

"REPDB..." [color="#00EE00"];
"REPSQ..." [color="#00EE00"];
"SETCAT..." [color=lightblue];
"CABMON..." [color=lightblue];
}
```

11.0.157 NonAssociativeAlgebra (NAALG)



— NonAssociativeAlgebra.input —

```
)set break resume
)sys rm -f NonAssociativeAlgebra.output
)spool NonAssociativeAlgebra.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show NonAssociativeAlgebra
--R NonAssociativeAlgebra R: CommutativeRing  is a category constructor
```

```
--R Abbreviation for NonAssociativeAlgebra is NAALG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for NAALG
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (R,%) -> %                      ?*? : (%,R) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 0 : () -> %                           antiCommutator : (%,%) -> %
--R associator : (%,%,%) -> %             coerce : % -> OutputForm
--R commutator : (%,%) -> %               hash : % -> SingleInteger
--R latex : % -> String                   sample : () -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R leftPower : (%,PositiveInteger) -> %
--R plenaryPower : (%,PositiveInteger) -> %
--R rightPower : (%,PositiveInteger) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **NonAssociativeAlgebra.help** —

```
=====================================================================
NonAssociativeAlgebra examples
=====================================================================

See Also:
o )show NonAssociativeAlgebra
```

———-

See:

⇒ “FiniteRankNonAssociativeAlgebra” (FINAALG) 12.0.162 on page 1019
⇐ “Module” (MODULE) 10.0.143 on page 874
⇐ “NonAssociativeRng” (NARNG) 8.0.116 on page 679

Exports:

0	antiCommutator	associator	coerce	commutator
hash	latex	leftPower	plenaryPower	rightPower
sample	subtractIfCan	zero?	?~=?	?*?
?**?	?+?	?-?	-?	?=?

Attributes exported:

- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are implemented by this category:

```
plenaryPower : (%,PositiveInteger) -> %
```

These exports come from (p679) NonAssociativeRng():

```
0 : () -> %
antiCommutator : (%,%) -> %
associator : (%,%,%) -> %
coerce : % -> OutputForm
commutator : (%,%) -> %
hash : % -> SingleInteger
latex : % -> String
leftPower : (%,PositiveInteger) -> %
rightPower : (%,PositiveInteger) -> %
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?~=? : (%,%) -> Boolean
?*? : (PositiveInteger,%) -> %
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?*? : (Integer,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?*? : (%,%) -> %
?**? : (%,PositiveInteger) -> %
```

These exports come from (p874) Module(R:CommutativeRing):

```
?*? : (R,%) -> %
?*? : (%,R) -> %
```

— category NAALG NonAssociativeAlgebra —

```
)abbrev category NAALG NonAssociativeAlgebra
```

```
++ Author: J. Grabmeier, R. Wisbauer
++ Date Created: 01 March 1991
++ Date Last Updated: 11 June 1991
++ Basic Operations: +, -, *, **
++ Related Constructors: Algebra
++ Also See:
++ AMS Classifications:
++ Keywords: nonassociative algebra
++ Reference:
++ R.D. Schafer: An Introduction to Nonassociative Algebras
++ Academic Press, New York, 1966
++ Description:
++ NonAssociativeAlgebra is the category of non associative algebras
++ (modules which are themselves non associative rngs).\br
++
++ Axioms\br
++ \tab{5}r*(a*b) = (r*a)*b = a*(r*b)

NonAssociativeAlgebra(R:CommutativeRing): Category == _
  Join(NonAssociativeRng, Module R) with
    plenaryPower : (%,PositiveInteger) -> %
      ++ plenaryPower(a,n) is recursively defined to be
      ++ \spad{plenaryPower(a,n-1)*plenaryPower(a,n-1)} for \spad{n>1}
      ++ and \spad{a} for \spad{n=1}.
  add
    plenaryPower(a,n) ==
--      one? n => a
      ( n = 1 ) => a
      n1 : PositiveInteger := (n-1)::NonNegativeInteger::PositiveInteger
      plenaryPower(a,n1) * plenaryPower(a,n1)
```

———-

— NAALG.dotabb —

```
"NAALG"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=NAALG"];
"NAALG" -> "NARNG"
"NAALG" -> "MODULE"
```

———-

— NAALG.dotfull —

```
"NonAssociativeAlgebra(a:CommutativeRing)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=NAALG"];
```

```
"NonAssociativeAlgebra(a:CommutativeRing)" -> "NonAssociativeRng()"
"NonAssociativeAlgebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"
```

———-

— NAALG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"NonAssociativeAlgebra(a:CommutativeRing)" [color=lightblue];
"NonAssociativeAlgebra(a:CommutativeRing)" -> "NonAssociativeRng()"
"NonAssociativeAlgebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"

"NonAssociativeRng()" [color=lightblue];
"NonAssociativeRng()" -> "ABELGRP..."
"NonAssociativeRng()" -> "Monad()"

"Monad()" [color=lightblue];
"Monad()" -> "SETCAT..."
"Monad()" -> "REPSQ..."

"Module(a:CommutativeRing)" [color=lightblue];
"Module(a:CommutativeRing)" ->
  "BiModule(a:CommutativeRing,b:CommutativeRing)"

"BiModule(a:CommutativeRing,b:CommutativeRing)" [color=seagreen];
"BiModule(a:CommutativeRing,b:CommutativeRing)" -> "BiModule(a:Ring,b:Ring)"

"BiModule(a:Ring,b:Ring)" [color=lightblue];
"BiModule(a:Ring,b:Ring)" -> "LeftModule(a:Ring)"
"BiModule(a:Ring,b:Ring)" -> "RightModule(a:Ring)"

"RightModule(a:Ring)" [color=seagreen];
"RightModule(a:Ring)" -> "RightModule(a:Rng)"

"RightModule(a:Rng)" [color=lightblue];
"RightModule(a:Rng)" -> "ABELGRP..."

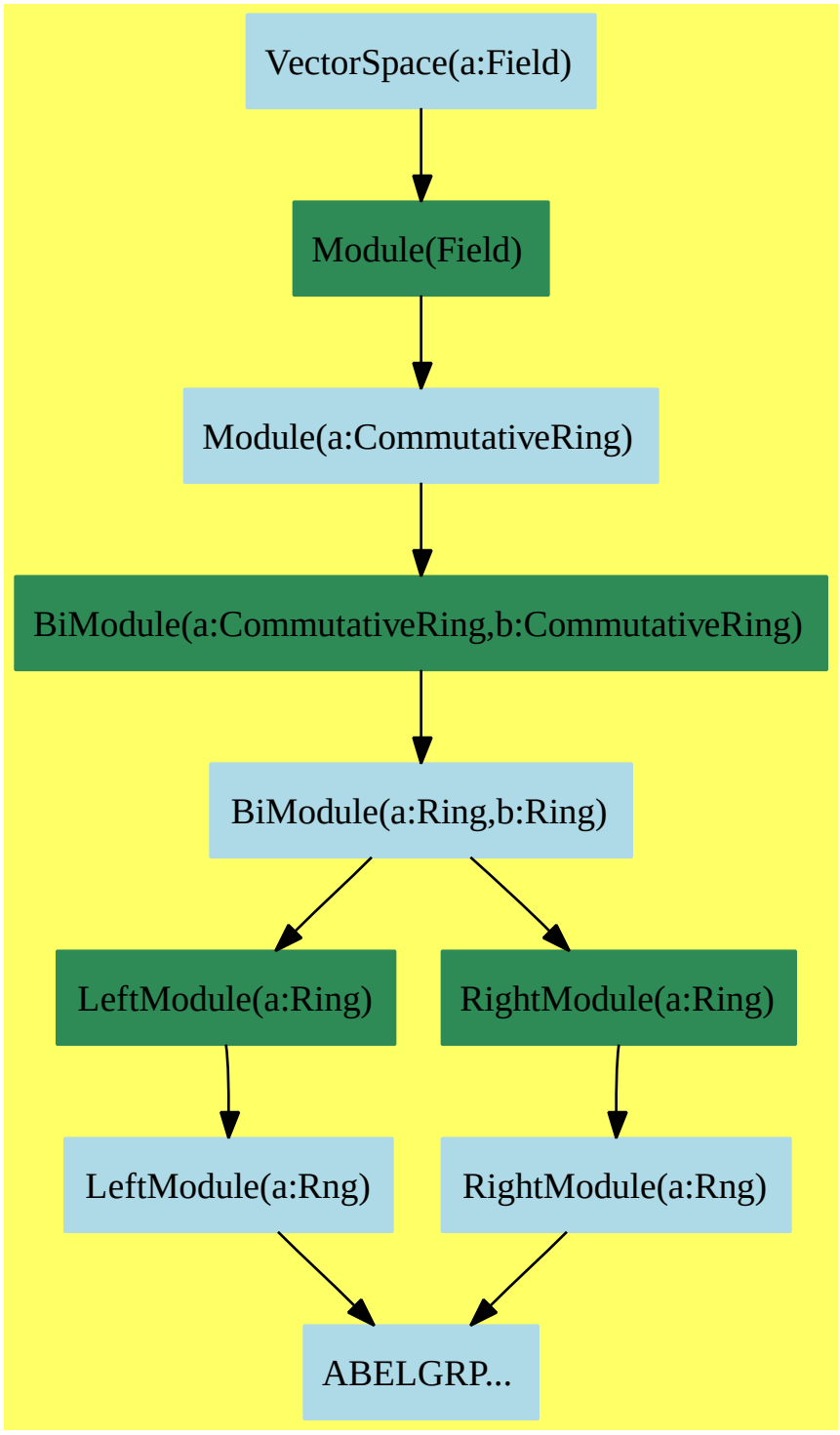
"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "ABELGRP..."

"REPSQ..." [color="#00EE00"];
```

```
"SETCAT..." [color=lightblue];
"ABELGRP..." [color=lightblue];
}
```

11.0.158 VectorSpace (VSPACE)



— VectorSpace.input —

```
)set break resume
)sys rm -f VectorSpace.output
)spool VectorSpace.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show VectorSpace
--R VectorSpace S: Field  is a category constructor
--R Abbreviation for VectorSpace is VSPACE
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for VSPACE
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (%,S) -> %                      ?*? : (S,%) -> %
--R ?*? : (Integer,%) -> %                ?*? : (PositiveInteger,%) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?/? : (%,S) -> %
--R ?=? : (%,%) -> Boolean                0 : () -> %
--R coerce : % -> OutputForm              dimension : () -> CardinalNumber
--R hash : % -> SingleInteger             latex : % -> String
--R sample : () -> %                      zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— VectorSpace.help —

```
=====================================================================
VectorSpace examples
=====================================================================

See Also:
o )show VectorSpace
```

———-

See:

⇒ "ExtensionField" (XF) 18.0.206 on page 1510
⇐ "Module" (MODULE) 10.0.143 on page 874

Exports:

0	coerce	dimension	hash	latex
sample	subtractIfCan	zero?	?~=?	?*?
?+?	?-?	-?	?/?	?=?

Attributes exported:

- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
dimension : () -> CardinalNumber
```

These are implemented by this category:

```
?/? : (%,S) -> %
```

These exports come from (p874) Module():

```
?*? : (%,S) -> %
0 : () -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (S,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?+? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
?=? : (%,%) -> Boolean
```

— category VSPACE VectorSpace —

```
)abbrev category VSPACE VectorSpace
```

```
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ Vector Spaces (not necessarily finite dimensional) over a field.

VectorSpace(S:Field): Category ==  Module(S) with
    "/"      : (%, S) -> %
      ++ x/y divides the vector x by the scalar y.
    dimension: () -> CardinalNumber
      ++ dimension() returns the dimensionality of the vector space.
  add
    (v:% / s:S):% == inv(s) * v
```

———-

— VSPACE.dotabb —

```
"VSPACE"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=VSPACE"];
"VSPACE" -> "MODULE"
```

———-

— VSPACE.dotfull —

```
"VectorSpace(a:Field)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=VSPACE"];
"VectorSpace(a:Field)" -> "Module(Field)"
```

———-

— VSPACE.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];
```

```
"VectorSpace(a:Field)" [color=lightblue];
"VectorSpace(a:Field)" -> "Module(Field)"

"Module(Field)" [color=seagreen];
"Module(Field)" -> "Module(a:CommutativeRing)"

"Module(a:CommutativeRing)" [color=lightblue];
"Module(a:CommutativeRing)" ->
  "BiModule(a:CommutativeRing,b:CommutativeRing)"

"BiModule(a:CommutativeRing,b:CommutativeRing)" [color=seagreen];
"BiModule(a:CommutativeRing,b:CommutativeRing)" -> "BiModule(a:Ring,b:Ring)"

"BiModule(a:Ring,b:Ring)" [color=lightblue];
"BiModule(a:Ring,b:Ring)" -> "LeftModule(a:Ring)"
"BiModule(a:Ring,b:Ring)" -> "RightModule(a:Ring)"

"RightModule(a:Ring)" [color=seagreen];
"RightModule(a:Ring)" -> "RightModule(a:Rng)"

"RightModule(a:Rng)" [color=lightblue];
"RightModule(a:Rng)" -> "ABELGRP..."

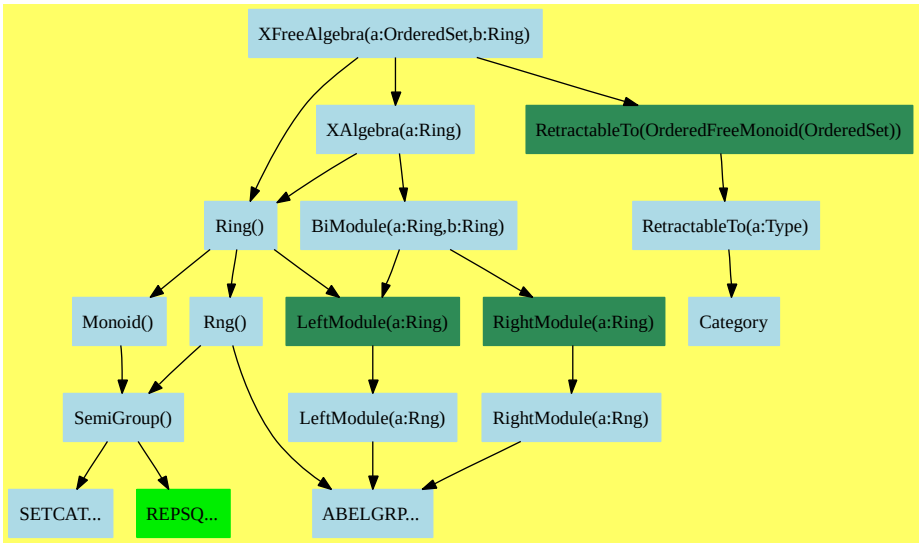
"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "ABELGRP..."

"ABELGRP..." [color=lightblue];
}
```

———-

11.0.159 XFreeAlgebra (XFALG)



— XFreeAlgebra.input —

```
)set break resume
)sys rm -f XFreeAlgebra.output
)spool XFreeAlgebra.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show XFreeAlgebra
--R XFreeAlgebra(vl: OrderedSet,R: Ring)  is a category constructor
--R Abbreviation for XFreeAlgebra is XFALG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for XFALG
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (vl,%) -> %                    ?*? : (%,R) -> %
--R ?*? : (R,%) -> %                     ?*? : (%,%) -> %
--R ?*? : (Integer,%) -> %               ?*? : (PositiveInteger,%) -> %
--R ?**? : (%,PositiveInteger) -> %      ?+? : (%,%) -> %
--R ?-? : (%,%) -> %                     -? : % -> %
--R ?=? : (%,%) -> Boolean               1 : () -> %
--R 0 : () -> %                          ?^? : (%,PositiveInteger) -> %
--R coef : (%,%) -> R                    coerce : vl -> %
--R coerce : R -> %                      coerce : Integer -> %
```

```
--R coerce : % -> OutputForm              constant : % -> R
--R constant? : % -> Boolean              hash : % -> SingleInteger
--R latex : % -> String                   lquo : (%,%) -> %
--R lquo : (%,vl) -> %                    map : ((R -> R),%) -> %
--R mirror : % -> %                       monomial? : % -> Boolean
--R one? : % -> Boolean                   quasiRegular : % -> %
--R quasiRegular? : % -> Boolean          recip : % -> Union(%,"failed")
--R rquo : (%,%) -> %                     rquo : (%,vl) -> %
--R sample : () -> %                      varList : % -> List vl
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R coef : (%,OrderedFreeMonoid vl) -> R
--R coerce : OrderedFreeMonoid vl -> %
--R lquo : (%,OrderedFreeMonoid vl) -> %
--R mindeg : % -> OrderedFreeMonoid vl
--R mindegTerm : % -> Record(k: OrderedFreeMonoid vl,c: R)
--R monom : (OrderedFreeMonoid vl,R) -> %
--R retract : % -> OrderedFreeMonoid vl
--R retractIfCan : % -> Union(OrderedFreeMonoid vl,"failed")
--R rquo : (%,OrderedFreeMonoid vl) -> %
--R sh : (%,NonNegativeInteger) -> % if R has COMRING
--R sh : (%,%) -> % if R has COMRING
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **XFreeAlgebra.help** —

```
=====================================================================
XFreeAlgebra examples
=====================================================================

See Also:
o )show XFreeAlgebra
```

———-

See:

⇒ “XPolynomialsCat” (XPOLYC) 12.0.169 on page 1094

⇐ "RetractableTo" (RETRACT) 2.0.18 on page 55
⇐ "Ring" (RING) 9.0.129 on page 770
⇐ "XAlgebra" (XALG) 10.0.151 on page 940

Exports:

0	1	characteristic	coef	coerce
constant	constant?	hash	latex	lquo
map	mindeg	mindegTerm	mirror	monom
monomial?	one?	quasiRegular	quasiRegular?	recip
retract	retractIfCan	rquo	sample	sh
subtractIfCan	varList	zero?	?*?	?**?
?+?	?-?	-?	?=?	?^?
?~=?				

Attributes Exported:

- if Ring has noZeroDivisors then noZeroDivisors where **noZeroDivisors** is true if $x*y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
coef : (%,OrderedFreeMonoid vl) -> R
coef : (%,%) -> R
coerce : vl -> %
constant : % -> R
constant? : % -> Boolean
lquo : (%,vl) -> %
lquo : (%,%) -> %
lquo : (%,OrderedFreeMonoid vl) -> %
map : ((R -> R),%) -> %
mindeg : % -> OrderedFreeMonoid vl
mindegTerm : % -> Record(k: OrderedFreeMonoid vl,c: R)
mirror : % -> %
monom : (OrderedFreeMonoid vl,R) -> %
monomial? : % -> Boolean
quasiRegular : % -> %
quasiRegular? : % -> Boolean
rquo : (%,OrderedFreeMonoid vl) -> %
rquo : (%,%) -> %
```

```
rquo : (%,vl) -> %
sh : (%,NonNegativeInteger) -> % if R has COMRING
sh : (%,%) -> % if R has COMRING
varList : % -> List vl
?*? : (vl,%) -> %
?*? : (%,R) -> %
```

These exports come from (p770) Ring():

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : % -> OutputForm
coerce : Integer -> %
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (Integer,%) -> %
?*? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
```

These exports come from (p940) XAlgebra(a:Ring):

```
coerce : R -> %
?*? : (R,%) -> %
```

These exports come from (p55) RetractableTo(WORD)
where WORD:OrderedFreeMonoid(OrderedSet))

```
coerce : OrderedFreeMonoid vl -> %
retract : % -> OrderedFreeMonoid vl
retractIfCan : % -> Union(OrderedFreeMonoid vl,"failed")
```

— category XFALG XFreeAlgebra —

```
)abbrev category XFALG XFreeAlgebra
++ Author: Michel Petitot petitot@lifl.fr
++ Date Created: 91
++ Date Last Updated: 7 Juillet 92
++ Fix History: compilation v 2.1 le 13 dec 98
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ This category specifies opeations for  polynomials
++ and formal series with non-commutative variables.

XFreeAlgebra(vl:OrderedSet,R:Ring):Category == Catdef where
   WORD   ==> OrderedFreeMonoid(vl)          -- monoide libre
   NNI    ==> NonNegativeInteger
   I      ==> Integer
   TERM   ==> Record(k: WORD, c: R)

   Catdef == Join(Ring, XAlgebra(R), RetractableTo WORD)
     with
       "*": (vl,%) -> %
         ++ \spad{v * x} returns the product of a variable \spad{x}
         ++ by \spad{x}.
       "*": (%, R) -> %
         ++ \spad{x * r} returns the product of \spad{x} by \spad{r}.
         ++ Usefull if \spad{R} is a non-commutative Ring.
       mindeg: % -> WORD
         ++ \spad{mindeg(x)} returns the little word which appears
         ++ in \spad{x}. Error if \spad{x=0}.
       mindegTerm: % -> TERM
         ++ \spad{mindegTerm(x)} returns the term whose word is
         ++ \spad{mindeg(x)}.
       coef  : (%,WORD) -> R
         ++ \spad{coef(x,w)} returns the coefficient of the word \spad{w}
         ++ in \spad{x}.
       coef  : (%,%) -> R
         ++ \spad{coef(x,y)} returns scalar product of \spad{x} by \spad{y},
         ++ the set of words being regarded as an orthogonal basis.
       lquo  : (%,vl) -> %
         ++ \spad{lquo(x,v)} returns the left simplification of \spad{x}
         ++ by the variable \spad{v}.
       lquo  : (%,WORD) -> %
         ++ \spad{lquo(x,w)} returns the left simplification of \spad{x}
         ++ by the word \spad{w}.
       lquo  : (%,%) -> %
         ++ \spad{lquo(x,y)} returns the left simplification of \spad{x}
         ++ by \spad{y}.
```

```
  rquo  : (%,vl) -> %
    ++ \spad{rquo(x,v)} returns the right simplification of \spad{x}
    ++ by the variable \spad{v}.
  rquo  : (%,WORD) -> %
    ++ \spad{rquo(x,w)} returns the right simplification of \spad{x}
    ++ by \spad{w}.
  rquo  : (%,%) -> %
    ++ \spad{rquo(x,y)} returns the right simplification of \spad{x}
    ++ by \spad{y}.
  monom : (WORD , R) -> %
    ++ \spad{monom(w,r)} returns the product of the word \spad{w}
    ++ by the coefficient \spad{r}.
  monomial? : % -> Boolean
    ++ \spad{monomial?(x)} returns true if \spad{x} is a monomial
  mirror: % -> %
    ++ \spad{mirror(x)} returns \spad{Sum(r_i mirror(w_i))} if
    ++ \spad{x} writes \spad{Sum(r_i w_i)}.
  coerce : vl -> %
    ++ \spad{coerce(v)} returns \spad{v}.
  constant?:% -> Boolean
    ++ \spad{constant?(x)} returns true if \spad{x} is constant.
  constant: % -> R
    ++ \spad{constant(x)} returns the constant term of \spad{x}.
  quasiRegular? : % -> Boolean
    ++ \spad{quasiRegular?(x)} return true if \spad{constant(x)} is zero
  quasiRegular : % -> %
    ++ \spad{quasiRegular(x)} return \spad{x} minus its constant term.
  if R has CommutativeRing then
     sh :(%,%) -> %
        ++ \spad{sh(x,y)} returns the shuffle-product of \spad{x}
        ++ by \spad{y}.
        ++ This multiplication is associative and commutative.
     sh :(%,NNI) -> %
        ++ \spad{sh(x,n)} returns the shuffle power of \spad{x} to
        ++ the \spad{n}.
  map   : (R -> R, %) -> %
    ++ \spad{map(fn,x)} returns \spad{Sum(fn(r_i) w_i)} if \spad{x}
    ++ writes \spad{Sum(r_i w_i)}.
  varList: % -> List vl
    ++ \spad{varList(x)} returns the list of variables which
    ++ appear in \spad{x}.

-- Attributs
  if R has noZeroDivisors then noZeroDivisors
```

———-

— **XFALG.dotabb** —

```
"XFALG"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=XFALG"];
"XFALG" -> "RETRACT"
"XFALG" -> "RING"
"XFALG" -> "XALG"
```

———-

— XFALG.dotfull —

```
"XFreeAlgebra(a:OrderedSet,b:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=XFALG"];
"XFreeAlgebra(a:OrderedSet,b:Ring)" -> "Ring()"
"XFreeAlgebra(a:OrderedSet,b:Ring)" -> "XAlgebra(a:Ring)"
"XFreeAlgebra(a:OrderedSet,b:Ring)" ->
    "RetractableTo(OrderedFreeMonoid(OrderedSet))"
```

———-

— XFALG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"XFreeAlgebra(a:OrderedSet,b:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=XFALG"];
"XFreeAlgebra(a:OrderedSet,b:Ring)" -> "Ring()"
"XFreeAlgebra(a:OrderedSet,b:Ring)" -> "XAlgebra(a:Ring)"
"XFreeAlgebra(a:OrderedSet,b:Ring)" ->
    "RetractableTo(OrderedFreeMonoid(OrderedSet))"

"RetractableTo(OrderedFreeMonoid(OrderedSet))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=RETRACT"];
"RetractableTo(OrderedFreeMonoid(OrderedSet))" -> "RetractableTo(a:Type)"

"XAlgebra(a:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=XALG"];
"XAlgebra(a:Ring)" -> "Ring()"
"XAlgebra(a:Ring)" -> "BiModule(a:Ring,b:Ring)"

"Ring()" [color=lightblue];
"Ring()" -> "Rng()"
"Ring()" -> "Monoid()"
"Ring()" -> "LeftModule(a:Ring)"
```

```
"BiModule(a:Ring,b:Ring)" [color=lightblue];
"BiModule(a:Ring,b:Ring)" -> "LeftModule(a:Ring)"
"BiModule(a:Ring,b:Ring)" -> "RightModule(a:Ring)"

"RightModule(a:Ring)" [color=seagreen];
"RightModule(a:Ring)" -> "RightModule(a:Rng)"

"RightModule(a:Rng)" [color=lightblue];
"RightModule(a:Rng)" -> "ABELGRP..."

"Rng()" [color=lightblue];
"Rng()" -> "ABELGRP..."
"Rng()" -> "SemiGroup()"

"Monoid()" [color=lightblue];
"Monoid()" -> "SemiGroup()"

"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "ABELGRP..."

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SETCAT..."
"SemiGroup()" -> "REPSQ..."

"RetractableTo(a:Type)" [color=lightblue];
"RetractableTo(a:Type)" -> "Category"

"Category" [color=lightblue];

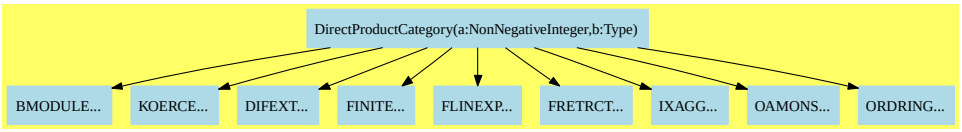
"REPSQ..." [color="#00EE00"];
"SETCAT..." [color=lightblue];
"ABELGRP..." [color=lightblue];
}
```

———————-

Chapter 12

Category Layer 11

12.0.160 DirectProductCategory (DIRPCAT)



— **DirectProductCategory.input** —

```
)set break resume
)sys rm -f DirectProductCategory.output
)spool DirectProductCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show DirectProductCategory
--R DirectProductCategory(dim: NonNegativeInteger,R: Type)  is a category constructor
--R Abbreviation for DirectProductCategory is DIRPCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for DIRPCAT
--R
--R------------------------------ Operations ------------------------------
--R -? : % -> % if R has RING             1 : () -> % if R has MONOID
--R 0 : () -> % if R has CABMON           coerce : % -> Vector R
--R copy : % -> %                         directProduct : Vector R -> %
--R ?.? : (%,Integer) -> R                elt : (%,Integer,R) -> R
--R empty : () -> %                       empty? : % -> Boolean
--R entries : % -> List R                 eq? : (%,%) -> Boolean
```

```
--R index? : (Integer,%) -> Boolean       indices : % -> List Integer
--R map : ((R -> R),%) -> %               qelt : (%,Integer) -> R
--R sample : () -> %
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?*? : (PositiveInteger,%) -> % if R has ABELSG
--R ?*? : (NonNegativeInteger,%) -> % if R has CABMON
--R ?*? : (R,%) -> % if R has RING
--R ?*? : (%,R) -> % if R has RING
--R ?*? : (%,%) -> % if R has MONOID
--R ?*? : (Integer,%) -> % if R has RING
--R ?**? : (%,PositiveInteger) -> % if R has MONOID
--R ?**? : (%,NonNegativeInteger) -> % if R has MONOID
--R ?+? : (%,%) -> % if R has ABELSG
--R ?-? : (%,%) -> % if R has RING
--R ?/? : (%,R) -> % if R has FIELD
--R ?<? : (%,%) -> Boolean if R has ORDRING or R has OAMONS
--R ?<=? : (%,%) -> Boolean if R has ORDRING or R has OAMONS
--R ?=? : (%,%) -> Boolean if R has SETCAT
--R ?>? : (%,%) -> Boolean if R has ORDRING or R has OAMONS
--R ?>=? : (%,%) -> Boolean if R has ORDRING or R has OAMONS
--R D : (%,(R -> R)) -> % if R has RING
--R D : (%,(R -> R),NonNegativeInteger) -> % if R has RING
--R D : (%,List Symbol,List NonNegativeInteger) -> % if and(has(R,PartialDifferentialRing Symbol
--R D : (%,Symbol,NonNegativeInteger) -> % if and(has(R,PartialDifferentialRing Symbol),has(R,Ri
--R D : (%,List Symbol) -> % if and(has(R,PartialDifferentialRing Symbol),has(R,Ring))
--R D : (%,Symbol) -> % if and(has(R,PartialDifferentialRing Symbol),has(R,Ring))
--R D : (%,NonNegativeInteger) -> % if and(has(R,DifferentialRing),has(R,Ring))
--R D : % -> % if and(has(R,DifferentialRing),has(R,Ring))
--R ?^? : (%,PositiveInteger) -> % if R has MONOID
--R ?^? : (%,NonNegativeInteger) -> % if R has MONOID
--R abs : % -> % if R has ORDRING
--R any? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
--R characteristic : () -> NonNegativeInteger if R has RING
--R coerce : R -> % if R has SETCAT
--R coerce : Fraction Integer -> % if and(has(R,RetractableTo Fraction Integer),has(R,SetCategor
--R coerce : Integer -> % if and(has(R,RetractableTo Integer),has(R,SetCategory)) or R has RING
--R coerce : % -> OutputForm if R has SETCAT
--R count : (R,%) -> NonNegativeInteger if R has SETCAT and $ has finiteAggregate
--R count : ((R -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R differentiate : (%,(R -> R)) -> % if R has RING
--R differentiate : (%,(R -> R),NonNegativeInteger) -> % if R has RING
--R differentiate : (%,List Symbol,List NonNegativeInteger) -> % if and(has(R,PartialDifferentia
--R differentiate : (%,Symbol,NonNegativeInteger) -> % if and(has(R,PartialDifferentialRing Symb
--R differentiate : (%,List Symbol) -> % if and(has(R,PartialDifferentialRing Symbol),has(R,Ring
--R differentiate : (%,Symbol) -> % if and(has(R,PartialDifferentialRing Symbol),has(R,Ring))
--R differentiate : (%,NonNegativeInteger) -> % if and(has(R,DifferentialRing),has(R,Ring))
--R differentiate : % -> % if and(has(R,DifferentialRing),has(R,Ring))
--R dimension : () -> CardinalNumber if R has FIELD
--R dot : (%,%) -> R if R has RING
--R entry? : (R,%) -> Boolean if $ has finiteAggregate and R has SETCAT
```

```
--R eval : (%,List R,List R) -> % if R has EVALAB R and R has SETCAT
--R eval : (%,R,R) -> % if R has EVALAB R and R has SETCAT
--R eval : (%,Equation R) -> % if R has EVALAB R and R has SETCAT
--R eval : (%,List Equation R) -> % if R has EVALAB R and R has SETCAT
--R every? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
--R fill! : (%,R) -> % if $ has shallowlyMutable
--R first : % -> R if Integer has ORDSET
--R hash : % -> SingleInteger if R has SETCAT
--R index : PositiveInteger -> % if R has FINITE
--R latex : % -> String if R has SETCAT
--R less? : (%,NonNegativeInteger) -> Boolean
--R lookup : % -> PositiveInteger if R has FINITE
--R map! : ((R -> R),%) -> % if $ has shallowlyMutable
--R max : (%,%) -> % if R has ORDRING or R has OAMONS
--R maxIndex : % -> Integer if Integer has ORDSET
--R member? : (R,%) -> Boolean if R has SETCAT and $ has finiteAggregate
--R members : % -> List R if $ has finiteAggregate
--R min : (%,%) -> % if R has ORDRING or R has OAMONS
--R minIndex : % -> Integer if Integer has ORDSET
--R more? : (%,NonNegativeInteger) -> Boolean
--R negative? : % -> Boolean if R has ORDRING
--R one? : % -> Boolean if R has MONOID
--R parts : % -> List R if $ has finiteAggregate
--R positive? : % -> Boolean if R has ORDRING
--R qsetelt! : (%,Integer,R) -> R if $ has shallowlyMutable
--R random : () -> % if R has FINITE
--R recip : % -> Union(%,"failed") if R has MONOID
--R reducedSystem : Matrix % -> Matrix R if R has RING
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix R,vec: Vector R) if R has RING
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer) if and(has(R,Line
--R reducedSystem : Matrix % -> Matrix Integer if and(has(R,LinearlyExplicitRingOver Integer),has(R,Ring))
--R retract : % -> R if R has SETCAT
--R retract : % -> Fraction Integer if and(has(R,RetractableTo Fraction Integer),has(R,SetCategory))
--R retract : % -> Integer if and(has(R,RetractableTo Integer),has(R,SetCategory))
--R retractIfCan : % -> Union(R,"failed") if R has SETCAT
--R retractIfCan : % -> Union(Fraction Integer,"failed") if and(has(R,RetractableTo Fraction Integer),has(R
--R retractIfCan : % -> Union(Integer,"failed") if and(has(R,RetractableTo Integer),has(R,SetCategory))
--R setelt : (%,Integer,R) -> R if $ has shallowlyMutable
--R sign : % -> Integer if R has ORDRING
--R size : () -> NonNegativeInteger if R has FINITE
--R size? : (%,NonNegativeInteger) -> Boolean
--R subtractIfCan : (%,%) -> Union(%,"failed") if R has CABMON
--R sup : (%,%) -> % if R has OAMONS
--R swap! : (%,Integer,Integer) -> Void if $ has shallowlyMutable
--R unitVector : PositiveInteger -> % if R has RING
--R zero? : % -> Boolean if R has CABMON
--R ?~=? : (%,%) -> Boolean if R has SETCAT
--R
--E 1
```

```
)spool
)lisp (bye)
```

———-

— **DirectProductCategory.help** —

```
====================================================================
DirectProductCategory examples
====================================================================

See Also:
o )show DirectProductCategory
```

———-

See:

⇐ “BiModule” (BMODULE) 9.0.122 on page 726
⇐ “CoercibleTo” (KOERCE) 2.0.6 on page 17
⇐ “DifferentialExtension” (DIFEXT) 11.0.153 on page 953
⇐ “IndexedAggregate” (IXAGG) 5.0.74 on page 318
⇐ “Finite” (FINITE) 4.0.46 on page 168
⇐ “FullyLinearlyExplicitRingOver” (FLINEXP) 11.0.154 on page 960
⇐ “FullyRetractableTo” (FRETRCT) 3.0.27 on page 87
⇐ “OrderedRing” (ORDRING) 10.0.144 on page 879
⇐ “OrderedAbelianMonoidSup” (OAMONS) 9.0.127 on page 759

Exports:

0	1	abs	any?	characteristic
coerce	copy	count	D	differentiate
dimension	directProduct	dot	elt	empty
empty?	entry?	entries	eq?	eval
every?	fill!	first	hash	index
index?	indices	latex	less?	lookup
map	map!	max	maxIndex	member?
members	min	minIndex	more?	negative?
one?	parts	positive?	qelt	qsetelt!
random	recip	reducedSystem	retract	retractIfCan
sample	setelt	sign	size	size?
subtractIfCan	sup	swap!	unitVector	zero?
?~=?	-?	?.?	#?	?*?
?**?	?+?	?-?	?/? ?<?	
?<=?	?=?	?>?	?>=?	?^?

Attributes Exported:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- if #2 has commutativeRing then commutative("*") where **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- if #2 has unitsKnown then unitsKnown where **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- if #2 has Ring then rightUnitary where **leftUnitary** is true if $1 * x = x$ for all x.
- if #2 has Ring then rightUnitary where **rightUnitary** is true if $x * 1 = x$ for all x.
- **nil**

These are directly exported but not implemented:

```
directProduct : Vector R -> %
dot : (%,%) -> R if R has RING
unitVector : PositiveInteger -> % if R has RING
?*? : (R,%) -> % if R has MONOID
?*? : (%,R) -> % if R has MONOID
```

These are implemented by this category:

```
characteristic : () -> NonNegativeInteger if R has RING
coerce : Integer -> %
  if and(has(R,RetractableTo Integer),
         has(R,SetCategory))
  or R has RING
differentiate : (%,(R -> R)) -> % if R has RING
dimension : () -> CardinalNumber if R has FIELD
reducedSystem : Matrix % -> Matrix R if R has RING
reducedSystem :
   (Matrix %,Vector %) ->
      Record(mat: Matrix R,vec: Vector R)
         if R has RING
size : () -> NonNegativeInteger if R has FINITE
?/? : (%,R) -> % if R has FIELD
```

These exports come from (p318) IndexedAggregate(a:SetCategory,R:Type):

```
any? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
coerce : % -> OutputForm if R has SETCAT
copy : % -> %
count : (R,%) -> NonNegativeInteger
  if R has SETCAT
  and $ has finiteAggregate
count : ((R -> Boolean),%) -> NonNegativeInteger
  if $ has finiteAggregate
```

```
elt : (%,Integer,R) -> R
empty : () -> %
empty? : % -> Boolean
entries : % -> List R
entry? : (R,%) -> Boolean
  if $ has finiteAggregate
  and R has SETCAT
eq? : (%,%) -> Boolean
eval : (%,List R,List R) -> %
  if R has EVALAB R
  and R has SETCAT
eval : (%,R,R) -> %
  if R has EVALAB R
  and R has SETCAT
eval : (%,Equation R) -> %
  if R has EVALAB R
  and R has SETCAT
eval : (%,List Equation R) -> %
  if R has EVALAB R
  and R has SETCAT
every? : ((R -> Boolean),%) -> Boolean
  if $ has finiteAggregate
fill! : (%,R) -> % if $ has shallowlyMutable
first : % -> R if Integer has ORDSET
hash : % -> SingleInteger if R has SETCAT
index? : (Integer,%) -> Boolean
indices : % -> List Integer
less? : (%,NonNegativeInteger) -> Boolean
latex : % -> String if R has SETCAT
map : ((R -> R),%) -> %
map! : ((R -> R),%) -> % if $ has shallowlyMutable
maxIndex : % -> Integer if Integer has ORDSET
member? : (R,%) -> Boolean
  if R has SETCAT
  and $ has finiteAggregate
members : % -> List R if $ has finiteAggregate
minIndex : % -> Integer if Integer has ORDSET
more? : (%,NonNegativeInteger) -> Boolean
parts : % -> List R if $ has finiteAggregate
qelt : (%,Integer) -> R
qsetelt! : (%,Integer,R) -> R if $ has shallowlyMutable
sample : () -> %
setelt : (%,Integer,R) -> R if $ has shallowlyMutable
size? : (%,NonNegativeInteger) -> Boolean
swap! : (%,Integer,Integer) -> Void if $ has shallowlyMutable
#? : % -> NonNegativeInteger if $ has finiteAggregate
?.? : (%,Integer) -> R
?=? : (%,%) -> Boolean if R has SETCAT
?~=? : (%,%) -> Boolean if R has SETCAT
```

These exports come from (p17) CoercibleTo(Vector(R:Type)):

```
coerce : % -> Vector R
```

These exports come from (p87) FullyRetractableTo(R:SetCategory):

```
coerce : R -> % if R has SETCAT
coerce : Fraction Integer -> %
  if and(has(R,RetractableTo Fraction Integer),
         has(R,SetCategory))
retract : % -> Integer
  if and(has(R,RetractableTo Integer),
         has(R,SetCategory))
retract : % -> R if R has SETCAT
retract : % -> Fraction Integer
  if and(has(R,RetractableTo Fraction Integer),
         has(R,SetCategory))
retractIfCan : % -> Union(Integer,"failed")
  if and(has(R,RetractableTo Integer),
         has(R,SetCategory))
retractIfCan : % -> Union(R,"failed") if R has SETCAT
retractIfCan : % -> Union(Fraction Integer,"failed")
  if and(has(R,RetractableTo Fraction Integer),
         has(R,SetCategory))
```

These exports come from (p726) BiModule(R:Ring,R:Ring):

```
0 : () -> % if R has CABMON
subtractIfCan : (%,%) -> Union(%,"failed") if R has CABMON
zero? : % -> Boolean if R has CABMON
?+? : (%,%) -> % if R has ABELSG
?*? : (PositiveInteger,%) -> % if R has ABELSG
?*? : (NonNegativeInteger,%) -> % if R has CABMON
?*? : (Integer,%) -> % if R has RING
?-? : (%,%) -> % if R has RING
-? : % -> % if R has RING
```

These exports come from (p953) DifferentialExtension(R:Ring):

```
1 : () -> % if R has RING
D : (%,(R -> R)) -> % if R has RING
D : (%,(R -> R),NonNegativeInteger) -> % if R has RING
D : % -> % if and(has(R,DifferentialRing),has(R,Ring))
D : (%,NonNegativeInteger) -> %
  if and(has(R,DifferentialRing),has(R,Ring))
differentiate : (%,NonNegativeInteger) -> %
  if and(has(R,DifferentialRing),has(R,Ring))
D : (%,List Symbol,List NonNegativeInteger) -> %
  if and(has(R,PartialDifferentialRing Symbol),has(R,Ring))
```

```
D : (%,Symbol,NonNegativeInteger) -> %
  if and(has(R,PartialDifferentialRing Symbol),has(R,Ring))
D : (%,List Symbol) -> %
  if and(has(R,PartialDifferentialRing Symbol),has(R,Ring))
D : (%,Symbol) -> %
  if and(has(R,PartialDifferentialRing Symbol),has(R,Ring))
differentiate : (%,List Symbol,List NonNegativeInteger) -> %
  if and(has(R,PartialDifferentialRing Symbol),has(R,Ring))
differentiate : (%,Symbol,NonNegativeInteger) -> %
  if and(has(R,PartialDifferentialRing Symbol),has(R,Ring))
differentiate : (%,List Symbol) -> %
  if and(has(R,PartialDifferentialRing Symbol),has(R,Ring))
differentiate : % -> %
  if and(has(R,DifferentialRing),has(R,Ring))
differentiate : (%,(R -> R),NonNegativeInteger) -> %
  if R has RING
differentiate : (%,Symbol) -> %
  if and(has(R,PartialDifferentialRing Symbol),has(R,Ring))
one? : % -> Boolean if R has RING
recip : % -> Union(%,"failed") if R has RING
?*? : (%,%) -> % if R has RING
?**? : (%,PositiveInteger) -> % if R has RING
?**? : (%,NonNegativeInteger) -> % if R has RING
?^? : (%,PositiveInteger) -> % if R has RING
?^? : (%,NonNegativeInteger) -> % if R has RING
```

These exports come from (p960) FullyLinearlyExplicitRingOver(R:Ring):

```
reducedSystem :
  (Matrix %,Vector %) ->
    Record(mat: Matrix Integer,vec: Vector Integer)
      if and(has(R,LinearlyExplicitRingOver Integer),has(R,Ring))
reducedSystem : Matrix % -> Matrix Integer
  if and(has(R,LinearlyExplicitRingOver Integer),has(R,Ring))
```

These exports come from (p168) Finite():

```
index : PositiveInteger -> % if R has FINITE
lookup : % -> PositiveInteger if R has FINITE
random : () -> % if R has FINITE
```

These exports come from (p879) OrderedRing():

```
abs : % -> % if R has ORDRING
max : (%,%) -> % if R has ORDRING or R has OAMONS
min : (%,%) -> % if R has ORDRING or R has OAMONS
negative? : % -> Boolean if R has ORDRING
positive? : % -> Boolean if R has ORDRING
sign : % -> Integer if R has ORDRING
```

```
?<? : (%,%) -> Boolean if R has ORDRING or R has OAMONS
?<=? : (%,%) -> Boolean if R has ORDRING or R has OAMONS
?>? : (%,%) -> Boolean if R has ORDRING or R has OAMONS
?>=? : (%,%) -> Boolean if R has ORDRING or R has OAMONS
```

These exports come from (p759) OrderedAbelianMonoidSup():

```
sup : (%,%) -> % if R has OAMONS
```

— category DIRPCAT DirectProductCategory —

```
)abbrev category DIRPCAT DirectProductCategory
-- all direct product category domains must be compiled
-- without subsumption, set SourceLevelSubset to EQUAL
--)bo $noSubsumption := true

--% DirectProductCategory

++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors: DirectProduct
++ Also See: VectorCategory
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ This category represents a finite cartesian product of a given type.
++ Many categorical properties are preserved under this construction.

DirectProductCategory(dim:NonNegativeInteger, R:Type): Category ==
  Join(IndexedAggregate(Integer, R), CoercibleTo Vector R) with
         finiteAggregate
           ++ attribute to indicate an aggregate of finite size
         directProduct: Vector R -> %
           ++ directProduct(v) converts the vector v to become
           ++ a direct product. Error: if the length of v is
           ++ different from dim.
         if R has SetCategory then FullyRetractableTo R
         if R has Ring then
           BiModule(R, R)
           DifferentialExtension R
           FullyLinearlyExplicitRingOver R
           unitVector: PositiveInteger -> %
             ++ unitVector(n) produces a vector with 1 in position n and
             ++ zero elsewhere.
           dot: (%, %) -> R
             ++ dot(x,y) computes the inner product of the vectors x and y.
```

```
        if R has AbelianSemiGroup then AbelianSemiGroup
        if R has CancellationAbelianMonoid then CancellationAbelianMonoid
        if R has Monoid then
           Monoid
          _* : (R, %) -> %
            ++ r * y multiplies the element r times each component of the
            ++ vector y.
          _* : (%, R) -> %
            ++ y*r multiplies each component of the vector y by the element r.
        if R has Finite then Finite
        if R has CommutativeRing then
          Algebra R
          CommutativeRing
        if R has unitsKnown then unitsKnown
        if R has OrderedRing then OrderedRing
        if R has OrderedAbelianMonoidSup then OrderedAbelianMonoidSup
        if R has Field then VectorSpace R
add
      if R has Ring then
        equation2R: Vector % -> Matrix R

        coerce(n:Integer):%          == n::R::%

        characteristic()             == characteristic()$R

        differentiate(z:%, d:R -> R) == map(d, z)

        equation2R v ==
          ans:Matrix(R) := new(dim, #v, 0)
          for i in minRowIndex ans .. maxRowIndex ans repeat
            for j in minColIndex ans .. maxColIndex ans repeat
              qsetelt_!(ans, i, j, qelt(qelt(v, j), i))
          ans

        reducedSystem(m:Matrix %):Matrix(R) ==
          empty? m => new(0, 0, 0)
          reduce(vertConcat, [equation2R row(m, i)
                 for i in minRowIndex m .. maxRowIndex m])$List(Matrix R)

        reducedSystem(m:Matrix %, v:Vector %):
          Record(mat:Matrix R, vec:Vector R) ==
            vh:Vector(R) :=
              empty? v => empty()
              rh := reducedSystem(v::Matrix %)@Matrix(R)
              column(rh, minColIndex rh)
            [reducedSystem(m)@Matrix(R), vh]

      if R has Finite then size == size$R ** dim

      if R has Field then
```

```
        x / b       == x * inv b

        dimension() == dim::CardinalNumber
```

———-

— **DIRPCAT.dotabb** —

```
"DIRPCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=DIRPCAT"];
"DIRPCAT" -> "IXAGG"
"DIRPCAT" -> "KOERCE"
"DIRPCAT" -> "FRETRCT"
"DIRPCAT" -> "BMODULE"
"DIRPCAT" -> "DIFEXT"
"DIRPCAT" -> "FLINEXP"
"DIRPCAT" -> "FINITE"
"DIRPCAT" -> "ORDRING"
"DIRPCAT" -> "OAMONS"
```

———-

— **DIRPCAT.dotfull** —

```
"DirectProductCategory(a:NonNegativeInteger,b:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=DIRPCAT"];
"DirectProductCategory(a:NonNegativeInteger,b:Type)"
  -> "IndexedAggregate(a:SetCategory,b:Type)"
"DirectProductCategory(a:NonNegativeInteger,b:Type)"
  -> "CoercibleTo(a:Type)"
"DirectProductCategory(a:NonNegativeInteger,b:Type)"
  -> "FullyRetractableTo(a:Type)"
"DirectProductCategory(a:NonNegativeInteger,b:Type)"
  -> "BiModule(a:Ring,b:Ring)"
"DirectProductCategory(a:NonNegativeInteger,b:Type)"
  -> "DifferentialExtension(a:Ring)"
"DirectProductCategory(a:NonNegativeInteger,b:Type)"
  -> "FullyLinearlyExplicitRingOver(a:Ring)"
"DirectProductCategory(a:NonNegativeInteger,b:Type)"
  -> "Finite()"
"DirectProductCategory(a:NonNegativeInteger,b:Type)"
  -> "OrderedRing()"
"DirectProductCategory(a:NonNegativeInteger,b:Type)"
  -> "OrderedAbelianMonoidSup()"
```

———-

— DIRPCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

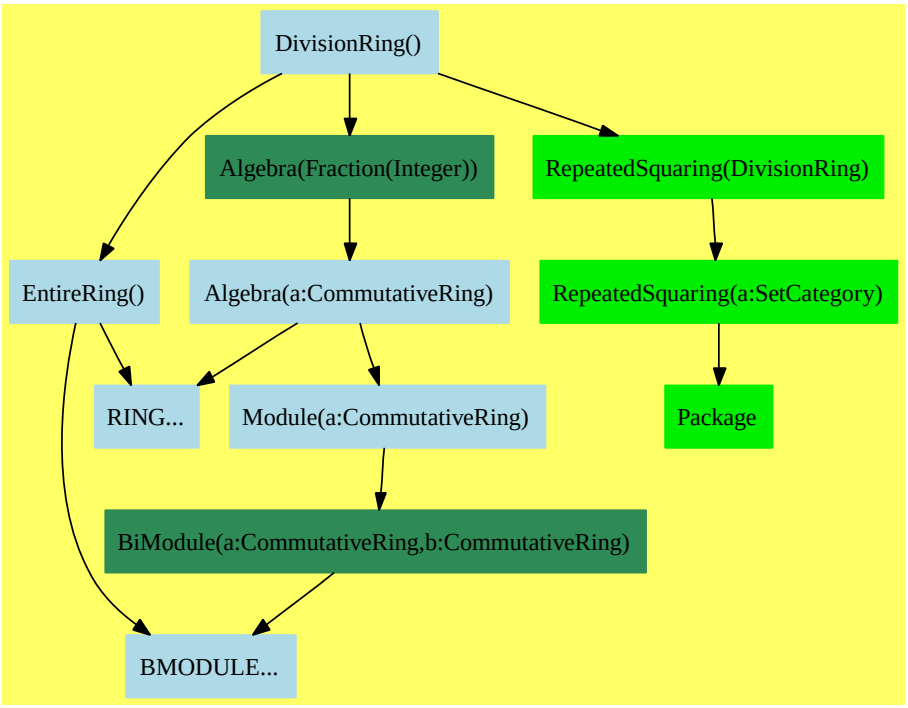
"DirectProductCategory(a:NonNegativeInteger,b:Type)" [color=lightblue];
"DirectProductCategory(a:NonNegativeInteger,b:Type)" -> "BMODULE..."
"DirectProductCategory(a:NonNegativeInteger,b:Type)" -> "KOERCE..."
"DirectProductCategory(a:NonNegativeInteger,b:Type)" -> "DIFEXT..."
"DirectProductCategory(a:NonNegativeInteger,b:Type)" -> "FINITE..."
"DirectProductCategory(a:NonNegativeInteger,b:Type)" -> "FLINEXP..."
"DirectProductCategory(a:NonNegativeInteger,b:Type)" -> "FRETRCT..."
"DirectProductCategory(a:NonNegativeInteger,b:Type)" -> "IXAGG..."
"DirectProductCategory(a:NonNegativeInteger,b:Type)" -> "OAMONS..."
"DirectProductCategory(a:NonNegativeInteger,b:Type)" -> "ORDRING..."

"BMODULE..." [color=lightblue];
"KOERCE..." [color=lightblue];
"DIFEXT..." [color=lightblue];
"FINITE..." [color=lightblue];
"FLINEXP..." [color=lightblue];
"FRETRCT..." [color=lightblue];
"IXAGG..." [color=lightblue];
"OAMONS..." [color=lightblue];
"ORDRING..." [color=lightblue];

}
```

———-

12.0.161 DivisionRing (DIVRING)



— DivisionRing.input —

```
)set break resume
)sys rm -f DivisionRing.output
)spool DivisionRing.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show DivisionRing
--R DivisionRing  is a category constructor
--R Abbreviation for DivisionRing is DIVRING
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for DIVRING
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,Fraction Integer) -> %       ?*? : (Fraction Integer,%) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,Integer) -> %
--R ?**? : (%,PositiveInteger) -> %       ?+? : (%,%) -> %
```

```
--R ?-? : (%,%) -> %                        -? : % -> %
--R ?=? : (%,%) -> Boolean                  1 : () -> %
--R 0 : () -> %                             ?^? : (%,Integer) -> %
--R ?^? : (%,PositiveInteger) -> %          coerce : Fraction Integer -> %
--R coerce : Integer -> %                   coerce : % -> OutputForm
--R hash : % -> SingleInteger               inv : % -> %
--R latex : % -> String                     one? : % -> Boolean
--R recip : % -> Union(%,"failed")          sample : () -> %
--R zero? : % -> Boolean                    ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— **DivisionRing.help** —

```
====================================================================
DivisionRing examples
====================================================================

See Also:
o )show DivisionRing
```

———-

See:

⇒ “Field” (FIELD) 16.0.185 on page 1221
⇐ “Algebra” (ALGEBRA) 11.0.152 on page 947
⇐ “EntireRing” (ENTIRER) 10.0.139 on page 853

Exports:

1	0	characteristic	coerce	hash
inv	latex	one?	recip	sample
subtractIfCan	zero?	?^?	?~=?	?**?
?*?	?+?	?-?	-?	?=?

Attributes exported:

- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.
- **nil**

These are directly exported but not implemented:

```
inv : % -> %
```

These are implemented by this category:

```
?^? : (%,Integer) -> %
?**? : (%,Integer) -> %
```

These exports come from (p853) EntireRing():

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : Integer -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
```

These exports come from (p947) Algebra(Fraction(Integer)):

```
coerce : Fraction Integer -> %
?*? : (%,Fraction Integer) -> %
?*? : (Fraction Integer,%) -> %
```

— category DIVRING DivisionRing —

```
)abbrev category DIVRING DivisionRing
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A division ring (sometimes called a skew field),
++ i.e. a not necessarily commutative ring where
++ all non-zero elements have multiplicative inverses.

DivisionRing(): Category ==
 Join(EntireRing, Algebra Fraction Integer) with
      "**": (%,Integer) -> %
          ++ x**n returns x raised to the integer power n.
      "^" : (%,Integer) -> %
          ++ x^n returns x raised to the integer power n.
      inv : % -> %
          ++ inv x returns the multiplicative inverse of x.
          ++ Error: if x is 0.
-- Q-algebra is a lie, should be conditional on characteristic 0,
-- but knownInfo cannot handle the following commented
--    if % has CharacteristicZero then Algebra Fraction Integer
    add
      n: Integer
      x: %
      _^(x:%, n:Integer):% == x ** n
      import RepeatedSquaring(%)
      x ** n: Integer ==
         zero? n => 1
         zero? x =>
            n<0 => error "division by zero"
            x
         n<0 =>
            expt(inv x,(-n) pretend PositiveInteger)
         expt(x,n pretend PositiveInteger)
--    if % has CharacteristicZero() then
      q:Fraction(Integer) * x:% == numer(q) * inv(denom(q)::%) * x
```

———-

— **DIVRING.dotabb** —

```
"DIVRING"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=DIVRING"];
"DIVRING" -> "ENTIRER"
"DIVRING" -> "ALGEBRA"
```

———-

— **DIVRING.dotfull** —

```
"DivisionRing()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=DIVRING"];
"DivisionRing()" -> "EntireRing()"
"DivisionRing()" -> "Algebra(Fraction(Integer))"
"DivisionRing()" -> "RepeatedSquaring(DivisionRing)"
```

———-

— **DIVRING.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"DivisionRing()" [color=lightblue];
"DivisionRing()" -> "EntireRing()"
"DivisionRing()" -> "Algebra(Fraction(Integer))"
"DivisionRing()" -> "RepeatedSquaring(DivisionRing)"

"RepeatedSquaring(DivisionRing)" [color="#00EE00"];
"RepeatedSquaring(DivisionRing)" -> "RepeatedSquaring(a:SetCategory)"

"RepeatedSquaring(a:SetCategory)" [color="#00EE00"];
"RepeatedSquaring(a:SetCategory)" -> "Package"

"Package" [color="#00EE00"];

"EntireRing()" [color=lightblue];
"EntireRing()" -> "RING..."
"EntireRing()" -> "BMODULE..."

"Algebra(Fraction(Integer))" [color=seagreen];
"Algebra(Fraction(Integer))" -> "Algebra(a:CommutativeRing)"
```

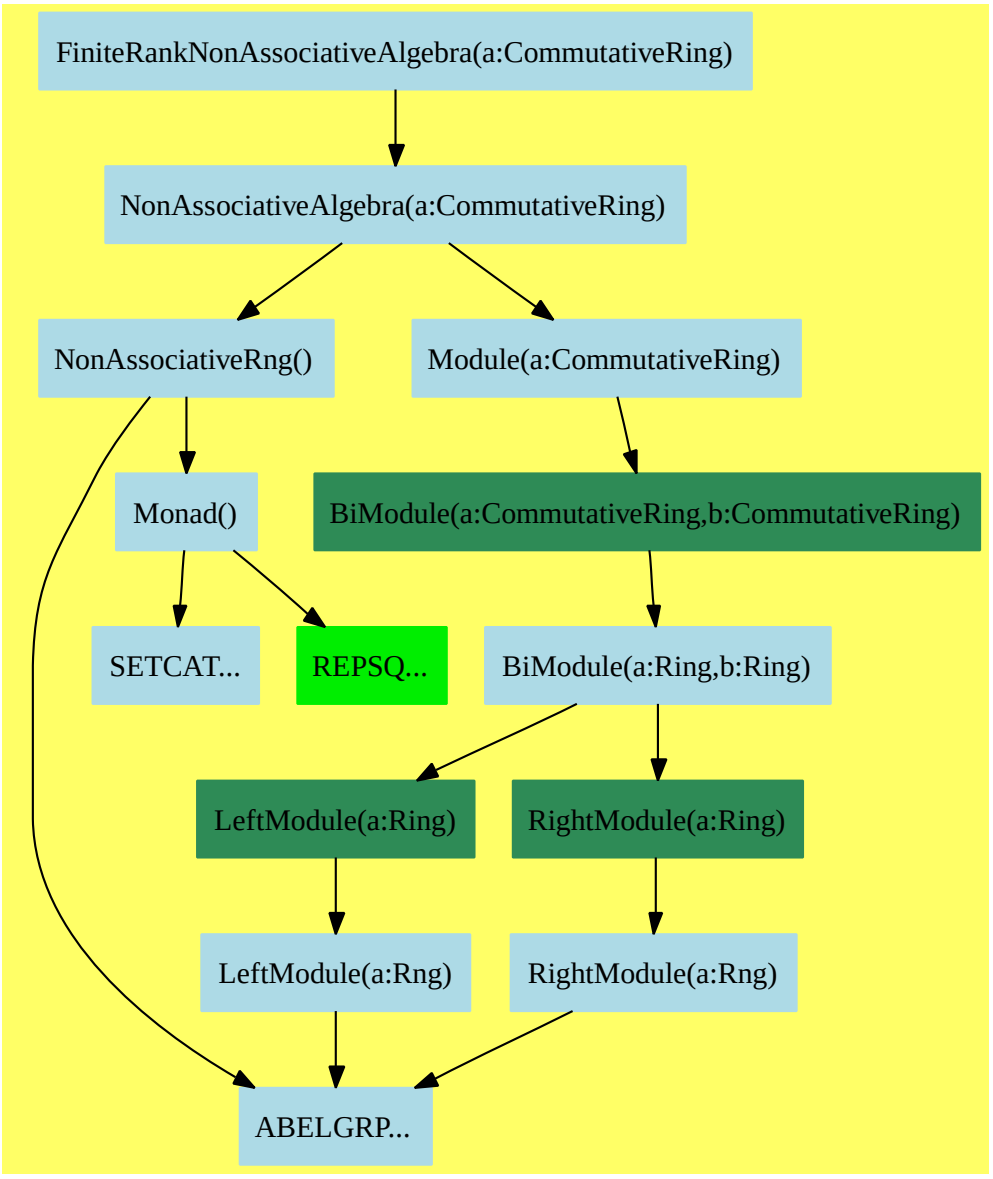
```
"Algebra(a:CommutativeRing)" [color=lightblue];
"Algebra(a:CommutativeRing)" -> "RING..."
"Algebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"

"Module(a:CommutativeRing)" [color=lightblue];
"Module(a:CommutativeRing)" ->
  "BiModule(a:CommutativeRing,b:CommutativeRing)"

"BiModule(a:CommutativeRing,b:CommutativeRing)" [color=seagreen];
"BiModule(a:CommutativeRing,b:CommutativeRing)" -> "BMODULE..."

"RING..." [color=lightblue];
"BMODULE..." [color=lightblue];
}
```

12.0.162 FiniteRankNonAssociativeAlgebra (FINAALG)



— FiniteRankNonAssociativeAlgebra.input —

```
)set break resume
)sys rm -f FiniteRankNonAssociativeAlgebra.output
)spool FiniteRankNonAssociativeAlgebra.output
```

```
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FiniteRankNonAssociativeAlgebra
--R FiniteRankNonAssociativeAlgebra R: CommutativeRing  is a category constructor
--R Abbreviation for FiniteRankNonAssociativeAlgebra is FINAALG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FINAALG
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (R,%) -> %                      ?*? : (%,R) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 0 : () -> %                           alternative? : () -> Boolean
--R antiAssociative? : () -> Boolean      antiCommutative? : () -> Boolean
--R antiCommutator : (%,%) -> %           associative? : () -> Boolean
--R associator : (%,%,%) -> %             coerce : % -> OutputForm
--R commutative? : () -> Boolean          commutator : (%,%) -> %
--R flexible? : () -> Boolean             hash : % -> SingleInteger
--R jacobiIdentity? : () -> Boolean       jordanAdmissible? : () -> Boolean
--R jordanAlgebra? : () -> Boolean        latex : % -> String
--R leftAlternative? : () -> Boolean      leftDiscriminant : Vector % -> R
--R leftNorm : % -> R                     leftTrace : % -> R
--R lieAdmissible? : () -> Boolean        lieAlgebra? : () -> Boolean
--R powerAssociative? : () -> Boolean     rank : () -> PositiveInteger
--R rightAlternative? : () -> Boolean     rightDiscriminant : Vector % -> R
--R rightNorm : % -> R                    rightTrace : % -> R
--R sample : () -> %                      someBasis : () -> Vector %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R associatorDependence : () -> List Vector R if R has INTDOM
--R conditionsForIdempotents : Vector % -> List Polynomial R
--R coordinates : (Vector %,Vector %) -> Matrix R
--R coordinates : (%,Vector %) -> Vector R
--R leftCharacteristicPolynomial : % -> SparseUnivariatePolynomial R
--R leftMinimalPolynomial : % -> SparseUnivariatePolynomial R if R has INTDOM
--R leftPower : (%,PositiveInteger) -> %
--R leftRecip : % -> Union(%,"failed") if R has INTDOM
--R leftRegularRepresentation : (%,Vector %) -> Matrix R
--R leftTraceMatrix : Vector % -> Matrix R
--R leftUnit : () -> Union(%,"failed") if R has INTDOM
--R leftUnits : () -> Union(Record(particular: %,basis: List %),"failed") if R has INTDOM
--R noncommutativeJordanAlgebra? : () -> Boolean
--R plenaryPower : (%,PositiveInteger) -> %
--R recip : % -> Union(%,"failed") if R has INTDOM
--R represents : (Vector R,Vector %) -> %
```

```
--R rightCharacteristicPolynomial : % -> SparseUnivariatePolynomial R
--R rightMinimalPolynomial : % -> SparseUnivariatePolynomial R if R has INTDOM
--R rightPower : (%,PositiveInteger) -> %
--R rightRecip : % -> Union(%,"failed") if R has INTDOM
--R rightRegularRepresentation : (%,Vector %) -> Matrix R
--R rightTraceMatrix : Vector % -> Matrix R
--R rightUnit : () -> Union(%,"failed") if R has INTDOM
--R rightUnits : () -> Union(Record(particular: %,basis: List %),"failed") if R has INTDOM
--R structuralConstants : Vector % -> Vector Matrix R
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unit : () -> Union(%,"failed") if R has INTDOM
--R
--E 1

)spool
)lisp (bye)
```

———-

— **FiniteRankNonAssociativeAlgebra.help** —

```
==========================================================================
FiniteRankNonAssociativeAlgebra examples
==========================================================================

See Also:
o )show FiniteRankNonAssociativeAlgebra
```

———-

See:

⇒ “FramedNonAssociativeAlgebra” (FRNAALG) 13.0.172 on page 1115
⇐ “NonAssociativeAlgebra” (NAALG) 11.0.157 on page 980

Exports:

0	alternative?
antiAssociative?	antiCommutative?
antiCommutator	associative?
associator	associatorDependence
coerce	commutative?
commutator	conditionsForIdempotents
coordinates	flexible?
hash	jacobiIdentity?
jordanAdmissible?	jordanAlgebra?
latex	leftAlternative?
leftCharacteristicPolynomial	leftDiscriminant
leftMinimalPolynomial	leftNorm
leftPower	leftRecip
leftRegularRepresentation	leftTrace
leftTraceMatrix	leftUnit
leftUnits	lieAdmissible?
lieAlgebra?	noncommutativeJordanAlgebra?
plenaryPower	powerAssociative?
rank	recip
represents	rightAlternative?
rightCharacteristicPolynomial	rightDiscriminant
rightMinimalPolynomial	rightNorm
rightPower	rightRecip
rightRegularRepresentation	rightTrace
rightTraceMatrix	rightUnit
rightUnits	sample
someBasis	structuralConstants
subtractIfCan	unit
zero?	?*?
?**?	?+?
?-?	-?
?=?	?~=?

Attributes Exported:

- if R has IntegralDomain then unitsKnown where **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
conditionsForIdempotents : Vector % -> List Polynomial R
coordinates : (%,Vector %) -> Vector R
leftUnit : () -> Union(%,"failed") if R has INTDOM
```

```
leftUnits : () -> Union(Record(particular: %,basis: List %),"failed")
    if R has INTDOM
powerAssociative? : () -> Boolean
rank : () -> PositiveInteger
rightUnit : () -> Union(%,"failed") if R has INTDOM
rightUnits : () -> Union(Record(particular: %,basis: List %),"failed")
    if R has INTDOM
someBasis : () -> Vector %
unit : () -> Union(%,"failed") if R has INTDOM
```

These are implemented by this category:

```
alternative? : () -> Boolean
antiAssociative? : () -> Boolean
antiCommutative? : () -> Boolean
associative? : () -> Boolean
associatorDependence : () -> List Vector R if R has INTDOM
commutative? : () -> Boolean
coordinates : (Vector %,Vector %) -> Matrix R
flexible? : () -> Boolean
jacobiIdentity? : () -> Boolean
jordanAdmissible? : () -> Boolean
jordanAlgebra? : () -> Boolean
leftAlternative? : () -> Boolean
leftCharacteristicPolynomial : % -> SparseUnivariatePolynomial R
leftDiscriminant : Vector % -> R
leftMinimalPolynomial : % -> SparseUnivariatePolynomial R if R has INTDOM
leftNorm : % -> R
leftRecip : % -> Union(%,"failed") if R has INTDOM
leftRegularRepresentation : (%,Vector %) -> Matrix R
leftTrace : % -> R
leftTraceMatrix : Vector % -> Matrix R
lieAdmissible? : () -> Boolean
lieAlgebra? : () -> Boolean
noncommutativeJordanAlgebra? : () -> Boolean
recip : % -> Union(%,"failed") if R has INTDOM
represents : (Vector R,Vector %) -> %
rightAlternative? : () -> Boolean
rightCharacteristicPolynomial : % -> SparseUnivariatePolynomial R
rightDiscriminant : Vector % -> R
rightMinimalPolynomial : % -> SparseUnivariatePolynomial R if R has INTDOM
rightNorm : % -> R
rightRecip : % -> Union(%,"failed") if R has INTDOM
rightRegularRepresentation : (%,Vector %) -> Matrix R
rightTrace : % -> R
rightTraceMatrix : Vector % -> Matrix R
structuralConstants : Vector % -> Vector Matrix R
```

These exports come from (p980) NonAssociativeAlgebra(R:CommutativeRing):

```
0 : () -> %
antiCommutator : (%,%) -> %
associator : (%,%,%) -> %
coerce : % -> OutputForm
commutator : (%,%) -> %
hash : % -> SingleInteger
latex : % -> String
leftPower : (%,PositiveInteger) -> %
plenaryPower : (%,PositiveInteger) -> %
rightPower : (%,PositiveInteger) -> %
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?~=? : (%,%) -> Boolean
?*? : (PositiveInteger,%) -> %
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (NonNegativeInteger,%) -> %
?*? : (R,%) -> %
?*? : (%,R) -> %
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
```

— category FINAALG FiniteRankNonAssociativeAlgebra —

```
)abbrev category FINAALG FiniteRankNonAssociativeAlgebra
++ Author: J. Grabmeier, R. Wisbauer
++ Date Created: 01 March 1991
++ Date Last Updated: 12 June 1991
++ Basic Operations: +,-,*,**, someBasis
++ Related Constructors: FramedNonAssociativeAlgebra, FramedAlgebra,
++   FiniteRankAssociativeAlgebra
++ Also See:
++ AMS Classifications:
++ Keywords: nonassociative algebra, basis
++ References:
++   R.D. Schafer: An Introduction to Nonassociative Algebras
++   Academic Press, New York, 1966
++
++   R. Wisbauer: Bimodule Structure of Algebra
++   Lecture Notes Univ. Duesseldorf 1991
++ Description:
++ A FiniteRankNonAssociativeAlgebra is a non associative algebra over
++ a commutative ring R which is a free \spad{R}-module of finite rank.

FiniteRankNonAssociativeAlgebra(R:CommutativeRing):
```

```
Category == NonAssociativeAlgebra R with
   someBasis : () -> Vector %
     ++ someBasis() returns some \spad{R}-module basis.
   rank : () -> PositiveInteger
     ++ rank() returns the rank of the algebra as \spad{R}-module.
   conditionsForIdempotents: Vector % -> List Polynomial R
     ++ conditionsForIdempotents([v1,...,vn]) determines a complete list
     ++ of polynomial equations for the coefficients of idempotents
     ++ with respect to the \spad{R}-module basis \spad{v1},...,\spad{vn}.
   structuralConstants: Vector % -> Vector Matrix R
     ++ structuralConstants([v1,v2,...,vm]) calculates the structural
     ++ constants \spad{[(gammaijk) for k in 1..m]} defined by
     ++ \spad{vi * vj = gammaij1 * v1 + ... + gammaijm * vm},
     ++ where \spad{[v1,...,vm]} is an \spad{R}-module basis
     ++ of a subalgebra.
   leftRegularRepresentation: (% , Vector %) -> Matrix R
     ++ leftRegularRepresentation(a,[v1,...,vn]) returns the matrix of
     ++ the linear map defined by left multiplication by \spad{a}
     ++ with respect to the \spad{R}-module basis \spad{[v1,...,vn]}.
   rightRegularRepresentation: (% , Vector %) -> Matrix R
     ++ rightRegularRepresentation(a,[v1,...,vn]) returns the matrix of
     ++ the linear map defined by right multiplication by \spad{a}
     ++ with respect to the \spad{R}-module basis \spad{[v1,...,vn]}.
   leftTrace: %  -> R
     ++ leftTrace(a) returns the trace of the left regular representation
     ++ of \spad{a}.
   rightTrace: %  -> R
     ++ rightTrace(a) returns the trace of the right regular representation
     ++ of \spad{a}.
   leftNorm: %  -> R
     ++ leftNorm(a) returns the determinant of the left regular
     ++ representation of \spad{a}.
   rightNorm: %  -> R
     ++ rightNorm(a) returns the determinant of the right regular
     ++ representation of \spad{a}.
   coordinates: (%, Vector %) -> Vector R
     ++ coordinates(a,[v1,...,vn]) returns the coordinates of \spad{a}
     ++ with respect to the \spad{R}-module basis \spad{v1},...,\spad{vn}.
   coordinates: (Vector %, Vector %) -> Matrix R
     ++ coordinates([a1,...,am],[v1,...,vn]) returns a matrix whose
     ++ i-th row is formed by the coordinates of \spad{ai}
     ++ with respect to the \spad{R}-module basis \spad{v1},...,\spad{vn}.
   represents: (Vector R, Vector %) -> %
     ++ represents([a1,...,am],[v1,...,vm]) returns the linear
     ++ combination \spad{a1*vm + ... + an*vm}.
   leftDiscriminant: Vector % -> R
     ++ leftDiscriminant([v1,...,vn]) returns  the determinant of the
     ++ \spad{n}-by-\spad{n} matrix whose element at the \spad{i}-th row
     ++ and \spad{j}-th column is given by the left trace of the product
     ++ \spad{vi*vj}. Note that this is the same as
```

```
    ++ \spad{determinant(leftTraceMatrix([v1,...,vn]))}.
  rightDiscriminant: Vector % -> R
    ++ rightDiscriminant([v1,...,vn]) returns  the determinant of the
    ++ \spad{n}-by-\spad{n} matrix whose element at the \spad{i}-th row
    ++ and \spad{j}-th column is given by the right trace of the product
    ++ \spad{vi*vj}. Note that this is the same as
    ++ \spad{determinant(rightTraceMatrix([v1,...,vn]))}.
  leftTraceMatrix: Vector % -> Matrix R
    ++ leftTraceMatrix([v1,...,vn]) is the \spad{n}-by-\spad{n} matrix
    ++ whose element at the \spad{i}-th row and \spad{j}-th column is given
    ++ by the left trace of the product \spad{vi*vj}.
  rightTraceMatrix: Vector % -> Matrix R
    ++ rightTraceMatrix([v1,...,vn]) is the \spad{n}-by-\spad{n} matrix
    ++ whose element at the \spad{i}-th row and \spad{j}-th column is given
    ++ by the right trace of the product \spad{vi*vj}.
  leftCharacteristicPolynomial: % -> SparseUnivariatePolynomial R
    ++ leftCharacteristicPolynomial(a) returns the characteristic
    ++ polynomial of the left regular representation of \spad{a}
    ++ with respect to any basis.
  rightCharacteristicPolynomial: % -> SparseUnivariatePolynomial R
    ++ rightCharacteristicPolynomial(a) returns the characteristic
    ++ polynomial of the right regular representation of \spad{a}
    ++ with respect to any basis.

  --we not necessarily have a unit
  --if R has CharacteristicZero then CharacteristicZero
  --if R has CharacteristicNonZero then CharacteristicNonZero

  commutative?:()-> Boolean
    ++ commutative?() tests if multiplication in the algebra
    ++ is commutative.
  antiCommutative?:()-> Boolean
    ++ antiCommutative?() tests if \spad{a*a = 0}
    ++ for all \spad{a} in the algebra.
    ++ Note that this implies \spad{a*b + b*a = 0} for all
    ++ \spad{a} and \spad{b}.
  associative?:()-> Boolean
    ++ associative?() tests if multiplication in algebra
    ++ is associative.
  antiAssociative?:()-> Boolean
    ++ antiAssociative?() tests if multiplication in algebra
    ++ is anti-associative, i.e. \spad{(a*b)*c + a*(b*c) = 0}
    ++ for all \spad{a},b,c in the algebra.
  leftAlternative?: ()-> Boolean
    ++ leftAlternative?() tests if \spad{2*associator(a,a,b) = 0}
    ++ for all \spad{a}, b in the algebra.
    ++ Note that we only can test this; in general we don't know
    ++ whether \spad{2*a=0} implies \spad{a=0}.
  rightAlternative?: ()-> Boolean
    ++ rightAlternative?() tests if \spad{2*associator(a,b,b) = 0}
```

```
  ++ for all \spad{a}, b in the algebra.
  ++ Note that we only can test this; in general we don't know
  ++ whether \spad{2*a=0} implies \spad{a=0}.
flexible?: ()->  Boolean
  ++ flexible?() tests if \spad{2*associator(a,b,a) = 0}
  ++ for all \spad{a}, b in the algebra.
  ++ Note that we only can test this; in general we don't know
  ++ whether \spad{2*a=0} implies \spad{a=0}.
alternative?: ()-> Boolean
  ++ alternative?() tests if
  ++ \spad{2*associator(a,a,b) = 0 = 2*associator(a,b,b)}
  ++ for all \spad{a}, b in the algebra.
  ++ Note that we only can test this; in general we don't know
  ++ whether \spad{2*a=0} implies \spad{a=0}.
powerAssociative?:()-> Boolean
  ++ powerAssociative?() tests if all subalgebras
  ++ generated by a single element are associative.
jacobiIdentity?:() -> Boolean
  ++ jacobiIdentity?() tests if \spad{(a*b)*c + (b*c)*a + (c*a)*b = 0}
  ++ for all \spad{a},b,c in the algebra. For example, this holds
  ++ for crossed products of 3-dimensional vectors.
lieAdmissible?: () -> Boolean
  ++ lieAdmissible?() tests if the algebra defined by the commutators
  ++ is a Lie algebra, i.e. satisfies the Jacobi identity.
  ++ The property of anticommutativity follows from definition.
jordanAdmissible?: () -> Boolean
  ++ jordanAdmissible?() tests if 2 is invertible in the
  ++ coefficient domain and the multiplication defined by
  ++ \spad{(1/2)(a*b+b*a)} determines a
  ++ Jordan algebra, i.e. satisfies the Jordan identity.
  ++ The property of \spadatt{commutative("*")}
  ++ follows from by definition.
noncommutativeJordanAlgebra?: () -> Boolean
  ++ noncommutativeJordanAlgebra?() tests if the algebra
  ++ is flexible and Jordan admissible.
jordanAlgebra?:() -> Boolean
  ++ jordanAlgebra?() tests if the algebra is commutative,
  ++ characteristic is not 2, and \spad{(a*b)*a**2 - a*(b*a**2) = 0}
  ++ for all \spad{a},b,c in the algebra (Jordan identity).
  ++ Example:
  ++ for every associative algebra \spad{(A,+,@)} we can construct a
  ++ Jordan algebra \spad{(A,+,*)}, where \spad{a*b := (a@b+b@a)/2}.
lieAlgebra?:() -> Boolean
  ++ lieAlgebra?() tests if the algebra is anticommutative
  ++ and \spad{(a*b)*c + (b*c)*a + (c*a)*b = 0}
  ++ for all \spad{a},b,c in the algebra (Jacobi identity).
  ++ Example:
  ++ for every associative algebra \spad{(A,+,@)} we can construct a
  ++ Lie algebra \spad{(A,+,*)}, where \spad{a*b := a@b-b@a}.
```

```
if R has IntegralDomain then
  -- we not neccessarily have a unit, hence we don't inherit
  -- the next 3 functions anc hence copy them from MonadWithUnit:
  recip: % -> Union(%,"failed")
    ++ recip(a) returns an element, which is both a left and a right
    ++ inverse of \spad{a},
    ++ or \spad{"failed"} if there is no unit element, if such an
    ++ element doesn't exist or cannot be determined (see unitsKnown).
  leftRecip: % -> Union(%,"failed")
    ++ leftRecip(a) returns an element, which is a left inverse of
    ++ \spad{a}, or \spad{"failed"} if there is no unit element, if such
    ++ an element doesn't exist or cannot be determined (see unitsKnown).
  rightRecip: % -> Union(%,"failed")
    ++ rightRecip(a) returns an element, which is a right inverse of
    ++ \spad{a},
    ++ or \spad{"failed"} if there is no unit element, if such an
    ++ element doesn't exist or cannot be determined (see unitsKnown).
  associatorDependence:() -> List Vector R
    ++ associatorDependence() looks for the associator identities, i.e.
    ++ finds a basis of the solutions of the linear combinations of the
    ++ six permutations of \spad{associator(a,b,c)} which yield 0,
    ++ for all \spad{a},b,c in the algebra.
    ++ The order of the permutations is \spad{123 231 312 132 321 213}.
  leftMinimalPolynomial : % -> SparseUnivariatePolynomial R
    ++ leftMinimalPolynomial(a) returns the polynomial determined by the
    ++ smallest non-trivial linear combination of left powers of
    ++ \spad{a}. Note that the polynomial never has a constant term as in
    ++ general the algebra has no unit.
  rightMinimalPolynomial : % -> SparseUnivariatePolynomial R
    ++ rightMinimalPolynomial(a) returns the polynomial determined by the
    ++ smallest non-trivial linear
    ++ combination of right powers of \spad{a}.
    ++ Note that the polynomial never has a constant term as in general
    ++ the algebra has no unit.
  leftUnits:() -> Union(Record(particular: %, basis: List %), "failed")
    ++ leftUnits() returns the affine space of all left units of the
    ++ algebra, or \spad{"failed"} if there is none.
  rightUnits:() -> Union(Record(particular: %, basis: List %), "failed")
    ++ rightUnits() returns the affine space of all right units of the
    ++ algebra, or \spad{"failed"} if there is none.
  leftUnit:() -> Union(%, "failed")
    ++ leftUnit() returns a left unit of the algebra
    ++ (not necessarily unique), or \spad{"failed"} if there is none.
  rightUnit:() -> Union(%, "failed")
    ++ rightUnit() returns a right unit of the algebra
    ++ (not necessarily unique), or \spad{"failed"} if there is none.
  unit:() -> Union(%, "failed")
    ++ unit() returns a unit of the algebra (necessarily unique),
    ++ or \spad{"failed"} if there is none.
  -- we not necessarily have a unit, hence we can't say anything
```

```
    -- about characteristic
    -- if R has CharacteristicZero then CharacteristicZero
    -- if R has CharacteristicNonZero then CharacteristicNonZero
    unitsKnown
      ++ unitsKnown means that \spadfun{recip} truly yields reciprocal
      ++ or \spad{"failed"} if not a unit,
      ++ similarly for \spadfun{leftRecip} and
      ++ \spadfun{rightRecip}. The reason is that we use left, respectively
      ++ right, minimal polynomials to decide this question.
add
  --n := rank()
  --b := someBasis()
  --gamma : Vector Matrix R := structuralConstants b
  -- here is a problem: there seems to be a problem having local
  -- variables in the capsule of a category, furthermore
  -- see the commented code of conditionsForIdempotents, where
  -- we call structuralConstants, which also doesn't work
  -- at runtime, i.e. is not properly inherited, hence for
  -- the moment we put the code for
  -- conditionsForIdempotents, structuralConstants, unit, leftUnit,
  -- rightUnit into the domain constructor ALGSC
  V  ==> Vector
  M  ==> Matrix
  REC  ==> Record(particular: Union(V R,"failed"),basis: List V R)
  LSMP ==> LinearSystemMatrixPackage(R,V R,V R, M R)


  SUP ==>  SparseUnivariatePolynomial
  NNI ==>  NonNegativeInteger
  -- next 2 functions: use a general characteristicPolynomial
  leftCharacteristicPolynomial a ==
     n := rank()$%
     ma : Matrix R := leftRegularRepresentation(a,someBasis()$%)
     mb : Matrix SUP R := zero(n,n)
     for i in 1..n repeat
       for j in 1..n repeat
         mb(i,j):=
           i=j => monomial(ma(i,j),0)$SUP(R) - monomial(1,1)$SUP(R)
           monomial(ma(i,j),1)$SUP(R)
     determinant mb

  rightCharacteristicPolynomial a ==
     n := rank()$%
     ma : Matrix R := rightRegularRepresentation(a,someBasis()$%)
     mb : Matrix SUP R := zero(n,n)
     for i in 1..n repeat
       for j in 1..n repeat
         mb(i,j):=
           i=j => monomial(ma(i,j),0)$SUP(R) - monomial(1,1)$SUP(R)
           monomial(ma(i,j),1)$SUP(R)
```

```
   determinant mb

leftTrace a ==
  t : R := 0
  ma : Matrix R := leftRegularRepresentation(a,someBasis()$%)
  for i in 1..rank()$% repeat
    t := t + elt(ma,i,i)
  t

rightTrace a ==
  t : R := 0
  ma : Matrix R := rightRegularRepresentation(a,someBasis()$%)
  for i in 1..rank()$% repeat
    t := t + elt(ma,i,i)
  t

leftNorm a == determinant leftRegularRepresentation(a,someBasis()$%)

rightNorm a == determinant rightRegularRepresentation(a,someBasis()$%)

antiAssociative?() ==
  b := someBasis()
  n := rank()
  for i in 1..n repeat
    for j in 1..n repeat
      for k in 1..n repeat
        not zero? ( (b.i*b.j)*b.k + b.i*(b.j*b.k) )  =>
          messagePrint("algebra is not anti-associative")$OutputForm
          return false
  messagePrint("algebra is anti-associative")$OutputForm
  true

jordanAdmissible?() ==
  b := someBasis()
  n := rank()
  recip(2 * 1$R) case "failed" =>
    messagePrint("this algebra is not Jordan admissible, " _
     "as 2 is not invertible in the ground ring")$OutputForm
    false
  for i in 1..n repeat
   for j in 1..n repeat
    for k in 1..n repeat
     for l in 1..n repeat
       not zero? ( _
         antiCommutator(antiCommutator(b.i,b.j),_
                        antiCommutator(b.l,b.k)) + _
         antiCommutator(antiCommutator(b.l,b.j),_
                        antiCommutator(b.k,b.i)) + _
         antiCommutator(antiCommutator(b.k,b.j),_
                        antiCommutator(b.i,b.l))   _
```

```
                ) =>
          messagePrint(_
                    "this algebra is not Jordan admissible")$OutputForm
          return false
  messagePrint("this algebra is Jordan admissible")$OutputForm
  true

lieAdmissible?() ==
  n := rank()
  b := someBasis()
  for i in 1..n repeat
   for j in 1..n repeat
    for k in 1..n repeat
      not zero? (commutator(commutator(b.i,b.j),b.k) _
              + commutator(commutator(b.j,b.k),b.i) _
              + commutator(commutator(b.k,b.i),b.j))   =>
        messagePrint("this algebra is not Lie admissible")$OutputForm
        return false
  messagePrint("this algebra is Lie admissible")$OutputForm
  true

-- conditionsForIdempotents b  ==
--   n := rank()
--   gamma : Vector Matrix R := structuralConstants b
--   listOfNumbers : List String :=  [STRINGIMAGE(q)$Lisp for q in 1..n]
--   symbolsForCoef : Vector Symbol :=
--     [concat("%", concat("x", i))::Symbol  for i in listOfNumbers]
--   conditions : List Polynomial R := []
--  for k in 1..n repeat
--    xk := symbolsForCoef.k
--    p : Polynomial R :=  monomial( - 1$Polynomial(R), [xk], [1] )
--    for i in 1..n repeat
--      for j in 1..n repeat
--        xi := symbolsForCoef.i
--        xj := symbolsForCoef.j
--        p := p + monomial(_
--          elt((gamma.k),i,j) :: Polynomial(R), [xi,xj], [1,1])
--    conditions := cons(p,conditions)
--  conditions

structuralConstants b ==
  --n := rank()
  -- be careful with the possibility that b is not a basis
  m : NonNegativeInteger := (maxIndex b) :: NonNegativeInteger
  sC : Vector Matrix R := [new(m,m,0$R) for k in 1..m]
  for i in 1..m repeat
    for j in 1..m repeat
      covec : Vector R := coordinates(b.i * b.j, b)
      for k in 1..m repeat
         setelt( sC.k, i, j, covec.k )
```

```
    sC

  if R has IntegralDomain then

    leftRecip x ==
      zero? x => "failed"
      lu := leftUnit()
      lu case "failed" => "failed"
      b := someBasis()
      xx : % := (lu :: %)
      k  : PositiveInteger := 1
      cond : Matrix R := coordinates(xx,b) :: Matrix(R)
      listOfPowers : List % := [xx]
      while rank(cond) = k repeat
        k := k+1
        xx := xx*x
        listOfPowers := cons(xx,listOfPowers)
        cond := horizConcat(cond, coordinates(xx,b) :: Matrix(R) )
      vectorOfCoef : Vector R := (nullSpace(cond)$Matrix(R)).first
      invC := recip vectorOfCoef.1
      invC case "failed" => "failed"
      invCR : R :=  - (invC :: R)
      reduce(_+,[(invCR*vectorOfCoef.i)*power for i in _
       2..maxIndex vectorOfCoef for power in reverse listOfPowers])

    rightRecip x ==
      zero? x => "failed"
      ru := rightUnit()
      ru case "failed" => "failed"
      b := someBasis()
      xx : % := (ru :: %)
      k  : PositiveInteger := 1
      cond : Matrix R := coordinates(xx,b) :: Matrix(R)
      listOfPowers : List % := [xx]
      while rank(cond) = k repeat
        k := k+1
        xx := x*xx
        listOfPowers := cons(xx,listOfPowers)
        cond := horizConcat(cond, coordinates(xx,b) :: Matrix(R) )
      vectorOfCoef : Vector R := (nullSpace(cond)$Matrix(R)).first
      invC := recip vectorOfCoef.1
      invC case "failed" => "failed"
      invCR : R :=  - (invC :: R)
      reduce(_+,[(invCR*vectorOfCoef.i)*power for i in _
       2..maxIndex vectorOfCoef for power in reverse listOfPowers])

    recip x ==
      lrx := leftRecip x
      lrx case "failed" => "failed"
      rrx := rightRecip x
```

```
  rrx case "failed" => "failed"
  (lrx :: %) ^= (rrx :: %)  => "failed"
  lrx :: %

leftMinimalPolynomial x ==
  zero? x =>  monomial(1$R,1)$(SparseUnivariatePolynomial R)
  b := someBasis()
  xx : % := x
  k  : PositiveInteger := 1
  cond : Matrix R := coordinates(xx,b) :: Matrix(R)
  while rank(cond) = k repeat
    k := k+1
    xx := x*xx
    cond := horizConcat(cond, coordinates(xx,b) :: Matrix(R) )
  vectorOfCoef : Vector R := (nullSpace(cond)$Matrix(R)).first
  res : SparseUnivariatePolynomial R := 0
  for i in 1..k repeat
    res:=res+monomial(vectorOfCoef.i,i)$(SparseUnivariatePolynomial R)
  res

rightMinimalPolynomial x ==
  zero? x =>  monomial(1$R,1)$(SparseUnivariatePolynomial R)
  b := someBasis()
  xx : % := x
  k  : PositiveInteger := 1
  cond : Matrix R := coordinates(xx,b) :: Matrix(R)
  while rank(cond) = k repeat
    k := k+1
    xx := xx*x
    cond := horizConcat(cond, coordinates(xx,b) :: Matrix(R) )
  vectorOfCoef : Vector R := (nullSpace(cond)$Matrix(R)).first
  res : SparseUnivariatePolynomial R := 0
  for i in 1..k repeat
    res:=res+monomial(vectorOfCoef.i,i)$(SparseUnivariatePolynomial R)
  res

associatorDependence() ==
  n := rank()
  b := someBasis()
  cond : Matrix(R) := new(n**4,6,0$R)$Matrix(R)
  z : Integer := 0
  for i in 1..n repeat
   for j in 1..n repeat
    for k in 1..n repeat
     a123 : Vector R := coordinates(associator(b.i,b.j,b.k),b)
     a231 : Vector R := coordinates(associator(b.j,b.k,b.i),b)
     a312 : Vector R := coordinates(associator(b.k,b.i,b.j),b)
     a132 : Vector R := coordinates(associator(b.i,b.k,b.j),b)
     a321 : Vector R := coordinates(associator(b.k,b.j,b.i),b)
     a213 : Vector R := coordinates(associator(b.j,b.i,b.k),b)
```

```
        for r in 1..n repeat
         z:= z+1
         setelt(cond,z,1,elt(a123,r))
         setelt(cond,z,2,elt(a231,r))
         setelt(cond,z,3,elt(a312,r))
         setelt(cond,z,4,elt(a132,r))
         setelt(cond,z,5,elt(a321,r))
         setelt(cond,z,6,elt(a213,r))
    nullSpace(cond)

jacobiIdentity?()  ==
  n := rank()
  b := someBasis()
  for i in 1..n repeat
   for j in 1..n repeat
    for k in 1..n repeat
      not zero? ((b.i*b.j)*b.k + (b.j*b.k)*b.i + (b.k*b.i)*b.j) =>
        messagePrint("Jacobi identity does not hold")$OutputForm
        return false
  messagePrint("Jacobi identity holds")$OutputForm
  true

lieAlgebra?()  ==
  not antiCommutative?() =>
    messagePrint("this is not a Lie algebra")$OutputForm
    false
  not jacobiIdentity?() =>
    messagePrint("this is not a Lie algebra")$OutputForm
    false
  messagePrint("this is a Lie algebra")$OutputForm
  true

jordanAlgebra?()  ==
  b := someBasis()
  n := rank()
  recip(2 * 1$R) case "failed" =>
    messagePrint("this is not a Jordan algebra, as 2 is not " _
     "invertible in the ground ring")$OutputForm
    false
  not commutative?() =>
    messagePrint("this is not a Jordan algebra")$OutputForm
    false
  for i in 1..n repeat
   for j in 1..n repeat
    for k in 1..n repeat
     for l in 1..n repeat
       not zero? (associator(b.i,b.j,b.l*b.k)+_
           associator(b.l,b.j,b.k*b.i)+associator(b.k,b.j,b.i*b.l)) =>
         messagePrint("not a Jordan algebra")$OutputForm
         return false
```

```
      messagePrint("this is a Jordan algebra")$OutputForm
      true

    noncommutativeJordanAlgebra?() ==
      b := someBasis()
      n := rank()
      recip(2 * 1$R) case "failed" =>
       messagePrint("this is not a noncommutative Jordan algebra,_
as 2 is not invertible in the ground ring")$OutputForm
       false
      not flexible?()$% =>
       messagePrint("this is not a noncommutative Jordan algebra,_
as it is not flexible")$OutputForm
       false
      not jordanAdmissible?()$% =>
       messagePrint("this is not a noncommutative Jordan algebra,_
as it is not Jordan admissible")$OutputForm
       false
      messagePrint("this is a noncommutative Jordan algebra")$OutputForm
      true

    antiCommutative?() ==
      b := someBasis()
      n := rank()
      for i in 1..n repeat
        for j in i..n repeat
          not zero? (i=j => b.i*b.i; b.i*b.j + b.j*b.i) =>
            messagePrint("algebra is not anti-commutative")$OutputForm
            return false
      messagePrint("algebra is anti-commutative")$OutputForm
      true

    commutative?() ==
      b := someBasis()
      n := rank()
      for i in 1..n repeat
       for j in i+1..n repeat
         not zero? commutator(b.i,b.j) =>
           messagePrint("algebra is not commutative")$OutputForm
           return false
      messagePrint("algebra is commutative")$OutputForm
      true

    associative?() ==
      b := someBasis()
      n := rank()
      for i in 1..n repeat
       for j in 1..n repeat
        for k in 1..n repeat
         not zero? associator(b.i,b.j,b.k) =>
```

```
        messagePrint("algebra is not associative")$OutputForm
        return false
    messagePrint("algebra is associative")$OutputForm
    true

  leftAlternative?() ==
    b := someBasis()
    n := rank()
    for i in 1..n repeat
     for j in 1..n repeat
      for k in 1..n repeat
       not zero? (associator(b.i,b.j,b.k) + associator(b.j,b.i,b.k)) =>
         messagePrint("algebra is not left alternative")$OutputForm
         return false
    messagePrint("algebra satisfies 2*associator(a,a,b) = 0")$OutputForm
    true

  rightAlternative?() ==
    b := someBasis()
    n := rank()
    for i in 1..n repeat
     for j in 1..n repeat
      for k in 1..n repeat
       not zero? (associator(b.i,b.j,b.k) + associator(b.i,b.k,b.j)) =>
         messagePrint("algebra is not right alternative")$OutputForm
         return false
    messagePrint("algebra satisfies 2*associator(a,b,b) = 0")$OutputForm
    true

  flexible?() ==
    b := someBasis()
    n := rank()
    for i in 1..n repeat
     for j in 1..n repeat
      for k in 1..n repeat
       not zero? (associator(b.i,b.j,b.k) + associator(b.k,b.j,b.i)) =>
         messagePrint("algebra is not flexible")$OutputForm
         return false
    messagePrint("algebra satisfies 2*associator(a,b,a) = 0")$OutputForm
    true

  alternative?() ==
    b := someBasis()
    n := rank()
    for i in 1..n repeat
     for j in 1..n repeat
      for k in 1..n repeat
       not zero? (associator(b.i,b.j,b.k) + associator(b.j,b.i,b.k)) =>
         messagePrint("algebra is not alternative")$OutputForm
         return false
```

```
        not zero? (associator(b.i,b.j,b.k) + associator(b.i,b.k,b.j)) =>
          messagePrint("algebra is not alternative")$OutputForm
          return false
      messagePrint("algebra satisfies 2*associator(a,b,b) = 0 " _
                   "=  2*associator(a,a,b) = 0")$OutputForm
      true

    leftDiscriminant v == determinant leftTraceMatrix v
    rightDiscriminant v == determinant rightTraceMatrix v

    coordinates(v:Vector %, b:Vector %) ==
      m := new(#v, #b, 0)$Matrix(R)
      for i in minIndex v .. maxIndex v for j in minRowIndex m .. repeat
        setRow_!(m, j, coordinates(qelt(v, i), b))
      m

    represents(v, b) ==
      m := minIndex v - 1
      reduce(_+,[v(i+m) * b(i+m) for i in 1..maxIndex b])

    leftTraceMatrix v ==
      matrix [[leftTrace(v.i*v.j) for j in minIndex v..maxIndex v]$List(R)
               for i in minIndex v .. maxIndex v]$List(List R)

    rightTraceMatrix v ==
      matrix [[rightTrace(v.i*v.j) for j in minIndex v..maxIndex v]$List(R)
               for i in minIndex v .. maxIndex v]$List(List R)

    leftRegularRepresentation(x, b) ==
      m := minIndex b - 1
      matrix
       [parts coordinates(x*b(i+m),b) for i in 1..rank()]$List(List R)

    rightRegularRepresentation(x, b) ==
      m := minIndex b - 1
      matrix
       [parts coordinates(b(i+m)*x,b) for i in 1..rank()]$List(List R)
```

———-

— FINAALG.dotabb —

```
"FINAALG"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FINAALG"];
"FINAALG" -> "NAALG"
```

———-

— **FINAALG.dotfull** —

```
"FiniteRankNonAssociativeAlgebra(a:CommutativeRing)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FINAALG"];
"FiniteRankNonAssociativeAlgebra(a:CommutativeRing)" ->
    "NonAssociativeAlgebra(a:CommutativeRing)"
```

———-

— **FINAALG.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FiniteRankNonAssociativeAlgebra(a:CommutativeRing)" [color=lightblue];
"FiniteRankNonAssociativeAlgebra(a:CommutativeRing)" ->
    "NonAssociativeAlgebra(a:CommutativeRing)"

"NonAssociativeAlgebra(a:CommutativeRing)" [color=lightblue];
"NonAssociativeAlgebra(a:CommutativeRing)" -> "NonAssociativeRng()"
"NonAssociativeAlgebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"

"NonAssociativeRng()" [color=lightblue];
"NonAssociativeRng()" -> "ABELGRP..."
"NonAssociativeRng()" -> "Monad()"

"Monad()" [color=lightblue];
"Monad()" -> "SETCAT..."
"Monad()" -> "REPSQ..."

"Module(a:CommutativeRing)" [color=lightblue];
"Module(a:CommutativeRing)" ->
  "BiModule(a:CommutativeRing,b:CommutativeRing)"

"BiModule(a:CommutativeRing,b:CommutativeRing)" [color=seagreen];
"BiModule(a:CommutativeRing,b:CommutativeRing)" -> "BiModule(a:Ring,b:Ring)"

"BiModule(a:Ring,b:Ring)" [color=lightblue];
"BiModule(a:Ring,b:Ring)" -> "LeftModule(a:Ring)"
"BiModule(a:Ring,b:Ring)" -> "RightModule(a:Ring)"

"RightModule(a:Ring)" [color=seagreen];
"RightModule(a:Ring)" -> "RightModule(a:Rng)"

"RightModule(a:Rng)" [color=lightblue];
```

```
"RightModule(a:Rng)" -> "ABELGRP..."

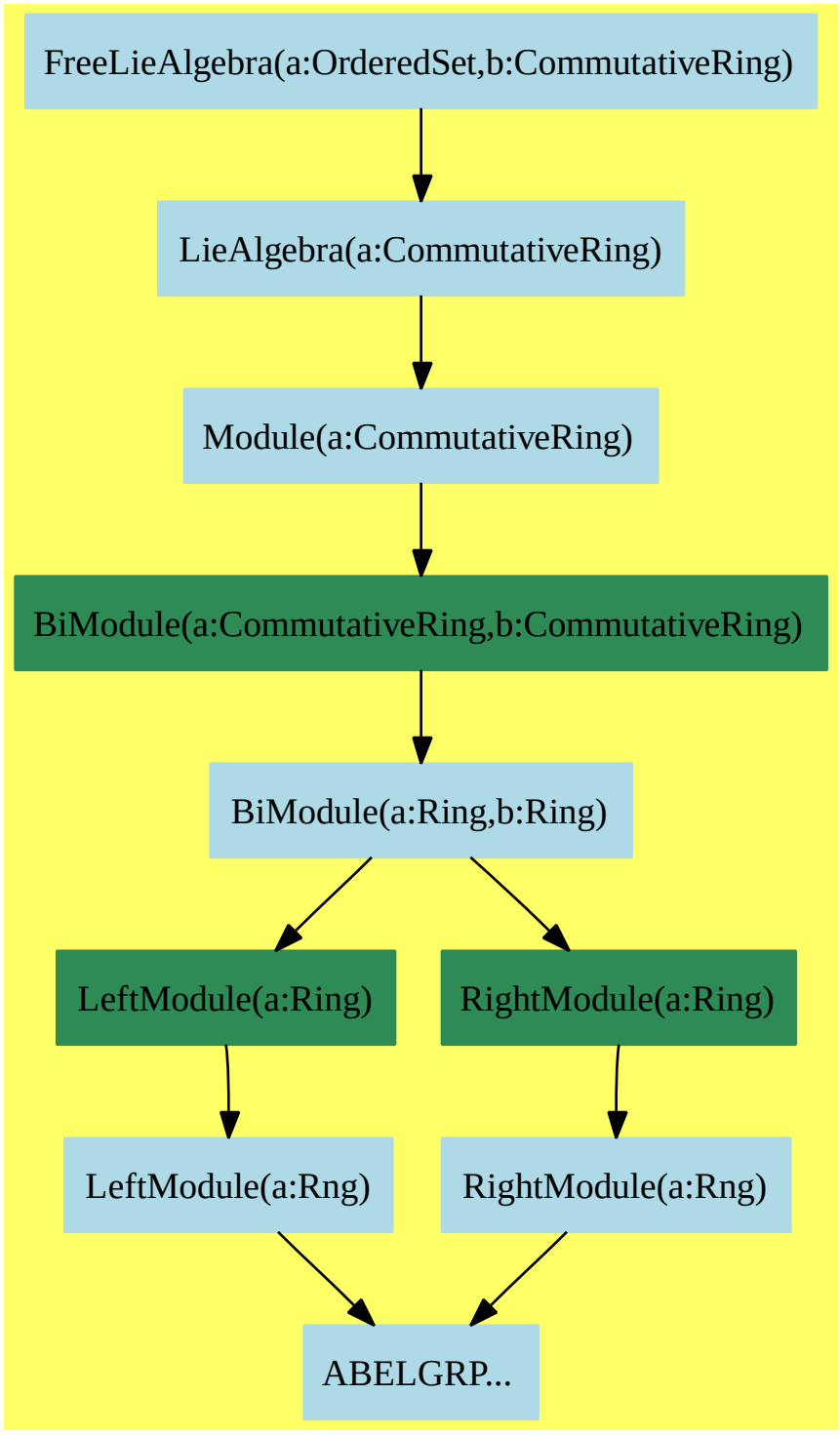
"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "ABELGRP..."

"REPSQ..." [color="#00EE00"];
"SETCAT..." [color=lightblue];
"ABELGRP..." [color=lightblue];
}
```

———-

12.0.163 FreeLieAlgebra (FLALG)



— FreeLieAlgebra.input —

```
)set break resume
)sys rm -f FreeLieAlgebra.output
)spool FreeLieAlgebra.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FreeLieAlgebra
--R FreeLieAlgebra(VarSet: OrderedSet,R: CommutativeRing)  is a category constructor
--R Abbreviation for FreeLieAlgebra is FLALG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FLALG
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,R) -> %                      ?*? : (R,%) -> %
--R ?*? : (Integer,%) -> %                ?*? : (PositiveInteger,%) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R LiePoly : LyndonWord VarSet -> %      0 : () -> %
--R coerce : VarSet -> %                  coerce : % -> OutputForm
--R construct : (%,%) -> %                degree : % -> NonNegativeInteger
--R eval : (%,VarSet,%) -> %              hash : % -> SingleInteger
--R latex : % -> String                   mirror : % -> %
--R sample : () -> %                      varList : % -> List VarSet
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?/? : (%,R) -> % if R has FIELD
--R coef : (XRecursivePolynomial(VarSet,R),%) -> R
--R coerce : % -> XRecursivePolynomial(VarSet,R)
--R coerce : % -> XDistributedPolynomial(VarSet,R)
--R eval : (%,List VarSet,List %) -> %
--R lquo : (XRecursivePolynomial(VarSet,R),%) -> XRecursivePolynomial(VarSet,R)
--R rquo : (XRecursivePolynomial(VarSet,R),%) -> XRecursivePolynomial(VarSet,R)
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R trunc : (%,NonNegativeInteger) -> %
--R
--E 1

)spool
)lisp (bye)
```

————-

— FreeLieAlgebra.help —

```
================================================================
FreeLieAlgebra examples
================================================================

See Also:
o )show FreeLieAlgebra
```

———-

See:

⇐ “LieAlgebra” (LIECAT) 11.0.155 on page 966

Exports:

0	coef	coerce	construct	degree
eval	hash	latex	LiePoly	lquo
mirror	rquo	sample	subtractIfCan	trunc
varList	zero?	?~=?	?*?	?/?
?+?	?-?	-?	?=?	

Attributes Exported:

- **NullSquare** means that $[x, x] = 0$ holds. See `LieAlgebra`.
- **JacobiIdentity** means that $[x, [y, z]] + [y, [z, x]] + [z, [x, y]] = 0$ holds. See `LieAlgebra`.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
coef : (XRecursivePolynomial(VarSet,R),%) -> R
coerce : VarSet -> %
coerce : % -> XRecursivePolynomial(VarSet,R)
coerce : % -> XDistributedPolynomial(VarSet,R)
degree : % -> NonNegativeInteger
eval : (%,List VarSet,List %) -> %
eval : (%,VarSet,%) -> %
LiePoly : LyndonWord VarSet -> %
lquo : (XRecursivePolynomial(VarSet,R),%) -> XRecursivePolynomial(VarSet,R)
mirror : % -> %
rquo : (XRecursivePolynomial(VarSet,R),%) -> XRecursivePolynomial(VarSet,R)
trunc : (%,NonNegativeInteger) -> %
varList : % -> List VarSet
```

These exports come from (p966) LieAlgebra(CommutativeRing):

```
 0 : () -> %
 coerce : % -> OutputForm
 construct : (%,%) -> %
 hash : % -> SingleInteger
 latex : % -> String
 sample : () -> %
 subtractIfCan : (%,%) -> Union(%,"failed")
 zero? : % -> Boolean
 ?/? : (%,R) -> % if R has FIELD
 ?~=? : (%,%) -> Boolean
 ?*? : (NonNegativeInteger,%) -> %
 ?*? : (%,R) -> %
 ?*? : (R,%) -> %
 ?*? : (Integer,%) -> %
 ?*? : (PositiveInteger,%) -> %
 ?+? : (%,%) -> %
 ?-? : (%,%) -> %
 -? : % -> %
 ?=? : (%,%) -> Boolean
```

— category FLALG FreeLieAlgebra —

```
)abbrev category FLALG FreeLieAlgebra
++ Author: Michel Petitot (petitot@lifl.fr)
++ Date Created: 91
++ Date Last Updated: 7 Juillet 92
++ Fix History: compilation v 2.1 le 13 dec 98
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The category of free Lie algebras.
++ It is used by domains of non-commutative algebra:
++ \spadtype{LiePolynomial} and
++ \spadtype{XPBWPolynomial}.

FreeLieAlgebra(VarSet:OrderedSet, R:CommutativeRing) :Category == _
 CatDef where
   XRPOLY  ==> XRecursivePolynomial(VarSet,R)
   XDPOLY  ==> XDistributedPolynomial(VarSet,R)
   RN      ==> Fraction Integer
   LWORD   ==> LyndonWord(VarSet)

   CatDef ==  Join(LieAlgebra(R)) with
     coef      : (XRPOLY , $) -> R
       ++ \axiom{coef(x,y)} returns the scalar product of \axiom{x} by
```

```
      ++ \axiom{y}, the set of words being regarded as an orthogonal basis.
    coerce    : VarSet -> $
      ++ \axiom{coerce(x)} returns \axiom{x} as a Lie polynomial.
    coerce    : $ -> XDPOLY
      ++ \axiom{coerce(x)} returns \axiom{x} as distributed polynomial.
    coerce    : $ -> XRPOLY
      ++ \axiom{coerce(x)} returns \axiom{x} as a recursive polynomial.
    degree    : $ -> NonNegativeInteger
      ++ \axiom{degree(x)} returns the greatest length of a word in the
      ++ support of \axiom{x}.
    --if R has Module(RN) then
    --  Hausdorff : ($,$,PositiveInteger) -> $
    lquo      : (XRPOLY , $) -> XRPOLY
      ++ \axiom{lquo(x,y)} returns the left simplification of \axiom{x}
      ++ by \axiom{y}.
    rquo      : (XRPOLY , $) -> XRPOLY
      ++ \axiom{rquo(x,y)} returns the right simplification of \axiom{x}
      ++ by \axiom{y}.
    LiePoly   : LWORD -> $
      ++ \axiom{LiePoly(l)} returns the bracketed form of \axiom{l} as
      ++ a Lie polynomial.
    mirror    : $ -> $
      ++ \axiom{mirror(x)} returns \axiom{Sum(r_i mirror(w_i))}
      ++ if \axiom{x} is \axiom{Sum(r_i w_i)}.
    trunc     : ($, NonNegativeInteger) -> $
      ++ \axiom{trunc(p,n)} returns the polynomial \axiom{p}
      ++ truncated at order \axiom{n}.
    varList   : $ -> List VarSet
      ++ \axiom{varList(x)} returns the list of distinct entries
      ++ of \axiom{x}.
    eval      : ($, VarSet, $) -> $
      ++ \axiom{eval(p, x, v)} replaces \axiom{x} by \axiom{v}
      ++ in \axiom{p}.
    eval      : ($, List VarSet, List $) -> $
      ++ \axiom{eval(p, [x1,...,xn], [v1,...,vn])} replaces \axiom{xi}
      ++ by \axiom{vi} in \axiom{p}.
```

— FLALG.dotabb —

```
"FLALG"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FLALG"];
"FLALG" -> "LIECAT"
```

— **FLALG.dotfull** —

```
"FreeLieAlgebra(a:OrderedSet,b:CommutativeRing)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FLALG"];
"FreeLieAlgebra(a:OrderedSet,b:CommutativeRing)" ->
   "LieAlgebra(a:CommutativeRing)"
```

———-

— **FLALG.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FreeLieAlgebra(a:OrderedSet,b:CommutativeRing)" [color=lightblue];
"FreeLieAlgebra(a:OrderedSet,b:CommutativeRing)" ->
   "LieAlgebra(a:CommutativeRing)"

"LieAlgebra(a:CommutativeRing)" [color=lightblue];
"LieAlgebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"

"Module(a:CommutativeRing)" [color=lightblue];
"Module(a:CommutativeRing)" ->
  "BiModule(a:CommutativeRing,b:CommutativeRing)"

"BiModule(a:CommutativeRing,b:CommutativeRing)" [color=seagreen];
"BiModule(a:CommutativeRing,b:CommutativeRing)" -> "BiModule(a:Ring,b:Ring)"

"BiModule(a:Ring,b:Ring)" [color=lightblue];
"BiModule(a:Ring,b:Ring)" -> "LeftModule(a:Ring)"
"BiModule(a:Ring,b:Ring)" -> "RightModule(a:Ring)"

"RightModule(a:Ring)" [color=seagreen];
"RightModule(a:Ring)" -> "RightModule(a:Rng)"

"RightModule(a:Rng)" [color=lightblue];
"RightModule(a:Rng)" -> "ABELGRP..."

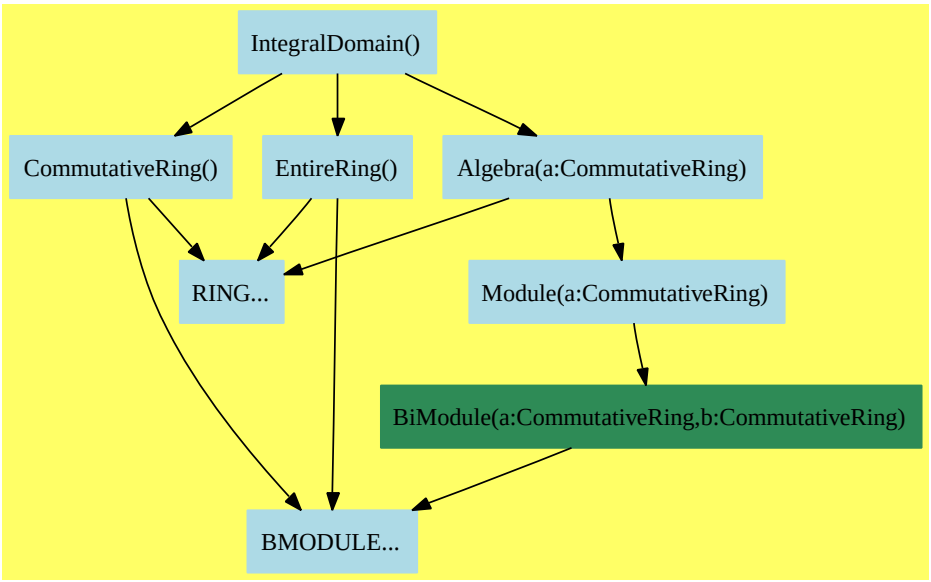
"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "ABELGRP..."

"ABELGRP..." [color=lightblue];
}
```

———-

12.0.164 IntegralDomain (INTDOM)



— **IntegralDomain.input** —

```
)set break resume
)sys rm -f IntegralDomain.output
)spool IntegralDomain.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show IntegralDomain
--R IntegralDomain  is a category constructor
--R Abbreviation for IntegralDomain is INTDOM
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for INTDOM
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,%) -> %                    ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %      ?**? : (%,PositiveInteger) -> %
```

```
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        associates? : (%,%) -> Boolean
--R coerce : % -> %                       coerce : Integer -> %
--R coerce : % -> OutputForm              hash : % -> SingleInteger
--R latex : % -> String                   one? : % -> Boolean
--R recip : % -> Union(%,"failed")        sample : () -> %
--R unit? : % -> Boolean                  unitCanonical : % -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R exquo : (%,%) -> Union(%,"failed")
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R
--E 1

)spool
)lisp (bye)
```

———-

— **IntegralDomain.help** —

```
=====================================================================
IntegralDomain examples
=====================================================================

See Also:
o )show IntegralDomain
```

———-

All Commutative Rings are Integral Domains.
⇐ “CommutativeRing” (COMRING) 10.0.137 on page 843 Integral Domains are a subset of Unique Factorization domains.
⇒ “UniqueFactorizationDomain” (UFD) 14.0.179 on page 1172.

See:

⇒ “FortranMachineTypeCategory” (FMTC) 13.0.171 on page 1108
⇒ “FunctionSpace” (FS) 17.0.195 on page 1336
⇒ “GcdDomain” (GCDDOM) 13.0.173 on page 1129
⇒ “OrderedIntegralDomain” (OINTDOM) 13.0.174 on page 1135
⇒ “UnivariatePolynomialCategory” (UPOLYC) 17.0.204 on page 1471

⇐ "Algebra" (ALGEBRA) 11.0.152 on page 947
⇐ "CommutativeRing" (COMRING) 10.0.137 on page 843
⇐ "EntireRing" (ENTIRER) 10.0.139 on page 853

Exports:

0	1	associates?	characteristic	coerce
exquo	hash	latex	one?	recip
sample	subtractIfCan	unit?	unitCanonical	unitNormal
zero?	?~=?	?*?	?**?	?^?
?+?	?-?	-?	?=?	

Attributes exported:

- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

Attributes Used:

- **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.

These are directly exported but not implemented:

```
exquo : (%,%) -> Union(%,"failed")
```

These are implemented by this category:

```
associates? : (%,%) -> Boolean
unit? : % -> Boolean
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
```

These exports come from (p843) CommutativeRing():

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : Integer -> %
```

```
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?^? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
```

TPDHERE: Should we construct this coercion?

These exports come from (p947) Algebra(a:IntegralDomain):

```
coerce : % -> %
```

— category INTDOM IntegralDomain —

```
)abbrev category INTDOM IntegralDomain
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References: Davenport & Trager I
++ Description:
++ The category of commutative integral domains, i.e. commutative
++ rings with no zero divisors.
++
++ Conditional attributes\br
++ canonicalUnitNormal\tab{5}the canonical field is the same for
++ all associates\br
++ canonicalsClosed\tab{5}the product of two canonicals is itself canonical
```

```
IntegralDomain(): Category ==
  Join(CommutativeRing, Algebra(%), EntireRing) with
    "exquo": (%,%) -> Union(%,"failed")
        ++ exquo(a,b) either returns an element c such that
        ++ \spad{c*b=a} or "failed" if no such element can be found.
    unitNormal: % -> Record(unit:%,canonical:%,associate:%)
        ++ unitNormal(x) tries to choose a canonical element
        ++ from the associate class of x.
        ++ The attribute canonicalUnitNormal, if asserted, means that
        ++ the "canonical" element is the same across all associates of x
        ++ if \spad{unitNormal(x) = [u,c,a]} then
        ++ \spad{u*c = x}, \spad{a*u = 1}.
    unitCanonical: % -> %
        ++ \spad{unitCanonical(x)} returns \spad{unitNormal(x).canonical}.
    associates?: (%,%) -> Boolean
        ++ associates?(x,y) tests whether x and y are associates, i.e.
        ++ differ by a unit factor.
    unit?: % -> Boolean
        ++ unit?(x) tests whether x is a unit, i.e. is invertible.
 add
     x,y: %

     UCA ==> Record(unit:%,canonical:%,associate:%)
     if not (% has Field) then
       unitNormal(x) == [1$%,x,1$%]$UCA -- the non-canonical definition
     unitCanonical(x) == unitNormal(x).canonical -- always true
     recip(x) == if zero? x then "failed" else _exquo(1$%,x)
     unit?(x) == (recip x case "failed" => false; true)
     if % has canonicalUnitNormal then
        associates?(x,y) ==
          (unitNormal x).canonical = (unitNormal y).canonical
      else
        associates?(x,y) ==
          zero? x => zero? y
          zero? y => false
          x exquo y case "failed" => false
          y exquo x case "failed" => false
          true
```

———-

— INTDOM.dotabb —

```
"INTDOM"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=INTDOM"];
"INTDOM" -> "COMRING"
"INTDOM" -> "ALGEBRA"
"INTDOM" -> "ENTIRER"
```

———-

— INTDOM.dotfull —

```
"IntegralDomain()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=INTDOM"];
"IntegralDomain()" -> "CommutativeRing()"
"IntegralDomain()" -> "Algebra(a:CommutativeRing)"
"IntegralDomain()" -> "EntireRing()"
```

———-

— INTDOM.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"IntegralDomain()" [color=lightblue];
"IntegralDomain()" -> "CommutativeRing()"
"IntegralDomain()" -> "Algebra(a:CommutativeRing)"
"IntegralDomain()" -> "EntireRing()"

"CommutativeRing()" [color=lightblue];
"CommutativeRing()" -> "RING..."
"CommutativeRing()" -> "BMODULE..."

"EntireRing()" [color=lightblue];
"EntireRing()" -> "RING..."
"EntireRing()" -> "BMODULE..."

"Algebra(a:CommutativeRing)" [color=lightblue];
"Algebra(a:CommutativeRing)" -> "RING..."
"Algebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"

"Module(a:CommutativeRing)" [color=lightblue];
"Module(a:CommutativeRing)" ->
  "BiModule(a:CommutativeRing,b:CommutativeRing)"

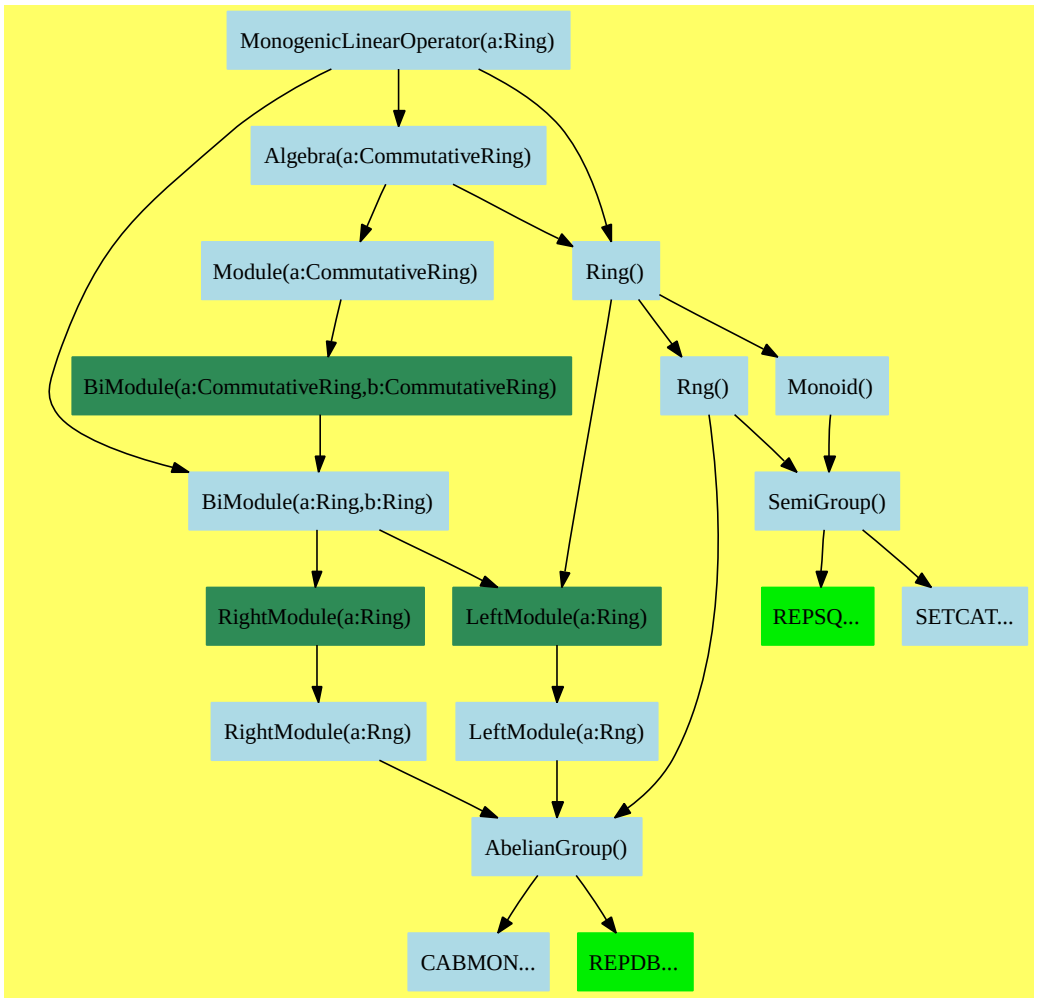
"BiModule(a:CommutativeRing,b:CommutativeRing)" [color=seagreen];
"BiModule(a:CommutativeRing,b:CommutativeRing)" -> "BMODULE..."

"BMODULE..." [color=lightblue];
"RING..." [color=lightblue];
```

```
}
```

———-

12.0.165 MonogenicLinearOperator (MLO)



— **MonogenicLinearOperator.input** —

```
)set break resume
```

```
)sys rm -f MonogenicLinearOperator.output
)spool MonogenicLinearOperator.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show MonogenicLinearOperator
--R MonogenicLinearOperator R: Ring  is a category constructor
--R Abbreviation for MonogenicLinearOperator is MLO
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for MLO
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (R,%) -> %                      ?*? : (%,R) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        coerce : Integer -> %
--R coerce : % -> OutputForm              degree : % -> NonNegativeInteger
--R hash : % -> SingleInteger             latex : % -> String
--R leadingCoefficient : % -> R           one? : % -> Boolean
--R recip : % -> Union(%,"failed")        reductum : % -> %
--R sample : () -> %                      zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R coefficient : (%,NonNegativeInteger) -> R
--R coerce : R -> % if R has COMRING
--R minimumDegree : % -> NonNegativeInteger
--R monomial : (R,NonNegativeInteger) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— MonogenicLinearOperator.help —

```
====================================================================
MonogenicLinearOperator examples
====================================================================
```

```
See Also:
o )show MonogenicLinearOperator
```

———————-

See:

⇐ “Algebra” (ALGEBRA) 11.0.152 on page 947
⇐ “BiModule” (BMODULE) 9.0.122 on page 726
⇐ “Ring” (RING) 9.0.129 on page 770

Exports:

0	1	characteristic	coefficient
coerce	degree	hash	latex
leadingCoefficient	minimumDegree	monomial	one?
recip	reductum	sample	subtractIfCan
zero?	?*?	?**?	?+?
?-?	-?	?=?	?^?
?~=?			

Attributes Exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return “failed” if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
coefficient : (%,NonNegativeInteger) -> R
degree : % -> NonNegativeInteger
leadingCoefficient : % -> R
minimumDegree : % -> NonNegativeInteger
monomial : (R,NonNegativeInteger) -> %
reductum : % -> %
```

These exports come from (p770) Ring():

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : % -> OutputForm
coerce : Integer -> %
hash : % -> SingleInteger
```

```
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (Integer,%) -> %
?*? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?^? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,NonNegativeInteger) -> %
```

These exports come from (p726) BiModule(R:Ring,R:Ring):

```
?*? : (R,%) -> %
?*? : (%,R) -> %
```

These exports come from (p947) Algebra(R:CommutativeRing):

```
coerce : R -> % if R has COMRING
```

— category MLO MonogenicLinearOperator —

```
)abbrev category MLO MonogenicLinearOperator
++ Author: Stephen M. Watt
++ Date Created: 1986
++ Date Last Updated: May 30, 1991
++ Basic Operations:
++ Related Domains:  NonCommutativeOperatorDivision
++ Also See:
++ AMS Classifications:
++ Keywords:
++ Examples:
++ References:
++ Description:
++ This is the category of linear operator rings with one generator.
++ The generator is not named by the category but can always be
++ constructed as \spad{monomial(1,1)}.
++
++ For convenience, call the generator \spad{G}.
```

```
++ Then each value is equal to
++ \spad{sum(a(i)*G**i, i = 0..n)}
++ for some unique \spad{n} and \spad{a(i)} in \spad{R}.
++
++ Note that multiplication is not necessarily commutative.
++ In fact,  if \spad{a} is in \spad{R}, it is quite normal
++ to have \spad{a*G \^= G*a}.

MonogenicLinearOperator(R): Category == Defn where
    E ==> NonNegativeInteger
    R: Ring
    Defn == Join(Ring, BiModule(R,R)) with
        if R has CommutativeRing then Algebra(R)
        degree: $ -> E
            ++ degree(l) is \spad{n} if
            ++   \spad{l = sum(monomial(a(i),i), i = 0..n)}.
        minimumDegree: $ -> E
            ++ minimumDegree(l) is the smallest \spad{k} such that
            ++ \spad{a(k) \^= 0} if
            ++   \spad{l = sum(monomial(a(i),i), i = 0..n)}.
        leadingCoefficient: $ -> R
            ++ leadingCoefficient(l) is \spad{a(n)} if
            ++   \spad{l = sum(monomial(a(i),i), i = 0..n)}.
        reductum: $ -> $
            ++ reductum(l) is \spad{l - monomial(a(n),n)} if
            ++   \spad{l = sum(monomial(a(i),i), i = 0..n)}.
        coefficient: ($, E) -> R
            ++ coefficient(l,k) is \spad{a(k)} if
            ++   \spad{l = sum(monomial(a(i),i), i = 0..n)}.
        monomial: (R, E) -> $
            ++ monomial(c,k) produces c times the k-th power of
            ++ the generating operator, \spad{monomial(1,1)}.
```

———-

— **MLO.dotabb** —

```
"MLO"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=MLO"];
"MLO" -> "BMODULE"
"MLO" -> "RING"
"MLO" -> "ALGEBRA"
```

———-

— **MLO.dotfull** —

```
"MonogenicLinearOperator(a:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=MLO"];
"MonogenicLinearOperator(a:Ring)" -> "Ring()"
"MonogenicLinearOperator(a:Ring)" -> "BiModule(a:Ring,b:Ring)"
"MonogenicLinearOperator(a:Ring)" -> "Algebra(a:CommutativeRing)"
```

———-

— MLO.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"MonogenicLinearOperator(a:Ring)" [color=lightblue];
"MonogenicLinearOperator(a:Ring)" -> "Ring()"
"MonogenicLinearOperator(a:Ring)" -> "BiModule(a:Ring,b:Ring)"
"MonogenicLinearOperator(a:Ring)" -> "Algebra(a:CommutativeRing)"

"Algebra(a:CommutativeRing)" [color=lightblue];
"Algebra(a:CommutativeRing)" -> "Ring()"
"Algebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"

"Ring()" [color=lightblue];
"Ring()" -> "Rng()"
"Ring()" -> "Monoid()"
"Ring()" -> "LeftModule(a:Ring)"

"Module(a:CommutativeRing)" [color=lightblue];
"Module(a:CommutativeRing)" ->
  "BiModule(a:CommutativeRing,b:CommutativeRing)"

"BiModule(a:CommutativeRing,b:CommutativeRing)" [color=seagreen];
"BiModule(a:CommutativeRing,b:CommutativeRing)" -> "BiModule(a:Ring,b:Ring)"

"BiModule(a:Ring,b:Ring)" [color=lightblue];
"BiModule(a:Ring,b:Ring)" -> "LeftModule(a:Ring)"
"BiModule(a:Ring,b:Ring)" -> "RightModule(a:Ring)"

"Rng()" [color=lightblue];
"Rng()" -> "AbelianGroup()"
"Rng()" -> "SemiGroup()"

"Monoid()" [color=lightblue];
"Monoid()" -> "SemiGroup()"

"RightModule(a:Ring)" [color=seagreen];
```

```
"RightModule(a:Ring)" -> "RightModule(a:Rng)"

"RightModule(a:Rng)" [color=lightblue];
"RightModule(a:Rng)" -> "AbelianGroup()"

"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "AbelianGroup()"

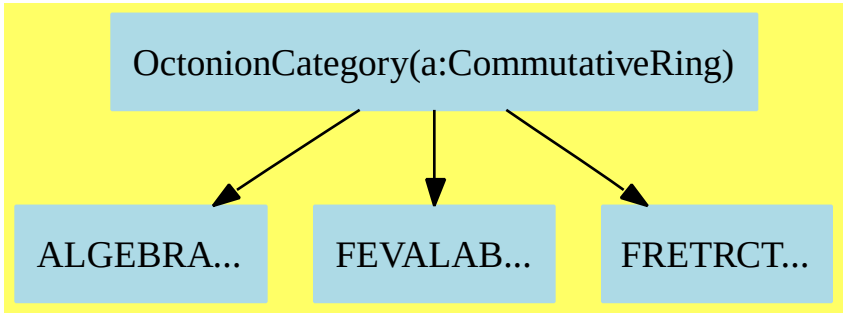
"AbelianGroup()" [color=lightblue];
"AbelianGroup()" -> "CABMON..."
"AbelianGroup()" -> "REPDB..."

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SETCAT..."
"SemiGroup()" -> "REPSQ..."

"REPDB..." [color="#00EE00"];
"REPSQ..." [color="#00EE00"];
"SETCAT..." [color=lightblue];
"CABMON..." [color=lightblue];
}
```

———-

12.0.166 OctonionCategory (OC)



— OctonionCategory.input —

```
)set break resume
)sys rm -f OctonionCategory.output
```

```
)spool OctonionCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show OctonionCategory
--R OctonionCategory R: CommutativeRing  is a category constructor
--R Abbreviation for OctonionCategory is OC
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for OC
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (R,%) -> %                      ?*? : (%,R) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        abs : % -> R if R has RNS
--R coerce : R -> %                       coerce : Integer -> %
--R coerce : % -> OutputForm              conjugate : % -> %
--R hash : % -> SingleInteger             imagE : % -> R
--R imagI : % -> R                        imagJ : % -> R
--R imagK : % -> R                        imagi : % -> R
--R imagj : % -> R                        imagk : % -> R
--R inv : % -> % if R has FIELD           latex : % -> String
--R map : ((R -> R),%) -> %               norm : % -> R
--R octon : (R,R,R,R,R,R,R,R) -> %        one? : % -> Boolean
--R real : % -> R                         recip : % -> Union(%,"failed")
--R retract : % -> R                      sample : () -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?<? : (%,%) -> Boolean if R has ORDSET
--R ?<=? : (%,%) -> Boolean if R has ORDSET
--R ?>? : (%,%) -> Boolean if R has ORDSET
--R ?>=? : (%,%) -> Boolean if R has ORDSET
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if R has CHARNZ
--R coerce : Fraction Integer -> % if R has RETRACT FRAC INT
--R convert : % -> InputForm if R has KONVERT INFORM
--R ?.? : (%,R) -> % if R has ELTAB(R,R)
--R eval : (%,Symbol,R) -> % if R has IEVALAB(SYMBOL,R)
--R eval : (%,List Symbol,List R) -> % if R has IEVALAB(SYMBOL,R)
--R eval : (%,List Equation R) -> % if R has EVALAB R
--R eval : (%,Equation R) -> % if R has EVALAB R
--R eval : (%,R,R) -> % if R has EVALAB R
--R eval : (%,List R,List R) -> % if R has EVALAB R
```

```
--R index : PositiveInteger -> % if R has FINITE
--R lookup : % -> PositiveInteger if R has FINITE
--R max : (%,%) -> % if R has ORDSET
--R min : (%,%) -> % if R has ORDSET
--R random : () -> % if R has FINITE
--R rational : % -> Fraction Integer if R has INS
--R rational? : % -> Boolean if R has INS
--R rationalIfCan : % -> Union(Fraction Integer,"failed") if R has INS
--R retract : % -> Fraction Integer if R has RETRACT FRAC INT
--R retract : % -> Integer if R has RETRACT INT
--R retractIfCan : % -> Union(R,"failed")
--R retractIfCan : % -> Union(Fraction Integer,"failed") if R has RETRACT FRAC INT
--R retractIfCan : % -> Union(Integer,"failed") if R has RETRACT INT
--R size : () -> NonNegativeInteger if R has FINITE
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— OctonionCategory.help —

```
=====================================================================
OctonionCategory examples
=====================================================================

See Also:
o )show OctonionCategory
```

———-

See:

⇐ “Algebra” (ALGEBRA) 11.0.152 on page 947
⇐ “FullyEvalableOver” (FEVALAB) 4.0.44 on page 159
⇐ “FullyRetractableTo” (FRETRCT) 3.0.27 on page 87

Exports:

0	1	abs	characteristic	charthRoot
coerce	conjugate	convert	eval	hash
imagE	imagI	imagJ	imagK	imagi
imagj	imagk	index	inv	latex
lookup	map	max	min	norm
octon	one?	random	rational	rational?
rationalIfCan	real	recip	retract	retractIfCan
sample	size	subtractIfCan	zero?	?*?
?**?	?+?	?-?	-?	?=?
?^?	?~=?	?<?	?<=?	?>?
?>=?	?.?			

Attributes Exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
imagi : % -> R
imagj : % -> R
imagk : % -> R
imagE : % -> R
imagI : % -> R
imagJ : % -> R
imagK : % -> R
octon : (R,R,R,R,R,R,R,R) -> %
real : % -> R
```

These are implemented by this category:

```
abs : % -> R if R has RNS
characteristic : () -> NonNegativeInteger
coerce : R -> %
coerce : Integer -> %
coerce : % -> OutputForm
conjugate : % -> %
convert : % -> InputForm if R has KONVERT INFORM
inv : % -> % if R has FIELD
map : ((R -> R),%) -> %
norm : % -> R
rational : % -> Fraction Integer if R has INS
rational? : % -> Boolean if R has INS
rationalIfCan : % -> Union(Fraction Integer,"failed") if R has INS
retract : % -> R
```

```
retractIfCan : % -> Union(R,"failed")
zero? : % -> Boolean
?<? : (%,%) -> Boolean if R has ORDSET
?=? : (%,%) -> Boolean
?+? : (%,%) -> %
-? : % -> %
?*? : (R,%) -> %
?*? : (Integer,%) -> %
```

These exports come from (p947) Algebra(R:CommutativeRing):

```
0 : () -> %
1 : () -> %
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (%,%) -> %
?-? : (%,%) -> %
?**? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?*? : (%,R) -> %
```

These exports come from (p87) FullyRetractableTo(R:CommutativeRing):

```
coerce : Fraction Integer -> % if R has RETRACT FRAC INT
retract : % -> Fraction Integer if R has RETRACT FRAC INT
retract : % -> Integer if R has RETRACT INT
retractIfCan : % -> Union(Fraction Integer,"failed") if R has RETRACT FRAC INT
retractIfCan : % -> Union(Integer,"failed") if R has RETRACT INT
```

These exports come from (p159) FullyEvalableOver(R:CommutativeRing):

```
eval : (%,Equation R) -> % if R has EVALAB R
eval : (%,List Symbol,List R) -> % if R has IEVALAB(SYMBOL,R)
eval : (%,List Equation R) -> % if R has EVALAB R
eval : (%,R,R) -> % if R has EVALAB R
eval : (%,List R,List R) -> % if R has EVALAB R
eval : (%,Symbol,R) -> % if R has IEVALAB(SYMBOL,R)
?.? : (%,R) -> % if R has ELTAB(R,R)
```

These exports come from (p168) Finite():

```
index : PositiveInteger -> % if R has FINITE
lookup : % -> PositiveInteger if R has FINITE
random : () -> % if R has FINITE
size : () -> NonNegativeInteger if R has FINITE
```

These exports come from (p216) OrderedSet():

```
max : (%,%) -> % if R has ORDSET
min : (%,%) -> % if R has ORDSET
?<=? : (%,%) -> Boolean if R has ORDSET
?>? : (%,%) -> Boolean if R has ORDSET
?>=? : (%,%) -> Boolean if R has ORDSET
```

These exports come from (p833) CharacteristicNonZero():

```
charthRoot : % -> Union(%,"failed") if R has CHARNZ
```

— category OC OctonionCategory —

```
)abbrev category OC OctonionCategory
++ Author: R. Wisbauer, J. Grabmeier
++ Date Created: 05 September 1990
++ Date Last Updated: 19 September 1990
++ Basic Operations: _+, _*, octon, real, imagi, imagj, imagk,
++  imagE, imagI, imagJ, imagK
++ Related Constructors: QuaternionCategory
++ Also See:
++ AMS Classifications:
++ Keywords: octonion, non-associative algebra, Cayley-Dixon
++ References: e.g. I.L Kantor, A.S. Solodovnikov:
++  Hypercomplex Numbers, Springer Verlag Heidelberg, 1989,
++  ISBN 0-387-96980-2
++ Description:
++ OctonionCategory gives the categorial frame for the
++ octonions, and eight-dimensional non-associative algebra,
++ doubling the the quaternions in the same way as doubling
++ the Complex numbers to get the quaternions.

OctonionCategory(R: CommutativeRing): Category ==
  -- we are cheating a little bit, algebras in \Language{}
  -- are mainly considered to be associative, but that's not
  -- an attribute and we can't guarantee that there is no piece
  -- of code which implicitly
  -- uses this. In a later version we shall properly combine
  -- all this code in the context of general, non-associative
  -- algebras, which are meanwhile implemented in \Language{}
  Join(Algebra R, FullyRetractableTo R, FullyEvalableOver R) with
     conjugate: % -> %
```

```
      ++ conjugate(o) negates the imaginary parts i,j,k,E,I,J,K of octonian o.
    real:    % -> R
      ++ real(o) extracts real part of octonion o.
    imagi:   % -> R
      ++ imagi(o) extracts the i part of octonion o.
    imagj:   % -> R
      ++ imagj(o) extracts the j part of octonion o.
    imagk:   % -> R
      ++ imagk(o) extracts the k part of octonion o.
    imagE:   % -> R
      ++ imagE(o) extracts the imaginary E part of octonion o.
    imagI:   % -> R
      ++ imagI(o) extracts the imaginary I part of octonion o.
    imagJ:   % -> R
      ++ imagJ(o) extracts the imaginary J part of octonion o.
    imagK:   % -> R
      ++ imagK(o) extracts the imaginary K part of octonion o.
    norm:    % -> R
      ++ norm(o) returns the norm of an octonion, equal to
      ++ the sum of the squares
      ++ of its coefficients.
    octon: (R,R,R,R,R,R,R,R) -> %
      ++ octon(re,ri,rj,rk,rE,rI,rJ,rK) constructs an octonion
      ++ from scalars.
    if R has Finite then Finite
    if R has OrderedSet then OrderedSet
    if R has ConvertibleTo InputForm then ConvertibleTo InputForm
    if R has CharacteristicZero then CharacteristicZero
    if R has CharacteristicNonZero then CharacteristicNonZero
    if R has RealNumberSystem then
      abs:   % -> R
        ++ abs(o) computes the absolute value of an octonion, equal to
        ++ the square root of the \spadfunFrom{norm}{Octonion}.
    if R has IntegerNumberSystem then
      rational?    : % -> Boolean
        ++ rational?(o) tests if o is rational, i.e. that all seven
        ++ imaginary parts are 0.
      rational     : % -> Fraction Integer
        ++ rational(o) returns the real part if all seven
        ++ imaginary parts are 0.
        ++ Error: if o is not rational.
      rationalIfCan: % -> Union(Fraction Integer, "failed")
        ++ rationalIfCan(o) returns the real part if
        ++ all seven imaginary parts are 0, and "failed" otherwise.
    if R has Field then
      inv : % -> %
        ++ inv(o) returns the inverse of o if it exists.
add
    characteristic() ==
      characteristic()$R
```

```
conjugate x ==
  octon(real x, - imagi x, - imagj x, - imagk x, - imagE x,_
  - imagI x, - imagJ x, - imagK x)

map(fn, x)       ==
  octon(fn real x,fn imagi x,fn imagj x,fn imagk x, fn imagE x,_
  fn imagI x, fn imagJ x,fn imagK x)

norm x ==
  real x * real x + imagi x * imagi x + _
  imagj x * imagj x + imagk x * imagk x + _
  imagE x * imagE x + imagI x * imagI x + _
  imagJ x * imagJ x + imagK x * imagK x

x = y            ==
  (real x = real y) and (imagi x = imagi y) and _
  (imagj x = imagj y) and (imagk x = imagk y) and _
  (imagE x = imagE y) and (imagI x = imagI y) and _
  (imagJ x = imagJ y) and (imagK x = imagK y)

x + y            ==
  octon(real x + real y, imagi x + imagi y,_
  imagj x + imagj y, imagk x + imagk y,_
  imagE x + imagE y, imagI x + imagI y,_
  imagJ x + imagJ y, imagK x + imagK y)

- x             ==
  octon(- real x, - imagi x, - imagj x, - imagk x,_
  - imagE x, - imagI x, - imagJ x, - imagK x)

r:R * x:%        ==
  octon(r * real x, r * imagi x, r * imagj x, r * imagk x,_
  r * imagE x, r * imagI x, r * imagJ x, r * imagK x)

n:Integer * x:%  ==
  octon(n * real x, n * imagi x, n * imagj x, n * imagk x,_
  n * imagE x, n * imagI x, n * imagJ x, n * imagK x)

coerce(r:R)      ==
  octon(r,0$R,0$R,0$R,0$R,0$R,0$R,0$R)

coerce(n:Integer)      ==
  octon(n :: R,0$R,0$R,0$R,0$R,0$R,0$R,0$R)

zero? x ==
  zero? real x and zero? imagi x and _
  zero? imagj x and zero? imagk x and _
  zero? imagE x and zero? imagI x and _
  zero? imagJ x and zero? imagK x
```

```
    retract(x):R ==
      not (zero? imagi x and zero? imagj x and zero? imagk x and _
      zero? imagE x and zero? imagI x and zero? imagJ x and zero? imagK x)=>
        error "Cannot retract octonion."
      real x

    retractIfCan(x):Union(R,"failed") ==
      not (zero? imagi x and zero? imagj x and zero? imagk x and _
      zero? imagE x and zero? imagI x and zero? imagJ x and zero? imagK x)=>
        "failed"
      real x

    coerce(x:%):OutputForm ==
        part,z : OutputForm
        y : %
        zero? x => (0$R) :: OutputForm
        not zero?(real x) =>
          y := octon(0$R,imagi(x),imagj(x),imagk(x),imagE(x),
            imagI(x),imagJ(x),imagK(x))
          zero? y => real(x) :: OutputForm
          (real(x) :: OutputForm) + (y :: OutputForm)
        -- we know that the real part is 0
        not zero?(imagi(x)) =>
          y := octon(0$R,0$R,imagj(x),imagk(x),imagE(x),
            imagI(x),imagJ(x),imagK(x))
          z :=
            part := "i"::Symbol::OutputForm
--            one? imagi(x) => part
            (imagi(x) = 1) => part
            (imagi(x) :: OutputForm) * part
          zero? y => z
          z + (y :: OutputForm)
        -- we know that the real part and i part are 0
        not zero?(imagj(x)) =>
          y := octon(0$R,0$R,0$R,imagk(x),imagE(x),
            imagI(x),imagJ(x),imagK(x))
          z :=
            part := "j"::Symbol::OutputForm
--            one? imagj(x) => part
            (imagj(x) = 1) => part
            (imagj(x) :: OutputForm) * part
          zero? y => z
          z + (y :: OutputForm)
        -- we know that the real part and i and j parts are 0
        not zero?(imagk(x)) =>
          y := octon(0$R,0$R,0$R,0$R,imagE(x),
            imagI(x),imagJ(x),imagK(x))
          z :=
            part := "k"::Symbol::OutputForm
```

```
--            one? imagk(x) => part
            (imagk(x) = 1) => part
            (imagk(x) :: OutputForm) * part
          zero? y => z
          z + (y :: OutputForm)
        -- we know that the real part,i,j,k parts are 0
        not zero?(imagE(x)) =>
          y := octon(0$R,0$R,0$R,0$R,0$R,
            imagI(x),imagJ(x),imagK(x))
          z :=
            part := "E"::Symbol::OutputForm
--            one? imagE(x) => part
            (imagE(x) = 1) => part
            (imagE(x) :: OutputForm) * part
          zero? y => z
          z + (y :: OutputForm)
        -- we know that the real part,i,j,k,E parts are 0
        not zero?(imagI(x)) =>
          y := octon(0$R,0$R,0$R,0$R,0$R,0$R,imagJ(x),imagK(x))
          z :=
            part := "I"::Symbol::OutputForm
--            one? imagI(x) => part
            (imagI(x) = 1) => part
            (imagI(x) :: OutputForm) * part
          zero? y => z
          z + (y :: OutputForm)
        -- we know that the real part,i,j,k,E,I parts are 0
        not zero?(imagJ(x)) =>
          y := octon(0$R,0$R,0$R,0$R,0$R,0$R,0$R,imagK(x))
          z :=
            part := "J"::Symbol::OutputForm
--            one? imagJ(x) => part
            (imagJ(x) = 1) => part
            (imagJ(x) :: OutputForm) * part
          zero? y => z
          z + (y :: OutputForm)
        -- we know that the real part,i,j,k,E,I,J parts are 0
        part := "K"::Symbol::OutputForm
--        one? imagK(x) => part
        (imagK(x) = 1) => part
        (imagK(x) :: OutputForm) * part

    if R has Field then
      inv x ==
        (norm x) = 0 => error "This octonion is not invertible."
        (inv norm x) * conjugate x

    if R has ConvertibleTo InputForm then
      convert(x:%):InputForm ==
        l : List InputForm := [convert("octon" :: Symbol),
```

```
        convert(real x)$R, convert(imagi x)$R, convert(imagj x)$R,_
          convert(imagk x)$R, convert(imagE x)$R,_
          convert(imagI x)$R, convert(imagJ x)$R,_
          convert(imagK x)$R]
      convert(l)$InputForm

    if R has OrderedSet then
      x < y ==
        real x = real y =>
         imagi x = imagi y =>
          imagj x = imagj y =>
           imagk x = imagk y =>
            imagE x = imagE y =>
             imagI x = imagI y =>
              imagJ x = imagJ y =>
               imagK x < imagK y
              imagJ x < imagJ y
             imagI x < imagI y
            imagE x < imagE y
           imagk x < imagk y
          imagj x < imagj y
         imagi x < imagi y
        real x < real y

    if R has RealNumberSystem then
      abs x == sqrt norm x

    if R has IntegerNumberSystem then
      rational? x ==
        (zero? imagi x) and (zero? imagj x) and (zero? imagk x) and _
        (zero? imagE x) and (zero? imagI x) and (zero? imagJ x) and _
        (zero? imagK x)

      rational  x ==
        rational? x => rational real x
        error "Not a rational number"

      rationalIfCan x ==
        rational? x => rational real x
        "failed"
```

———-

— OC.dotabb —

```
"OC"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OC"];
"OC" -> "ALGEBRA"
```

```
"OC" -> "FEVALAB"
"OC" -> "FRETRCT"
```

———-

— OC.dotfull —

```
"OctonionCategory(a:CommutativeRing)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OC"];
"OctonionCategory(a:CommutativeRing)" -> "Algebra(a:CommutativeRing)"
"OctonionCategory(a:CommutativeRing)" -> "FullyEvalableOver(CommutativeRing)"
"OctonionCategory(a:CommutativeRing)" ->
   "FullyRetractableTo(a:CommutativeRing)"
```

———-

— OC.dotpic —

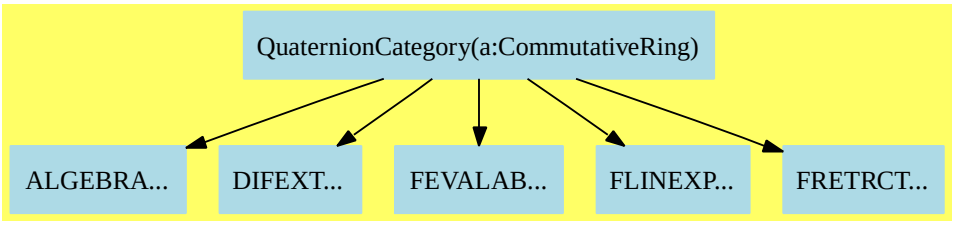
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"OctonionCategory(a:CommutativeRing)" [color=lightblue];
"OctonionCategory(a:CommutativeRing)" -> "ALGEBRA..."
"OctonionCategory(a:CommutativeRing)" -> "FEVALAB..."
"OctonionCategory(a:CommutativeRing)" -> "FRETRCT..."

"ALGEBRA..." [color=lightblue];
"FEVALAB..." [color=lightblue];
"FRETRCT..." [color=lightblue];
}
```

———-

12.0.167 QuaternionCategory (QUATCAT)



— QuaternionCategory.input —

```
)set break resume
)sys rm -f QuaternionCategory.output
)spool QuaternionCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show QuaternionCategory
--R QuaternionCategory R: CommutativeRing  is a category constructor
--R Abbreviation for QuaternionCategory is QUATCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for QUATCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (R,%) -> %                      ?*? : (%,R) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R D : (%,(R -> R)) -> %                 D : % -> % if R has DIFRING
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        abs : % -> R if R has RNS
--R coerce : R -> %                       coerce : Integer -> %
--R coerce : % -> OutputForm              conjugate : % -> %
--R hash : % -> SingleInteger             imagI : % -> R
--R imagJ : % -> R                        imagK : % -> R
--R inv : % -> % if R has FIELD           latex : % -> String
--R map : ((R -> R),%) -> %               norm : % -> R
--R one? : % -> Boolean                   quatern : (R,R,R,R) -> %
--R real : % -> R                        recip : % -> Union(%,"failed")
--R retract : % -> R                     sample : () -> %
--R zero? : % -> Boolean                 ?~=? : (%,%) -> Boolean
--R ?*? : (Fraction Integer,%) -> % if R has FIELD
--R ?*? : (%,Fraction Integer) -> % if R has FIELD
```

```
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,Integer) -> % if R has FIELD
--R ?**? : (%,NonNegativeInteger) -> %
--R ?<? : (%,%) -> Boolean if R has ORDSET
--R ?<=? : (%,%) -> Boolean if R has ORDSET
--R ?>? : (%,%) -> Boolean if R has ORDSET
--R ?>=? : (%,%) -> Boolean if R has ORDSET
--R D : (%,(R -> R),NonNegativeInteger) -> %
--R D : (%,List Symbol,List NonNegativeInteger) -> % if R has PDRING SYMBOL
--R D : (%,Symbol,NonNegativeInteger) -> % if R has PDRING SYMBOL
--R D : (%,List Symbol) -> % if R has PDRING SYMBOL
--R D : (%,Symbol) -> % if R has PDRING SYMBOL
--R D : (%,NonNegativeInteger) -> % if R has DIFRING
--R ?^? : (%,Integer) -> % if R has FIELD
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if R has CHARNZ
--R coerce : Fraction Integer -> % if R has FIELD or R has RETRACT FRAC INT
--R convert : % -> InputForm if R has KONVERT INFORM
--R differentiate : (%,(R -> R)) -> %
--R differentiate : (%,(R -> R),NonNegativeInteger) -> %
--R differentiate : (%,List Symbol,List NonNegativeInteger) -> % if R has PDRING SYMBOL
--R differentiate : (%,Symbol,NonNegativeInteger) -> % if R has PDRING SYMBOL
--R differentiate : (%,List Symbol) -> % if R has PDRING SYMBOL
--R differentiate : (%,Symbol) -> % if R has PDRING SYMBOL
--R differentiate : (%,NonNegativeInteger) -> % if R has DIFRING
--R differentiate : % -> % if R has DIFRING
--R ?.? : (%,R) -> % if R has ELTAB(R,R)
--R eval : (%,Symbol,R) -> % if R has IEVALAB(SYMBOL,R)
--R eval : (%,List Symbol,List R) -> % if R has IEVALAB(SYMBOL,R)
--R eval : (%,List Equation R) -> % if R has EVALAB R
--R eval : (%,Equation R) -> % if R has EVALAB R
--R eval : (%,R,R) -> % if R has EVALAB R
--R eval : (%,List R,List R) -> % if R has EVALAB R
--R max : (%,%) -> % if R has ORDSET
--R min : (%,%) -> % if R has ORDSET
--R rational : % -> Fraction Integer if R has INS
--R rational? : % -> Boolean if R has INS
--R rationalIfCan : % -> Union(Fraction Integer,"failed") if R has INS
--R reducedSystem : Matrix % -> Matrix R
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix R,vec: Vector R)
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer) if R
--R reducedSystem : Matrix % -> Matrix Integer if R has LINEXP INT
--R retract : % -> Fraction Integer if R has RETRACT FRAC INT
--R retract : % -> Integer if R has RETRACT INT
--R retractIfCan : % -> Union(R,"failed")
--R retractIfCan : % -> Union(Fraction Integer,"failed") if R has RETRACT FRAC INT
--R retractIfCan : % -> Union(Integer,"failed") if R has RETRACT INT
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
```

```
--E 1

)spool
)lisp (bye)
```

———-

— **QuaternionCategory.help** —

```
=====================================================================
QuaternionCategory examples
=====================================================================

See Also:
o )show QuaternionCategory
```

———-

See:

⇐ “Algebra” (ALGEBRA) 11.0.152 on page 947
⇐ “DifferentialExtension” (DIFEXT) 11.0.153 on page 953
⇐ “FullyEvalableOver” (FEVALAB) 4.0.44 on page 159
⇐ “FullyLinearlyExplicitRingOver” (FLINEXP) 11.0.154 on page 960
⇐ “FullyRetractableTo” (FRETRCT) 3.0.27 on page 87

Exports:

0	1	abs	characteristic
charthRoot	coerce	conjugate	convert
D	differentiate	eval	hash
imagI	imagJ	imagK	inv
latex	map	max	min
norm	one?	quatern	rational
rational?	rationalIfCan	real	recip
reducedSystem	retract	retractIfCan	sample
subtractIfCan	zero?	?*?	?**?
?+?	?-?	-?	?=?
?^?	?~=?	?<?	?<=?
?>?	?>=?	?.?	

Attributes Exported:

- if #1 has EntireRing then noZeroDivisors where **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return “failed” if its argument is not a unit.

- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
imagI : % -> R
imagJ : % -> R
imagK : % -> R
quatern : (R,R,R,R) -> %
real : % -> R
```

These are implemented by this category:

```
abs : % -> R if R has RNS
characteristic : () -> NonNegativeInteger
coerce : R -> %
coerce : Integer -> %
coerce : % -> OutputForm
conjugate : % -> %
convert : % -> InputForm if R has KONVERT INFORM
differentiate : (%,(R -> R)) -> %
inv : % -> % if R has FIELD
map : ((R -> R),%) -> %
norm : % -> R
one? : % -> Boolean
rational : % -> Fraction Integer if R has INS
rational? : % -> Boolean if R has INS
rationalIfCan : % -> Union(Fraction Integer,"failed") if R has INS
retract : % -> R
retractIfCan : % -> Union(R,"failed")
zero? : % -> Boolean
?=? : (%,%) -> Boolean
?+? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
?*? : (R,%) -> %
?*? : (Integer,%) -> %
?<? : (%,%) -> Boolean if R has ORDSET
```

These exports come from (p947) Algebra(R:CommutativeRing):

```
0 : () -> %
1 : () -> %
hash : % -> SingleInteger
latex : % -> String
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
?~=? : (%,%) -> Boolean
```

```
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?*? : (%,R) -> %
```

These exports come from (p87) FullyRetractableTo(R:CommutativeRing):

```
coerce : Fraction Integer -> % if R has FIELD or R has RETRACT FRAC INT
retract : % -> Fraction Integer if R has RETRACT FRAC INT
retract : % -> Integer if R has RETRACT INT
retractIfCan : % -> Union(Fraction Integer,"failed")
  if R has RETRACT FRAC INT
retractIfCan : % -> Union(Integer,"failed") if R has RETRACT INT
```

These exports come from (p953) DifferentialExtension(R:CommutativeRing):

```
D : % -> % if R has DIFRING
D : (%,NonNegativeInteger) -> % if R has DIFRING
D : (%,(R -> R)) -> %
D : (%,(R -> R),NonNegativeInteger) -> %
D : (%,Symbol) -> % if R has PDRING SYMBOL
D : (%,List Symbol) -> % if R has PDRING SYMBOL
D : (%,Symbol,NonNegativeInteger) -> % if R has PDRING SYMBOL
D : (%,List Symbol,List NonNegativeInteger) -> % if R has PDRING SYMBOL
differentiate : (%,List Symbol,List NonNegativeInteger) -> %
  if R has PDRING SYMBOL
differentiate : (%,Symbol,NonNegativeInteger) -> % if R has PDRING SYMBOL
differentiate : (%,List Symbol) -> % if R has PDRING SYMBOL
differentiate : (%,NonNegativeInteger) -> % if R has DIFRING
differentiate : % -> % if R has DIFRING
differentiate : (%,(R -> R),NonNegativeInteger) -> %
differentiate : (%,Symbol) -> % if R has PDRING SYMBOL
?*? : (%,%) -> %
```

These exports come from (p159) FullyEvalableOver(R:CommutativeRing):

```
eval : (%,Equation R) -> % if R has EVALAB R
eval : (%,List Symbol,List R) -> % if R has IEVALAB(SYMBOL,R)
eval : (%,List Equation R) -> % if R has EVALAB R
eval : (%,R,R) -> % if R has EVALAB R
eval : (%,List R,List R) -> % if R has EVALAB R
eval : (%,Symbol,R) -> % if R has IEVALAB(SYMBOL,R)
?.? : (%,R) -> % if R has ELTAB(R,R)
```

These exports come from (p960) FullyLinearlyExplicitRingOver(R)
where R:CommutativeRing:

```
recip : % -> Union(%,"failed")
reducedSystem : Matrix % -> Matrix Integer if R has LINEXP INT
reducedSystem :
  (Matrix %,Vector %) ->
     Record(mat: Matrix Integer,vec: Vector Integer)
       if R has LINEXP INT
reducedSystem : Matrix % -> Matrix R
reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix R,vec: Vector R)
```

These exports come from (p216) OrderedSet():

```
max : (%,%) -> % if R has ORDSET
min : (%,%) -> % if R has ORDSET
?<=? : (%,%) -> Boolean if R has ORDSET
?>? : (%,%) -> Boolean if R has ORDSET
?>=? : (%,%) -> Boolean if R has ORDSET
```

These exports come from (p1013) DivisionRing():

```
?**? : (%,Integer) -> % if R has FIELD
?^? : (%,Integer) -> % if R has FIELD
?*? : (Fraction Integer,%) -> % if R has FIELD
?*? : (%,Fraction Integer) -> % if R has FIELD
```

These exports come from (p833) CharacteristicNonZero():

```
charthRoot : % -> Union(%,"failed") if R has CHARNZ
```

— category QUATCAT QuaternionCategory —

```
)abbrev category QUATCAT QuaternionCategory
++ Author: Robert S. Sutor
++ Date Created: 23 May 1990
++ Change History:
++   10 September 1990
++ Basic Operations: (Algebra)
++   abs, conjugate, imagI, imagJ, imagK, norm, quatern, rational,
++   rational?, real
++ Related Constructors: Quaternion, QuaternionCategoryFunctions2
++ Also See: DivisionRing
++ AMS Classifications: 11R52
++ Keywords: quaternions, division ring, algebra
++ Description:
++ \spadtype{QuaternionCategory} describes the category of quaternions
++ and implements functions that are not representation specific.

QuaternionCategory(R: CommutativeRing): Category ==
  Join(Algebra R, FullyRetractableTo R, DifferentialExtension R,
```

```
  FullyEvalableOver R, FullyLinearlyExplicitRingOver R) with

    conjugate: $ -> $
      ++ conjugate(q) negates the imaginary parts of quaternion \spad{q}.
    imagI:   $ -> R
      ++ imagI(q) extracts the imaginary i part of quaternion \spad{q}.
    imagJ:   $ -> R
      ++ imagJ(q) extracts the imaginary j part of quaternion \spad{q}.
    imagK:   $ -> R
      ++ imagK(q) extracts the imaginary k part of quaternion \spad{q}.
    norm:    $ -> R
      ++ norm(q) computes the norm of \spad{q} (the sum of the
      ++ squares of the components).
    quatern: (R,R,R,R) -> $
      ++ quatern(r,i,j,k) constructs a quaternion from scalars.
    real:    $ -> R
      ++ real(q) extracts the real part of quaternion \spad{q}.

    if R has EntireRing then EntireRing
    if R has OrderedSet then OrderedSet
    if R has Field then DivisionRing
    if R has ConvertibleTo InputForm then ConvertibleTo InputForm
    if R has CharacteristicZero then CharacteristicZero
    if R has CharacteristicNonZero then CharacteristicNonZero
    if R has RealNumberSystem then
      abs    : $ -> R
        ++ abs(q) computes the absolute value of quaternion \spad{q}
        ++ (sqrt of norm).
    if R has IntegerNumberSystem then
      rational?    : $ -> Boolean
        ++ rational?(q) returns {\it true} if all the imaginary
        ++ parts of \spad{q} are zero and the real part can be
        ++ converted into a rational number, and {\it false}
        ++ otherwise.
      rational     : $ -> Fraction Integer
        ++ rational(q) tries to convert \spad{q} into a
        ++ rational number. Error: if this is not
        ++ possible. If \spad{rational?(q)} is true, the
        ++ conversion will be done and the rational number returned.
      rationalIfCan: $ -> Union(Fraction Integer, "failed")
        ++ rationalIfCan(q) returns \spad{q} as a rational number,
        ++ or "failed" if this is not possible.
        ++ Note that if \spad{rational?(q)} is true, the conversion
        ++ can be done and the rational number will be returned.

add

      characteristic() ==
        characteristic()$R
```

```
      conjugate x      ==
        quatern(real x, - imagI x, - imagJ x, - imagK x)

      map(fn, x)       ==
        quatern(fn real x, fn imagI x, fn imagJ x, fn imagK x)

      norm x ==
        real x * real x + imagI x * imagI x +
          imagJ x * imagJ x + imagK x * imagK x

      x = y            ==
        (real x = real y) and (imagI x = imagI y) and
          (imagJ x = imagJ y) and (imagK x = imagK y)

      x + y            ==
        quatern(real x + real y, imagI x + imagI y,
          imagJ x + imagJ y, imagK x + imagK y)

      x - y            ==
        quatern(real x - real y, imagI x - imagI y,
          imagJ x - imagJ y, imagK x - imagK y)

      - x              ==
        quatern(- real x, - imagI x, - imagJ x, - imagK x)

      r:R * x:$        ==
        quatern(r * real x, r * imagI x, r * imagJ x, r * imagK x)

      n:Integer * x:$  ==
        quatern(n * real x, n * imagI x, n * imagJ x, n * imagK x)

      differentiate(x:$, d:R -> R) ==
        quatern(d real x, d imagI x, d imagJ x, d imagK x)

      coerce(r:R)      ==
        quatern(r,0$R,0$R,0$R)

      coerce(n:Integer)      ==
        quatern(n :: R,0$R,0$R,0$R)

      one? x ==
--        one? real x and zero? imagI x and
        (real x) = 1 and zero? imagI x and
          zero? imagJ x and zero? imagK x

      zero? x ==
        zero? real x and zero? imagI x and
          zero? imagJ x and zero? imagK x

      retract(x):R ==
```

```
      not (zero? imagI x and zero? imagJ x and zero? imagK x) =>
        error "Cannot retract quaternion."
      real x

    retractIfCan(x):Union(R,"failed") ==
      not (zero? imagI x and zero? imagJ x and zero? imagK x) =>
        "failed"
      real x

    coerce(x:$):OutputForm ==
      part,z : OutputForm
      y : $
      zero? x => (0$R) :: OutputForm
      not zero?(real x) =>
        y := quatern(0$R,imagI(x),imagJ(x),imagK(x))
        zero? y => real(x) :: OutputForm
        (real(x) :: OutputForm) + (y :: OutputForm)
      -- we know that the real part is 0
      not zero?(imagI(x)) =>
        y := quatern(0$R,0$R,imagJ(x),imagK(x))
        z :=
          part := "i"::Symbol::OutputForm
--          one? imagI(x) => part
          (imagI(x) = 1) => part
          (imagI(x) :: OutputForm) * part
        zero? y => z
        z + (y :: OutputForm)
      -- we know that the real part and i part are 0
      not zero?(imagJ(x)) =>
        y := quatern(0$R,0$R,0$R,imagK(x))
        z :=
          part := "j"::Symbol::OutputForm
--          one? imagJ(x) => part
          (imagJ(x) = 1) => part
          (imagJ(x) :: OutputForm) * part
        zero? y => z
        z + (y :: OutputForm)
      -- we know that the real part and i and j parts are 0
      part := "k"::Symbol::OutputForm
--        one? imagK(x) => part
      (imagK(x) = 1) => part
      (imagK(x) :: OutputForm) * part

    if R has Field then
      inv x ==
        norm x = 0 => error "This quaternion is not invertible."
        (inv norm x) * conjugate x

    if R has ConvertibleTo InputForm then
      convert(x:$):InputForm ==
```

```
          l : List InputForm := [convert("quatern" :: Symbol),
            convert(real x)$R, convert(imagI x)$R, convert(imagJ x)$R,
              convert(imagK x)$R]
          convert(l)$InputForm

      if R has OrderedSet then
        x < y ==
          real x = real y =>
            imagI x = imagI y =>
              imagJ x = imagJ y =>
                imagK x < imagK y
              imagJ x < imagJ y
            imagI x < imagI y
          real x < real y

      if R has RealNumberSystem then
        abs x == sqrt norm x

      if R has IntegerNumberSystem then
        rational? x ==
          (zero? imagI x) and (zero? imagJ x) and (zero? imagK x)

        rational  x ==
          rational? x => rational real x
          error "Not a rational number"

        rationalIfCan x ==
          rational? x => rational real x
          "failed"
```

———-

— QUATCAT.dotabb —

```
"QUATCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=QUATCAT"];
"QUATCAT" -> "ALGEBRA"
"QUATCAT" -> "DIFEXT"
"QUATCAT" -> "FEVALAB"
"QUATCAT" -> "FLINEXP"
"QUATCAT" -> "FRETRCT"
```

———-

— QUATCAT.dotfull —

```
"QuaternionCategory(a:CommutativeRing)"
```

```
 [color=lightblue,href="bookvol10.2.pdf#nameddest=QUATCAT"];
"QuaternionCategory(a:CommutativeRing)" ->
   "Algebra(a:CommutativeRing)"
"QuaternionCategory(a:CommutativeRing)" ->
   "DifferentialExtension(CommutativeRing)"
"QuaternionCategory(a:CommutativeRing)" ->
   "FullyEvalableOver(CommutativeRing)"
"QuaternionCategory(a:CommutativeRing)" ->
   "FullyLinearlyExplicitRingOver(a:CommutativeRing)"
"QuaternionCategory(a:CommutativeRing)" ->
   "FullyRetractableTo(a:CommutativeRing)"
```

———-

— QUATCAT.dotpic —

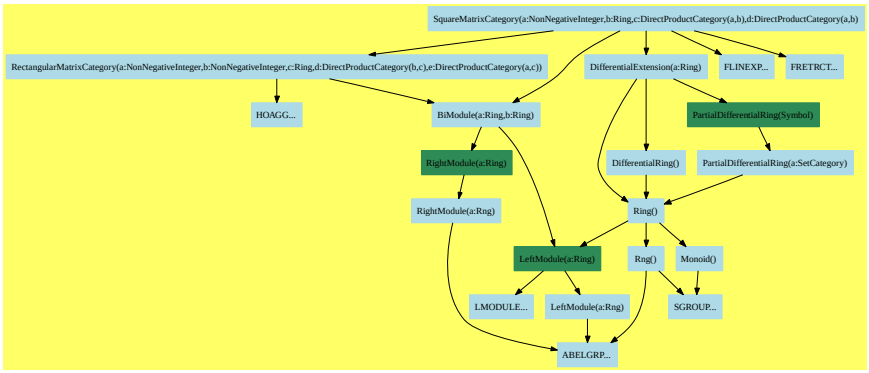
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"QuaternionCategory(a:CommutativeRing)" [color=lightblue];
"QuaternionCategory(a:CommutativeRing)" -> "ALGEBRA..."
"QuaternionCategory(a:CommutativeRing)" -> "DIFEXT..."
"QuaternionCategory(a:CommutativeRing)" -> "FEVALAB..."
"QuaternionCategory(a:CommutativeRing)" -> "FLINEXP..."
"QuaternionCategory(a:CommutativeRing)" -> "FRETRCT..."

"ALGEBRA..." [color=lightblue];
"DIFEXT..." [color=lightblue];
"FEVALAB..." [color=lightblue];
"FLINEXP..." [color=lightblue];
"FRETRCT..." [color=lightblue];
}
```

———-

12.0.168 SquareMatrixCategory (SMATCAT)



We define three categories for matrices

- MatrixCategory is the category of all matrices
- RectangularMatrixCategory is the category of all matrices of a given dimension
- SquareMatrixCategory inherits from RectangularMatrixCategory

The SquareMatrix domain is for square matrices of fixed dimension.

— SquareMatrixCategory.input —

```
)set break resume
)sys rm -f SquareMatrixCategory.output
)spool SquareMatrixCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show SquareMatrixCategory
--R SquareMatrixCategory(ndim: NonNegativeInteger,R: Ring,Row: DirectProductCategory(t#1,t#2),Co
--R Abbreviation for SquareMatrixCategory is SMATCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for SMATCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (Row,%) -> Row                  ?*? : (%,Col) -> Col
--R ?*? : (R,%) -> %                      ?*? : (%,R) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
```

```
--R D : % -> % if R has DIFRING           D : (%,(R -> R)) -> %
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        antisymmetric? : % -> Boolean
--R coerce : R -> %                       coerce : Integer -> %
--R coerce : % -> OutputForm              column : (%,Integer) -> Col
--R copy : % -> %                         diagonal : % -> Row
--R diagonal? : % -> Boolean              diagonalMatrix : List R -> %
--R diagonalProduct : % -> R              elt : (%,Integer,Integer) -> R
--R elt : (%,Integer,Integer,R) -> R      empty : () -> %
--R empty? : % -> Boolean                 eq? : (%,%) -> Boolean
--R hash : % -> SingleInteger             latex : % -> String
--R listOfLists : % -> List List R        map : ((R -> R),%) -> %
--R map : (((R,R) -> R),%,%) -> %         matrix : List List R -> %
--R maxColIndex : % -> Integer            maxRowIndex : % -> Integer
--R minColIndex : % -> Integer            minRowIndex : % -> Integer
--R ncols : % -> NonNegativeInteger       nrows : % -> NonNegativeInteger
--R one? : % -> Boolean                   qelt : (%,Integer,Integer) -> R
--R recip : % -> Union(%,"failed")        retract : % -> R
--R row : (%,Integer) -> Row              sample : () -> %
--R scalarMatrix : R -> %                 square? : % -> Boolean
--R symmetric? : % -> Boolean             trace : % -> R
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R #? : % -> NonNegativeInteger if $ has finiteAggregate
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,Integer) -> % if R has FIELD
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (%,R) -> % if R has FIELD
--R D : (%,NonNegativeInteger) -> % if R has DIFRING
--R D : (%,Symbol) -> % if R has PDRING SYMBOL
--R D : (%,List Symbol) -> % if R has PDRING SYMBOL
--R D : (%,Symbol,NonNegativeInteger) -> % if R has PDRING SYMBOL
--R D : (%,List Symbol,List NonNegativeInteger) -> % if R has PDRING SYMBOL
--R D : (%,(R -> R),NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R any? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
--R characteristic : () -> NonNegativeInteger
--R coerce : Fraction Integer -> % if R has RETRACT FRAC INT
--R count : (R,%) -> NonNegativeInteger if R has SETCAT and $ has finiteAggregate
--R count : ((R -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
--R determinant : % -> R if R has commutative *
--R differentiate : % -> % if R has DIFRING
--R differentiate : (%,NonNegativeInteger) -> % if R has DIFRING
--R differentiate : (%,Symbol) -> % if R has PDRING SYMBOL
--R differentiate : (%,List Symbol) -> % if R has PDRING SYMBOL
--R differentiate : (%,Symbol,NonNegativeInteger) -> % if R has PDRING SYMBOL
--R differentiate : (%,List Symbol,List NonNegativeInteger) -> % if R has PDRING SYMBOL
--R differentiate : (%,(R -> R),NonNegativeInteger) -> %
--R differentiate : (%,(R -> R)) -> %
--R eval : (%,List R,List R) -> % if R has EVALAB R and R has SETCAT
--R eval : (%,R,R) -> % if R has EVALAB R and R has SETCAT
```

```
--R eval : (%,Equation R) -> % if R has EVALAB R and R has SETCAT
--R eval : (%,List Equation R) -> % if R has EVALAB R and R has SETCAT
--R every? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
--R exquo : (%,R) -> Union(%,"failed") if R has INTDOM
--R inverse : % -> Union(%,"failed") if R has FIELD
--R less? : (%,NonNegativeInteger) -> Boolean
--R map! : ((R -> R),%) -> % if $ has shallowlyMutable
--R member? : (R,%) -> Boolean if R has SETCAT and $ has finiteAggregate
--R members : % -> List R if $ has finiteAggregate
--R minordet : % -> R if R has commutative *
--R more? : (%,NonNegativeInteger) -> Boolean
--R nullSpace : % -> List Col if R has INTDOM
--R nullity : % -> NonNegativeInteger if R has INTDOM
--R parts : % -> List R if $ has finiteAggregate
--R rank : % -> NonNegativeInteger if R has INTDOM
--R reducedSystem : Matrix % -> Matrix R
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix R,vec: Vector R)
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer) if R
--R reducedSystem : Matrix % -> Matrix Integer if R has LINEXP INT
--R retract : % -> Fraction Integer if R has RETRACT FRAC INT
--R retract : % -> Integer if R has RETRACT INT
--R retractIfCan : % -> Union(R,"failed")
--R retractIfCan : % -> Union(Fraction Integer,"failed") if R has RETRACT FRAC INT
--R retractIfCan : % -> Union(Integer,"failed") if R has RETRACT INT
--R rowEchelon : % -> % if R has EUCDOM
--R size? : (%,NonNegativeInteger) -> Boolean
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R
--E 1

)spool
)lisp (bye)
```

———-

— SquareMatrixCategory.help —

```
=====================================================================
SquareMatrixCategory examples
=====================================================================

See Also:
o )show SquareMatrixCategory
```

———-

See:

⇐ "BiModule" (BMODULE) 9.0.122 on page 726
⇐ "DifferentialExtension" (DIFEXT) 11.0.153 on page 953
⇐ "FullyLinearlyExplicitRingOver" (FLINEXP) 11.0.154 on page 960
⇐ "FullyRetractableTo" (FRETRCT) 3.0.27 on page 87
⇐ "MatrixCategory" (MATCAT) 6.0.88 on page 412
⇐ "RectangularMatrixCategory" (RMATCAT) 10.0.147 on page 901

Exports:

0	1	antisymmetric?
any?	characteristic	coerce
column	copy	count
D	determinant	differentiate
diagonal	diagonal?	diagonalMatrix
diagonalProduct	elt	empty
empty?	eq?	eval
every?	exquo	hash
inverse	latex	less?
listOfLists	map	map!
matrix	maxColIndex	maxRowIndex
member?	members	minColIndex
minordet	minRowIndex	more?
ncols	nrows	nullSpace
nullity	one?	parts
qelt	rank	recip
reducedSystem	retract	retractIfCan
row	rowEchelon	sample
scalarMatrix	size?	square?
subtractIfCan	symmetric?	trace
zero?	#?	?^?
?*?	?**?	?+?
?-?	-?	?=?
?~=?	?/?	

Attributes Exported:

- **finiteAggregate** is true if it is an aggregate with a finite number of elements.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **nil**

TPDHERE: How did MATCAT get in the type tower?

```
determinant : % -> R if R has commutative *
inverse : % -> Union(%,"failed") if R has FIELD
```

These are directly exported but not implemented:

```
diagonalMatrix : List R -> %
minordet : % -> R if R has commutative *
scalarMatrix : R -> %
?*? : (Row,%) -> Row
?*? : (%,Col) -> Col
```

These are implemented by this category:

```
coerce : R -> %
diagonal : % -> Row
diagonalProduct : % -> R
differentiate : (%,(R -> R)) -> %
reducedSystem : Matrix % -> Matrix R
reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix R,vec: Vector R)
retract : % -> R
retractIfCan : % -> Union(R,"failed")
trace : % -> R
?**? : (%,Integer) -> % if R has FIELD
?**? : (%,NonNegativeInteger) -> %
```

These exports come from (p953) DifferentialExtension(R:Ring):

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : Integer -> %
coerce : % -> OutputForm
D : % -> % if R has DIFRING
D : (%,NonNegativeInteger) -> % if R has DIFRING
D : (%,(R -> R)) -> %
D : (%,(R -> R),NonNegativeInteger) -> %
D : (%,Symbol) -> % if R has PDRING SYMBOL
D : (%,List Symbol) -> % if R has PDRING SYMBOL
D : (%,Symbol,NonNegativeInteger) -> % if R has PDRING SYMBOL
D : (%,List Symbol,List NonNegativeInteger) -> % if R has PDRING SYMBOL
differentiate : (%,List Symbol) -> % if R has PDRING SYMBOL
differentiate : (%,Symbol,NonNegativeInteger) -> % if R has PDRING SYMBOL
differentiate : (%,List Symbol,List NonNegativeInteger) -> % if R has PDRING SYMBOL
differentiate : (%,NonNegativeInteger) -> % if R has DIFRING
differentiate : % -> % if R has DIFRING
differentiate : (%,(R -> R),NonNegativeInteger) -> %
differentiate : (%,Symbol) -> % if R has PDRING SYMBOL
hash : % -> SingleInteger
latex : % -> String
```

```
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
```

These exports come from (p726) BiModule(R:Ring,R:Ring):

```
?*? : (R,%) -> %
?*? : (%,R) -> %
```

These exports come from
(p901) RectangularMatrixCategory(ndim,ndim,R,Row,Col)
where ndim:NonNegativeInteger,R:Ring,Row:DirectProductCategory(ndim,R)
Col:DirectProductCategory(ndim,R):

```
antisymmetric? : % -> Boolean
any? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
column : (%,Integer) -> Col
copy : % -> %
count : (R,%) -> NonNegativeInteger if R has SETCAT and $ has finiteAggregate
count : ((R -> Boolean),%) -> NonNegativeInteger if $ has finiteAggregate
diagonal? : % -> Boolean
elt : (%,Integer,Integer) -> R
elt : (%,Integer,Integer,R) -> R
empty : () -> %
empty? : % -> Boolean
eq? : (%,%) -> Boolean
eval : (%,List R,List R) -> % if R has EVALAB R and R has SETCAT
eval : (%,R,R) -> % if R has EVALAB R and R has SETCAT
eval : (%,Equation R) -> % if R has EVALAB R and R has SETCAT
eval : (%,List Equation R) -> % if R has EVALAB R and R has SETCAT
every? : ((R -> Boolean),%) -> Boolean if $ has finiteAggregate
exquo : (%,R) -> Union(%,"failed") if R has INTDOM
less? : (%,NonNegativeInteger) -> Boolean
listOfLists : % -> List List R
map : ((R -> R),%) -> %
```

```
map : (((R,R) -> R),%,%) -> %
map! : ((R -> R),%) -> % if $ has shallowlyMutable
matrix : List List R -> %
maxColIndex : % -> Integer
maxRowIndex : % -> Integer
minColIndex : % -> Integer
minRowIndex : % -> Integer
members : % -> List R if $ has finiteAggregate
member? : (R,%) -> Boolean if R has SETCAT and $ has finiteAggregate
more? : (%,NonNegativeInteger) -> Boolean
ncols : % -> NonNegativeInteger
nrows : % -> NonNegativeInteger
nullity : % -> NonNegativeInteger if R has INTDOM
nullSpace : % -> List Col if R has INTDOM
parts : % -> List R if $ has finiteAggregate
qelt : (%,Integer,Integer) -> R
rank : % -> NonNegativeInteger if R has INTDOM
row : (%,Integer) -> Row
rowEchelon : % -> % if R has EUCDOM
size? : (%,NonNegativeInteger) -> Boolean
square? : % -> Boolean
symmetric? : % -> Boolean
#? : % -> NonNegativeInteger if $ has finiteAggregate
?/? : (%,R) -> % if R has FIELD
```

These exports come from (p87) FullyRetractableTo(R:Ring):

```
coerce : Fraction Integer -> % if R has RETRACT FRAC INT
retract : % -> Fraction Integer if R has RETRACT FRAC INT
retract : % -> Integer if R has RETRACT INT
retractIfCan : % -> Union(Fraction Integer,"failed") if R has RETRACT FRAC INT
retractIfCan : % -> Union(Integer,"failed") if R has RETRACT INT
```

These exports come from (p960) FullyLinearlyExplicitRingOver(R:Ring):

```
reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer) if R has
reducedSystem : Matrix % -> Matrix Integer if R has LINEXP INT
```

— category SMATCAT SquareMatrixCategory —

```
)abbrev category SMATCAT SquareMatrixCategory
++ Authors: Grabmeier, Gschnitzer, Williamson
++ Date Created: 1987
++ Date Last Updated: July 1990
++ Basic Operations:
++ Related Domains: SquareMatrix(ndim,R)
++ Also See:
++ AMS Classifications:
```

```
++ Keywords:
++ Examples:
++ References:
++ Description:
++ \spadtype{SquareMatrixCategory} is a general square matrix category which
++ allows different representations and indexing schemes.  Rows and
++ columns may be extracted with rows returned as objects of
++ type Row and colums returned as objects of type Col.

SquareMatrixCategory(ndim,R,Row,Col): Category == Definition where
  ndim : NonNegativeInteger
  R    : Ring
  Row  : DirectProductCategory(ndim,R)
  Col  : DirectProductCategory(ndim,R)
  I ==> Integer

  Definition ==> Join(DifferentialExtension R, BiModule(R, R),_
                      RectangularMatrixCategory(ndim,ndim,R,Row,Col),_
                      FullyRetractableTo R,_
                      FullyLinearlyExplicitRingOver R) with
    if R has CommutativeRing then Module(R)
    scalarMatrix: R -> %
      ++ \spad{scalarMatrix(r)} returns an n-by-n matrix with r's on the
      ++ diagonal and zeroes elsewhere.
    diagonalMatrix: List R -> %
      ++ \spad{diagonalMatrix(l)} returns a diagonal matrix with the elements
      ++ of l on the diagonal.
    diagonal: % -> Row
      ++ \spad{diagonal(m)} returns a row consisting of the elements on the
      ++ diagonal of the matrix m.
    trace: % -> R
      ++ \spad{trace(m)} returns the trace of the matrix m. this is the sum
      ++ of the elements on the diagonal of the matrix m.
    diagonalProduct: % -> R
      ++ \spad{diagonalProduct(m)} returns the product of the elements on the
      ++ diagonal of the matrix m.
    "*": (%,Col) -> Col
      ++ \spad{x * c} is the product of the matrix x and the column vector c.
      ++ Error: if the dimensions are incompatible.
    "*": (Row,%) -> Row
      ++ \spad{r * x} is the product of the row vector r and the matrix x.
      ++ Error: if the dimensions are incompatible.

--% Linear algebra

    if R has commutative("*") then
      Algebra R
      determinant: % -> R
        ++ \spad{determinant(m)} returns the determinant of the matrix m.
      minordet: % -> R
```

```
        ++ \spad{minordet(m)} computes the determinant of the matrix m
        ++ using minors.
    if R has Field then
      inverse: % -> Union(%,"failed")
        ++ \spad{inverse(m)} returns the inverse of the matrix m, if that
        ++ matrix is invertible and returns "failed" otherwise.
      "**": (%,Integer) -> %
        ++ \spad{m**n} computes an integral power of the matrix m.
        ++ Error: if the matrix is not invertible.

   add
    minr ==> minRowIndex
    maxr ==> maxRowIndex
    minc ==> minColIndex
    maxc ==> maxColIndex
    mini ==> minIndex
    maxi ==> maxIndex

    positivePower:(%,Integer) -> %
    positivePower(x,n) ==
--      one? n => x
      (n = 1) => x
      odd? n => x * positivePower(x,n - 1)
      y := positivePower(x,n quo 2)
      y * y

    x:% ** n:NonNegativeInteger ==
      zero? n => scalarMatrix 1
      positivePower(x,n)

    coerce(r:R) == scalarMatrix r

    equation2R: Vector % -> Matrix R

    differentiate(x:%,d:R -> R) == map(d,x)

    diagonal x ==
      v:Vector(R) := new(ndim,0)
      for i in minr x .. maxr x
        for j in minc x .. maxc x
          for k in minIndex v .. maxIndex v repeat
            qsetelt_!(v, k, qelt(x, i, j))
      directProduct v

    retract(x:%):R ==
      diagonal? x => retract diagonal x
      error "Not retractable"

    retractIfCan(x:%):Union(R, "failed") ==
      diagonal? x => retractIfCan diagonal x
```

```
      "failed"

    equation2R v ==
      ans:Matrix(Col) := new(ndim,#v,0)
      for i in minr ans .. maxr ans repeat
        for j in minc ans .. maxc ans repeat
          qsetelt_!(ans, i, j, column(qelt(v, j), i))
      reducedSystem ans

    reducedSystem(x:Matrix %):Matrix(R) ==
      empty? x => new(0,0,0)
      reduce(vertConcat, [equation2R row(x, i)
                              for i in minr x .. maxr x])$List(Matrix R)

    reducedSystem(m:Matrix %, v:Vector %):
     Record(mat:Matrix R, vec:Vector R) ==
      vh:Vector(R) :=
        empty? v => new(0,0)
        rh := reducedSystem(v::Matrix %)@Matrix(R)
        column(rh, minColIndex rh)
      [reducedSystem(m)@Matrix(R), vh]

    trace x ==
      tr : R := 0
      for i in minr(x)..maxr(x) for j in minc(x)..maxc(x) repeat
        tr := tr + x(i,j)
      tr

    diagonalProduct x ==
      pr : R := 1
      for i in minr(x)..maxr(x) for j in minc(x)..maxc(x) repeat
        pr := pr * x(i,j)
      pr

    if R has Field then

      x:% ** n:Integer ==
        zero? n => scalarMatrix 1
        positive? n => positivePower(x,n)
        (xInv := inverse x) case "failed" =>
          error "**: matrix must be invertible"
        positivePower(xInv :: %,-n)
```

———-

— SMATCAT.dotabb —

```
"SMATCAT"
```

```
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SMATCAT"];
"SMATCAT" -> "BMODULE"
"SMATCAT" -> "DIFEXT"
"SMATCAT" -> "FLINEXP"
"SMATCAT" -> "FRETRCT"
"SMATCAT" -> "RMATCAT"
```

———-

— SMATCAT.dotfull —

```
"SquareMatrixCategory(a:NonNegativeInteger,b:Ring,c:DirectProductCategory(a,b),d:DirectProductCa
 [color=lightblue,href="bookvol10.2.pdf#nameddest=SMATCAT"];
"SquareMatrixCategory(a:NonNegativeInteger,b:Ring,c:DirectProductCategory(a,b),d:DirectProductCa
 -> "BiModule(a:Ring,b:Ring)"
"SquareMatrixCategory(a:NonNegativeInteger,b:Ring,c:DirectProductCategory(a,b),d:DirectProductCa
 -> "DifferentialExtension(a:Ring)"
"SquareMatrixCategory(a:NonNegativeInteger,b:Ring,c:DirectProductCategory(a,b),d:DirectProductCa
 -> "FullyLinearlyExplicitRingOver(a:Ring)"
"SquareMatrixCategory(a:NonNegativeInteger,b:Ring,c:DirectProductCategory(a,b),d:DirectProductCa
 -> "FullyRetractableTo(a:Ring)"
"SquareMatrixCategory(a:NonNegativeInteger,b:Ring,c:DirectProductCategory(a,b),d:DirectProductCa
 -> "RectangularMatrixCategory(a:NonNegativeInteger,b:NonNegativeInteger,c:Ring,d:DirectProductC
```

———-

— SMATCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"SquareMatrixCategory(a:NonNegativeInteger,b:Ring,c:DirectProductCategory(a,b),d:DirectProductCa
 [color=lightblue];
"SquareMatrixCategory(a:NonNegativeInteger,b:Ring,c:DirectProductCategory(a,b),d:DirectProductCa
 -> "BiModule(a:Ring,b:Ring)"
"SquareMatrixCategory(a:NonNegativeInteger,b:Ring,c:DirectProductCategory(a,b),d:DirectProductCa
 -> "DifferentialExtension(a:Ring)"
"SquareMatrixCategory(a:NonNegativeInteger,b:Ring,c:DirectProductCategory(a,b),d:DirectProductCa
 -> "FLINEXP..."
"SquareMatrixCategory(a:NonNegativeInteger,b:Ring,c:DirectProductCategory(a,b),d:DirectProductCa
 -> "FRETRCT..."
"SquareMatrixCategory(a:NonNegativeInteger,b:Ring,c:DirectProductCategory(a,b),d:DirectProductCa
 -> "RectangularMatrixCategory(a:NonNegativeInteger,b:NonNegativeInteger,c:Ring,d:DirectProductC
```

```
"DifferentialExtension(a:Ring)" [color=lightblue];
"DifferentialExtension(a:Ring)" -> "Ring()"
"DifferentialExtension(a:Ring)" -> "DifferentialRing()"
"DifferentialExtension(a:Ring)" -> "PartialDifferentialRing(Symbol)"

"PartialDifferentialRing(Symbol)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=PDRING"];
"PartialDifferentialRing(Symbol)" ->
    "PartialDifferentialRing(a:SetCategory)"

"PartialDifferentialRing(a:SetCategory)" [color=lightblue];
"PartialDifferentialRing(a:SetCategory)" -> "Ring()"

"DifferentialRing()" [color=lightblue];
"DifferentialRing()" -> "Ring()"

"Ring()" [color=lightblue];
"Ring()" -> "Rng()"
"Ring()" -> "Monoid()"
"Ring()" -> "LeftModule(a:Ring)"

"Rng()" [color=lightblue];
"Rng()" -> "ABELGRP..."
"Rng()" -> "SGROUP..."

"Monoid()" [color=lightblue];
"Monoid()" -> "SGROUP..."

"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LMODULE..."

"RectangularMatrixCategory(a:NonNegativeInteger,b:NonNegativeInteger,c:Ring,d:DirectProductCategory(b,c),e:I
 [color=lightblue];
"RectangularMatrixCategory(a:NonNegativeInteger,b:NonNegativeInteger,c:Ring,d:DirectProductCategory(b,c),e:I
  -> "BiModule(a:Ring,b:Ring)"
"RectangularMatrixCategory(a:NonNegativeInteger,b:NonNegativeInteger,c:Ring,d:DirectProductCategory(b,c),e:I
  -> "HOAGG..."

"BiModule(a:Ring,b:Ring)" [color=lightblue];
"BiModule(a:Ring,b:Ring)" -> "LeftModule(a:Ring)"
"BiModule(a:Ring,b:Ring)" -> "RightModule(a:Ring)"

"RightModule(a:Ring)" [color=seagreen];
"RightModule(a:Ring)" -> "RightModule(a:Rng)"

"RightModule(a:Rng)" [color=lightblue];
"RightModule(a:Rng)" -> "ABELGRP..."

"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"
```

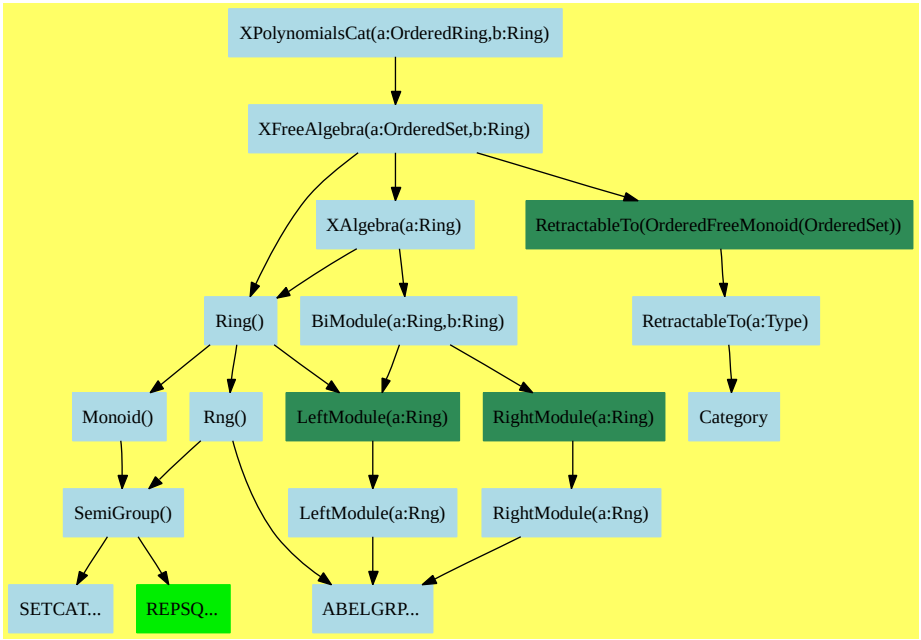
```
"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "ABELGRP..."

"FRETRCT..." [color=lightblue];
"FLINEXP..." [color=lightblue];
"SGROUP..." [color=lightblue];
"LMODULE..." [color=lightblue];
"HOAGG..." [color=lightblue];
"ABELGRP..." [color=lightblue];

}
```

———-

12.0.169 XPolynomialsCat (XPOLYC)



— **XPolynomialsCat.input** —

```
)set break resume
)sys rm -f XPolynomialsCat.output
)spool XPolynomialsCat.output
```

```
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show XPolynomialsCat
--R XPolynomialsCat(vl: OrderedSet,R: Ring)  is a category constructor
--R Abbreviation for XPolynomialsCat is XPOLYC
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for XPOLYC
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (vl,%) -> %                    ?*? : (%,R) -> %
--R ?*? : (R,%) -> %                     ?*? : (%,%) -> %
--R ?*? : (Integer,%) -> %               ?*? : (PositiveInteger,%) -> %
--R ?**? : (%,PositiveInteger) -> %      ?+? : (%,%) -> %
--R ?-? : (%,%) -> %                     -? : % -> %
--R ?=? : (%,%) -> Boolean               1 : () -> %
--R 0 : () -> %                          ?^? : (%,PositiveInteger) -> %
--R coef : (%,%) -> R                    coerce : vl -> %
--R coerce : R -> %                      coerce : Integer -> %
--R coerce : % -> OutputForm             constant : % -> R
--R constant? : % -> Boolean             degree : % -> NonNegativeInteger
--R hash : % -> SingleInteger            latex : % -> String
--R lquo : (%,%) -> %                    lquo : (%,vl) -> %
--R map : ((R -> R),%) -> %              mirror : % -> %
--R monomial? : % -> Boolean             one? : % -> Boolean
--R quasiRegular : % -> %                quasiRegular? : % -> Boolean
--R recip : % -> Union(%,"failed")       rquo : (%,%) -> %
--R rquo : (%,vl) -> %                   sample : () -> %
--R varList : % -> List vl               zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R coef : (%,OrderedFreeMonoid vl) -> R
--R coerce : OrderedFreeMonoid vl -> %
--R lquo : (%,OrderedFreeMonoid vl) -> %
--R maxdeg : % -> OrderedFreeMonoid vl
--R mindeg : % -> OrderedFreeMonoid vl
--R mindegTerm : % -> Record(k: OrderedFreeMonoid vl,c: R)
--R monom : (OrderedFreeMonoid vl,R) -> %
--R retract : % -> OrderedFreeMonoid vl
--R retractIfCan : % -> Union(OrderedFreeMonoid vl,"failed")
--R rquo : (%,OrderedFreeMonoid vl) -> %
--R sh : (%,NonNegativeInteger) -> % if R has COMRING
--R sh : (%,%) -> % if R has COMRING
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R trunc : (%,NonNegativeInteger) -> %
```

```
--R
--E 1

)spool
)lisp (bye)
```

———-

— **XPolynomialsCat.help** —

```
==========================================================================
XPolynomialsCat examples
==========================================================================

See Also:
o )show XPolynomialsCat
```

———-

See:

⇐ "XFreeAlgebra" (XFALG) 11.0.159 on page 992

Exports:

0	1	characteristic	coef	coerce
constant	constant?	degree	hash	latex
lquo	map	maxdeg	mindeg	mindegTerm
mirror	monom	monomial?	one?	quasiRegular
quasiRegular?	recip	retract	retractIfCan	rquo
sample	sh	subtractIfCan	trunc	varList
zero?	?*?	?**?	?+?	?-?
-?	?=?	?^?	?~=?	

Attributes Exported:

- if Ring has noZeroDivisors then noZeroDivisors where **noZeroDivisors** is true if $x*y \neq 0$ implies both x and y are non-zero.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
degree : % -> NonNegativeInteger
maxdeg : % -> OrderedFreeMonoid vl
trunc : (%,NonNegativeInteger) -> %
```

These exports come from (p69) Aggregate():

These exports come from (p992) XFreeAlgebra(vl:OrderedSet,R:Ring):

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coef : (%,OrderedFreeMonoid vl) -> R
coef : (%,%) -> R
coerce : % -> OutputForm
coerce : R -> %
coerce : OrderedFreeMonoid vl -> %
coerce : Integer -> %
coerce : vl -> %
constant : % -> R
constant? : % -> Boolean
hash : % -> SingleInteger
latex : % -> String
lquo : (%,OrderedFreeMonoid vl) -> %
lquo : (%,%) -> %
lquo : (%,vl) -> %
map : ((R -> R),%) -> %
mindeg : % -> OrderedFreeMonoid vl
mindegTerm : % -> Record(k: OrderedFreeMonoid vl,c: R)
mirror : % -> %
monom : (OrderedFreeMonoid vl,R) -> %
monomial? : % -> Boolean
one? : % -> Boolean
quasiRegular : % -> %
quasiRegular? : % -> Boolean
recip : % -> Union(%,"failed")
retract : % -> OrderedFreeMonoid vl
retractIfCan : % -> Union(OrderedFreeMonoid vl,"failed")
rquo : (%,OrderedFreeMonoid vl) -> %
rquo : (%,%) -> %
rquo : (%,vl) -> %
sample : () -> %
sh : (%,NonNegativeInteger) -> % if R has COMRING
sh : (%,%) -> % if R has COMRING
subtractIfCan : (%,%) -> Union(%,"failed")
varList : % -> List vl
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
```

```
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (Integer,%) -> %
?*? : (%,%) -> %
?*? : (R,%) -> %
?*? : (%,R) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?*? : (vl,%) -> %
```

These exports come from (p113) SetCategory():

— category XPOLYC XPolynomialsCat —

```
)abbrev category XPOLYC XPolynomialsCat
++ Author: Michel Petitot petitot@lifl.fr
++ Date Created: 91
++ Date Last Updated: 7 Juillet 92
++ Fix History: compilation v 2.1 le 13 dec 98
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The Category of polynomial rings with non-commutative variables.
++ The coefficient ring may be non-commutative too.
++ However coefficients commute with vaiables.

XPolynomialsCat(vl:OrderedSet,R:Ring):Category == Export where
  WORD ==> OrderedFreeMonoid(vl)

  Export == XFreeAlgebra(vl,R) with
    maxdeg: % -> WORD
      ++ \spad{maxdeg(p)} returns the greatest leading word in the
      ++ support of \spad{p}.
    degree: % -> NonNegativeInteger
      ++ \spad{degree(p)} returns the degree of \spad{p}.
      ++  Note that the degree of a word is its length.
    trunc : (% , NonNegativeInteger) -> %
      ++  \spad{trunc(p,n)} returns the polynomial \spad{p} truncated
```

```
      ++ at order \spad{n}.
```

———-

— **XPOLYC.dotabb** —

```
"XPOLYC"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=XPOLYC"];
"XPOLYC" -> "XFALG"
```

———-

— **XPOLYC.dotfull** —

```
"XPolynomialsCat(a:OrderedRing,b:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=XPOLYC"];
"XPolynomialsCat(a:OrderedRing,b:Ring)" ->
   "XFreeAlgebra(a:OrderedSet,b:Ring)"
```

———-

— **XPOLYC.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"XPolynomialsCat(a:OrderedRing,b:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=XPOLYC"];
"XPolynomialsCat(a:OrderedRing,b:Ring)" ->
   "XFreeAlgebra(a:OrderedSet,b:Ring)"

"XFreeAlgebra(a:OrderedSet,b:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=XFALG"];
"XFreeAlgebra(a:OrderedSet,b:Ring)" -> "Ring()"
"XFreeAlgebra(a:OrderedSet,b:Ring)" -> "XAlgebra(a:Ring)"
"XFreeAlgebra(a:OrderedSet,b:Ring)" ->
    "RetractableTo(OrderedFreeMonoid(OrderedSet))"

"RetractableTo(OrderedFreeMonoid(OrderedSet))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=RETRACT"];
"RetractableTo(OrderedFreeMonoid(OrderedSet))" -> "RetractableTo(a:Type)"

"XAlgebra(a:Ring)"
```

```
 [color=lightblue,href="bookvol10.2.pdf#nameddest=XALG"];
"XAlgebra(a:Ring)" -> "Ring()"
"XAlgebra(a:Ring)" -> "BiModule(a:Ring,b:Ring)"

"Ring()" [color=lightblue];
"Ring()" -> "Rng()"
"Ring()" -> "Monoid()"
"Ring()" -> "LeftModule(a:Ring)"

"BiModule(a:Ring,b:Ring)" [color=lightblue];
"BiModule(a:Ring,b:Ring)" -> "LeftModule(a:Ring)"
"BiModule(a:Ring,b:Ring)" -> "RightModule(a:Ring)"

"RightModule(a:Ring)" [color=seagreen];
"RightModule(a:Ring)" -> "RightModule(a:Rng)"

"RightModule(a:Rng)" [color=lightblue];
"RightModule(a:Rng)" -> "ABELGRP..."

"Rng()" [color=lightblue];
"Rng()" -> "ABELGRP..."
"Rng()" -> "SemiGroup()"

"Monoid()" [color=lightblue];
"Monoid()" -> "SemiGroup()"

"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "ABELGRP..."

"SemiGroup()" [color=lightblue];
"SemiGroup()" -> "SETCAT..."
"SemiGroup()" -> "REPSQ..."

"RetractableTo(a:Type)" [color=lightblue];
"RetractableTo(a:Type)" -> "Category"

"Category" [color=lightblue];

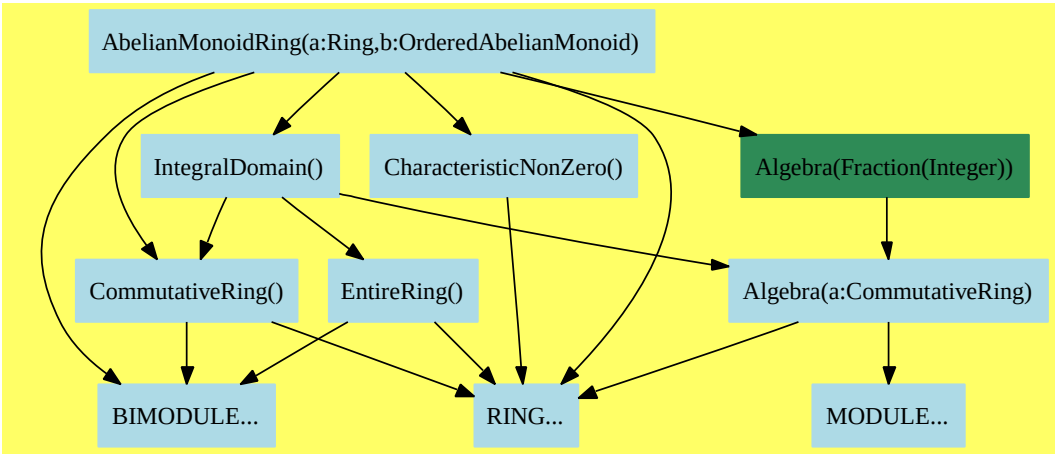
"REPSQ..." [color="#00EE00"];
"SETCAT..." [color=lightblue];
"ABELGRP..." [color=lightblue];
}
```

———-

Chapter 13

Category Layer 12

13.0.170 AbelianMonoidRing (AMR)



— AbelianMonoidRing.input —

```
)set break resume
)sys rm -f AbelianMonoidRing.output
)spool AbelianMonoidRing.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show AbelianMonoidRing
--R AbelianMonoidRing(R: Ring,E: OrderedAbelianMonoid)  is a category constructor
```

```
--R Abbreviation for AbelianMonoidRing is AMR
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for AMR
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (R,%) -> %                      ?*? : (%,R) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        coefficient : (%,E) -> R
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R degree : % -> E                       hash : % -> SingleInteger
--R latex : % -> String                   leadingCoefficient : % -> R
--R leadingMonomial : % -> %              map : ((R -> R),%) -> %
--R monomial : (R,E) -> %                 monomial? : % -> Boolean
--R one? : % -> Boolean                   recip : % -> Union(%,"failed")
--R reductum : % -> %                     sample : () -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (%,Fraction Integer) -> % if R has ALGEBRA FRAC INT
--R ?*? : (Fraction Integer,%) -> % if R has ALGEBRA FRAC INT
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (%,R) -> % if R has FIELD
--R ?^? : (%,NonNegativeInteger) -> %
--R associates? : (%,%) -> Boolean if R has INTDOM
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if R has CHARNZ
--R coerce : R -> % if R has COMRING
--R coerce : % -> % if R has INTDOM
--R coerce : Fraction Integer -> % if R has ALGEBRA FRAC INT
--R exquo : (%,%) -> Union(%,"failed") if R has INTDOM
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unit? : % -> Boolean if R has INTDOM
--R unitCanonical : % -> % if R has INTDOM
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %) if R has INTDOM
--R
--E 1

)spool
)lisp (bye)
```

———-

— AbelianMonoidRing.help —

```
=====================================================================
AbelianMonoidRing examples
```

```
================================================================
```

```
See Also:
o )show AbelianMonoidRing
```

———-

See:

⇒ “FiniteAbelianMonoidRing” (FAMR) 14.0.175 on page 1141
⇒ “PowerSeriesCategory” (PSCAT) 14.0.177 on page 1159
⇐ “BiModule” (BMODULE) 9.0.122 on page 726
⇐ “Ring” (RING) 9.0.129 on page 770

Exports:

0	1	associates?	characteristic
charthRoot	coefficient	coerce	degree
exquo	hash	latex	leadingCoefficient
leadingMonomial	map	monomial	monomial?
one?	recip	reductum	sample
subtractIfCan	unit?	unitCanonical	unitNormal
zero?	?*?	?**?	?+?
?-?	-?	?=?	?^?
?~=?	?/?		

Attributes exported:

- if $ has CommutativeRing then commutative(“*”) where **commutative(“*”)** is true if it has an operation ” $*$ ” : $(D, D) -> D$ which is commutative.
- if $ has IntegralDomain then noZeroDivisors where **noZeroDivisors** is true if $x*y \neq 0$ implies both x and y are non-zero.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return “failed” if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
coefficient : (%,E) -> R
degree : % -> E
leadingCoefficient : % -> R
leadingMonomial : % -> %
monomial : (R,E) -> %
reductum : % -> %
?/? : (%,R) -> % if R has FIELD
```

These are implemented by this category:

```
map : ((R -> R),%) -> %
monomial? : % -> Boolean
?*? : (Fraction Integer,%) -> % if R has ALGEBRA FRAC INT
```

These exports come from (p770) Ring():

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : % -> OutputForm
coerce : Integer -> %
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?**? : (%,NonNegativeInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (Integer,%) -> %
?*? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?^? : (%,PositiveInteger) -> %
```

These exports come from (p726) BiModule(R:Ring,R:Ring):

```
?*? : (R,%) -> %
?*? : (%,R) -> %
```

These exports come from (p1047) IntegralDomain():

```
associates? : (%,%) -> Boolean if R has INTDOM
coerce : % -> % if R has INTDOM
exquo : (%,%) -> Union(%,"failed") if R has INTDOM
unit? : % -> Boolean if R has INTDOM
unitCanonical : % -> % if R has INTDOM
unitNormal : % -> Record(unit: %,canonical: %,associate: %) if R has INTDOM
```

These exports come from (p833) CharacteristicNonZero():

```
 charthRoot : % -> Union(%,"failed") if R has CHARNZ
```

These exports come from (p843) CommutativeRing():

```
 coerce : R -> % if R has COMRING
```

These exports come from (p947) Algebra(Fraction(Integer)):

```
 coerce : Fraction Integer -> % if R has ALGEBRA FRAC INT
 ?*? : (%,Fraction Integer) -> % if R has ALGEBRA FRAC INT
```

— category AMR AbelianMonoidRing —

```
)abbrev category AMR AbelianMonoidRing
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ Abelian monoid ring elements (not necessarily of finite support)
++ of this ring are of the form formal SUM (r_i * e_i)
++ where the r_i are coefficents and the e_i, elements of the
++ ordered abelian monoid, are thought of as exponents or monomials.
++ The monomials commute with each other, and with
++ the coefficients (which themselves may or may not be commutative).
++ See \spadtype{FiniteAbelianMonoidRing} for the case of finite support
++ a useful common model for polynomials and power series.
++ Conceptually at least, only the non-zero terms are ever operated on.

AbelianMonoidRing(R:Ring, E:OrderedAbelianMonoid): Category ==
     Join(Ring,BiModule(R,R)) with
  leadingCoefficient: % -> R
    ++ leadingCoefficient(p) returns the coefficient highest
    ++ degree term of p.
  leadingMonomial: % -> %
    ++ leadingMonomial(p) returns the monomial of p with the highest degree.
  degree: % -> E
    ++ degree(p) returns the maximum of the exponents of the terms of p.
  map: (R -> R, %) -> %
    ++ map(fn,u) maps function fn onto the coefficients
    ++ of the non-zero monomials of u.
  monomial?: % -> Boolean
    ++ monomial?(p) tests if p is a single monomial.
```

```
 monomial: (R,E) -> %
   ++ monomial(r,e) makes a term from a coefficient r and an exponent e.
 reductum: % -> %
   ++ reductum(u) returns u minus its leading monomial
   ++ returns zero if handed the zero element.
 coefficient: (%,E) -> R
   ++ coefficient(p,e) extracts the coefficient of the monomial with
   ++ exponent e from polynomial p, or returns zero if exponent
   ++ is not present.
 if R has Field then "/": (%,R) -> %
   ++ p/c divides p by the coefficient c.
 if R has CommutativeRing then
    CommutativeRing
    Algebra R
 if R has CharacteristicZero then CharacteristicZero
 if R has CharacteristicNonZero then CharacteristicNonZero
 if R has IntegralDomain then IntegralDomain
 if R has Algebra Fraction Integer then Algebra Fraction Integer
add
 monomial? x == zero? reductum x

 map(fn:R -> R, x: %) ==
       -- this default definition assumes that reductum is cheap
    zero? x => 0
    r:=fn leadingCoefficient x
    zero? r => map(fn,reductum x)
    monomial(r, degree x) + map(fn,reductum x)

 if R has Algebra Fraction Integer then
   q:Fraction(Integer) * p:% == map(x1 +-> q * x1, p)
```

———-

— AMR.dotabb —

```
"AMR"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=AMR"];
"AMR" -> "RING"
"AMR" -> "BMODULE"
"AMR" -> "INTDOM"
"AMR" -> "CHARNZ"
"AMR" -> "COMRING"
"AMR" -> "ALGEBRA"
```

———-

— AMR.dotfull —

```
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=AMR"];
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" -> "Ring()"
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "BiModule(a:Ring,b:OrderedAbelianMonoid)"
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "IntegralDomain()"
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "CharacteristicNonZero()"
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "CommutativeRing()"
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "Algebra(Fraction(Integer))"
```

———-

— AMR.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" [color=lightblue];
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" -> "RING..."
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "BIMODULE..."
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "IntegralDomain()"
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "CharacteristicNonZero()"
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "CommutativeRing()"
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "Algebra(Fraction(Integer))"

"IntegralDomain()" [color=lightblue];
"IntegralDomain()" -> "CommutativeRing()"
"IntegralDomain()" -> "Algebra(a:CommutativeRing)"
"IntegralDomain()" -> "EntireRing()"

"EntireRing()" [color=lightblue];
"EntireRing()" -> "RING..."
"EntireRing()" -> "BIMODULE..."

"CharacteristicNonZero()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=CHARNZ"];
"CharacteristicNonZero()" -> "RING..."
```

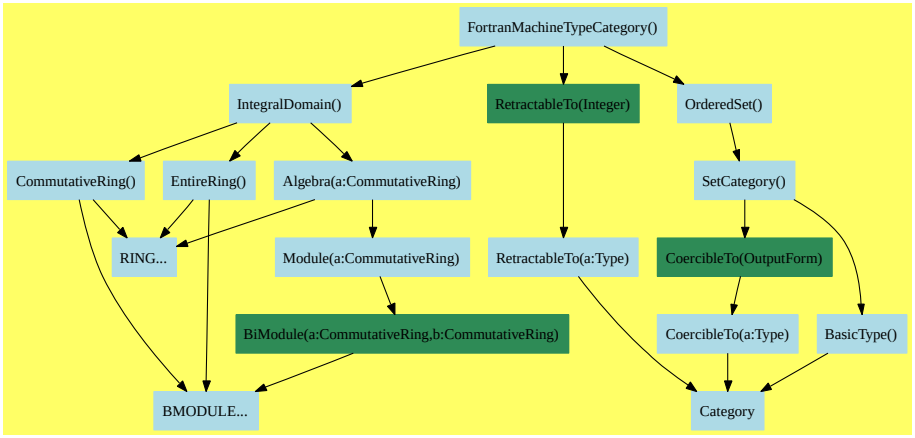
```
"Algebra(Fraction(Integer))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=ALGEBRA"];
"Algebra(Fraction(Integer))" -> "Algebra(a:CommutativeRing)"

"Algebra(a:CommutativeRing)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ALGEBRA"];
"Algebra(a:CommutativeRing)" -> "RING..."
"Algebra(a:CommutativeRing)" -> "MODULE..."

"CommutativeRing()" [color=lightblue];
"CommutativeRing()" -> "RING..."
"CommutativeRing()" -> "BIMODULE..."

"BIMODULE..." [color=lightblue];
"RING..." [color=lightblue];
"MODULE..." [color=lightblue];
}
```

13.0.171 FortranMachineTypeCategory (FMTC)



— FortranMachineTypeCategory.input —

```
)set break resume
)sys rm -f FortranMachineTypeCategory.output
)spool FortranMachineTypeCategory.output
)set message test on
```

```
)set message auto off
)clear all

--S 1 of 1
)show FortranMachineTypeCategory
--R FortranMachineTypeCategory  is a category constructor
--R Abbreviation for FortranMachineTypeCategory is FMTC
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FMTC
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?<? : (%,%) -> Boolean
--R ?<=? : (%,%) -> Boolean               ?=? : (%,%) -> Boolean
--R ?>? : (%,%) -> Boolean                ?>=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        associates? : (%,%) -> Boolean
--R coerce : Integer -> %                 coerce : % -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R hash : % -> SingleInteger             latex : % -> String
--R max : (%,%) -> %                      min : (%,%) -> %
--R one? : % -> Boolean                   recip : % -> Union(%,"failed")
--R retract : % -> Integer                sample : () -> %
--R unit? : % -> Boolean                  unitCanonical : % -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R exquo : (%,%) -> Union(%,"failed")
--R retractIfCan : % -> Union(Integer,"failed")
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R
--E 1

)spool
)lisp (bye)
```

———-

— FortranMachineTypeCategory.help —

```
====================================================================
FortranMachineTypeCategory examples
====================================================================
```

```
See Also:
o )show FortranMachineTypeCategory
```

See:

⇐ "IntegralDomain" (INTDOM) 12.0.164 on page 1047
⇐ "OrderedSet" (ORDSET) 4.0.56 on page 216
⇐ "RetractableTo" (RETRACT) 2.0.18 on page 55

Exports:

0	1	associates?	characteristic	coerce
exquo	hash	latex	max	min
one?	recip	retract	retractIfCan	sample
subtractIfCan	unit?	unitCanonical	unitNormal	zero?
?~=?	?^?	?*?	?**?	?+?
?-?	-?	?<?	?<=?	?=?
?>?	?>=?			

Attributes Exported:

- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These exports come from (p1047) IntegralDomain():

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean
characteristic : () -> NonNegativeInteger
coerce : Integer -> %
coerce : % -> OutputForm
coerce : % -> %
exquo : (%,%) -> Union(%,"failed")
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
```

```
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?^? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
```

These exports come from (p216) OrderedSet():

```
max : (%,%) -> %
min : (%,%) -> %
?<? : (%,%) -> Boolean
?>? : (%,%) -> Boolean
?<=? : (%,%) -> Boolean
?>=? : (%,%) -> Boolean
```

These exports come from (p55) RetractableTo(Integer):

```
coerce : Integer -> %
retract : % -> Integer
retractIfCan : % -> Union(Integer,"failed")
```

— category FMTC FortranMachineTypeCategory —

```
)abbrev category FMTC FortranMachineTypeCategory
++ Author: Mike Dewar
++ Date Created:  December 1993
++ Date Last Updated:
++ Basic Operations:
++ Related Domains:
++ Also See: FortranExpression, MachineInteger, MachineFloat, MachineComplex
++ AMS Classifications:
++ Keywords:
++ Examples:
++ References:
```

```
++ Description:
++ A category of domains which model machine arithmetic
++ used by machines in the AXIOM-NAG link.

FortranMachineTypeCategory():Category == Join(IntegralDomain,OrderedSet,
                                              RetractableTo(Integer) )
```

———-

— **FMTC.dotabb** —

```
"FMTC"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FMTC"];
"FMTC" -> "INTDOM"
"FMTC" -> "ORDSET"
"FMTC" -> "RETRACT"
```

———-

— **FMTC.dotfull** —

```
"FortranMachineTypeCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FMTC"];
"FortranMachineTypeCategory()" -> "IntegralDomain()"
"FortranMachineTypeCategory()" -> "OrderedSet()"
"FortranMachineTypeCategory()" -> "RetractableTo(Integer)"
```

———-

— **FMTC.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FortranMachineTypeCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FMTC"];
"FortranMachineTypeCategory()" -> "IntegralDomain()"
"FortranMachineTypeCategory()" -> "OrderedSet()"
"FortranMachineTypeCategory()" -> "RetractableTo(Integer)"

"RetractableTo(Integer)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=RETRACT"];
"RetractableTo(Integer)" -> "RetractableTo(a:Type)"
```

```
"RetractableTo(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RETRACT"];
"RetractableTo(a:Type)" -> "Category"

"OrderedSet()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ORDSET"];
"OrderedSet()" -> "SetCategory()"

"IntegralDomain()" [color=lightblue];
"IntegralDomain()" -> "CommutativeRing()"
"IntegralDomain()" -> "Algebra(a:CommutativeRing)"
"IntegralDomain()" -> "EntireRing()"

"CommutativeRing()" [color=lightblue];
"CommutativeRing()" -> "RING..."
"CommutativeRing()" -> "BMODULE..."

"EntireRing()" [color=lightblue];
"EntireRing()" -> "RING..."
"EntireRing()" -> "BMODULE..."

"Algebra(a:CommutativeRing)" [color=lightblue];
"Algebra(a:CommutativeRing)" -> "RING..."
"Algebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"

"Module(a:CommutativeRing)" [color=lightblue];
"Module(a:CommutativeRing)" ->
  "BiModule(a:CommutativeRing,b:CommutativeRing)"

"BiModule(a:CommutativeRing,b:CommutativeRing)" [color=seagreen];
"BiModule(a:CommutativeRing,b:CommutativeRing)" -> "BMODULE..."

"BMODULE..." [color=lightblue];
"RING..." [color=lightblue];

"SetCategory()" [color=lightblue];
"SetCategory()" -> "BasicType()"
"SetCategory()" -> "CoercibleTo(OutputForm)"

"BasicType()" [color=lightblue];
"BasicType()" -> "Category"

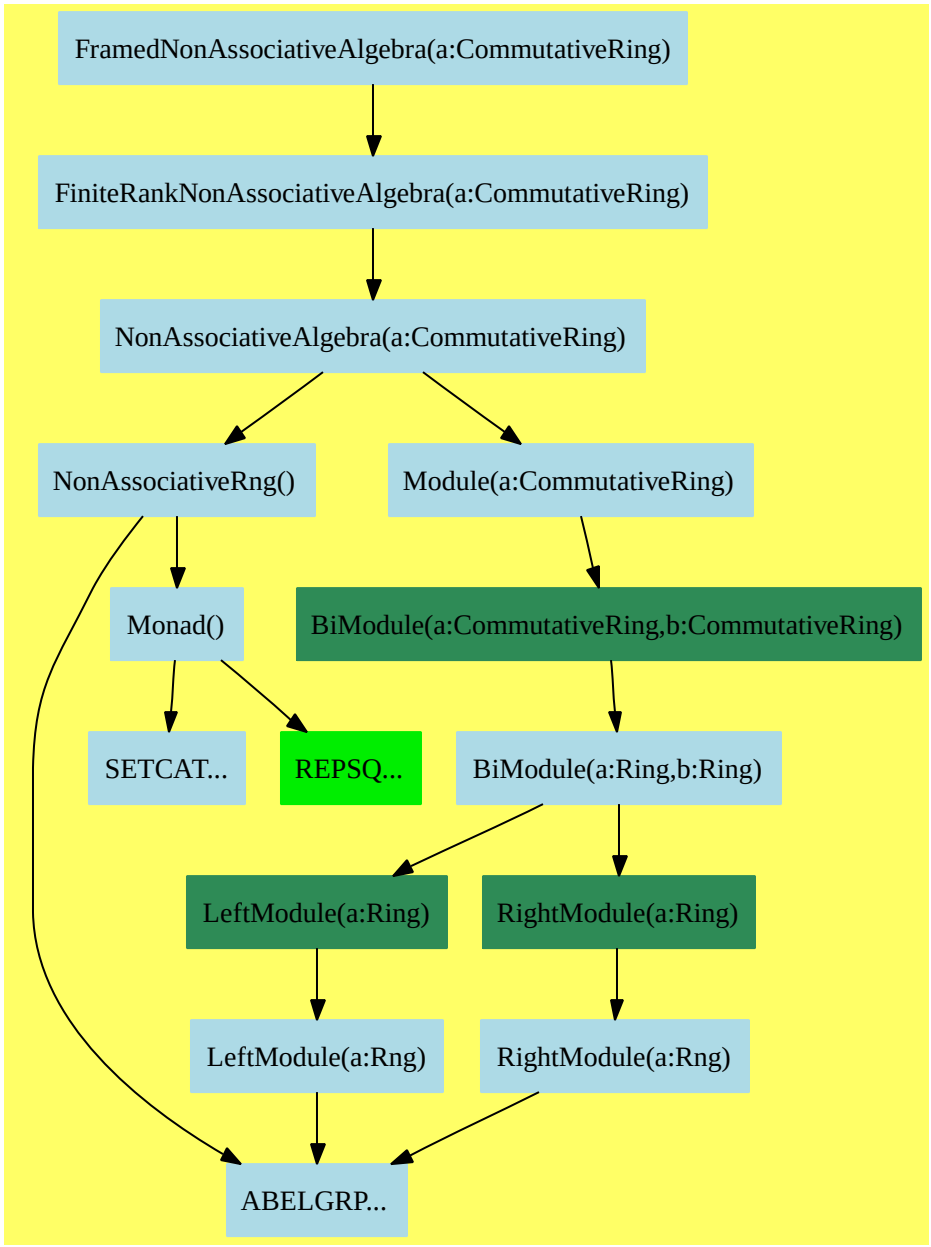
"CoercibleTo(OutputForm)" [color=seagreen];
"CoercibleTo(OutputForm)" -> "CoercibleTo(a:Type)"

"CoercibleTo(a:Type)" [color=lightblue];
"CoercibleTo(a:Type)" -> "Category"

"Category" [color=lightblue];
```

```
}
```

13.0.172 FramedNonAssociativeAlgebra (FRNAALG)



— **FramedNonAssociativeAlgebra.input** —

```
)set break resume
)sys rm -f FramedNonAssociativeAlgebra.output
)spool FramedNonAssociativeAlgebra.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FramedNonAssociativeAlgebra
--R FramedNonAssociativeAlgebra R: CommutativeRing  is a category constructor
--R Abbreviation for FramedNonAssociativeAlgebra is FRNAALG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FRNAALG
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (R,%) -> %                      ?*? : (%,R) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 0 : () -> %                           alternative? : () -> Boolean
--R antiAssociative? : () -> Boolean      antiCommutative? : () -> Boolean
--R antiCommutator : (%,%) -> %           apply : (Matrix R,%) -> %
--R associative? : () -> Boolean          associator : (%,%,%) -> %
--R basis : () -> Vector %                coerce : % -> OutputForm
--R commutative? : () -> Boolean          commutator : (%,%) -> %
--R convert : Vector R -> %               convert : % -> Vector R
--R coordinates : % -> Vector R           ?.? : (%,Integer) -> R
--R flexible? : () -> Boolean             hash : % -> SingleInteger
--R jacobiIdentity? : () -> Boolean       jordanAdmissible? : () -> Boolean
--R jordanAlgebra? : () -> Boolean        latex : % -> String
--R leftAlternative? : () -> Boolean      leftDiscriminant : () -> R
--R leftDiscriminant : Vector % -> R      leftNorm : % -> R
--R leftTrace : % -> R                    leftTraceMatrix : () -> Matrix R
--R lieAdmissible? : () -> Boolean        lieAlgebra? : () -> Boolean
--R powerAssociative? : () -> Boolean     rank : () -> PositiveInteger
--R represents : Vector R -> %            rightAlternative? : () -> Boolean
--R rightDiscriminant : () -> R           rightDiscriminant : Vector % -> R
--R rightNorm : % -> R                    rightTrace : % -> R
--R rightTraceMatrix : () -> Matrix R     sample : () -> %
--R someBasis : () -> Vector %            zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R associatorDependence : () -> List Vector R if R has INTDOM
--R conditionsForIdempotents : () -> List Polynomial R
--R conditionsForIdempotents : Vector % -> List Polynomial R
--R coordinates : Vector % -> Matrix R
--R coordinates : (Vector %,Vector %) -> Matrix R
--R coordinates : (%,Vector %) -> Vector R
--R leftCharacteristicPolynomial : % -> SparseUnivariatePolynomial R
```

```
--R leftMinimalPolynomial : % -> SparseUnivariatePolynomial R if R has INTDOM
--R leftPower : (%,PositiveInteger) -> %
--R leftRankPolynomial : () -> SparseUnivariatePolynomial Polynomial R if R has FIELD
--R leftRecip : % -> Union(%,"failed") if R has INTDOM
--R leftRegularRepresentation : % -> Matrix R
--R leftRegularRepresentation : (%,Vector %) -> Matrix R
--R leftTraceMatrix : Vector % -> Matrix R
--R leftUnit : () -> Union(%,"failed") if R has INTDOM
--R leftUnits : () -> Union(Record(particular: %,basis: List %),"failed") if R has INTDOM
--R noncommutativeJordanAlgebra? : () -> Boolean
--R plenaryPower : (%,PositiveInteger) -> %
--R recip : % -> Union(%,"failed") if R has INTDOM
--R represents : (Vector R,Vector %) -> %
--R rightCharacteristicPolynomial : % -> SparseUnivariatePolynomial R
--R rightMinimalPolynomial : % -> SparseUnivariatePolynomial R if R has INTDOM
--R rightPower : (%,PositiveInteger) -> %
--R rightRankPolynomial : () -> SparseUnivariatePolynomial Polynomial R if R has FIELD
--R rightRecip : % -> Union(%,"failed") if R has INTDOM
--R rightRegularRepresentation : % -> Matrix R
--R rightRegularRepresentation : (%,Vector %) -> Matrix R
--R rightTraceMatrix : Vector % -> Matrix R
--R rightUnit : () -> Union(%,"failed") if R has INTDOM
--R rightUnits : () -> Union(Record(particular: %,basis: List %),"failed") if R has INTDOM
--R structuralConstants : () -> Vector Matrix R
--R structuralConstants : Vector % -> Vector Matrix R
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unit : () -> Union(%,"failed") if R has INTDOM
--R
--E 1

)spool
)lisp (bye)
```

———-

— FramedNonAssociativeAlgebra.help —

```
====================================================================
FramedNonAssociativeAlgebra examples
====================================================================

See Also:
o )show FramedNonAssociativeAlgebra
```

———-

See:

⇐ “FiniteRankNonAssociativeAlgebra” (FINAALG) 12.0.162 on page 1019

Exports:

0	alternative?
antiAssociative?	antiCommutative?
antiCommutator	apply
associative?	associator
associatorDependence	basis
coerce	commutative?
commutator	conditionsForIdempotents
convert	coordinates
flexible?	hash
jacobiIdentity?	jordanAdmissible?
jordanAlgebra?	latex
leftAlternative?	leftCharacteristicPolynomial
leftDiscriminant	leftMinimalPolynomial
leftNorm	leftPower
leftRankPolynomial	leftRecip
leftRegularRepresentation	leftTrace
leftTraceMatrix	leftUnit
leftUnits	lieAdmissible?
lieAlgebra?	noncommutativeJordanAlgebra?
plenaryPower	powerAssociative?
rank	recip
represents	rightAlternative?
rightCharacteristicPolynomial	rightDiscriminant
rightMinimalPolynomial	rightNorm
rightPower	rightRankPolynomial
rightRecip	rightRegularRepresentation
rightTrace	rightTraceMatrix
rightUnit	rightUnits
sample	someBasis
structuralConstants	subtractIfCan
unit	zero?
?*?	?**?
?+?	?-?
-?	?=?
?.?	?~=?

Attributes exported:

- if $ has IntegralDomain then unitsKnown where **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
apply : (Matrix R,%) -> %
basis : () -> Vector %
convert : Vector R -> %
?.? : (%,Integer) -> R
```

These are implemented by this category:

```
conditionsForIdempotents : () -> List Polynomial R
convert : % -> Vector R
coordinates : % -> Vector R
coordinates : Vector % -> Matrix R
leftDiscriminant : () -> R
leftRankPolynomial : () ->
  SparseUnivariatePolynomial Polynomial R
    if R has FIELD
leftRegularRepresentation : % -> Matrix R
leftTraceMatrix : () -> Matrix R
leftUnit : () -> Union(%,"failed") if R has INTDOM
leftUnits : () ->
  Union(Record(particular: %,basis: List %),"failed")
    if R has INTDOM
represents : Vector R -> %
rightDiscriminant : () -> R
rightRankPolynomial : () ->
  SparseUnivariatePolynomial Polynomial R
    if R has FIELD
rightRegularRepresentation : % -> Matrix R
rightTraceMatrix : () -> Matrix R
rightUnit : () -> Union(%,"failed") if R has INTDOM
rightUnits : () ->
  Union(Record(particular: %,basis: List %),"failed")
    if R has INTDOM
structuralConstants : () -> Vector Matrix R
unit : () -> Union(%,"failed") if R has INTDOM
```

These exports come from (p1019) FiniteRankNonAssociativeAlgebra(R)
where R:CommutativeRing:

```
0 : () -> %
alternative? : () -> Boolean
antiAssociative? : () -> Boolean
antiCommutative? : () -> Boolean
antiCommutator : (%,%) -> %
associative? : () -> Boolean
associator : (%,%,%) -> %
associatorDependence : () -> List Vector R
    if R has INTDOM
```

```
coerce : % -> OutputForm
commutative? : () -> Boolean
commutator : (%,%) -> %
conditionsForIdempotents : Vector % -> List Polynomial R
coordinates : (Vector %,Vector %) -> Matrix R
coordinates : (%,Vector %) -> Vector R
flexible? : () -> Boolean
hash : % -> SingleInteger
jacobiIdentity? : () -> Boolean
jordanAdmissible? : () -> Boolean
jordanAlgebra? : () -> Boolean
latex : % -> String
leftAlternative? : () -> Boolean
leftCharacteristicPolynomial : % ->
  SparseUnivariatePolynomial R
leftDiscriminant : Vector % -> R
leftMinimalPolynomial : % ->
  SparseUnivariatePolynomial R
    if R has INTDOM
leftNorm : % -> R
leftPower : (%,PositiveInteger) -> %
leftRecip : % -> Union(%,"failed") if R has INTDOM
leftRegularRepresentation : (%,Vector %) -> Matrix R
leftTrace : % -> R
leftTraceMatrix : Vector % -> Matrix R
lieAdmissible? : () -> Boolean
lieAlgebra? : () -> Boolean
noncommutativeJordanAlgebra? : () -> Boolean
plenaryPower : (%,PositiveInteger) -> %
powerAssociative? : () -> Boolean
rank : () -> PositiveInteger
recip : % -> Union(%,"failed") if R has INTDOM
represents : (Vector R,Vector %) -> %
rightAlternative? : () -> Boolean
rightCharacteristicPolynomial : % ->
  SparseUnivariatePolynomial R
rightDiscriminant : Vector % -> R
rightMinimalPolynomial : % ->
  SparseUnivariatePolynomial R
    if R has INTDOM
rightNorm : % -> R
rightPower : (%,PositiveInteger) -> %
rightRecip : % -> Union(%,"failed") if R has INTDOM
rightRegularRepresentation : (%,Vector %) -> Matrix R
rightTrace : % -> R
rightTraceMatrix : Vector % -> Matrix R
sample : () -> %
someBasis : () -> Vector %
structuralConstants : Vector % -> Vector Matrix R
subtractIfCan : (%,%) -> Union(%,"failed")
```

```
 zero? : % -> Boolean
 ?~=? : (%,%) -> Boolean
 ?*? : (PositiveInteger,%) -> %
 ?*? : (%,%) -> %
 ?*? : (Integer,%) -> %
 ?*? : (NonNegativeInteger,%) -> %
 ?*? : (R,%) -> %
 ?*? : (%,R) -> %
 ?+? : (%,%) -> %
 ?=? : (%,%) -> Boolean
 ?-? : (%,%) -> %
 -? : % -> %
 ?**? : (%,PositiveInteger) -> %
```

— category FRNAALG FramedNonAssociativeAlgebra —

```
)abbrev category FRNAALG FramedNonAssociativeAlgebra
++ Author: J. Grabmeier, R. Wisbauer
++ Date Created: 01 March 1991
++ Date Last Updated: 11 June 1991
++ Basic Operations: +,-,*,**,basis
++ Related Constructors: FiniteRankNonAssociativeAlgebra, FramedAlgebra,
++   FiniteRankAssociativeAlgebra
++ Also See:
++ AMS Classifications:
++ Keywords: nonassociative algebra, basis
++ Reference:
++  R.D. Schafer: An Introduction to Nonassociative Algebras
++  Academic Press, New York, 1966
++ Description:
++ FramedNonAssociativeAlgebra(R) is a
++ \spadtype{FiniteRankNonAssociativeAlgebra} (i.e. a non associative
++ algebra over R which is a free \spad{R}-module of finite rank)
++ over a commutative ring R together with a fixed \spad{R}-module basis.

FramedNonAssociativeAlgebra(R:CommutativeRing):
        Category == FiniteRankNonAssociativeAlgebra(R) with
    basis: () -> Vector %
      ++ basis() returns the fixed \spad{R}-module basis.
    coordinates: % -> Vector R
      ++ coordinates(a) returns the coordinates of \spad{a}
      ++ with respect to the
      ++ fixed \spad{R}-module basis.
    coordinates: Vector % -> Matrix R
      ++ coordinates([a1,...,am]) returns a matrix whose i-th row
      ++ is formed by the coordinates of \spad{ai} with respect to the
      ++ fixed \spad{R}-module basis.
    elt : (%,Integer) -> R
      ++ elt(a,i) returns the i-th coefficient of \spad{a} with respect
```

```
    ++ to the fixed \spad{R}-module basis.
  structuralConstants:() -> Vector Matrix R
    ++ structuralConstants() calculates the structural constants
    ++ \spad{[(gammaijk) for k in 1..rank()]} defined by
    ++ \spad{vi * vj = gammaij1 * v1 + ... + gammaijn * vn},
    ++ where \spad{v1},...,\spad{vn} is the fixed \spad{R}-module basis.
  conditionsForIdempotents: () -> List Polynomial R
    ++ conditionsForIdempotents() determines a complete list
    ++ of polynomial equations for the coefficients of idempotents
    ++ with respect to the fixed \spad{R}-module basis.
  represents: Vector R -> %
    ++ represents([a1,...,an]) returns \spad{a1*v1 + ... + an*vn},
    ++ where \spad{v1}, ..., \spad{vn} are the elements of the
    ++ fixed \spad{R}-module basis.
  convert: % -> Vector R
    ++ convert(a) returns the coordinates of \spad{a} with respect to the
    ++ fixed \spad{R}-module basis.
  convert: Vector R -> %
    ++ convert([a1,...,an]) returns \spad{a1*v1 + ... + an*vn},
    ++ where \spad{v1}, ..., \spad{vn} are the elements of the
    ++ fixed \spad{R}-module basis.
  leftDiscriminant : () -> R
    ++ leftDiscriminant() returns the
    ++ determinant of the \spad{n}-by-\spad{n}
    ++ matrix whose element at the \spad{i}-th row and \spad{j}-th column
    ++ is given by the left trace of the product \spad{vi*vj}, where
    ++ \spad{v1},...,\spad{vn} are the
    ++ elements of the fixed \spad{R}-module basis.
    ++ Note that the same as \spad{determinant(leftTraceMatrix())}.
  rightDiscriminant : () -> R
    ++ rightDiscriminant() returns the determinant of the
    ++ \spad{n}-by-\spad{n} matrix whose element at the \spad{i}-th row
    ++ and \spad{j}-th column is
    ++ given by the right trace of the product \spad{vi*vj}, where
    ++ \spad{v1},...,\spad{vn} are the elements of
    ++ the fixed \spad{R}-module basis.
    ++ Note that the same as \spad{determinant(rightTraceMatrix())}.
  leftTraceMatrix : () -> Matrix R
    ++ leftTraceMatrix() is the \spad{n}-by-\spad{n}
    ++ matrix whose element at the \spad{i}-th row and \spad{j}-th column
    ++ is given by left trace of the product \spad{vi*vj},
    ++ where \spad{v1},...,\spad{vn} are the
    ++ elements of the fixed \spad{R}-module basis.
  rightTraceMatrix : () -> Matrix R
    ++ rightTraceMatrix() is the \spad{n}-by-\spad{n}
    ++ matrix whose element at the \spad{i}-th row and \spad{j}-th column
    ++ is given by the right trace of the product \spad{vi*vj}, where
    ++ \spad{v1},...,\spad{vn} are the elements
    ++ of the fixed \spad{R}-module basis.
  leftRegularRepresentation : % -> Matrix R
```

```
    ++ leftRegularRepresentation(a) returns the matrix of the linear
    ++ map defined by left multiplication by \spad{a} with respect
    ++ to the fixed \spad{R}-module basis.
  rightRegularRepresentation : % -> Matrix R
    ++ rightRegularRepresentation(a) returns the matrix of the linear
    ++ map defined by right multiplication by \spad{a} with respect
    ++ to the fixed \spad{R}-module basis.
  if R has Field then
    leftRankPolynomial : () -> SparseUnivariatePolynomial Polynomial R
      ++ leftRankPolynomial() calculates the left minimal polynomial
      ++ of the generic element in the algebra,
      ++ defined by the same structural
      ++ constants over the polynomial ring in symbolic coefficients with
      ++ respect to the fixed basis.
    rightRankPolynomial : () -> SparseUnivariatePolynomial Polynomial R
      ++ rightRankPolynomial() calculates the right minimal polynomial
      ++ of the generic element in the algebra,
      ++ defined by the same structural
      ++ constants over the polynomial ring in symbolic coefficients with
      ++ respect to the fixed basis.
  apply: (Matrix R, %) -> %
    ++ apply(m,a) defines a left operation of n by n matrices
    ++ where n is the rank of the algebra in terms of matrix-vector
    ++ multiplication, this is a substitute for a left module structure.
    ++ Error: if shape of matrix doesn't fit.
  --attributes
    --separable <=> discriminant() ^= 0
add

  V  ==> Vector
  M  ==> Matrix
  P  ==> Polynomial
  F  ==> Fraction
  REC  ==> Record(particular: Union(V R,"failed"),basis: List V R)
  LSMP ==> LinearSystemMatrixPackage(R,V R,V R, M R)
  CVMP ==> CoerceVectorMatrixPackage(R)

  --GA ==> GenericNonAssociativeAlgebra(R,rank()$%,_
  -- [random()$Character :: String :: Symbol for i in 1..rank()$%], _
  -- structuralConstants()$%)
  --y : GA := generic()
  if R has Field then
    leftRankPolynomial() ==
      n := rank()
      b := basis()
      gamma : Vector Matrix R := structuralConstants b
      listOfNumbers : List String :=  [STRINGIMAGE(q)$Lisp for q in 1..n]
      symbolsForCoef : Vector Symbol :=
        [concat("%", concat("x", i))::Symbol  for i in listOfNumbers]
      xx : M P R
```

```
  mo : P R
  x : M P R := new(1,n,0)
  for i in 1..n repeat
    mo := monomial(1, [symbolsForCoef.i], [1])$(P R)
    qsetelt_!(x,1,i,mo)
  y : M P R := copy x
  k  : PositiveInteger := 1
  cond : M P R := copy x
  -- multiplication in the generic algebra means using
  -- the structural matrices as bilinear forms.
  -- left multiplication by x, we prepare for that:
  genGamma : V M P R :=  coerceP$CVMP gamma
  x := reduce(horizConcat,[x*genGamma(i) for i in 1..#genGamma])
  while rank(cond) = k repeat
    k := k+1
    for i in 1..n repeat
      setelt(xx,[1],[i],x*transpose y)
    y := copy xx
    cond := horizConcat(cond, xx)
  vectorOfCoef : Vector P R := (nullSpace(cond)$Matrix(P R)).first
  res : SparseUnivariatePolynomial P R := 0
  for i in 1..k repeat
   res:=res+monomial(vectorOfCoef.i,i)$(SparseUnivariatePolynomial P R)
  res

rightRankPolynomial() ==
  n := rank()
  b := basis()
  gamma : Vector Matrix R := structuralConstants b
  listOfNumbers : List String :=  [STRINGIMAGE(q)$Lisp for q in 1..n]
  symbolsForCoef : Vector Symbol :=
    [concat("%", concat("x", i))::Symbol  for i in listOfNumbers]
  xx : M P R
  mo : P R
  x : M P R := new(1,n,0)
  for i in 1..n repeat
    mo := monomial(1, [symbolsForCoef.i], [1])$(P R)
    qsetelt_!(x,1,i,mo)
  y : M P R := copy x
  k  : PositiveInteger := 1
  cond : M P R := copy x
  -- multiplication in the generic algebra means using
  -- the structural matrices as bilinear forms.
  -- left multiplication by x, we prepare for that:
  genGamma : V M P R :=  coerceP$CVMP gamma
  x := _
   reduce(horizConcat,[genGamma(i)*transpose x for i in 1..#genGamma])
  while rank(cond) = k repeat
    k := k+1
    for i in 1..n repeat
```

```
      setelt(xx,[1],[i],y * transpose x)
    y := copy xx
    cond := horizConcat(cond, xx)
  vectorOfCoef : Vector P R := (nullSpace(cond)$Matrix(P R)).first
  res : SparseUnivariatePolynomial P R := 0
  for i in 1..k repeat
   res := _
    res+monomial(vectorOfCoef.i,i)$(SparseUnivariatePolynomial  P R)
  res

leftUnitsInternal : () -> REC
leftUnitsInternal() ==
  n := rank()
  b := basis()
  gamma : Vector Matrix R := structuralConstants b
  cond : Matrix(R) := new(n**2,n,0$R)$Matrix(R)
  rhs : Vector(R) := new(n**2,0$R)$Vector(R)
  z : Integer := 0
  addOn : R := 0
  for k in 1..n repeat
   for i in 1..n repeat
     z := z+1   -- index for the rows
     addOn :=
       k=i => 1
       0
     setelt(rhs,z,addOn)$Vector(R)
     for j in 1..n repeat  -- index for the columns
       setelt(cond,z,j,elt(gamma.k,j,i))$Matrix(R)
  solve(cond,rhs)$LSMP


leftUnit() ==
  res : REC := leftUnitsInternal()
  res.particular case "failed" =>
    messagePrint("this algebra has no left unit")$OutputForm
    "failed"
  represents (res.particular :: V R)

leftUnits() ==
  res : REC := leftUnitsInternal()
  res.particular case "failed" =>
    messagePrint("this algebra has no left unit")$OutputForm
    "failed"
  [represents(res.particular :: V R)$%, _
    map(represents, res.basis)$ListFunctions2(Vector R, %) ]

rightUnitsInternal : () -> REC
rightUnitsInternal() ==
  n := rank()
  b := basis()
```

```
  gamma : Vector Matrix R := structuralConstants b
  condo : Matrix(R) := new(n**2,n,0$R)$Matrix(R)
  rhs : Vector(R) := new(n**2,0$R)$Vector(R)
  z : Integer := 0
  addOn : R := 0
  for k in 1..n repeat
   for i in 1..n repeat
     z := z+1   -- index for the rows
     addOn :=
       k=i => 1
       0
     setelt(rhs,z,addOn)$Vector(R)
     for j in 1..n repeat  -- index for the columns
       setelt(condo,z,j,elt(gamma.k,i,j))$Matrix(R)
  solve(condo,rhs)$LSMP

rightUnit() ==
  res : REC := rightUnitsInternal()
  res.particular case "failed" =>
    messagePrint("this algebra has no right unit")$OutputForm
    "failed"
  represents (res.particular :: V R)

rightUnits() ==
  res : REC := rightUnitsInternal()
  res.particular case "failed" =>
    messagePrint("this algebra has no right unit")$OutputForm
    "failed"
  [represents(res.particular :: V R)$%, _
    map(represents, res.basis)$ListFunctions2(Vector R, %) ]

unit() ==
  n := rank()
  b := basis()
  gamma : Vector Matrix R := structuralConstants b
  cond : Matrix(R) := new(2*n**2,n,0$R)$Matrix(R)
  rhs : Vector(R) := new(2*n**2,0$R)$Vector(R)
  z : Integer := 0
  u : Integer := n*n
  addOn : R := 0
  for k in 1..n repeat
   for i in 1..n repeat
     z := z+1   -- index for the rows
     addOn :=
       k=i => 1
       0
     setelt(rhs,z,addOn)$Vector(R)
     setelt(rhs,u,addOn)$Vector(R)
     for j in 1..n repeat  -- index for the columns
       setelt(cond,z,j,elt(gamma.k,j,i))$Matrix(R)
```

```
            setelt(cond,u,j,elt(gamma.k,i,j))$Matrix(R)
        res : REC := solve(cond,rhs)$LSMP
        res.particular case "failed" =>
          messagePrint("this algebra has no unit")$OutputForm
          "failed"
        represents (res.particular :: V R)
    apply(m:Matrix(R),a:%) ==
      v : Vector R := coordinates(a)
      v := m *$Matrix(R) v
      convert v


    structuralConstants()   == structuralConstants basis()
    conditionsForIdempotents() == conditionsForIdempotents basis()
    convert(x:%):Vector(R)  == coordinates(x, basis())
    convert(v:Vector R):%   == represents(v, basis())
    leftTraceMatrix()       == leftTraceMatrix basis()
    rightTraceMatrix()      == rightTraceMatrix basis()
    leftDiscriminant()      == leftDiscriminant basis()
    rightDiscriminant()     == rightDiscriminant basis()
    leftRegularRepresentation x == leftRegularRepresentation(x, basis())
    rightRegularRepresentation x == rightRegularRepresentation(x, basis())
    coordinates x           == coordinates(x, basis())
    represents(v:Vector R):%== represents(v, basis())

    coordinates(v:Vector %) ==
      m := new(#v, rank(), 0)$Matrix(R)
      for i in minIndex v .. maxIndex v for j in minRowIndex m .. repeat
        setRow_!(m, j, coordinates qelt(v, i))
      m
```

———-

— FRNAALG.dotabb —

```
"FRNAALG"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FRNAALG"];
"FRNAALG" -> "FINAALG"
```

———-

— FRNAALG.dotfull —

```
"FramedNonAssociativeAlgebra(a:CommutativeRing)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FRNAALG"];
"FramedNonAssociativeAlgebra(a:CommutativeRing)" ->
```

```
    "FiniteRankNonAssociativeAlgebra(a:CommutativeRing)"
```

———-

— FRNAALG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FramedNonAssociativeAlgebra(a:CommutativeRing)" [color=lightblue];
"FramedNonAssociativeAlgebra(a:CommutativeRing)" ->
    "FiniteRankNonAssociativeAlgebra(a:CommutativeRing)"

"FiniteRankNonAssociativeAlgebra(a:CommutativeRing)" [color=lightblue];
"FiniteRankNonAssociativeAlgebra(a:CommutativeRing)" ->
    "NonAssociativeAlgebra(a:CommutativeRing)"

"NonAssociativeAlgebra(a:CommutativeRing)" [color=lightblue];
"NonAssociativeAlgebra(a:CommutativeRing)" -> "NonAssociativeRng()"
"NonAssociativeAlgebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"

"NonAssociativeRng()" [color=lightblue];
"NonAssociativeRng()" -> "ABELGRP..."
"NonAssociativeRng()" -> "Monad()"

"Monad()" [color=lightblue];
"Monad()" -> "SETCAT..."
"Monad()" -> "REPSQ..."

"Module(a:CommutativeRing)" [color=lightblue];
"Module(a:CommutativeRing)" ->
  "BiModule(a:CommutativeRing,b:CommutativeRing)"

"BiModule(a:CommutativeRing,b:CommutativeRing)" [color=seagreen];
"BiModule(a:CommutativeRing,b:CommutativeRing)" -> "BiModule(a:Ring,b:Ring)"

"BiModule(a:Ring,b:Ring)" [color=lightblue];
"BiModule(a:Ring,b:Ring)" -> "LeftModule(a:Ring)"
"BiModule(a:Ring,b:Ring)" -> "RightModule(a:Ring)"

"RightModule(a:Ring)" [color=seagreen];
"RightModule(a:Ring)" -> "RightModule(a:Rng)"

"RightModule(a:Rng)" [color=lightblue];
"RightModule(a:Rng)" -> "ABELGRP..."
```

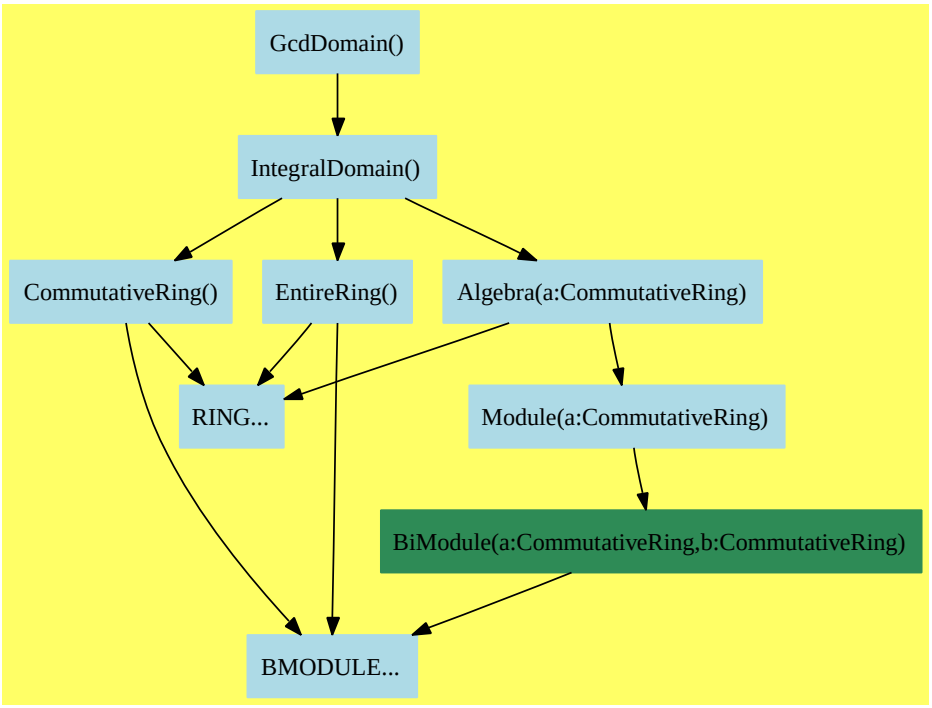
```
"LeftModule(a:Ring)" [color=seagreen];
"LeftModule(a:Ring)" -> "LeftModule(a:Rng)"

"LeftModule(a:Rng)" [color=lightblue];
"LeftModule(a:Rng)" -> "ABELGRP..."

"REPSQ..." [color="#00EE00"];
"SETCAT..." [color=lightblue];
"ABELGRP..." [color=lightblue];
}
```

———-

13.0.173 GcdDomain (GCDDOM)



— **GcdDomain.input** —

```
)set break resume
)sys rm -f GcdDomain.output
```

```
)spool GcdDomain.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show GcdDomain
--R GcdDomain  is a category constructor
--R Abbreviation for GcdDomain is GCDDOM
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for GCDDOM
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        associates? : (%,%) -> Boolean
--R coerce : % -> %                       coerce : Integer -> %
--R coerce : % -> OutputForm              gcd : List % -> %
--R gcd : (%,%) -> %                      hash : % -> SingleInteger
--R latex : % -> String                   lcm : List % -> %
--R lcm : (%,%) -> %                      one? : % -> Boolean
--R recip : % -> Union(%,"failed")        sample : () -> %
--R unit? : % -> Boolean                  unitCanonical : % -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R exquo : (%,%) -> Union(%,"failed")
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivari
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R
--E 1

)spool
)lisp (bye)
```

———-

— GcdDomain.help —

```
=====================================================================
GcdDomain examples
=====================================================================
```

```
See Also:
o )show GcdDomain
```

See:

⇒ “IntervalCategory” (INTCAT) 14.0.176 on page 1150
⇒ “PolynomialCategory” (POLYCAT) 16.0.189 on page 1255
⇒ “PrincipalIdealDomain” (PID) 14.0.178 on page 1166
⇒ “UniqueFactorizationDomain” (UFD) 14.0.179 on page 1172
⇒ “UnivariatePolynomialCategory” (UPOLYC) 17.0.204 on page 1471
⇐ “IntegralDomain” (INTDOM) 12.0.164 on page 1047

Exports:

0	1	associates?	characteristic	coerce
exquo	gcd	gcdPolynomial	hash	latex
lcm	one?	recip	sample	subtractIfCan
unit?	unitCanonical	unitNormal	zero?	?*?
?**?	?+?	?-?	-?	?=?
?^?	?~=?			

Attributes exported:

- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative(“*”)** is true if it has an operation $" * " : (D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return “failed” if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
gcd : (%,%) -> %
```

These are implemented by this category:

```
gcd : List % -> %
gcdPolynomial : (SparseUnivariatePolynomial %,
                 SparseUnivariatePolynomial %) ->
                    SparseUnivariatePolynomial %
lcm : (%,%) -> %
lcm : List % -> %
```

These exports come from (p1047) IntegralDomain():

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean
characteristic : () -> NonNegativeInteger
coerce : % -> %
coerce : Integer -> %
coerce : % -> OutputForm
exquo : (%,%) -> Union(%,"failed")
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
```

— category GCDDOM GcdDomain —

```
)abbrev category GCDDOM GcdDomain
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References: Davenport & Trager 1
++ Description:
++ This category describes domains where
++ \spadfun{gcd} can be computed but where there is no guarantee
```

```
++ of the existence of \spadfun{factor} operation for factorisation
++ into irreducibles. However, if such a \spadfun{factor} operation exist,
++ factorization will be unique up to order and units.

GcdDomain(): Category == IntegralDomain with
    gcd: (%,%) -> %
      ++ gcd(x,y) returns the greatest common divisor of x and y.
      -- gcd(x,y) = gcd(y,x) in the presence of canonicalUnitNormal,
      -- but not necessarily elsewhere
    gcd: List(%) -> %
      ++ gcd(l) returns the common gcd of the elements in the list l.
    lcm: (%,%) -> %
      ++ lcm(x,y) returns the least common multiple of x and y.
      -- lcm(x,y) = lcm(y,x) in the presence of canonicalUnitNormal,
      -- but not necessarily elsewhere
    lcm: List(%) -> %
      ++ lcm(l) returns the least common multiple of the elements of
      ++ the list l.
    gcdPolynomial: (SparseUnivariatePolynomial %, _
                    SparseUnivariatePolynomial %) -> _
                       SparseUnivariatePolynomial %
      ++ gcdPolynomial(p,q) returns the greatest common divisor (gcd) of
      ++ univariate polynomials over the domain
  add
   lcm(x: %,y: %) ==
     y = 0 => 0
     x = 0 => 0
     LCM : Union(%,"failed") := y exquo gcd(x,y)
     LCM case % =>  x * LCM
     error "bad gcd in lcm computation"
   lcm(l:List %) == reduce(lcm,l,1,0)
   gcd(l:List %) == reduce(gcd,l,0,1)
   SUP ==> SparseUnivariatePolynomial
   gcdPolynomial(p1,p2) ==
     zero? p1 => unitCanonical p2
     zero? p2 => unitCanonical p1
     c1:= content(p1); c2:= content(p2)
     p1:= (p1 exquo c1)::SUP %
     p2:= (p2 exquo c2)::SUP %
     if (e1:=minimumDegree p1) > 0 then p1:=(p1 exquo monomial(1,e1))::SUP %
     if (e2:=minimumDegree p2) > 0 then p2:=(p2 exquo monomial(1,e2))::SUP %
     e1:=min(e1,e2); c1:=gcd(c1,c2)
     p1:=
        degree p1 = 0 or degree p2 = 0 => monomial(c1,0)
        p:= subResultantGcd(p1,p2)
        degree p = 0 => monomial(c1,0)
        c2:= gcd(leadingCoefficient p1,leadingCoefficient p2)
        unitCanonical(_
          c1 * primitivePart(((c2*p) exquo leadingCoefficient p)::SUP %))
     zero? e1 => p1
```

```
    monomial(1,e1)*p1
```

———-

— GCDDOM.dotabb —

```
"GCDDOM"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=GCDDOM"];
"GCDDOM" -> "INTDOM"
```

———-

— GCDDOM.dotfull —

```
"GcdDomain()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=GCDDOM"];
"GcdDomain()" -> "IntegralDomain()"
```

———-

— GCDDOM.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"GcdDomain()" [color=lightblue];
"GcdDomain()" -> "IntegralDomain()"

"IntegralDomain()" [color=lightblue];
"IntegralDomain()" -> "CommutativeRing()"
"IntegralDomain()" -> "Algebra(a:CommutativeRing)"
"IntegralDomain()" -> "EntireRing()"

"CommutativeRing()" [color=lightblue];
"CommutativeRing()" -> "RING..."
"CommutativeRing()" -> "BMODULE..."

"EntireRing()" [color=lightblue];
"EntireRing()" -> "RING..."
"EntireRing()" -> "BMODULE..."

"Algebra(a:CommutativeRing)" [color=lightblue];
"Algebra(a:CommutativeRing)" -> "RING..."
```

```
"Algebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"

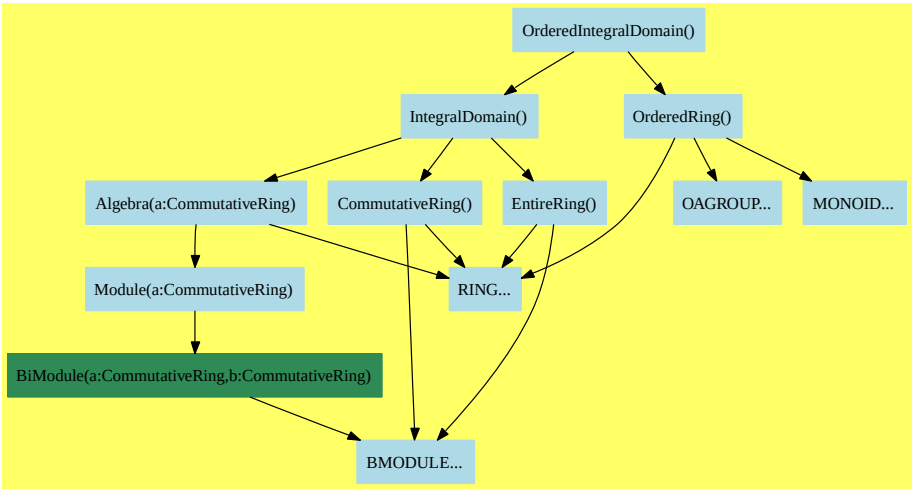
"Module(a:CommutativeRing)" [color=lightblue];
"Module(a:CommutativeRing)" ->
  "BiModule(a:CommutativeRing,b:CommutativeRing)"

"BiModule(a:CommutativeRing,b:CommutativeRing)" [color=seagreen];
"BiModule(a:CommutativeRing,b:CommutativeRing)" -> "BMODULE..."

"BMODULE..." [color=lightblue];
"RING..." [color=lightblue];
}
```

———-

13.0.174 OrderedIntegralDomain (OINTDOM)



— **OrderedIntegralDomain.input** —

```
)set break resume
)sys rm -f OrderedIntegralDomain.output
)spool OrderedIntegralDomain.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
```

```
)show OrderedIntegralDomain
--R OrderedIntegralDomain  is a category constructor
--R Abbreviation for OrderedIntegralDomain is OINTDOM
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for OINTDOM
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?<? : (%,%) -> Boolean
--R ?<=? : (%,%) -> Boolean               ?=? : (%,%) -> Boolean
--R ?>? : (%,%) -> Boolean                ?>=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        abs : % -> %
--R associates? : (%,%) -> Boolean        coerce : % -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R hash : % -> SingleInteger             latex : % -> String
--R max : (%,%) -> %                      min : (%,%) -> %
--R negative? : % -> Boolean              one? : % -> Boolean
--R positive? : % -> Boolean              recip : % -> Union(%,"failed")
--R sample : () -> %                      sign : % -> Integer
--R unit? : % -> Boolean                  unitCanonical : % -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R exquo : (%,%) -> Union(%,"failed")
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R
--E 1

)spool
)lisp (bye)
```

———-

— OrderedIntegralDomain.help —

```
=====================================================================
OrderedIntegralDomain examples
=====================================================================

See Also:
o )show OrderedIntegralDomain
```

———-

See:

⇒ “IntegerNumberSystem” (INS) 16.0.186 on page 1228
⇐ “IntegralDomain” (INTDOM) 12.0.164 on page 1047
⇐ “QuotientFieldCategory” (QFCAT) 17.0.198 on page 1376
⇐ “OrderedRing” (ORDRING) 10.0.144 on page 879

Exports:

1	0	abs	associates?	characteristic
coerce	exquo	hash	latex	max
min	negative?	one?	positive?	recip
sample	sign	subtractIfCan	unit?	unitCanonical
unitNormal	zero?	?*?	?**?	?+?
?-?	-?	?<?	?<=?	?=?
?>?	?>=?	?^?	?~=?	

Attributes exported:

- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative(“*”)** is true if it has an operation $" * " : (D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return “failed” if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These exports come from (p1047) IntegralDomain():

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean
characteristic : () -> NonNegativeInteger
coerce : % -> %
coerce : Integer -> %
coerce : % -> OutputForm
exquo : (%,%) -> Union(%,"failed")
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean
unitCanonical : % -> %
```

```
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
?<? : (%,%) -> Boolean
?^? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
```

These exports come from (p879) OrderedRing():

```
abs : % -> %
max : (%,%) -> %
min : (%,%) -> %
negative? : % -> Boolean
positive? : % -> Boolean
sign : % -> Integer
?<=? : (%,%) -> Boolean
?>? : (%,%) -> Boolean
?>=? : (%,%) -> Boolean
```

— category OINTDOM OrderedIntegralDomain —

```
)abbrev category OINTDOM OrderedIntegralDomain
++ Author: JH Davenport (after L Gonzalez-Vega)
++ Date Created: 30.1.96
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ Description:
++ The category of ordered commutative integral domains, where ordering
++ and the arithmetic operations are compatible

OrderedIntegralDomain(): Category ==
  Join(IntegralDomain, OrderedRing)
```

———-

— **OINTDOM.dotabb** —

```
"OINTDOM"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OINTDOM"];
"OINTDOM" -> "INTDOM"
"OINTDOM" -> "ORDRING"
```

———-

— **OINTDOM.dotfull** —

```
"OrderedIntegralDomain()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OINTDOM"];
"OrderedIntegralDomain()" -> "IntegralDomain()"
"OrderedIntegralDomain()" -> "OrderedRing()"
```

———-

— **OINTDOM.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"OrderedIntegralDomain()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=OINTDOM"];
"OrderedIntegralDomain()" -> "IntegralDomain()"
"OrderedIntegralDomain()" -> "OrderedRing()"

"IntegralDomain()" [color=lightblue];
"IntegralDomain()" -> "CommutativeRing()"
"IntegralDomain()" -> "Algebra(a:CommutativeRing)"
"IntegralDomain()" -> "EntireRing()"

"OrderedRing()" [color=lightblue];
"OrderedRing()" -> "OAGROUP..."
"OrderedRing()" -> "RING..."
"OrderedRing()" -> "MONOID..."

"CommutativeRing()" [color=lightblue];
"CommutativeRing()" -> "RING..."
"CommutativeRing()" -> "BMODULE..."

"EntireRing()" [color=lightblue];
```

```
"EntireRing()" -> "RING..."
"EntireRing()" -> "BMODULE..."

"Algebra(a:CommutativeRing)" [color=lightblue];
"Algebra(a:CommutativeRing)" -> "RING..."
"Algebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"

"Module(a:CommutativeRing)" [color=lightblue];
"Module(a:CommutativeRing)" ->
  "BiModule(a:CommutativeRing,b:CommutativeRing)"

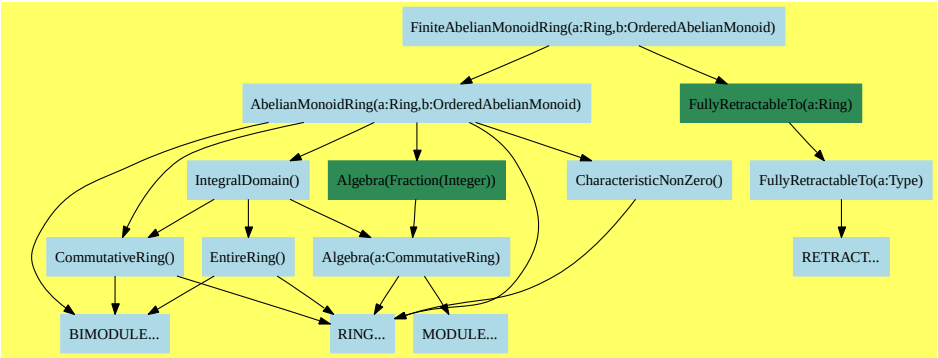
"BiModule(a:CommutativeRing,b:CommutativeRing)" [color=seagreen];
"BiModule(a:CommutativeRing,b:CommutativeRing)" -> "BMODULE..."

"OAGROUP..." [color=lightblue];
"BMODULE..." [color=lightblue];
"RING..." [color=lightblue];
"MONOID..." [color=lightblue];
}
```

Chapter 14

Category Layer 13

14.0.175 FiniteAbelianMonoidRing (FAMR)



— **FiniteAbelianMonoidRing.input** —

```
)set break resume
)sys rm -f FiniteAbelianMonoidRing.output
)spool FiniteAbelianMonoidRing.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FiniteAbelianMonoidRing
--R FiniteAbelianMonoidRing(R: Ring,E: OrderedAbelianMonoid)  is a category constructor
--R Abbreviation for FiniteAbelianMonoidRing is FAMR
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FAMR
```

```
--R
--R------------------------------- Operations --------------------------------
--R ?*? : (R,%) -> %                      ?*? : (%,R) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        coefficient : (%,E) -> R
--R coefficients : % -> List R            coerce : R -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R degree : % -> E                       ground : % -> R
--R ground? : % -> Boolean                hash : % -> SingleInteger
--R latex : % -> String                   leadingCoefficient : % -> R
--R leadingMonomial : % -> %              map : ((R -> R),%) -> %
--R mapExponents : ((E -> E),%) -> %      minimumDegree : % -> E
--R monomial : (R,E) -> %                 monomial? : % -> Boolean
--R one? : % -> Boolean                   pomopo! : (%,R,E,%) -> %
--R recip : % -> Union(%,"failed")        reductum : % -> %
--R retract : % -> R                      sample : () -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (%,Fraction Integer) -> % if R has ALGEBRA FRAC INT
--R ?*? : (Fraction Integer,%) -> % if R has ALGEBRA FRAC INT
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (%,R) -> % if R has FIELD
--R ?^? : (%,NonNegativeInteger) -> %
--R associates? : (%,%) -> Boolean if R has INTDOM
--R binomThmExpt : (%,%,NonNegativeInteger) -> % if R has COMRING
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if R has CHARNZ
--R coerce : Fraction Integer -> % if R has RETRACT FRAC INT or R has ALGEBRA FRAC INT
--R coerce : % -> % if R has INTDOM
--R content : % -> R if R has GCDDOM
--R exquo : (%,R) -> Union(%,"failed") if R has INTDOM
--R exquo : (%,%) -> Union(%,"failed") if R has INTDOM
--R numberOfMonomials : % -> NonNegativeInteger
--R primitivePart : % -> % if R has GCDDOM
--R retract : % -> Fraction Integer if R has RETRACT FRAC INT
--R retract : % -> Integer if R has RETRACT INT
--R retractIfCan : % -> Union(R,"failed")
--R retractIfCan : % -> Union(Fraction Integer,"failed") if R has RETRACT FRAC INT
--R retractIfCan : % -> Union(Integer,"failed") if R has RETRACT INT
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unit? : % -> Boolean if R has INTDOM
--R unitCanonical : % -> % if R has INTDOM
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %) if R has INTDOM
--R
--E 1
```

```
)spool
)lisp (bye)
```

———-

— **FiniteAbelianMonoidRing.help** —

```
====================================================================
FiniteAbelianMonoidRing examples
====================================================================

See Also:
o )show FiniteAbelianMonoidRing
```

———-

See:

⇒ “PolynomialCategory” (POLYCAT) 16.0.189 on page 1255
⇐ “AbelianMonoidRing” (AMR) 13.0.170 on page 1101
⇐ “FullyRetractableTo” (FRETRCT) 3.0.27 on page 87

Exports:

0	1	associates?	binomThmExpt
characteristic	charthRoot	coefficient	coefficients
coerce	content	degree	exquo
ground	ground?	hash	latex
leadingCoefficient	leadingMonomial	map	mapExponents
minimumDegree	monomial	monomial?	numberOfMonomials
one?	pomopo!	primitivePart	recip
reductum	retract	retractIfCan	sample
subtractIfCan	unit?	unitCanonical	unitNormal
zero?	?*?	?**?	?+?
?-?	-?	?=?	?^?
?~=?	?/?		

Attributes exported:

- if $ has CommutativeRing then commutative(“*”) where **commutative(“*”)** is true if it has an operation ” $*$ ” : $(D, D) -> D$ which is commutative.
- if $ has IntegralDomain then noZeroDivisors where **noZeroDivisors** is true if $x*y \neq 0$ implies both x and y are non-zero.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return “failed” if its argument is not a unit.

- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
minimumDegree : % -> E
numberOfMonomials : % -> NonNegativeInteger
```

These are implemented by this category:

```
binomThmExpt : (%,%,NonNegativeInteger) -> %
    if R has COMRING
coefficients : % -> List R
content : % -> R if R has GCDDOM
exquo : (%,R) -> Union(%,"failed") if R has INTDOM
ground : % -> R
ground? : % -> Boolean
mapExponents : ((E -> E),%) -> %
pomopo! : (%,R,E,%) -> %
primitivePart : % -> % if R has GCDDOM
?/? : (%,R) -> % if R has FIELD
```

These exports come from (p1101) AbelianMonoidRing(R,E)
where R:Ring and E:OrderedAbelianMonoid:

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean if R has INTDOM
characteristic : () -> NonNegativeInteger
charthRoot : % -> Union(%,"failed") if R has CHARNZ
coefficient : (%,E) -> R
coerce : R -> %
coerce : Fraction Integer -> %
    if R has RETRACT FRAC INT
    or R has ALGEBRA FRAC INT
coerce : % -> % if R has INTDOM
coerce : % -> OutputForm
coerce : Integer -> %
degree : % -> E
exquo : (%,%) -> Union(%,"failed") if R has INTDOM
hash : % -> SingleInteger
latex : % -> String
leadingCoefficient : % -> R
leadingMonomial : % -> %
map : ((R -> R),%) -> %
monomial : (R,E) -> %
monomial? : % -> Boolean
one? : % -> Boolean
```

```
recip : % -> Union(%,"failed")
reductum : % -> %
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean if R has INTDOM
unitCanonical : % -> % if R has INTDOM
unitNormal :
   % -> Record(unit: %,canonical: %,associate: %)
   if R has INTDOM
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (R,%) -> %
?*? : (%,R) -> %
?*? : (Fraction Integer,%) -> % if R has ALGEBRA FRAC INT
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (Integer,%) -> %
?*? : (%,%) -> %
?*? : (%,Fraction Integer) -> % if R has ALGEBRA FRAC INT
?**? : (%,NonNegativeInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?^? : (%,PositiveInteger) -> %
```

These exports come from (p87) FullyRetractableTo(R:Ring):

```
retract : % -> Fraction Integer
    if R has RETRACT FRAC INT
retract : % -> Integer if R has RETRACT INT
retract : % -> R
retractIfCan : % -> Union(R,"failed")
retractIfCan : % -> Union(Integer,"failed")
    if R has RETRACT INT
retractIfCan : % -> Union(Fraction Integer,"failed")
    if R has RETRACT FRAC INT
```

— category FAMR FiniteAbelianMonoidRing —

```
)abbrev category FAMR FiniteAbelianMonoidRing
++ Author:
++ Date Created:
++ Date Last Updated: 14.08.2000 Exported pomopo! and binomThmExpt [MMM]
++ Basic Functions:
++ Related Constructors:
++ Also See:
```

```
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ This category is similar to AbelianMonoidRing, except that the sum is
++ assumed to be finite. It is a useful model for polynomials,
++ but is somewhat more general.

FiniteAbelianMonoidRing(R:Ring, E:OrderedAbelianMonoid): Category ==
   Join(AbelianMonoidRing(R,E),FullyRetractableTo R) with
    ground?: % -> Boolean
      ++ ground?(p) tests if polynomial p is a member of the
      ++ coefficient ring.
      -- can't be defined earlier, since a power series
      -- might not know if there were other terms or not
    ground: % -> R
      ++ ground(p) retracts polynomial p to the coefficient ring.
    coefficients: % -> List R
      ++ coefficients(p) gives the list of non-zero coefficients
      ++ of polynomial p.
    numberOfMonomials: % -> NonNegativeInteger
      ++ numberOfMonomials(p) gives the number of non-zero monomials
      ++ in polynomial p.
    minimumDegree: % -> E
      ++ minimumDegree(p) gives the least exponent of a non-zero term
      ++ of polynomial p. Error: if applied to 0.
    mapExponents: (E -> E, %) -> %
      ++ mapExponents(fn,u) maps function fn onto the exponents
      ++ of the non-zero monomials of polynomial u.
    pomopo!: (%,R,E,%) -> %
      ++ \spad{pomopo!(p1,r,e,p2)} returns \spad{p1 + monomial(e,r) * p2}
      ++ and may use \spad{p1} as workspace. The constaant \spad{r} is
      ++ assumed to be nonzero.
    if R has CommutativeRing then
       binomThmExpt: (%,%,NonNegativeInteger) -> %
         ++ \spad{binomThmExpt(p,q,n)} returns \spad{(x+y)^n}
         ++ by means of the binomial theorem trick.
    if R has IntegralDomain then
       "exquo": (%,R) -> Union(%,"failed")
       ++ exquo(p,r) returns the exact quotient of polynomial p by r,
       ++ or "failed" if none exists.
    if R has GcdDomain then
       content: % -> R
         ++ content(p) gives the gcd of the coefficients of polynomial p.
       primitivePart: % -> %
         ++ primitivePart(p) returns the unit normalized form of polynomial p
         ++ divided by the content of p.
  add
    pomopo!(p1,r,e,p2) == p1 + r * mapExponents(x1+->x1+e,p2)
```

```
if R has CommutativeRing then
   binomThmExpt(x,y,nn) ==
           nn = 0 => 1$%
           ans,xn,yn: %
           bincoef: Integer
           powl: List(%):= [x]
           for i in 2..nn repeat powl:=[x * powl.first, :powl]
           yn:=y; ans:=powl.first; i:=1; bincoef:=nn
           for xn in powl.rest repeat
              ans:= bincoef * xn * yn + ans
              bincoef:= (nn-i) * bincoef quo (i+1);  i:= i+1
              -- last I and BINCOEF unused
              yn:= y * yn
           ans + yn
ground? x ==
  retractIfCan(x)@Union(R,"failed") case "failed" => false
  true

ground x == retract(x)@R

mapExponents (fn:E -> E, x: %) ==
     -- this default definition assumes that reductum is cheap
   zero? x => 0
   monomial(leadingCoefficient x,fn degree x)+mapExponents(fn,reductum x)

coefficients x ==
  zero? x => empty()
  concat(leadingCoefficient x, coefficients reductum x)

if R has Field then
   x/r == map(x1+->x1/r,x)

if R has IntegralDomain then
   x exquo r ==
      -- probably not a very good definition in most special cases
      zero? x => 0
      ans:% :=0
      t:=leadingCoefficient x exquo r
      while not (t case "failed") and not zero? x repeat
        ans:=ans+monomial(t::R,degree x)
        x:=reductum x
        if not zero? x then t:=leadingCoefficient x exquo r
      t case "failed" => "failed"
      ans

if R has GcdDomain then
   content x ==       -- this assumes  reductum is cheap
      zero? x => 0
      r:=leadingCoefficient x
      x:=reductum x
```

```
--          while not zero? x and not one? r repeat
          while not zero? x and not (r = 1) repeat
            r:=gcd(r,leadingCoefficient x)
            x:=reductum x
          r

       primitivePart x ==
          zero? x => x
          c := content x
          unitCanonical((x exquo c)::%)
```

———-

— **FAMR.dotabb** —

```
"FAMR"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FAMR"];
"FAMR" -> "AMR"
"FAMR" -> "FRETRCT"
```

———-

— **FAMR.dotfull** —

```
"FiniteAbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FAMR"];
"FiniteAbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)"
"FiniteAbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "FullyRetractableTo(a:Ring)"

"FiniteAbelianMonoidRing(a:Ring,b:OrderedAbelianMonoidSup)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=FAMR"];
"FiniteAbelianMonoidRing(a:Ring,b:OrderedAbelianMonoidSup)" ->
    "FiniteAbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)"
```

———-

— **FAMR.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];
```

```
"FiniteAbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" [color=lightblue];
"FiniteAbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)"
"FiniteAbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "FullyRetractableTo(a:Ring)"

"FullyRetractableTo(a:Ring)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=FRETRCT"];
"FullyRetractableTo(a:Ring)" -> "FullyRetractableTo(a:Type)"

"FullyRetractableTo(a:Type)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FRETRCT"];
"FullyRetractableTo(a:Type)" -> "RETRACT..."

"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" [color=lightblue];
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" -> "RING..."
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "BIMODULE..."
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "IntegralDomain()"
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "CharacteristicNonZero()"
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "CommutativeRing()"
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "Algebra(Fraction(Integer))"

"IntegralDomain()" [color=lightblue];
"IntegralDomain()" -> "CommutativeRing()"
"IntegralDomain()" -> "Algebra(a:CommutativeRing)"
"IntegralDomain()" -> "EntireRing()"

"EntireRing()" [color=lightblue];
"EntireRing()" -> "RING..."
"EntireRing()" -> "BIMODULE..."

"CharacteristicNonZero()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=CHARNZ"];
"CharacteristicNonZero()" -> "RING..."

"Algebra(Fraction(Integer))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=ALGEBRA"];
"Algebra(Fraction(Integer))" -> "Algebra(a:CommutativeRing)"

"Algebra(a:CommutativeRing)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ALGEBRA"];
"Algebra(a:CommutativeRing)" -> "RING..."
"Algebra(a:CommutativeRing)" -> "MODULE..."

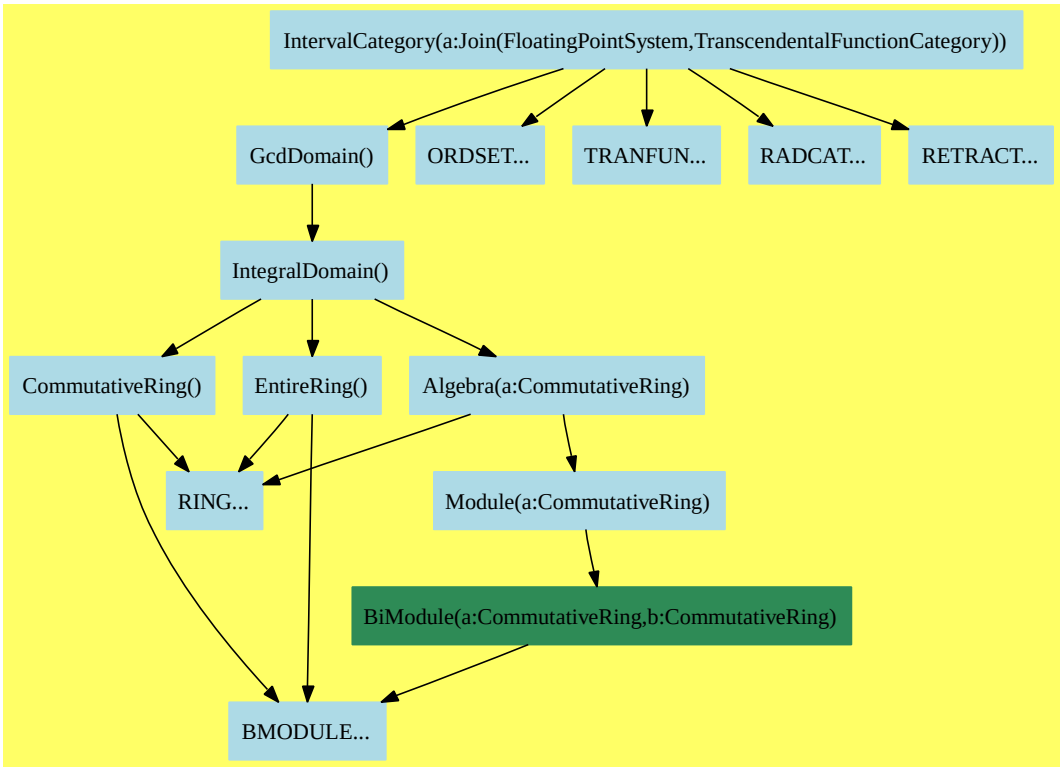
"CommutativeRing()" [color=lightblue];
```

```
"CommutativeRing()" -> "RING..."
"CommutativeRing()" -> "BIMODULE..."

"RETRACT..." [color=lightblue];
"BIMODULE..." [color=lightblue];
"RING..." [color=lightblue];
"MODULE..." [color=lightblue];
}
```

———-

14.0.176 IntervalCategory (INTCAT)



— **IntervalCategory.input** —

```
)set break resume
)sys rm -f IntervalCategory.output
```

```
)spool IntervalCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show IntervalCategory
--R IntervalCategory R: Join(FloatingPointSystem,TranscendentalFunctionCategory)  is a category constructor
--R Abbreviation for IntervalCategory is INTCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for INTCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,Fraction Integer) -> %
--R ?**? : (%,%) -> %                     ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?<? : (%,%) -> Boolean
--R ?<=? : (%,%) -> Boolean               ?=? : (%,%) -> Boolean
--R ?>? : (%,%) -> Boolean                ?>=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        acos : % -> %
--R acosh : % -> %                        acot : % -> %
--R acoth : % -> %                        acsc : % -> %
--R acsch : % -> %                        asec : % -> %
--R asech : % -> %                        asin : % -> %
--R asinh : % -> %                        associates? : (%,%) -> Boolean
--R atan : % -> %                         atanh : % -> %
--R coerce : Integer -> %                 coerce : % -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R contains? : (%,R) -> Boolean          cos : % -> %
--R cosh : % -> %                         cot : % -> %
--R coth : % -> %                         csc : % -> %
--R csch : % -> %                         exp : % -> %
--R gcd : List % -> %                     gcd : (%,%) -> %
--R hash : % -> SingleInteger             inf : % -> R
--R interval : Fraction Integer -> %      interval : R -> %
--R interval : (R,R) -> %                 latex : % -> String
--R lcm : List % -> %                     lcm : (%,%) -> %
--R log : % -> %                          max : (%,%) -> %
--R min : (%,%) -> %                      negative? : % -> Boolean
--R nthRoot : (%,Integer) -> %            one? : % -> Boolean
--R pi : () -> %                          positive? : % -> Boolean
--R qinterval : (R,R) -> %                recip : % -> Union(%,"failed")
--R retract : % -> Integer                sample : () -> %
--R sec : % -> %                          sech : % -> %
--R sin : % -> %                          sinh : % -> %
--R sqrt : % -> %                         sup : % -> R
--R tan : % -> %                          tanh : % -> %
--R unit? : % -> Boolean                  unitCanonical : % -> %
```

```
--R width : % -> R                          zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R exquo : (%,%) -> Union(%,"failed")
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivari
--R retractIfCan : % -> Union(Integer,"failed")
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R
--E 1

)spool
)lisp (bye)
```

———-

— IntervalCategory.help —

```
=====================================================================
IntervalCategory examples
=====================================================================

See Also:
o )show IntervalCategory
```

———-

See:

⇐ “GcdDomain” (GCDDOM) 13.0.173 on page 1129
⇐ “OrderedSet” (ORDSET) 4.0.56 on page 216
⇐ “RadicalCategory” (RADCAT) 2.0.17 on page 52
⇐ “RetractableTo” (RETRACT) 2.0.18 on page 55
⇐ “TranscendentalFunctionCategory” (TRANFUN) 3.0.35 on page 117

Exports:

0	1	acos	acosh	acot
acoth	acsc	acsch	asec	asech
asin	asinh	associates?	atan	atanh
characteristic	coerce	contains?	cos	cosh
cot	coth	csc	csch	exp
exquo	gcd	gcdPolynomial	hash	inf
interval	latex	lcm	log	max
min	negative?	nthRoot	one?	pi
positive?	qinterval	recip	retract	retractIfCan
sample	sec	sech	sin	sinh
sqrt	subtractIfCan	sup	tan	tanh
unit?	unitCanonical	unitNormal	width	zero?
?*?	?**?	?+?	?-?	-?
?<?	?<=?	?=?	?>?	?>=?
?^?	?~=?			

Attributes Exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) - > D$ which is commutative.
- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **approximate** means "is an approximation to the real numbers".

TPDHERE: Note that the signature `coerce : Integer -> %` shows up twice.

These are directly exported but not implemented:

```
contains? : (%,R) -> Boolean
inf : % -> R
interval : (R,R) -> %
interval : R -> %
interval : Fraction Integer -> %
negative? : % -> Boolean
positive? : % -> Boolean
qinterval : (R,R) -> %
sup : % -> R
width : % -> R
```

These exports come from (p1129) GcdDomain():

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean
characteristic : () -> NonNegativeInteger
coerce : % -> %
coerce : Integer -> %
coerce : % -> OutputForm
exquo : (%,%) -> Union(%,"failed")
gcd : (%,%) -> %
gcd : List % -> %
gcdPolynomial :
 (SparseUnivariatePolynomial %,
  SparseUnivariatePolynomial %) ->
    SparseUnivariatePolynomial %
hash : % -> SingleInteger
latex : % -> String
lcm : List % -> %
lcm : (%,%) -> %
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
```

These exports come from (p216) OrderedSet():

```
max : (%,%) -> %
min : (%,%) -> %
?<=? : (%,%) -> Boolean
?>? : (%,%) -> Boolean
?>=? : (%,%) -> Boolean
?<? : (%,%) -> Boolean
```

These exports come from (p117) TranscendentalFunctionCategory():

```
acos : % -> %
acosh : % -> %
acot : % -> %
acoth : % -> %
acsc : % -> %
acsch : % -> %
asec : % -> %
asech : % -> %
asin : % -> %
asinh : % -> %
atan : % -> %
atanh : % -> %
cos : % -> %
cosh : % -> %
cot : % -> %
coth : % -> %
csc : % -> %
csch : % -> %
exp : % -> %
log : % -> %
pi : () -> %
sec : % -> %
sech : % -> %
sin : % -> %
sinh : % -> %
tan : % -> %
tanh : % -> %
?**? : (%,%) -> %
```

These exports come from (p52) RadicalCategory():

```
nthRoot : (%,Integer) -> %
sqrt : % -> %
?**? : (%,Fraction Integer) -> %
```

These exports come from (p55) RetractableTo(Integer):

```
coerce : Integer -> %
retract : % -> Integer
retractIfCan : % -> Union(Integer,"failed")
```

— category INTCAT IntervalCategory —

```
)abbrev category INTCAT IntervalCategory
++ Author: Mike Dewar
++ Date Created: November 1996
++ Date Last Updated:
++ Basic Functions:
```

```
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ This category implements of interval arithmetic and transcendental
++ functions over intervals.

IntervalCategory(R: Join(FloatingPointSystem,TranscendentalFunctionCategory)):
 Category == _
  Join(GcdDomain, OrderedSet, TranscendentalFunctionCategory, _
       RadicalCategory, RetractableTo(Integer)) with
  approximate
  interval : (R,R) -> %
    ++ interval(inf,sup) creates a new interval, either \axiom{[inf,sup]} if
    ++ \axiom{inf <= sup} or \axiom{[sup,in]} otherwise.
  qinterval : (R,R) -> %
    ++ qinterval(inf,sup) creates a new interval \axiom{[inf,sup]}, without
    ++ checking the ordering on the elements.
  interval : R -> %
    ++ interval(f) creates a new interval around f.
  interval : Fraction Integer -> %
    ++ interval(f) creates a new interval around f.
  inf : % -> R
    ++ inf(u) returns the infinum of \axiom{u}.
  sup : % -> R
    ++ sup(u) returns the supremum of \axiom{u}.
  width : % -> R
    ++ width(u) returns \axiom{sup(u) - inf(u)}.
  positive? : % -> Boolean
    ++ positive?(u) returns \axiom{true} if every element of u is positive,
    ++ \axiom{false} otherwise.
  negative? : % -> Boolean
    ++ negative?(u) returns \axiom{true} if every element of u is negative,
    ++ \axiom{false} otherwise.
  contains? : (%,R) -> Boolean
    ++ contains?(i,f) returns true if \axiom{f} is contained within the
    ++ interval \axiom{i}, false otherwise.
```

— INTCAT.dotabb —

```
"INTCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=INTCAT"];
"INTCAT" -> "GCDDOM"
"INTCAT" -> "ORDSET"
```

```
"INTCAT" -> "RADCAT"
"INTCAT" -> "RETRACT"
"INTCAT" -> "TRANFUN"
```

———-

— **INTCAT.dotfull** —

```
"IntervalCategory(a:Join(FloatingPointSystem,TranscendentalFunctionCategory))"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=INTCAT"];
"IntervalCategory(a:Join(FloatingPointSystem,TranscendentalFunctionCategory))"
  -> "GcdDomain()"
"IntervalCategory(a:Join(FloatingPointSystem,TranscendentalFunctionCategory))"
  -> "OrderedSet()"
"IntervalCategory(a:Join(FloatingPointSystem,TranscendentalFunctionCategory))"
  -> "TranscendentalFunctionCategory()"
"IntervalCategory(a:Join(FloatingPointSystem,TranscendentalFunctionCategory))"
  -> "RadicalCategory()"
"IntervalCategory(a:Join(FloatingPointSystem,TranscendentalFunctionCategory))"
  -> "RetractableTo(Integer)"
```

———-

— **INTCAT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"IntervalCategory(a:Join(FloatingPointSystem,TranscendentalFunctionCategory))"
 [color=lightblue];
"IntervalCategory(a:Join(FloatingPointSystem,TranscendentalFunctionCategory))"
  -> "GcdDomain()"
"IntervalCategory(a:Join(FloatingPointSystem,TranscendentalFunctionCategory))"
  -> "ORDSET..."
"IntervalCategory(a:Join(FloatingPointSystem,TranscendentalFunctionCategory))"
  -> "TRANFUN..."
"IntervalCategory(a:Join(FloatingPointSystem,TranscendentalFunctionCategory))"
  -> "RADCAT..."
"IntervalCategory(a:Join(FloatingPointSystem,TranscendentalFunctionCategory))"
  -> "RETRACT..."

"GcdDomain()" [color=lightblue];
"GcdDomain()" -> "IntegralDomain()"
```

```
"IntegralDomain()" [color=lightblue];
"IntegralDomain()" -> "CommutativeRing()"
"IntegralDomain()" -> "Algebra(a:CommutativeRing)"
"IntegralDomain()" -> "EntireRing()"

"CommutativeRing()" [color=lightblue];
"CommutativeRing()" -> "RING..."
"CommutativeRing()" -> "BMODULE..."

"EntireRing()" [color=lightblue];
"EntireRing()" -> "RING..."
"EntireRing()" -> "BMODULE..."

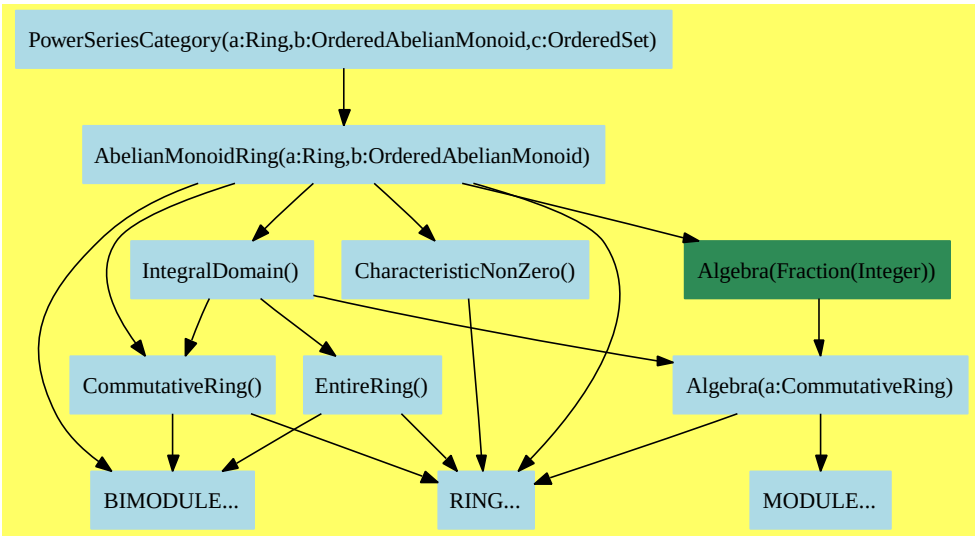
"Algebra(a:CommutativeRing)" [color=lightblue];
"Algebra(a:CommutativeRing)" -> "RING..."
"Algebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"

"Module(a:CommutativeRing)" [color=lightblue];
"Module(a:CommutativeRing)" ->
  "BiModule(a:CommutativeRing,b:CommutativeRing)"

"BiModule(a:CommutativeRing,b:CommutativeRing)" [color=seagreen];
"BiModule(a:CommutativeRing,b:CommutativeRing)" -> "BMODULE..."

"RETRACT..." [color=lightblue];
"RADCAT..." [color=lightblue];
"TRANFUN..." [color=lightblue];
"ORDSET..." [color=lightblue];
"BMODULE..." [color=lightblue];
"RING..." [color=lightblue];
}
```

14.0.177 PowerSeriesCategory (PSCAT)



— PowerSeriesCategory.input —

```
)set break resume
)sys rm -f PowerSeriesCategory.output
)spool PowerSeriesCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show PowerSeriesCategory
--R PowerSeriesCategory(Coef: Ring,Expon: OrderedAbelianMonoid,Var: OrderedSet)  is a category constructor
--R Abbreviation for PowerSeriesCategory is PSCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PSCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (Coef,%) -> %                  ?*? : (%,Coef) -> %
--R ?*? : (%,%) -> %                     ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %       ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                     ?-? : (%,%) -> %
--R -? : % -> %                          ?=? : (%,%) -> Boolean
--R 1 : () -> %                          0 : () -> %
--R ?^? : (%,PositiveInteger) -> %       coefficient : (%,Expon) -> Coef
--R coerce : Integer -> %                coerce : % -> OutputForm
--R complete : % -> %                    degree : % -> Expon
--R hash : % -> SingleInteger            latex : % -> String
```

```
--R leadingCoefficient : % -> Coef        leadingMonomial : % -> %
--R map : ((Coef -> Coef),%) -> %         monomial : (%,Var,Expon) -> %
--R monomial : (Coef,Expon) -> %          monomial? : % -> Boolean
--R one? : % -> Boolean                   pole? : % -> Boolean
--R recip : % -> Union(%,"failed")        reductum : % -> %
--R sample : () -> %                      variables : % -> List Var
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
--R ?*? : (Fraction Integer,%) -> % if Coef has ALGEBRA FRAC INT
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (%,Coef) -> % if Coef has FIELD
--R ?^? : (%,NonNegativeInteger) -> %
--R associates? : (%,%) -> Boolean if Coef has INTDOM
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if Coef has CHARNZ
--R coerce : Coef -> % if Coef has COMRING
--R coerce : % -> % if Coef has INTDOM
--R coerce : Fraction Integer -> % if Coef has ALGEBRA FRAC INT
--R exquo : (%,%) -> Union(%,"failed") if Coef has INTDOM
--R monomial : (%,List Var,List Expon) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unit? : % -> Boolean if Coef has INTDOM
--R unitCanonical : % -> % if Coef has INTDOM
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %) if Coef has INTDOM
--R
--E 1

)spool
)lisp (bye)
```

———-

— PowerSeriesCategory.help —

```
=====================================================================
PowerSeriesCategory examples
=====================================================================

See Also:
o )show PowerSeriesCategory
```

———-

See:

⇒ “MultivariateTaylorSeriesCategory” (MTSCAT) 15.0.182 on page 1193
⇒ “UnivariatePowerSeriesCategory” (UPSCAT) 15.0.184 on page 1210

⇐ "AbelianMonoidRing" (AMR) 13.0.170 on page 1101

Exports:

0	1	associates?	characteristic	charthRoot
coefficient	coerce	complete	degree	exquo
hash	latex	leadingCoefficient	leadingMonomial	map
monomial	monomial?	one?	pole?	recip
reductum	sample	subtractIfCan	variables	unit?
unitCanonical	unitNormal	zero?	?*?	?**?
?+?	?-?	-?	?=?	?~=?
?/?	?^?			

Attributes Exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.
- if #1 has IntegralDomain then noZeroDivisors where **noZeroDivisors** is true if $x*y \neq 0$ implies both x and y are non-zero.
- if #1 has CommutativeRing then commutative("*") where **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.

These are directly exported but not implemented:

```
complete : % -> %
degree : % -> Expon
leadingCoefficient : % -> Coef
leadingMonomial : % -> %
monomial : (%,List Var,List Expon) -> %
monomial : (%,Var,Expon) -> %
pole? : % -> Boolean
variables : % -> List Var
```

These are implemented by this category:

```
?*? : (Integer,%) -> %
?*? : (Coef,%) -> %
?*? : (%,Coef) -> %
?*? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
?*? : (Fraction Integer,%) -> % if Coef has ALGEBRA FRAC INT
?/? : (%,Coef) -> % if Coef has FIELD
-? : % -> %
```

These exports come from (p1101) AbelianMonoidRing(Coef,Expon)
where Coef:Ring and Expon:OrderedAbelianMonoid:

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean if Coef has INTDOM
characteristic : () -> NonNegativeInteger
charthRoot : % -> Union(%,"failed") if Coef has CHARNZ
coefficient : (%,Expon) -> Coef
coerce : Coef -> % if Coef has COMRING
coerce : Fraction Integer -> % if Coef has ALGEBRA FRAC INT
coerce : % -> % if Coef has INTDOM
coerce : % -> OutputForm
coerce : Integer -> %
exquo : (%,%) -> Union(%,"failed") if Coef has INTDOM
hash : % -> SingleInteger
latex : % -> String
map : ((Coef -> Coef),%) -> %
monomial : (Coef,Expon) -> %
monomial? : % -> Boolean
one? : % -> Boolean
recip : % -> Union(%,"failed")
reductum : % -> %
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean if Coef has INTDOM
unitCanonical : % -> % if Coef has INTDOM
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
    if Coef has INTDOM
zero? : % -> Boolean
?**? : (%,NonNegativeInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (%,%) -> %
?-? : (%,%) -> %
?**? : (%,PositiveInteger) -> %
?^? : (%,PositiveInteger) -> %
```

— category PSCAT PowerSeriesCategory —

```
)abbrev category PSCAT PowerSeriesCategory
++ Author: Clifton J. Williamson
++ Date Created: 21 December 1989
++ Date Last Updated: 25 February 1990
++ Basic Operations:
++ Related Domains:
++ Also See:
++ AMS Classifications:
```

```
++ Keywords: power series
++ Examples:
++ References:
++ Description:
++ \spadtype{PowerSeriesCategory} is the most general power series
++ category with exponents in an ordered abelian monoid.

PowerSeriesCategory(Coef,Expon,Var): Category == Definition where
  Coef  : Ring
  Expon : OrderedAbelianMonoid
  Var   : OrderedSet
  I   ==> Integer
  RN  ==> Fraction Integer

  Definition ==> AbelianMonoidRing(Coef,Expon) with
    monomial: (%,Var,Expon) -> %
      ++ \spad{monomial(a,x,n)} computes \spad{a*x**n}.
    monomial: (%,List Var,List Expon) -> %
      ++ \spad{monomial(a,[x1,..,xk],[n1,..,nk])} computes
      ++ \spad{a * x1**n1 * .. * xk**nk}.
    leadingMonomial: % -> %
      ++ leadingMonomial(f) returns the monomial of \spad{f} of lowest order.
    leadingCoefficient: % -> Coef
      ++ leadingCoefficient(f) returns the coefficient of the lowest order
      ++ term of \spad{f}
    degree : % -> Expon
      ++ degree(f) returns the exponent of the lowest order term of \spad{f}.
    variables: % -> List Var
      ++ \spad{variables(f)} returns a list of the variables occuring in the
      ++ power series f.
    pole?: % -> Boolean
      ++ \spad{pole?(f)} determines if the power series f has a pole.
    complete: % -> %
      ++ \spad{complete(f)} causes all terms of f to be computed.
      ++ Note that this results in an infinite loop
      ++ if f has infinitely many terms.

   add
    n:I    * ps:% == (zero? n => 0; map((r1:Coef):Coef +-> n * r1,ps))

    r:Coef * ps:% == (zero? r => 0; map((r1:Coef):Coef +-> r * r1,ps))

    ps:% * r:Coef == (zero? r => 0; map((r1:Coef):Coef +-> r1 * r,ps))

    - ps          == map((r1:Coef):Coef +-> -r1,ps)

    if Coef has Algebra Fraction Integer then
      r:RN * ps:% == (zero? r => 0; map((r1:Coef):Coef +-> r * r1,ps))

      ps:% * r:RN == (zero? r => 0; map((r1:Coef):Coef +-> r1 * r,ps))
```

```
    if Coef has Field then
      ps:% / r:Coef == map((r1:Coef):Coef +-> r1 / r,ps)
```

———-

— **PSCAT.dotabb** —

```
"PSCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PSCAT"];
"PSCAT" -> "AMR"
```

———-

— **PSCAT.dotfull** —

```
"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PSCAT"];
"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)" ->
    "AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)"

"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:SingletonAsOrderedSet)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=PSCAT"];
"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:SingletonAsOrderedSet)"
  -> "PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)"

"PowerSeriesCategory(a:Ring,IndexedExponents(b:OrderedSet),c:OrderedSet))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=PSCAT"];
"PowerSeriesCategory(a:Ring,IndexedExponents(b:OrderedSet),c:OrderedSet))"
  -> "PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)"
```

———-

— **PSCAT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)"
 [color=lightblue];
"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)" ->
    "AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)"
```

```
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" [color=lightblue];
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" -> "RING..."
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "BIMODULE..."
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "IntegralDomain()"
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "CharacteristicNonZero()"
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "CommutativeRing()"
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "Algebra(Fraction(Integer))"

"IntegralDomain()" [color=lightblue];
"IntegralDomain()" -> "CommutativeRing()"
"IntegralDomain()" -> "Algebra(a:CommutativeRing)"
"IntegralDomain()" -> "EntireRing()"

"EntireRing()" [color=lightblue];
"EntireRing()" -> "RING..."
"EntireRing()" -> "BIMODULE..."

"CharacteristicNonZero()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=CHARNZ"];
"CharacteristicNonZero()" -> "RING..."

"Algebra(Fraction(Integer))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=ALGEBRA"];
"Algebra(Fraction(Integer))" -> "Algebra(a:CommutativeRing)"

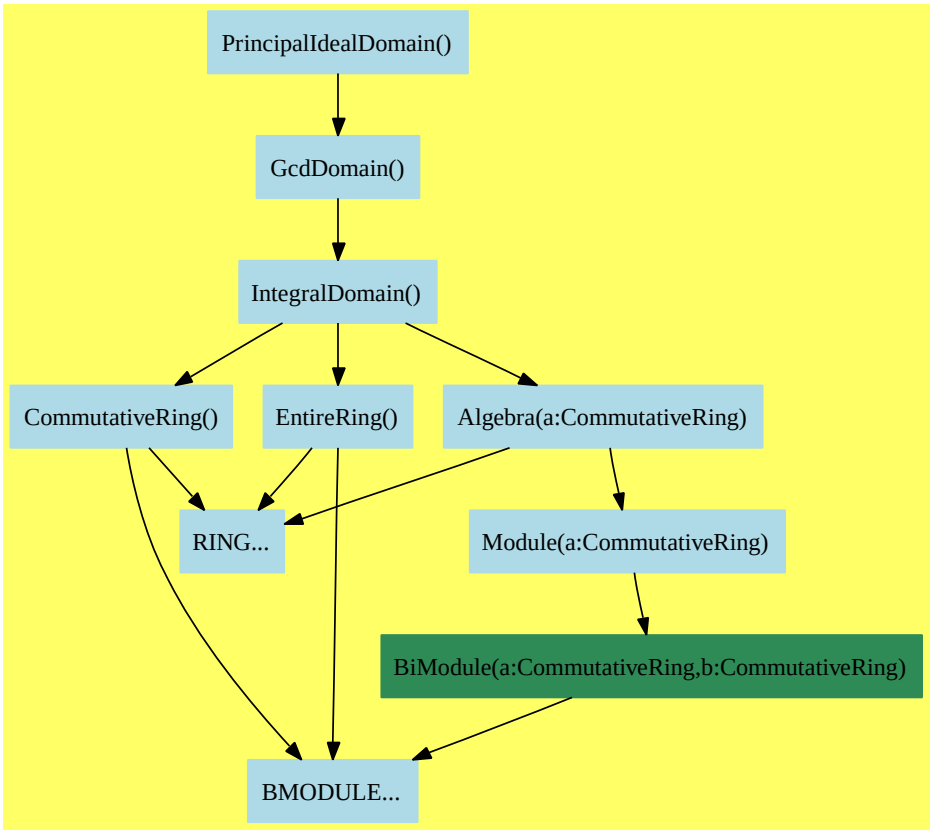
"Algebra(a:CommutativeRing)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ALGEBRA"];
"Algebra(a:CommutativeRing)" -> "RING..."
"Algebra(a:CommutativeRing)" -> "MODULE..."

"CommutativeRing()" [color=lightblue];
"CommutativeRing()" -> "RING..."
"CommutativeRing()" -> "BIMODULE..."

"BIMODULE..." [color=lightblue];
"RING..." [color=lightblue];
"MODULE..." [color=lightblue];
}
```

———-

14.0.178 PrincipalIdealDomain (PID)



— **PrincipalIdealDomain.input** —

```
)set break resume
)sys rm -f PrincipalIdealDomain.output
)spool PrincipalIdealDomain.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show PrincipalIdealDomain
--R PrincipalIdealDomain  is a category constructor
--R Abbreviation for PrincipalIdealDomain is PID
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PID
--R
--R------------------------------ Operations ------------------------------
```

```
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        associates? : (%,%) -> Boolean
--R coerce : % -> %                       coerce : Integer -> %
--R coerce : % -> OutputForm              gcd : List % -> %
--R gcd : (%,%) -> %                      hash : % -> SingleInteger
--R latex : % -> String                   lcm : List % -> %
--R lcm : (%,%) -> %                      one? : % -> Boolean
--R recip : % -> Union(%,"failed")        sample : () -> %
--R unit? : % -> Boolean                  unitCanonical : % -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R expressIdealMember : (List %,%) -> Union(List %,"failed")
--R exquo : (%,%) -> Union(%,"failed")
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivariatePolynomia
--R principalIdeal : List % -> Record(coef: List %,generator: %)
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R
--E 1

)spool
)lisp (bye)
```

———-

— **PrincipalIdealDomain.help** —

```
=====================================================================
PrincipalIdealDomain examples
=====================================================================

See Also:
o )show PrincipalIdealDomain
```

———-

See:

Unique Factorization Domains are a subset of Principal Ideal Domains.
⇐ “UniqueFactorizationDomain” (UFD) 14.0.179 on page 1172 Principal Ideal Domains are a subset of Euclidean Domains.
⇒ “EuclideanDomain” (EUCDOM) 15.0.181 on page 1185

⇐ "GcdDomain" (GCDDOM) 13.0.173 on page 1129

Exports:

0	1	associates?	characteristic	coerce
expressIdealMember	exquo	gcd	gcdPolynomial	hash
latex	lcm	one?	principalIdeal	recip
sample	subtractIfCan	unit?	unitCanonical	unitNormal
zero?	?+?	?-?	-?	?=?
?~=?	?*?	?**?	?^?	

Attributes exported:

- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
expressIdealMember : (List %,%) -> Union(List %,"failed")
principalIdeal : List % -> Record(coef: List %,generator: %)
```

These exports come from (p1129) GcdDomain():

```
associates? : (%,%) -> Boolean
exquo : (%,%) -> Union(%,"failed")
gcd : (%,%) -> %
gcd : List % -> %
gcdPolynomial : (SparseUnivariatePolynomial %,
                 SparseUnivariatePolynomial %) ->
                    SparseUnivariatePolynomial %
lcm : List % -> %
lcm : (%,%) -> %
unit? : % -> Boolean
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : % -> %
coerce : Integer -> %
```

```
 coerce : % -> OutputForm
 hash : % -> SingleInteger
 latex : % -> String
 one? : % -> Boolean
 recip : % -> Union(%,"failed")
 sample : () -> %
 subtractIfCan : (%,%) -> Union(%,"failed")
 zero? : % -> Boolean
 ?+? : (%,%) -> %
 ?=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
 ?*? : (%,%) -> %
 ?*? : (Integer,%) -> %
 ?*? : (PositiveInteger,%) -> %
 ?*? : (NonNegativeInteger,%) -> %
 ?-? : (%,%) -> %
 -? : % -> %
 ?**? : (%,PositiveInteger) -> %
 ?**? : (%,NonNegativeInteger) -> %
 ?^? : (%,PositiveInteger) -> %
 ?^? : (%,NonNegativeInteger) -> %
```

— category PID PrincipalIdealDomain —

```
)abbrev category PID PrincipalIdealDomain
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The category of constructive principal ideal domains, i.e.
++ where a single generator can be constructively found for
++ any ideal given by a finite set of generators.
++ Note that this constructive definition only implies that
++ finitely generated ideals are principal. It is not clear
++ what we would mean by an infinitely generated ideal.

PrincipalIdealDomain(): Category == GcdDomain with
      principalIdeal: List % -> Record(coef:List %,generator:%)
         ++ principalIdeal([f1,...,fn]) returns a record whose
         ++ generator component is a generator of the ideal
         ++ generated by \spad{[f1,...,fn]} whose coef component satisfies
         ++ \spad{generator = sum (input.i * coef.i)}
      expressIdealMember: (List %,%) -> Union(List %,"failed")
```

```
        ++ expressIdealMember([f1,...,fn],h) returns a representation
        ++ of h as a linear combination of the fi or "failed" if h
        ++ is not in the ideal generated by the fi.
```

———-

— **PID.dotabb** —

```
"PID"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PID"];
"PID" -> "GCDDOM"
```

———-

— **PID.dotfull** —

```
"PrincipalIdealDomain()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PID"];
"PrincipalIdealDomain()" -> "GcdDomain()"
```

———-

— **PID.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"PrincipalIdealDomain()" [color=lightblue];
"PrincipalIdealDomain()" -> "GcdDomain()"

"GcdDomain()" [color=lightblue];
"GcdDomain()" -> "IntegralDomain()"

"IntegralDomain()" [color=lightblue];
"IntegralDomain()" -> "CommutativeRing()"
"IntegralDomain()" -> "Algebra(a:CommutativeRing)"
"IntegralDomain()" -> "EntireRing()"

"CommutativeRing()" [color=lightblue];
"CommutativeRing()" -> "RING..."
"CommutativeRing()" -> "BMODULE..."

"EntireRing()" [color=lightblue];
```

```
"EntireRing()" -> "RING..."
"EntireRing()" -> "BMODULE..."

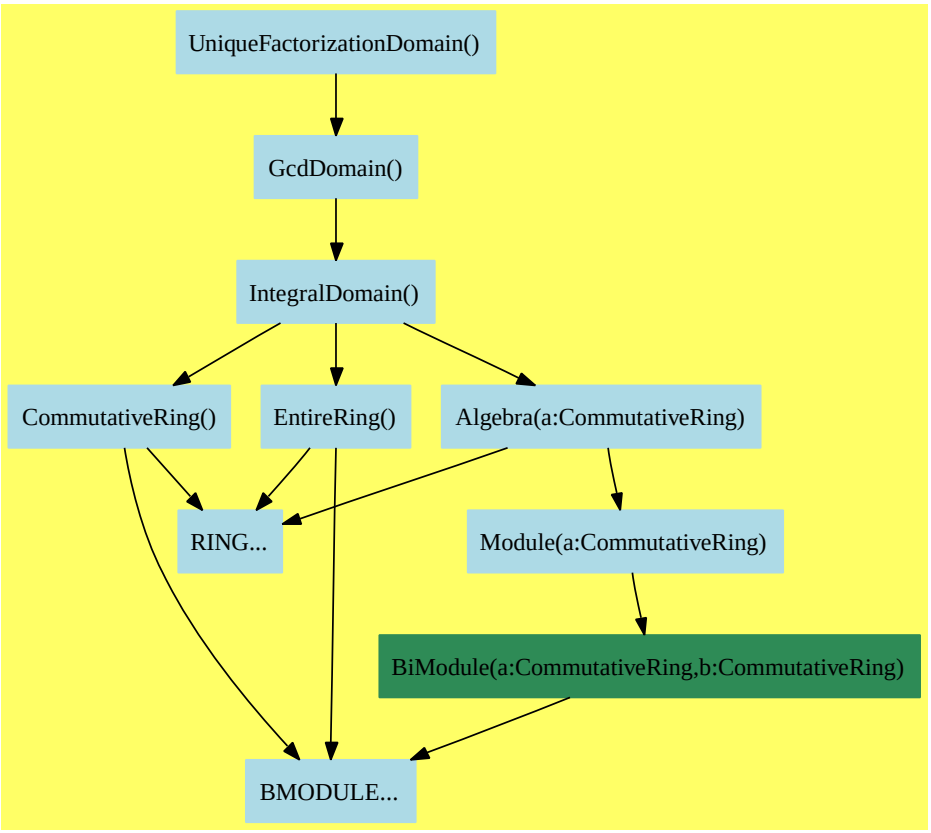
"Algebra(a:CommutativeRing)" [color=lightblue];
"Algebra(a:CommutativeRing)" -> "RING..."
"Algebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"

"Module(a:CommutativeRing)" [color=lightblue];
"Module(a:CommutativeRing)" ->
  "BiModule(a:CommutativeRing,b:CommutativeRing)"

"BiModule(a:CommutativeRing,b:CommutativeRing)" [color=seagreen];
"BiModule(a:CommutativeRing,b:CommutativeRing)" -> "BMODULE..."

"BMODULE..." [color=lightblue];
"RING..." [color=lightblue];
}
```

14.0.179 UniqueFactorizationDomain (UFD)



— UniqueFactorizationDomain.input —

```
)set break resume
)sys rm -f UniqueFactorizationDomain.output
)spool UniqueFactorizationDomain.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show UniqueFactorizationDomain
--R UniqueFactorizationDomain  is a category constructor
--R Abbreviation for UniqueFactorizationDomain is UFD
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for UFD
--R
--R------------------------------ Operations --------------------------------
```

```
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        associates? : (%,%) -> Boolean
--R coerce : % -> %                       coerce : Integer -> %
--R coerce : % -> OutputForm              factor : % -> Factored %
--R gcd : List % -> %                     gcd : (%,%) -> %
--R hash : % -> SingleInteger             latex : % -> String
--R lcm : List % -> %                     lcm : (%,%) -> %
--R one? : % -> Boolean                   prime? : % -> Boolean
--R recip : % -> Union(%,"failed")        sample : () -> %
--R squareFree : % -> Factored %          squareFreePart : % -> %
--R unit? : % -> Boolean                  unitCanonical : % -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R exquo : (%,%) -> Union(%,"failed")
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivariatePolynomia
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R
--E 1

)spool
)lisp (bye)
```

———-

— **UniqueFactorizationDomain.help** —

```
=====================================================================
UniqueFactorizationDomain examples
=====================================================================

See Also:
o )show UniqueFactorizationDomain
```

———-

All Integral Domains are UniqueFactorizationDomains.
⇐ “IntegralDomain” (INTDOM) 12.0.164 on page 1047. Unique Factorization Domains are a subset of Principal Ideal Domains.
⇒ “PrincipalIdealDomain” (PID) 14.0.178 on page 1166

See:

⇒ "Field" (FIELD) 16.0.185 on page 1221
⇒ "IntegerNumberSystem" (INS) 16.0.186 on page 1228
⇒ "PolynomialFactorizationExplicit" (PFECAT) 15.0.183 on page 1202
⇐ "GcdDomain" (GCDDOM) 13.0.173 on page 1129

Exports:

0	1	associates?	characteristic	coerce
exquo	factor	gcd	gcdPolynomial	hash
latex	lcm	one?	prime?	recip
sample	squareFree	squareFreePart	subtractIfCan	unit?
unitCanonical	unitNormal	zero?	?*?	?**?
?+?	?-?	-?	?=?	?~=?
?^?				

Attributes exported:

- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
factor : % -> Factored %
squareFree : % -> Factored %
```

These are implemented by this category:

```
prime? : % -> Boolean
squareFreePart : % -> %
```

These exports come from (p1129) GcdDomain():

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean
characteristic : () -> NonNegativeInteger
coerce : % -> %
coerce : Integer -> %
coerce : % -> OutputForm
```

```
exquo : (%,%) -> Union(%,"failed")
gcd : (%,%) -> %
gcd : List % -> %
gcdPolynomial : (SparseUnivariatePolynomial %,
                 SparseUnivariatePolynomial %) ->
                    SparseUnivariatePolynomial %
hash : % -> SingleInteger
latex : % -> String
lcm : List % -> %
lcm : (%,%) -> %
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
```

— category UFD UniqueFactorizationDomain —

```
)abbrev category UFD UniqueFactorizationDomain
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A constructive unique factorization domain, i.e. where
++ we can constructively factor members into a product of
++ a finite number of irreducible elements.
```

```
UniqueFactorizationDomain(): Category == GcdDomain with
   prime?: % -> Boolean
     ++ prime?(x) tests if x can never be written as the product of two
     ++ non-units of the ring,
     ++ i.e., x is an irreducible element.
   squareFree    : % -> Factored(%)
     ++ squareFree(x) returns the square-free factorization of x
     ++ i.e. such that the factors are pairwise relatively prime
     ++ and each has multiple prime factors.
   squareFreePart: % -> %
     ++ squareFreePart(x) returns a product of prime factors of
     ++ x each taken with multiplicity one.
   factor: % -> Factored(%)
     ++ factor(x) returns the factorization of x into irreducibles.
 add
  squareFreePart x ==
    unit(s := squareFree x) * _*/[f.factor for f in factors s]

  prime? x == # factorList factor x = 1
```

———-

— UFD.dotabb —

```
"UFD"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=UFD"];
"UFD" -> "GCDDOM"
```

———-

— UFD.dotfull —

```
"UniqueFactorizationDomain()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=UFD"];
"UniqueFactorizationDomain()" -> "GcdDomain()"
```

———-

— UFD.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];
```

```
"UniqueFactorizationDomain()" [color=lightblue];
"UniqueFactorizationDomain()" -> "GcdDomain()"

"GcdDomain()" [color=lightblue];
"GcdDomain()" -> "IntegralDomain()"

"IntegralDomain()" [color=lightblue];
"IntegralDomain()" -> "CommutativeRing()"
"IntegralDomain()" -> "Algebra(a:CommutativeRing)"
"IntegralDomain()" -> "EntireRing()"

"CommutativeRing()" [color=lightblue];
"CommutativeRing()" -> "RING..."
"CommutativeRing()" -> "BMODULE..."

"EntireRing()" [color=lightblue];
"EntireRing()" -> "RING..."
"EntireRing()" -> "BMODULE..."

"Algebra(a:CommutativeRing)" [color=lightblue];
"Algebra(a:CommutativeRing)" -> "RING..."
"Algebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"

"Module(a:CommutativeRing)" [color=lightblue];
"Module(a:CommutativeRing)" ->
  "BiModule(a:CommutativeRing,b:CommutativeRing)"

"BiModule(a:CommutativeRing,b:CommutativeRing)" [color=seagreen];
"BiModule(a:CommutativeRing,b:CommutativeRing)" -> "BMODULE..."

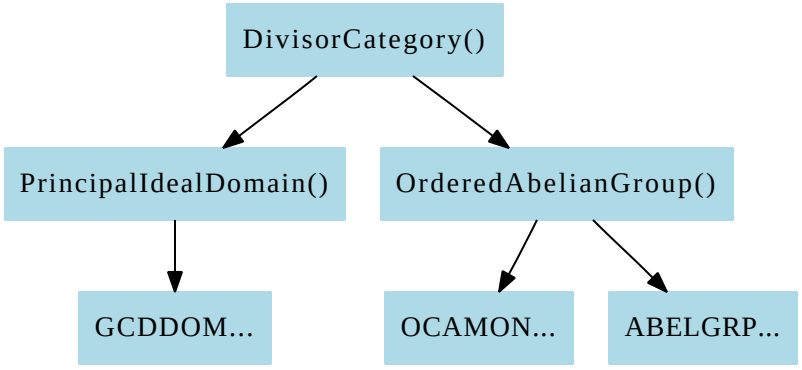
"BMODULE..." [color=lightblue];
"RING..." [color=lightblue];
}
```

———-

Chapter 15

Category Layer 14

15.0.180 DivisorCategory (DIVCAT)



— **DivisorCategory.input** —

```
)set break resume
)sys rm -f DivisorCategory.output
)spool DivisorCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show DivisorCategory
--R DivisorCategory S: SetCategory  is a category constructor
--R Abbreviation for DivisorCategory is DIVCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for DIVCAT
```

```
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (Integer,S) -> %                ?*? : (Integer,%) -> %
--R ?*? : (%,Integer) -> %                ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?+? : (S,%) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?<=? : (%,%) -> Boolean
--R ?=? : (%,%) -> Boolean                0 : () -> %
--R coefficient : (S,%) -> Integer        coerce : S -> %
--R coerce : % -> OutputForm              collect : % -> %
--R concat : (%,%) -> %                   degree : % -> Integer
--R divOfPole : % -> %                    divOfZero : % -> %
--R effective? : % -> Boolean             hash : % -> SingleInteger
--R incr : % -> %                         latex : % -> String
--R mapGen : ((S -> S),%) -> %            nthCoef : (%,Integer) -> Integer
--R nthFactor : (%,Integer) -> S          retract : % -> S
--R sample : () -> %                      size : % -> NonNegativeInteger
--R split : % -> List %                   supp : % -> List S
--R suppOfPole : % -> List S              suppOfZero : % -> List S
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R highCommonTerms : (%,%) -> % if Integer has OAMON
--R mapCoef : ((Integer -> Integer),%) -> %
--R retractIfCan : % -> Union(S,"failed")
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R terms : % -> List Record(gen: S,exp: Integer)
--R
--E 1

)spool
)lisp (bye)
```

———-

— DivisorCategory.help —

```
=====================================================================
DivisorCategory examples
=====================================================================

See Also:
o )show DivisorCategory
```

———-

See:

⇐ “OrderedAbelianGroup” (OAGROUP) 9.0.126 on page 755

⇐ “PrincipalIdealDomain” (PID) 14.0.178 on page 1166

Exports:

0	-?	?*?	?+?
?-?	?<=?	?=?	?~=?
coefficient	coerce	collect	concat
degree	divOfPole	divOfZero	effective?
hash	highCommonTerms	incr	latex
mapCoef	mapGen	nthCoef	nthFactor
retract	retractIfCan	sample	size
split	subtractIfCan	supp	suppOfPole
suppOfZero	terms	zero?	

These operations are exported but not implemented:

```
?<=? : (%,%) -> Boolean
collect : % -> %
concat : (%,%) -> %
degree : % -> Integer
divOfPole : % -> %
divOfZero : % -> %
effective? : % -> Boolean
incr : % -> %
split : % -> List %
supp : % -> List S
suppOfPole : % -> List S
suppOfZero : % -> List S
```

These operations come from (p520) AbelianGroup:

```
0 : () -> %
-? : % -> %
?*? : (Integer,%) -> %
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?+? : (%,%) -> %
?-? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
```

These operations come from (p874) Module(Integer):

```
?*? : (Integer,%) -> %
```

```
 ?*? : (%,Integer) -> %
```

These operations come from (p564) FreeAbelianMonoidCategory(S: SetCategory,E: CancellationAbelianMonoid)

```
 ?*? : (E,S) -> %
 ?+? : (S,%) -> %
 coefficient : (S,%) -> E
 coerce : S -> %
 highCommonTerms : (%,%) -> % if E has OAMON
 mapCoef : ((E -> E),%) -> %
 mapGen : ((S -> S),%) -> %
 nthCoef : (%,Integer) -> E
 nthFactor : (%,Integer) -> S
 retract : % -> S
 retractIfCan : % -> Union(S,"failed")
 size : % -> NonNegativeInteger
 terms : % -> List Record(gen: S,exp: E)
```

— category DIVCAT DivisorCategory —

```
)abbrev category DIVCAT DivisorCategory
++ Authors: Gaetan Hache
++ Date Created: may 1997
++ Date Last Updated: April 2010, by Tim Daly
++ Description: This category exports the function for domains
DivisorCategory(S:SetCategory):Category == Exports where

 INT     ==> Integer
 BOOLEAN ==> Boolean
 LIST    ==> List

 Exports ==> _
  Join(AbelianGroup,Module Integer,FreeAbelianMonoidCategory(S,Integer)) with

    degree: % -> INT
      ++ degree(d) returns the degree of the divisor d

    split: % -> List %
      ++ split(d) splits the divisor d. For example,
      ++ split( 2 p1 + 3p2 ) returns the list [ 2 p1, 3 p2 ].

    "<=" : (%,%) -> BOOLEAN

    collect: % -> %
      ++ collect collects the duplicative points in the divisor.

    concat: (%,%) -> %
```

```
      ++ concat(a,b) concats the divisor a and b
      ++ without collecting the duplicative points.

    effective?: % -> BOOLEAN
      ++ effective?(d) returns true if d >= 0.

    supp: % -> LIST(S)
      ++ supp(d) returns the support of the divisor d.

    suppOfZero: % -> LIST(S)
      ++ suppOfZero(d) returns the elements of the support of d that
      ++ have a positive coefficient.

    suppOfPole: % -> LIST(S)
      ++ suppOfZero(d) returns the elements of the support of d that
      ++ have a negative coefficient.

    divOfZero: % -> %
      ++ divOfZero(d) returns the positive part of d.

    divOfPole: % -> %
      ++ divOfPole(d) returns the negative part of d.

    incr: % -> %
```

———-

— DIVCAT.dotabb —

```
"DIVCAT" [color=lightblue,href="bookvol10.2.pdf#nameddest=DIVCAT"];
"OAGROUP" [color=lightblue,href="bookvol10.2.pdf#nameddest=OAGROUP"];
"PID" [color=lightblue,href="bookvol10.2.pdf#nameddest=PID"];
"DIVCAT" -> "OAGROUP"
"DIVCAT" -> "PID"
```

———-

— DIVCAT.dotfull —

```
"DivisorCategory()" [color=lightblue,href="bookvol10.2.pdf#nameddest=DIVCAT"];
"DivisorCategory()" -> "PrincipalIdealDomain()"
```

———-

— DIVCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"DivisorCategory()" [color=lightblue];
"DivisorCategory()" -> "PrincipalIdealDomain()"
"DivisorCategory()" -> "OrderedAbelianGroup()"

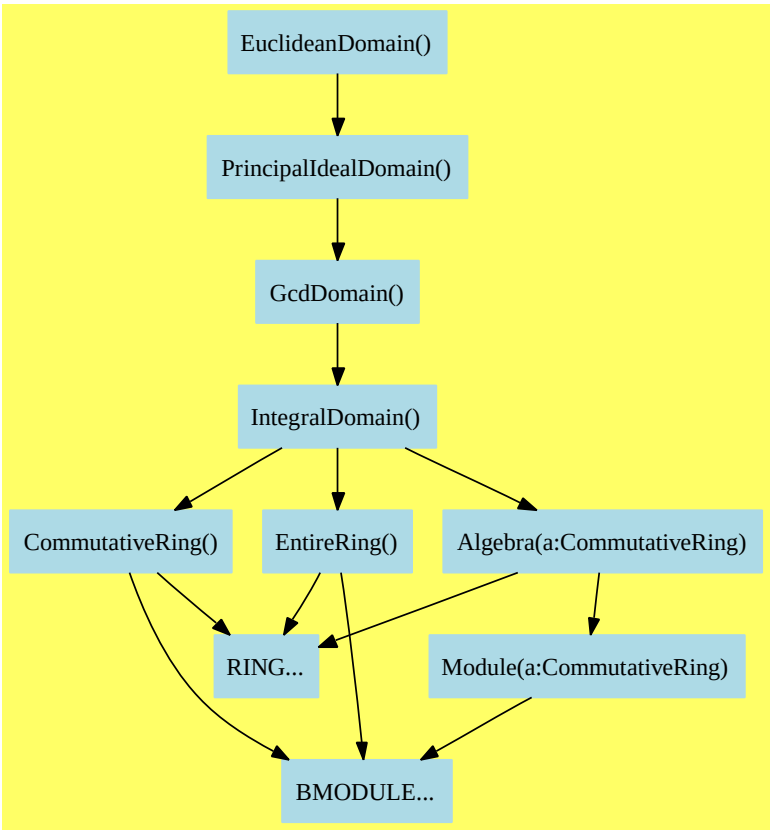
"PrincipalIdealDomain()" [color=lightblue];
"PrincipalIdealDomain()" -> "GCDDOM..."

"OrderedAbelianGroup()" [color=lightblue];
"OrderedAbelianGroup()" -> "OCAMON..."
"OrderedAbelianGroup()" -> "ABELGRP..."

"GCDDOM..." [color=lightblue];
"OCAMON..." [color=lightblue];
"ABELGRP..." [color=lightblue];

}
```

15.0.181 EuclideanDomain (EUCDOM)



— **EuclideanDomain.input** —

```
)set break resume
)sys rm -f EuclideanDomain.output
)spool EuclideanDomain.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show EuclideanDomain
--R EuclideanDomain  is a category constructor
--R Abbreviation for EuclideanDomain is EUCDOM
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for EUCDOM
--R
--R------------------------------ Operations --------------------------------
```

```
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        associates? : (%,%) -> Boolean
--R coerce : % -> %                       coerce : Integer -> %
--R coerce : % -> OutputForm              gcd : List % -> %
--R gcd : (%,%) -> %                      hash : % -> SingleInteger
--R latex : % -> String                   lcm : List % -> %
--R lcm : (%,%) -> %                      one? : % -> Boolean
--R ?quo? : (%,%) -> %                    recip : % -> Union(%,"failed")
--R ?rem? : (%,%) -> %                    sample : () -> %
--R sizeLess? : (%,%) -> Boolean          unit? : % -> Boolean
--R unitCanonical : % -> %                zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R divide : (%,%) -> Record(quotient: %,remainder: %)
--R euclideanSize : % -> NonNegativeInteger
--R expressIdealMember : (List %,%) -> Union(List %,"failed")
--R exquo : (%,%) -> Union(%,"failed")
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivari
--R multiEuclidean : (List %,%) -> Union(List %,"failed")
--R principalIdeal : List % -> Record(coef: List %,generator: %)
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R
--E 1

)spool
)lisp (bye)
```

———-

— EuclideanDomain.help —

```
=====================================================================
EuclideanDomain examples
=====================================================================

See Also:
o )show EuclideanDomain
```

———-

Principal Ideal Domains are a subset of Euclidean Domains.
⇐ "PrincipalIdealDomain" (PID) 14.0.178 on page 1166. Euclidean Domains are a subset of Fields.
⇒ "Field" (FIELD) 16.0.185 on page 1221

See:

⇒ "Field" (FIELD) 16.0.185 on page 1221
⇒ "IntegerNumberSystem" (INS) 16.0.186 on page 1228
⇒ "PAdicIntegerCategory" (PADICCT) 16.0.188 on page 1249
⇒ "QuotientFieldCategory" (QFCAT) 17.0.198 on page 1376
⇐ "PrincipalIdealDomain" (PID) 14.0.178 on page 1166

Exports:

1	0	associates?	characteristic	coerce
divide	euclideanSize	expressIdealMember	exquo	extendedEuclidean
gcd	gcdPolynomial	hash	latex	lcm
multiEuclidean	one?	principalIdeal	recip	sample
sizeLess?	subtractIfCan	unit?	unitCanonical	unitNormal
zero?	?+?	?-?	-?	?=?
?quo?	?rem?	?~=?	?*?	?**?
?^?				

Attributes exported:

- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
divide : (%,%) -> Record(quotient: %,remainder: %)
euclideanSize : % -> NonNegativeInteger
```

These are implemented by this category:

```
expressIdealMember : (List %,%) -> Union(List %,"failed")
exquo : (%,%) -> Union(%,"failed")
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
```

```
gcd : (%,%) -> %
multiEuclidean : (List %,%) -> Union(List %,"failed")
principalIdeal : List % -> Record(coef: List %,generator: %)
sizeLess? : (%,%) -> Boolean
?quo? : (%,%) -> %
?rem? : (%,%) -> %
```

These exports come from (p1166) PrincipalIdealDomain():

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean
characteristic : () -> NonNegativeInteger
coerce : % -> %
coerce : Integer -> %
coerce : % -> OutputForm
gcd : List % -> %
gcdPolynomial : (SparseUnivariatePolynomial %,
                 SparseUnivariatePolynomial %) ->
                    SparseUnivariatePolynomial %
hash : % -> SingleInteger
latex : % -> String
lcm : List % -> %
lcm : (%,%) -> %
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
```

— category EUCDOM EuclideanDomain —

```
)abbrev category EUCDOM EuclideanDomain
```

```
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A constructive euclidean domain, i.e. one can divide producing
++ a quotient and a remainder where the remainder is either zero
++ or is smaller (\spadfun{euclideanSize}) than the divisor.
++
++ Conditional attributes\br
++ \tab{5}multiplicativeValuation\tab{5}Size(a*b)=Size(a)*Size(b)\br
++ \tab{5}additiveValuation\tab{11}Size(a*b)=Size(a)+Size(b)

EuclideanDomain(): Category == PrincipalIdealDomain with
   sizeLess?: (%,%) -> Boolean
     ++ sizeLess?(x,y) tests whether x is strictly
     ++ smaller than y with respect to the
     ++ \spadfunFrom{euclideanSize}{EuclideanDomain}.
   euclideanSize: % -> NonNegativeInteger
     ++ euclideanSize(x) returns the euclidean size of the element x.
     ++ Error: if x is zero.
   divide: (%,%) -> Record(quotient:%,remainder:%)
     ++ divide(x,y) divides x by y producing a record containing a
     ++ \spad{quotient} and \spad{remainder},
     ++ where the remainder is smaller (see
     ++ \spadfunFrom{sizeLess?}{EuclideanDomain}) than the divisor y.
   "quo" : (%,%) -> %
     ++ x quo y is the same as \spad{divide(x,y).quotient}.
     ++ See \spadfunFrom{divide}{EuclideanDomain}.
   "rem": (%,%) -> %
     ++ x rem y is the same as \spad{divide(x,y).remainder}.
     ++ See \spadfunFrom{divide}{EuclideanDomain}.
   extendedEuclidean: (%,%) -> Record(coef1:%,coef2:%,generator:%)
     -- formerly called princIdeal
     ++ extendedEuclidean(x,y) returns a record rec where
     ++ \spad{rec.coef1*x+rec.coef2*y = rec.generator} and
     ++ rec.generator is a gcd of x and y.
     ++ The gcd is unique only
     ++ up to associates if \spadatt{canonicalUnitNormal} is not asserted.
     ++ \spadfun{principalIdeal} provides a version of this operation
     ++ which accepts an arbitrary length list of arguments.
   extendedEuclidean: (%,%,%) -> Union(Record(coef1:%,coef2:%),"failed")
     -- formerly called expressIdealElt
     ++ extendedEuclidean(x,y,z) either returns a record rec
     ++ where \spad{rec.coef1*x+rec.coef2*y=z} or returns "failed"
```

```
    ++ if z cannot be expressed as a linear combination of x and y.
  multiEuclidean: (List %,%) -> Union(List %,"failed")
    ++ multiEuclidean([f1,...,fn],z) returns a list of coefficients
    ++ \spad{[a1, ..., an]} such that
    ++ \spad{ z / prod fi = sum aj/fj}.
    ++ If no such list of coefficients exists, "failed" is returned.
  add
     x,y,z: %
     l: List %
     sizeLess?(x,y) ==
           zero? y => false
           zero? x => true
           euclideanSize(x)<euclideanSize(y)
     x quo y == divide(x,y).quotient --divide must be user-supplied
     x rem y == divide(x,y).remainder
     x exquo y ==
        zero? x => 0
        zero? y => "failed"
        qr:=divide(x,y)
        zero?(qr.remainder) => qr.quotient
        "failed"
     gcd(x,y) ==                --Euclidean Algorithm
        x:=unitCanonical x
        y:=unitCanonical y
        while not zero? y repeat
           (x,y):= (y,x rem y)
           y:=unitCanonical y   -- this doesn't affect the
                                -- correctness of Euclid's algorithm,
                                -- but
                                -- a) may improve performance
                                -- b) ensures gcd(x,y)=gcd(y,x)
                                --    if canonicalUnitNormal
        x
     IdealElt ==> Record(coef1:%,coef2:%,generator:%)
     unitNormalizeIdealElt(s:IdealElt):IdealElt ==
        (u,c,a):=unitNormal(s.generator)
--        one? a => s
        (a = 1) => s
        [a*s.coef1,a*s.coef2,c]$IdealElt
     extendedEuclidean(x,y) ==         --Extended Euclidean Algorithm
        s1:=unitNormalizeIdealElt([1$%,0$%,x]$IdealElt)
        s2:=unitNormalizeIdealElt([0$%,1$%,y]$IdealElt)
        zero? y => s1
        zero? x => s2
        while not zero?(s2.generator) repeat
           qr:= divide(s1.generator, s2.generator)
           s3:=[s1.coef1 - qr.quotient * s2.coef1,
                s1.coef2 - qr.quotient * s2.coef2, qr.remainder]$IdealElt
           s1:=s2
           s2:=unitNormalizeIdealElt s3
```

```
   if not(zero?(s1.coef1)) and not sizeLess?(s1.coef1,y)
     then
        qr:= divide(s1.coef1,y)
        s1.coef1:= qr.remainder
        s1.coef2:= s1.coef2 + qr.quotient * x
        s1 := unitNormalizeIdealElt s1
   s1

TwoCoefs ==> Record(coef1:%,coef2:%)
extendedEuclidean(x,y,z) ==
   zero? z => [0,0]$TwoCoefs
   s:= extendedEuclidean(x,y)
   (w:= z exquo s.generator) case "failed" => "failed"
   zero? y =>
      [s.coef1 * w, s.coef2 * w]$TwoCoefs
   qr:= divide((s.coef1 * w), y)
   [qr.remainder, s.coef2 * w + qr.quotient * x]$TwoCoefs
principalIdeal l ==
   l = [] => error "empty list passed to principalIdeal"
   rest l = [] =>
        uca:=unitNormal(first l)
        [[uca.unit],uca.canonical]
   rest rest l = [] =>
       u:= extendedEuclidean(first l,second l)
       [[u.coef1, u.coef2], u.generator]
   v:=principalIdeal rest l
   u:= extendedEuclidean(first l,v.generator)
   [[u.coef1,:[u.coef2*vv for vv in v.coef]],u.generator]
expressIdealMember(l,z) ==
   z = 0 => [0 for v in l]
   pid := principalIdeal l
   (q := z exquo (pid.generator)) case "failed" => "failed"
   [q*v for v in pid.coef]
multiEuclidean(l,z) ==
   n := #l
   zero? n => error "empty list passed to multiEuclidean"
   n = 1 => [z]
   l1 := copy l
   l2 := split!(l1, n quo 2)
   u:= extendedEuclidean(*/l1, */l2, z)
   u case "failed" => "failed"
   v1 := multiEuclidean(l1,u.coef2)
   v1 case "failed" => "failed"
   v2 := multiEuclidean(l2,u.coef1)
   v2 case "failed" => "failed"
   concat(v1,v2)
```

———-

— **EUCDOM.dotabb** —

```
"EUCDOM"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=EUCDOM"];
"EUCDOM" -> "PID"
```

———-

— **EUCDOM.dotfull** —

```
"EuclideanDomain()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=EUCDOM"];
"EuclideanDomain()" -> "PrincipalIdealDomain()"
```

———-

— **EUCDOM.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"EuclideanDomain()" [color=lightblue];
"EuclideanDomain()" -> "PrincipalIdealDomain()"

"PrincipalIdealDomain()" [color=lightblue];
"PrincipalIdealDomain()" -> "GcdDomain()"

"GcdDomain()" [color=lightblue];
"GcdDomain()" -> "IntegralDomain()"

"IntegralDomain()" [color=lightblue];
"IntegralDomain()" -> "CommutativeRing()"
"IntegralDomain()" -> "Algebra(a:CommutativeRing)"
"IntegralDomain()" -> "EntireRing()"

"CommutativeRing()" [color=lightblue];
"CommutativeRing()" -> "RING..."
"CommutativeRing()" -> "BMODULE..."

"EntireRing()" [color=lightblue];
"EntireRing()" -> "RING..."
"EntireRing()" -> "BMODULE..."
```

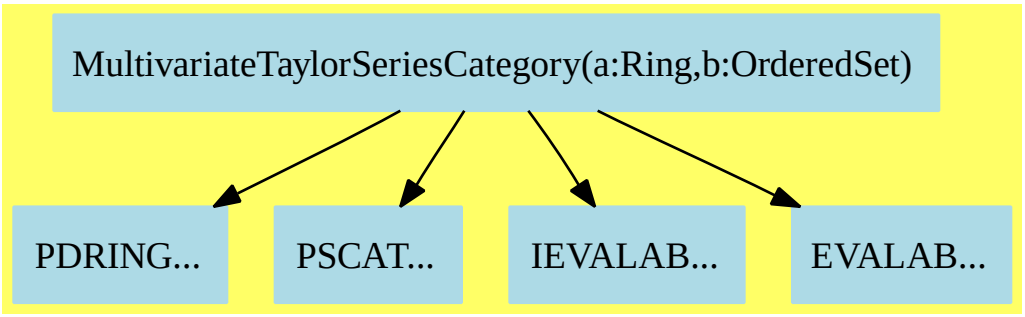
```
"Algebra(a:CommutativeRing)" [color=lightblue];
"Algebra(a:CommutativeRing)" -> "RING..."
"Algebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"

"Module(a:CommutativeRing)" [color=lightblue];
"Module(a:CommutativeRing)" -> "BMODULE..."

"BMODULE..." [color=lightblue];
"RING..." [color=lightblue];
}
```

———-

15.0.182 MultivariateTaylorSeriesCategory (MTSCAT)



— **MultivariateTaylorSeriesCategory.input** —

```
)set break resume
)sys rm -f MultivariateTaylorSeriesCategory.output
)spool MultivariateTaylorSeriesCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show MultivariateTaylorSeriesCategory
--R MultivariateTaylorSeriesCategory(Coef: Ring,Var: OrderedSet)  is a category constructor
--R Abbreviation for MultivariateTaylorSeriesCategory is MTSCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for MTSCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,Coef) -> %                  ?*? : (Coef,%) -> %
```

```
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R D : (%,List Var) -> %                 D : (%,Var) -> %
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        coerce : Integer -> %
--R coerce : % -> OutputForm              complete : % -> %
--R differentiate : (%,Var) -> %          eval : (%,List %,List %) -> %
--R eval : (%,%,%) -> %                   eval : (%,Equation %) -> %
--R eval : (%,List Equation %) -> %       eval : (%,List Var,List %) -> %
--R eval : (%,Var,%) -> %                 hash : % -> SingleInteger
--R latex : % -> String                   leadingCoefficient : % -> Coef
--R leadingMonomial : % -> %              map : ((Coef -> Coef),%) -> %
--R monomial? : % -> Boolean              one? : % -> Boolean
--R pole? : % -> Boolean                  recip : % -> Union(%,"failed")
--R reductum : % -> %                     sample : () -> %
--R variables : % -> List Var            zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (Fraction Integer,%) -> % if Coef has ALGEBRA FRAC INT
--R ?*? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
--R ?**? : (%,%) -> % if Coef has ALGEBRA FRAC INT
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (%,Coef) -> % if Coef has FIELD
--R D : (%,List Var,List NonNegativeInteger) -> %
--R D : (%,Var,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R acos : % -> % if Coef has ALGEBRA FRAC INT
--R acosh : % -> % if Coef has ALGEBRA FRAC INT
--R acot : % -> % if Coef has ALGEBRA FRAC INT
--R acoth : % -> % if Coef has ALGEBRA FRAC INT
--R acsc : % -> % if Coef has ALGEBRA FRAC INT
--R acsch : % -> % if Coef has ALGEBRA FRAC INT
--R asec : % -> % if Coef has ALGEBRA FRAC INT
--R asech : % -> % if Coef has ALGEBRA FRAC INT
--R asin : % -> % if Coef has ALGEBRA FRAC INT
--R asinh : % -> % if Coef has ALGEBRA FRAC INT
--R associates? : (%,%) -> Boolean if Coef has INTDOM
--R atan : % -> % if Coef has ALGEBRA FRAC INT
--R atanh : % -> % if Coef has ALGEBRA FRAC INT
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if Coef has CHARNZ
--R coefficient : (%,List Var,List NonNegativeInteger) -> %
--R coefficient : (%,Var,NonNegativeInteger) -> %
--R coefficient : (%,IndexedExponents Var) -> Coef
--R coerce : Fraction Integer -> % if Coef has ALGEBRA FRAC INT
--R coerce : % -> % if Coef has INTDOM
--R coerce : Coef -> % if Coef has COMRING
```

```
--R cos : % -> % if Coef has ALGEBRA FRAC INT
--R cosh : % -> % if Coef has ALGEBRA FRAC INT
--R cot : % -> % if Coef has ALGEBRA FRAC INT
--R coth : % -> % if Coef has ALGEBRA FRAC INT
--R csc : % -> % if Coef has ALGEBRA FRAC INT
--R csch : % -> % if Coef has ALGEBRA FRAC INT
--R degree : % -> IndexedExponents Var
--R differentiate : (%,List Var,List NonNegativeInteger) -> %
--R differentiate : (%,Var,NonNegativeInteger) -> %
--R differentiate : (%,List Var) -> %
--R exp : % -> % if Coef has ALGEBRA FRAC INT
--R exquo : (%,%) -> Union(%,"failed") if Coef has INTDOM
--R extend : (%,NonNegativeInteger) -> %
--R integrate : (%,Var) -> % if Coef has ALGEBRA FRAC INT
--R log : % -> % if Coef has ALGEBRA FRAC INT
--R monomial : (%,List Var,List NonNegativeInteger) -> %
--R monomial : (%,Var,NonNegativeInteger) -> %
--R monomial : (Coef,IndexedExponents Var) -> %
--R monomial : (%,Var,IndexedExponents Var) -> %
--R monomial : (%,List Var,List IndexedExponents Var) -> %
--R nthRoot : (%,Integer) -> % if Coef has ALGEBRA FRAC INT
--R order : (%,Var,NonNegativeInteger) -> NonNegativeInteger
--R order : (%,Var) -> NonNegativeInteger
--R pi : () -> % if Coef has ALGEBRA FRAC INT
--R polynomial : (%,NonNegativeInteger,NonNegativeInteger) -> Polynomial Coef
--R polynomial : (%,NonNegativeInteger) -> Polynomial Coef
--R sec : % -> % if Coef has ALGEBRA FRAC INT
--R sech : % -> % if Coef has ALGEBRA FRAC INT
--R sin : % -> % if Coef has ALGEBRA FRAC INT
--R sinh : % -> % if Coef has ALGEBRA FRAC INT
--R sqrt : % -> % if Coef has ALGEBRA FRAC INT
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R tan : % -> % if Coef has ALGEBRA FRAC INT
--R tanh : % -> % if Coef has ALGEBRA FRAC INT
--R unit? : % -> Boolean if Coef has INTDOM
--R unitCanonical : % -> % if Coef has INTDOM
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %) if Coef has INTDOM
--R
--E 1

)spool
)lisp (bye)
```

———-

— MultivariateTaylorSeriesCategory.help —

```
=====================================================================
MultivariateTaylorSeriesCategory examples
```

```
=====================================================================

See Also:
o )show MultivariateTaylorSeriesCategory
```

See:

⇐ "Evalable" (EVALAB) 3.0.25 on page 80
⇐ "InnerEvalable" (IEVALAB) 2.0.12 on page 35
⇐ "PartialDifferentialRing" (PDRING) 10.0.145 on page 885
⇐ "PowerSeriesCategory" (PSCAT) 14.0.177 on page 1159

Exports:

0	1	acos	acosh	acot
acoth	acsc	acsch	asec	asech
asin	asinh	associates?	atan	atanh
characteristic	charthRoot	coefficient	coerce	complete
cos	cosh	cot	coth	csc
csch	D	degree	differentiate	eval
exp	exquo	extend	hash	integrate
latex	leadingCoefficient	leadingMonomial	log	map
monomial	monomial?	nthRoot	order	one?
pi	pole?	polynomial	recip	reductum
sample	sec	sech	sin	sinh
sqrt	subtractIfCan	tan	tanh	unit?
unitCanonical	unitNormal	variables	zero?	?*?
?**?	?+?	?-?	-?	?=?
?^?	?~=?	?/?		

Attributes Exported:

- if $ has CommutativeRing then commutative("*") where **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- if $ has IntegralDomain then noZeroDivisors where **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
coefficient : (%,Var,NonNegativeInteger) -> %
coefficient : (%,List Var,List NonNegativeInteger) -> %
extend : (%,NonNegativeInteger) -> %
integrate : (%,Var) -> % if Coef has ALGEBRA FRAC INT
monomial : (%,Var,NonNegativeInteger) -> %
monomial : (%,List Var,List NonNegativeInteger) -> %
order : (%,Var,NonNegativeInteger) -> NonNegativeInteger
order : (%,Var) -> NonNegativeInteger
polynomial : (%,NonNegativeInteger,NonNegativeInteger) -> Polynomial Coef
polynomial : (%,NonNegativeInteger) -> Polynomial Coef
```

These exports come from (p885) PartialDifferentialRing(OrderedSet):

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : Integer -> %
coerce : % -> OutputForm
D : (%,Var) -> %
D : (%,List Var) -> %
D : (%,List Var,List NonNegativeInteger) -> %
D : (%,Var,NonNegativeInteger) -> %
differentiate : (%,Var) -> %
differentiate : (%,List Var,List NonNegativeInteger) -> %
differentiate : (%,Var,NonNegativeInteger) -> %
differentiate : (%,List Var) -> %
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
```

These exports come from (p1159) PowerSeriesCategory(A,B,C)
where A:Ring, B:IndexedExponents(OrderedSet) and C:OrderedSet:

```
associates? : (%,%) -> Boolean if Coef has INTDOM
charthRoot : % -> Union(%,"failed") if Coef has CHARNZ
coefficient : (%,IndexedExponents Var) -> Coef
coerce : Coef -> % if Coef has COMRING
coerce : Fraction Integer -> % if Coef has ALGEBRA FRAC INT
coerce : % -> % if Coef has INTDOM
complete : % -> %
degree : % -> IndexedExponents Var
exquo : (%,%) -> Union(%,"failed") if Coef has INTDOM
leadingCoefficient : % -> Coef
leadingMonomial : % -> %
map : ((Coef -> Coef),%) -> %
monomial : (Coef,IndexedExponents Var) -> %
monomial? : % -> Boolean
monomial : (%,List Var,List IndexedExponents Var) -> %
monomial : (%,Var,IndexedExponents Var) -> %
pole? : % -> Boolean
reductum : % -> %
unit? : % -> Boolean if Coef has INTDOM
unitCanonical : % -> % if Coef has INTDOM
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
  if Coef has INTDOM
variables : % -> List Var
?*? : (%,Coef) -> %
?*? : (Coef,%) -> %
?*? : (Fraction Integer,%) -> % if Coef has ALGEBRA FRAC INT
?*? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
?/? : (%,Coef) -> % if Coef has FIELD
```

These exports come from (p35) InnerEvalable(OrderedSet,

```
eval : (%,Var,%) -> %
eval : (%,List Var,List %) -> %
```

These exports come from (p80) Evalable(

```
eval : (%,List Equation %) -> %
eval : (%,Equation %) -> %
eval : (%,List %,List %) -> %
eval : (%,%,%) -> %
```

These exports come from (p52) RadicalCategory():

```
nthRoot : (%,Integer) -> % if Coef has ALGEBRA FRAC INT
sqrt : % -> % if Coef has ALGEBRA FRAC INT
?**? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
```

These exports come from (p117) TranscendentalFunctionCategory():

```
 acos : % -> % if Coef has ALGEBRA FRAC INT
 acosh : % -> % if Coef has ALGEBRA FRAC INT
 acot : % -> % if Coef has ALGEBRA FRAC INT
 acoth : % -> % if Coef has ALGEBRA FRAC INT
 acsc : % -> % if Coef has ALGEBRA FRAC INT
 acsch : % -> % if Coef has ALGEBRA FRAC INT
 asec : % -> % if Coef has ALGEBRA FRAC INT
 asech : % -> % if Coef has ALGEBRA FRAC INT
 asin : % -> % if Coef has ALGEBRA FRAC INT
 asinh : % -> % if Coef has ALGEBRA FRAC INT
 atan : % -> % if Coef has ALGEBRA FRAC INT
 atanh : % -> % if Coef has ALGEBRA FRAC INT
 cos : % -> % if Coef has ALGEBRA FRAC INT
 cosh : % -> % if Coef has ALGEBRA FRAC INT
 cot : % -> % if Coef has ALGEBRA FRAC INT
 coth : % -> % if Coef has ALGEBRA FRAC INT
 csc : % -> % if Coef has ALGEBRA FRAC INT
 csch : % -> % if Coef has ALGEBRA FRAC INT
 exp : % -> % if Coef has ALGEBRA FRAC INT
 log : % -> % if Coef has ALGEBRA FRAC INT
 pi : () -> % if Coef has ALGEBRA FRAC INT
 sec : % -> % if Coef has ALGEBRA FRAC INT
 sech : % -> % if Coef has ALGEBRA FRAC INT
 sin : % -> % if Coef has ALGEBRA FRAC INT
 sinh : % -> % if Coef has ALGEBRA FRAC INT
 tan : % -> % if Coef has ALGEBRA FRAC INT
 tanh : % -> % if Coef has ALGEBRA FRAC INT
 ?**? : (%,%) -> % if Coef has ALGEBRA FRAC INT
```

— category MTSCAT MultivariateTaylorSeriesCategory —

```
)abbrev category MTSCAT MultivariateTaylorSeriesCategory
++ Author: Clifton J. Williamson
++ Date Created: 6 March 1990
++ Date Last Updated: 6 March 1990
++ Basic Operations:
++ Related Domains:
++ Also See:
++ AMS Classifications:
++ Keywords: multivariate, Taylor, series
++ Examples:
++ References:
++ Description:
++ \spadtype{MultivariateTaylorSeriesCategory} is the most general
++ multivariate Taylor series category.

MultivariateTaylorSeriesCategory(Coef,Var): Category == Definition where
  Coef  : Ring
  Var   : OrderedSet
```

```
L   ==> List
NNI ==> NonNegativeInteger

Definition ==> Join(PartialDifferentialRing Var,_
                    PowerSeriesCategory(Coef,IndexedExponents Var,Var),_
                    InnerEvalable(Var,%),Evalable %) with
  coefficient: (%,Var,NNI) -> %
    ++ \spad{coefficient(f,x,n)} returns the coefficient of \spad{x^n} in f.
  coefficient: (%,L Var,L NNI) -> %
    ++ \spad{coefficient(f,[x1,x2,...,xk],[n1,n2,...,nk])} returns the
    ++ coefficient of \spad{x1^n1 * ... * xk^nk} in f.
  extend: (%,NNI) -> %
    ++ \spad{extend(f,n)} causes all terms of f of degree
    ++ \spad{<= n} to be computed.
  monomial: (%,Var,NNI) -> %
    ++ \spad{monomial(a,x,n)} returns  \spad{a*x^n}.
  monomial: (%,L Var,L NNI) -> %
    ++ \spad{monomial(a,[x1,x2,...,xk],[n1,n2,...,nk])} returns
    ++ \spad{a * x1^n1 * ... * xk^nk}.
  order: (%,Var) -> NNI
    ++ \spad{order(f,x)} returns the order of f viewed as a series in x
    ++ may result in an infinite loop if f has no non-zero terms.
  order: (%,Var,NNI) -> NNI
    ++ \spad{order(f,x,n)} returns \spad{min(n,order(f,x))}.
  polynomial: (%,NNI) -> Polynomial Coef
    ++ \spad{polynomial(f,k)} returns a polynomial consisting of the sum
    ++ of all terms of f of degree \spad{<= k}.
  polynomial: (%,NNI,NNI) -> Polynomial Coef
    ++ \spad{polynomial(f,k1,k2)} returns a polynomial consisting of the
    ++ sum of all terms of f of degree d with \spad{k1 <= d <= k2}.
  if Coef has Algebra Fraction Integer then
    integrate: (%,Var) -> %
      ++ \spad{integrate(f,x)} returns the anti-derivative of the power
      ++ series \spad{f(x)} with respect to the variable x with constant
      ++ coefficient 1.  We may integrate a series when we can divide
      ++ coefficients by integers.
    RadicalCategory
      --++ We provide rational powers when we can divide coefficients
      --++ by integers.
    TranscendentalFunctionCategory
      --++ We provide transcendental functions when we can divide
      --++ coefficients by integers.
```

———-

— MTSCAT.dotabb —

```
"MTSCAT"
```

```
 [color=lightblue,href="bookvol10.2.pdf#nameddest=MTSCAT"];
"MTSCAT" -> "PDRING"
"MTSCAT" -> "PSCAT"
"MTSCAT" -> "IEVALAB"
"MTSCAT" -> "EVALAB"
```

———-

— MTSCAT.dotfull —

```
"MultivariateTaylorSeriesCategory(a:Ring,b:OrderedSet)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=MTSCAT"];
"MultivariateTaylorSeriesCategory(a:Ring,b:OrderedSet)" ->
    "PartialDifferentialRing(a:OrderedSet)"
"MultivariateTaylorSeriesCategory(a:Ring,b:OrderedSet)" ->
   "PowerSeriesCategory(a:Ring,IndexedExponents(b:OrderedSet),c:OrderedSet))"
"MultivariateTaylorSeriesCategory(a:Ring,b:OrderedSet)" ->
 "InnerEvalable(a:Ring,MultivariateTaylorSeriesCategory(a:Ring,b:OrderedSet))"
"MultivariateTaylorSeriesCategory(a:Ring,b:OrderedSet)" ->
 "Evalable(MultivariateTaylorSeriesCategory(a:Ring,b:OrderedSet))"
```

———-

— MTSCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

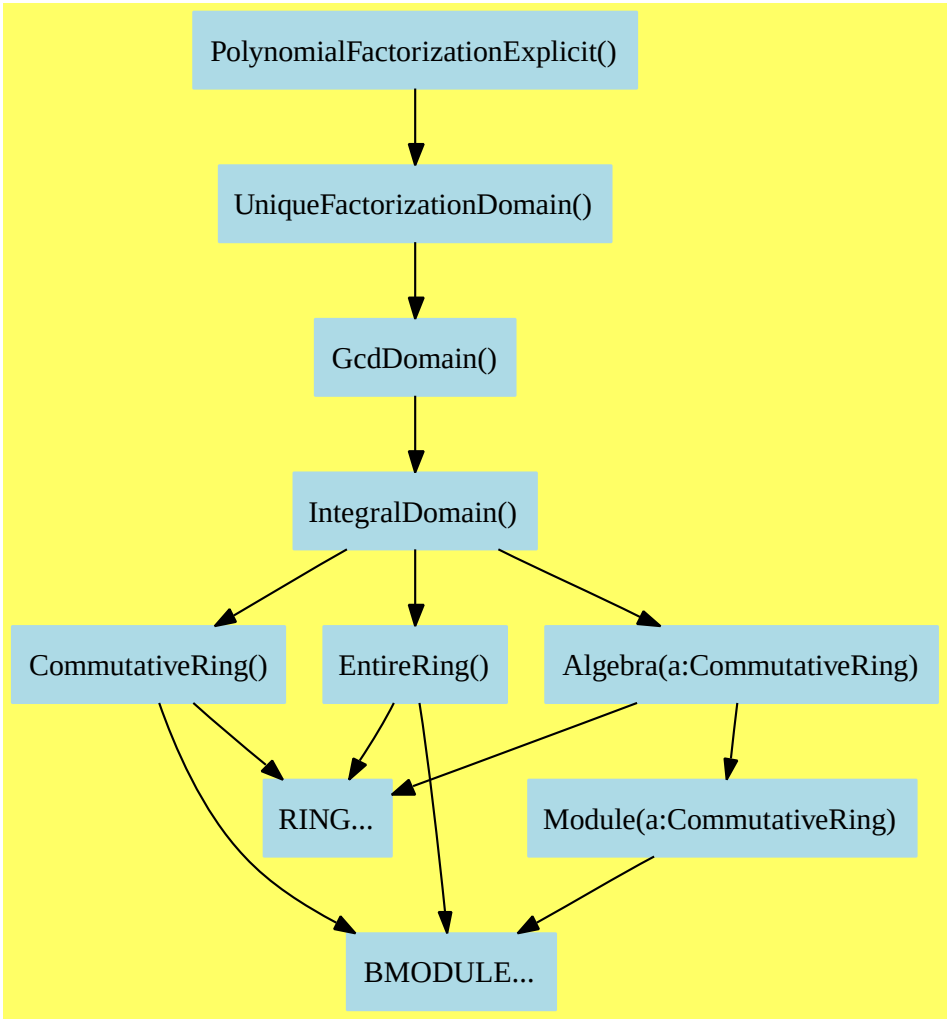
"MultivariateTaylorSeriesCategory(a:Ring,b:OrderedSet)" [color=lightblue];
"MultivariateTaylorSeriesCategory(a:Ring,b:OrderedSet)" ->
    "PDRING..."
"MultivariateTaylorSeriesCategory(a:Ring,b:OrderedSet)" ->
   "PSCAT..."
"MultivariateTaylorSeriesCategory(a:Ring,b:OrderedSet)" ->
 "IEVALAB..."
"MultivariateTaylorSeriesCategory(a:Ring,b:OrderedSet)" ->
 "EVALAB..."

"PDRING..." [color=lightblue];
"PSCAT..." [color=lightblue];
"IEVALAB..." [color=lightblue];
"EVALAB..." [color=lightblue];

}
```

———-

15.0.183 PolynomialFactorizationExplicit (PFECAT)



— PolynomialFactorizationExplicit.input —

```
)set break resume
)sys rm -f PolynomialFactorizationExplicit.output
)spool PolynomialFactorizationExplicit.output
)set message test on
```

```
)set message auto off
)clear all

--S 1 of 1
)show PolynomialFactorizationExplicit
--R PolynomialFactorizationExplicit  is a category constructor
--R Abbreviation for PolynomialFactorizationExplicit is PFECAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PFECAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        associates? : (%,%) -> Boolean
--R coerce : % -> %                       coerce : Integer -> %
--R coerce : % -> OutputForm              factor : % -> Factored %
--R gcd : List % -> %                     gcd : (%,%) -> %
--R hash : % -> SingleInteger             latex : % -> String
--R lcm : List % -> %                     lcm : (%,%) -> %
--R one? : % -> Boolean                   prime? : % -> Boolean
--R recip : % -> Union(%,"failed")        sample : () -> %
--R squareFree : % -> Factored %          squareFreePart : % -> %
--R unit? : % -> Boolean                  unitCanonical : % -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if $ has CHARNZ
--R conditionP : Matrix % -> Union(Vector %,"failed") if $ has CHARNZ
--R exquo : (%,%) -> Union(%,"failed")
--R factorPolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial %
--R factorSquareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial %
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivariatePolynomia
--R solveLinearPolynomialEquation : (List SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> Unio
--R squareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R
--E 1

)spool
)lisp (bye)
```

— **PolynomialFactorizationExplicit.help** —

```
==========================================================================
PolynomialFactorizationExplicit examples
==========================================================================

See Also:
o )show PolynomialFactorizationExplicit
```

———-

See:

⇒ "PolynomialCategory" (POLYCAT) 16.0.189 on page 1255
⇒ "QuotientFieldCategory" (QFCAT) 17.0.198 on page 1376
⇒ "UnivariatePolynomialCategory" (UPOLYC) 17.0.204 on page 1471
⇐ "UniqueFactorizationDomain" (UFD) 14.0.179 on page 1172

Exports:

0	1	associates?
characteristic	charthRoot	coerce
conditionP	exquo	factor
factorPolynomial	factorSquareFreePolynomial	gcd
gcdPolynomial	hash	latex
lcm	one?	prime?
recip	sample	solveLinearPolynomialEquation
squareFree	squareFreePart	squareFreePolynomial
subtractIfCan	unit?	unitNormal
unitCanonical	zero?	?*?
?**?	?+?	?-?
-?	?=?	?^?
?~=?		

Attributes exported:

- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
conditionP : Matrix % -> Union(Vector %,"failed")
    if $ has CHARNZ
factorPolynomial :
  SparseUnivariatePolynomial % ->
     Factored SparseUnivariatePolynomial %
factorSquareFreePolynomial :
  SparseUnivariatePolynomial % ->
     Factored SparseUnivariatePolynomial %
squareFreePolynomial :
  SparseUnivariatePolynomial % ->
     Factored SparseUnivariatePolynomial %
```

These are implemented by this category:

```
charthRoot : % -> Union(%,"failed") if $ has CHARNZ
gcdPolynomial : (SparseUnivariatePolynomial %,
                 SparseUnivariatePolynomial %) ->
                    SparseUnivariatePolynomial %
solveLinearPolynomialEquation :
  (List SparseUnivariatePolynomial %,
   SparseUnivariatePolynomial %) ->
     Union(List SparseUnivariatePolynomial %,"failed")
```

These exports come from (p1172) UniqueFactorizationDomain():

```
factor : % -> Factored %
squareFree : % -> Factored %
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean
characteristic : () -> NonNegativeInteger
coerce : % -> %
coerce : Integer -> %
coerce : % -> OutputForm
exquo : (%,%) -> Union(%,"failed")
gcd : List % -> %
gcd : (%,%) -> %
hash : % -> SingleInteger
latex : % -> String
lcm : List % -> %
lcm : (%,%) -> %
one? : % -> Boolean
prime? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
squareFreePart : % -> %
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
unitCanonical : % -> %
```

```
 zero? : % -> Boolean
 ?+? : (%,%) -> %
 ?=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
 ?*? : (%,%) -> %
 ?*? : (Integer,%) -> %
 ?*? : (PositiveInteger,%) -> %
 ?*? : (NonNegativeInteger,%) -> %
 ?-? : (%,%) -> %
 -? : % -> %
 ?**? : (%,PositiveInteger) -> %
 ?**? : (%,NonNegativeInteger) -> %
 ?^? : (%,PositiveInteger) -> %
 ?^? : (%,NonNegativeInteger) -> %
```

— category PFECAT PolynomialFactorizationExplicit —

```
)abbrev category PFECAT PolynomialFactorizationExplicit
++ Author: James Davenport
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ This is the category of domains that know "enough" about
++ themselves in order to factor univariate polynomials over themselves.
++ This will be used in future releases for supporting factorization
++ over finitely generated coefficient fields, it is not yet available
++ in the current release of axiom.

PolynomialFactorizationExplicit(): Category == Definition where
  P ==> SparseUnivariatePolynomial %
  Definition ==>
   UniqueFactorizationDomain with
     squareFreePolynomial: P -> Factored(P)
       ++ squareFreePolynomial(p) returns the
       ++ square-free factorization of the
       ++ univariate polynomial p.
     factorPolynomial: P -> Factored(P)
       ++ factorPolynomial(p) returns the factorization
       ++ into irreducibles of the univariate polynomial p.
     factorSquareFreePolynomial: P -> Factored(P)
       ++ factorSquareFreePolynomial(p) factors the
       ++ univariate polynomial p into irreducibles
       ++ where p is known to be square free
```

```
   ++ and primitive with respect to its main variable.
 gcdPolynomial: (P, P) -> P
   ++ gcdPolynomial(p,q) returns the gcd of the univariate
   ++ polynomials p qnd q.
   -- defaults to Euclidean, but should be implemented via
   -- modular or p-adic methods.
 solveLinearPolynomialEquation: (List P, P) -> Union(List P,"failed")
   ++ solveLinearPolynomialEquation([f1, ..., fn], g)
   ++ (where the fi are relatively prime to each other)
   ++ returns a list of ai such that
   ++ \spad{g/prod fi = sum ai/fi}
   ++ or returns "failed" if no such list of ai's exists.
 if % has CharacteristicNonZero then
   conditionP: Matrix % -> Union(Vector %,"failed")
     ++ conditionP(m) returns a vector of elements, not all zero,
     ++ whose \spad{p}-th powers (p is the characteristic of the domain)
     ++ are a solution of the homogenous linear system represented
     ++ by m, or "failed" is there is no such vector.
   charthRoot: % -> Union(%,"failed")
     ++ charthRoot(r) returns the \spad{p}-th root of r, or "failed"
     ++ if none exists in the domain.
     -- this is a special case of conditionP, but often the one we want
add
    gcdPolynomial(f,g) ==
       zero? f => g
       zero? g => f
       cf:=content f
       if not one? cf then f:=(f exquo cf)::P
       cg:=content g
       if not one? cg then g:=(g exquo cg)::P
       ans:=subResultantGcd(f,g)$P
       gcd(cf,cg)*(ans exquo content ans)::P
    if % has CharacteristicNonZero then
      charthRoot f ==
         -- to take p'th root of f, solve the system X-fY=0,
         -- so solution is [x,y]
         -- with x^p=X and y^p=Y, then (x/y)^p = f
         zero? f => 0
         m:Matrix % := matrix [[1,-f]]
         ans:= conditionP m
         ans case "failed" => "failed"
         (ans.1) exquo (ans.2)
    if % has Field then
      solveLinearPolynomialEquation(lf,g) ==
        multiEuclidean(lf,g)$P
    else solveLinearPolynomialEquation(lf,g) ==
           LPE ==> LinearPolynomialEquationByFractions %
           solveLinearPolynomialEquationByFractions(lf,g)$LPE
```

———-

— **PFECAT.dotabb** —

```
"PFECAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PFECAT"];
"PFECAT" -> "UFD"
```

———-

— **PFECAT.dotfull** —

```
"PolynomialFactorizationExplicit()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PFECAT"];
"PolynomialFactorizationExplicit()" -> "UniqueFactorizationDomain()"
```

———-

— **PFECAT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"PolynomialFactorizationExplicit()" [color=lightblue];
"PolynomialFactorizationExplicit()" -> "UniqueFactorizationDomain()"

"UniqueFactorizationDomain()" [color=lightblue];
"UniqueFactorizationDomain()" -> "GcdDomain()"

"GcdDomain()" [color=lightblue];
"GcdDomain()" -> "IntegralDomain()"

"IntegralDomain()" [color=lightblue];
"IntegralDomain()" -> "CommutativeRing()"
"IntegralDomain()" -> "Algebra(a:CommutativeRing)"
"IntegralDomain()" -> "EntireRing()"

"CommutativeRing()" [color=lightblue];
"CommutativeRing()" -> "RING..."
"CommutativeRing()" -> "BMODULE..."

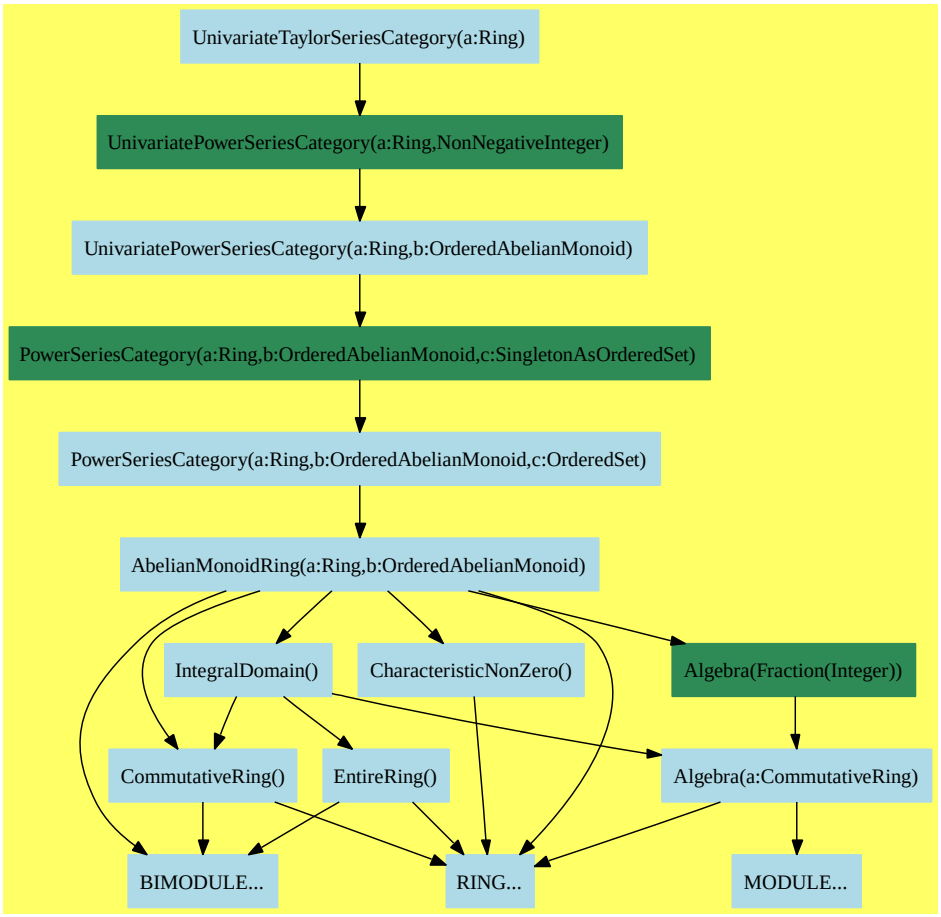
"EntireRing()" [color=lightblue];
"EntireRing()" -> "RING..."
"EntireRing()" -> "BMODULE..."
```

```
"Algebra(a:CommutativeRing)" [color=lightblue];
"Algebra(a:CommutativeRing)" -> "RING..."
"Algebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"

"Module(a:CommutativeRing)" [color=lightblue];
"Module(a:CommutativeRing)" -> "BMODULE..."

"BMODULE..." [color=lightblue];
"RING..." [color=lightblue];
}
```

15.0.184 UnivariatePowerSeriesCategory (UPSCAT)



— UnivariatePowerSeriesCategory.input —

```
)set break resume
)sys rm -f UnivariatePowerSeriesCategory.output
)spool UnivariatePowerSeriesCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show UnivariatePowerSeriesCategory
--R UnivariatePowerSeriesCategory(Coef: Ring,Expon: OrderedAbelianMonoid)  is a category constru
--R Abbreviation for UnivariatePowerSeriesCategory is UPSCAT
--R This constructor is exposed in this frame.
```

```
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for UPSCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (Coef,%) -> %                  ?*? : (%,Coef) -> %
--R ?*? : (%,%) -> %                     ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %       ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                     ?-? : (%,%) -> %
--R -? : % -> %                          ?=? : (%,%) -> Boolean
--R 1 : () -> %                          0 : () -> %
--R ?^? : (%,PositiveInteger) -> %       center : % -> Coef
--R coefficient : (%,Expon) -> Coef      coerce : Integer -> %
--R coerce : % -> OutputForm             complete : % -> %
--R degree : % -> Expon                  ?.? : (%,Expon) -> Coef
--R extend : (%,Expon) -> %              hash : % -> SingleInteger
--R latex : % -> String                  leadingCoefficient : % -> Coef
--R leadingMonomial : % -> %             map : ((Coef -> Coef),%) -> %
--R monomial : (Coef,Expon) -> %         monomial? : % -> Boolean
--R one? : % -> Boolean                  order : (%,Expon) -> Expon
--R order : % -> Expon                   pole? : % -> Boolean
--R recip : % -> Union(%,"failed")       reductum : % -> %
--R sample : () -> %                     truncate : (%,Expon,Expon) -> %
--R truncate : (%,Expon) -> %            variable : % -> Symbol
--R zero? : % -> Boolean                 ?~=? : (%,%) -> Boolean
--R ?*? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
--R ?*? : (Fraction Integer,%) -> % if Coef has ALGEBRA FRAC INT
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (%,Coef) -> % if Coef has FIELD
--R D : % -> % if Coef has *: (Expon,Coef) -> Coef
--R D : (%,NonNegativeInteger) -> % if Coef has *: (Expon,Coef) -> Coef
--R D : (%,Symbol) -> % if Coef has PDRING SYMBOL and Coef has *: (Expon,Coef) -> Coef
--R D : (%,List Symbol) -> % if Coef has PDRING SYMBOL and Coef has *: (Expon,Coef) -> Coef
--R D : (%,Symbol,NonNegativeInteger) -> % if Coef has PDRING SYMBOL and Coef has *: (Expon,Coef) -> Coef
--R D : (%,List Symbol,List NonNegativeInteger) -> % if Coef has PDRING SYMBOL and Coef has *: (Expon,Coef)
--R ?^? : (%,NonNegativeInteger) -> %
--R approximate : (%,Expon) -> Coef if Coef has **: (Coef,Expon) -> Coef and Coef has coerce: Symbol -> Coe
--R associates? : (%,%) -> Boolean if Coef has INTDOM
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if Coef has CHARNZ
--R coerce : Coef -> % if Coef has COMRING
--R coerce : % -> % if Coef has INTDOM
--R coerce : Fraction Integer -> % if Coef has ALGEBRA FRAC INT
--R differentiate : % -> % if Coef has *: (Expon,Coef) -> Coef
--R differentiate : (%,NonNegativeInteger) -> % if Coef has *: (Expon,Coef) -> Coef
--R differentiate : (%,Symbol) -> % if Coef has PDRING SYMBOL and Coef has *: (Expon,Coef) -> Coef
--R differentiate : (%,List Symbol) -> % if Coef has PDRING SYMBOL and Coef has *: (Expon,Coef) -> Coef
--R differentiate : (%,Symbol,NonNegativeInteger) -> % if Coef has PDRING SYMBOL and Coef has *: (Expon,Coe
--R differentiate : (%,List Symbol,List NonNegativeInteger) -> % if Coef has PDRING SYMBOL and Coef has *:
--R ?.? : (%,%) -> % if Expon has SGROUP
--R eval : (%,Coef) -> Stream Coef if Coef has **: (Coef,Expon) -> Coef
```

```
--R exquo : (%,%) -> Union(%,"failed") if Coef has INTDOM
--R monomial : (%,List SingletonAsOrderedSet,List Expon) -> %
--R monomial : (%,SingletonAsOrderedSet,Expon) -> %
--R multiplyExponents : (%,PositiveInteger) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R terms : % -> Stream Record(k: Expon,c: Coef)
--R unit? : % -> Boolean if Coef has INTDOM
--R unitCanonical : % -> % if Coef has INTDOM
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %) if Coef has INTDOM
--R variables : % -> List SingletonAsOrderedSet
--R
--E 1

)spool
)lisp (bye)
```

———-

— UnivariatePowerSeriesCategory.help —

```
=====================================================================
UnivariatePowerSeriesCategory examples
=====================================================================

See Also:
o )show UnivariatePowerSeriesCategory
```

———-

See:

⇒ “LocalPowerSeriesCategory” (LOCPOWC) 16.0.187 on page 1239
⇒ “UnivariateLaurentSeriesCategory” (ULSCAT) 17.0.202 on page 1448
⇒ “UnivariatePuiseuxSeriesCategory” (UPXSCAT) 17.0.203 on page 1460
⇒ “UnivariateTaylorSeriesCategory” (UTSCAT) 16.0.190 on page 1277
⇐ “PowerSeriesCategory” (PSCAT) 14.0.177 on page 1159

Exports:

0	1	approximate	associates?	center
characteristic	charthRoot	coefficient	coerce	complete
D	degree	differentiate	eval	exquo
extend	hash	latex	leadingCoefficient	leadingMonomial
map	monomial	monomial?	multiplyExponents	one?
order	pole?	recip	reductum	sample
subtractIfCan	truncate	terms	unit?	unitCanonical
unitNormal	variable	variables	zero?	?*?
?**?	?+?	?-?	-?	?=?
?^?	?.?	?~=?	?/?	

Attributes Exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.
- if #1 has IntegralDomain then noZeroDivisors where **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- if #1 has CommutativeRing then commutative("*") where **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.

These are directly exported but not implemented:

```
approximate : (%,Expon) -> Coef
    if Coef has **: (Coef,Expon) -> Coef
    and Coef has coerce: Symbol -> Coef
center : % -> Coef
eval : (%,Coef) -> Stream Coef if Coef has **: (Coef,Expon) -> Coef
extend : (%,Expon) -> %
multiplyExponents : (%,PositiveInteger) -> %
order : % -> Expon
order : (%,Expon) -> Expon
terms : % -> Stream Record(k: Expon,c: Coef)
truncate : (%,Expon) -> %
truncate : (%,Expon,Expon) -> %
variable : % -> Symbol
?.? : (%,Expon) -> Coef
```

These are implemented by this category:

```
degree : % -> Expon
leadingCoefficient : % -> Coef
leadingMonomial : % -> %
monomial : (%,SingletonAsOrderedSet,Expon) -> %
reductum : % -> %
variables : % -> List SingletonAsOrderedSet
```

These exports come from (p1159) PowerSeriesCategory(C,E,S)
where C:Ring, E:OrderedAbelianMonoid, S:SingletonAsOrderedSet:

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean if Coef has INTDOM
characteristic : () -> NonNegativeInteger
charthRoot : % -> Union(%,"failed") if Coef has CHARNZ
coefficient : (%,Expon) -> Coef
coerce : Coef -> % if Coef has COMRING
coerce : % -> % if Coef has INTDOM
coerce : Fraction Integer -> % if Coef has ALGEBRA FRAC INT
coerce : Integer -> %
coerce : % -> OutputForm
complete : % -> %
exquo : (%,%) -> Union(%,"failed") if Coef has INTDOM
hash : % -> SingleInteger
latex : % -> String
map : ((Coef -> Coef),%) -> %
monomial : (%,List Var,List Expon) -> %
monomial : (Coef,Expon) -> %
monomial? : % -> Boolean
one? : % -> Boolean
recip : % -> Union(%,"failed")
pole? : % -> Boolean
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean if Coef has INTDOM
unitCanonical : % -> % if Coef has INTDOM
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
    if Coef has INTDOM
zero? : % -> Boolean
?**? : (%,NonNegativeInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (%,%) -> %
?-? : (%,%) -> %
?**? : (%,PositiveInteger) -> %
?^? : (%,PositiveInteger) -> %
?*? : (Integer,%) -> %
?*? : (Coef,%) -> %
?*? : (%,Coef) -> %
?*? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
?*? : (Fraction Integer,%) -> % if Coef has ALGEBRA FRAC INT
?/? : (%,Coef) -> % if Coef has FIELD
```

```
-? : % -> %
```

These exports come from (p29) Eltable(%,%):

```
?.? : (%,%) -> % if Expon has SGROUP
```

These exports come from (p848) DifferentialRing():

```
D : % -> %
    if Coef has *: (Expon,Coef) -> Coef
D : (%,NonNegativeInteger) -> %
    if Coef has *: (Expon,Coef) -> Coef
differentiate : (%,NonNegativeInteger) -> %
    if Coef has *: (Expon,Coef) -> Coef
differentiate : % -> %
    if Coef has *: (Expon,Coef) -> Coef
```

These exports come from (p885) PartialDifferentialRing(Symbol):

```
D : (%,Symbol) -> %
    if Coef has PDRING SYMBOL
    and Coef has *: (Expon,Coef) -> Coef
D : (%,List Symbol) -> %
    if Coef has PDRING SYMBOL
    and Coef has *: (Expon,Coef) -> Coef
D : (%,Symbol,NonNegativeInteger) -> %
    if Coef has PDRING SYMBOL
    and Coef has *: (Expon,Coef) -> Coef
D : (%,List Symbol,List NonNegativeInteger) -> %
    if Coef has PDRING SYMBOL
    and Coef has *: (Expon,Coef) -> Coef
differentiate : (%,List Symbol) -> %
    if Coef has PDRING SYMBOL
    and Coef has *: (Expon,Coef) -> Coef
differentiate : (%,Symbol,NonNegativeInteger) -> %
    if Coef has PDRING SYMBOL
    and Coef has *: (Expon,Coef) -> Coef
differentiate : (%,List Symbol,List NonNegativeInteger) -> %
    if Coef has PDRING SYMBOL
    and Coef has *: (Expon,Coef) -> Coef
differentiate : (%,Symbol) -> %
    if Coef has PDRING SYMBOL
    and Coef has *: (Expon,Coef) -> Coef
```

— category UPSCAT UnivariatePowerSeriesCategory —

```
)abbrev category UPSCAT UnivariatePowerSeriesCategory
++ Author: Clifton J. Williamson
```

```
++ Date Created: 21 December 1989
++ Date Last Updated: 20 September 1993
++ Basic Operations:
++ Related Domains:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ Examples:
++ References:
++ Description:
++ \spadtype{UnivariatePowerSeriesCategory} is the most general
++ univariate power series category with exponents in an ordered
++ abelian monoid.
++ Note that this category exports a substitution function if it is
++ possible to multiply exponents.
++ Also note that this category exports a derivative operation if it is
++ possible to multiply coefficients by exponents.

UnivariatePowerSeriesCategory(Coef,Expon): Category == Definition where
  Coef   : Ring
  Expon  : OrderedAbelianMonoid
  Term ==> Record(k:Expon,c:Coef)

  Definition ==> PowerSeriesCategory(Coef,Expon,SingletonAsOrderedSet) with

    terms: % -> Stream Term
      ++ \spad{terms(f(x))} returns a stream of non-zero terms, where a
      ++ a term is an exponent-coefficient pair.  The terms in the stream
      ++ are ordered by increasing order of exponents.
    --series: Stream Term -> %
      --++ \spad{series(st)} creates a series from a stream of non-zero terms,
      --++ where a term is an exponent-coefficient pair.  The terms in the
      --++ stream should be ordered by increasing order of exponents.
    elt: (%,Expon) -> Coef
      ++ \spad{elt(f(x),r)} returns the coefficient of the term of degree r in
      ++ \spad{f(x)}.  This is the same as the function \spadfun{coefficient}.
    variable: % -> Symbol
      ++ \spad{variable(f)} returns the (unique) power series variable of
      ++ the power series f.
    center: % -> Coef
      ++ \spad{center(f)} returns the point about which the series f is
      ++ expanded.
    multiplyExponents: (%,PositiveInteger) -> %
      ++ \spad{multiplyExponents(f,n)} multiplies all exponents of the power
      ++ series f by the positive integer n.
    order: % -> Expon
      ++ \spad{order(f)} is the degree of the lowest order non-zero term in f.
      ++ This will result in an infinite loop if f has no non-zero terms.
    order: (%,Expon) -> Expon
      ++ \spad{order(f,n) = min(m,n)}, where m is the degree of the
```

```
    ++ lowest order non-zero term in f.
  truncate: (%,Expon) -> %
    ++ \spad{truncate(f,k)} returns a (finite) power series consisting of
    ++ the sum of all terms of f of degree \spad{<= k}.
  truncate: (%,Expon,Expon) -> %
    ++ \spad{truncate(f,k1,k2)} returns a (finite) power
    ++ series consisting of
    ++ the sum of all terms of f of degree d with \spad{k1 <= d <= k2}.
  if Coef has coerce: Symbol -> Coef then
    if Coef has "**":(Coef,Expon) -> Coef then
      approximate: (%,Expon) -> Coef
        ++ \spad{approximate(f)} returns a truncated power series with the
        ++ series variable viewed as an element of the coefficient domain.
  extend: (%,Expon) -> %
    ++ \spad{extend(f,n)} causes all terms of f of degree <= n
    ++ to be computed.
  if Expon has SemiGroup then Eltable(%,%)
  if Coef has "*": (Expon,Coef) -> Coef then
    DifferentialRing
    --!! DifferentialExtension Coef
    if Coef has PartialDifferentialRing Symbol then
      PartialDifferentialRing Symbol
  if Coef has "**": (Coef,Expon) -> Coef then
    eval: (%,Coef) -> Stream Coef
      ++ \spad{eval(f,a)} evaluates a power series at a value in the
      ++ ground ring by returning a stream of partial sums.

add
  degree f == order f

  leadingCoefficient f == coefficient(f,order f)

  leadingMonomial f ==
    ord := order f
    monomial(coefficient(f,ord),ord)

  monomial(f:%,listVar:List SingletonAsOrderedSet,listExpon:List Expon) ==
    empty? listVar or not empty? rest listVar =>
      error "monomial: variable list must have exactly one entry"
    empty? listExpon or not empty? rest listExpon =>
      error "monomial: exponent list must have exactly one entry"
    f * monomial(1,first listExpon)

  monomial(f:%,v:SingletonAsOrderedSet,n:Expon) ==
    f * monomial(1,n)

  reductum f == f - leadingMonomial f

  variables f == list create()
```

———-

— **UPSCAT.dotabb** —

```
"UPSCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=UPSCAT"];
"UPSCAT" -> "PSCAT"
```

———-

— **UPSCAT.dotfull** —

```
"UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=UPSCAT"];
"UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)" ->
 "PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:SingletonAsOrderedSet)"

"UnivariatePowerSeriesCategory(a:Ring,NonNegativeInteger)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=UPSCAT"];
"UnivariatePowerSeriesCategory(a:Ring,NonNegativeInteger)" ->
    "UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)"

"UnivariatePowerSeriesCategory(a:Ring,Integer)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=UPSCAT"];
"UnivariatePowerSeriesCategory(a:Ring,Integer)" ->
    "UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)"

"UnivariatePowerSeriesCategory(a:Ring,Fraction(Integer))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=UPSCAT"];
"UnivariatePowerSeriesCategory(a:Ring,Fraction(Integer))" ->
    "UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)"
```

———-

— **UPSCAT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)"
 [color=lightblue];
"UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)" ->
 "PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:SingletonAsOrderedSet)"
```

```
"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:SingletonAsOrderedSet)"
 [color=seagreen];
"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:SingletonAsOrderedSet)"
  -> "PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)"

"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)"
 [color=lightblue];
"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)" ->
    "AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)"

"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" [color=lightblue];
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" -> "RING..."
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "BIMODULE..."
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "IntegralDomain()"
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "CharacteristicNonZero()"
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "CommutativeRing()"
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "Algebra(Fraction(Integer))"

"IntegralDomain()" [color=lightblue];
"IntegralDomain()" -> "CommutativeRing()"
"IntegralDomain()" -> "Algebra(a:CommutativeRing)"
"IntegralDomain()" -> "EntireRing()"

"EntireRing()" [color=lightblue];
"EntireRing()" -> "RING..."
"EntireRing()" -> "BIMODULE..."

"CharacteristicNonZero()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=CHARNZ"];
"CharacteristicNonZero()" -> "RING..."

"Algebra(Fraction(Integer))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=ALGEBRA"];
"Algebra(Fraction(Integer))" -> "Algebra(a:CommutativeRing)"

"Algebra(a:CommutativeRing)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ALGEBRA"];
"Algebra(a:CommutativeRing)" -> "RING..."
"Algebra(a:CommutativeRing)" -> "MODULE..."

"CommutativeRing()" [color=lightblue];
"CommutativeRing()" -> "RING..."
"CommutativeRing()" -> "BIMODULE..."

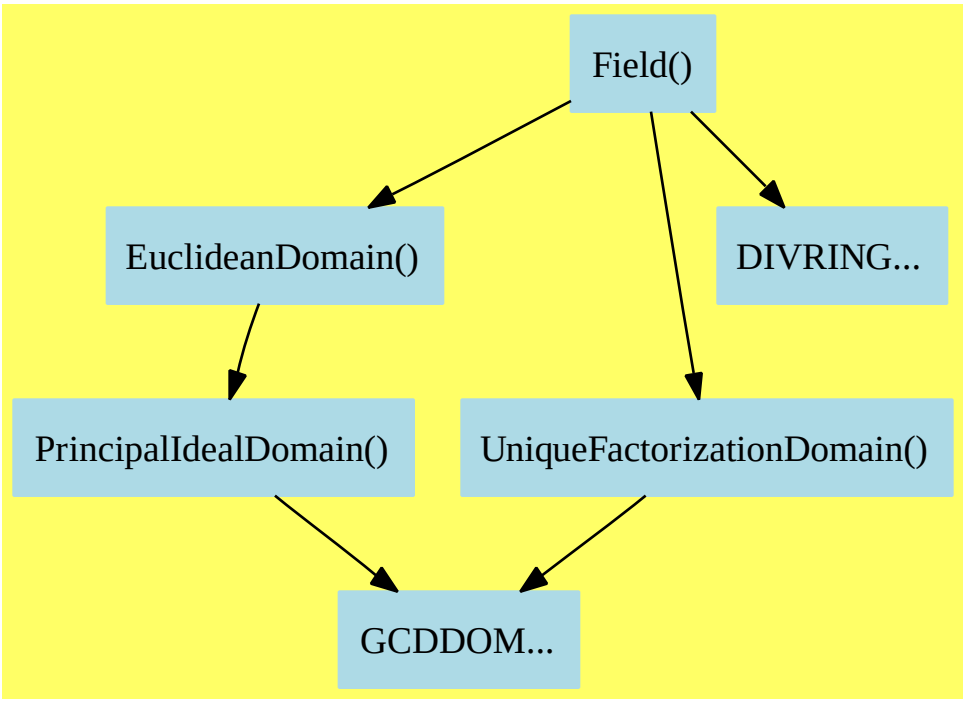
"BIMODULE..." [color=lightblue];
```

```
"RING..." [color=lightblue];
"MODULE..." [color=lightblue];
}
```

Chapter 16

Category Layer 15

16.0.185 Field (FIELD)



— **Field.input** —

```
)set break resume
)sys rm -f Field.output
```

```
)spool Field.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show Field
--R Field  is a category constructor
--R Abbreviation for Field is FIELD
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FIELD
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (Fraction Integer,%) -> %       ?*? : (%,Fraction Integer) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,Integer) -> %
--R ?**? : (%,PositiveInteger) -> %       ?+? : (%,%) -> %
--R ?-? : (%,%) -> %                      -? : % -> %
--R ?/? : (%,%) -> %                      ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,Integer) -> %                ?^? : (%,PositiveInteger) -> %
--R associates? : (%,%) -> Boolean        coerce : Fraction Integer -> %
--R coerce : % -> %                       coerce : Integer -> %
--R coerce : % -> OutputForm              factor : % -> Factored %
--R gcd : List % -> %                     gcd : (%,%) -> %
--R hash : % -> SingleInteger             inv : % -> %
--R latex : % -> String                   lcm : List % -> %
--R lcm : (%,%) -> %                      one? : % -> Boolean
--R prime? : % -> Boolean                 ?quo? : (%,%) -> %
--R recip : % -> Union(%,"failed")        ?rem? : (%,%) -> %
--R sample : () -> %                      sizeLess? : (%,%) -> Boolean
--R squareFree : % -> Factored %          squareFreePart : % -> %
--R unit? : % -> Boolean                  unitCanonical : % -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R divide : (%,%) -> Record(quotient: %,remainder: %)
--R euclideanSize : % -> NonNegativeInteger
--R expressIdealMember : (List %,%) -> Union(List %,"failed")
--R exquo : (%,%) -> Union(%,"failed")
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivari
--R multiEuclidean : (List %,%) -> Union(List %,"failed")
--R principalIdeal : List % -> Record(coef: List %,generator: %)
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R
```

```
--E 1

)spool
)lisp (bye)
```

———-

— **Field.help** —

```
====================================================================
Field examples
====================================================================

See Also:
o )show Field
```

———-

Euclidean Domains are a subset of Fields.
⇐ “EuclideanDomain” (EUCDOM) 15.0.181 on page 1185
⇒ “PseudoAlgebraicClosureOfPerfectFieldCategory” (PACPERC) 17.0.197 on page 1369

See:

⇒ “AlgebraicallyClosedField” (ACF) 17.0.191 on page 1293
⇒ “ExtensionField” (XF) 18.0.206 on page 1510
⇒ “FieldOfPrimeCharacteristic” (FPC) 17.0.193 on page 1323
⇒ “FiniteRankAlgebra” (FINRALG) 17.0.194 on page 1329
⇒ “FunctionSpace” (FS) 17.0.195 on page 1336
⇒ “QuotientFieldCategory” (QFCAT) 17.0.198 on page 1376
⇒ “RealClosedField” (RCFIELD) 17.0.199 on page 1390
⇒ “RealNumberSystem” (RNS) 17.0.200 on page 1400
⇒ “UnivariateLaurentSeriesCategory” (ULSCAT) 17.0.202 on page 1448
⇒ “UnivariatePolynomialCategory” (UPOLYC) 17.0.204 on page 1471
⇒ “UnivariatePuiseuxSeriesCategory” (UPXSCAT) 17.0.203 on page 1460
⇐ “DivisionRing” (DIVRING) 12.0.161 on page 1013
⇐ “EuclideanDomain” (EUCDOM) 15.0.181 on page 1185
⇐ “UniqueFactorizationDomain” (UFD) 14.0.179 on page 1172

Exports:

0	1	associates?	characteristic	coerce
divide	euclideanSize	expressIdealMember	exquo	extendedEuclidean
factor	gcd	gcdPolynomial	hash	inv
latex	lcm	multiEuclidean	one?	prime?
principalIdeal	recip	sample	sizeLess?	squareFree
squareFreePart	subtractIfCan	unit?	unitCanonical	unitNormal
zero?	?*?	?**?	?+?	?-?
-?	?/?	?=?	?^?	?quo?
?rem?	?~=?			

Attributes Exported:

- **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.
- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are implemented by this category:

```
associates? : (%,%) -> Boolean
divide : (%,%) -> Record(quotient: %,remainder: %)
euclideanSize : % -> NonNegativeInteger
exquo : (%,%) -> Union(%,"failed")
factor : % -> Factored %
gcd : (%,%) -> %
inv : % -> %
prime? : % -> Boolean
squareFree : % -> Factored %
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
?/? : (%,%) -> %
```

These exports come from (p1185) EuclideanDomain():

```
0 : () -> %
1 : () -> %
```

```
characteristic : () -> NonNegativeInteger
coerce : % -> %
coerce : Integer -> %
coerce : % -> OutputForm
expressIdealMember : (List %,%) -> Union(List %,"failed")
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
gcd : List % -> %
gcdPolynomial : (SparseUnivariatePolynomial %,
                 SparseUnivariatePolynomial %) ->
                     SparseUnivariatePolynomial %
hash : % -> SingleInteger
latex : % -> String
lcm : List % -> %
lcm : (%,%) -> %
multiEuclidean : (List %,%) -> Union(List %,"failed")
one? : % -> Boolean
principalIdeal : List % -> Record(coef: List %,generator: %)
recip : % -> Union(%,"failed")
sample : () -> %
sizeLess? : (%,%) -> Boolean
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?quo? : (%,%) -> %
?rem? : (%,%) -> %
```

These exports come from (p1172) UniqueFactorizationDomain():

```
squareFreePart : % -> %
```

These exports come from (p1013) DivisionRing():

```
coerce : Fraction Integer -> %
?*? : (Fraction Integer,%) -> %
?*? : (%,Fraction Integer) -> %
?**? : (%,Integer) -> %
```

```
 ?^? : (%,Integer) -> %
```

— category FIELD Field —

```
)abbrev category FIELD Field
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The category of commutative fields, i.e. commutative rings
++ where all non-zero elements have multiplicative inverses.
++ The \spadfun{factor} operation while trivial is useful to have defined.
++
++ Axioms\br
++ \tab{5}\spad{a*(b/a) = b}\br
++ \tab{5}\spad{inv(a) = 1/a}

Field(): Category == Join(EuclideanDomain,UniqueFactorizationDomain,
  DivisionRing) with
      "/": (%,%) -> %
        ++ x/y divides the element x by the element y.
        ++ Error: if y is 0.
      canonicalUnitNormal  ++ either 0 or 1.
      canonicalsClosed     ++ since \spad{0*0=0}, \spad{1*1=1}
    add
      x,y: %
      n: Integer
      UCA ==> Record(unit:%,canonical:%,associate:%)
      unitNormal(x) ==
          if zero? x then [1$%,0$%,1$%]$UCA else [x,1$%,inv(x)]$UCA
      unitCanonical(x) == if zero? x then x else 1
      associates?(x,y) == if zero? x then zero? y else not(zero? y)
      inv x ==((u:=recip x) case "failed" => error "not invertible"; u)
      x exquo y == (y=0 => "failed"; x / y)
      gcd(x,y) == 1
      euclideanSize(x) == 0
      prime? x == false
      squareFree x == x::Factored(%)
      factor x == x::Factored(%)
      x / y == (zero? y => error "catdef: division by zero"; x * inv(y))
      divide(x,y) == [x / y,0]
```

———-

— **FIELD.dotabb** —

```
"FIELD"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FIELD"];
"FIELD" -> "EUCDOM"
"FIELD" -> "UFD"
"FIELD" -> "DIVRING"
```

———-

— **FIELD.dotfull** —

```
"Field()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FIELD"];
"Field()" -> "EuclideanDomain()"
"Field()" -> "UniqueFactorizationDomain()"
"Field()" -> "DivisionRing()"
```

———-

— **FIELD.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"Field()" [color=lightblue];
"Field()" -> "EuclideanDomain()"
"Field()" -> "UniqueFactorizationDomain()"
"Field()" -> "DIVRING..."

"EuclideanDomain()" [color=lightblue];
"EuclideanDomain()" -> "PrincipalIdealDomain()"

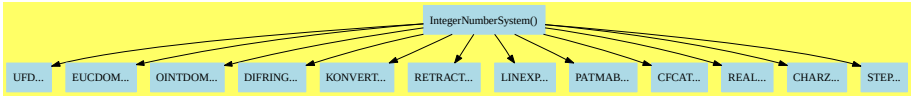
"UniqueFactorizationDomain()" [color=lightblue];
"UniqueFactorizationDomain()" -> "GCDDOM..."

"PrincipalIdealDomain()" [color=lightblue];
"PrincipalIdealDomain()" -> "GCDDOM..."

"DIVRING..." [color=lightblue];
"GCDDOM..." [color=lightblue];
}
```

———-

16.0.186 IntegerNumberSystem (INS)



— IntegerNumberSystem.input —

```
)set break resume
)sys rm -f IntegerNumberSystem.output
)spool IntegerNumberSystem.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show IntegerNumberSystem
--R IntegerNumberSystem  is a category constructor
--R Abbreviation for IntegerNumberSystem is INS
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for INS
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?<? : (%,%) -> Boolean
--R ?<=? : (%,%) -> Boolean               ?=? : (%,%) -> Boolean
--R ?>? : (%,%) -> Boolean                ?>=? : (%,%) -> Boolean
--R D : % -> %                            D : (%,NonNegativeInteger) -> %
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        abs : % -> %
--R addmod : (%,%,%) -> %                 associates? : (%,%) -> Boolean
--R base : () -> %                        binomial : (%,%) -> %
--R bit? : (%,%) -> Boolean               coerce : Integer -> %
--R coerce : % -> %                       coerce : Integer -> %
--R coerce : % -> OutputForm              convert : % -> DoubleFloat
--R convert : % -> Float                  convert : % -> Pattern Integer
--R convert : % -> InputForm              convert : % -> Integer
--R copy : % -> %                         dec : % -> %
--R differentiate : % -> %                even? : % -> Boolean
--R factor : % -> Factored %              factorial : % -> %
--R gcd : List % -> %                     gcd : (%,%) -> %
```

```
--R hash : % -> %                          hash : % -> SingleInteger
--R inc : % -> %                           init : () -> %
--R invmod : (%,%) -> %                    latex : % -> String
--R lcm : List % -> %                      lcm : (%,%) -> %
--R length : % -> %                        mask : % -> %
--R max : (%,%) -> %                       min : (%,%) -> %
--R mulmod : (%,%,%) -> %                  negative? : % -> Boolean
--R odd? : % -> Boolean                    one? : % -> Boolean
--R permutation : (%,%) -> %               positive? : % -> Boolean
--R positiveRemainder : (%,%) -> %         powmod : (%,%,%) -> %
--R prime? : % -> Boolean                  ?quo? : (%,%) -> %
--R random : % -> %                        random : () -> %
--R rational : % -> Fraction Integer       rational? : % -> Boolean
--R recip : % -> Union(%,"failed")         ?rem? : (%,%) -> %
--R retract : % -> Integer                 sample : () -> %
--R shift : (%,%) -> %                     sign : % -> Integer
--R sizeLess? : (%,%) -> Boolean           squareFree : % -> Factored %
--R squareFreePart : % -> %                submod : (%,%,%) -> %
--R symmetricRemainder : (%,%) -> %        unit? : % -> Boolean
--R unitCanonical : % -> %                 zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R differentiate : (%,NonNegativeInteger) -> %
--R divide : (%,%) -> Record(quotient: %,remainder: %)
--R euclideanSize : % -> NonNegativeInteger
--R expressIdealMember : (List %,%) -> Union(List %,"failed")
--R exquo : (%,%) -> Union(%,"failed")
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivariatePolynomia
--R multiEuclidean : (List %,%) -> Union(List %,"failed")
--R nextItem : % -> Union(%,"failed")
--R patternMatch : (%,Pattern Integer,PatternMatchResult(Integer,%)) -> PatternMatchResult(Integer,%)
--R principalIdeal : List % -> Record(coef: List %,generator: %)
--R rationalIfCan : % -> Union(Fraction Integer,"failed")
--R reducedSystem : Matrix % -> Matrix Integer
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer)
--R retractIfCan : % -> Union(Integer,"failed")
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R
--E 1

)spool
)lisp (bye)
```

— IntegerNumberSystem.help —

```
==========================================================================
IntegerNumberSystem examples
==========================================================================

See Also:
o )show IntegerNumberSystem
```

———-

See:

⇐ “CharacteristicZero” (CHARZ) 10.0.136 on page 838
⇐ “CombinatorialFunctionCategory” (CFCAT) 2.0.7 on page 20
⇐ “ConvertibleTo” (KONVERT) 2.0.8 on page 23
⇐ “DifferentialRing” (DIFRING) 10.0.138 on page 848
⇐ “EuclideanDomain” (EUCDOM) 15.0.181 on page 1185
⇐ “LinearlyExplicitRingOver” (LINEXP) 10.0.142 on page 869
⇐ “OrderedIntegralDomain” (OINTDOM) 13.0.174 on page 1135
⇐ “Patternable” (PATAB) 2.0.15 on page 47
⇐ “RealConstant” (REAL) 3.0.32 on page 106
⇐ “RetractableTo” (RETRACT) 2.0.18 on page 55
⇐ “StepThrough” (STEP) 4.0.64 on page 252
⇐ “UniqueFactorizationDomain” (UFD) 14.0.179 on page 1172

Exports:

0	1	abs	addmod
associates?	base	binomial	bit?
characteristic	coerce	convert	copy
D	dec	differentiate	divide
euclideanSize	even?	expressIdealMember	exquo
extendedEuclidean	factor	factorial	gcd
gcdPolynomial	hash	inc	init
invmod	latex	lcm	length
mask	max	min	mulmod
multiEuclidean	negative?	nextItem	odd?
one?	patternMatch	permutation	positive?
positiveRemainder	powmod	prime?	principalIdeal
random	rational	rational?	rationalIfCan
recip	reducedSystem	retract	retractIfCan
sample	shift	sign	sizeLess?
squareFree	squareFreePart	submod	subtractIfCan
symmetricRemainder	unit?	unitCanonical	unitNormal
zero?	?*?	?**?	?+?
?-?	-?	?<?	?<=?
?=?	?>?	?>=?	?^?
?~=?	?quo?	?rem?	

Attributes Exported:

- **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- **multiplicativeValuation** implies `euclideanSize(a*b)=euclideanSize(a)*euclideanSize(b)`.
- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
addmod : (%,%,%) -> %
base : () -> %
dec : % -> %
hash : % -> %
```

```
inc : % -> %
length : % -> %
mulmod : (%,%,%) -> %
odd? : % -> Boolean
positiveRemainder : (%,%) -> %
random : () -> %
random : % -> %
shift : (%,%) -> %
submod : (%,%,%) -> %
```

These are implemented by this category:

```
binomial : (%,%) -> %
bit? : (%,%) -> Boolean
characteristic : () -> NonNegativeInteger
convert : % -> Float
convert : % -> DoubleFloat
convert : % -> InputForm
convert : % -> Pattern Integer
copy : % -> %
differentiate : % -> %
euclideanSize : % -> NonNegativeInteger
even? : % -> Boolean
factor : % -> Factored %
factorial : % -> %
init : () -> %
invmod : (%,%) -> %
mask : % -> %
nextItem : % -> Union(%,"failed")
patternMatch :
 (%,Pattern Integer,PatternMatchResult(Integer,%)) ->
   PatternMatchResult(Integer,%)
permutation : (%,%) -> %
positive? : % -> Boolean
powmod : (%,%,%) -> %
prime? : % -> Boolean
rational : % -> Fraction Integer
rational? : % -> Boolean
rationalIfCan : % -> Union(Fraction Integer,"failed")
retract : % -> Integer
retractIfCan : % -> Union(Integer,"failed")
squareFree : % -> Factored %
symmetricRemainder : (%,%) -> %
```

These exports come from (p1172) UniqueFactorizationDomain():

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean
coerce : % -> %
```

```
coerce : Integer -> %
coerce : Integer -> %
coerce : % -> OutputForm
exquo : (%,%) -> Union(%,"failed")
gcd : List % -> %
gcd : (%,%) -> %
gcdPolynomial :
 (SparseUnivariatePolynomial %,
  SparseUnivariatePolynomial %) ->
    SparseUnivariatePolynomial %
hash : % -> SingleInteger
latex : % -> String
lcm : List % -> %
lcm : (%,%) -> %
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
squareFreePart : % -> %
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
```

These exports come from (p1185) EuclideanDomain():

```
divide : (%,%) -> Record(quotient: %,remainder: %)
expressIdealMember : (List %,%) -> Union(List %,"failed")
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
multiEuclidean : (List %,%) -> Union(List %,"failed")
principalIdeal : List % -> Record(coef: List %,generator: %)
sizeLess? : (%,%) -> Boolean
?quo? : (%,%) -> %
?rem? : (%,%) -> %
```

These exports come from (p1135) OrderedIntegralDomain():

```
abs : % -> %
max : (%,%) -> %
min : (%,%) -> %
negative? : % -> Boolean
sign : % -> Integer
?<? : (%,%) -> Boolean
?<=? : (%,%) -> Boolean
?>? : (%,%) -> Boolean
?>=? : (%,%) -> Boolean
```

These exports come from (p848) DifferentialRing():

```
D : % -> %
D : (%,NonNegativeInteger) -> %
differentiate : (%,NonNegativeInteger) -> %
```

These exports come from (p23) ConvertibleTo(Integer):

```
convert : % -> Integer
```

These exports come from (p869) LinearlyExplicitRingOver(Integer):

```
reducedSystem : Matrix % -> Matrix Integer
reducedSystem : (Matrix %,Vector %) ->
  Record(mat: Matrix Integer,vec: Vector Integer)
```

— category INS IntegerNumberSystem —

```
)abbrev category INS IntegerNumberSystem
++ Author: Stephen M. Watt
++ Date Created: January 1988
++ Change History:
++ Description:
++ An \spad{IntegerNumberSystem} is a model for the integers.

IntegerNumberSystem(): Category ==
  Join(UniqueFactorizationDomain, EuclideanDomain, OrderedIntegralDomain,
         DifferentialRing, ConvertibleTo Integer, RetractableTo Integer,
           LinearlyExplicitRingOver Integer, ConvertibleTo InputForm,
             ConvertibleTo Pattern Integer, PatternMatchable Integer,
               CombinatorialFunctionCategory, RealConstant,
                 CharacteristicZero, StepThrough) with
   odd?     : % -> Boolean
      ++ odd?(n) returns true if and only if n is odd.
   even?    : % -> Boolean
      ++ even?(n) returns true if and only if n is even.
   multiplicativeValuation
      ++ euclideanSize(a*b) returns \spad{euclideanSize(a)*euclideanSize(b)}.
```

```
   base     : () -> %
      ++ base() returns the base for the operations of
      ++ \spad{IntegerNumberSystem}.
   length   : % -> %
      ++ length(a) length of \spad{a} in digits.
   shift    : (%, %) -> %
      ++ shift(a,i) shift \spad{a} by i digits.
   bit?     : (%, %) -> Boolean
      ++ bit?(n,i) returns true if and only if i-th bit of n is a 1.
   positiveRemainder     : (%, %) -> %
      ++ positiveRemainder(a,b) (where \spad{b > 1}) yields r
      ++ where \spad{0 <= r < b} and \spad{r == a rem b}.
   symmetricRemainder     : (%, %) -> %
      ++ symmetricRemainder(a,b) (where \spad{b > 1}) yields r
      ++ where \spad{ -b/2 <= r < b/2 }.
   rational?: % -> Boolean
      ++ rational?(n) tests if n is a rational number
      ++ (see \spadtype{Fraction Integer}).
   rational : % -> Fraction Integer
      ++ rational(n) creates a rational number
      ++ (see \spadtype{Fraction Integer})..
   rationalIfCan: % -> Union(Fraction Integer, "failed")
      ++ rationalIfCan(n) creates a rational number, or returns "failed"
      ++ if this is not possible.
   random   : () -> %
      ++ random() creates a random element.
   random   : % -> %
      ++ random(a) creates a random element from 0 to \spad{n-1}.
   hash     : % -> %
      ++ hash(n) returns the hash code of n.
   copy     : % -> %
      ++ copy(n) gives a copy of n.
   inc      : % -> %
      ++ inc(x) returns \spad{x + 1}.
   dec      : % -> %
      ++ dec(x) returns \spad{x - 1}.
   mask     : % -> %
      ++ mask(n) returns \spad{2**n-1} (an n bit mask).
   addmod   : (%,%,%) -> %
      ++ addmod(a,b,p), \spad{0<=a,b<p>1}, means \spad{a+b mod p}.
   submod   : (%,%,%) -> %
      ++ submod(a,b,p), \spad{0<=a,b<p>1}, means \spad{a-b mod p}.
   mulmod   : (%,%,%) -> %
      ++ mulmod(a,b,p), \spad{0<=a,b<p>1}, means \spad{a*b mod p}.
   powmod   : (%,%,%) -> %
      ++ powmod(a,b,p), \spad{0<=a,b<p>1}, means \spad{a**b mod p}.
   invmod   : (%,%) -> %
      ++ invmod(a,b), \spad{0<=a<b>1}, \spad{(a,b)=1} means \spad{1/a mod b}.
   canonicalUnitNormal
--   commutative("*")    -- follows from the above
```

```
add
  characteristic()         == 0

  differentiate x          == 0

  even? x                  == not odd? x

  positive? x              == x > 0

  copy x                   == x

  bit?(x, i)               == odd? shift(x, -i)

  mask n                   == dec shift(1, n)

  rational? x              == true

  euclideanSize(x)         ==
       x=0 => error "euclideanSize called on zero"
       x<0 => (-convert(x)@Integer)::NonNegativeInteger
       convert(x)@Integer::NonNegativeInteger

  convert(x:%):Float       == (convert(x)@Integer)::Float

  convert(x:%):DoubleFloat  == (convert(x)@Integer)::DoubleFloat

  convert(x:%):InputForm   == convert(convert(x)@Integer)

  retract(x:%):Integer     == convert(x)@Integer

  convert(x:%):Pattern(Integer)== convert(x)@Integer ::Pattern(Integer)

  factor x          == factor(x)$IntegerFactorizationPackage(%)

  squareFree x      == squareFree(x)$IntegerFactorizationPackage(%)

  prime? x          == prime?(x)$IntegerPrimesPackage(%)

  factorial x       == factorial(x)$IntegerCombinatoricFunctions(%)

  binomial(n, m)    == binomial(n, m)$IntegerCombinatoricFunctions(%)

  permutation(n, m) == permutation(n,m)$IntegerCombinatoricFunctions(%)

  retractIfCan(x:%):Union(Integer, "failed") == convert(x)@Integer

  init() == 0

  -- iterates in order 0,1,-1,2,-2,3,-3,...
```

```
    nextItem(n) ==
      zero? n => 1
      n>0 => -n
      1-n

    patternMatch(x, p, l) ==
      patternMatch(x, p, l)$PatternMatchIntegerNumberSystem(%)

    rational(x:%):Fraction(Integer) ==
      (convert(x)@Integer)::Fraction(Integer)

    rationalIfCan(x:%):Union(Fraction Integer, "failed") ==
      (convert(x)@Integer)::Fraction(Integer)

    symmetricRemainder(x, n) ==
       r := x rem n
       r = 0 => r
       if n < 0 then n:=-n
       r > 0 =>
          2 * r > n => r - n
          r
       2*r + n <= 0 => r + n
       r

    invmod(a, b) ==
       if negative? a then a := positiveRemainder(a, b)
       c := a; c1:% := 1
       d := b; d1:% := 0
       while not zero? d repeat
          q := c quo d
          r := c-q*d
          r1 := c1-q*d1
          c := d; c1 := d1
          d := r; d1 := r1
--      not one? c => error "inverse does not exist"
       not (c = 1) => error "inverse does not exist"
       negative? c1 => c1 + b
       c1

    powmod(x, n, p) ==
       if negative? x then x := positiveRemainder(x, p)
       zero? x => 0
       zero? n => 1
       y:% := 1
       z := x
       repeat
          if odd? n then y := mulmod(y, z, p)
          zero?(n := shift(n, -1)) => return y
          z := mulmod(z, z, p)
```

———-

— **INS.dotabb** —

```
"INS"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=INS"];
"INS" -> "UFD"
"INS" -> "EUCDOM"
"INS" -> "OINTDOM"
"INS" -> "DIFRING"
"INS" -> "KONVERT"
"INS" -> "RETRACT"
"INS" -> "LINEXP"
"INS" -> "PATMAB"
"INS" -> "CFCAT"
"INS" -> "REAL"
"INS" -> "CHARZ"
"INS" -> "STEP"
```

———-

— **INS.dotfull** —

```
"IntegerNumberSystem()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=INS"];
"IntegerNumberSystem()" -> "UniqueFactorizationDomain()"
"IntegerNumberSystem()" -> "EuclideanDomain()"
"IntegerNumberSystem()" -> "OrderedIntegralDomain()"
"IntegerNumberSystem()" -> "DifferentialRing()"
"IntegerNumberSystem()" -> "ConvertibleTo(Integer)"
"IntegerNumberSystem()" -> "ConvertibleTo(InputForm)"
"IntegerNumberSystem()" -> "ConvertibleTo(Pattern(Integer))"
"IntegerNumberSystem()" -> "RetractableTo(Integer)"
"IntegerNumberSystem()" -> "LinearlyExplicitRingOver(Integer)"
"IntegerNumberSystem()" -> "PatternMatchable(Integer)"
"IntegerNumberSystem()" -> "CombinatorialFunctionCategory()"
"IntegerNumberSystem()" -> "RealConstant()"
"IntegerNumberSystem()" -> "CharacteristicZero()"
"IntegerNumberSystem()" -> "StepThrough()"
```

———-

— **INS.dotpic** —

```
digraph pic {
 fontsize=10;
```

```
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

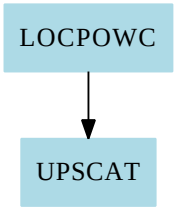
"IntegerNumberSystem()" [color=lightblue];
"IntegerNumberSystem()" -> "UFD..."
"IntegerNumberSystem()" -> "EUCDOM..."
"IntegerNumberSystem()" -> "OINTDOM..."
"IntegerNumberSystem()" -> "DIFRING..."
"IntegerNumberSystem()" -> "KONVERT..."
"IntegerNumberSystem()" -> "RETRACT..."
"IntegerNumberSystem()" -> "LINEXP..."
"IntegerNumberSystem()" -> "PATMAB..."
"IntegerNumberSystem()" -> "CFCAT..."
"IntegerNumberSystem()" -> "REAL..."
"IntegerNumberSystem()" -> "CHARZ..."
"IntegerNumberSystem()" -> "STEP..."

"UFD..." [color=lightblue];
"EUCDOM..." [color=lightblue];
"OINTDOM..." [color=lightblue];
"DIFRING..." [color=lightblue];
"KONVERT..." [color=lightblue];
"RETRACT..." [color=lightblue];
"LINEXP..." [color=lightblue];
"PATMAB..." [color=lightblue];
"CFCAT..." [color=lightblue];
"REAL..." [color=lightblue];
"CHARZ..." [color=lightblue];
"STEP..." [color=lightblue];

}
```

———-

16.0.187 LocalPowerSeriesCategory (LOCPOWC)



— **LocalPowerSeriesCategory.input** —

```
)set break resume
)sys rm -f LocalPowerSeriesCategory.output
)spool LocalPowerSeriesCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show LocalPowerSeriesCategory
--R LocalPowerSeriesCategory K: Field  is a category constructor
--R Abbreviation for LocalPowerSeriesCategory is LOCPOWC
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for LOCPOWC
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,K) -> %                      ?*? : (K,%) -> %
--R ?*? : (Fraction Integer,%) -> %       ?*? : (%,Fraction Integer) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,Integer) -> %
--R ?**? : (%,PositiveInteger) -> %       ?+? : (%,%) -> %
--R ?-? : (%,%) -> %                      -? : % -> %
--R ?/? : (%,%) -> %                      ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,Integer) -> %                ?^? : (%,PositiveInteger) -> %
--R associates? : (%,%) -> Boolean        center : % -> K
--R coefOfFirstNonZeroTerm : % -> K       coefficient : (%,Integer) -> K
--R coerce : Fraction Integer -> %        coerce : % -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R complete : % -> %                     degree : % -> Integer
--R delay : (() -> %) -> %                ?.? : (%,Integer) -> K
--R extend : (%,Integer) -> %             factor : % -> Factored %
--R filterUpTo : (%,Integer) -> %         findCoef : (%,Integer) -> K
--R gcd : List % -> %                     gcd : (%,%) -> %
--R hash : % -> SingleInteger             inv : % -> %
--R latex : % -> String                   lcm : List % -> %
--R lcm : (%,%) -> %                      leadingCoefficient : % -> K
--R leadingMonomial : % -> %              map : ((K -> K),%) -> %
--R monomial : (K,Integer) -> %           monomial? : % -> Boolean
--R one? : % -> Boolean                   order : % -> Integer
--R order : % -> Integer                  order : (%,Integer) -> Integer
--R pole? : % -> Boolean                  posExpnPart : % -> %
--R prime? : % -> Boolean                 printInfo : () -> Boolean
--R printInfo : Boolean -> Boolean        ?quo? : (%,%) -> %
--R recip : % -> Union(%,"failed")        reductum : % -> %
--R ?rem? : (%,%) -> %                    removeFirstZeroes : % -> %
--R removeZeroes : % -> %                 removeZeroes : (Integer,%) -> %
--R sample : () -> %                      sbt : (%,%) -> %
--R series : (Integer,K,%) -> %           shift : (%,Integer) -> %
--R sizeLess? : (%,%) -> Boolean          squareFree : % -> Factored %
--R squareFreePart : % -> %               truncate : (%,Integer) -> %
```

```
--R unit? : % -> Boolean                  unitCanonical : % -> %
--R variable : % -> Symbol                zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (%,K) -> % if K has FIELD
--R D : (%,List Symbol,List NonNegativeInteger) -> % if K has PDRING SYMBOL and K has *: (Integer,K) -> K
--R D : (%,Symbol,NonNegativeInteger) -> % if K has PDRING SYMBOL and K has *: (Integer,K) -> K
--R D : (%,List Symbol) -> % if K has PDRING SYMBOL and K has *: (Integer,K) -> K
--R D : (%,Symbol) -> % if K has PDRING SYMBOL and K has *: (Integer,K) -> K
--R D : (%,NonNegativeInteger) -> % if K has *: (Integer,K) -> K
--R D : % -> % if K has *: (Integer,K) -> K
--R ?^? : (%,NonNegativeInteger) -> %
--R approximate : (%,Integer) -> K if K has **: (K,Integer) -> K and K has coerce: Symbol -> K
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if K has CHARNZ
--R coerce : % -> Stream Record(k: Integer,c: K)
--R coerce : Stream Record(k: Integer,c: K) -> %
--R coerce : K -> % if K has COMRING
--R differentiate : (%,List Symbol,List NonNegativeInteger) -> % if K has PDRING SYMBOL and K has *: (Integ
--R differentiate : (%,Symbol,NonNegativeInteger) -> % if K has PDRING SYMBOL and K has *: (Integer,K) -> K
--R differentiate : (%,List Symbol) -> % if K has PDRING SYMBOL and K has *: (Integer,K) -> K
--R differentiate : (%,Symbol) -> % if K has PDRING SYMBOL and K has *: (Integer,K) -> K
--R differentiate : (%,NonNegativeInteger) -> % if K has *: (Integer,K) -> K
--R differentiate : % -> % if K has *: (Integer,K) -> K
--R divide : (%,%) -> Record(quotient: %,remainder: %)
--R ?.? : (%,%) -> % if Integer has SGROUP
--R euclideanSize : % -> NonNegativeInteger
--R eval : (%,K) -> Stream K if K has **: (K,Integer) -> K
--R expressIdealMember : (List %,%) -> Union(List %,"failed")
--R exquo : (%,%) -> Union(%,"failed")
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivariatePolynomia
--R monomial : (%,SingletonAsOrderedSet,Integer) -> %
--R monomial : (%,List SingletonAsOrderedSet,List Integer) -> %
--R monomial2series : (List %,List NonNegativeInteger,Integer) -> %
--R multiEuclidean : (List %,%) -> Union(List %,"failed")
--R multiplyExponents : (%,PositiveInteger) -> %
--R orderIfNegative : % -> Union(Integer,"failed")
--R principalIdeal : List % -> Record(coef: List %,generator: %)
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R terms : % -> Stream Record(k: Integer,c: K)
--R truncate : (%,Integer,Integer) -> %
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R variables : % -> List SingletonAsOrderedSet
--R
--E 1

)spool
```

```
)lisp (bye)
```

———-

— **LocalPowerSeriesCategory.help** —

```
====================================================================
LocalPowerSeriesCategory examples
====================================================================

See Also:
o )show LocalPowerSeriesCategory
```

———-

See:

⇐ “CharacteristicZero” (CHARZ) 10.0.136 on page 838
⇐ “EuclideanDomain” (EUCDOM) 15.0.181 on page 1185
⇐ “UnivariatePowerSeriesCategory” (UPSCAT) 15.0.184 on page 1210

Exports:

0	1	-?
?**?	?+?	?-?
?.?	?/?	?=?
?^?	?~=?	?quo?
?rem?	D	approximate
associates?	center	characteristic
charthRoot	coefOfFirstNonZeroTerm	coefficient
coerce	complete	degree
delay	differentiate	divide
euclideanSize	eval	expressIdealMember
exquo	extend	extendedEuclidean
factor	filterUpTo	findCoef
gcd	gcdPolynomial	hash
inv	latex	lcm
leadingCoefficient	leadingMonomial	map
monomial	monomial2series	monomial?
multiEuclidean	multiplyExponents	one?
order	orderIfNegative	pole?
posExpnPart	prime?	principalIdeal
printInfo	recip	reductum
removeFirstZeroes	removeZeroes	sample
sbt	series	shift
sizeLess?	squareFree	squareFreePart
subtractIfCan	terms	truncate
unit?	unitCanonical	unitNormal
variable	variables	zero?

Attributes Exported:

- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) - > D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.
- **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.

These are directly exported but not implemented

```
coefOfFirstNonZeroTerm : % -> K
coerce : % -> Stream Record(k: Integer,c: K)
coerce : Stream Record(k: Integer,c: K) -> %
delay : (() -> %) -> %
filterUpTo : (%,Integer) -> %
findCoef : (%,Integer) -> K
monomial2series : (List %,List NonNegativeInteger,Integer) -> %
order : % -> Integer
orderIfNegative : % -> Union(Integer,"failed")
posExpnPart : % -> %
printInfo : () -> Boolean
printInfo : Boolean -> Boolean
removeFirstZeroes : % -> %
removeZeroes : % -> %
removeZeroes : (Integer,%) -> %
sbt : (%,%) -> %
series : (Integer,K,%) -> %
shift : (%,Integer) -> %
```

These exports come from (p1210) UnivariatePowerSeriesCategory():

```
-? : % -> %
0 : () -> %
1 : () -> %
?**? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
?*? : (%,%) -> %
?*? : (%,Fraction Integer) -> %
?*? : (Fraction Integer,%) -> %
?*? : (Integer,%) -> %
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?+? : (%,%) -> %
?-? : (%,%) -> %
?.? : (%,%) -> % if Integer has SGROUP
?.? : (%,Integer) -> K
?/? : (%,%) -> %
?=? : (%,%) -> Boolean
?^? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?~=? : (%,%) -> Boolean
D : % -> % if K has *: (Integer,K) -> K
D : (%,List Symbol) -> %
    if K has PDRING SYMBOL and K has *: (Integer,K) -> K
D : (%,List Symbol,List NonNegativeInteger) -> %
    if K has PDRING SYMBOL and K has *: (Integer,K) -> K
D : (%,NonNegativeInteger) -> % if K has *: (Integer,K) -> K
D : (%,Symbol) -> % if K has PDRING SYMBOL and K has *: (Integer,K) -> K
D : (%,Symbol,NonNegativeInteger) -> %
    if K has PDRING SYMBOL and K has *: (Integer,K) -> K
```

```
approximate : (%,Integer) -> K
    if K has **: (K,Integer) -> K and K has coerce: Symbol -> K
associates? : (%,%) -> Boolean
center : % -> K
characteristic : () -> NonNegativeInteger
charthRoot : % -> Union(%,"failed") if K has CHARNZ
coefficient : (%,Integer) -> K
coerce : % -> %
coerce : % -> OutputForm
coerce : Fraction Integer -> %
coerce : Integer -> %
coerce : K -> % if K has COMRING
complete : % -> %
degree : % -> Integer
differentiate : % -> % if K has *: (Integer,K) -> K
differentiate : (%,List Symbol) -> %
   if K has PDRING SYMBOL and K has *: (Integer,K) -> K
differentiate : (%,List Symbol,List NonNegativeInteger) -> %
   if K has PDRING SYMBOL and K has *: (Integer,K) -> K
differentiate : (%,NonNegativeInteger) -> %
   if K has *: (Integer,K) -> K
differentiate : (%,Symbol) -> %
   if K has PDRING SYMBOL and K has *: (Integer,K) -> K
differentiate : (%,Symbol,NonNegativeInteger) -> %
   if K has PDRING SYMBOL and K has *: (Integer,K) -> K
eval : (%,K) -> Stream K if K has **: (K,Integer) -> K
exquo : (%,%) -> Union(%,"failed")
extend : (%,Integer) -> %
hash : % -> SingleInteger
latex : % -> String
leadingCoefficient : % -> K
leadingMonomial : % -> %
map : ((K -> K),%) -> %
monomial : (%,List SingletonAsOrderedSet,List Integer) -> %
monomial : (%,SingletonAsOrderedSet,Integer) -> %
monomial : (K,Integer) -> %
monomial? : % -> Boolean
multiplyExponents : (%,PositiveInteger) -> %
one? : % -> Boolean
order : % -> Integer
order : (%,Integer) -> Integer
pole? : % -> Boolean
recip : % -> Union(%,"failed")
reductum : % -> %
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
terms : % -> Stream Record(k: Integer,c: K)
truncate : (%,Integer) -> %
truncate : (%,Integer,Integer) -> %
unit? : % -> Boolean
```

```
 unitCanonical : % -> %
 unitNormal : % -> Record(unit: %,canonical: %,associate: %)
 variable : % -> Symbol
 variables : % -> List SingletonAsOrderedSet
 zero? : % -> Boolean
```

These exports come from (p1221) Field():

```
 ?**? : (%,Integer) -> %
 ?*? : (%,K) -> %
 ?*? : (K,%) -> %
 ?/? : (%,K) -> % if K has FIELD
 ?^? : (%,Integer) -> %
 ?quo? : (%,%) -> %
 ?rem? : (%,%) -> %
 divide : (%,%) -> Record(quotient: %,remainder: %)
 euclideanSize : % -> NonNegativeInteger
 expressIdealMember : (List %,%) -> Union(List %,"failed")
 extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
 extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
 factor : % -> Factored %
 gcd : (%,%) -> %
 gcd : List % -> %
 gcdPolynomial :
  (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) ->
    SparseUnivariatePolynomial %
 inv : % -> %
 lcm : (%,%) -> %
 lcm : List % -> %
 multiEuclidean : (List %,%) -> Union(List %,"failed")
 prime? : % -> Boolean
 principalIdeal : List % -> Record(coef: List %,generator: %)
 sizeLess? : (%,%) -> Boolean
 squareFree : % -> Factored %
 squareFreePart : % -> %
```

— category LOCPOWC LocalPowerSeriesCategory —

```
)abbrev category LOCPOWC LocalPowerSeriesCategory
LocalPowerSeriesCategory(K:Field):Category == Implementation where

  INT   ==> Integer
  TERM  ==> Record(k:INT,c:K)
  SER   ==> Stream(TERM)
  NNI   ==> NonNegativeInteger

  Implementation ==>
    Join(Field,UnivariatePowerSeriesCategory(K,INT))  with
```

```
order: % -> Integer
  ++ order(s) returns the order of s.

findCoef: (%,Integer) -> K

coerce: SER -> %

coerce: % -> SER

posExpnPart: % -> %
  ++ posExpnPart(s) returns the series s less the terms with
  ++ negative exponant.

orderIfNegative: % -> Union(Integer,"failed")

removeFirstZeroes: % -> %

sbt: (%,%) -> %

delay: ( () -> % ) -> %
  ++ delay delayed the computation of the next term of the series given
  ++ by the input function.

monomial2series: (List %, List NNI, INT) -> %
  ++ monomial2series(ls,le,n) returns
  ++ t**n * reduce("*",[s ** e for s in ls for e in le])

removeZeroes: (INT,%) -> %
  ++ removeZeroes(n,s) removes the zero terms in the first n terms of s.

removeZeroes: % -> %
  ++ removeZeroes(s) removes the zero terms in  s.

series: (INT,K,%) -> %
  ++ series(e,c,s) create the series c*t**e + s.

shift: (%,INT) -> %
  ++ shift(s,n) returns t**n * s

filterUpTo : (%,INT)  -> %
  ++ filterUpTo(s,n) returns the series consisting of the terms
  ++ of s having degree strictly less than n.

coefOfFirstNonZeroTerm: % -> K
  ++ coefOfFirstNonZeroTerm(s) returns the first non zero coefficient
  ++ of the series.

printInfo: Boolean -> Boolean
  ++ printInfo(b) set a flag such that when true (b <- true) prints
  ++ some information during some critical computation.
```

```
      printInfo: () -> Boolean
        ++ returns the value of the \spad{printInfo} flag.
```

———-

— **LOCPOWC.dotabb** —

```
"LOCPOWC"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=LOCPOWC"];
"LOCPOWC" -> "UPSCAT"
```

———-

— **LOCPOWC.dotfull** —

```
"LocalPowerSeriesCategory(f:Field)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=LOCPOWC"];
"LocalPowerSeriesCategory(f:Field)" ->
  "UnivariatePowerSeriesCategory(c:Ring,e:OrderedAbelianMonoid)"
```

———-

— **LOCPOWC.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

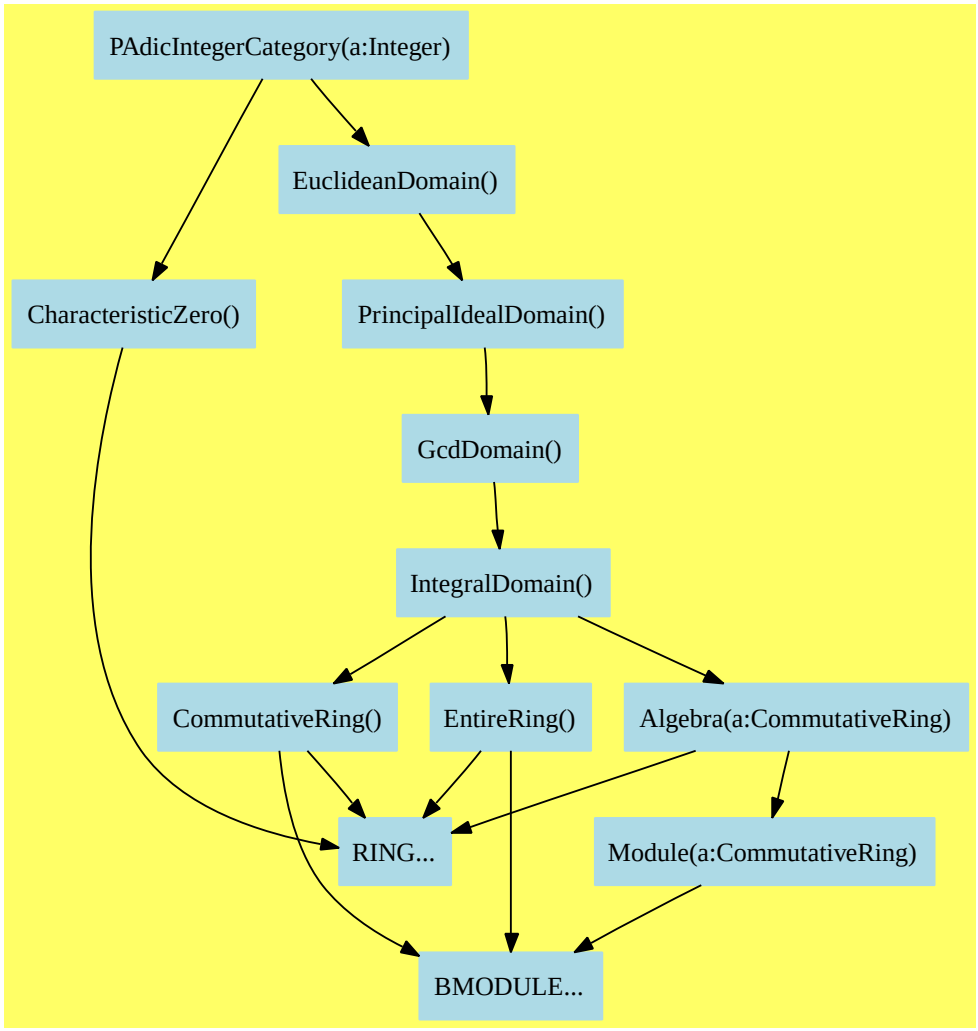
"LocalPowerSeriesCategory(f:Field)" [color=lightblue];
"LocalPowerSeriesCategory(f:Field)" ->
  "UnivariatePowerSeriesCategory(c:Ring,e:OrderedAbelianMonoid)"

"Ring" [color=lightblue];
"OrderedAbelianMonoid"  [color=lightblue];

}
```

———-

16.0.188 PAdicIntegerCategory (PADICCT)



— PAdicIntegerCategory.input —

```
)set break resume
)sys rm -f PAdicIntegerCategory.output
)spool PAdicIntegerCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
```

```
)show PAdicIntegerCategory
--R PAdicIntegerCategory p: Integer  is a category constructor
--R Abbreviation for PAdicIntegerCategory is PADICCT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PADICCT
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        associates? : (%,%) -> Boolean
--R coerce : % -> %                       coerce : Integer -> %
--R coerce : % -> OutputForm              complete : % -> %
--R digits : % -> Stream Integer          extend : (%,Integer) -> %
--R gcd : List % -> %                     gcd : (%,%) -> %
--R hash : % -> SingleInteger             latex : % -> String
--R lcm : List % -> %                     lcm : (%,%) -> %
--R moduloP : % -> Integer                modulus : () -> Integer
--R one? : % -> Boolean                   order : % -> NonNegativeInteger
--R ?quo? : (%,%) -> %                    quotientByP : % -> %
--R recip : % -> Union(%,"failed")        ?rem? : (%,%) -> %
--R sample : () -> %                      sizeLess? : (%,%) -> Boolean
--R sqrt : (%,Integer) -> %               unit? : % -> Boolean
--R unitCanonical : % -> %                zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R approximate : (%,Integer) -> Integer
--R characteristic : () -> NonNegativeInteger
--R divide : (%,%) -> Record(quotient: %,remainder: %)
--R euclideanSize : % -> NonNegativeInteger
--R expressIdealMember : (List %,%) -> Union(List %,"failed")
--R exquo : (%,%) -> Union(%,"failed")
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivari
--R multiEuclidean : (List %,%) -> Union(List %,"failed")
--R principalIdeal : List % -> Record(coef: List %,generator: %)
--R root : (SparseUnivariatePolynomial Integer,Integer) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R
--E 1

)spool
)lisp (bye)
```

———-

— **PAdicIntegerCategory.help** —

```
====================================================================
PAdicIntegerCategory examples
====================================================================

See Also:
o )show PAdicIntegerCategory
```

———-

See:

⇐ "CharacteristicZero" (CHARZ) 10.0.136 on page 838
⇐ "EuclideanDomain" (EUCDOM) 15.0.181 on page 1185

Exports:

0	1	approximate	associates?
characteristic	coerce	complete	digits
divide	euclideanSize	expressIdealMember	exquo
extend	extendedEuclidean	gcd	gcdPolynomial
hash	latex	lcm	moduloP
modulus	multiEuclidean	one?	order
principalIdeal	quotientByP	recip	root
sample	sizeLess?	sqrt	subtractIfCan
unit?	unitCanonical	unitNormal	zero?
?*?	?**?	?+?	?-?
-?	?=?	?quo?	?rem?
?~=?	?^?		

Attributes Exported:

- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) - > D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
approximate : (%,Integer) -> Integer
complete : % -> %
digits : % -> Stream Integer
extend : (%,Integer) -> %
moduloP : % -> Integer
modulus : () -> Integer
order : % -> NonNegativeInteger
quotientByP : % -> %
root : (SparseUnivariatePolynomial Integer,Integer) -> %
sqrt : (%,Integer) -> %
```

These exports come from (p1185) EuclideanDomain():

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean
characteristic : () -> NonNegativeInteger
coerce : % -> %
coerce : Integer -> %
coerce : % -> OutputForm
divide : (%,%) -> Record(quotient: %,remainder: %)
euclideanSize : % -> NonNegativeInteger
expressIdealMember : (List %,%) -> Union(List %,"failed")
exquo : (%,%) -> Union(%,"failed")
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
gcd : (%,%) -> %
gcd : List % -> %
gcdPolynomial :
  (SparseUnivariatePolynomial %,
   SparseUnivariatePolynomial %) ->
     SparseUnivariatePolynomial %
hash : % -> SingleInteger
latex : % -> String
lcm : List % -> %
lcm : (%,%) -> %
multiEuclidean : (List %,%) -> Union(List %,"failed")
one? : % -> Boolean
principalIdeal : List % -> Record(coef: List %,generator: %)
recip : % -> Union(%,"failed")
sample : () -> %
sizeLess? : (%,%) -> Boolean
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
```

```
 ?*? : (%,%) -> %
 ?*? : (Integer,%) -> %
 ?*? : (PositiveInteger,%) -> %
 ?*? : (NonNegativeInteger,%) -> %
 ?-? : (%,%) -> %
 -? : % -> %
 ?**? : (%,PositiveInteger) -> %
 ?**? : (%,NonNegativeInteger) -> %
 ?^? : (%,PositiveInteger) -> %
 ?^? : (%,NonNegativeInteger) -> %
 ?quo? : (%,%) -> %
 ?rem? : (%,%) -> %
```

— category PADICCT PAdicIntegerCategory —

```
)abbrev category PADICCT PAdicIntegerCategory
++ Author: Clifton J. Williamson
++ Date Created: 15 May 1990
++ Date Last Updated: 15 May 1990
++ Basic Operations:
++ Related Domains:
++ Also See:
++ AMS Classifications:
++ Keywords: p-adic, completion
++ Examples:
++ References:
++ Description:
++ This is the category of stream-based representations of
++ the p-adic integers.

PAdicIntegerCategory(p): Category == Definition where
  p   :   Integer
  I   ==> Integer
  NNI ==> NonNegativeInteger
  ST  ==> Stream
  SUP ==> SparseUnivariatePolynomial

  Definition ==> Join(EuclideanDomain,CharacteristicZero) with
    digits: % -> ST I
      ++ \spad{digits(x)} returns a stream of p-adic digits of x.
    order: % -> NNI
      ++ \spad{order(x)} returns the exponent of the highest power of p
      ++ dividing x.
    extend: (%,I) -> %
      ++ \spad{extend(x,n)} forces the computation of digits up to order n.
    complete: % -> %
      ++ \spad{complete(x)} forces the computation of all digits.
    modulus: () -> I
      ++ \spad{modulus()} returns the value of p.
```

```
    moduloP: % -> I
      ++ \spad{modulo(x)} returns a, where \spad{x = a + b p}.
    quotientByP: % -> %
      ++ \spad{quotientByP(x)} returns b, where \spad{x = a + b p}.
    approximate: (%,I) -> I
      ++ \spad{approximate(x,n)} returns an integer y such that
      ++ \spad{y = x (mod p^n)}
      ++ when n is positive, and 0 otherwise.
    sqrt: (%,I) -> %
      ++ \spad{sqrt(b,a)} returns a square root of b.
      ++ Argument \spad{a} is a square root of b \spad{(mod p)}.
    root: (SUP I,I) -> %
      ++ \spad{root(f,a)} returns a root of the polynomial \spad{f}.
      ++ Argument \spad{a} must be a root of \spad{f} \spad{(mod p)}.
```

———-

— PADICCT.dotabb —

```
"PADICCT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PADICCT"];
"PADICCT" -> "CHARZ"
"PADICCT" -> "EUCDOM"
```

———-

— PADICCT.dotfull —

```
"PAdicIntegerCategory(a:Integer)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PADICCT"];
"PAdicIntegerCategory(a:Integer)" -> "CharacteristicZero()"
"PAdicIntegerCategory(a:Integer)" -> "EuclideanDomain()"
```

———-

— PADICCT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"PAdicIntegerCategory(a:Integer)" [color=lightblue];
"PAdicIntegerCategory(a:Integer)" -> "CharacteristicZero()"
"PAdicIntegerCategory(a:Integer)" -> "EuclideanDomain()"
```

```
"CharacteristicZero()" [color=lightblue];
"CharacteristicZero()" -> "RING..."

"EuclideanDomain()" [color=lightblue];
"EuclideanDomain()" -> "PrincipalIdealDomain()"

"PrincipalIdealDomain()" [color=lightblue];
"PrincipalIdealDomain()" -> "GcdDomain()"

"GcdDomain()" [color=lightblue];
"GcdDomain()" -> "IntegralDomain()"

"IntegralDomain()" [color=lightblue];
"IntegralDomain()" -> "CommutativeRing()"
"IntegralDomain()" -> "Algebra(a:CommutativeRing)"
"IntegralDomain()" -> "EntireRing()"

"CommutativeRing()" [color=lightblue];
"CommutativeRing()" -> "RING..."
"CommutativeRing()" -> "BMODULE..."

"EntireRing()" [color=lightblue];
"EntireRing()" -> "RING..."
"EntireRing()" -> "BMODULE..."

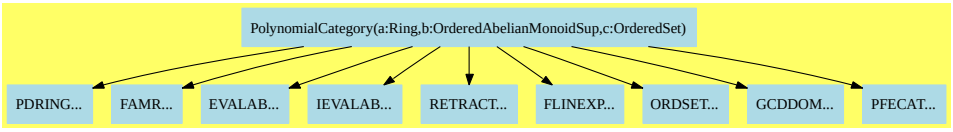
"Algebra(a:CommutativeRing)" [color=lightblue];
"Algebra(a:CommutativeRing)" -> "RING..."
"Algebra(a:CommutativeRing)" -> "Module(a:CommutativeRing)"

"Module(a:CommutativeRing)" [color=lightblue];
"Module(a:CommutativeRing)" -> "BMODULE..."

"RING..." [color=lightblue];
"BMODULE..." [color=lightblue];
}
```

———-

16.0.189 PolynomialCategory (POLYCAT)



— PolynomialCategory.input —

```
)set break resume
)sys rm -f PolynomialCategory.output
)spool PolynomialCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show PolynomialCategory
--R PolynomialCategory(R: Ring,E: OrderedAbelianMonoidSup,VarSet: OrderedSet)  is a category con
--R Abbreviation for PolynomialCategory is POLYCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for POLYCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,R) -> %                      ?*? : (R,%) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R D : (%,List VarSet) -> %              D : (%,VarSet) -> %
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        coefficient : (%,E) -> R
--R coefficients : % -> List R            coerce : VarSet -> %
--R coerce : R -> %                       coerce : Integer -> %
--R coerce : % -> OutputForm              degree : % -> E
--R differentiate : (%,VarSet) -> %       eval : (%,VarSet,%) -> %
--R eval : (%,VarSet,R) -> %              eval : (%,List %,List %) -> %
--R eval : (%,%,%) -> %                   eval : (%,Equation %) -> %
--R eval : (%,List Equation %) -> %       ground : % -> R
--R ground? : % -> Boolean                hash : % -> SingleInteger
--R latex : % -> String                   leadingCoefficient : % -> R
--R leadingMonomial : % -> %              map : ((R -> R),%) -> %
--R mapExponents : ((E -> E),%) -> %      minimumDegree : % -> E
--R monomial : (R,E) -> %                 monomial? : % -> Boolean
--R monomials : % -> List %               one? : % -> Boolean
--R pomopo! : (%,R,E,%) -> %              primitiveMonomials : % -> List %
--R recip : % -> Union(%,"failed")        reductum : % -> %
--R retract : % -> VarSet                 retract : % -> R
--R sample : () -> %                      variables : % -> List VarSet
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (Fraction Integer,%) -> % if R has ALGEBRA FRAC INT
--R ?*? : (%,Fraction Integer) -> % if R has ALGEBRA FRAC INT
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (%,R) -> % if R has FIELD
--R ?<? : (%,%) -> Boolean if R has ORDSET
```

```
--R ?<=? : (%,%) -> Boolean if R has ORDSET
--R ?>? : (%,%) -> Boolean if R has ORDSET
--R ?>=? : (%,%) -> Boolean if R has ORDSET
--R D : (%,List VarSet,List NonNegativeInteger) -> %
--R D : (%,VarSet,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R associates? : (%,%) -> Boolean if R has INTDOM
--R binomThmExpt : (%,%,NonNegativeInteger) -> % if R has COMRING
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if and(has($,CharacteristicNonZero),has(R,PolynomialFactorizationExp
--R coefficient : (%,List VarSet,List NonNegativeInteger) -> %
--R coefficient : (%,VarSet,NonNegativeInteger) -> %
--R coerce : Fraction Integer -> % if R has RETRACT FRAC INT or R has ALGEBRA FRAC INT
--R coerce : % -> % if R has INTDOM
--R conditionP : Matrix % -> Union(Vector %,"failed") if and(has($,CharacteristicNonZero),has(R,PolynomialFa
--R content : (%,VarSet) -> % if R has GCDDOM
--R content : % -> R if R has GCDDOM
--R convert : % -> InputForm if VarSet has KONVERT INFORM and R has KONVERT INFORM
--R convert : % -> Pattern Integer if VarSet has KONVERT PATTERN INT and R has KONVERT PATTERN INT
--R convert : % -> Pattern Float if VarSet has KONVERT PATTERN FLOAT and R has KONVERT PATTERN FLOAT
--R degree : (%,List VarSet) -> List NonNegativeInteger
--R degree : (%,VarSet) -> NonNegativeInteger
--R differentiate : (%,List VarSet,List NonNegativeInteger) -> %
--R differentiate : (%,VarSet,NonNegativeInteger) -> %
--R differentiate : (%,List VarSet) -> %
--R discriminant : (%,VarSet) -> % if R has COMRING
--R eval : (%,List VarSet,List %) -> %
--R eval : (%,List VarSet,List R) -> %
--R exquo : (%,%) -> Union(%,"failed") if R has INTDOM
--R exquo : (%,R) -> Union(%,"failed") if R has INTDOM
--R factor : % -> Factored % if R has PFECAT
--R factorPolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial % if R has PFECA
--R factorSquareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial % if R
--R gcd : (%,%) -> % if R has GCDDOM
--R gcd : List % -> % if R has GCDDOM
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivariatePolynomia
--R isExpt : % -> Union(Record(var: VarSet,exponent: NonNegativeInteger),"failed")
--R isPlus : % -> Union(List %,"failed")
--R isTimes : % -> Union(List %,"failed")
--R lcm : (%,%) -> % if R has GCDDOM
--R lcm : List % -> % if R has GCDDOM
--R mainVariable : % -> Union(VarSet,"failed")
--R max : (%,%) -> % if R has ORDSET
--R min : (%,%) -> % if R has ORDSET
--R minimumDegree : (%,List VarSet) -> List NonNegativeInteger
--R minimumDegree : (%,VarSet) -> NonNegativeInteger
--R monicDivide : (%,%,VarSet) -> Record(quotient: %,remainder: %)
--R monomial : (%,List VarSet,List NonNegativeInteger) -> %
--R monomial : (%,VarSet,NonNegativeInteger) -> %
--R multivariate : (SparseUnivariatePolynomial %,VarSet) -> %
```

```
--R multivariate : (SparseUnivariatePolynomial R,VarSet) -> %
--R numberOfMonomials : % -> NonNegativeInteger
--R patternMatch : (%,Pattern Integer,PatternMatchResult(Integer,%)) -> PatternMatchResult(Integ
--R patternMatch : (%,Pattern Float,PatternMatchResult(Float,%)) -> PatternMatchResult(Float,%)
--R prime? : % -> Boolean if R has PFECAT
--R primitivePart : (%,VarSet) -> % if R has GCDDOM
--R primitivePart : % -> % if R has GCDDOM
--R reducedSystem : Matrix % -> Matrix R
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix R,vec: Vector R)
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer) if R
--R reducedSystem : Matrix % -> Matrix Integer if R has LINEXP INT
--R resultant : (%,%,VarSet) -> % if R has COMRING
--R retract : % -> Integer if R has RETRACT INT
--R retract : % -> Fraction Integer if R has RETRACT FRAC INT
--R retractIfCan : % -> Union(VarSet,"failed")
--R retractIfCan : % -> Union(Integer,"failed") if R has RETRACT INT
--R retractIfCan : % -> Union(Fraction Integer,"failed") if R has RETRACT FRAC INT
--R retractIfCan : % -> Union(R,"failed")
--R solveLinearPolynomialEquation : (List SparseUnivariatePolynomial %,SparseUnivariatePolynomia
--R squareFree : % -> Factored % if R has GCDDOM
--R squareFreePart : % -> % if R has GCDDOM
--R squareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R totalDegree : (%,List VarSet) -> NonNegativeInteger
--R totalDegree : % -> NonNegativeInteger
--R unit? : % -> Boolean if R has INTDOM
--R unitCanonical : % -> % if R has INTDOM
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %) if R has INTDOM
--R univariate : % -> SparseUnivariatePolynomial R
--R univariate : (%,VarSet) -> SparseUnivariatePolynomial %
--R
--E 1

)spool
)lisp (bye)
```

———-

— PolynomialCategory.help —

```
=====================================================================
PolynomialCategory examples
=====================================================================

See Also:
o )show PolynomialCategory
```

———-

See:

⇒ “DifferentialPolynomialCategory” (DPOLCAT) 17.0.192 on page 1306
⇒ “RecursivePolynomialCategory” (RPOLCAT) 17.0.201 on page 1408
⇒ “UnivariatePolynomialCategory” (UPOLYC) 17.0.204 on page 1471

⇐ “InnerEvalable” (IEVALAB) 2.0.12 on page 35
⇐ “Evalable” (EVALAB) 3.0.25 on page 80
⇐ “FiniteAbelianMonoidRing” (FAMR) 14.0.175 on page 1141
⇐ “GcdDomain” (GCDDOM) 13.0.173 on page 1129
⇐ “FullyLinearlyExplicitRingOver” (FLINEXP) 11.0.154 on page 960
⇐ “OrderedSet” (ORDSET) 4.0.56 on page 216
⇐ “PartialDifferentialRing” (PDRING) 10.0.145 on page 885
⇐ “PolynomialFactorizationExplicit” (PFECAT) 15.0.183 on page 1202
⇐ “RetractableTo” (RETRACT) 2.0.18 on page 55

Exports:

0	1	associates?
binomThmExpt	characteristic	charthRoot
coefficient	coefficients	coerce
conditionP	content	convert
D	degree	differentiate
discriminant	eval	exquo
factor	factorPolynomial	factorSquareFreePolynomial
gcd	gcdPolynomial	ground
ground?	hash	isExpt
isPlus	isTimes	latex
lcm	leadingCoefficient	leadingMonomial
mainVariable	map	mapExponents
max	min	minimumDegree
monicDivide	monomial	monomial?
monomials	multivariate	numberOfMonomials
one?	patternMatch	pomopo!
prime?	primitiveMonomials	primitivePart
recip	reducedSystem	reductum
resultant	retract	retractIfCan
sample	solveLinearPolynomialEquation	squareFree
squareFreePart	squareFreePolynomial	subtractIfCan
totalDegree	unit?	unitCanonical
unitNormal	univariate	variables
zero?	?*?	?**?
?+?	?-?	-?
?=?	?^?	?~=?
?/?	?<?	?<=?
?>?	?>=?	

Attributes Exported:

-
- if R has canonicalUnitNormal then canonicalUnitNormal where **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- if $ has IntegralDomain then noZeroDivisors where **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- if $ has CommutativeRing then commutative("*") where **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.

- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
degree : (%,VarSet) -> NonNegativeInteger
degree : (%,List VarSet) -> List NonNegativeInteger
mainVariable : % -> Union(VarSet,"failed")
minimumDegree : (%,List VarSet) -> List NonNegativeInteger
minimumDegree : (%,VarSet) -> NonNegativeInteger
monomial : (%,VarSet,NonNegativeInteger) -> %
multivariate : (SparseUnivariatePolynomial %,VarSet) -> %
multivariate : (SparseUnivariatePolynomial R,VarSet) -> %
univariate : (%,VarSet) -> SparseUnivariatePolynomial %
univariate : % -> SparseUnivariatePolynomial R
variables : % -> List VarSet
```

These are implemented by this category:

```
charthRoot : % -> Union(%,"failed")
 if
   and(has($,CharacteristicNonZero),
       has(R,PolynomialFactorizationExplicit))
   or R has CHARNZ
coefficient : (%,VarSet,NonNegativeInteger) -> %
coefficient : (%,List VarSet,List NonNegativeInteger) -> %
conditionP : Matrix % -> Union(Vector %,"failed")
 if
   and(has($,CharacteristicNonZero),
       has(R,PolynomialFactorizationExplicit))
content : (%,VarSet) -> % if R has GCDDOM
convert : % -> Pattern Integer
    if VarSet has KONVERT PATTERN INT
    and R has KONVERT PATTERN INT
convert : % -> Pattern Float
    if VarSet has KONVERT PATTERN FLOAT
    and R has KONVERT PATTERN FLOAT
convert : % -> InputForm
    if VarSet has KONVERT INFORM
    and R has KONVERT INFORM
discriminant : (%,VarSet) -> % if R has COMRING
eval : (%,List Equation %) -> %
factor : % -> Factored % if R has PFECAT
factorPolynomial :
   SparseUnivariatePolynomial % ->
      Factored SparseUnivariatePolynomial %
        if R has PFECAT
factorSquareFreePolynomial :
   SparseUnivariatePolynomial % ->
      Factored SparseUnivariatePolynomial %
        if R has PFECAT
```

```
gcdPolynomial : (SparseUnivariatePolynomial %,
                 SparseUnivariatePolynomial %) ->
                    SparseUnivariatePolynomial %
    if R has GCDDOM
isExpt : % ->
  Union(Record(var: VarSet,exponent: NonNegativeInteger),"failed")
isPlus : % -> Union(List %,"failed")
isTimes : % -> Union(List %,"failed")
monicDivide : (%,%,VarSet) -> Record(quotient: %,remainder: %)
monomial : (%,List VarSet,List NonNegativeInteger) -> %
monomials : % -> List %
patternMatch :
  (%,Pattern Integer,PatternMatchResult(Integer,%)) ->
    PatternMatchResult(Integer,%)
      if VarSet has PATMAB INT
      and R has PATMAB INT
patternMatch :
   (%,Pattern Float,PatternMatchResult(Float,%)) ->
     PatternMatchResult(Float,%)
       if VarSet has PATMAB FLOAT
       and R has PATMAB FLOAT
primitiveMonomials : % -> List %
primitivePart : % -> % if R has GCDDOM
primitivePart : (%,VarSet) -> % if R has GCDDOM
reducedSystem : Matrix % -> Matrix R
reducedSystem : (Matrix %,Vector %) ->
   Record(mat: Matrix R,vec: Vector R)
resultant : (%,%,VarSet) -> % if R has COMRING
retract : % -> VarSet
retractIfCan : % -> Union(VarSet,"failed")
solveLinearPolynomialEquation :
   (List SparseUnivariatePolynomial %,
    SparseUnivariatePolynomial %) ->
       Union(List SparseUnivariatePolynomial %,"failed")
         if R has PFECAT
squareFree : % -> Factored % if R has GCDDOM
squareFreePart : % -> % if R has GCDDOM
totalDegree : % -> NonNegativeInteger
totalDegree : (%,List VarSet) -> NonNegativeInteger
?<? : (%,%) -> Boolean if R has ORDSET
```

These exports come from (p885) PartialDifferentialRing(VarSet)
where VarSet:OrderedSet:

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : Integer -> %
coerce : % -> OutputForm
D : (%,List VarSet) -> %
```

```
D : (%,VarSet) -> %
D : (%,List VarSet,List NonNegativeInteger) -> %
D : (%,VarSet,NonNegativeInteger) -> %
differentiate : (%,VarSet) -> %
differentiate : (%,List VarSet,List NonNegativeInteger) -> %
differentiate : (%,VarSet,NonNegativeInteger) -> %
differentiate : (%,List VarSet) -> %
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
```

These exports come from (p1141) FiniteAbelianMonoidRing(R,E) where R:Ring and E:OrderedAbelianMonoidSup:

```
associates? : (%,%) -> Boolean if R has INTDOM
binomThmExpt : (%,%,NonNegativeInteger) -> %
    if R has COMRING
coefficient : (%,E) -> R
coefficients : % -> List R
coerce : R -> %
coerce : Fraction Integer -> %
    if R has RETRACT FRAC INT
    or R has ALGEBRA FRAC INT
coerce : % -> % if R has INTDOM
content : % -> R if R has GCDDOM
degree : % -> E
exquo : (%,R) -> Union(%,"failed") if R has INTDOM
exquo : (%,%) -> Union(%,"failed") if R has INTDOM
ground : % -> R
ground? : % -> Boolean
leadingCoefficient : % -> R
leadingMonomial : % -> %
map : ((R -> R),%) -> %
```

```
mapExponents : ((E -> E),%) -> %
minimumDegree : % -> E
monomial : (R,E) -> %
monomial? : % -> Boolean
numberOfMonomials : % -> NonNegativeInteger
pomopo! : (%,R,E,%) -> %
reductum : % -> %
retract : % -> Integer if R has RETRACT INT
retract : % -> Fraction Integer
    if R has RETRACT FRAC INT
retractIfCan : % -> Union(Integer,"failed")
    if R has RETRACT INT
retractIfCan : % -> Union(Fraction Integer,"failed")
    if R has RETRACT FRAC INT
unit? : % -> Boolean if R has INTDOM
unitCanonical : % -> % if R has INTDOM
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
    if R has INTDOM
?*? : (%,R) -> %
?*? : (R,%) -> %
?*? : (Fraction Integer,%) -> %
    if R has ALGEBRA FRAC INT
?*? : (%,Fraction Integer) -> %
    if R has ALGEBRA FRAC INT
?/? : (%,R) -> % if R has FIELD
```

These exports come from (p80) Evalable(PolynomialCategory(...)):

```
eval : (%,Equation %) -> %
eval : (%,List %,List %) -> %
eval : (%,%,%) -> %
```

These exports come from (p35) InnerEvalable(VarSet,R)
where VarSet:OrderedSet and R:Ring

```
eval : (%,VarSet,R) -> %
eval : (%,List VarSet,List R) -> %
```

These exports come from (p35) InnerEvalable(VarSet,R)
where VarSet:OrderedSet and R:PolynomialCategory(...):

```
eval : (%,VarSet,%) -> %
eval : (%,List VarSet,List %) -> %
```

These exports come from (p55) RetractableTo(VarSet)
where VarSet:OrderedSet:

```
coerce : VarSet -> %
retract : % -> R
retractIfCan : % -> Union(R,"failed")
```

These exports come from (p960) FullyLinearlyExplicitRingOver(R)
where R:Ring:

```
reducedSystem : (Matrix %,Vector %) ->
   Record(mat: Matrix Integer,vec: Vector Integer)
      if R has LINEXP INT
reducedSystem : Matrix % -> Matrix Integer
      if R has LINEXP INT
```

These exports come from (p216) OrderedSet():

```
max : (%,%) -> % if R has ORDSET
min : (%,%) -> % if R has ORDSET
?<=? : (%,%) -> Boolean if R has ORDSET
?>? : (%,%) -> Boolean if R has ORDSET
?>=? : (%,%) -> Boolean if R has ORDSET
```

These exports come from (p1129) GcdDomain():

```
gcd : (%,%) -> % if R has GCDDOM
gcd : List % -> % if R has GCDDOM
lcm : (%,%) -> % if R has GCDDOM
lcm : List % -> % if R has GCDDOM
```

These exports come from (p1202) PolynomialFactorizationExplicit():

```
prime? : % -> Boolean if R has PFECAT
squareFreePolynomial : SparseUnivariatePolynomial % ->
   Factored SparseUnivariatePolynomial %
     if R has PFECAT
```

— category POLYCAT PolynomialCategory —

```
)abbrev category POLYCAT PolynomialCategory
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions: Ring, monomial, coefficient, differentiate, eval
++ Related Constructors: Polynomial, DistributedMultivariatePolynomial
++ Also See: UnivariatePolynomialCategory
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The category for general multi-variate polynomials over a ring
++ R, in variables from VarSet, with exponents from the
++ \spadtype{OrderedAbelianMonoidSup}.
```

```
PolynomialCategory(R:Ring, E:OrderedAbelianMonoidSup, VarSet:OrderedSet):
       Category ==
  Join(PartialDifferentialRing VarSet, FiniteAbelianMonoidRing(R, E),
       Evalable %, InnerEvalable(VarSet, R),
       InnerEvalable(VarSet, %), RetractableTo VarSet,
       FullyLinearlyExplicitRingOver R) with
    degree : (%,VarSet) -> NonNegativeInteger
      ++ degree(p,v) gives the degree of polynomial p with respect
      ++ to the variable v.
    degree : (%,List(VarSet)) -> List(NonNegativeInteger)
      ++ degree(p,lv) gives the list of degrees of polynomial p
      ++ with respect to each of the variables in the list lv.
    coefficient: (%,VarSet,NonNegativeInteger) -> %
      ++ coefficient(p,v,n) views the polynomial p as a univariate
      ++ polynomial in v and returns the coefficient of the \spad{v**n} term.
    coefficient: (%,List VarSet,List NonNegativeInteger) -> %
      ++ coefficient(p, lv, ln) views the polynomial p as a polynomial
      ++ in the variables of lv and returns the coefficient of the term
      ++ \spad{lv**ln}, i.e. \spad{prod(lv_i ** ln_i)}.
    monomials: % -> List %
      ++ monomials(p) returns the list of non-zero monomials of
      ++ polynomial p, i.e.
      ++ \spad{monomials(sum(a_(i) X^(i))) = [a_(1) X^(1),...,a_(n) X^(n)]}.
    univariate   : (%,VarSet) -> SparseUnivariatePolynomial(%)
      ++ univariate(p,v) converts the multivariate polynomial p
      ++ into a univariate polynomial in v, whose coefficients are still
      ++ multivariate polynomials (in all the other variables).
    univariate   : % -> SparseUnivariatePolynomial(R)
      ++ univariate(p) converts the multivariate polynomial p,
      ++ which should actually involve only one variable,
      ++ into a univariate polynomial
      ++ in that variable, whose coefficients are in the ground ring.
      ++ Error: if polynomial is genuinely multivariate
    mainVariable  : % -> Union(VarSet,"failed")
      ++ mainVariable(p) returns the biggest variable which actually
      ++ occurs in the polynomial p, or "failed" if no variables are
      ++ present.
      ++ fails precisely if polynomial satisfies ground?
    minimumDegree : (%,VarSet) -> NonNegativeInteger
      ++ minimumDegree(p,v) gives the minimum degree of polynomial p
      ++ with respect to v, i.e. viewed a univariate polynomial in v
    minimumDegree : (%,List(VarSet)) -> List(NonNegativeInteger)
      ++ minimumDegree(p, lv) gives the list of minimum degrees of the
      ++ polynomial p with respect to each of the variables in the list lv
    monicDivide : (%,%,VarSet) -> Record(quotient:%,remainder:%)
      ++ monicDivide(a,b,v) divides the polynomial a by the polynomial b,
      ++ with each viewed as a univariate polynomial in v returning
      ++ both the quotient and remainder.
      ++ Error: if b is not monic with respect to v.
    monomial : (%,VarSet,NonNegativeInteger) -> %
```

```
  ++ monomial(a,x,n) creates the monomial \spad{a*x**n} where \spad{a} is
  ++ a polynomial, x is a variable and n is a nonnegative integer.
monomial : (%,List VarSet,List NonNegativeInteger) -> %
  ++ monomial(a,[v1..vn],[e1..en]) returns \spad{a*prod(vi**ei)}.
multivariate : (SparseUnivariatePolynomial(R),VarSet) -> %
  ++ multivariate(sup,v) converts an anonymous univariable
  ++ polynomial sup to a polynomial in the variable v.
multivariate : (SparseUnivariatePolynomial(%),VarSet) -> %
  ++ multivariate(sup,v) converts an anonymous univariable
  ++ polynomial sup to a polynomial in the variable v.
isPlus: % -> Union(List %, "failed")
  ++ isPlus(p) returns \spad{[m1,...,mn]} if polynomial
  ++ \spad{p = m1 + ... + mn} and
  ++ \spad{n >= 2} and each mi is a nonzero monomial.
isTimes: % -> Union(List %, "failed")
  ++ isTimes(p) returns \spad{[a1,...,an]} if polynomial
  ++ \spad{p = a1 ... an} and \spad{n >= 2}, and, for each i,
  ++ ai is either a nontrivial constant in R or else of the
  ++ form \spad{x**e}, where \spad{e > 0} is an integer
  ++ and x in a member of VarSet.
isExpt: % -> Union(Record(var:VarSet, exponent:NonNegativeInteger),_
                   "failed")
  ++ isExpt(p) returns \spad{[x, n]} if polynomial p has the
  ++ form \spad{x**n} and \spad{n > 0}.
totalDegree : % -> NonNegativeInteger
  ++ totalDegree(p) returns the largest sum over all monomials
  ++ of all exponents of a monomial.
totalDegree : (%,List VarSet) -> NonNegativeInteger
  ++ totalDegree(p, lv) returns the maximum sum (over all monomials
  ++ of polynomial p) of the variables in the list lv.
variables : % -> List(VarSet)
  ++ variables(p) returns the list of those variables actually
  ++ appearing in the polynomial p.
primitiveMonomials: % -> List %
  ++ primitiveMonomials(p) gives the list of monomials of the
  ++ polynomial p with their coefficients removed. Note that
  ++ \spad{primitiveMonomials(sum(a_(i) X^(i))) = [X^(1),...,X^(n)]}.
if R has OrderedSet  then OrderedSet
-- OrderedRing view removed to allow EXPR to define abs
--if R has OrderedRing then OrderedRing
if (R has ConvertibleTo InputForm) and
   (VarSet has ConvertibleTo InputForm) then
     ConvertibleTo InputForm
if (R has ConvertibleTo Pattern Integer) and
   (VarSet has ConvertibleTo Pattern Integer) then
     ConvertibleTo Pattern Integer
if (R has ConvertibleTo Pattern Float) and
   (VarSet has ConvertibleTo Pattern Float) then
     ConvertibleTo Pattern Float
if (R has PatternMatchable Integer) and
```

```
      (VarSet has PatternMatchable Integer) then
        PatternMatchable Integer
   if (R has PatternMatchable Float) and
      (VarSet has PatternMatchable Float) then
        PatternMatchable Float
   if R has CommutativeRing then
     resultant : (%,%,VarSet) -> %
        ++ resultant(p,q,v) returns the resultant of the polynomials
        ++ p and q with respect to the variable v.
     discriminant : (%,VarSet) -> %
        ++ discriminant(p,v) returns the disriminant of the polynomial p
        ++ with respect to the variable v.
   if R has GcdDomain then
     GcdDomain
     content: (%,VarSet) -> %
       ++ content(p,v) is the gcd of the coefficients of the polynomial p
       ++ when p is viewed as a univariate polynomial with respect to the
       ++ variable v.
       ++ Thus, for polynomial 7*x**2*y + 14*x*y**2, the gcd of the
       ++ coefficients with respect to x is 7*y.
     primitivePart: % -> %
       ++ primitivePart(p) returns the unitCanonical associate of the
       ++ polynomial p with its content divided out.
     primitivePart: (%,VarSet) -> %
       ++ primitivePart(p,v) returns the unitCanonical associate of the
       ++ polynomial p with its content with respect to the variable v
       ++ divided out.
     squareFree: % -> Factored %
       ++ squareFree(p) returns the square free factorization of the
       ++ polynomial p.
     squareFreePart: % -> %
       ++ squareFreePart(p) returns product of all the irreducible factors
       ++ of polynomial p each taken with multiplicity one.

   -- assertions
   if R has canonicalUnitNormal then canonicalUnitNormal
            ++ we can choose a unique representative for each
            ++ associate class.
            ++ This normalization is chosen to be normalization of
            ++ leading coefficient (by default).
   if R has PolynomialFactorizationExplicit then
      PolynomialFactorizationExplicit
add
   p:%
   v:VarSet
   ln:List NonNegativeInteger
   lv:List VarSet
   n:NonNegativeInteger
   pp,qq:SparseUnivariatePolynomial %
```

```
    eval(p:%, l:List Equation %) ==
      empty? l => p
      for e in l repeat
        retractIfCan(lhs e)@Union(VarSet,"failed") case "failed" =>
             error "cannot find a variable to evaluate"
      lvar:=[retract(lhs e)@VarSet for e in l]
      eval(p, lvar,[rhs e for e in l]$List(%))

    monomials p ==
--    zero? p => empty()
--    concat(leadingMonomial p, monomials reductum p)
--    replaced by sequential version for efficiency, by WMSIT, 7/30/90
      ml:= empty$List(%)
      while p ^= 0 repeat
        ml:=concat(leadingMonomial p, ml)
        p:= reductum p
      reverse ml

    isPlus p ==
      empty? rest(l := monomials p) => "failed"
      l

    isTimes p ==
      empty?(lv := variables p) or not monomial? p => "failed"
      l := [monomial(1, v, degree(p, v)) for v in lv]
--      one?(r := leadingCoefficient p) =>
      ((r := leadingCoefficient p) = 1) =>
        empty? rest lv => "failed"
        l
      concat(r::%, l)

    isExpt p ==
      (u := mainVariable p) case "failed" => "failed"
      p = monomial(1, u::VarSet, d := degree(p, u::VarSet)) =>
        [u::VarSet, d]
      "failed"

    coefficient(p,v,n) == coefficient(univariate(p,v),n)

    coefficient(p,lv,ln) ==
       empty? lv =>
         empty? ln => p
         error "mismatched lists in coefficient"
       empty? ln  => error "mismatched lists in coefficient"
       coefficient(coefficient(univariate(p,first lv),first ln),
                   rest lv,rest ln)

    monomial(p,lv,ln) ==
       empty? lv =>
         empty? ln => p
```

```
      error "mismatched lists in monomial"
    empty? ln  => error "mismatched lists in monomial"
    monomial(monomial(p,first lv, first ln),rest lv, rest ln)

  retract(p:%):VarSet ==
    q := mainVariable(p)::VarSet
    q::% = p => q
    error "Polynomial is not a single variable"

  retractIfCan(p:%):Union(VarSet, "failed") ==
    ((q := mainVariable p) case VarSet) and (q::VarSet::% = p) => q
    "failed"

  mkPrim(p:%):% == monomial(1,degree p)

  primitiveMonomials p == [mkPrim q for q in monomials p]

  totalDegree p ==
      ground? p => 0
      u := univariate(p, mainVariable(p)::VarSet)
      d: NonNegativeInteger := 0
      while u ^= 0 repeat
        d := max(d, degree u + totalDegree leadingCoefficient u)
        u := reductum u
      d

  totalDegree(p,lv) ==
      ground? p => 0
      u := univariate(p, v:=(mainVariable(p)::VarSet))
      d: NonNegativeInteger := 0
      w: NonNegativeInteger := 0
      if member?(v, lv) then w:=1
      while u ^= 0 repeat
        d := max(d, w*(degree u) + totalDegree(leadingCoefficient u,lv))
        u := reductum u
      d

  if R has CommutativeRing then
      resultant(p1,p2,mvar) ==
        resultant(univariate(p1,mvar),univariate(p2,mvar))

      discriminant(p,var) ==
        discriminant(univariate(p,var))

  if R has IntegralDomain then
    allMonoms(l:List %):List(%) ==
      removeDuplicates_! concat [primitiveMonomials p for p in l]

    P2R(p:%, b:List E, n:NonNegativeInteger):Vector(R) ==
      w := new(n, 0)$Vector(R)
```

```
    for i in minIndex w .. maxIndex w for bj in b repeat
      qsetelt_!(w, i, coefficient(p, bj))
    w

  eq2R(l:List %, b:List E):Matrix(R) ==
    matrix [[coefficient(p, bj) for p in l] for bj in b]

  reducedSystem(m:Matrix %):Matrix(R) ==
    l := listOfLists m
    b := removeDuplicates_!
                      concat [allMonoms r for r in l]$List(List(%))
    d := [degree bj for bj in b]
    mm := eq2R(first l, d)
    l := rest l
    while not empty? l repeat
      mm := vertConcat(mm, eq2R(first l, d))
      l := rest l
    mm

  reducedSystem(m:Matrix %, v:Vector %):
   Record(mat:Matrix R, vec:Vector R) ==
    l := listOfLists m
    r := entries v
    b : List % := removeDuplicates_! concat(allMonoms r,
                     concat [allMonoms s for s in l]$List(List(%)))
    d := [degree bj for bj in b]
    n := #d
    mm := eq2R(first l, d)
    w := P2R(first r, d, n)
    l := rest l
    r := rest r
    while not empty? l repeat
      mm := vertConcat(mm, eq2R(first l, d))
      w := concat(w, P2R(first r, d, n))
      l := rest l
      r := rest r
    [mm, w]

if R has PolynomialFactorizationExplicit then
   -- we might be in trouble if its actually only
   -- a univariate polynomial category - have to remember to
   -- over-ride these in UnivariatePolynomialCategory

   PFBR ==>PolynomialFactorizationByRecursion(R,E,VarSet,%)

   gcdPolynomial(pp,qq) ==
      gcdPolynomial(pp,qq)$GeneralPolynomialGcdPackage(E,VarSet,R,%)

   solveLinearPolynomialEquation(lpp,pp) ==
     solveLinearPolynomialEquationByRecursion(lpp,pp)$PFBR
```

```
    factorPolynomial(pp) ==
      factorByRecursion(pp)$PFBR

    factorSquareFreePolynomial(pp) ==
      factorSquareFreeByRecursion(pp)$PFBR

    factor p ==
      v:Union(VarSet,"failed"):=mainVariable p
      v case "failed" =>
        ansR:=factor leadingCoefficient p
        makeFR(unit(ansR)::%,
               [[w.flg,w.fctr::%,w.xpnt] for w in factorList ansR])
      up:SparseUnivariatePolynomial %:=univariate(p,v)
      ansSUP:=factorByRecursion(up)$PFBR
      makeFR(multivariate(unit(ansSUP),v),
             [[ww.flg,multivariate(ww.fctr,v),ww.xpnt]
              for ww in factorList ansSUP])
    if R has CharacteristicNonZero then
       mat: Matrix %

       conditionP mat ==
         ll:=listOfLists transpose mat  --hence each list corresponds to a
                                        --column, i.e. to one variable
         llR:List List R := [ empty() for z in first ll]
         monslist:List List % := empty()
         ch:=characteristic()$%
         for l in ll repeat
             mons:= "setUnion"/[primitiveMonomials u for u in l]
             redmons:List % :=[]
             for m in mons repeat
                 vars:=variables m
                 degs:=degree(m,vars)
                 deg1:List NonNegativeInteger
                 deg1:=[ ((nd:=d:Integer exquo ch:Integer)
                             case "failed" => return "failed" ;
                              nd::Integer::NonNegativeInteger)
                        for d in degs ]
                 redmons:=[monomial(1,vars,deg1),:redmons]
                 llR:=[[ground coefficient(u,vars,degs),:v]_
                         for u in l for v in llR]
             monslist:=[redmons,:monslist]
         ans:=conditionP transpose matrix llR
         ans case "failed" => "failed"
         i:NonNegativeInteger:=0
         [ +/[m*(ans.(i:=i+1))::% for m in mons ]
           for mons in monslist]

  if R has CharacteristicNonZero then
       charthRootlv:(%,List VarSet,NonNegativeInteger) ->_
```

```
                                                  Union(%,"failed")
      charthRoot p ==
        vars:= variables p
        empty? vars =>
          ans := charthRoot ground p
          ans case "failed" => "failed"
          ans::R::%
        ch:=characteristic()$%
        charthRootlv(p,vars,ch)

      charthRootlv(p,vars,ch) ==
        empty? vars =>
          ans := charthRoot ground p
          ans case "failed" => "failed"
          ans::R::%
        v:=first vars
        vars:=rest vars
        d:=degree(p,v)
        ans:% := 0
        while (d>0) repeat
           (dd:=(d::Integer exquo ch::Integer)) case "failed" =>
                  return "failed"
           cp:=coefficient(p,v,d)
           p:=p-monomial(cp,v,d)
           ansx:=charthRootlv(cp,vars,ch)
           ansx case "failed" => return "failed"
           d:=degree(p,v)
           ans:=ans+monomial(ansx,v,dd::Integer::NonNegativeInteger)
        ansx:=charthRootlv(p,vars,ch)
        ansx case "failed" => return "failed"
        return ans+ansx

monicDivide(p1,p2,mvar) ==
   result:=monicDivide(univariate(p1,mvar),univariate(p2,mvar))
   [multivariate(result.quotient,mvar),
    multivariate(result.remainder,mvar)]

if R has GcdDomain then
  if R has EuclideanDomain and R has CharacteristicZero then
   squareFree p == squareFree(p)$MultivariateSquareFree(E,VarSet,R,%)
  else
    squareFree p == squareFree(p)$PolynomialSquareFree(VarSet,E,R,%)

  squareFreePart p ==
    unit(s := squareFree p) * */[f.factor for f in factors s]

  content(p,v) == content univariate(p,v)

  primitivePart p ==
    zero? p => p
```

```
      unitNormal((p exquo content p) ::%).canonical

    primitivePart(p,v) ==
      zero? p => p
      unitNormal((p exquo content(p,v)) ::%).canonical

  if R has OrderedSet then
    p:% < q:% ==
      (dp:= degree p) < (dq := degree q) => (leadingCoefficient q) > 0
      dq < dp => (leadingCoefficient p) < 0
      leadingCoefficient(p - q) < 0

    if (R has PatternMatchable Integer) and
       (VarSet has PatternMatchable Integer) then
         patternMatch(p:%, pat:Pattern Integer,
          l:PatternMatchResult(Integer, %)) ==
            patternMatch(p, pat,
              l)$PatternMatchPolynomialCategory(Integer,E,VarSet,R,%)

    if (R has PatternMatchable Float) and
       (VarSet has PatternMatchable Float) then
         patternMatch(p:%, pat:Pattern Float,
          l:PatternMatchResult(Float, %)) ==
            patternMatch(p, pat,
              l)$PatternMatchPolynomialCategory(Float,E,VarSet,R,%)

  if (R has ConvertibleTo Pattern Integer) and
     (VarSet has ConvertibleTo Pattern Integer) then
       convert(x:%):Pattern(Integer) ==
         map(convert, convert,
            x)$PolynomialCategoryLifting(E,VarSet,R,%,Pattern Integer)

  if (R has ConvertibleTo Pattern Float) and
     (VarSet has ConvertibleTo Pattern Float) then
       convert(x:%):Pattern(Float) ==
         map(convert, convert,
          x)$PolynomialCategoryLifting(E, VarSet, R, %, Pattern Float)

  if (R has ConvertibleTo InputForm) and
     (VarSet has ConvertibleTo InputForm) then
       convert(p:%):InputForm ==
         map(convert, convert,
                  p)$PolynomialCategoryLifting(E,VarSet,R,%,InputForm)
```

———-

— POLYCAT.dotabb —

```
"POLYCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=POLYCAT"];
"POLYCAT" -> "PDRING"
"POLYCAT" -> "FAMR"
"POLYCAT" -> "EVALAB"
"POLYCAT" -> "IEVALAB"
"POLYCAT" -> "RETRACT"
"POLYCAT" -> "FLINEXP"
"POLYCAT" -> "ORDSET"
"POLYCAT" -> "GCDDOM"
"POLYCAT" -> "PFECAT"
```

———-

— POLYCAT.dotfull —

```
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=POLYCAT"];
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "PartialDifferentialRing(a:OrderedSet)"
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "FiniteAbelianMonoidRing(a:Ring,b:OrderedAbelianMonoidSup)"
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "Evalable(PolynomialCategory(...))"
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "InnerEvalable(a:OrderedSet,b:Ring)"
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "InnerEvalable(a:OrderedSet,b:PolynomialCategory(...))"
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "RetractableTo(a:OrderedSet)"
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "FullyLinearlyExplicitRingOver(a:Ring)"
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "OrderedSet()"
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "GcdDomain()"
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "PolynomialFactorizationExplicit()"

"PolynomialCategory(a:Ring,b:NonNegativeInteger,c:SingletonAsOrderedSet)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=POLYCAT"];
"PolynomialCategory(a:Ring,b:NonNegativeInteger,c:SingletonAsOrderedSet)"
 -> "PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
```

———-

— POLYCAT.dotpic —

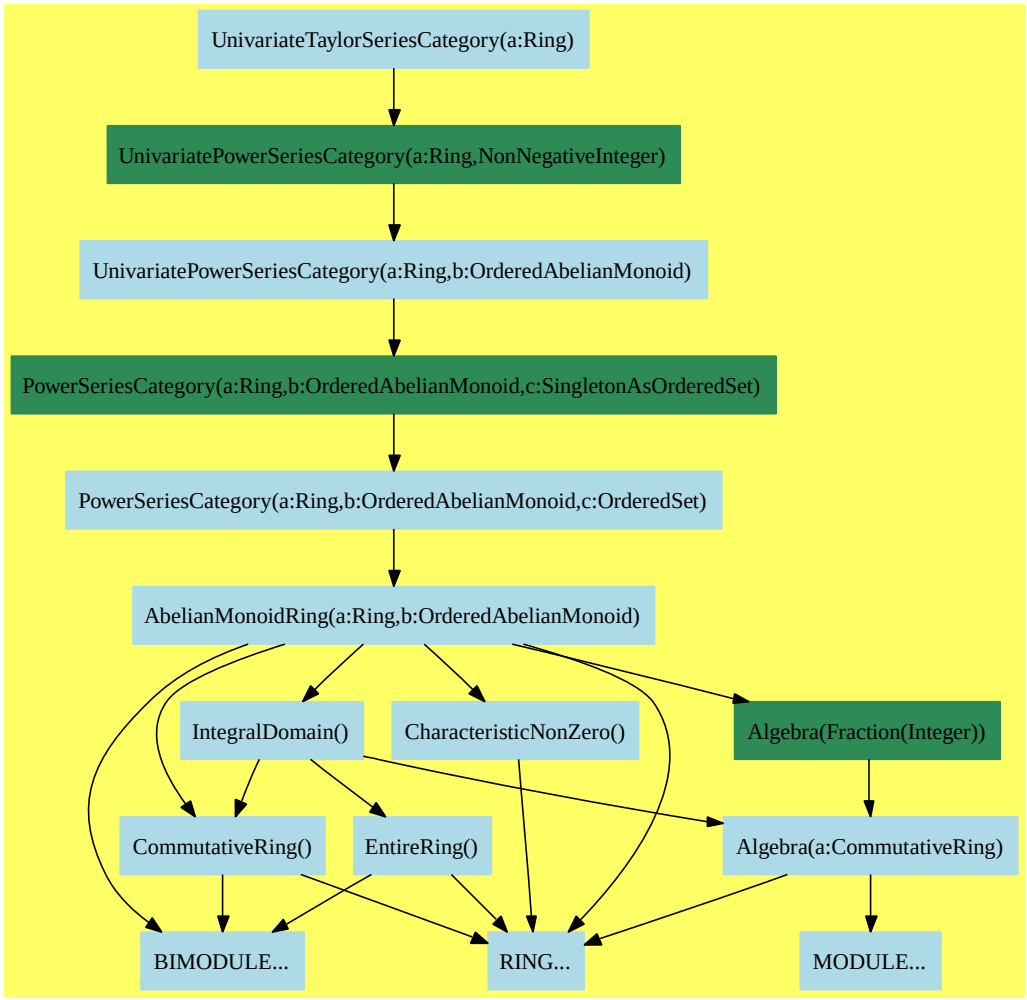
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=POLYCAT"];
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "PDRING..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "FAMR..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "EVALAB..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "IEVALAB..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "RETRACT..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "FLINEXP..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "ORDSET..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "GCDDOM..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "PFECAT..."

"PDRING..." [color=lightblue];
"FAMR..." [color=lightblue];
"EVALAB..." [color=lightblue];
"IEVALAB..." [color=lightblue];
"RETRACT..." [color=lightblue];
"FLINEXP..." [color=lightblue];
"ORDSET..." [color=lightblue];
"GCDDOM..." [color=lightblue];
"PFECAT..." [color=lightblue];

}
```

16.0.190 UnivariateTaylorSeriesCategory (UTSCAT)



— UnivariateTaylorSeriesCategory.input —

```
)set break resume
)sys rm -f UnivariateTaylorSeriesCategory.output
)spool UnivariateTaylorSeriesCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show UnivariateTaylorSeriesCategory
```

```
--R UnivariateTaylorSeriesCategory Coef: Ring  is a category constructor
--R Abbreviation for UnivariateTaylorSeriesCategory is UTSCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for UTSCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (Coef,%) -> %                  ?*? : (%,Coef) -> %
--R ?*? : (%,%) -> %                     ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %       ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                     ?-? : (%,%) -> %
--R -? : % -> %                          ?=? : (%,%) -> Boolean
--R 1 : () -> %                          0 : () -> %
--R ?^? : (%,PositiveInteger) -> %       center : % -> Coef
--R coefficients : % -> Stream Coef      coerce : Integer -> %
--R coerce : % -> OutputForm             complete : % -> %
--R degree : % -> NonNegativeInteger     hash : % -> SingleInteger
--R latex : % -> String                  leadingCoefficient : % -> Coef
--R leadingMonomial : % -> %             map : ((Coef -> Coef),%) -> %
--R monomial? : % -> Boolean             one? : % -> Boolean
--R order : % -> NonNegativeInteger      pole? : % -> Boolean
--R quoByVar : % -> %                    recip : % -> Union(%,"failed")
--R reductum : % -> %                    sample : () -> %
--R series : Stream Coef -> %            variable : % -> Symbol
--R zero? : % -> Boolean                 ?~=? : (%,%) -> Boolean
--R ?*? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
--R ?*? : (Fraction Integer,%) -> % if Coef has ALGEBRA FRAC INT
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
--R ?**? : (%,%) -> % if Coef has ALGEBRA FRAC INT
--R ?**? : (%,Coef) -> % if Coef has FIELD
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (%,Coef) -> % if Coef has FIELD
--R D : % -> % if Coef has *: (NonNegativeInteger,Coef) -> Coef
--R D : (%,NonNegativeInteger) -> % if Coef has *: (NonNegativeInteger,Coef) -> Coef
--R D : (%,Symbol) -> % if Coef has PDRING SYMBOL and Coef has *: (NonNegativeInteger,Coef) -> C
--R D : (%,List Symbol) -> % if Coef has PDRING SYMBOL and Coef has *: (NonNegativeInteger,Coef)
--R D : (%,Symbol,NonNegativeInteger) -> % if Coef has PDRING SYMBOL and Coef has *: (NonNegativ
--R D : (%,List Symbol,List NonNegativeInteger) -> % if Coef has PDRING SYMBOL and Coef has *: (
--R ?^? : (%,NonNegativeInteger) -> %
--R acos : % -> % if Coef has ALGEBRA FRAC INT
--R acosh : % -> % if Coef has ALGEBRA FRAC INT
--R acot : % -> % if Coef has ALGEBRA FRAC INT
--R acoth : % -> % if Coef has ALGEBRA FRAC INT
--R acsc : % -> % if Coef has ALGEBRA FRAC INT
--R acsch : % -> % if Coef has ALGEBRA FRAC INT
--R approximate : (%,NonNegativeInteger) -> Coef if Coef has **: (Coef,NonNegativeInteger) -> Co
--R asec : % -> % if Coef has ALGEBRA FRAC INT
--R asech : % -> % if Coef has ALGEBRA FRAC INT
--R asin : % -> % if Coef has ALGEBRA FRAC INT
--R asinh : % -> % if Coef has ALGEBRA FRAC INT
```

```
--R associates? : (%,%) -> Boolean if Coef has INTDOM
--R atan : % -> % if Coef has ALGEBRA FRAC INT
--R atanh : % -> % if Coef has ALGEBRA FRAC INT
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if Coef has CHARNZ
--R coefficient : (%,NonNegativeInteger) -> Coef
--R coerce : Coef -> % if Coef has COMRING
--R coerce : % -> % if Coef has INTDOM
--R coerce : Fraction Integer -> % if Coef has ALGEBRA FRAC INT
--R cos : % -> % if Coef has ALGEBRA FRAC INT
--R cosh : % -> % if Coef has ALGEBRA FRAC INT
--R cot : % -> % if Coef has ALGEBRA FRAC INT
--R coth : % -> % if Coef has ALGEBRA FRAC INT
--R csc : % -> % if Coef has ALGEBRA FRAC INT
--R csch : % -> % if Coef has ALGEBRA FRAC INT
--R differentiate : % -> % if Coef has *: (NonNegativeInteger,Coef) -> Coef
--R differentiate : (%,NonNegativeInteger) -> % if Coef has *: (NonNegativeInteger,Coef) -> Coef
--R differentiate : (%,Symbol) -> % if Coef has PDRING SYMBOL and Coef has *: (NonNegativeInteger,Coef) -> C
--R differentiate : (%,List Symbol) -> % if Coef has PDRING SYMBOL and Coef has *: (NonNegativeInteger,Coef)
--R differentiate : (%,Symbol,NonNegativeInteger) -> % if Coef has PDRING SYMBOL and Coef has *: (NonNegativ
--R differentiate : (%,List Symbol,List NonNegativeInteger) -> % if Coef has PDRING SYMBOL and Coef has *: C
--R ?.? : (%,%) -> % if NonNegativeInteger has SGROUP
--R ?.? : (%,NonNegativeInteger) -> Coef
--R eval : (%,Coef) -> Stream Coef if Coef has **: (Coef,NonNegativeInteger) -> Coef
--R exp : % -> % if Coef has ALGEBRA FRAC INT
--R exquo : (%,%) -> Union(%,"failed") if Coef has INTDOM
--R extend : (%,NonNegativeInteger) -> %
--R integrate : (%,Symbol) -> % if Coef has ACFS INT and Coef has PRIMCAT and Coef has TRANFUN and Coef has
--R integrate : % -> % if Coef has ALGEBRA FRAC INT
--R log : % -> % if Coef has ALGEBRA FRAC INT
--R monomial : (%,List SingletonAsOrderedSet,List NonNegativeInteger) -> %
--R monomial : (%,SingletonAsOrderedSet,NonNegativeInteger) -> %
--R monomial : (Coef,NonNegativeInteger) -> %
--R multiplyCoefficients : ((Integer -> Coef),%) -> %
--R multiplyExponents : (%,PositiveInteger) -> %
--R nthRoot : (%,Integer) -> % if Coef has ALGEBRA FRAC INT
--R order : (%,NonNegativeInteger) -> NonNegativeInteger
--R pi : () -> % if Coef has ALGEBRA FRAC INT
--R polynomial : (%,NonNegativeInteger,NonNegativeInteger) -> Polynomial Coef
--R polynomial : (%,NonNegativeInteger) -> Polynomial Coef
--R sec : % -> % if Coef has ALGEBRA FRAC INT
--R sech : % -> % if Coef has ALGEBRA FRAC INT
--R series : Stream Record(k: NonNegativeInteger,c: Coef) -> %
--R sin : % -> % if Coef has ALGEBRA FRAC INT
--R sinh : % -> % if Coef has ALGEBRA FRAC INT
--R sqrt : % -> % if Coef has ALGEBRA FRAC INT
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R tan : % -> % if Coef has ALGEBRA FRAC INT
--R tanh : % -> % if Coef has ALGEBRA FRAC INT
--R terms : % -> Stream Record(k: NonNegativeInteger,c: Coef)
```

```
--R truncate : (%,NonNegativeInteger,NonNegativeInteger) -> %
--R truncate : (%,NonNegativeInteger) -> %
--R unit? : % -> Boolean if Coef has INTDOM
--R unitCanonical : % -> % if Coef has INTDOM
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %) if Coef has INTDOM
--R variables : % -> List SingletonAsOrderedSet
--R
--E 1

)spool
)lisp (bye)
```

———-

— **UnivariateTaylorSeriesCategory.help** —

```
====================================================================
UnivariateTaylorSeriesCategory examples
====================================================================

See Also:
o )show UnivariateTaylorSeriesCategory
```

———-

See:

⇐ “RadicalCategory” (RADCAT) 2.0.17 on page 52
⇐ “TranscendentalFunctionCategory” (TRANFUN) 3.0.35 on page 117
⇐ “UnivariatePowerSeriesCategory” (UPSCAT) 15.0.184 on page 1210

Exports:

0	1	acos	acosh
acot	acoth	acsc	acsch
approximate	asec	asech	asin
asinh	associates?	atan	atanh
center	characteristic	charthRoot	coefficient
coefficients	coerce	complete	cos
cosh	cot	coth	csc
csch	D	degree	differentiate
eval	exp	exquo	extend
hash	integrate	latex	leadingCoefficient
leadingMonomial	log	map	monomial
monomial?	multiplyCoefficients	multiplyExponents	nthRoot
one?	order	pi	pole?
polynomial	quoByVar	recip	reductum
sample	sec	sech	series
sin	sinh	sqrt	subtractIfCan
tan	tanh	terms	truncate
unit?	unitCanonical	unitNormal	variable
variables	zero?	?*?	?**?
?+?	?-?	-?	?=?
?^?	?~=?	?/?	?.?

Attributes Exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.
- if #1 has IntegralDomain then noZeroDivisors where **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- if #1 has CommutativeRing then commutative("*") where **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.

These are directly exported but not implemented:

```
coefficients : % -> Stream Coef
integrate : (%,Symbol) -> %
    if Coef has ACFS INT
    and Coef has PRIMCAT
    and Coef has TRANFUN
    and Coef has ALGEBRA FRAC INT
    or Coef has variables: Coef -> List Symbol
    and Coef has integrate: (Coef,Symbol) -> Coef
    and Coef has ALGEBRA FRAC INT
integrate : % -> % if Coef has ALGEBRA FRAC INT
```

```
multiplyCoefficients : ((Integer -> Coef),%) -> %
polynomial : (%,NonNegativeInteger,NonNegativeInteger) -> Polynomial Coef
polynomial : (%,NonNegativeInteger) -> Polynomial Coef
quoByVar : % -> %
series : Stream Coef -> %
series : Stream Record(k: NonNegativeInteger,c: Coef) -> %
```

These are implemented by this category:

```
acos : % -> % if Coef has ALGEBRA FRAC INT
acosh : % -> % if Coef has ALGEBRA FRAC INT
acot : % -> % if Coef has ALGEBRA FRAC INT
acoth : % -> % if Coef has ALGEBRA FRAC INT
acsc : % -> % if Coef has ALGEBRA FRAC INT
acsch : % -> % if Coef has ALGEBRA FRAC INT
asec : % -> % if Coef has ALGEBRA FRAC INT
asech : % -> % if Coef has ALGEBRA FRAC INT
asin : % -> % if Coef has ALGEBRA FRAC INT
asinh : % -> % if Coef has ALGEBRA FRAC INT
atan : % -> % if Coef has ALGEBRA FRAC INT
atanh : % -> % if Coef has ALGEBRA FRAC INT
coerce : % -> OutputForm
cos : % -> % if Coef has ALGEBRA FRAC INT
cosh : % -> % if Coef has ALGEBRA FRAC INT
cot : % -> % if Coef has ALGEBRA FRAC INT
coth : % -> % if Coef has ALGEBRA FRAC INT
csc : % -> % if Coef has ALGEBRA FRAC INT
csch : % -> % if Coef has ALGEBRA FRAC INT
exp : % -> % if Coef has ALGEBRA FRAC INT
log : % -> % if Coef has ALGEBRA FRAC INT
sinh : % -> % if Coef has ALGEBRA FRAC INT
sec : % -> % if Coef has ALGEBRA FRAC INT
sech : % -> % if Coef has ALGEBRA FRAC INT
sin : % -> % if Coef has ALGEBRA FRAC INT
tan : % -> % if Coef has ALGEBRA FRAC INT
tanh : % -> % if Coef has ALGEBRA FRAC INT
zero? : % -> Boolean
?**? : (%,Coef) -> % if Coef has FIELD
?**? : (%,%) -> % if Coef has ALGEBRA FRAC INT
?**? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
```

These exports come from (p1210) UnivariatePowerSeriesCategory(Coef,NNI) where Coef:Ring and NNI:NonNegativeInteger:

```
0 : () -> %
1 : () -> %
approximate : (%,NonNegativeInteger) -> Coef
    if Coef has **: (Coef,NonNegativeInteger) -> Coef
    and Coef has coerce: Symbol -> Coef
associates? : (%,%) -> Boolean if Coef has INTDOM
```

```
center : % -> Coef
characteristic : () -> NonNegativeInteger
charthRoot : % -> Union(%,"failed") if Coef has CHARNZ
coefficient : (%,NonNegativeInteger) -> Coef
coerce : Coef -> % if Coef has COMRING
coerce : % -> % if Coef has INTDOM
coerce : Fraction Integer -> % if Coef has ALGEBRA FRAC INT
coerce : Integer -> %
complete : % -> %
D : % -> % if Coef has *: (NonNegativeInteger,Coef) -> Coef
D : (%,NonNegativeInteger) -> %
    if Coef has *: (NonNegativeInteger,Coef) -> Coef
D : (%,Symbol) -> %
    if Coef has PDRING SYMBOL
    and Coef has *: (NonNegativeInteger,Coef) -> Coef
D : (%,List Symbol) -> %
    if Coef has PDRING SYMBOL
    and Coef has *: (NonNegativeInteger,Coef) -> Coef
D : (%,Symbol,NonNegativeInteger) -> %
    if Coef has PDRING SYMBOL
    and Coef has *: (NonNegativeInteger,Coef) -> Coef
D : (%,List Symbol,List NonNegativeInteger) -> %
    if Coef has PDRING SYMBOL
    and Coef has *: (NonNegativeInteger,Coef) -> Coef
differentiate : (%,List Symbol) -> %
    if Coef has PDRING SYMBOL
    and Coef has *: (NonNegativeInteger,Coef) -> Coef
differentiate : (%,Symbol,NonNegativeInteger) -> %
    if Coef has PDRING SYMBOL
    and Coef has *: (NonNegativeInteger,Coef) -> Coef
differentiate : (%,List Symbol,List NonNegativeInteger) -> %
    if Coef has PDRING SYMBOL
    and Coef has *: (NonNegativeInteger,Coef) -> Coef
differentiate : (%,Symbol) -> %
    if Coef has PDRING SYMBOL
    and Coef has *: (NonNegativeInteger,Coef) -> Coef
differentiate : (%,NonNegativeInteger) -> %
    if Coef has *: (NonNegativeInteger,Coef) -> Coef
differentiate : % -> %
    if Coef has *: (NonNegativeInteger,Coef) -> Coef
degree : % -> NonNegativeInteger
extend : (%,NonNegativeInteger) -> %
exquo : (%,%) -> Union(%,"failed") if Coef has INTDOM
eval : (%,Coef) -> Stream Coef
    if Coef has **: (Coef,NonNegativeInteger) -> Coef
hash : % -> SingleInteger
latex : % -> String
leadingCoefficient : % -> Coef
leadingMonomial : % -> %
map : ((Coef -> Coef),%) -> %
```

```
monomial : (Coef,NonNegativeInteger) -> %
monomial : (%,SingletonAsOrderedSet,NonNegativeInteger) -> %
monomial : (%,List SingletonAsOrderedSet,List NonNegativeInteger) -> %
monomial? : % -> Boolean
multiplyExponents : (%,PositiveInteger) -> %
one? : % -> Boolean
order : % -> NonNegativeInteger
order : (%,NonNegativeInteger) -> NonNegativeInteger
pole? : % -> Boolean
recip : % -> Union(%,"failed")
reductum : % -> %
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
terms : % -> Stream Record(k: NonNegativeInteger,c: Coef)
truncate : (%,NonNegativeInteger,NonNegativeInteger) -> %
truncate : (%,NonNegativeInteger) -> %
unit? : % -> Boolean if Coef has INTDOM
unitCanonical : % -> % if Coef has INTDOM
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
    if Coef has INTDOM
variable : % -> Symbol
variables : % -> List SingletonAsOrderedSet
?^? : (%,NonNegativeInteger) -> %
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (%,%) -> %
?-? : (%,%) -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?*? : (Integer,%) -> %
?*? : (Coef,%) -> %
?*? : (%,Coef) -> %
?*? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
?*? : (Fraction Integer,%) -> % if Coef has ALGEBRA FRAC INT
?/? : (%,Coef) -> % if Coef has FIELD
-? : % -> %
?.? : (%,%) -> % if NonNegativeInteger has SGROUP
?.? : (%,NonNegativeInteger) -> Coef
```

These exports come from (p117) TranscendentalFunctionCategory():

```
pi : () -> % if Coef has ALGEBRA FRAC INT
```

These exports come from (p52) RadicalCategory():

```
nthRoot : (%,Integer) -> % if Coef has ALGEBRA FRAC INT
```

```
 sqrt : % -> % if Coef has ALGEBRA FRAC INT
```

— category UTSCAT UnivariateTaylorSeriesCategory —

```
)abbrev category UTSCAT UnivariateTaylorSeriesCategory
++ Author: Clifton J. Williamson
++ Date Created: 21 December 1989
++ Date Last Updated: 26 May 1994
++ Basic Operations:
++ Related Domains:
++ Also See:
++ AMS Classifications:
++ Keywords: series, Taylor, linebacker
++ Examples:
++ References:
++ Description:
++ \spadtype{UnivariateTaylorSeriesCategory} is the category of Taylor
++ series in one variable.

UnivariateTaylorSeriesCategory(Coef): Category == Definition where
  Coef  : Ring
  I    ==> Integer
  L    ==> List
  NNI  ==> NonNegativeInteger
  OUT  ==> OutputForm
  RN   ==> Fraction Integer
  STTA ==> StreamTaylorSeriesOperations Coef
  STTF ==> StreamTranscendentalFunctions Coef
  STNC ==> StreamTranscendentalFunctionsNonCommutative Coef
  Term ==> Record(k:NNI,c:Coef)

  Definition ==> UnivariatePowerSeriesCategory(Coef,NNI) with

    series: Stream Term -> %
      ++ \spad{series(st)} creates a series from a stream of non-zero terms,
      ++ where a term is an exponent-coefficient pair.  The terms in the
      ++ stream should be ordered by increasing order of exponents.
    coefficients: % -> Stream Coef
      ++ \spad{coefficients(a0 + a1 x + a2 x**2 + ...)} returns a stream
      ++ of coefficients: \spad{[a0,a1,a2,...]}. The entries of the stream
      ++ may be zero.
    series: Stream Coef -> %
      ++ \spad{series([a0,a1,a2,...])} is the Taylor series
      ++ \spad{a0 + a1 x + a2 x**2 + ...}.
    quoByVar: % -> %
      ++ \spad{quoByVar(a0 + a1 x + a2 x**2 + ...)}
      ++ returns \spad{a1 + a2 x + a3 x**2 + ...}
      ++ Thus, this function substracts the constant term and divides by
      ++ the series variable.  This function is used when Laurent series
```

```
    ++ are represented by a Taylor series and an order.
  multiplyCoefficients: (I -> Coef,%) -> %
    ++ \spad{multiplyCoefficients(f,sum(n = 0..infinity,a[n] * x**n))}
    ++ returns \spad{sum(n = 0..infinity,f(n) * a[n] * x**n)}.
    ++ This function is used when Laurent series are represented by
    ++ a Taylor series and an order.
  polynomial: (%,NNI) -> Polynomial Coef
    ++ \spad{polynomial(f,k)} returns a polynomial consisting of the sum
    ++ of all terms of f of degree \spad{<= k}.
  polynomial: (%,NNI,NNI) -> Polynomial Coef
    ++ \spad{polynomial(f,k1,k2)} returns a polynomial consisting of the
    ++ sum of all terms of f of degree d with \spad{k1 <= d <= k2}.

  if Coef has Field then
    "**": (%,Coef) -> %
      ++ \spad{f(x) ** a} computes a power of a power series.
      ++ When the coefficient ring is a field, we may raise a series
      ++ to an exponent from the coefficient ring provided that the
      ++ constant coefficient of the series is 1.

  if Coef has Algebra Fraction Integer then
    integrate: % -> %
      ++ \spad{integrate(f(x))} returns an anti-derivative of the power
      ++ series \spad{f(x)} with constant coefficient 0.
      ++ We may integrate a series when we can divide coefficients
      ++ by integers.
    if Coef has integrate: (Coef,Symbol) -> Coef and _
       Coef has variables: Coef -> List Symbol then
      integrate: (%,Symbol) -> %
        ++ \spad{integrate(f(x),y)} returns an anti-derivative of the
        ++ power series \spad{f(x)} with respect to the variable \spad{y}.
    if Coef has TranscendentalFunctionCategory and _
       Coef has PrimitiveFunctionCategory and _
       Coef has AlgebraicallyClosedFunctionSpace Integer then
      integrate: (%,Symbol) -> %
        ++ \spad{integrate(f(x),y)} returns an anti-derivative of
        ++ the power series \spad{f(x)} with respect to the variable
        ++ \spad{y}.
    RadicalCategory
      --++ We provide rational powers when we can divide coefficients
      --++ by integers.
    TranscendentalFunctionCategory
      --++ We provide transcendental functions when we can divide
      --++ coefficients by integers.

  add

  zero? x ==
    empty? (coefs := coefficients x) => true
    (zero? frst coefs) and (empty? rst coefs) => true
```

```
      false

--% OutputForms

--  We provide defaulr output functions on UTSCAT using the functions
--  'coefficients', 'center', and 'variable'.

    factorials?: () -> Boolean
    -- check a global Lisp variable
    factorials?() == false

    termOutput: (I,Coef,OUT) -> OUT
    termOutput(k,c,vv) ==
    -- creates a term c * vv ** k
      k = 0 => c :: OUT
      mon := (k = 1 => vv; vv ** (k :: OUT))
--       if factorials?() and k > 1 then
--         c := factorial(k)$IntegerCombinatoricFunctions * c
--         mon := mon / hconcat(k :: OUT,"!" :: OUT)
      c = 1 => mon
      c = -1 => -mon
      (c :: OUT) * mon

    showAll?: () -> Boolean
    -- check a global Lisp variable
    showAll?() == true

    coerce(p:%):OUT ==
      empty? (uu := coefficients p) => (0$Coef) :: OUT
      var := variable p; cen := center p
      vv :=
        zero? cen => var :: OUT
        paren(var :: OUT - cen :: OUT)
      n : NNI ; count : NNI := _$streamCount$Lisp
      l : L OUT := empty()
      for n in 0..count while not empty? uu repeat
        if frst(uu) ^= 0 then
          l := concat(termOutput(n :: I,frst uu,vv),l)
        uu := rst uu
      if showAll?() then
        for n in (count + 1).. while explicitEntries? uu and _
              not eq?(uu,rst uu) repeat
          if frst(uu) ^= 0 then
            l := concat(termOutput(n :: I,frst uu,vv),l)
          uu := rst uu
      l :=
        explicitlyEmpty? uu => l
        eq?(uu,rst uu) and frst uu = 0 => l
        concat(prefix("O" :: OUT,[vv ** (n :: OUT)]),l)
      empty? l => (0$Coef) :: OUT
```

```
      reduce("+",reverse_! l)

    if Coef has Field then
      (x:%) ** (r:Coef) == series power(r,coefficients x)$STTA

    if Coef has Algebra Fraction Integer then
      if Coef has CommutativeRing then
        (x:%) ** (y:%)    == series(coefficients x **$STTF coefficients y)

        (x:%) ** (r:RN)   == series powern(r,coefficients x)$STTA

        exp x == series exp(coefficients x)$STTF

        log x == series log(coefficients x)$STTF

        sin x == series sin(coefficients x)$STTF

        cos x == series cos(coefficients x)$STTF

        tan x == series tan(coefficients x)$STTF

        cot x == series cot(coefficients x)$STTF

        sec x == series sec(coefficients x)$STTF

        csc x == series csc(coefficients x)$STTF

        asin x == series asin(coefficients x)$STTF

        acos x == series acos(coefficients x)$STTF

        atan x == series atan(coefficients x)$STTF

        acot x == series acot(coefficients x)$STTF

        asec x == series asec(coefficients x)$STTF

        acsc x == series acsc(coefficients x)$STTF

        sinh x == series sinh(coefficients x)$STTF

        cosh x == series cosh(coefficients x)$STTF

        tanh x == series tanh(coefficients x)$STTF

        coth x == series coth(coefficients x)$STTF

        sech x == series sech(coefficients x)$STTF

        csch x == series csch(coefficients x)$STTF
```

```
      asinh x == series asinh(coefficients x)$STTF

      acosh x == series acosh(coefficients x)$STTF

      atanh x == series atanh(coefficients x)$STTF

      acoth x == series acoth(coefficients x)$STTF

      asech x == series asech(coefficients x)$STTF

      acsch x == series acsch(coefficients x)$STTF

    else
      (x:%) ** (y:%) == series(coefficients x **$STNC coefficients y)

      (x:%) ** (r:RN) ==
        coefs := coefficients x
        empty? coefs =>
          positive? r => 0
          zero? r => error "0**0 undefined"
          error "0 raised to a negative power"
--        not one? frst coefs =>
        not (frst coefs = 1) =>
          error "**: constant coefficient should be 1"
        coefs := concat(0,rst coefs)
        onePlusX := monom(1,0)$STTA + $STTA monom(1,1)$STTA
        ratPow := powern(r,onePlusX)$STTA
        series compose(ratPow,coefs)$STTA

      exp x == series exp(coefficients x)$STNC

      log x == series log(coefficients x)$STNC

      sin x == series sin(coefficients x)$STNC

      cos x == series cos(coefficients x)$STNC

      tan x == series tan(coefficients x)$STNC

      cot x == series cot(coefficients x)$STNC

      sec x == series sec(coefficients x)$STNC

      csc x == series csc(coefficients x)$STNC

      asin x == series asin(coefficients x)$STNC

      acos x == series acos(coefficients x)$STNC
```

```
        atan x == series atan(coefficients x)$STNC

        acot x == series acot(coefficients x)$STNC

        asec x == series asec(coefficients x)$STNC

        acsc x == series acsc(coefficients x)$STNC

        sinh x == series sinh(coefficients x)$STNC

        cosh x == series cosh(coefficients x)$STNC

        tanh x == series tanh(coefficients x)$STNC

        coth x == series coth(coefficients x)$STNC

        sech x == series sech(coefficients x)$STNC

        csch x == series csch(coefficients x)$STNC

        asinh x == series asinh(coefficients x)$STNC

        acosh x == series acosh(coefficients x)$STNC

        atanh x == series atanh(coefficients x)$STNC

        acoth x == series acoth(coefficients x)$STNC

        asech x == series asech(coefficients x)$STNC

        acsch x == series acsch(coefficients x)$STNC
```

———-

— UTSCAT.dotabb —

```
"UTSCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=UTSCAT"];
"UTSCAT" -> "UPSCAT"
```

———-

— UTSCAT.dotfull —

```
"UnivariateTaylorSeriesCategory(a:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=UTSCAT"];
```

```
"UnivariateTaylorSeriesCategory(a:Ring)" ->
    "UnivariatePowerSeriesCategory(a:Ring,NonNegativeInteger)"
```

———-

— UTSCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"UnivariateTaylorSeriesCategory(a:Ring)" [color=lightblue];
"UnivariateTaylorSeriesCategory(a:Ring)" ->
    "UnivariatePowerSeriesCategory(a:Ring,NonNegativeInteger)"

"UnivariatePowerSeriesCategory(a:Ring,NonNegativeInteger)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=UPSCAT"];
"UnivariatePowerSeriesCategory(a:Ring,NonNegativeInteger)" ->
    "UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)"

"UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)"
 [color=lightblue];
"UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)" ->
 "PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:SingletonAsOrderedSet)"

"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:SingletonAsOrderedSet)"
 [color=seagreen];
"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:SingletonAsOrderedSet)"
  -> "PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)"

"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)"
 [color=lightblue];
"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)" ->
    "AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)"

"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" [color=lightblue];
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" -> "RING..."
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "BIMODULE..."
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "IntegralDomain()"
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "CharacteristicNonZero()"
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "CommutativeRing()"
"AbelianMonoidRing(a:Ring,b:OrderedAbelianMonoid)" ->
    "Algebra(Fraction(Integer))"
```

```
"IntegralDomain()" [color=lightblue];
"IntegralDomain()" -> "CommutativeRing()"
"IntegralDomain()" -> "Algebra(a:CommutativeRing)"
"IntegralDomain()" -> "EntireRing()"

"EntireRing()" [color=lightblue];
"EntireRing()" -> "RING..."
"EntireRing()" -> "BIMODULE..."

"CharacteristicNonZero()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=CHARNZ"];
"CharacteristicNonZero()" -> "RING..."

"Algebra(Fraction(Integer))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=ALGEBRA"];
"Algebra(Fraction(Integer))" -> "Algebra(a:CommutativeRing)"

"Algebra(a:CommutativeRing)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ALGEBRA"];
"Algebra(a:CommutativeRing)" -> "RING..."
"Algebra(a:CommutativeRing)" -> "MODULE..."

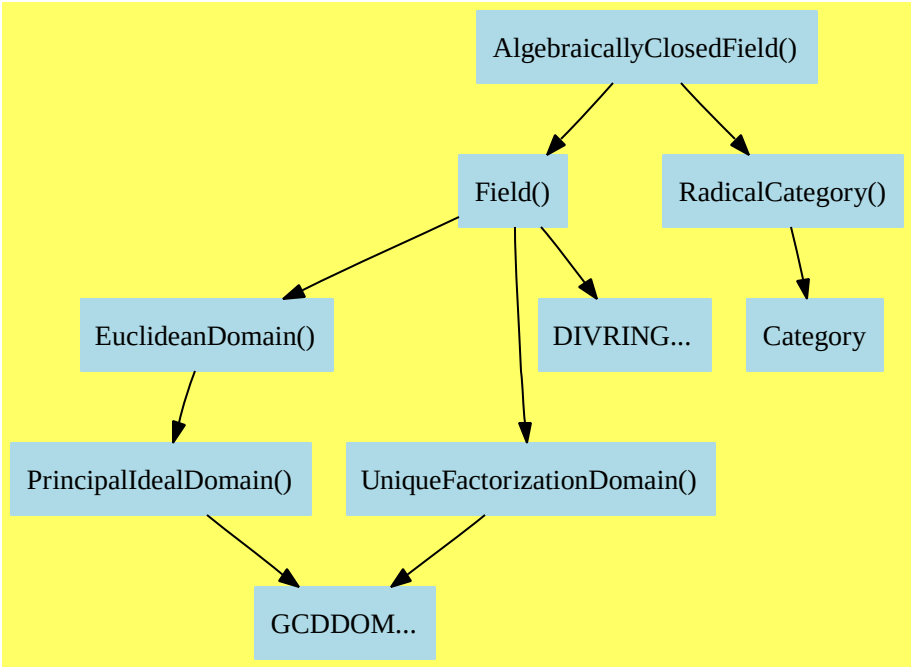
"CommutativeRing()" [color=lightblue];
"CommutativeRing()" -> "RING..."
"CommutativeRing()" -> "BIMODULE..."

"BIMODULE..." [color=lightblue];
"RING..." [color=lightblue];
"MODULE..." [color=lightblue];
}
```

Chapter 17

Category Layer 16

17.0.191 AlgebraicallyClosedField (ACF)



— **AlgebraicallyClosedField.input** —

```
)set break resume
)sys rm -f AlgebraicallyClosedField.output
)spool AlgebraicallyClosedField.output
```

```
)set message test on
)set message auto off
)clear all

--S 1 of 14
pi:Polynomial(Integer):=-3*x^3+2*x+13
--R
--R
--R            3
--R   (1)  - 3x  + 2x + 13
--R                                                    Type: Polynomial Integer
--E 1

--S 2 of 14
rootOf(pi)
--R
--R
--R   (2)  x
--R                                                       Type: AlgebraicNumber
--E 2

--S 3 of 14
rootsOf(pi)
--R
--R
--R   (3)  [%x0,%x1,- %x1 - %x0]
--R                                                   Type: List AlgebraicNumber
--E 3

--S 4 of 14
zeroOf(pi)
--R
--R
--R   (4)  x
--R                                                       Type: AlgebraicNumber
--E 4

--S 5 of 14
zerosOf(pi)
--R
--R
--R                +-------------+         +-------------+
--R                |       2               |       2
--R             - \|- 27%x3  + 24  - 3%x3 \|- 27%x3  + 24  - 3%x3
--R   (5)  [%x3,-------------------------,-----------------------]
--R                          6                       6
--R                                                    Type: List AlgebraicNumber
--E 5

--S 6 of 14
```

```
sup:SparseUnivariatePolynomial(Integer):=-3*x^3+2*x+13
--R
--R
--R            3
--R   (6)  - 3?  + 2? + 13
--R                                      Type: SparseUnivariatePolynomial Integer
--E 6

--S 7 of 14
rootOf(sup)
--R
--R
--R   (7)  %B
--R                                                        Type: AlgebraicNumber
--E 7

--S 8 of 14
rootOf(sup,x)
--R
--R
--R   (8)  x
--R                                                        Type: AlgebraicNumber
--E 8

--S 9 of 14
rootsOf(sup)
--R
--R
--R   (9)  [%%C0,%%C1,- %%C1 - %%C0]
--R                                                   Type: List AlgebraicNumber
--E 9

--S 10 of 14
rootsOf(sup,x)
--R
--R
--R   (10)  [%x6,%x7,- %x7 - %x6]
--R                                                   Type: List AlgebraicNumber
--E 10

--S 11 of 14
zeroOf(sup)
--R
--R
--R   (11)  %D
--R                                                        Type: AlgebraicNumber
--E 11

--S 12 of 14
zeroOf(sup,x)
```

```
--R
--R
--R   (12)  x
--R                                                    Type: AlgebraicNumber
--E 12

--S 13 of 14
zerosOf(sup)
--R
--R
--R                  +--------------+          +--------------+
--R                  |        2                |        2
--R               - \|- 27%%E0  + 24  - 3%%E0 \|- 27%%E0  + 24  - 3%%E0
--R   (13)  [%%E0,--------------------------,------------------------]
--R                             6                         6
--R                                                Type: List AlgebraicNumber
--E 13

--S 14 of 14
zerosOf(sup,x)
--R
--R
--R                 +-------------+         +-------------+
--R                 |       2               |       2
--R              - \|- 27%x9  + 24  - 3%x9 \|- 27%x9  + 24  - 3%x9
--R   (14)  [%x9,-------------------------,----------------------]
--R                           6                       6
--R                                                Type: List AlgebraicNumber
--E 14

)spool
)lisp (bye)
```

———-

— AlgebraicallyClosedField.help —

```
=====================================================================
AlgebraicallyClosedField examples
=====================================================================

Given the polynomial:

pi:Polynomial(Integer):=-3*x^3+2*x+13
              3
           - 3x  + 2x + 13


rootOf(pi)
```

```
          x


rootsOf(pi)
          [%x0,%x1,- %x1 - %x0]


zeroOf(pi)
          x


zerosOf(pi)
                  +-------------+         +-------------+
                  |       2               |       2
               - \|- 27%x3  + 24  - 3%x3 \|- 27%x3  + 24  - 3%x3
          [%x3,------------------------,-----------------------]
                                6                       6


These functions can also be applied to Sparse Univariate Polynomials:

sup:SparseUnivariatePolynomial(Integer):=-3*x^3+2*x+13
              3
          - 3?  + 2? + 13


rootOf(sup)
          %B


rootOf(sup,x)
          x


rootsOf(sup)
          [%%C0,%%C1,- %%C1 - %%C0]


rootsOf(sup,x)
          [%x6,%x7,- %x7 - %x6]


zeroOf(sup)
          %D


zeroOf(sup,x)
        x
```

```
zerosOf(sup)
                 +--------------+          +--------------+
                 |        2                 |        2
              - \|- 27%%E0  + 24  - 3%%E0 \|- 27%%E0  + 24  - 3%%E0
        [%%E0,--------------------------,------------------------]
                            6                          6


zerosOf(sup,x)
                +-------------+          +-------------+
                |       2                |       2
             - \|- 27%x9  + 24  - 3%x9 \|- 27%x9  + 24  - 3%x9
        [%x9,-----------------------,----------------------]
                           6                        6
```

```
See Also:
o )show AlgebraicallyClosedField
```

———-

See:

⇒ “AlgebraicallyClosedFunctionSpace” (ACFS) 18.0.205 on page 1495
⇐ “Field” (FIELD) 16.0.185 on page 1221
⇐ “RadicalCategory” (RADCAT) 2.0.17 on page 52

Exports:

0	1	associates?	characteristic
coerce	divide	euclideanSize	expressIdealMember
exquo	extendedEuclidean	factor	gcd
gcdPolynomial	hash	inv	latex
lcm	multiEuclidean	nthRoot	one?
prime?	principalIdeal	recip	rootOf
rootsOf	sample	sizeLess?	sqrt
squareFree	squareFreePart	subtractIfCan	unit?
unitCanonical	unitNormal	zero?	zeroOf
zerosOf	?*?	?**?	?+?
?-?	-?	?/?	?=?
?quo?	?rem?	?~=?	?^?

Attributes Exported:

- **canonicalUnitNormal** is true if we can choose a canonical representative for each

class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.

- **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.
- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
rootOf : Polynomial % -> %
```

These are implemented by this category:

```
rootOf : SparseUnivariatePolynomial % -> %
rootOf : (SparseUnivariatePolynomial %,Symbol) -> %
rootsOf : Polynomial % -> List %
rootsOf : (SparseUnivariatePolynomial %,Symbol) -> List %
rootsOf : SparseUnivariatePolynomial % -> List %
zeroOf : (SparseUnivariatePolynomial %,Symbol) -> %
zeroOf : Polynomial % -> %
zeroOf : SparseUnivariatePolynomial % -> %
zerosOf : Polynomial % -> List %
zerosOf : (SparseUnivariatePolynomial %,Symbol) -> List %
zerosOf : SparseUnivariatePolynomial % -> List %
```

These exports come from (p1221) Field():

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean
characteristic : () -> NonNegativeInteger
coerce : % -> %
coerce : Integer -> %
coerce : % -> OutputForm
coerce : Fraction Integer -> %
divide : (%,%) -> Record(quotient: %,remainder: %)
euclideanSize : % -> NonNegativeInteger
expressIdealMember : (List %,%) -> Union(List %,"failed")
exquo : (%,%) -> Union(%,"failed")
```

```
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
factor : % -> Factored %
gcd : (%,%) -> %
gcd : List % -> %
gcdPolynomial :
  (SparseUnivariatePolynomial %,
   SparseUnivariatePolynomial %) ->
      SparseUnivariatePolynomial %
hash : % -> SingleInteger
inv : % -> %
latex : % -> String
lcm : List % -> %
lcm : (%,%) -> %
multiEuclidean : (List %,%) -> Union(List %,"failed")
one? : % -> Boolean
prime? : % -> Boolean
principalIdeal : List % -> Record(coef: List %,generator: %)
recip : % -> Union(%,"failed")
sample : () -> %
sizeLess? : (%,%) -> Boolean
squareFree : % -> Factored %
squareFreePart : % -> %
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
zero? : % -> Boolean
?*? : (Fraction Integer,%) -> %
?*? : (%,Fraction Integer) -> %
?**? : (%,Integer) -> %
?^? : (%,Integer) -> %
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?/? : (%,%) -> %
?quo? : (%,%) -> %
?rem? : (%,%) -> %
```

These exports come from (p52) RadicalCategory():

```
 nthRoot : (%,Integer) -> %
 sqrt : % -> %
 ?**? : (%,Fraction Integer) -> %
```

— category ACF AlgebraicallyClosedField —

```
)abbrev category ACF AlgebraicallyClosedField
++ Author: Manuel Bronstein
++ Date Created: 22 Mar 1988
++ Date Last Updated: 27 November 1991
++ Keywords: algebraic, closure, field.
++ Description:
++ Model for algebraically closed fields.

AlgebraicallyClosedField(): Category == Join(Field,RadicalCategory) with
    rootOf: Polynomial $ -> $
      ++ rootOf(p) returns y such that \spad{p(y) = 0}.
      ++ Error: if p has more than one variable y.
      ++
      ++X a:Polynomial(Integer):=-3*x^3+2*x+13
      ++X rootOf(a)
    rootOf: SparseUnivariatePolynomial $ -> $
      ++ rootOf(p) returns y such that \spad{p(y) = 0}.
      ++
      ++X a:SparseUnivariatePolynomial(Integer):=-3*x^3+2*x+13
      ++X rootOf(a)
    rootOf: (SparseUnivariatePolynomial $, Symbol) -> $
      ++ rootOf(p, y) returns y such that \spad{p(y) = 0}.
      ++ The object returned displays as \spad{'y}.
      ++
      ++X a:SparseUnivariatePolynomial(Integer):=-3*x^3+2*x+13
      ++X rootOf(a,x)
    rootsOf: Polynomial $ -> List $
      ++ rootsOf(p) returns \spad{[y1,...,yn]} such that \spad{p(yi) = 0}.
      ++ Note that the returned symbols y1,...,yn are bound in the
      ++ interpreter to respective root values.
      ++ Error: if p has more than one variable y.
      ++
      ++X a:Polynomial(Integer):=-3*x^3+2*x+13
      ++X rootsOf(a)
    rootsOf: SparseUnivariatePolynomial $ -> List $
      ++ rootsOf(p) returns \spad{[y1,...,yn]} such that \spad{p(yi) = 0}.
      ++ Note that the returned symbols y1,...,yn are bound in the interpreter
      ++ to respective root values.
      ++
      ++X a:SparseUnivariatePolynomial(Integer):=-3*x^3+2*x+13
      ++X rootsOf(a)
    rootsOf: (SparseUnivariatePolynomial $, Symbol) -> List $
      ++ rootsOf(p, y) returns \spad{[y1,...,yn]} such that \spad{p(yi) = 0};
```

```
    ++ The returned roots display as \spad{'y1},...,\spad{'yn}.
    ++ Note that the returned symbols y1,...,yn are bound in the interpreter
    ++ to respective root values.
    ++
    ++X a:SparseUnivariatePolynomial(Integer):=-3*x^3+2*x+13
    ++X rootsOf(a,x)
  zeroOf: Polynomial $ -> $
    ++ zeroOf(p) returns y such that \spad{p(y) = 0}.
    ++ If possible, y is expressed in terms of radicals.
    ++ Otherwise it is an implicit algebraic quantity.
    ++ Error: if p has more than one variable y.
    ++
    ++X a:Polynomial(Integer):=-3*x^2+2*x-13
    ++X zeroOf(a)
  zeroOf: SparseUnivariatePolynomial $ -> $
    ++ zeroOf(p) returns y such that \spad{p(y) = 0};
    ++ if possible, y is expressed in terms of radicals.
    ++ Otherwise it is an implicit algebraic quantity.
    ++
    ++X a:SparseUnivariatePolynomial(Integer):=-3*x^3+2*x+13
    ++X zeroOf(a)
  zeroOf: (SparseUnivariatePolynomial $, Symbol) -> $
    ++ zeroOf(p, y) returns y such that \spad{p(y) = 0};
    ++ if possible, y is expressed in terms of radicals.
    ++ Otherwise it is an implicit algebraic quantity which
    ++ displays as \spad{'y}.
    ++
    ++X a:SparseUnivariatePolynomial(Integer):=-3*x^3+2*x+13
    ++X zeroOf(a,x)
  zerosOf: Polynomial $ -> List $
    ++ zerosOf(p) returns \spad{[y1,...,yn]} such that \spad{p(yi) = 0}.
    ++ The yi's are expressed in radicals if possible.
    ++ Otherwise they are implicit algebraic quantities.
    ++ The returned symbols y1,...,yn are bound in the interpreter
    ++ to respective root values.
    ++ Error: if p has more than one variable y.
    ++
    ++X a:Polynomial(Integer):=-3*x^2+2*x-13
    ++X zerosOf(a)
  zerosOf: SparseUnivariatePolynomial $ -> List $
    ++ zerosOf(p) returns \spad{[y1,...,yn]} such that \spad{p(yi) = 0}.
    ++ The yi's are expressed in radicals if possible, and otherwise
    ++ as implicit algebraic quantities.
    ++ The returned symbols y1,...,yn are bound in the interpreter
    ++ to respective root values.
    ++
    ++X a:SparseUnivariatePolynomial(Integer):=-3*x^3+2*x+13
    ++X zerosOf(a)
  zerosOf: (SparseUnivariatePolynomial $, Symbol) -> List $
    ++ zerosOf(p, y) returns \spad{[y1,...,yn]} such that \spad{p(yi) = 0}.
```

```
     ++ The yi's are expressed in radicals if possible, and otherwise
     ++ as implicit algebraic quantities
     ++ which display as \spad{'yi}.
     ++ The returned symbols y1,...,yn are bound in the interpreter
     ++ to respective root values.
     ++
     ++X a:SparseUnivariatePolynomial(Integer):=-3*x^3+2*x+13
     ++X zerosOf(a,x)
add
   SUP ==> SparseUnivariatePolynomial $

   assign  : (Symbol, $) -> $
   allroots: (SUP, Symbol, (SUP, Symbol) -> $) -> List $
   binomialRoots: (SUP, Symbol, (SUP, Symbol) -> $) -> List $

   zeroOf(p:SUP)            == assign(x := new(), zeroOf(p, x))

   rootOf(p:SUP)            == assign(x := new(), rootOf(p, x))

   zerosOf(p:SUP)           == zerosOf(p, new())

   rootsOf(p:SUP)           == rootsOf(p, new())

   rootsOf(p:SUP, y:Symbol) == allroots(p, y, rootOf)

   zerosOf(p:SUP, y:Symbol) == allroots(p, y, zeroOf)

   assign(x, f)             == (assignSymbol(x, f, $)$Lisp; f)

   zeroOf(p:Polynomial $) ==
     empty?(l := variables p) => error "zeroOf: constant polynomial"
     zeroOf(univariate p, first l)

   rootOf(p:Polynomial $) ==
     empty?(l := variables p) => error "rootOf: constant polynomial"
     rootOf(univariate p, first l)

   zerosOf(p:Polynomial $) ==
     empty?(l := variables p) => error "zerosOf: constant polynomial"
     zerosOf(univariate p, first l)

   rootsOf(p:Polynomial $) ==
     empty?(l := variables p) => error "rootsOf: constant polynomial"
     rootsOf(univariate p, first l)

   zeroOf(p:SUP, y:Symbol) ==
     zero?(d := degree p) => error "zeroOf: constant polynomial"
     zero? coefficient(p, 0) => 0
     a := leadingCoefficient p
     d = 2 =>
```

```
        b := coefficient(p, 1)
        (sqrt(b**2 - 4 * a * coefficient(p, 0)) - b) / (2 * a)
      (r := retractIfCan(reductum p)@Union($,"failed")) case "failed" =>
        rootOf(p, y)
      nthRoot(- (r::$ / a), d)

    binomialRoots(p, y, fn) ==
     -- p = a * x**n + b
      alpha := assign(x := new(y)$Symbol, fn(p, x))
--      one?(n := degree p) =>  [ alpha ]
      ((n := degree p) = 1) =>  [ alpha ]
      cyclo := cyclotomic(n,monomial(1,1)$SUP)_
                    $NumberTheoreticPolynomialFunctions(SUP)
      beta := assign(x := new(y)$Symbol, fn(cyclo, x))
      [alpha*beta**i for i in 0..(n-1)::NonNegativeInteger]

    import PolynomialDecomposition(SUP,$)

    allroots(p, y, fn) ==
      zero? p => error "allroots: polynomial must be nonzero"
      zero? coefficient(p,0) =>
         concat(0, allroots(p quo monomial(1,1), y, fn))
      zero?(p1:=reductum p) => empty()
      zero? reductum p1 => binomialRoots(p, y, fn)
      decompList := decompose(p)
      # decompList > 1 =>
          h := last decompList
          g := leftFactor(p,h) :: SUP
          groots := allroots(g, y, fn)
          "append"/[allroots(h-r::SUP, y, fn) for r in groots]
      ans := nil()$List($)
      while not ground? p repeat
        alpha := assign(x := new(y)$Symbol, fn(p, x))
        q     := monomial(1, 1)$SUP - alpha::SUP
        if not zero?(p alpha) then
          p   := p quo q
          ans := concat(alpha, ans)
        else while zero?(p alpha) repeat
          p   := (p exquo q)::SUP
          ans := concat(alpha, ans)
      reverse_! ans
```

———-

— ACF.dotabb —

```
"ACF"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ACF"];
```

```
"ACF" -> "FIELD"
"ACF" -> "RADCAT"
```

———-

— ACF.dotfull —

```
"AlgebraicallyClosedField()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ACF"];
"AlgebraicallyClosedField()" -> "Field()"
"AlgebraicallyClosedField()" -> "RadicalCategory()"
```

———-

— ACF.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"AlgebraicallyClosedField()" [color=lightblue];
"AlgebraicallyClosedField()" -> "Field()"
"AlgebraicallyClosedField()" -> "RadicalCategory()"

"Field()" [color=lightblue];
"Field()" -> "EuclideanDomain()"
"Field()" -> "UniqueFactorizationDomain()"
"Field()" -> "DIVRING..."

"EuclideanDomain()" [color=lightblue];
"EuclideanDomain()" -> "PrincipalIdealDomain()"

"UniqueFactorizationDomain()" [color=lightblue];
"UniqueFactorizationDomain()" -> "GCDDOM..."

"PrincipalIdealDomain()" [color=lightblue];
"PrincipalIdealDomain()" -> "GCDDOM..."

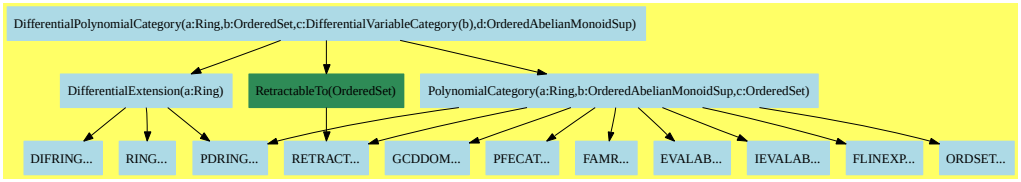
"DIVRING..." [color=lightblue];
"GCDDOM..." [color=lightblue];

"RadicalCategory()" [color=lightblue];
"RadicalCategory()" -> "Category"

"Category" [color=lightblue];
```

```
}
```

17.0.192 DifferentialPolynomialCategory (DPOLCAT)



— DifferentialPolynomialCategory.input —

```
)set break resume
)sys rm -f DifferentialPolynomialCategory.output
)spool DifferentialPolynomialCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show DifferentialPolynomialCategory
--R DifferentialPolynomialCategory(R: Ring,S: OrderedSet,V: DifferentialVariableCategory t#2,E:
--R Abbreviation for DifferentialPolynomialCategory is DPOLCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for DPOLCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,R) -> %                      ?*? : (R,%) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R D : (%,(R -> R)) -> %                 D : % -> % if R has DIFRING
--R D : (%,List V) -> %                   D : (%,V) -> %
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        coefficient : (%,E) -> R
--R coefficients : % -> List R            coerce : S -> %
--R coerce : V -> %                       coerce : R -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R degree : % -> E                       differentiate : (%,List V) -> %
--R differentiate : (%,V) -> %            eval : (%,List V,List %) -> %
```

```
--R eval : (%,V,%) -> %                  eval : (%,List V,List R) -> %
--R eval : (%,V,R) -> %                  eval : (%,List %,List %) -> %
--R eval : (%,%,%) -> %                  eval : (%,Equation %) -> %
--R eval : (%,List Equation %) -> %      ground : % -> R
--R ground? : % -> Boolean               hash : % -> SingleInteger
--R initial : % -> %                     isobaric? : % -> Boolean
--R latex : % -> String                  leader : % -> V
--R leadingCoefficient : % -> R          leadingMonomial : % -> %
--R map : ((R -> R),%) -> %              mapExponents : ((E -> E),%) -> %
--R minimumDegree : % -> E               monomial : (R,E) -> %
--R monomial? : % -> Boolean             monomials : % -> List %
--R one? : % -> Boolean                  order : % -> NonNegativeInteger
--R pomopo! : (%,R,E,%) -> %             primitiveMonomials : % -> List %
--R recip : % -> Union(%,"failed")       reductum : % -> %
--R retract : % -> S                     retract : % -> V
--R retract : % -> R                     sample : () -> %
--R separant : % -> %                    variables : % -> List V
--R weight : % -> NonNegativeInteger     zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (Fraction Integer,%) -> % if R has ALGEBRA FRAC INT
--R ?*? : (%,Fraction Integer) -> % if R has ALGEBRA FRAC INT
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (%,R) -> % if R has FIELD
--R ?<? : (%,%) -> Boolean if R has ORDSET
--R ?<=? : (%,%) -> Boolean if R has ORDSET
--R ?>? : (%,%) -> Boolean if R has ORDSET
--R ?>=? : (%,%) -> Boolean if R has ORDSET
--R D : (%,(R -> R),NonNegativeInteger) -> %
--R D : (%,List Symbol,List NonNegativeInteger) -> % if R has PDRING SYMBOL
--R D : (%,Symbol,NonNegativeInteger) -> % if R has PDRING SYMBOL
--R D : (%,List Symbol) -> % if R has PDRING SYMBOL
--R D : (%,Symbol) -> % if R has PDRING SYMBOL
--R D : (%,NonNegativeInteger) -> % if R has DIFRING
--R D : (%,List V,List NonNegativeInteger) -> %
--R D : (%,V,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R associates? : (%,%) -> Boolean if R has INTDOM
--R binomThmExpt : (%,%,NonNegativeInteger) -> % if R has COMRING
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if and(has($,CharacteristicNonZero),has(R,PolynomialFactorizationExp
--R coefficient : (%,List V,List NonNegativeInteger) -> %
--R coefficient : (%,V,NonNegativeInteger) -> %
--R coerce : % -> % if R has INTDOM
--R coerce : Fraction Integer -> % if R has RETRACT FRAC INT or R has ALGEBRA FRAC INT
--R conditionP : Matrix % -> Union(Vector %,"failed") if and(has($,CharacteristicNonZero),has(R,PolynomialFa
--R content : (%,V) -> % if R has GCDDOM
--R content : % -> R if R has GCDDOM
--R convert : % -> InputForm if V has KONVERT INFORM and R has KONVERT INFORM
--R convert : % -> Pattern Integer if V has KONVERT PATTERN INT and R has KONVERT PATTERN INT
```

```
--R convert : % -> Pattern Float if V has KONVERT PATTERN FLOAT and R has KONVERT PATTERN FLOAT
--R degree : (%,S) -> NonNegativeInteger
--R degree : (%,List V) -> List NonNegativeInteger
--R degree : (%,V) -> NonNegativeInteger
--R differentialVariables : % -> List S
--R differentiate : (%,(R -> R)) -> %
--R differentiate : (%,(R -> R),NonNegativeInteger) -> %
--R differentiate : (%,List Symbol,List NonNegativeInteger) -> % if R has PDRING SYMBOL
--R differentiate : (%,Symbol,NonNegativeInteger) -> % if R has PDRING SYMBOL
--R differentiate : (%,List Symbol) -> % if R has PDRING SYMBOL
--R differentiate : (%,Symbol) -> % if R has PDRING SYMBOL
--R differentiate : (%,NonNegativeInteger) -> % if R has DIFRING
--R differentiate : % -> % if R has DIFRING
--R differentiate : (%,List V,List NonNegativeInteger) -> %
--R differentiate : (%,V,NonNegativeInteger) -> %
--R discriminant : (%,V) -> % if R has COMRING
--R eval : (%,List S,List R) -> % if R has DIFRING
--R eval : (%,S,R) -> % if R has DIFRING
--R eval : (%,List S,List %) -> % if R has DIFRING
--R eval : (%,S,%) -> % if R has DIFRING
--R exquo : (%,%) -> Union(%,"failed") if R has INTDOM
--R exquo : (%,R) -> Union(%,"failed") if R has INTDOM
--R factor : % -> Factored % if R has PFECAT
--R factorPolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial % if
--R factorSquareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolyno
--R gcd : (%,%) -> % if R has GCDDOM
--R gcd : List % -> % if R has GCDDOM
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivari
--R isExpt : % -> Union(Record(var: V,exponent: NonNegativeInteger),"failed")
--R isPlus : % -> Union(List %,"failed")
--R isTimes : % -> Union(List %,"failed")
--R lcm : (%,%) -> % if R has GCDDOM
--R lcm : List % -> % if R has GCDDOM
--R mainVariable : % -> Union(V,"failed")
--R makeVariable : % -> (NonNegativeInteger -> %) if R has DIFRING
--R makeVariable : S -> (NonNegativeInteger -> %)
--R max : (%,%) -> % if R has ORDSET
--R min : (%,%) -> % if R has ORDSET
--R minimumDegree : (%,List V) -> List NonNegativeInteger
--R minimumDegree : (%,V) -> NonNegativeInteger
--R monicDivide : (%,%,V) -> Record(quotient: %,remainder: %)
--R monomial : (%,List V,List NonNegativeInteger) -> %
--R monomial : (%,V,NonNegativeInteger) -> %
--R multivariate : (SparseUnivariatePolynomial %,V) -> %
--R multivariate : (SparseUnivariatePolynomial R,V) -> %
--R numberOfMonomials : % -> NonNegativeInteger
--R order : (%,S) -> NonNegativeInteger
--R patternMatch : (%,Pattern Integer,PatternMatchResult(Integer,%)) -> PatternMatchResult(Integ
--R patternMatch : (%,Pattern Float,PatternMatchResult(Float,%)) -> PatternMatchResult(Float,%)
--R prime? : % -> Boolean if R has PFECAT
```

```
--R primitivePart : (%,V) -> % if R has GCDDOM
--R primitivePart : % -> % if R has GCDDOM
--R reducedSystem : Matrix % -> Matrix R
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix R,vec: Vector R)
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer) if R has LINEXP I
--R reducedSystem : Matrix % -> Matrix Integer if R has LINEXP INT
--R resultant : (%,%,V) -> % if R has COMRING
--R retract : % -> Integer if R has RETRACT INT
--R retract : % -> Fraction Integer if R has RETRACT FRAC INT
--R retractIfCan : % -> Union(S,"failed")
--R retractIfCan : % -> Union(V,"failed")
--R retractIfCan : % -> Union(Integer,"failed") if R has RETRACT INT
--R retractIfCan : % -> Union(Fraction Integer,"failed") if R has RETRACT FRAC INT
--R retractIfCan : % -> Union(R,"failed")
--R solveLinearPolynomialEquation : (List SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> Unio
--R squareFree : % -> Factored % if R has GCDDOM
--R squareFreePart : % -> % if R has GCDDOM
--R squareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial % if R has PI
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R totalDegree : (%,List V) -> NonNegativeInteger
--R totalDegree : % -> NonNegativeInteger
--R unit? : % -> Boolean if R has INTDOM
--R unitCanonical : % -> % if R has INTDOM
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %) if R has INTDOM
--R univariate : % -> SparseUnivariatePolynomial R
--R univariate : (%,V) -> SparseUnivariatePolynomial %
--R weight : (%,S) -> NonNegativeInteger
--R weights : (%,S) -> List NonNegativeInteger
--R weights : % -> List NonNegativeInteger
--R
--E 1

)spool
)lisp (bye)
```

———-

— DifferentialPolynomialCategory.help —

```
=====================================================================
DifferentialPolynomialCategory examples
=====================================================================

See Also:
o )show DifferentialPolynomialCategory
```

———-

See:

⇐ "DifferentialExtension" (DIFEXT) 11.0.153 on page 953
⇐ "PolynomialCategory" (POLYCAT) 16.0.189 on page 1255
⇐ "RetractableTo" (RETRACT) 2.0.18 on page 55

Exports:

0	1	associates?
binomThmExpt	characteristic	charthRoot
coefficient	coefficients	coerce
conditionP	content	convert
D	degree	differentialVariables
differentiate	discriminant	eval
exquo	factor	factorPolynomial
factorSquareFreePolynomial	gcd	gcdPolynomial
ground	ground?	hash
initial	isExpt	isobaric?
isPlus	isTimes	latex
lcm	leader	leadingCoefficient
leadingMonomial	makeVariable	map
mapExponents	max	min
minimumDegree	monicDivide	monomial
monomial?	monomials	multivariate
numberOfMonomials	one?	order
patternMatch	pomopo!	prime?
primitiveMonomials	primitivePart	recip
reducedSystem	reductum	resultant
retract	retractIfCan	sample
separant	solveLinearPolynomialEquation	squareFree
squareFreePart	squareFreePolynomial	subtractIfCan
totalDegree	unit?	unitCanonical
unitNormal	univariate	variables
weight	weights	zero?
?*?	?**?	?+?
?-?	-?	?=?
?^?	?~=?	?/?
?<?	?<=?	?>?
?>=?		

Attributes Exported:

-
- if R has canonicalUnitNormal then canonicalUnitNormal where **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.

- if $ has IntegralDomain then noZeroDivisors where **noZeroDivisors** is true if $x*y \neq 0$ implies both x and y are non-zero.
- if $ has CommutativeRing then commutative("*") where **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are implemented by this category:

```
coerce : S -> %
degree : (%,S) -> NonNegativeInteger
differentialVariables : % -> List S
differentiate : (%,(R -> R)) -> %
eval : (%,List S,List R) -> % if R has DIFRING
eval : (%,List S,List %) -> % if R has DIFRING
eval : (%,List Equation %) -> %
initial : % -> %
isobaric? : % -> Boolean
leader : % -> V
makeVariable : S -> (NonNegativeInteger -> %)
makeVariable : % -> (NonNegativeInteger -> %) if R has DIFRING
order : % -> NonNegativeInteger
order : (%,S) -> NonNegativeInteger
retractIfCan : % -> Union(S,"failed")
separant : % -> %
weight : % -> NonNegativeInteger
weight : (%,S) -> NonNegativeInteger
weights : (%,S) -> List NonNegativeInteger
weights : % -> List NonNegativeInteger
```

These exports come from (p1255) PolynomialCategory(R,E,V)
where R:Ring, E:OrderedAbelianMonoidSup,
V:DifferentialVariableCategory(S:OrderedSet):

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean if R has INTDOM
binomThmExpt : (%,%,NonNegativeInteger) -> % if R has COMRING
characteristic : () -> NonNegativeInteger
charthRoot : % -> Union(%,"failed")
  if and(has($,CharacteristicNonZero),
     has(R,PolynomialFactorizationExplicit))
  or R has CHARNZ
```

```
coefficient : (%,List V,List NonNegativeInteger) -> %
coefficient : (%,V,NonNegativeInteger) -> %
coefficient : (%,E) -> R
coefficients : % -> List R
coerce : R -> %
coerce : Fraction Integer -> %
  if R has RETRACT FRAC INT
  or R has ALGEBRA FRAC INT
coerce : % -> % if R has INTDOM
coerce : Integer -> %
coerce : % -> OutputForm
coerce : V -> %
conditionP : Matrix % -> Union(Vector %,"failed")
  if and(has($,CharacteristicNonZero),
         has(R,PolynomialFactorizationExplicit))
content : % -> R if R has GCDDOM
content : (%,V) -> % if R has GCDDOM
convert : % -> Pattern Integer
  if V has KONVERT PATTERN INT
  and R has KONVERT PATTERN INT
convert : % -> Pattern Float
  if V has KONVERT PATTERN FLOAT
  and R has KONVERT PATTERN FLOAT
convert : % -> InputForm
  if V has KONVERT INFORM
  and R has KONVERT INFORM
D : (%,List V) -> %
D : (%,V) -> %
D : (%,List V,List NonNegativeInteger) -> %
D : (%,V,NonNegativeInteger) -> %
degree : % -> E
degree : (%,List V) -> List NonNegativeInteger
degree : (%,V) -> NonNegativeInteger
differentiate : (%,V) -> %
differentiate : (%,List V,List NonNegativeInteger) -> %
differentiate : (%,V,NonNegativeInteger) -> %
differentiate : (%,List V) -> %
discriminant : (%,V) -> % if R has COMRING
eval : (%,Equation %) -> %
eval : (%,List %,List %) -> %
eval : (%,%,%) -> %
eval : (%,List V,List R) -> %
eval : (%,V,R) -> %
eval : (%,List V,List %) -> %
eval : (%,V,%) -> %
exquo : (%,%) -> Union(%,"failed") if R has INTDOM
exquo : (%,R) -> Union(%,"failed") if R has INTDOM
factor : % -> Factored % if R has PFECAT
factorPolynomial :
  SparseUnivariatePolynomial % ->
```

```
    Factored SparseUnivariatePolynomial %
      if R has PFECAT
factorSquareFreePolynomial :
  SparseUnivariatePolynomial % ->
    Factored SparseUnivariatePolynomial %
      if R has PFECAT
gcd : (%,%) -> % if R has GCDDOM
gcd : List % -> % if R has GCDDOM
gcdPolynomial :
  (SparseUnivariatePolynomial %,
   SparseUnivariatePolynomial %) ->
     SparseUnivariatePolynomial %
       if R has GCDDOM
ground : % -> R
ground? : % -> Boolean
hash : % -> SingleInteger
isExpt : % ->
  Union(Record(var: V,exponent: NonNegativeInteger),"failed")
isPlus : % -> Union(List %,"failed")
isTimes : % -> Union(List %,"failed")
latex : % -> String
lcm : (%,%) -> % if R has GCDDOM
lcm : List % -> % if R has GCDDOM
leadingCoefficient : % -> R
leadingMonomial : % -> %
mainVariable : % -> Union(V,"failed")
map : ((R -> R),%) -> %
mapExponents : ((E -> E),%) -> %
max : (%,%) -> % if R has ORDSET
min : (%,%) -> % if R has ORDSET
minimumDegree : % -> E
minimumDegree : (%,List V) -> List NonNegativeInteger
minimumDegree : (%,V) -> NonNegativeInteger
monicDivide : (%,%,V) -> Record(quotient: %,remainder: %)
monomial : (%,V,NonNegativeInteger) -> %
monomial : (%,List V,List NonNegativeInteger) -> %
monomial : (R,E) -> %
monomial? : % -> Boolean
monomials : % -> List %
multivariate : (SparseUnivariatePolynomial %,V) -> %
multivariate : (SparseUnivariatePolynomial R,V) -> %
numberOfMonomials : % -> NonNegativeInteger
one? : % -> Boolean
patternMatch :
  (%,Pattern Integer,PatternMatchResult(Integer,%)) ->
    PatternMatchResult(Integer,%)
      if V has PATMAB INT
      and R has PATMAB INT
patternMatch :
  (%,Pattern Float,PatternMatchResult(Float,%)) ->
```

```
    PatternMatchResult(Float,%)
      if V has PATMAB FLOAT
      and R has PATMAB FLOAT
pomopo! : (%,R,E,%) -> %
prime? : % -> Boolean if R has PFECAT
primitiveMonomials : % -> List %
primitivePart : (%,V) -> % if R has GCDDOM
primitivePart : % -> % if R has GCDDOM
recip : % -> Union(%,"failed")
reducedSystem : Matrix % -> Matrix R
reducedSystem :
  (Matrix %,Vector %) -> Record(mat: Matrix R,vec: Vector R)
reducedSystem :
  (Matrix %,Vector %) ->
    Record(mat: Matrix Integer,vec: Vector Integer)
      if R has LINEXP INT
reducedSystem : Matrix % -> Matrix Integer
    if R has LINEXP INT
reductum : % -> %
resultant : (%,%,V) -> % if R has COMRING
retract : % -> R
retract : % -> Integer if R has RETRACT INT
retract : % -> Fraction Integer if R has RETRACT FRAC INT
retract : % -> V
retractIfCan : % -> Union(R,"failed")
retractIfCan : % -> Union(Integer,"failed")
    if R has RETRACT INT
retractIfCan : % -> Union(Fraction Integer,"failed")
    if R has RETRACT FRAC INT
retractIfCan : % -> Union(V,"failed")
sample : () -> %
solveLinearPolynomialEquation :
  (List SparseUnivariatePolynomial %,
   SparseUnivariatePolynomial %) ->
    Union(List SparseUnivariatePolynomial %,"failed")
      if R has PFECAT
squareFree : % -> Factored % if R has GCDDOM
squareFreePart : % -> % if R has GCDDOM
squareFreePolynomial :
  SparseUnivariatePolynomial % ->
    Factored SparseUnivariatePolynomial %
      if R has PFECAT
subtractIfCan : (%,%) -> Union(%,"failed")
totalDegree : (%,List V) -> NonNegativeInteger
totalDegree : % -> NonNegativeInteger
unit? : % -> Boolean if R has INTDOM
unitCanonical : % -> % if R has INTDOM
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
    if R has INTDOM
univariate : % -> SparseUnivariatePolynomial R
```

```
univariate : (%,V) -> SparseUnivariatePolynomial %
variables : % -> List V
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,R) -> %
?*? : (R,%) -> %
?*? : (Fraction Integer,%) -> % if R has ALGEBRA FRAC INT
?*? : (%,Fraction Integer) -> % if R has ALGEBRA FRAC INT
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?/? : (%,R) -> % if R has FIELD
?-? : (%,%) -> %
-? : % -> %
?^? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?<? : (%,%) -> Boolean if R has ORDSET
?<=? : (%,%) -> Boolean if R has ORDSET
?>? : (%,%) -> Boolean if R has ORDSET
?>=? : (%,%) -> Boolean if R has ORDSET
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
```

These exports come from (p953) DifferentialExtension(R:Ring):

```
D : (%,(R -> R)) -> %
D : (%,(R -> R),NonNegativeInteger) -> %
D : % -> % if R has DIFRING
D : (%,NonNegativeInteger) -> % if R has DIFRING
D : (%,Symbol) -> % if R has PDRING SYMBOL
D : (%,List Symbol) -> % if R has PDRING SYMBOL
D : (%,Symbol,NonNegativeInteger) -> %
    if R has PDRING SYMBOL
D : (%,List Symbol,List NonNegativeInteger) -> %
    if R has PDRING SYMBOL
differentiate : (%,NonNegativeInteger) -> %
    if R has DIFRING
differentiate : (%,List Symbol) -> %
    if R has PDRING SYMBOL
differentiate : (%,Symbol,NonNegativeInteger) -> %
    if R has PDRING SYMBOL
differentiate : (%,List Symbol,List NonNegativeInteger) -> %
    if R has PDRING SYMBOL
differentiate : % -> % if R has DIFRING
differentiate : (%,(R -> R),NonNegativeInteger) -> %
differentiate : (%,Symbol) -> % if R has PDRING SYMBOL
```

These exports come from (p55) RetractableTo(S:OrderedSet):

```
retract : % -> S
```

These exports come from (p35) InnerEvalable(S,R)
where S:OrderedSet, R:Ring:

```
eval : (%,S,R) -> % if R has DIFRING
```

These exports come from (p35) InnerEvalable(S,where S:OrderedSet,

```
eval : (%,S,%) -> % if R has DIFRING
```

These exports come from (p80) Evalable(

— category DPOLCAT DifferentialPolynomialCategory —

```
)abbrev category DPOLCAT DifferentialPolynomialCategory
++ Author:  William Sit
++ Date Created: 19 July 1990
++ Date Last Updated: 13 September 1991
++ Basic Operations:PolynomialCategory
++ Related Constructors:DifferentialVariableCategory
++ See Also:
++ AMS Classifications:12H05
++ Keywords: differential indeterminates, ranking, differential polynomials,
++           order, weight, leader, separant, initial, isobaric
++ References:Kolchin, E.R. "Differential Algebra and Algebraic Groups"
++   (Academic Press, 1973).
++ Description:
++ \spadtype{DifferentialPolynomialCategory} is a category constructor
++ specifying basic functions in an ordinary differential polynomial
++ ring with a given ordered set of differential indeterminates.
++ In addition, it implements defaults for the basic functions.
++ The functions \spadfun{order} and \spadfun{weight} are extended
++ from the set of derivatives of differential indeterminates
++ to the set of differential polynomials.  Other operations
++ provided on differential polynomials are
++ \spadfun{leader}, \spadfun{initial},
++ \spadfun{separant}, \spadfun{differentialVariables}, and
++ \spadfun{isobaric?}.   Furthermore, if the ground ring is
++ a differential ring, then evaluation (substitution
++ of differential indeterminates by elements of the ground ring
++ or by differential polynomials) is
++ provided by \spadfun{eval}.
++ A convenient way of referencing derivatives is provided by
```

```
++ the functions \spadfun{makeVariable}.
++
++ To construct a domain using this constructor, one needs
++ to provide a ground ring R, an ordered set S of differential
++ indeterminates, a ranking V on the set of derivatives
++ of the differential indeterminates, and a set E of
++ exponents in bijection with the set of differential monomials
++ in the given differential indeterminates.

DifferentialPolynomialCategory(R:Ring,S:OrderedSet,
  V:DifferentialVariableCategory S, E:OrderedAbelianMonoidSup):
              Category ==
  Join(PolynomialCategory(R,E,V),
       DifferentialExtension R, RetractableTo S) with
    -- Examples:
    -- s:=makeVariable('s)
    -- p:= 3*(s 1)**2 + s*(s 2)**3
    --  all functions below have default implementations
    --  using primitives from V

    makeVariable: S -> (NonNegativeInteger -> $)
       ++ makeVariable(s) views s as a differential
       ++ indeterminate,  in such a way that the n-th
       ++ derivative of s may be simply referenced as z.n
       ++ where z :=makeVariable(s).
       ++ Note that In the interpreter, z is
       ++ given as an internal map, which may be ignored.
       -- Example: makeVariable('s); %.5

    differentialVariables: $ ->  List S
      ++ differentialVariables(p) returns a list of differential
      ++ indeterminates occurring in a differential polynomial p.
    order : ($, S) -> NonNegativeInteger
      ++ order(p,s) returns the order of the differential
      ++ polynomial p in differential indeterminate s.
    order : $   -> NonNegativeInteger
      ++ order(p) returns the order of the differential polynomial p,
      ++ which is the maximum number of differentiations of a
      ++ differential indeterminate, among all those appearing in p.
    degree: ($, S) -> NonNegativeInteger
      ++ degree(p, s) returns the maximum degree of
      ++ the differential polynomial p viewed as a differential polynomial
      ++ in the differential indeterminate s alone.
    weights: $ -> List NonNegativeInteger
      ++ weights(p) returns a list of weights of differential monomials
      ++ appearing in differential polynomial p.
    weight: $   -> NonNegativeInteger
      ++ weight(p) returns the maximum weight of all differential monomials
      ++ appearing in the differential polynomial p.
    weights: ($, S) -> List NonNegativeInteger
```

```
      ++ weights(p, s) returns a list of
      ++ weights of differential monomials
      ++ appearing in the differential polynomial p when p is viewed
      ++ as a differential polynomial in the differential indeterminate s
      ++ alone.
    weight: ($, S) -> NonNegativeInteger
      ++ weight(p, s) returns the maximum weight of all differential
      ++ monomials appearing in the differential polynomial p
      ++ when p is viewed as a differential polynomial in
      ++ the differential indeterminate s alone.
    isobaric?: $ -> Boolean
      ++ isobaric?(p) returns true if every differential monomial appearing
      ++ in the differential polynomial p has same weight,
      ++ and returns false otherwise.
    leader: $   -> V
      ++ leader(p) returns the derivative of the highest rank
      ++ appearing in the differential polynomial p
      ++ Note that an error occurs if p is in the ground ring.
    initial:$   -> $
      ++ initial(p) returns the
      ++ leading coefficient when the differential polynomial p
      ++ is written as a univariate polynomial in its leader.
    separant:$  -> $
      ++ separant(p) returns the
      ++ partial derivative of the differential polynomial p
      ++ with respect to its leader.
    if R has DifferentialRing then
      InnerEvalable(S, R)
      InnerEvalable(S, $)
      Evalable $
      makeVariable: $ -> (NonNegativeInteger -> $)
       ++ makeVariable(p) views p as an element of a differential
       ++ ring,  in such a way that the n-th
       ++ derivative of p may be simply referenced as z.n
       ++ where z := makeVariable(p).
       ++ Note that In the interpreter, z is
       ++ given as an internal map, which may be ignored.
       -- Example: makeVariable(p); %.5; makeVariable(%**2); %.2

  add
    p:$
    s:S

    makeVariable s == n +-> makeVariable(s,n)::$

    if R has IntegralDomain then
      differentiate(p:$, d:R -> R) ==
        ans:$ := 0
        l := variables p
        while (u:=retractIfCan(p)@Union(R, "failed")) case "failed" repeat
```

```
      t := leadingMonomial p
      lc := leadingCoefficient t
      ans := ans + d(lc)::$ * (t exquo lc)::$
          + +/[differentiate(t, v) * (differentiate v)::$ for v in l]
      p := reductum p
    ans + d(u::R)::$

order (p:$):NonNegativeInteger ==
  ground? p => 0
  "max"/[order v for v in variables p]

order (p:$,s:S):NonNegativeInteger ==
  ground? p => 0
  empty? (vv:= [order v for v in variables p | (variable v) = s ]) =>0
  "max"/vv

degree (p, s) ==
  d:NonNegativeInteger:=0
  for lp in monomials p repeat
    lv:= [v for v in variables lp | (variable v) = s ]
    if not empty? lv then d:= max(d, +/degree(lp, lv))
  d

weights p ==
  ws:List NonNegativeInteger := nil
  empty? (mp:=monomials p) => ws
  for lp in mp repeat
    lv:= variables lp
    if not empty? lv then
      dv:= degree(lp, lv)
      w:=+/[(weight v) * d _
             for v in lv for d in dv]$(List NonNegativeInteger)
      ws:= concat(ws, w)
  ws

weight p ==
  empty? (ws:=weights p) => 0
  "max"/ws

weights (p, s) ==
  ws:List NonNegativeInteger := nil
  empty?(mp:=monomials p) => ws
  for lp in mp repeat
    lv:= [v for v in variables lp | (variable v) = s ]
    if not empty? lv then
      dv:= degree(lp, lv)
      w:=+/[(weight v) * d _
           for v in lv for d in dv]$(List NonNegativeInteger)
      ws:= concat(ws, w)
  ws
```

```
weight (p,s)  ==
  empty? (ws:=weights(p,s)) => 0
  "max"/ws

isobaric? p == (# removeDuplicates weights p) = 1

leader p ==             -- depends on the ranking
  vl:= variables p
  -- it's not enough just to look at leadingMonomial p
  -- the term-ordering need not respect the ranking
  empty? vl => error "leader is not defined "
  "max"/vl

initial p == leadingCoefficient univariate(p,leader p)

separant p == differentiate(p, leader p)

coerce(s:S):$   == s::V::$

retractIfCan(p:$):Union(S, "failed") ==
  (v := retractIfCan(p)@Union(V,"failed")) case "failed" => "failed"
  retractIfCan(v::V)

differentialVariables p ==
  removeDuplicates [variable v for v in variables p]

if R has DifferentialRing then

  makeVariable p == n +-> differentiate(p, n)

  eval(p:$, sl:List S, rl:List R) ==
    ordp:= order p
    vl  := concat [[makeVariable(s,j)$V for j in  0..ordp]
                            for s in sl]$List(List V)
    rrl:=nil$List(R)
    for r in rl repeat
      t:= r
      rrl:= concat(rrl,
             concat(r, [t := differentiate t for i in 1..ordp]))
    eval(p, vl, rrl)

  eval(p:$, sl:List S, rl:List $) ==
    ordp:= order p
    vl  := concat [[makeVariable(s,j)$V for j in  0..ordp]
                            for s in sl]$List(List V)
    rrl:=nil$List($)
    for r in rl repeat
      t:=r
      rrl:=concat(rrl,
```

```
              concat(r, [t:=differentiate t for i in 1..ordp]))
        eval(p, vl, rrl)

      eval(p:$, l:List Equation $) ==
        eval(p, [retract(lhs e)@S for e in l]$List(S),
              [rhs e for e in l]$List($))
```

———-

— **DPOLCAT.dotabb** —

```
"DPOLCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=DPOLCAT"];
"DPOLCAT" -> "DIFEXT"
"DPOLCAT" -> "POLYCAT"
"DPOLCAT" -> "RETRACT"
```

———-

— **DPOLCAT.dotfull** —

```
"DifferentialPolynomialCategory(a:Ring,b:OrderedSet,c:DifferentialVariableCategory(b),d:OrderedAbelianMonoi
 [color=lightblue,href="bookvol10.2.pdf#nameddest=DPOLCAT"];
"DifferentialPolynomialCategory(a:Ring,b:OrderedSet,c:DifferentialVariableCategory(b),d:OrderedAbelianMonoi
 -> "DifferentialExtension(a:Ring)"
"DifferentialPolynomialCategory(a:Ring,b:OrderedSet,c:DifferentialVariableCategory(b),d:OrderedAbelianMonoi
 -> "PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
"DifferentialPolynomialCategory(a:Ring,b:OrderedSet,c:DifferentialVariableCategory(b),d:OrderedAbelianMonoi
 -> "RetractableTo(OrderedSet)"
```

———-

— **DPOLCAT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"DifferentialPolynomialCategory(a:Ring,b:OrderedSet,c:DifferentialVariableCategory(b),d:OrderedAbelianMonoi
 [color=lightblue];

"DifferentialPolynomialCategory(a:Ring,b:OrderedSet,c:DifferentialVariableCategory(b),d:OrderedAbelianMonoi
 -> "DifferentialExtension(a:Ring)"
```

```
"DifferentialPolynomialCategory(a:Ring,b:OrderedSet,c:DifferentialVariableCategory(b),d:OrderedA
 -> "PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"

"DifferentialPolynomialCategory(a:Ring,b:OrderedSet,c:DifferentialVariableCategory(b),d:OrderedA
 -> "RetractableTo(OrderedSet)"

"DifferentialExtension(a:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=DIFEXT"];
"DifferentialExtension(a:Ring)" -> "RING..."
"DifferentialExtension(a:Ring)" -> "DIFRING..."
"DifferentialExtension(a:Ring)" -> "PDRING..."

"RetractableTo(OrderedSet)"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=RETRACT"];
"RetractableTo(OrderedSet)" -> "RETRACT..."

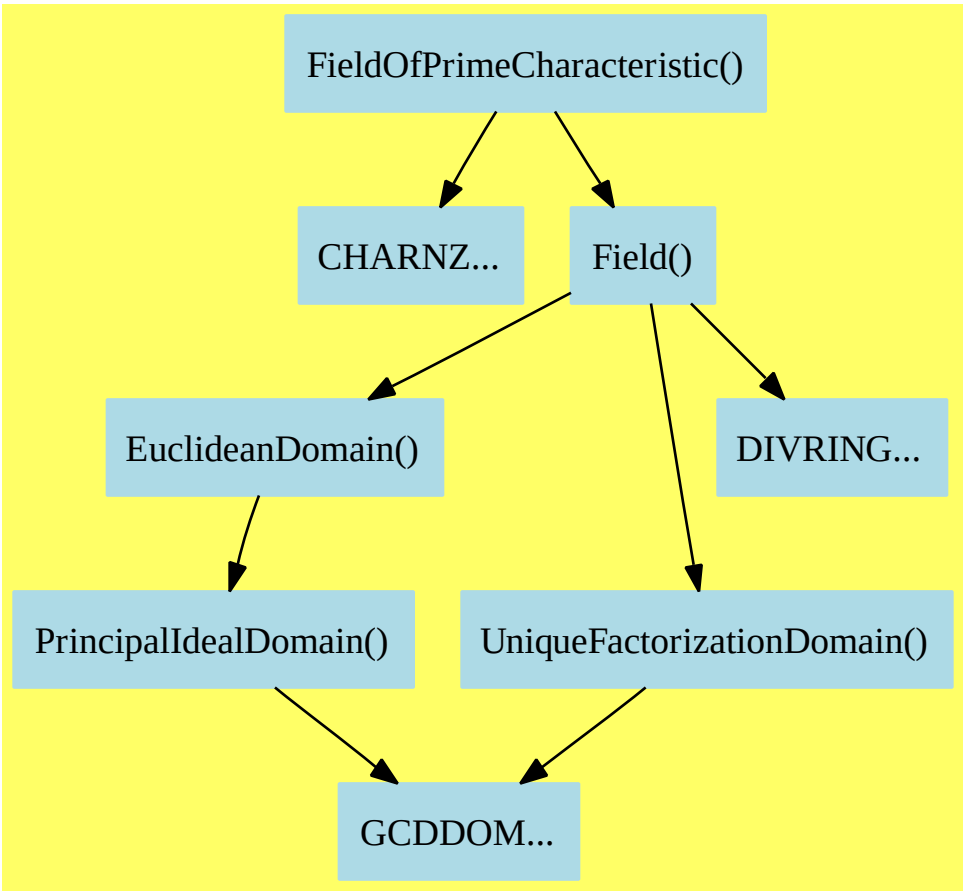
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=POLYCAT"];
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "PDRING..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "FAMR..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "EVALAB..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "IEVALAB..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "RETRACT..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "FLINEXP..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "ORDSET..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "GCDDOM..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "PFECAT..."

"DIFRING..." [color=lightblue];
"RING..." [color=lightblue];
"PDRING..." [color=lightblue];
"FAMR..." [color=lightblue];
"EVALAB..." [color=lightblue];
"IEVALAB..." [color=lightblue];
"RETRACT..." [color=lightblue];
"FLINEXP..." [color=lightblue];
"ORDSET..." [color=lightblue];
"GCDDOM..." [color=lightblue];
"PFECAT..." [color=lightblue];

}
```

———-

17.0.193 FieldOfPrimeCharacteristic (FPC)



— **FieldOfPrimeCharacteristic.input** —

```
)set break resume
)sys rm -f FieldOfPrimeCharacteristic.output
)spool FieldOfPrimeCharacteristic.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 1
)show FieldOfPrimeCharacteristic
--R FieldOfPrimeCharacteristic  is a category constructor
--R Abbreviation for FieldOfPrimeCharacteristic is FPC
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FPC
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (Fraction Integer,%) -> %       ?*? : (%,Fraction Integer) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,Integer) -> %
--R ?**? : (%,PositiveInteger) -> %       ?+? : (%,%) -> %
--R ?-? : (%,%) -> %                      -? : % -> %
--R ?/? : (%,%) -> %                      ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,Integer) -> %                ?^? : (%,PositiveInteger) -> %
--R associates? : (%,%) -> Boolean        coerce : Fraction Integer -> %
--R coerce : % -> %                       coerce : Integer -> %
--R coerce : % -> OutputForm              factor : % -> Factored %
--R gcd : List % -> %                     gcd : (%,%) -> %
--R hash : % -> SingleInteger             inv : % -> %
--R latex : % -> String                   lcm : List % -> %
--R lcm : (%,%) -> %                      one? : % -> Boolean
--R prime? : % -> Boolean                 primeFrobenius : % -> %
--R ?quo? : (%,%) -> %                    recip : % -> Union(%,"failed")
--R ?rem? : (%,%) -> %                    sample : () -> %
--R sizeLess? : (%,%) -> Boolean          squareFree : % -> Factored %
--R squareFreePart : % -> %               unit? : % -> Boolean
--R unitCanonical : % -> %                zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed")
--R discreteLog : (%,%) -> Union(NonNegativeInteger,"failed")
--R divide : (%,%) -> Record(quotient: %,remainder: %)
--R euclideanSize : % -> NonNegativeInteger
--R expressIdealMember : (List %,%) -> Union(List %,"failed")
--R exquo : (%,%) -> Union(%,"failed")
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivari
--R multiEuclidean : (List %,%) -> Union(List %,"failed")
--R order : % -> OnePointCompletion PositiveInteger
--R primeFrobenius : (%,NonNegativeInteger) -> %
--R principalIdeal : List % -> Record(coef: List %,generator: %)
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
```

```
--R
--E 1

)spool
)lisp (bye)
```

———-

— **FieldOfPrimeCharacteristic.help** —

```
=====================================================================
FieldOfPrimeCharacteristic examples
=====================================================================

See Also:
o )show FieldOfPrimeCharacteristic
```

———-

See:

⇒ “FiniteFieldCategory” (FFIELDC) 18.0.207 on page 1518
⇐ “CharacteristicNonZero” (CHARNZ) 10.0.135 on page 833
⇐ “Field” (FIELD) 16.0.185 on page 1221

Exports:

0	1	associates?	characteristic
charthRoot	coerce	discreteLog	divide
euclideanSize	expressIdealMember	exquo	extendedEuclidean
factor	gcd	gcdPolynomial	hash
inv	latex	lcm	multiEuclidean
one?	order	prime?	primeFrobenius
principalIdeal	recip	sample	sizeLess?
squareFree	squareFreePart	subtractIfCan	unit?
unitCanonical	unitNormal	zero?	?*?
?**?	?+?	?-?	-?
?/?	?=?	?^?	?rem?
?quo?	?~=?		

Attributes Exported:

- **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.

- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
order : % -> OnePointCompletion PositiveInteger
discreteLog : (%,%) -> Union(NonNegativeInteger,"failed")
```

These are implemented by this category:

```
primeFrobenius : % -> %
primeFrobenius : (%,NonNegativeInteger) -> %
```

These exports come from (p1221) Field():

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean
characteristic : () -> NonNegativeInteger
coerce : % -> %
coerce : Integer -> %
coerce : % -> OutputForm
coerce : Fraction Integer -> %
divide : (%,%) -> Record(quotient: %,remainder: %)
euclideanSize : % -> NonNegativeInteger
expressIdealMember : (List %,%) -> Union(List %,"failed")
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
exquo : (%,%) -> Union(%,"failed")
factor : % -> Factored %
gcd : (%,%) -> %
gcd : List % -> %
gcdPolynomial : (SparseUnivariatePolynomial %,
                 SparseUnivariatePolynomial %) ->
                     SparseUnivariatePolynomial %
hash : % -> SingleInteger
inv : % -> %
latex : % -> String
lcm : List % -> %
lcm : (%,%) -> %
multiEuclidean : (List %,%) -> Union(List %,"failed")
```

```
one? : % -> Boolean
prime? : % -> Boolean
principalIdeal : List % -> Record(coef: List %,generator: %)
recip : % -> Union(%,"failed")
sample : () -> %
sizeLess? : (%,%) -> Boolean
squareFree : % -> Factored %
squareFreePart : % -> %
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (Fraction Integer,%) -> %
?*? : (%,Fraction Integer) -> %
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,Integer) -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?^? : (%,Integer) -> %
?/? : (%,%) -> %
?quo? : (%,%) -> %
?rem? : (%,%) -> %
```

These exports come from (p833) CharacteristicNonZero():

```
charthRoot : % -> Union(%,"failed")
```

— category FPC FieldOfPrimeCharacteristic —

```
)abbrev category FPC FieldOfPrimeCharacteristic
++ Author: J. Grabmeier, A. Scheerhorn
++ Date Created: 10 March 1991
++ Date Last Updated: 31 March 1991
++ Basic Operations: _+, _*
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords: field, finite field, prime characteristic
```

```
++ References:
++  J. Grabmeier, A. Scheerhorn: Finite Fields in AXIOM.
++  AXIOM Technical Report Series, ATR/5 NP2522.
++ Description:
++ FieldOfPrimeCharacteristic is the category of fields of prime
++ characteristic, e.g. finite fields, algebraic closures of
++ fields of prime characteristic, transcendental extensions of
++ of fields of prime characteristic.

FieldOfPrimeCharacteristic:Category == _
 Join(Field,CharacteristicNonZero) with
   order: $ -> OnePointCompletion PositiveInteger
     ++ order(a) computes the order of an element in the multiplicative
     ++ group of the field.
     ++ Error: if \spad{a} is 0.
   discreteLog: ($,$) -> Union(NonNegativeInteger,"failed")
     ++ discreteLog(b,a) computes s with \spad{b**s = a} if such an s exists.
   primeFrobenius: $ -> $
     ++ primeFrobenius(a) returns \spad{a**p} where p is the characteristic.
   primeFrobenius: ($,NonNegativeInteger) -> $
     ++ primeFrobenius(a,s) returns \spad{a**(p**s)} where p
     ++ is the characteristic.
 add
   primeFrobenius(a) == a ** characteristic()
   primeFrobenius(a,s) == a ** (characteristic()**s)
```

———-

— FPC.dotabb —

```
"FPC"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FPC"];
"FPC" -> "CHARNZ"
"FPC" -> "FIELD"
```

———-

— FPC.dotfull —

```
"FieldOfPrimeCharacteristic()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FPC"];
"FieldOfPrimeCharacteristic()" -> "CharacteristicNonZero()"
```

———-

— FPC.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FieldOfPrimeCharacteristic()" [color=lightblue];
"FieldOfPrimeCharacteristic()" -> "CHARNZ..."
"FieldOfPrimeCharacteristic()" -> "Field()"

"Field()" [color=lightblue];
"Field()" -> "EuclideanDomain()"
"Field()" -> "UniqueFactorizationDomain()"
"Field()" -> "DIVRING..."

"EuclideanDomain()" [color=lightblue];
"EuclideanDomain()" -> "PrincipalIdealDomain()"

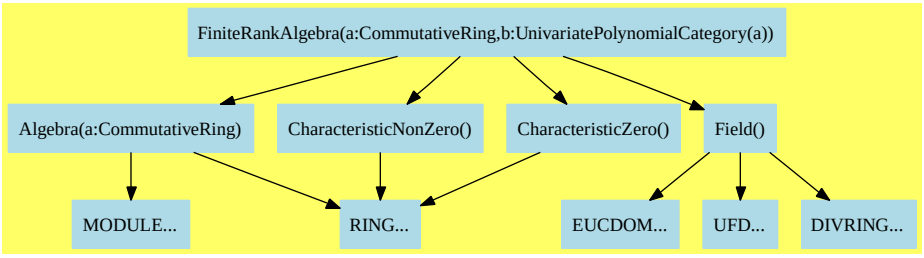
"UniqueFactorizationDomain()" [color=lightblue];
"UniqueFactorizationDomain()" -> "GCDDOM..."

"PrincipalIdealDomain()" [color=lightblue];
"PrincipalIdealDomain()" -> "GCDDOM..."

"GCDDOM..." [color=lightblue];
"DIVRING..." [color=lightblue];
"CHARNZ..." [color=lightblue];
}
```

———-

17.0.194 FiniteRankAlgebra (FINRALG)



— FiniteRankAlgebra.input —

```
)set break resume
```

```
)sys rm -f FiniteRankAlgebra.output
)spool FiniteRankAlgebra.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FiniteRankAlgebra
--R FiniteRankAlgebra(R: CommutativeRing,UP: UnivariatePolynomialCategory t#1)  is a category co
--R Abbreviation for FiniteRankAlgebra is FINRALG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FINRALG
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (R,%) -> %                      ?*? : (%,R) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        coerce : R -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R discriminant : Vector % -> R          hash : % -> SingleInteger
--R latex : % -> String                   norm : % -> R
--R one? : % -> Boolean                   rank : () -> PositiveInteger
--R recip : % -> Union(%,"failed")        sample : () -> %
--R trace : % -> R                        zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R characteristicPolynomial : % -> UP
--R charthRoot : % -> Union(%,"failed") if R has CHARNZ
--R coordinates : (Vector %,Vector %) -> Matrix R
--R coordinates : (%,Vector %) -> Vector R
--R minimalPolynomial : % -> UP if R has FIELD
--R regularRepresentation : (%,Vector %) -> Matrix R
--R represents : (Vector R,Vector %) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R traceMatrix : Vector % -> Matrix R
--R
--E 1

)spool
)lisp (bye)
```

— **FiniteRankAlgebra.help** —

```
==================================================================
FiniteRankAlgebra examples
==================================================================

See Also:
o )show FiniteRankAlgebra
```

———-

See:

⇒ “FramedAlgebra” (FRAMALG) 18.0.209 on page 1539
⇐ “Algebra” (ALGEBRA) 11.0.152 on page 947
⇐ “CharacteristicNonZero” (CHARNZ) 10.0.135 on page 833
⇐ “CharacteristicZero” (CHARZ) 10.0.136 on page 838
⇐ “Field” (FIELD) 16.0.185 on page 1221

Exports:

0	1	characteristic
characteristicPolynomial	charthRoot	coerce
coordinates	discriminant	hash
latex	minimalPolynomial	norm
one?	rank	recip
regularRepresentation	represents	sample
subtractIfCan	trace	traceMatrix
zero?	?*?	?**?
?+?	?-?	-?
?=?	?^?	?~=?

Attributes Exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return “failed” if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
characteristicPolynomial : % -> UP
coordinates : (%,Vector %) -> Vector R
minimalPolynomial : % -> UP if R has FIELD
norm : % -> R
rank : () -> PositiveInteger
trace : % -> R
```

These are implemented by this category:

```
coordinates : (Vector %,Vector %) -> Matrix R
discriminant : Vector % -> R
regularRepresentation : (%,Vector %) -> Matrix R
represents : (Vector R,Vector %) -> %
traceMatrix : Vector % -> Matrix R
```

These exports come from (p947) Algebra(R:CommutativeRing):

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : Integer -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
```

These exports come from (p1221) Field():

```
coerce : R -> %
?*? : (R,%) -> %
?*? : (%,R) -> %
```

These exports come from (p833) CharacteristicNonZero():

```
charthRoot : % -> Union(%,"failed") if R has CHARNZ
```

These exports come from (p838) CharacteristicZero():

— category FINRALG FiniteRankAlgebra —

```
)abbrev category FINRALG FiniteRankAlgebra
++ Author: Barry Trager
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A FiniteRankAlgebra is an algebra over a commutative ring R which
++ is a free R-module of finite rank.

FiniteRankAlgebra(R:CommutativeRing, UP:UnivariatePolynomialCategory R):
 Category == Algebra R with
    rank                    : () -> PositiveInteger
      ++ rank() returns the rank of the algebra.
    regularRepresentation   : (% , Vector %) -> Matrix R
      ++ regularRepresentation(a,basis) returns the matrix of the
      ++ linear map defined by left multiplication by \spad{a} with respect
      ++ to the basis \spad{basis}.
    trace                   : %  -> R
      ++ trace(a) returns the trace of the regular representation
      ++ of \spad{a} with respect to any basis.
    norm                    : %  -> R
      ++ norm(a) returns the determinant of the regular representation
      ++ of \spad{a} with respect to any basis.
    coordinates             : (%, Vector %) -> Vector R
      ++ coordinates(a,basis) returns the coordinates of \spad{a} with
      ++ respect to the basis \spad{basis}.
    coordinates             : (Vector %, Vector %) -> Matrix R
      ++ coordinates([v1,...,vm], basis) returns the coordinates of the
      ++ vi's with to the basis \spad{basis}.  The coordinates of vi are
      ++ contained in the ith row of the matrix returned by this
      ++ function.
    represents              : (Vector R, Vector %) -> %
      ++ represents([a1,..,an],[v1,..,vn]) returns \spad{a1*v1+...+an*vn}.
    discriminant            : Vector % -> R
      ++ discriminant([v1,..,vn]) returns
      ++ \spad{determinant(traceMatrix([v1,..,vn]))}.
    traceMatrix             : Vector % -> Matrix R
      ++ traceMatrix([v1,..,vn]) is the n-by-n matrix ( Tr(vi * vj) )
    characteristicPolynomial: % -> UP
      ++ characteristicPolynomial(a) returns the characteristic
      ++ polynomial of the regular representation of \spad{a} with respect
      ++ to any basis.
```

```
    if R has Field then minimalPolynomial : % -> UP
      ++ minimalPolynomial(a) returns the minimal polynomial of \spad{a}.
    if R has CharacteristicZero then CharacteristicZero
    if R has CharacteristicNonZero then CharacteristicNonZero

  add

    discriminant v == determinant traceMatrix v

    coordinates(v:Vector %, b:Vector %) ==
      m := new(#v, #b, 0)$Matrix(R)
      for i in minIndex v .. maxIndex v for j in minRowIndex m .. repeat
        setRow_!(m, j, coordinates(qelt(v, i), b))
      m

    represents(v, b) ==
      m := minIndex v - 1
      _+/[v(i+m) * b(i+m) for i in 1..rank()]

    traceMatrix v ==
      matrix [[trace(v.i*v.j) for j in minIndex v..maxIndex v]$List(R)
               for i in minIndex v .. maxIndex v]$List(List R)

    regularRepresentation(x, b) ==
      m := minIndex b - 1
      matrix
       [parts coordinates(x*b(i+m),b) for i in 1..rank()]$List(List R)
```

———-

— FINRALG.dotabb —

```
"FINRALG"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FINRALG"];
"FINRALG" -> "ALGEBRA"
"FINRALG" -> "FIELD"
"FINRALG" -> "CHARNZ"
"FINRALG" -> "CHARZ"
```

———-

— FINRALG.dotfull —

```
"FiniteRankAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FINRALG"];
"FiniteRankAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
```

```
    "Algebra(a:CommutativeRing)"
"FiniteRankAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "Field()"
"FiniteRankAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "CharacteristicNonZero()"
"FiniteRankAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "CharacteristicZero()"
```

———-

— FINRALG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FiniteRankAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))"
 [color=lightblue];
"FiniteRankAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "Algebra(a:CommutativeRing)"
"FiniteRankAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "Field()"
"FiniteRankAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "CharacteristicNonZero()"
"FiniteRankAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "CharacteristicZero()"

"Algebra(a:CommutativeRing)" [color=lightblue];
"Algebra(a:CommutativeRing)" -> "RING..."
"Algebra(a:CommutativeRing)" -> "MODULE..."

"Field()" [color=lightblue];
"Field()" -> "EUCDOM..."
"Field()" -> "UFD..."
"Field()" -> "DIVRING..."

"CharacteristicNonZero()" [color=lightblue];
"CharacteristicNonZero()" -> "RING..."

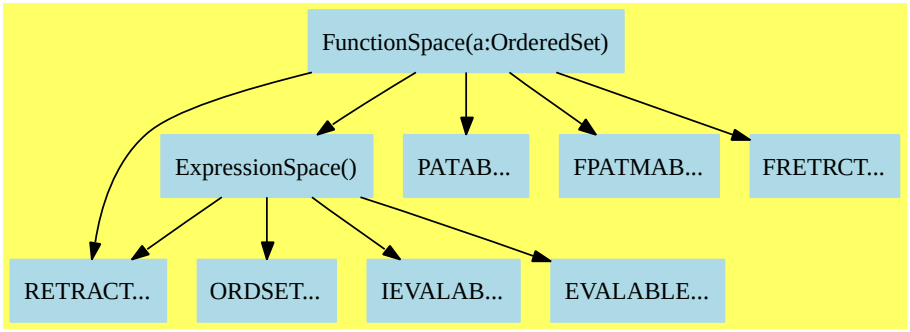
"CharacteristicZero()" [color=lightblue];
"CharacteristicZero()" -> "RING..."

"EUCDOM..." [color=lightblue];
"UFD..." [color=lightblue];
"DIVRING..." [color=lightblue];
"RING..." [color=lightblue];
"MODULE..." [color=lightblue];
```

```
}
```

———-

17.0.195 FunctionSpace (FS)



— **FunctionSpace.input** —

```
)set break resume
)sys rm -f FunctionSpace.output
)spool FunctionSpace.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FunctionSpace
--R FunctionSpace R: OrderedSet  is a category constructor
--R Abbreviation for FunctionSpace is FS
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FS
--R
--R------------------------------ Operations ------------------------------
--R -? : % -> % if R has ABELGRP          ?<? : (%,%) -> Boolean
--R ?<=? : (%,%) -> Boolean               ?=? : (%,%) -> Boolean
--R ?>? : (%,%) -> Boolean                ?>=? : (%,%) -> Boolean
--R 1 : () -> % if R has SGROUP           0 : () -> % if R has ABELSG
--R applyQuote : (Symbol,%,%) -> %        applyQuote : (Symbol,%) -> %
--R belong? : BasicOperator -> Boolean    box : List % -> %
--R box : % -> %                          coerce : R -> %
--R coerce : Symbol -> %                  coerce : Kernel % -> %
--R coerce : % -> OutputForm              distribute : (%,%) -> %
```

```
--R distribute : % -> %                   elt : (BasicOperator,%,%) -> %
--R elt : (BasicOperator,%) -> %          eval : (%,List %,List %) -> %
--R eval : (%,%,%) -> %                   eval : (%,Equation %) -> %
--R eval : (%,List Equation %) -> %       eval : (%,Kernel %,%) -> %
--R freeOf? : (%,Symbol) -> Boolean       freeOf? : (%,%) -> Boolean
--R ground : % -> R                       ground? : % -> Boolean
--R hash : % -> SingleInteger             height : % -> NonNegativeInteger
--R is? : (%,Symbol) -> Boolean           kernel : (BasicOperator,%) -> %
--R kernels : % -> List Kernel %          latex : % -> String
--R map : ((% -> %),Kernel %) -> %        max : (%,%) -> %
--R min : (%,%) -> %                      paren : List % -> %
--R paren : % -> %                        retract : % -> R
--R retract : % -> Symbol                 retract : % -> Kernel %
--R subst : (%,Equation %) -> %           tower : % -> List Kernel %
--R variables : % -> List Symbol          ?~=? : (%,%) -> Boolean
--R ?*? : (%,%) -> % if R has SGROUP
--R ?*? : (PositiveInteger,%) -> % if R has ABELSG
--R ?*? : (NonNegativeInteger,%) -> % if R has ABELSG
--R ?*? : (Integer,%) -> % if R has ABELGRP
--R ?*? : (%,R) -> % if R has COMRING
--R ?*? : (R,%) -> % if R has COMRING
--R ?*? : (%,Fraction Integer) -> % if R has INTDOM
--R ?*? : (Fraction Integer,%) -> % if R has INTDOM
--R ?**? : (%,PositiveInteger) -> % if R has SGROUP
--R ?**? : (%,NonNegativeInteger) -> % if R has SGROUP
--R ?**? : (%,Integer) -> % if R has GROUP or R has INTDOM
--R ?+? : (%,%) -> % if R has ABELSG
--R ?-? : (%,%) -> % if R has ABELGRP
--R ?/? : (%,%) -> % if R has GROUP or R has INTDOM
--R ?/? : (SparseMultivariatePolynomial(R,Kernel %),SparseMultivariatePolynomial(R,Kernel %)) -> % if R has
--R D : (%,Symbol) -> % if R has RING
--R D : (%,List Symbol) -> % if R has RING
--R D : (%,Symbol,NonNegativeInteger) -> % if R has RING
--R D : (%,List Symbol,List NonNegativeInteger) -> % if R has RING
--R ?^? : (%,PositiveInteger) -> % if R has SGROUP
--R ?^? : (%,NonNegativeInteger) -> % if R has SGROUP
--R ?^? : (%,Integer) -> % if R has GROUP or R has INTDOM
--R applyQuote : (Symbol,List %) -> %
--R applyQuote : (Symbol,%,%,%,%) -> %
--R applyQuote : (Symbol,%,%,%) -> %
--R associates? : (%,%) -> Boolean if R has INTDOM
--R characteristic : () -> NonNegativeInteger if R has RING
--R charthRoot : % -> Union(%,"failed") if R has CHARNZ
--R coerce : Integer -> % if R has RING or R has RETRACT INT
--R coerce : Fraction Integer -> % if R has INTDOM or R has RETRACT INT and R has INTDOM or R has RETRACT FI
--R coerce : Polynomial R -> % if R has RING
--R coerce : % -> % if R has INTDOM
--R coerce : Fraction Polynomial R -> % if R has INTDOM
--R coerce : Fraction Polynomial Fraction R -> % if R has INTDOM
--R coerce : Polynomial Fraction R -> % if R has INTDOM
```

```
--R coerce : Fraction R -> % if R has INTDOM
--R coerce : SparseMultivariatePolynomial(R,Kernel %) -> % if R has RING
--R commutator : (%,%) -> % if R has GROUP
--R conjugate : (%,%) -> % if R has GROUP
--R convert : % -> InputForm if R has KONVERT INFORM
--R convert : Factored % -> % if R has INTDOM
--R convert : % -> Pattern Float if R has KONVERT PATTERN FLOAT
--R convert : % -> Pattern Integer if R has KONVERT PATTERN INT
--R definingPolynomial : % -> % if $ has RING
--R denom : % -> SparseMultivariatePolynomial(R,Kernel %) if R has INTDOM
--R denominator : % -> % if R has INTDOM
--R differentiate : (%,Symbol) -> % if R has RING
--R differentiate : (%,List Symbol) -> % if R has RING
--R differentiate : (%,Symbol,NonNegativeInteger) -> % if R has RING
--R differentiate : (%,List Symbol,List NonNegativeInteger) -> % if R has RING
--R divide : (%,%) -> Record(quotient: %,remainder: %) if R has INTDOM
--R elt : (BasicOperator,List %) -> %
--R elt : (BasicOperator,%,%,%,%) -> %
--R elt : (BasicOperator,%,%,%) -> %
--R euclideanSize : % -> NonNegativeInteger if R has INTDOM
--R eval : (%,Symbol,NonNegativeInteger,(% -> %)) -> % if R has RING
--R eval : (%,Symbol,NonNegativeInteger,(List % -> %)) -> % if R has RING
--R eval : (%,List Symbol,List NonNegativeInteger,List (List % -> %)) -> % if R has RING
--R eval : (%,List Symbol,List NonNegativeInteger,List (% -> %)) -> % if R has RING
--R eval : (%,List BasicOperator,List %,Symbol) -> % if R has KONVERT INFORM
--R eval : (%,BasicOperator,%,Symbol) -> % if R has KONVERT INFORM
--R eval : % -> % if R has KONVERT INFORM
--R eval : (%,List Symbol) -> % if R has KONVERT INFORM
--R eval : (%,Symbol) -> % if R has KONVERT INFORM
--R eval : (%,BasicOperator,(% -> %)) -> %
--R eval : (%,BasicOperator,(List % -> %)) -> %
--R eval : (%,List BasicOperator,List (List % -> %)) -> %
--R eval : (%,List BasicOperator,List (% -> %)) -> %
--R eval : (%,Symbol,(% -> %)) -> %
--R eval : (%,Symbol,(List % -> %)) -> %
--R eval : (%,List Symbol,List (List % -> %)) -> %
--R eval : (%,List Symbol,List (% -> %)) -> %
--R eval : (%,List Kernel %,List %) -> %
--R even? : % -> Boolean if $ has RETRACT INT
--R expressIdealMember : (List %,%) -> Union(List %,"failed") if R has INTDOM
--R exquo : (%,%) -> Union(%,"failed") if R has INTDOM
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %) if R has INTDOM
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed") if R has INTDOM
--R factor : % -> Factored % if R has INTDOM
--R gcd : (%,%) -> % if R has INTDOM
--R gcd : List % -> % if R has INTDOM
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivari
--R inv : % -> % if R has GROUP or R has INTDOM
--R is? : (%,BasicOperator) -> Boolean
--R isExpt : (%,Symbol) -> Union(Record(var: Kernel %,exponent: Integer),"failed") if R has RING
```

```
--R isExpt : (%,BasicOperator) -> Union(Record(var: Kernel %,exponent: Integer),"failed") if R has RING
--R isExpt : % -> Union(Record(var: Kernel %,exponent: Integer),"failed") if R has SGROUP
--R isMult : % -> Union(Record(coef: Integer,var: Kernel %),"failed") if R has ABELSG
--R isPlus : % -> Union(List %,"failed") if R has ABELSG
--R isPower : % -> Union(Record(val: %,exponent: Integer),"failed") if R has RING
--R isTimes : % -> Union(List %,"failed") if R has SGROUP
--R kernel : (BasicOperator,List %) -> %
--R lcm : (%,%) -> % if R has INTDOM
--R lcm : List % -> % if R has INTDOM
--R mainKernel : % -> Union(Kernel %,"failed")
--R minPoly : Kernel % -> SparseUnivariatePolynomial % if $ has RING
--R multiEuclidean : (List %,%) -> Union(List %,"failed") if R has INTDOM
--R numer : % -> SparseMultivariatePolynomial(R,Kernel %) if R has RING
--R numerator : % -> % if R has RING
--R odd? : % -> Boolean if $ has RETRACT INT
--R one? : % -> Boolean if R has SGROUP
--R operator : BasicOperator -> BasicOperator
--R operators : % -> List BasicOperator
--R patternMatch : (%,Pattern Float,PatternMatchResult(Float,%)) -> PatternMatchResult(Float,%) if R has PA
--R patternMatch : (%,Pattern Integer,PatternMatchResult(Integer,%)) -> PatternMatchResult(Integer,%) if R
--R prime? : % -> Boolean if R has INTDOM
--R principalIdeal : List % -> Record(coef: List %,generator: %) if R has INTDOM
--R ?quo? : (%,%) -> % if R has INTDOM
--R recip : % -> Union(%,"failed") if R has SGROUP
--R reducedSystem : Matrix % -> Matrix R if R has RING
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix R,vec: Vector R) if R has RING
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer) if and(has(R,Lin
--R reducedSystem : Matrix % -> Matrix Integer if and(has(R,LinearlyExplicitRingOver Integer),has(R,Ring))
--R ?rem? : (%,%) -> % if R has INTDOM
--R retract : % -> Fraction Integer if R has RETRACT INT and R has INTDOM or R has RETRACT FRAC INT
--R retract : % -> Polynomial R if R has RING
--R retract : % -> Fraction Polynomial R if R has INTDOM
--R retract : % -> Integer if R has RETRACT INT
--R retractIfCan : % -> Union(Fraction Integer,"failed") if R has RETRACT INT and R has INTDOM or R has RET
--R retractIfCan : % -> Union(Polynomial R,"failed") if R has RING
--R retractIfCan : % -> Union(Fraction Polynomial R,"failed") if R has INTDOM
--R retractIfCan : % -> Union(R,"failed")
--R retractIfCan : % -> Union(Integer,"failed") if R has RETRACT INT
--R retractIfCan : % -> Union(Symbol,"failed")
--R retractIfCan : % -> Union(Kernel %,"failed")
--R sample : () -> % if R has SGROUP or R has ABELSG
--R sizeLess? : (%,%) -> Boolean if R has INTDOM
--R squareFree : % -> Factored % if R has INTDOM
--R squareFreePart : % -> % if R has INTDOM
--R subst : (%,List Kernel %,List %) -> %
--R subst : (%,List Equation %) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed") if R has ABELGRP
--R unit? : % -> Boolean if R has INTDOM
--R unitCanonical : % -> % if R has INTDOM
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %) if R has INTDOM
```

```
--R univariate : (%,Kernel %) -> Fraction SparseUnivariatePolynomial % if R has INTDOM
--R zero? : % -> Boolean if R has ABELSG
--R
--E 1

)spool
)lisp (bye)
```

———-

— **FunctionSpace.help** —

```
=====================================================================
FunctionSpace examples
=====================================================================

See Also:
o )show FunctionSpace
```

———-

See:

⇐ “AbelianGroup” (ABELGRP) 7.0.97 on page 520
⇐ “AbelianMonoid” (ABELMON) 5.0.66 on page 267
⇐ “AbelianSemiGroup” (ABELSG) 4.0.36 on page 123
⇐ “Algebra” (ALGEBRA) 11.0.152 on page 947
⇐ “AlgebraicallyClosedFunctionSpace” (ACFS) 18.0.205 on page 1495
⇐ “CharacteristicNonZero” (CHARNZ) 10.0.135 on page 833
⇐ “CharacteristicZero” (CHARZ) 10.0.136 on page 838
⇐ “CommutativeRing” (COMRING) 10.0.137 on page 843
⇐ “ConvertibleTo” (KONVERT) 2.0.8 on page 23
⇐ “ExpressionSpace” (ES) 5.0.72 on page 300
⇐ “Field” (FIELD) 16.0.185 on page 1221
⇐ “FullyLinearlyExplicitRingOver” (FLINEXP) 11.0.154 on page 960
⇐ “FullyPatternMatchable” (FPATMAB) 3.0.28 on page 91
⇐ “FullyRetractableTo” (FRETRCT) 3.0.27 on page 87
⇐ “Group” (GROUP) 6.0.86 on page 398
⇐ “IntegralDomain” (INTDOM) 12.0.164 on page 1047
⇐ “Monoid” (MONOID) 5.0.76 on page 331
⇐ “PartialDifferentialRing” (PDRING) 10.0.145 on page 885
⇐ “Patternable” (PATAB) 2.0.15 on page 47
⇐ “RetractableTo” (RETRACT) 2.0.18 on page 55
⇐ “Ring” (RING) 9.0.129 on page 770
⇐ “SemiGroup” (SGROUP) 4.0.61 on page 239

Exports:

0	1	applyQuote	associates?
belong?	box	characteristic	charthRoot
coerce	commutator	conjugate	convert
D	definingPolynomial	denom	denominator
differentiate	distribute	divide	elt
eval	euclideanSize	even?	expressIdealMember
exquo	extendedEuclidean	factor	freeOf?
gcd	gcdPolynomial	ground	ground?
hash	height	inv	is?
isExpt	isMult	isPlus	isPower
isTimes	kernel	kernels	latex
lcm	mainKernel	map	max
min	minPoly	multiEuclidean	numer
numerator	odd?	one?	operator
operators	paren	patternMatch	prime?
principalIdeal	recip	reducedSystem	retract
retractIfCan	sample	sizeLess?	squareFree
squareFreePart	subst	subtractIfCan	tower
unit?	unitCanonical	unitNormal	univariate
variables	zero?	-?	?<?
?<=?	?=?	?>?	?>=?
?~=?	?*?	?**?	?+?
?-?	?/?	?^?	?quo?
?rem?			

Attributes Exported:

- if $ has IntegralDomain then noZeroDivisors where **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- if $ has IntegralDomain then canonicalUnitNormal where **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- if $ has IntegralDomain then canonicalsClosed where **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.
- if $ has IntegralDomain then commutative("*") where **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- if $ has Ring or Group then unitsKnown where **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- if $ has CommutativeRing then leftUnitary where **leftUnitary** is true if $1 * x = x$ for all x.

- if $ has CommutativeRing then rightUnitary where **rightUnitary** is true if $x * 1 = x$ for all x.
- **nil**

These are directly exported but not implemented:

```
coerce : SparseMultivariatePolynomial(R,Kernel %) -> %
    if R has RING
coerce : Fraction R -> % if R has INTDOM
coerce : Polynomial Fraction R -> % if R has INTDOM
denom : % -> SparseMultivariatePolynomial(R,Kernel %)
    if R has INTDOM
eval : (%,List BasicOperator,List %,Symbol) -> %
    if R has KONVERT INFORM
numer : % -> SparseMultivariatePolynomial(R,Kernel %)
    if R has RING
?/? :
  (SparseMultivariatePolynomial(R,Kernel %),
   SparseMultivariatePolynomial(R,Kernel %)) -> %
    if R has INTDOM
?**? : (%,NonNegativeInteger) -> % if R has SGROUP
```

These are implemented by this category:

```
applyQuote : (Symbol,%) -> %
applyQuote : (Symbol,%,%) -> %
applyQuote : (Symbol,%,%,%) -> %
applyQuote : (Symbol,%,%,%,%) -> %
applyQuote : (Symbol,List %) -> %
belong? : BasicOperator -> Boolean
characteristic : () -> NonNegativeInteger if R has RING
coerce : Symbol -> %
coerce : Kernel % -> %
coerce : % -> OutputForm
coerce : Fraction Integer -> %
    if R has INTDOM
    or R has RETRACT INT
    and R has INTDOM
    or R has RETRACT FRAC INT
coerce : Fraction Polynomial R -> % if R has INTDOM
coerce : Fraction Polynomial Fraction R -> %
    if R has INTDOM
coerce : Polynomial R -> % if R has RING
convert : % -> Pattern Float if R has KONVERT PATTERN FLOAT
convert : % -> Pattern Integer if R has KONVERT PATTERN INT
convert : Factored % -> % if R has INTDOM
denominator : % -> % if R has INTDOM
differentiate : (%,Symbol) -> % if R has RING
elt : (BasicOperator,List %) -> %
```

```
eval : (%,List Symbol) -> % if R has KONVERT INFORM
eval : % -> % if R has KONVERT INFORM
eval : (%,Symbol) -> % if R has KONVERT INFORM
eval : (%,BasicOperator,%,Symbol) -> %
    if R has KONVERT INFORM
eval : (%,Symbol,NonNegativeInteger,(% -> %)) -> %
    if R has RING
eval : (%,Symbol,NonNegativeInteger,(List % -> %)) -> %
    if R has RING
eval : (%,List Symbol,List NonNegativeInteger,List (List % -> %)) -> %
    if R has RING
eval : (%,List Symbol,List NonNegativeInteger,List (% -> %)) -> %
    if R has RING
eval : (%,List Kernel %,List %) -> %
ground : % -> R
ground? : % -> Boolean
isExpt : (%,BasicOperator) ->
   Union(Record(var: Kernel %,exponent: Integer),"failed")
    if R has RING
isExpt : % ->
  Union(Record(var: Kernel %,exponent: Integer),"failed")
    if R has SGROUP
isExpt : (%,Symbol) ->
  Union(Record(var: Kernel %,exponent: Integer),"failed")
    if R has RING
isMult : % ->
  Union(Record(coef: Integer,var: Kernel %),"failed")
    if R has ABELSG
isPlus : % -> Union(List %,"failed") if R has ABELSG
isPower : % ->
  Union(Record(val: %,exponent: Integer),"failed")
    if R has RING
isTimes : % -> Union(List %,"failed") if R has SGROUP
kernels : % -> List Kernel %
mainKernel : % -> Union(Kernel %,"failed")
numerator : % -> % if R has RING
operator : BasicOperator -> BasicOperator
retract : % -> Fraction Polynomial R if R has INTDOM
retract : % -> Polynomial R if R has RING
retract : % -> R
retract : % -> Symbol
retractIfCan : % -> Union(R,"failed")
retractIfCan : % -> Union(Fraction Polynomial R,"failed")
    if R has INTDOM
retractIfCan : % -> Union(Polynomial R,"failed")
    if R has RING
retractIfCan : % -> Union(Symbol,"failed")
subst : (%,List Kernel %,List %) -> %
univariate : (%,Kernel %) ->
  Fraction SparseUnivariatePolynomial %
```

```
    if R has INTDOM
variables : % -> List Symbol
?*? : (%,R) -> % if R has COMRING
?*? : (R,%) -> % if R has COMRING
```

These exports come from (p300) ExpressionSpace():

```
box : List % -> %
box : % -> %
definingPolynomial : % -> % if $ has RING
distribute : (%,%) -> %
distribute : % -> %
elt : (BasicOperator,%,%,%,%) -> %
elt : (BasicOperator,%,%,%) -> %
elt : (BasicOperator,%,%) -> %
elt : (BasicOperator,%) -> %
eval : (%,List BasicOperator,List (% -> %)) -> %
eval : (%,List Equation %) -> %
eval : (%,Symbol,(% -> %)) -> %
eval : (%,Symbol,(List % -> %)) -> %
eval : (%,BasicOperator,(% -> %)) -> %
eval : (%,BasicOperator,(List % -> %)) -> %
eval : (%,List Symbol,List (% -> %)) -> %
eval : (%,List BasicOperator,List (List % -> %)) -> %
eval : (%,List Symbol,List (List % -> %)) -> %
eval : (%,List %,List %) -> %
eval : (%,%,%) -> %
eval : (%,Equation %) -> %
eval : (%,Kernel %,%) -> %
even? : % -> Boolean if $ has RETRACT INT
freeOf? : (%,Symbol) -> Boolean
freeOf? : (%,%) -> Boolean
hash : % -> SingleInteger
height : % -> NonNegativeInteger
is? : (%,BasicOperator) -> Boolean
is? : (%,Symbol) -> Boolean
kernel : (BasicOperator,%) -> %
kernel : (BasicOperator,List %) -> %
latex : % -> String
map : ((% -> %),Kernel %) -> %
max : (%,%) -> %
min : (%,%) -> %
minPoly : Kernel % -> SparseUnivariatePolynomial %
    if $ has RING
odd? : % -> Boolean if $ has RETRACT INT
operators : % -> List BasicOperator
paren : List % -> %
paren : % -> %
retract : % -> Kernel %
retractIfCan : % -> Union(Kernel %,"failed")
```

```
subst : (%,List Equation %) -> %
subst : (%,Equation %) -> %
tower : % -> List Kernel %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?<? : (%,%) -> Boolean
?<=? : (%,%) -> Boolean
?>? : (%,%) -> Boolean
?>=? : (%,%) -> Boolean
```

These exports come from (p91) FullyPatternMatchable(OrderedSet):

```
patternMatch :
  (%,Pattern Integer,PatternMatchResult(Integer,%)) ->
    PatternMatchResult(Integer,%)
      if R has PATMAB INT
patternMatch :
  (%,Pattern Float,PatternMatchResult(Float,%)) ->
    PatternMatchResult(Float,%)
      if R has PATMAB FLOAT
```

These exports come from (p87) FullyRetractableTo(OrderedSet):

```
coerce : Integer -> % if R has RING or R has RETRACT INT
coerce : R -> %
```

These exports come from (p23) ConvertibleTo(InputForm):

```
convert : % -> InputForm if R has KONVERT INFORM
```

These exports come from (p331) Monoid():

```
1 : () -> % if R has SGROUP
one? : % -> Boolean if R has SGROUP
recip : % -> Union(%,"failed") if R has SGROUP
sample : () -> % if R has SGROUP or R has ABELSG
?^? : (%,NonNegativeInteger) -> % if R has SGROUP
?*? : (%,%) -> % if R has SGROUP
?**? : (%,PositiveInteger) -> % if R has SGROUP
?^? : (%,PositiveInteger) -> % if R has SGROUP
```

These exports come from (p398) Group():

```
commutator : (%,%) -> % if R has GROUP
conjugate : (%,%) -> % if R has GROUP
inv : % -> % if R has GROUP or R has INTDOM
?/? : (%,%) -> % if R has GROUP or R has INTDOM
?^? : (%,Integer) -> % if R has GROUP or R has INTDOM
?**? : (%,Integer) -> % if R has GROUP or R has INTDOM
```

These exports come from (p267) AbelianMonoid():

```
0 : () -> % if R has ABELSG
zero? : % -> Boolean if R has ABELSG
?*? : (PositiveInteger,%) -> % if R has ABELSG
?*? : (NonNegativeInteger,%) -> % if R has ABELSG
?+? : (%,%) -> % if R has ABELSG
```

These exports come from (p520) AbelianGroup():

```
subtractIfCan : (%,%) -> Union(%,"failed") if R has ABELGRP
?*? : (Integer,%) -> % if R has ABELGRP
?-? : (%,%) -> % if R has ABELGRP
-? : % -> % if R has ABELGRP
```

These exports come from (p885) PartialDifferentialRing(Symbol):

```
D : (%,Symbol) -> % if R has RING
D : (%,List Symbol) -> % if R has RING
D : (%,Symbol,NonNegativeInteger) -> % if R has RING
D : (%,List Symbol,List NonNegativeInteger) -> % if R has RING
differentiate : (%,List Symbol) -> % if R has RING
differentiate : (%,Symbol,NonNegativeInteger) -> % if R has RING
differentiate : (%,List Symbol,List NonNegativeInteger) -> %
    if R has RING
```

These exports come from (p960) FullyLinearlyExplicitRingOver(R) where R:OrderedSet:

```
reducedSystem : Matrix % -> Matrix R if R has RING
reducedSystem : (Matrix %,Vector %) ->
  Record(mat: Matrix R,vec: Vector R) if R has RING
reducedSystem : (Matrix %,Vector %) ->
  Record(mat: Matrix Integer,vec: Vector Integer)
    if and(has(R,LinearlyExplicitRingOver Integer),has(R,Ring))
reducedSystem : Matrix % -> Matrix Integer
    if and(has(R,LinearlyExplicitRingOver Integer),has(R,Ring))
```

These exports come from (p833) CharacteristicNonZero():

```
charthRoot : % -> Union(%,"failed") if R has CHARNZ
```

These exports come from (p1047) IntegralDomain():

```
associates? : (%,%) -> Boolean if R has INTDOM
exquo : (%,%) -> Union(%,"failed") if R has INTDOM
unit? : % -> Boolean if R has INTDOM
unitCanonical : % -> % if R has INTDOM
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
    if R has INTDOM
```

These exports come from (p1221) Field():

```
coerce : % -> % if R has INTDOM
divide : (%,%) -> Record(quotient: %,remainder: %)
    if R has INTDOM
euclideanSize : % -> NonNegativeInteger if R has INTDOM
expressIdealMember : (List %,%) -> Union(List %,"failed")
    if R has INTDOM
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
    if R has INTDOM
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
    if R has INTDOM
factor : % -> Factored % if R has INTDOM
gcd : (%,%) -> % if R has INTDOM
gcd : List % -> % if R has INTDOM
gcdPolynomial :
  (SparseUnivariatePolynomial %,
   SparseUnivariatePolynomial %) ->
     SparseUnivariatePolynomial %
       if R has INTDOM
lcm : (%,%) -> % if R has INTDOM
lcm : List % -> % if R has INTDOM
multiEuclidean : (List %,%) -> Union(List %,"failed")
    if R has INTDOM
prime? : % -> Boolean if R has INTDOM
principalIdeal : List % -> Record(coef: List %,generator: %)
    if R has INTDOM
sizeLess? : (%,%) -> Boolean if R has INTDOM
squareFree : % -> Factored % if R has INTDOM
squareFreePart : % -> % if R has INTDOM
?quo? : (%,%) -> % if R has INTDOM
?rem? : (%,%) -> % if R has INTDOM
```

These exports come from (p55) RetractableTo(Integer):

```
retract : % -> Integer if R has RETRACT INT
retractIfCan : % -> Union(Integer,"failed")
    if R has RETRACT INT
```

These exports come from (p55) RetractableTo(Fraction(Integer)):

```
retract : % -> Fraction Integer
    if R has RETRACT INT
    and R has INTDOM
    or R has RETRACT FRAC INT
retractIfCan : % -> Union(Fraction Integer,"failed")
    if R has RETRACT INT
    and R has INTDOM
    or R has RETRACT FRAC INT
```

```
 ?*? : (%,Fraction Integer) -> % if R has INTDOM
 ?*? : (Fraction Integer,%) -> % if R has INTDOM
```

— category FS FunctionSpace —

```
)abbrev category FS FunctionSpace
++ Category for formal functions
++ Author: Manuel Bronstein
++ Date Created: 22 March 1988
++ Date Last Updated: 14 February 1994
++ Keywords: operator, kernel, function.
++ Description:
++ A space of formal functions with arguments in an arbitrary ordered set.

FunctionSpace(R:OrderedSet): Category == Definition where
  OP ==> BasicOperator
  O  ==> OutputForm
  SY ==> Symbol
  N  ==> NonNegativeInteger
  Z  ==> Integer
  K  ==> Kernel %
  Q  ==> Fraction R
  PR ==> Polynomial R
  MP ==> SparseMultivariatePolynomial(R, K)
  QF==> PolynomialCategoryQuotientFunctions(IndexedExponents K,K,R,MP,%)

  ODD  ==> "odd"
  EVEN ==> "even"

  SPECIALDIFF  ==> "%specialDiff"
  SPECIALDISP  ==> "%specialDisp"
  SPECIALEQUAL ==> "%specialEqual"
  SPECIALINPUT ==> "%specialInput"

  Definition ==> Join(ExpressionSpace, RetractableTo SY, Patternable R,
                      FullyPatternMatchable R, FullyRetractableTo R) with
       ground?   : % -> Boolean
         ++ ground?(f) tests if f is an element of R.
       ground    : % -> R
         ++ ground(f) returns f as an element of R.
         ++ An error occurs if f is not an element of R.
       variables : %  -> List SY
         ++ variables(f) returns the list of all the variables of f.
       applyQuote: (SY, %) -> %
         ++ applyQuote(foo, x) returns \spad{'foo(x)}.
       applyQuote: (SY, %, %) -> %
         ++ applyQuote(foo, x, y) returns \spad{'foo(x,y)}.
       applyQuote: (SY, %, %, %) -> %
         ++ applyQuote(foo, x, y, z) returns \spad{'foo(x,y,z)}.
```

```
applyQuote: (SY, %, %, %, %) -> %
  ++ applyQuote(foo, x, y, z, t) returns \spad{'foo(x,y,z,t)}.
applyQuote: (SY, List %) -> %
  ++ applyQuote(foo, [x1,...,xn]) returns \spad{'foo(x1,...,xn)}.
if R has ConvertibleTo InputForm then
  ConvertibleTo InputForm
  eval     : (%, SY) -> %
    ++ eval(f, foo) unquotes all the foo's in f.
  eval     : (%, List SY) -> %
    ++ eval(f, [foo1,...,foon]) unquotes all the \spad{fooi}'s in f.
  eval     : % -> %
    ++ eval(f) unquotes all the quoted operators in f.
  eval     : (%, OP, %, SY) -> %
    ++ eval(x, s, f, y) replaces every \spad{s(a)} in x by \spad{f(y)}
    ++ with \spad{y} replaced by \spad{a} for any \spad{a}.
  eval     : (%, List OP, List %, SY) -> %
    ++ eval(x, [s1,...,sm], [f1,...,fm], y) replaces every
    ++ \spad{si(a)} in x by \spad{fi(y)}
    ++ with \spad{y} replaced by \spad{a} for any \spad{a}.
if R has SemiGroup then
  Monoid
  -- the following line is necessary because of a compiler bug
  "**"   : (%, N) -> %
    ++ x**n returns x * x * x * ... * x (n times).
  isTimes: % -> Union(List %, "failed")
    ++ isTimes(p) returns \spad{[a1,...,an]}
    ++ if \spad{p = a1*...*an} and \spad{n > 1}.
  isExpt : % -> Union(Record(var:K,exponent:Z),"failed")
    ++ isExpt(p) returns \spad{[x, n]} if \spad{p = x**n}
    ++ and \spad{n <> 0}.
if R has Group then Group
if R has AbelianSemiGroup then
  AbelianMonoid
  isPlus: % -> Union(List %, "failed")
    ++ isPlus(p) returns \spad{[m1,...,mn]}
    ++ if \spad{p = m1 +...+ mn} and \spad{n > 1}.
  isMult: % -> Union(Record(coef:Z, var:K),"failed")
    ++ isMult(p) returns \spad{[n, x]} if \spad{p = n * x}
    ++ and \spad{n <> 0}.
if R has AbelianGroup then AbelianGroup
if R has Ring then
  Ring
  RetractableTo PR
  PartialDifferentialRing SY
  FullyLinearlyExplicitRingOver R
  coerce    : MP -> %
    ++ coerce(p) returns p as an element of %.
  numer     : %  -> MP
    ++ numer(f) returns the
    ++ numerator of f viewed as a polynomial in the kernels over R
```

```
    ++ if R is an integral domain. If not, then numer(f) = f viewed
    ++ as a polynomial in the kernels over R.
    -- DO NOT change this meaning of numer!  MB 1/90
  numerator : % -> %
    ++ numerator(f) returns the numerator of \spad{f} converted to %.
  isExpt:(%,OP) -> Union(Record(var:K,exponent:Z),"failed")
    ++ isExpt(p,op) returns \spad{[x, n]} if \spad{p = x**n}
    ++ and \spad{n <> 0} and \spad{x = op(a)}.
  isExpt:(%,SY) -> Union(Record(var:K,exponent:Z),"failed")
    ++ isExpt(p,f) returns \spad{[x, n]} if \spad{p = x**n}
    ++ and \spad{n <> 0} and \spad{x = f(a)}.
  isPower   : % -> Union(Record(val:%,exponent:Z),"failed")
    ++ isPower(p) returns \spad{[x, n]} if \spad{p = x**n}
    ++ and \spad{n <> 0}.
  eval: (%, List SY, List N, List(% -> %)) -> %
    ++ eval(x, [s1,...,sm], [n1,...,nm], [f1,...,fm]) replaces
    ++ every \spad{si(a)**ni} in x by \spad{fi(a)} for any \spad{a}.
  eval: (%, List SY, List N, List(List % -> %)) -> %
    ++ eval(x, [s1,...,sm], [n1,...,nm], [f1,...,fm]) replaces
    ++ every \spad{si(a1,...,an)**ni} in x by \spad{fi(a1,...,an)}
    ++ for any a1,...,am.
  eval: (%, SY, N, List % -> %) -> %
    ++ eval(x, s, n, f) replaces every \spad{s(a1,...,am)**n} in x
    ++ by \spad{f(a1,...,am)} for any a1,...,am.
  eval: (%, SY, N, % -> %) -> %
    ++ eval(x, s, n, f) replaces every \spad{s(a)**n} in x
    ++ by \spad{f(a)} for any \spad{a}.
if R has CharacteristicZero then CharacteristicZero
if R has CharacteristicNonZero then CharacteristicNonZero
if R has CommutativeRing then
  Algebra R
if R has IntegralDomain then
  Field
  RetractableTo Fraction PR
  convert   : Factored % -> %
    ++ convert(f1\^e1 ... fm\^em) returns \spad{(f1)\^e1 ... (fm)\^em}
    ++ as an element of %, using formal kernels
    ++ created using a \spadfunFrom{paren}{ExpressionSpace}.
  denom     : %  -> MP
    ++ denom(f) returns the denominator of f viewed as a
    ++ polynomial in the kernels over R.
  denominator : % -> %
    ++ denominator(f) returns the denominator of \spad{f}
    ++ converted to %.
  "/"       : (MP, MP) -> %
    ++ p1/p2 returns the quotient of p1 and p2 as an element of %.
  coerce    : Q  -> %
    ++ coerce(q) returns q as an element of %.
  coerce    : Polynomial Q -> %
    ++ coerce(p) returns p as an element of %.
```

```
      coerce    : Fraction Polynomial Q -> %
        ++ coerce(f) returns f as an element of %.
      univariate: (%, K) -> Fraction SparseUnivariatePolynomial %
        ++ univariate(f, k) returns f viewed as a univariate fraction in k.
      if R has RetractableTo Z then RetractableTo Fraction Z
add
 import BasicOperatorFunctions1(%)

 -- these are needed in Ring only, but need to be declared here
 -- because of compiler bug: if they are declared inside the Ring
 -- case, then they are not visible inside the IntegralDomain case.
 smpIsMult : MP -> Union(Record(coef:Z, var:K),"failed")
 smpret    : MP -> Union(PR, "failed")
 smpeval   : (MP, List K, List %) -> %
 smpsubst  : (MP, List K, List %) -> %
 smpderiv  : (MP, SY) -> %
 smpunq    : (MP, List SY, Boolean) -> %
 kerderiv  : (K, SY)  -> %
 kderiv    : K -> List %
 opderiv   : (OP, N) -> List(List % -> %)
 smp2O     : MP -> O
 bestKernel: List K -> K
 worse?    : (K, K) -> Boolean
 diffArg   : (List %, OP, N) -> List %
 substArg  : (OP, List %, Z, %) -> %
 dispdiff  : List % -> Record(name:O, sub:O, arg:List O, level:N)
 ddiff     : List % -> O
 diffEval  : List % -> %
 dfeval    : (List %, K) -> %
 smprep    : (List SY, List N, List(List % -> %), MP) -> %
 diffdiff  : (List %, SY) -> %
 diffdiff0 : (List %, SY, %, K, List %) -> %
 subs      : (% -> %, K) -> %
 symsub    : (SY, Z) -> SY
 kunq      : (K, List SY, Boolean) -> %
 pushunq   : (List SY, List %) -> List %
 notfound  : (K -> %, List K, K) -> %

 equaldiff : (K,K)->Boolean
 debugA: (List % ,List %,Boolean) -> Boolean
 opdiff := operator("%diff"::SY)$CommonOperators()
 opquote := operator("applyQuote"::SY)$CommonOperators

 ground? x               == retractIfCan(x)@Union(R,"failed") case R

 ground  x               == retract x

 coerce(x:SY):%            == kernel(x)@K :: %

 retract(x:%):SY           == symbolIfCan(retract(x)@K)::SY
```

```
applyQuote(s:SY, x:%)      == applyQuote(s, [x])

applyQuote(s, x, y)        == applyQuote(s, [x, y])

applyQuote(s, x, y, z)     == applyQuote(s, [x, y, z])

applyQuote(s, x, y, z, t)  == applyQuote(s, [x, y, z, t])

applyQuote(s:SY, l:List %) == opquote concat(s::%, l)

belong? op                 == op = opdiff or op = opquote

subs(fn, k) == kernel(operator k,[fn x for x in argument k]$List(%))

operator op ==
  is?(op, "%diff"::SY) => opdiff
  is?(op, "%quote"::SY) => opquote
  error "Unknown operator"

if R has ConvertibleTo InputForm then
  INP==>InputForm
  import MakeUnaryCompiledFunction(%, %, %)
  indiff: List % -> INP
  pint  : List INP-> INP
  differentiand: List % -> %

  differentiand l    == eval(first l, retract(second l)@K, third l)

  pint l  == convert concat(convert("D"::SY)@INP, l)

  indiff l ==
     r2:= convert([convert("::"::SY)@INP,_
                   convert(third l)@INP,_
                   convert("Symbol"::SY)@INP]@List INP)@INP
     pint [convert(differentiand l)@INP, r2]

  eval(f:%, s:SY)            == eval(f, [s])

  eval(f:%, s:OP, g:%, x:SY) == eval(f, [s], [g], x)

  eval(f:%, ls:List OP, lg:List %, x:SY) ==
    eval(f, ls, [compiledFunction(g, x) for g in lg])

  setProperty(opdiff,SPECIALINPUT,_
              indiff@(List % -> InputForm) pretend None)

variables x ==
  l := empty()$List(SY)
  for k in tower x repeat
```

```
        if ((s := symbolIfCan k) case SY) then l := concat(s::SY, l)
      reverse_! l

    retractIfCan(x:%):Union(SY, "failed") ==
      (k := retractIfCan(x)@Union(K,"failed")) case "failed" => "failed"
      symbolIfCan(k::K)

    if R has Ring then
      import UserDefinedPartialOrdering(SY)

-- cannot use new()$Symbol because of possible re-instantiation
      gendiff := "%%0"::SY

      characteristic()    == characteristic()$R

      coerce(k:K):%       == k::MP::%

      symsub(sy, i)       == concat(string sy, convert(i)@String)::SY

      numerator x         == numer(x)::%

      eval(x:%, s:SY, n:N, f:% -> %) ==
        eval(x,[s],[n],[(y:List %):% +-> f(first(y))])

      eval(x:%, s:SY, n:N, f:List % -> %) == eval(x, [s], [n], [f])

      eval(x:%, l:List SY, f:List(List % -> %)) == eval(x, l, new(#l, 1), f)

      elt(op:OP, args:List %) ==
        unary? op and ((od? := has?(op, ODD)) or has?(op, EVEN)) and
          leadingCoefficient(numer first args) < 0 =>
            x := op(- first args)
            od? => -x
            x
        elt(op, args)$ExpressionSpace_&(%)

      eval(x:%, s:List SY, n:List N, l:List(% -> %)) ==
        eval(x, s, n, [y+-> f(first(y)) for f in l]$List(List % -> %))

      -- op(arg)**m ==> func(arg)**(m quo n) * op(arg)**(m rem n)
      smprep(lop, lexp, lfunc, p) ==
        (v := mainVariable p) case "failed" => p::%
        symbolIfCan(k := v::K) case SY => p::%
        g := (op := operator k)
           (arg := [eval(a,lop,lexp,lfunc) for a in argument k]$List(%))
        q := map(y+->eval(y::%, lop, lexp, lfunc),
                 univariate(p, k))$SparseUnivariatePolynomialFunctions2(MP, %)
        (n := position(name op, lop)) < minIndex lop => q g
        a:%  := 0
        f    := eval((lfunc.n) arg, lop, lexp, lfunc)
```

```
    e    := lexp.n
    while q ^= 0 repeat
      m  := degree q
      qr := divide(m, e)
      t1 := f ** (qr.quotient)::N
      t2 := g ** (qr.remainder)::N
      a  := a + leadingCoefficient(q) * t1 * t2
      q  := reductum q
    a

  dispdiff l ==
    s := second(l)::O
    t := third(l)::O
    a := argument(k := retract(first l)@K)
    is?(k, opdiff) =>
      rec := dispdiff a
      i   := position(s, rec.arg)
      rec.arg.i := t
      [rec.name,
         hconcat(rec.sub, hconcat(",":SY::O, (i+1-minIndex a)::O)),
                   rec.arg, (zero?(rec.level) => 0; rec.level + 1)]
    i   := position(second l, a)
    m   := [x::O for x in a]$List(O)
    m.i := t
    [name(operator k)::O, hconcat(",":SY::O, (i+1-minIndex a)::O),
                                          m, (empty? rest a => 1; 0)]

  ddiff l ==
    rec := dispdiff l
    opname :=
      zero?(rec.level) => sub(rec.name, rec.sub)
      differentiate(rec.name, rec.level)
    prefix(opname, rec.arg)

  substArg(op, l, i, g) ==
    z := copy l
    z.i := g
    kernel(op, z)


  diffdiff(l, x) ==
    f := kernel(opdiff, l)
    diffdiff0(l, x, f, retract(f)@K, empty())

  diffdiff0(l, x, expr, kd, done) ==
    op  := operator(k := retract(first l)@K)
    gg  := second l
    u   := third l
    arg := argument k
    ans:% := 0
```

```
        if (not member?(u,done)) and (ans := differentiate(u,x))^=0 then
          ans := ans * kernel(opdiff,
               [subst(expr, [kd], [kernel(opdiff, [first l, gg, gg])]),
                           gg, u])
        done := concat(gg, done)
        is?(k, opdiff) => ans + diffdiff0(arg, x, expr, k, done)
        for i in minIndex arg .. maxIndex arg for b in arg repeat
          if (not member?(b,done)) and (bp:=differentiate(b,x))^=0 then
            g   := symsub(gendiff, i)::%
            ans := ans + bp * kernel(opdiff, [subst(expr, [kd],
             [kernel(opdiff, [substArg(op, arg, i, g), gg, u])]), g, b])
        ans

      dfeval(l, g) ==
        eval(differentiate(first l, symbolIfCan(g)::SY), g, third l)

      diffEval l ==
        k:K
        g := retract(second l)@K
        ((u := retractIfCan(first l)@Union(K, "failed")) case "failed")
          or (u case K and symbolIfCan(k := u::K) case SY) => dfeval(l, g)
        op := operator k
        (ud := derivative op) case "failed" =>
             -- possible trouble
             -- make sure it is a dummy var
             dumm:%:=symsub(gendiff,1)::%
             ss:=subst(l.1,l.2=dumm)
             -- output(nl::OutputForm)$OutputPackage
             -- output("fixed"::OutputForm)$OutputPackage
             nl:=[ss,dumm,l.3]
             kernel(opdiff, nl)
        (n := position(second l,argument k)) < minIndex l =>
              dfeval(l,g)
        d := ud::List(List % -> %)
        eval((d.n)(argument k), g, third l)

      diffArg(l, op, i) ==
        n := i - 1 + minIndex l
        z := copy l
        z.n := g := symsub(gendiff, n)::%
        [kernel(op, z), g, l.n]

      opderiv(op, n) ==
--      one? n =>
        (n = 1) =>
          g := symsub(gendiff, n)::%
          [x +-> kernel(opdiff,[kernel(op, g), g, first x])]
        [y +-> kernel(opdiff, diffArg(y, op, i)) for i in 1..n]

      kderiv k ==
```

```
      zero?(n := #(args := argument k)) => empty()
      op := operator k
      grad :=
        (u := derivative op) case "failed" => opderiv(op, n)
        u::List(List % -> %)
      if #grad ^= n then grad := opderiv(op, n)
      [g args for g in grad]

  -- SPECIALDIFF contains a map (List %, Symbol) -> %
  -- it is used when the usual chain rule does not apply,
  -- for instance with implicit algebraics.
    kerderiv(k, x) ==
      (v := symbolIfCan(k)) case SY =>
        v::SY = x => 1
        0
      (fn := property(operator k, SPECIALDIFF)) case None =>
         ((fn::None) pretend ((List %, SY) -> %)) (argument k, x)
      +/[g * differentiate(y,x) for g in kderiv k for y in argument k]

    smpderiv(p, x) ==
      map((s:R):R +-> retract differentiate(s::PR, x), p)::% +
       +/[differentiate(p,k)::% * kerderiv(k, x) for k in variables p]

    coerce(p:PR):% ==
      map(s +-> s::%, r +-> r::%, p)$PolynomialCategoryLifting(
                                      IndexedExponents SY, SY, R, PR, %)

    worse?(k1, k2) ==
      (u := less?(name operator k1,name operator k2)) case "failed" =>
        k1 < k2
      u::Boolean

    bestKernel l ==
      empty? rest l => first l
      a := bestKernel rest l
      worse?(first l, a) => a
      first l

    smp20 p ==
      (r:=retractIfCan(p)@Union(R,"failed")) case R =>r::R::OutputForm
      a :=
        userOrdered?() => bestKernel variables p
        mainVariable(p)::K
      outputForm(map((x:MP):% +-> x::%, univariate(p, a))_
          $SparseUnivariatePolynomialFunctions2(MP, %), a::OutputForm)

    smpsubst(p, lk, lv) ==
      map(x +-> match(lk, lv, x,
          notfound((z:K):%+->subs(s+->subst(s, lk, lv), z), lk, x))_
           $ListToMap(K,%),y+->y::%,p)_
```

```
            $PolynomialCategoryLifting(IndexedExponents K,K,R,MP,%)

      smpeval(p, lk, lv) ==
        map(x +-> match(lk, lv, x,
            notfound((z:K):%+->map(s+->eval(s,lk,lv),z),lk,x))_
             $ListToMap(K,%),y+->y::%,p)_
              $PolynomialCategoryLifting(IndexedExponents K,K,R,MP,%)

-- this is called on k when k is not a member of lk
      notfound(fn, lk, k) ==
        empty? setIntersection(tower(f := k::%), lk) => f
        fn k

      if R has ConvertibleTo InputForm then
        pushunq(l, arg) ==
           empty? l => [eval a for a in arg]
           [eval(a, l) for a in arg]

        kunq(k, l, givenlist?) ==
          givenlist? and empty? l => k::%
          is?(k, opquote) and
            (member?(s:=retract(first argument k)@SY, l) or empty? l) =>
              interpret(convert(concat(convert(s)@InputForm,
                [convert a for a in pushunq(l, rest argument k)
                   ]@List(InputForm)))@InputForm)$InputFormFunctions1(%)
          (operator k) pushunq(l, argument k)

        smpunq(p, l, givenlist?) ==
          givenlist? and empty? l => p::%
          map(x +-> kunq(x, l, givenlist?), y+->y::%, p)_
            $PolynomialCategoryLifting(IndexedExponents K,K,R,MP,%)

      smpret p ==
        "or"/[symbolIfCan(k) case "failed" for k in variables p] =>
          "failed"
        map(x+->symbolIfCan(x)::SY::PR, y+->y::PR,p)_
          $PolynomialCategoryLifting(IndexedExponents K, K, R, MP, PR)

      isExpt(x:%, op:OP) ==
        (u := isExpt x) case "failed" => "failed"
        is?((u::Record(var:K, exponent:Z)).var, op) => u
        "failed"

      isExpt(x:%, sy:SY) ==
        (u := isExpt x) case "failed" => "failed"
        is?((u::Record(var:K, exponent:Z)).var, sy) => u
        "failed"

      if R has RetractableTo Z then
          smpIsMult p ==
```

```
--          (u := mainVariable p) case K and one? degree(q:=univariate(p,u::K))
            (u := mainVariable p) case K and (degree(q:=univariate(p,u::K))=1)
              and zero?(leadingCoefficient reductum q)
                and ((r:=retractIfCan(leadingCoefficient q)@Union(R,"failed"))
                   case R)
                     and (n := retractIfCan(r::R)@Union(Z, "failed")) case Z =>
                       [n::Z, u::K]
            "failed"

      evaluate(opdiff, diffEval)

      debugA(a1,a2,t) ==
         -- uncomment for debugging
         -- output(hconcat [a1::OutputForm,_
         --                 a2::OutputForm,t::OutputForm])$OutputPackage
         t

      equaldiff(k1,k2) ==
        a1:=argument k1
        a2:=argument k2
        -- check the operator
        res:=operator k1 = operator k2
        not res => debugA(a1,a2,res)
        -- check the evaluation point
        res:= (a1.3 = a2.3)
        not res => debugA(a1,a2,res)
        -- check all the arguments
        res:= (a1.1 = a2.1) and (a1.2 = a2.2)
        res => debugA(a1,a2,res)
        -- check the substituted arguments
        (subst(a1.1,[retract(a1.2)@K],[a2.2]) = a2.1) => debugA(a1,a2,true)
        debugA(a1,a2,false)

      setProperty(opdiff,SPECIALEQUAL,
                          equaldiff@((K,K) -> Boolean) pretend None)

      setProperty(opdiff, SPECIALDIFF,
                          diffdiff@((List %, SY) -> %) pretend None)

      setProperty(opdiff, SPECIALDISP,
                              ddiff@(List % -> OutputForm) pretend None)

      if not(R has IntegralDomain) then
        mainKernel x         == mainVariable numer x

        kernels x            == variables numer x

        retract(x:%):R       == retract numer x

        retract(x:%):PR      == smpret(numer x)::PR
```

```
    retractIfCan(x:%):Union(R,  "failed") == retract numer x

    retractIfCan(x:%):Union(PR, "failed") == smpret numer x

    eval(x:%, lk:List K, lv:List %)  == smpeval(numer x, lk, lv)

    subst(x:%, lk:List K, lv:List %) == smpsubst(numer x, lk, lv)

    differentiate(x:%, s:SY)         == smpderiv(numer x, s)

    coerce(x:%):OutputForm           == smp2O numer x

    if R has ConvertibleTo InputForm then
      eval(f:%, l:List SY) == smpunq(numer f, l, true)

      eval f               == smpunq(numer f, empty(), false)

    eval(x:%, s:List SY, n:List N, f:List(List % -> %)) ==
      smprep(s, n, f, numer x)

    isPlus x ==
      (u := isPlus numer x) case "failed" => "failed"
      [p::% for p in u::List(MP)]

    isTimes x ==
      (u := isTimes numer x) case "failed" => "failed"
      [p::% for p in u::List(MP)]

    isExpt x ==
      (u := isExpt numer x) case "failed" => "failed"
      r := u::Record(var:K, exponent:NonNegativeInteger)
      [r.var, r.exponent::Z]

    isPower x ==
      (u := isExpt numer x) case "failed" => "failed"
      r := u::Record(var:K, exponent:NonNegativeInteger)
      [r.var::%, r.exponent::Z]

    if R has ConvertibleTo Pattern Z then
      convert(x:%):Pattern(Z) == convert numer x

    if R has ConvertibleTo Pattern Float then
      convert(x:%):Pattern(Float) == convert numer x

    if R has RetractableTo Z then
      isMult x == smpIsMult numer x

if R has CommutativeRing then
  r:R * x:% == r::MP::% * x
```

```
    if R has IntegralDomain then
      par   : % -> %

      mainKernel x                    == mainVariable(x)$QF

      kernels x                       == variables(x)$QF

      univariate(x:%, k:K)            == univariate(x, k)$QF

      isPlus x                        == isPlus(x)$QF

      isTimes x                       == isTimes(x)$QF

      isExpt x                        == isExpt(x)$QF

      isPower x                       == isPower(x)$QF

      denominator x                   == denom(x)::%

      coerce(q:Q):%                   == (numer q)::MP / (denom q)::MP

      coerce(q:Fraction PR):%         == (numer q)::% / (denom q)::%

      coerce(q:Fraction Polynomial Q) == (numer q)::% / (denom q)::%

      retract(x:%):PR                == retract(retract(x)@Fraction(PR))

      retract(x:%):Fraction(PR) == smpret(numer x)::PR / smpret(denom x)::PR

      retract(x:%):R == (retract(numer x)@R exquo retract(denom x)@R)::R

      coerce(x:%):OutputForm ==
--        one?(denom x) => smp2O numer x
        ((denom x) = 1) => smp2O numer x
        smp2O(numer x) / smp2O(denom x)

      retractIfCan(x:%):Union(R, "failed") ==
        (n := retractIfCan(numer x)@Union(R, "failed")) case "failed" or
          (d := retractIfCan(denom x)@Union(R, "failed")) case "failed"
            or (r := n::R exquo d::R) case "failed" => "failed"
        r::R

      eval(f:%, l:List SY) ==
        smpunq(numer f, l, true) / smpunq(denom f, l, true)

      if R has ConvertibleTo InputForm then
        eval f ==
          smpunq(numer f, empty(), false) / smpunq(denom f, empty(), false)
```

```
  eval(x:%, s:List SY, n:List N, f:List(List % -> %)) ==
    smprep(s, n, f, numer x) / smprep(s, n, f, denom x)

differentiate(f:%, x:SY) ==
  (smpderiv(numer f, x) * denom(f)::% -
    numer(f)::% * smpderiv(denom f, x))
      / (denom(f)::% ** 2)

eval(x:%, lk:List K, lv:List %) ==
  smpeval(numer x, lk, lv) / smpeval(denom x, lk, lv)

subst(x:%, lk:List K, lv:List %) ==
  smpsubst(numer x, lk, lv) / smpsubst(denom x, lk, lv)

par x ==
  (r := retractIfCan(x)@Union(R, "failed")) case R => x
  paren x

convert(x:Factored %):% ==
  par(unit x) * */[par(f.factor) ** f.exponent for f in factors x]

retractIfCan(x:%):Union(PR, "failed") ==
  (u := retractIfCan(x)@Union(Fraction PR,"failed")) case "failed"
    => "failed"
  retractIfCan(u::Fraction(PR))

retractIfCan(x:%):Union(Fraction PR, "failed") ==
  (n := smpret numer x) case "failed" => "failed"
  (d := smpret denom x) case "failed" => "failed"
  n::PR / d::PR

coerce(p:Polynomial Q):% ==
  map(x+->x::%, y+->y::%,p)_
   $PolynomialCategoryLifting(IndexedExponents SY, SY,
                                          Q, Polynomial Q, %)

if R has RetractableTo Z then
  coerce(x:Fraction Z):% == numer(x)::MP / denom(x)::MP

  isMult x ==
     (u := smpIsMult numer x) case "failed"
        or (v := retractIfCan(denom x)@Union(R, "failed")) case "failed"
           or (w := retractIfCan(v::R)@Union(Z, "failed")) case "failed"
               => "failed"
     r := u::Record(coef:Z, var:K)
     (q := r.coef exquo w::Z) case "failed" => "failed"
     [q::Z, r.var]

if R has ConvertibleTo Pattern Z then
  convert(x:%):Pattern(Z) == convert(numer x) / convert(denom x)
```

```
      if R has ConvertibleTo Pattern Float then
        convert(x:%):Pattern(Float) ==
          convert(numer x) / convert(denom x)
```

———-

— **FS.dotabb** —

```
"FS"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FS"];
"FS" -> "ES"
"FS" -> "FPATMAB"
"FS" -> "FRETRCT"
"FS" -> "PATAB"
"FS" -> "RETRACT"
"FS" -> "KONVERT"
"FS" -> "MONOID"
"FS" -> "GROUP"
"FS" -> "ABELMON"
"FS" -> "ABELGRP"
"FS" -> "PDRING"
"FS" -> "FLINEXP"
"FS" -> "CHARNZ"
"FS" -> "INTDOM"
"FS" -> "FIELD"
```

———-

— **FS.dotfull** —

```
"FunctionSpace(a:OrderedSet)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FS"];
"FunctionSpace(a:OrderedSet)" -> "ExpressionSpace()"
"FunctionSpace(a:OrderedSet)" -> "RetractableTo(Symbol)"
"FunctionSpace(a:OrderedSet)" -> "Patternable(OrderedSet)"
"FunctionSpace(a:OrderedSet)" -> "FullyPatternMatchable(OrderedSet)"
"FunctionSpace(a:OrderedSet)" -> "FullyRetractableTo(OrderedSet)"
"FunctionSpace(a:OrderedSet)" -> "ConvertibleTo(InputForm)"
"FunctionSpace(a:OrderedSet)" -> "Monoid()"
"FunctionSpace(a:OrderedSet)" -> "Group()"
"FunctionSpace(a:OrderedSet)" -> "AbelianMonoid()"
"FunctionSpace(a:OrderedSet)" -> "AbelianGroup()"
"FunctionSpace(a:OrderedSet)" -> "PartialDifferentialRing(Symbol)"
"FunctionSpace(a:OrderedSet)" -> "FullyLinearlyExplicitRingOver(OrderedSet)"
"FunctionSpace(a:OrderedSet)" -> "CharacteristicNonZero()"
```

```
"FunctionSpace(a:OrderedSet)" -> "IntegralDomain()"
"FunctionSpace(a:OrderedSet)" -> "Field()"
"FunctionSpace(a:OrderedSet)" -> "RetractableTo(Integer)"
"FunctionSpace(a:OrderedSet)" -> "RetractableTo(Fraction(Integer))"
```

———-

— **FS.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

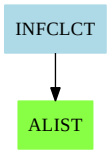
"FunctionSpace(a:OrderedSet)" [color=lightblue];
"FunctionSpace(a:OrderedSet)" -> "ES..."
"FunctionSpace(a:OrderedSet)" -> "RETRACT..."
"FunctionSpace(a:OrderedSet)" -> "PATAB..."
"FunctionSpace(a:OrderedSet)" -> "FPATMAB..."
"FunctionSpace(a:OrderedSet)" -> "FRETRCT..."
"FunctionSpace(a:OrderedSet)" -> "KONVERT..."
"FunctionSpace(a:OrderedSet)" -> "MONOID..."
"FunctionSpace(a:OrderedSet)" -> "GROUP..."
"FunctionSpace(a:OrderedSet)" -> "ABELMON..."
"FunctionSpace(a:OrderedSet)" -> "ABELGRP..."
"FunctionSpace(a:OrderedSet)" -> "PDRING..."
"FunctionSpace(a:OrderedSet)" -> "FLINEXP..."
"FunctionSpace(a:OrderedSet)" -> "CHARNZ..."
"FunctionSpace(a:OrderedSet)" -> "INTDOM..."
"FunctionSpace(a:OrderedSet)" -> "FIELD..."
"FunctionSpace(a:OrderedSet)" -> "RETRACT..."

"ES..." [color=lightblue];
"EVALABLE..." [color=lightblue];
"FRETRCT..." [color=lightblue];
"FPATMAB..." [color=lightblue];
"IEVALAB..." [color=lightblue];
"ORDSET..." [color=lightblue];
"PATAB..." [color=lightblue];
"RETRACT..." [color=lightblue];
"KONVERT..." [color=lightblue];
"MONOID..." [color=lightblue];
"GROUP..." [color=lightblue];
"ABELMON..." [color=lightblue];
"ABELGRP..." [color=lightblue];
"PDRING..." [color=lightblue];
"FLINEXP..." [color=lightblue];
"CHARNZ..." [color=lightblue];
```

```
"INTDOM..." [color=lightblue];
"FIELD..." [color=lightblue];
"RETRACT..." [color=lightblue];
}
```

———-

17.0.196 InfinitlyClosePointCategory (INFCLCT)



— InfinitlyClosePointCategory.input —

```
)set break resume
)sys rm -f InfinitlyClosePointCategory.output
)spool InfinitlyClosePointCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show InfinitlyClosePointCategory
--I InfinitlyClosePointCategory(K: Field,
--I symb: List Symbol,
--I PolyRing: PolynomialCategory(t#1,t#4,OrderedVariableList t#2),
--I E: DirectProductCategory(# t#2,NonNegativeInteger),
--I ProjPt: ProjectiveSpaceCategory t#1,
--I PCS: LocalPowerSeriesCategory t#1,
--I Plc: PlacesCategory(t#1,t#6),
--I DIVISOR: DivisorCategory t#7,
--I BLMET: BlowUpMethodCategory) is a category constructor
--I Abbreviation for InfinitlyClosePointCategory is INFCLCT
--I This constructor is exposed in this frame.
--I Issue )edit bookvol10.2.pamphlet to see algebra source code for INFCLCT
--I
--I------------------------------ Operations --------------------------------
--I ?=? : (%,%) -> Boolean                actualExtensionV : % -> K
--I chartV : % -> BLMET                   coerce : % -> OutputForm
--I create : (ProjPt,PolyRing) -> %       degree : % -> PositiveInteger
--I excpDivV : % -> DIVISOR               hash : % -> SingleInteger
```

```
--I latex : % -> String                    localParamV : % -> List PCS
--I localPointV : % -> AffinePlane K       multV : % -> NonNegativeInteger
--I pointV : % -> ProjPt                   setchart! : (%,BLMET) -> BLMET
--I setpoint! : (%,ProjPt) -> ProjPt       symbNameV : % -> Symbol
--I ?~=? : (%,%) -> Boolean
--I create : (ProjPt,
--I  DistributedMultivariatePolynomial([construct,QUOTEX,QUOTEY],K),
--I  AffinePlane K,
--I  NonNegativeInteger,
--I  BLMET,
--I  NonNegativeInteger,
--I  DIVISOR,
--I  K,
--I  Symbol) -> %
--I curveV : % ->
--I  DistributedMultivariatePolynomial([construct,QUOTEX,QUOTEY],K)
--I setcurve! :
--I  (%,DistributedMultivariatePolynomial([construct,QUOTEX,QUOTEY],K)) ->
--I  DistributedMultivariatePolynomial([construct,QUOTEX,QUOTEY],K)
--I setexcpDiv! : (%,DIVISOR) -> DIVISOR
--I setlocalParam! : (%,List PCS) -> List PCS
--I setlocalPoint! : (%,AffinePlane K) -> AffinePlane K
--I setmult! : (%,NonNegativeInteger) -> NonNegativeInteger
--I setsubmult! : (%,NonNegativeInteger) -> NonNegativeInteger
--I setsymbName! : (%,Symbol) -> Symbol
--I subMultV : % -> NonNegativeInteger
--I
--E 1

)spool
)lisp (bye)
```

———-

— InfinitlyClosePointCategory.help —

```
=====================================================================
InfinitlyClosePointCategory examples
=====================================================================

See Also:
o )show InfinitlyClosePointCategory
```

———-

⇐ “SetCategoryWithDegree” (SETCATD) 4.0.62 on page 243

Exports:

?=?	?~=?	actualExtensionV	chartV
coerce	create	curveV	degree
excpDivV	hash	latex	localParamV
localPointV	multV	pointV	setchart!
setcurve!	setexcpDiv!	setlocalParam!	setlocalPoint!
setmult!	setpoint!	setsubmult!	setsymbName!
subMultV	symbNameV		

These are directly exported but not implemented:

```
actualExtensionV : % -> K
chartV : % -> BLMET
create :
 (ProjPt,DistributedMultivariatePolynomial([construct,QUOTEX,QUOTEY],K),
  AffinePlane K,NonNegativeInteger,BLMET,NonNegativeInteger,DIVISOR,K,Symbol)
    -> %
create : (ProjPt,PolyRing) -> %
curveV : % -> DistributedMultivariatePolynomial([construct,QUOTEX,QUOTEY],K)
excpDivV : % -> DIVISOR
localParamV : % -> List PCS
localPointV : % -> AffinePlane K
multV : % -> NonNegativeInteger
pointV : % -> ProjPt
setchart! : (%,BLMET) -> BLMET
setcurve! :
  (%,DistributedMultivariatePolynomial([construct,QUOTEX,QUOTEY],K)) ->
    DistributedMultivariatePolynomial([construct,QUOTEX,QUOTEY],K)
setexcpDiv! : (%,DIVISOR) -> DIVISOR
setlocalParam! : (%,List PCS) -> List PCS
setlocalPoint! : (%,AffinePlane K) -> AffinePlane K
setmult! : (%,NonNegativeInteger) -> NonNegativeInteger
setpoint! : (%,ProjPt) -> ProjPt
setsubmult! : (%,NonNegativeInteger) -> NonNegativeInteger
setsymbName! : (%,Symbol) -> Symbol
subMultV : % -> NonNegativeInteger
symbNameV : % -> Symbol
```

These exports come from (p243) SetCategoryWithDegree:

```
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
coerce : % -> OutputForm
degree : % -> PositiveInteger
hash : % -> SingleInteger
latex : % -> String
```

— category INFCLCT InfinitlyClosePointCategory —

```
)abbrev category INFCLCT InfinitlyClosePointCategory
```

```
++ Authors: Gaetan Hache
++ Date Created: may 1997
++ Date Last Updated: April 2010, by Tim Daly
++ Description:
++ This category is part of the PAFF package
InfinitlyClosePointCategory(_
     K        :Field,_
     symb     :List(Symbol),_
     PolyRing :PolynomialCategory(K,E,OrderedVariableList(symb)),_
     E        :DirectProductCategory(#symb,NonNegativeInteger),_
     ProjPt   :ProjectiveSpaceCategory(K),_
     PCS      :LocalPowerSeriesCategory(K),_
     Plc      :PlacesCategory(K,PCS),_
     DIVISOR  :DivisorCategory(Plc),_
     BLMET    :BlowUpMethodCategory_
       ):Category == Exports where

 bls      ==> ['X,'Y]
 BlUpRing ==> DistributedMultivariatePolynomial(bls , K)
 AFP      ==> AffinePlane(K)

 Exports ==> SetCategoryWithDegree with

    create:  (ProjPt ,  BlUpRing, AFP , NonNegativeInteger,BLMET, _
              NonNegativeInteger,  DIVISOR,K,Symbol) -> %
      ++ create an infinitly close point

    create:  (ProjPt,PolyRing) -> %

    setpoint_!:  (%,ProjPt) -> ProjPt

    setcurve_!:  (%,BlUpRing) -> BlUpRing

    setlocalPoint_!:   (%,AFP) -> AFP

    setsubmult_! : (%, NonNegativeInteger) -> NonNegativeInteger

    setmult_!:    (%,NonNegativeInteger) -> NonNegativeInteger

    setchart_!:   (%,BLMET) -> BLMET -- CHH

    setexcpDiv_!: (%,DIVISOR) -> DIVISOR

    setlocalParam_!: (%,List PCS) -> List(PCS)

    setsymbName_!: (%,Symbol) -> Symbol

    subMultV: % -> NonNegativeInteger

    localParamV: % -> List PCS
```

```
    symbNameV: % -> Symbol

    pointV:  % -> ProjPt
      ++ pointV returns the infinitly close point.

    curveV:  % -> BlUpRing
      ++ curveV(p) returns the defining polynomial of the strict transform
      ++ on which lies the corresponding infinitly close point.

    localPointV: % -> AFP
      ++ localPointV returns the coordinates of the local infinitly
      ++ close point

    multV:       % -> NonNegativeInteger
      ++ multV returns the multiplicity of the infinitly close point.

    chartV:      % -> BLMET -- CHH
      ++ chartV is the chart of the infinitly close point. The first integer
      ++ correspond to variable defining the exceptional line, the last one
      ++ the affine  neighboorhood and the second one is the
      ++ remaining integer. For example [1,2,3] means that
      ++ Z=1, X=X and Y=XY. [2,3,1] means that X=1, Y=Y and Z=YZ.

    excpDivV:    % -> DIVISOR
      ++ excpDivV returns the exceptional divisor of the infinitly close point.

    actualExtensionV: % -> K
```

———-

— INFCLCT.dotabb —

```
"INFCLCT" [color=lightblue,href="bookvol10.2.pdf#nameddest=INFCLCT"];
"ALIST" [color="#88FF44",href="bookvol10.3.pdf#nameddest=ALIST"]
"INFCLCT" -> "ALIST"
```

———-

— INFCLCT.dotfull —

```
"InfinitlyClosePointCategory"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=INFCLCT"];
"InfinitlyClosePointCategory" -> "AssocationList(SetCategory,SetCategory)"
```

———-

— **INFCLCT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"InfinitlyClosePointCategory" [color=lightblue];
"InfinitlyClosePointCategory" -> "AssocationList(SetCategory,SetCategory)"

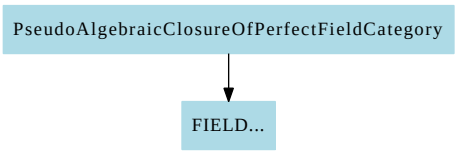
"AssocationList(SetCategory,SetCategory)" -> "AssocationList()"

"AssocationList()" [color=lightblue];

}
```

———-

17.0.197 PseudoAlgebraicClosureOfPerfectFieldCategory (PACPERC)



— **PseudoAlgebraicClosureOfPerfectFieldCategory.input** —

```
)set break resume
)sys rm -f PseudoAlgebraicClosureOfPerfectFieldCategory.output
)spool PseudoAlgebraicClosureOfPerfectFieldCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show PseudoAlgebraicClosureOfPerfectFieldCategory
--R
--R PseudoAlgebraicClosureOfPerfectFieldCategory  is a category constructor
--R Abbreviation for PseudoAlgebraicClosureOfPerfectFieldCategory is PACPERC
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PACPERC
--R
```

```
--R------------------------------ Operations ------------------------------
--R ?*? : (Fraction Integer,%) -> %       ?*? : (%,Fraction Integer) -> %
--R ?*? : (%,%) -> %                     ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %       ?**? : (%,Integer) -> %
--R ?**? : (%,PositiveInteger) -> %      ?+? : (%,%) -> %
--R ?-? : (%,%) -> %                     -? : % -> %
--R ?/? : (%,%) -> %                     ?=? : (%,%) -> Boolean
--R 1 : () -> %                          0 : () -> %
--R ?^? : (%,Integer) -> %               ?^? : (%,PositiveInteger) -> %
--R associates? : (%,%) -> Boolean       coerce : Fraction Integer -> %
--R coerce : % -> %                      coerce : Integer -> %
--R coerce : % -> OutputForm             conjugate : % -> %
--R extDegree : % -> PositiveInteger     factor : % -> Factored %
--R fullOutput : % -> OutputForm         gcd : List % -> %
--R gcd : (%,%) -> %                     ground? : % -> Boolean
--R hash : % -> SingleInteger            inv : % -> %
--R latex : % -> String                  lcm : List % -> %
--R lcm : (%,%) -> %                     maxTower : List % -> %
--R one? : % -> Boolean                  previousTower : % -> %
--R prime? : % -> Boolean                ?quo? : (%,%) -> %
--R recip : % -> Union(%,"failed")       ?rem? : (%,%) -> %
--R sample : () -> %                     setTower! : % -> Void
--R sizeLess? : (%,%) -> Boolean         squareFree : % -> Factored %
--R squareFreePart : % -> %              unit? : % -> Boolean
--R unitCanonical : % -> %               vectorise : (%,%) -> Vector %
--R zero? : % -> Boolean                 ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R definingPolynomial : % -> SparseUnivariatePolynomial %
--R definingPolynomial : () -> SparseUnivariatePolynomial %
--R distinguishedRootsOf : (SparseUnivariatePolynomial %,%) -> List %
--R divide : (%,%) -> Record(quotient: %,remainder: %)
--R euclideanSize : % -> NonNegativeInteger
--R expressIdealMember : (List %,%) -> Union(List %,"failed")
--R exquo : (%,%) -> Union(%,"failed")
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivari
--R lift : (%,%) -> SparseUnivariatePolynomial %
--R lift : % -> SparseUnivariatePolynomial %
--R multiEuclidean : (List %,%) -> Union(List %,"failed")
--R newElement : (SparseUnivariatePolynomial %,Symbol) -> %
--R newElement : (SparseUnivariatePolynomial %,%,Symbol) -> %
--R principalIdeal : List % -> Record(coef: List %,generator: %)
--R reduce : SparseUnivariatePolynomial % -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R
```

```
--E 1

)spool
)lisp (bye)
```

———-

— **PseudoAlgebraicClosureOfPerfectFieldCategory.help** —

```
====================================================================
PseudoAlgebraicClosureOfPerfectFieldCategory examples
====================================================================

See Also:
o )show PseudoAlgebraicClosureOfPerfectFieldCategory
```

———-

⇐ "Field" (FIELD) 16.0.185 on page 1221

Exports:

0	1	associates?	characteristic
coerce	conjugate	definingPolynomial	distinguishedRootsOf
divide	euclideanSize	expressIdealMember	exquo
extDegree	extendedEuclidean	factor	fullOutput
gcd	gcdPolynomial	ground?	hash
inv	latex	lcm	lift
maxTower	multiEuclidean	newElement	one?
previousTower	prime?	principalIdeal	?quo?
recip	reduce	?rem?	sample
setTower!	sizeLess?	squareFree	squareFreePart
subtractIfCan	unit?	unitCanonical	unitNormal
vectorise	zero?	?*?	?**?
?+?	?-?	-?	?/?
?=?	?^?	?~=?	

Attributes Exported:

- **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.
- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.

- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
conjugate: % -> %
definingPolynomial: () -> SUP(%)
definingPolynomial: % -> SUP %
distinguishedRootsOf: (SparseUnivariatePolynomial %,%) -> List %
extDegree: % -> PI
fullOutput: % -> OutputForm
ground_? : % -> Boolean
lift: % -> SUP(%)
lift: (%,%) -> SUP(%)
maxTower: List % -> %
newElement: (SUP(%), %, Symbol) -> %
newElement: (SUP(%), Symbol) -> %
previousTower: % -> %
reduce: SUP(%) -> %
setTower_!: % -> Void
vectorise: (%,%) -> Vector(%)
```

These exports come from (p1221) Field():

```
associates? : (%,%) -> Boolean
divide : (%,%) -> Record(quotient: %,remainder: %)
euclideanSize : % -> NonNegativeInteger
exquo : (%,%) -> Union(%,"failed")
factor : % -> Factored %
gcd : (%,%) -> %
inv : % -> %
prime? : % -> Boolean
squareFree : % -> Factored %
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
?/? : (%,%) -> %
```

These exports come from (p1185) EuclideanDomain():

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
coerce : % -> %
```

```
coerce : Integer -> %
coerce : % -> OutputForm
expressIdealMember : (List %,%) -> Union(List %,"failed")
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
gcd : List % -> %
gcdPolynomial : (SparseUnivariatePolynomial %,
                 SparseUnivariatePolynomial %) ->
                   SparseUnivariatePolynomial %
hash : % -> SingleInteger
latex : % -> String
lcm : List % -> %
lcm : (%,%) -> %
multiEuclidean : (List %,%) -> Union(List %,"failed")
one? : % -> Boolean
principalIdeal : List % -> Record(coef: List %,generator: %)
recip : % -> Union(%,"failed")
sample : () -> %
sizeLess? : (%,%) -> Boolean
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?quo? : (%,%) -> %
?rem? : (%,%) -> %
```

These exports come from (p1172) UniqueFactorizationDomain():

```
squareFreePart : % -> %
```

These exports come from (p1013) DivisionRing():

```
coerce : Fraction Integer -> %
?*? : (Fraction Integer,%) -> %
?*? : (%,Fraction Integer) -> %
?**? : (%,Integer) -> %
?^? : (%,Integer) -> %
```

— category PACPERC PseudoAlgebraicClosureOfPerfectFieldCategory —

```
)abbrev category PACPERC PseudoAlgebraicClosureOfPerfectFieldCategory
++ Authors: Gaetan Hache
++ Date Created: may 1997
++ Date Last Updated: April 2010, by Tim Daly
++ Description: This category exports the function for domains
++ which implement dynamic extension using the simple notion of tower
++ extensions. ++ A tower extension T  of the ground
++ field K is any sequence of field extension
++ (T : K_0, K_1, ..., K_i...,K_n) where K_0 = K
++ and for i =1,2,...,n, K_i is an extension of K_{i-1} of degree > 1
++ and defined by an irreducible polynomial p(Z) in K_{i-1}.
++ Two towers (T_1: K_01, K_11,...,K_i1,...,K_n1)
++ and (T_2: K_02, K_12,...,K_i2,...,K_n2)
++ are said to be related if T_1 <= T_2 (or T_1 >= T_2),
++ that is if K_i1 = K_i2 for i=1,2,...,n1 (or i=1,2,...,n2).
++ Any algebraic operations defined for several elements
++ are only defined if all of the concerned elements are comming from
++ a set of related tour extensions.
PseudoAlgebraicClosureOfPerfectFieldCategory() : Category == PUB where

 INT      ==> Integer
 K        ==> Fraction Integer
 NNI      ==> NonNegativeInteger
 SUP      ==> SparseUnivariatePolynomial
 BOOLEAN  ==> Boolean
 PI       ==> PositiveInteger
 FFFACTSE ==> FiniteFieldFactorizationWithSizeParseBySideEffect

 PUB ==> Field with

  definingPolynomial: () -> SUP(%)
  definingPolynomial: % -> SUP %

  lift: % -> SUP(%)
  lift: (%,%) -> SUP(%)
  reduce: SUP(%) -> %

  distinguishedRootsOf: (SparseUnivariatePolynomial %,%) -> List %
    ++ distinguishedRootsOf(p,a) returns a (distinguised) root for each
    ++ irreducible factor of the polynomial p (factored over the field defined
    ++ by the element a).

  ground_? : % -> Boolean
  maxTower: List % -> %
    ++ maxTower(l) returns the tower in the list having the maximal extension
    ++ degree over the ground field. It has no meaning if the towers are
```

```
    ++ not related.
  extDegree: % -> PI
    ++ extDegree(a) returns the extension degree of the extension tower
    ++ over which the element is defined.
  previousTower: % -> %
    ++ previousTower(a) returns the previous tower extension over which
    ++ the element a is defined.

  vectorise: (%,%) -> Vector(%)

  conjugate: % -> %
  newElement: (SUP(%), %, Symbol) -> %
  newElement: (SUP(%), Symbol) -> %
  setTower_!: % -> Void
  fullOutput: % -> OutputForm
```

———-

— PACPERC.dotabb —

```
"PACPERC" [color=lightblue,href="bookvol10.2.pdf#nameddest=PACPERC"];
"PACPERC" -> "FIELD"
```

———-

— PACPERC.dotfull —

```
"PseudoAlgebraicClosureOfPerfectFieldCategory"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PACPERC"];
"PseudoAlgebraicClosureOfPerfectFieldCategory" -> "Field()"
```

———-

— PACPERC.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

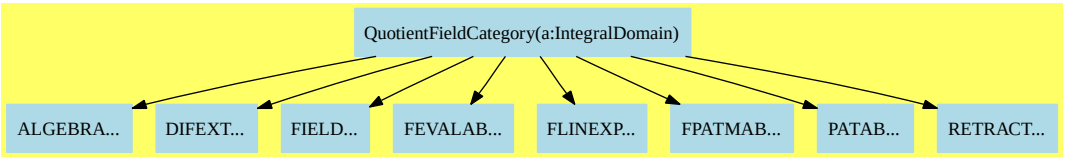
"PseudoAlgebraicClosureOfPerfectFieldCategory" [color=lightblue];
"PseudoAlgebraicClosureOfPerfectFieldCategory" -> "FIELD..."

"FIELD..." [color=lightblue];
```

```
}
```

———-

17.0.198 QuotientFieldCategory (QFCAT)



— **QuotientFieldCategory.input** —

```
)set break resume
)sys rm -f QuotientFieldCategory.output
)spool QuotientFieldCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show QuotientFieldCategory
--R QuotientFieldCategory S: IntegralDomain  is a category constructor
--R Abbreviation for QuotientFieldCategory is QFCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for QFCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,S) -> %                      ?*? : (S,%) -> %
--R ?*? : (Fraction Integer,%) -> %       ?*? : (%,Fraction Integer) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,Integer) -> %
--R ?**? : (%,PositiveInteger) -> %       ?+? : (%,%) -> %
--R ?-? : (%,%) -> %                      -? : % -> %
--R ?/? : (S,S) -> %                      ?/? : (%,%) -> %
--R ?=? : (%,%) -> Boolean                D : (%,(S -> S)) -> %
--R D : % -> % if S has DIFRING           1 : () -> %
--R 0 : () -> %                           ?^? : (%,Integer) -> %
--R ?^? : (%,PositiveInteger) -> %        associates? : (%,%) -> Boolean
--R ceiling : % -> S if S has INS         coerce : S -> %
--R coerce : Fraction Integer -> %        coerce : % -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R denom : % -> S                        denominator : % -> %
```

```
--R factor : % -> Factored %              floor : % -> S if S has INS
--R gcd : List % -> %                     gcd : (%,%) -> %
--R hash : % -> SingleInteger             init : () -> % if S has STEP
--R inv : % -> %                          latex : % -> String
--R lcm : List % -> %                     lcm : (%,%) -> %
--R map : ((S -> S),%) -> %               numer : % -> S
--R numerator : % -> %                    one? : % -> Boolean
--R prime? : % -> Boolean                 ?quo? : (%,%) -> %
--R random : () -> % if S has INS         recip : % -> Union(%,"failed")
--R ?rem? : (%,%) -> %                    retract : % -> S
--R sample : () -> %                      sizeLess? : (%,%) -> Boolean
--R squareFree : % -> Factored %          squareFreePart : % -> %
--R unit? : % -> Boolean                  unitCanonical : % -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?<? : (%,%) -> Boolean if S has ORDSET
--R ?<=? : (%,%) -> Boolean if S has ORDSET
--R ?>? : (%,%) -> Boolean if S has ORDSET
--R ?>=? : (%,%) -> Boolean if S has ORDSET
--R D : (%,(S -> S),NonNegativeInteger) -> %
--R D : (%,List Symbol,List NonNegativeInteger) -> % if S has PDRING SYMBOL
--R D : (%,Symbol,NonNegativeInteger) -> % if S has PDRING SYMBOL
--R D : (%,List Symbol) -> % if S has PDRING SYMBOL
--R D : (%,Symbol) -> % if S has PDRING SYMBOL
--R D : (%,NonNegativeInteger) -> % if S has DIFRING
--R ?^? : (%,NonNegativeInteger) -> %
--R abs : % -> % if S has OINTDOM
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if S has CHARNZ or and(has($,CharacteristicNonZero),has(S,Polynomial
--R coerce : Symbol -> % if S has RETRACT SYMBOL
--R conditionP : Matrix % -> Union(Vector %,"failed") if and(has($,CharacteristicNonZero),has(S,PolynomialFa
--R convert : % -> DoubleFloat if S has REAL
--R convert : % -> Float if S has REAL
--R convert : % -> InputForm if S has KONVERT INFORM
--R convert : % -> Pattern Float if S has KONVERT PATTERN FLOAT
--R convert : % -> Pattern Integer if S has KONVERT PATTERN INT
--R differentiate : (%,(S -> S)) -> %
--R differentiate : (%,(S -> S),NonNegativeInteger) -> %
--R differentiate : (%,List Symbol,List NonNegativeInteger) -> % if S has PDRING SYMBOL
--R differentiate : (%,Symbol,NonNegativeInteger) -> % if S has PDRING SYMBOL
--R differentiate : (%,List Symbol) -> % if S has PDRING SYMBOL
--R differentiate : (%,Symbol) -> % if S has PDRING SYMBOL
--R differentiate : (%,NonNegativeInteger) -> % if S has DIFRING
--R differentiate : % -> % if S has DIFRING
--R divide : (%,%) -> Record(quotient: %,remainder: %)
--R ?.? : (%,S) -> % if S has ELTAB(S,S)
--R euclideanSize : % -> NonNegativeInteger
--R eval : (%,Symbol,S) -> % if S has IEVALAB(SYMBOL,S)
--R eval : (%,List Symbol,List S) -> % if S has IEVALAB(SYMBOL,S)
```

```
--R eval : (%,List Equation S) -> % if S has EVALAB S
--R eval : (%,Equation S) -> % if S has EVALAB S
--R eval : (%,S,S) -> % if S has EVALAB S
--R eval : (%,List S,List S) -> % if S has EVALAB S
--R expressIdealMember : (List %,%) -> Union(List %,"failed")
--R exquo : (%,%) -> Union(%,"failed")
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
--R factorPolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial % if
--R factorSquareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolyno
--R fractionPart : % -> % if S has EUCDOM
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivari
--R max : (%,%) -> % if S has ORDSET
--R min : (%,%) -> % if S has ORDSET
--R multiEuclidean : (List %,%) -> Union(List %,"failed")
--R negative? : % -> Boolean if S has OINTDOM
--R nextItem : % -> Union(%,"failed") if S has STEP
--R patternMatch : (%,Pattern Float,PatternMatchResult(Float,%)) -> PatternMatchResult(Float,%)
--R patternMatch : (%,Pattern Integer,PatternMatchResult(Integer,%)) -> PatternMatchResult(Integ
--R positive? : % -> Boolean if S has OINTDOM
--R principalIdeal : List % -> Record(coef: List %,generator: %)
--R reducedSystem : Matrix % -> Matrix S
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix S,vec: Vector S)
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer) if S
--R reducedSystem : Matrix % -> Matrix Integer if S has LINEXP INT
--R retract : % -> Integer if S has RETRACT INT
--R retract : % -> Fraction Integer if S has RETRACT INT
--R retract : % -> Symbol if S has RETRACT SYMBOL
--R retractIfCan : % -> Union(Integer,"failed") if S has RETRACT INT
--R retractIfCan : % -> Union(Fraction Integer,"failed") if S has RETRACT INT
--R retractIfCan : % -> Union(Symbol,"failed") if S has RETRACT SYMBOL
--R retractIfCan : % -> Union(S,"failed")
--R sign : % -> Integer if S has OINTDOM
--R solveLinearPolynomialEquation : (List SparseUnivariatePolynomial %,SparseUnivariatePolynomia
--R squareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R wholePart : % -> S if S has EUCDOM
--R
--E 1

)spool
)lisp (bye)
```

———-

— QuotientFieldCategory.help —

```
=====================================================================
```

```
QuotientFieldCategory examples
==============================================================================

See Also:
o )show QuotientFieldCategory
```

———-

See:

⇐ “Algebra” (ALGEBRA) 11.0.152 on page 947
⇐ “CharacteristicNonZero” (CHARNZ) 10.0.135 on page 833
⇐ “CharacteristicZero” (CHARZ) 10.0.136 on page 838
⇐ “ConvertibleTo” (KONVERT) 2.0.8 on page 23
⇐ “DifferentialExtension” (DIFEXT) 11.0.153 on page 953
⇐ “EuclideanDomain” (EUCDOM) 15.0.181 on page 1185
⇐ “Field” (FIELD) 16.0.185 on page 1221
⇐ “FullyEvalableOver” (FEVALAB) 4.0.44 on page 159
⇐ “FullyLinearlyExplicitRingOver” (FLINEXP) 11.0.154 on page 960
⇐ “FullyPatternMatchable” (FPATMAB) 3.0.28 on page 91
⇐ “OrderedIntegralDomain” (OINTDOM) 13.0.174 on page 1135
⇐ “OrderedSet” (ORDSET) 4.0.56 on page 216
⇐ “Patternable” (PATAB) 2.0.15 on page 47
⇐ “PolynomialFactorizationExplicit” (PFECAT) 15.0.183 on page 1202
⇐ “RealConstant” (REAL) 3.0.32 on page 106
⇐ “RetractableTo” (RETRACT) 2.0.18 on page 55
⇐ “StepThrough” (STEP) 4.0.64 on page 252

Exports:

0	1	abs
associates?	ceiling	characteristic
charthRoot	coerce	conditionP
convert	D	denom
denominator	differentiate	divide
euclideanSize	eval	expressIdealMember
exquo	extendedEuclidean	factor
factorPolynomial	factorSquareFreePolynomial	floor
fractionPart	gcd	gcdPolynomial
hash	init	inv
latex	lcm	map
max	min	multiEuclidean
negative?	nextItem	numer
numerator	one?	patternMatch
positive?	prime?	principalIdeal
random	recip	reducedSystem
retract	retractIfCan	sample
sign	sizeLess?	solveLinearPolynomialEquation
squareFree	squareFreePart	squareFreePolynomial
subtractIfCan	unit?	unitNormal
unitCanonical	wholePart	zero?
?.?	?*?	?**?
?+?	?-?	-?
?/?	?=?	?^?
?quo?	?rem?	?~=?
?<?	?<=?	?>?
?>=?		

Attributes Exported:

- **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.
- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

- **nil**

These are directly exported but not implemented:

```
ceiling : % -> S if S has INS
denom : % -> S
floor : % -> S if S has INS
numer : % -> S
wholePart : % -> S if S has EUCDOM
?/? : (S,S) -> %
```

These are implemented by this category:

```
characteristic : () -> NonNegativeInteger
coerce : Symbol -> % if S has RETRACT SYMBOL
coerce : Fraction Integer -> %
convert : % -> InputForm if S has KONVERT INFORM
convert : % -> DoubleFloat if S has REAL
convert : % -> Float if S has REAL
convert : % -> Pattern Integer if S has KONVERT PATTERN INT
convert : % -> Pattern Float if S has KONVERT PATTERN FLOAT
denominator : % -> %
differentiate : (%,(S -> S)) -> %
fractionPart : % -> % if S has EUCDOM
init : () -> % if S has STEP
map : ((S -> S),%) -> %
nextItem : % -> Union(%,"failed") if S has STEP
numerator : % -> %
patternMatch :
  (%,Pattern Float,PatternMatchResult(Float,%)) ->
    PatternMatchResult(Float,%)
     if S has PATMAB FLOAT
patternMatch :
  (%,Pattern Integer,PatternMatchResult(Integer,%)) ->
    PatternMatchResult(Integer,%)
      if S has PATMAB INT
random : () -> % if S has INS
reducedSystem : Matrix % -> Matrix S
reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix S,vec: Vector S)
retract : % -> Symbol if S has RETRACT SYMBOL
retract : % -> Integer if S has RETRACT INT
retractIfCan : % -> Union(Integer,"failed") if S has RETRACT INT
retractIfCan : % -> Union(Symbol,"failed") if S has RETRACT SYMBOL
?<? : (%,%) -> Boolean if S has ORDSET
```

These exports come from (p1221) Field():

```
0 : () -> %
1 : () -> %
```

```
associates? : (%,%) -> Boolean
coerce : % -> %
coerce : Integer -> %
coerce : % -> OutputForm
divide : (%,%) -> Record(quotient: %,remainder: %)
euclideanSize : % -> NonNegativeInteger
expressIdealMember : (List %,%) -> Union(List %,"failed")
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
exquo : (%,%) -> Union(%,"failed")
factor : % -> Factored %
gcd : (%,%) -> %
gcd : List % -> %
gcdPolynomial :
  (SparseUnivariatePolynomial %,
   SparseUnivariatePolynomial %) ->
     SparseUnivariatePolynomial %
hash : % -> SingleInteger
inv : % -> %
latex : % -> String
lcm : List % -> %
lcm : (%,%) -> %
multiEuclidean : (List %,%) -> Union(List %,"failed")
one? : % -> Boolean
prime? : % -> Boolean
principalIdeal : List % -> Record(coef: List %,generator: %)
recip : % -> Union(%,"failed")
sample : () -> %
sizeLess? : (%,%) -> Boolean
squareFree : % -> Factored %
squareFreePart : % -> %
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
zero? : % -> Boolean
?*? : (Fraction Integer,%) -> %
?*? : (%,Fraction Integer) -> %
?**? : (%,Integer) -> %
?^? : (%,Integer) -> %
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
```

```
?**? : (%,NonNegativeInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?/? : (%,%) -> %
?quo? : (%,%) -> %
?rem? : (%,%) -> %
```

These exports come from (p947) Algebra(S:IntegralDomain):

```
coerce : S -> %
?*? : (%,S) -> %
?*? : (S,%) -> %
```

These exports come from (p55) RetractableTo(S:IntegralDomain):

```
retract : % -> S
retractIfCan : % -> Union(S,"failed")
```

These exports come from (p159) FullyEvalableOver(S:IntegralDomain):

```
?.? : (%,S) -> % if S has ELTAB(S,S)
eval : (%,Equation S) -> % if S has EVALAB S
eval : (%,List Symbol,List S) -> % if S has IEVALAB(SYMBOL,S)
eval : (%,List Equation S) -> % if S has EVALAB S
eval : (%,S,S) -> % if S has EVALAB S
eval : (%,List S,List S) -> % if S has EVALAB S
eval : (%,Symbol,S) -> % if S has IEVALAB(SYMBOL,S)
```

These exports come from (p953) DifferentialExtension(S:IntegralDomain):

```
D : (%,(S -> S)) -> %
D : (%,(S -> S),NonNegativeInteger) -> %
D : % -> % if S has DIFRING
D : (%,NonNegativeInteger) -> % if S has DIFRING
D : (%,List Symbol,List NonNegativeInteger) -> %
    if S has PDRING SYMBOL
D : (%,Symbol,NonNegativeInteger) -> %
    if S has PDRING SYMBOL
D : (%,List Symbol) -> % if S has PDRING SYMBOL
D : (%,Symbol) -> % if S has PDRING SYMBOL
differentiate : (%,List Symbol) -> %
    if S has PDRING SYMBOL
differentiate : (%,Symbol,NonNegativeInteger) -> %
    if S has PDRING SYMBOL
differentiate : (%,List Symbol,List NonNegativeInteger) -> %
    if S has PDRING SYMBOL
differentiate : (%,NonNegativeInteger) -> % if S has DIFRING
differentiate : % -> % if S has DIFRING
differentiate : (%,Symbol) -> % if S has PDRING SYMBOL
differentiate : (%,(S -> S),NonNegativeInteger) -> %
```

These exports come from (p960) FullyLinearlyExplicitRingOver(S:IntegralDomain):

```
reducedSystem : (Matrix %,Vector %) ->
  Record(mat: Matrix Integer,vec: Vector Integer)
    if S has LINEXP INT
reducedSystem : Matrix % -> Matrix Integer if S has LINEXP INT
```

These exports come from (p55) RetractableTo(Fraction(Integer)):

```
retract : % -> Fraction Integer if S has RETRACT INT
retractIfCan : % -> Union(Fraction Integer,"failed")
  if S has RETRACT INT
```

These exports come from (p216) OrderedSet():

```
max : (%,%) -> % if S has ORDSET
min : (%,%) -> % if S has ORDSET
?<=? : (%,%) -> Boolean if S has ORDSET
?>? : (%,%) -> Boolean if S has ORDSET
?>=? : (%,%) -> Boolean if S has ORDSET
```

These exports come from (p1135) OrderedIntegralDomain():

```
abs : % -> % if S has OINTDOM
negative? : % -> Boolean if S has OINTDOM
positive? : % -> Boolean if S has OINTDOM
sign : % -> Integer if S has OINTDOM
```

These exports come from (p833) CharacteristicNonZero():

```
charthRoot : % -> Union(%,"failed")
  if S has CHARNZ
  or and(has($,CharacteristicNonZero),
         has(S,PolynomialFactorizationExplicit))
```

These exports come from (p1202) PolynomialFactorizationExplicit():

```
conditionP : Matrix % -> Union(Vector %,"failed")
  if and(has($,CharacteristicNonZero),
         has(S,PolynomialFactorizationExplicit))
factorPolynomial :
  SparseUnivariatePolynomial % ->
    Factored SparseUnivariatePolynomial %
      if S has PFECAT
factorSquareFreePolynomial :
  SparseUnivariatePolynomial % ->
    Factored SparseUnivariatePolynomial %
      if S has PFECAT
solveLinearPolynomialEquation :
```

```
   (List SparseUnivariatePolynomial %,
    SparseUnivariatePolynomial %) ->
      Union(List SparseUnivariatePolynomial %,"failed")
       if S has PFECAT
 squareFreePolynomial :
   SparseUnivariatePolynomial % ->
     Factored SparseUnivariatePolynomial %
       if S has PFECAT
```

— category QFCAT QuotientFieldCategory —

```
)abbrev category QFCAT QuotientFieldCategory
++ Author:
++ Date Created:
++ Date Last Updated: 5th March 1996
++ Basic Functions: + - * / numer denom
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ QuotientField(S) is the category of fractions of an Integral Domain S.

QuotientFieldCategory(S: IntegralDomain): Category ==
  Join(Field, Algebra S, RetractableTo S, FullyEvalableOver S,
         DifferentialExtension S, FullyLinearlyExplicitRingOver S,
           Patternable S, FullyPatternMatchable S) with
    _/     : (S, S) -> %
       ++ d1 / d2 returns the fraction d1 divided by d2.
    numer  : % -> S
       ++ numer(x) returns the numerator of the fraction x.
    denom  : % -> S
       ++ denom(x) returns the denominator of the fraction x.
    numerator : % -> %
       ++ numerator(x) is the numerator of the fraction x converted to %.
    denominator : % -> %
       ++ denominator(x) is the denominator of the fraction x converted to %.
    if S has StepThrough then StepThrough
    if S has RetractableTo Integer then
             RetractableTo Integer
             RetractableTo Fraction Integer
    if S has OrderedSet then OrderedSet
    if S has OrderedIntegralDomain then OrderedIntegralDomain
    if S has RealConstant then RealConstant
    if S has ConvertibleTo InputForm then ConvertibleTo InputForm
    if S has CharacteristicZero then CharacteristicZero
    if S has CharacteristicNonZero then CharacteristicNonZero
    if S has RetractableTo Symbol then RetractableTo Symbol
```

```
    if S has EuclideanDomain then
      wholePart: % -> S
        ++ wholePart(x) returns the whole part of the fraction x
        ++ i.e. the truncated quotient of the numerator by the denominator.
      fractionPart: % -> %
        ++ fractionPart(x) returns the fractional part of x.
        ++ x = wholePart(x) + fractionPart(x)
    if S has IntegerNumberSystem then
      random: () -> %
        ++ random() returns a random fraction.
      ceiling : % -> S
        ++ ceiling(x) returns the smallest integral element above x.
      floor: % -> S
        ++ floor(x) returns the largest integral element below x.
    if S has PolynomialFactorizationExplicit then
      PolynomialFactorizationExplicit

  add
    import MatrixCommonDenominator(S, %)

    numerator(x) == numer(x)::%

    denominator(x) == denom(x) ::%

    if S has StepThrough then
       init() == init()$S / 1$S

       nextItem(n) ==
         m:= nextItem(numer(n))
         m case "failed" =>
           error "We seem to have a Fraction of a finite object"
         m / 1

    map(fn, x)                          == (fn numer x) / (fn denom x)

    reducedSystem(m:Matrix %):Matrix S == clearDenominator m

    characteristic()                    == characteristic()$S

    differentiate(x:%, deriv:S -> S) ==
        n := numer x
        d := denom x
        (deriv n * d - n * deriv d) / (d**2)

    if S has ConvertibleTo InputForm then
      convert(x:%):InputForm == (convert numer x) / (convert denom x)

    if S has RealConstant then
      convert(x:%):Float == (convert numer x) / (convert denom x)
```

```
  convert(x:%):DoubleFloat == (convert numer x) / (convert denom x)

-- Note that being a Join(OrderedSet,IntegralDomain) is not the same
-- as being an OrderedIntegralDomain.
if S has OrderedIntegralDomain then
   if S has canonicalUnitNormal then
       x:% < y:% ==
         (numer x  * denom y) < (numer y * denom x)
     else
       x:% < y:% ==
         if denom(x) < 0 then (x,y):=(y,x)
         if denom(y) < 0 then (x,y):=(y,x)
         (numer x  * denom y) < (numer y * denom x)
else if S has OrderedSet then
   x:% < y:% ==
     (numer x  * denom y) < (numer y * denom x)

if (S has EuclideanDomain) then
  fractionPart x == x - (wholePart(x)::%)

if S has RetractableTo Symbol then
  coerce(s:Symbol):%  == s::S::%

  retract(x:%):Symbol == retract(retract(x)@S)

  retractIfCan(x:%):Union(Symbol, "failed") ==
    (r := retractIfCan(x)@Union(S,"failed")) case "failed" =>"failed"
    retractIfCan(r::S)

if (S has ConvertibleTo Pattern Integer) then
  convert(x:%):Pattern(Integer)==(convert numer x)/(convert denom x)

  if (S has PatternMatchable Integer) then
    patternMatch(x:%, p:Pattern Integer,
     l:PatternMatchResult(Integer, %)) ==
       patternMatch(x, p,
                 l)$PatternMatchQuotientFieldCategory(Integer, S, %)

if (S has ConvertibleTo Pattern Float) then
  convert(x:%):Pattern(Float) == (convert numer x)/(convert denom x)

  if (S has PatternMatchable Float) then
    patternMatch(x:%, p:Pattern Float,
     l:PatternMatchResult(Float, %)) ==
       patternMatch(x, p,
                   l)$PatternMatchQuotientFieldCategory(Float, S, %)

if S has RetractableTo Integer then
  coerce(x:Fraction Integer):% == numer(x)::% / denom(x)::%
```

```
      if not(S is Integer) then
        retract(x:%):Integer == retract(retract(x)@S)

        retractIfCan(x:%):Union(Integer, "failed") ==
          (u := retractIfCan(x)@Union(S, "failed")) case "failed" =>
            "failed"
          retractIfCan(u::S)

    if S has IntegerNumberSystem then
      random():% ==
        while zero?(d:=random()$S) repeat d
        random()$S / d

    reducedSystem(m:Matrix %, v:Vector %):
      Record(mat:Matrix S, vec:Vector S) ==
        n := reducedSystem(horizConcat(v::Matrix(%), m))@Matrix(S)
        [subMatrix(n, minRowIndex n, maxRowIndex n, 1 + minColIndex n,
                                  maxColIndex n), column(n, minColIndex n)]
```

———-

— **QFCAT.dotabb** —

```
"QFCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=QFCAT"];
"QFCAT" -> "ALGEBRA"
"QFCAT" -> "DIFEXT"
"QFCAT" -> "FIELD"
"QFCAT" -> "FEVALAB"
"QFCAT" -> "FLINEXP"
"QFCAT" -> "FPATMAB"
"QFCAT" -> "PATAB"
"QFCAT" -> "RETRACT"
```

———-

— **QFCAT.dotfull** —

```
"QuotientFieldCategory(a:IntegralDomain)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=QFCAT"];
"QuotientFieldCategory(a:IntegralDomain)" -> "Field()"
"QuotientFieldCategory(a:IntegralDomain)" -> "Algebra(IntegralDomain)"
"QuotientFieldCategory(a:IntegralDomain)" -> "RetractableTo(IntegralDomain)"
"QuotientFieldCategory(a:IntegralDomain)" ->
  "FullyEvalableOver(IntegralDomain)"
"QuotientFieldCategory(a:IntegralDomain)" ->
```

```
  "DifferentialExtension(IntegralDomain)"
"QuotientFieldCategory(a:IntegralDomain)" ->
  "FullyLinearlyExplicitRingOver(IntegralDomain)"
"QuotientFieldCategory(a:IntegralDomain)" ->
  "Patternable(IntegralDomain)"
"QuotientFieldCategory(a:IntegralDomain)" ->
  "FullyPatternMatchable(IntegralDomain)"
```

———-

— QFCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

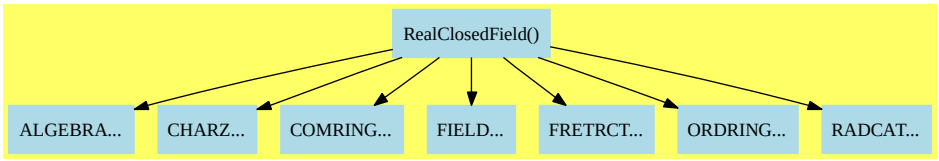
"QuotientFieldCategory(a:IntegralDomain)" [color=lightblue];
"QuotientFieldCategory(a:IntegralDomain)" -> "ALGEBRA..."
"QuotientFieldCategory(a:IntegralDomain)" -> "DIFEXT..."
"QuotientFieldCategory(a:IntegralDomain)" -> "FIELD..."
"QuotientFieldCategory(a:IntegralDomain)" -> "FEVALAB..."
"QuotientFieldCategory(a:IntegralDomain)" -> "FLINEXP..."
"QuotientFieldCategory(a:IntegralDomain)" -> "FPATMAB..."
"QuotientFieldCategory(a:IntegralDomain)" -> "PATAB..."
"QuotientFieldCategory(a:IntegralDomain)" -> "RETRACT..."

"ALGEBRA..." [color=lightblue];
"DIFEXT..." [color=lightblue];
"FIELD..." [color=lightblue];
"FEVALAB..." [color=lightblue];
"FLINEXP..." [color=lightblue];
"FPATMAB..." [color=lightblue];
"PATAB..." [color=lightblue];
"RETRACT..." [color=lightblue];

}
```

———-

17.0.199 RealClosedField (RCFIELD)



— RealClosedField.input —

```
)set break resume
)sys rm -f RealClosedField.output
)spool RealClosedField.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show RealClosedField
--R RealClosedField  is a category constructor
--R Abbreviation for RealClosedField is RCFIELD
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for RCFIELD
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,Fraction Integer) -> %       ?*? : (Fraction Integer,%) -> %
--R ?*? : (%,Integer) -> %                ?*? : (Integer,%) -> %
--R ?*? : (%,Fraction Integer) -> %       ?*? : (Fraction Integer,%) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,Fraction Integer) -> %
--R ?**? : (%,Integer) -> %               ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?/? : (%,%) -> %
--R ?<? : (%,%) -> Boolean                ?<=? : (%,%) -> Boolean
--R ?=? : (%,%) -> Boolean                ?>? : (%,%) -> Boolean
--R ?>=? : (%,%) -> Boolean               1 : () -> %
--R 0 : () -> %                           ?^? : (%,Integer) -> %
--R ?^? : (%,PositiveInteger) -> %        abs : % -> %
--R associates? : (%,%) -> Boolean        coerce : Fraction Integer -> %
--R coerce : Integer -> %                 coerce : Fraction Integer -> %
--R coerce : % -> %                       coerce : Fraction Integer -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R factor : % -> Factored %              gcd : (%,%) -> %
--R gcd : List % -> %                     hash : % -> SingleInteger
--R inv : % -> %                          latex : % -> String
--R lcm : (%,%) -> %                      lcm : List % -> %
--R max : (%,%) -> %                      min : (%,%) -> %
--R negative? : % -> Boolean              nthRoot : (%,Integer) -> %
```

```
--R one? : % -> Boolean                  positive? : % -> Boolean
--R prime? : % -> Boolean                ?quo? : (%,%) -> %
--R recip : % -> Union(%,"failed")       ?rem? : (%,%) -> %
--R rename : (%,OutputForm) -> %         rename! : (%,OutputForm) -> %
--R retract : % -> Fraction Integer      sample : () -> %
--R sign : % -> Integer                  sizeLess? : (%,%) -> Boolean
--R sqrt : Integer -> %                  sqrt : Fraction Integer -> %
--R sqrt : % -> %                        squareFree : % -> Factored %
--R squareFreePart : % -> %              unit? : % -> Boolean
--R unitCanonical : % -> %               zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R allRootsOf : Polynomial Integer -> List %
--R allRootsOf : Polynomial Fraction Integer -> List %
--R allRootsOf : Polynomial % -> List %
--R allRootsOf : SparseUnivariatePolynomial Integer -> List %
--R allRootsOf : SparseUnivariatePolynomial Fraction Integer -> List %
--R allRootsOf : SparseUnivariatePolynomial % -> List %
--R approximate : (%,%) -> Fraction Integer
--R characteristic : () -> NonNegativeInteger
--R divide : (%,%) -> Record(quotient: %,remainder: %)
--R euclideanSize : % -> NonNegativeInteger
--R expressIdealMember : (List %,%) -> Union(List %,"failed")
--R exquo : (%,%) -> Union(%,"failed")
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivariatePolynomia
--R mainDefiningPolynomial : % -> Union(SparseUnivariatePolynomial %,"failed")
--R mainForm : % -> Union(OutputForm,"failed")
--R mainValue : % -> Union(SparseUnivariatePolynomial %,"failed")
--R multiEuclidean : (List %,%) -> Union(List %,"failed")
--R principalIdeal : List % -> Record(coef: List %,generator: %)
--R retract : % -> Fraction Integer if Fraction Integer has RETRACT FRAC INT
--R retract : % -> Integer if Fraction Integer has RETRACT INT
--R retractIfCan : % -> Union(Fraction Integer,"failed")
--R retractIfCan : % -> Union(Fraction Integer,"failed") if Fraction Integer has RETRACT FRAC INT
--R retractIfCan : % -> Union(Integer,"failed") if Fraction Integer has RETRACT INT
--R rootOf : (SparseUnivariatePolynomial %,PositiveInteger) -> Union(%,"failed")
--R rootOf : (SparseUnivariatePolynomial %,PositiveInteger,OutputForm) -> Union(%,"failed")
--R sqrt : (%,NonNegativeInteger) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R
--E 1

)spool
)lisp (bye)
```

———-

— **RealClosedField.help** —

```
==================================================================
RealClosedField examples
==================================================================

See Also:
o )show RealClosedField
```

———-

See:

⇐ “Algebra” (ALGEBRA) 11.0.152 on page 947
⇐ “CharacteristicZero” (CHARZ) 10.0.136 on page 838
⇐ “CommutativeRing” (COMRING) 10.0.137 on page 843
⇐ “Field” (FIELD) 16.0.185 on page 1221
⇐ “FullyRetractableTo” (FRETRCT) 3.0.27 on page 87
⇐ “OrderedRing” (ORDRING) 10.0.144 on page 879
⇐ “RadicalCategory” (RADCAT) 2.0.17 on page 52

Exports:

0	1	abs	allRootsOf
approximate	associates?	characteristic	coerce
divide	euclideanSize	expressIdealMember	exquo
extendedEuclidean	factor	gcd	gcdPolynomial
hash	inv	latex	lcm
mainDefiningPolynomial	mainForm	mainValue	max
min	multiEuclidean	negative?	nthRoot
one?	positive?	prime?	principalIdeal
recip	rename	rename!	retract
retractIfCan	rootOf	sample	sign
sizeLess?	sqrt	squareFree	squareFreePart
subtractIfCan	unit?	unitCanonical	unitNormal
zero?	?*?	?**?	?+?
?-?	-?	?/?	?<?
?<=?	?=?	?>?	?>=?
?^?	?~=?	?quo?	?rem?

Attributes Exported:

- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **canonicalUnitNormal** is true if we can choose a canonical representative for each

class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.

- **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.

These are directly exported but not implemented:

```
allRootsOf : SparseUnivariatePolynomial % -> List %
approximate : (%,%) -> Fraction Integer
mainDefiningPolynomial :
   % -> Union(SparseUnivariatePolynomial %,"failed")
mainForm : % -> Union(OutputForm,"failed")
mainValue : % -> Union(SparseUnivariatePolynomial %,"failed")
rename : (%,OutputForm) -> %
rename! : (%,OutputForm) -> %
```

These are implemented by this category:

```
allRootsOf : Polynomial Integer -> List %
allRootsOf : Polynomial Fraction Integer -> List %
allRootsOf : Polynomial % -> List %
allRootsOf : SparseUnivariatePolynomial Integer -> List %
allRootsOf : SparseUnivariatePolynomial Fraction Integer -> List %
characteristic : () -> NonNegativeInteger
nthRoot : (%,Integer) -> %
rootOf :
  (SparseUnivariatePolynomial %,PositiveInteger) ->
     Union(%,"failed")
rootOf :
  (SparseUnivariatePolynomial %,PositiveInteger,OutputForm) ->
     Union(%,"failed")
sqrt : (%,NonNegativeInteger) -> %
sqrt : Integer -> %
sqrt : Fraction Integer -> %
sqrt : % -> %
?**? : (%,Fraction Integer) -> %
```

These exports come from (p838) CharacteristicZero():

```
0 : () -> %
1 : () -> %
coerce : Integer -> %
coerce : % -> OutputForm
hash : % -> SingleInteger
latex : % -> String
one? : % -> Boolean
recip : % -> Union(%,"failed")
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
zero? : % -> Boolean
?~=? : (%,%) -> Boolean
?^? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?*? : (%,%) -> %
?*? : (NonNegativeInteger,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?+? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
?=? : (%,%) -> Boolean
```

These exports come from (p879) OrderedRing():

```
abs : % -> %
coerce : Integer -> %
max : (%,%) -> %
min : (%,%) -> %
negative? : % -> Boolean
positive? : % -> Boolean
sign : % -> Integer
?<? : (%,%) -> Boolean
?<=? : (%,%) -> Boolean
?>? : (%,%) -> Boolean
?>=? : (%,%) -> Boolean
?*? : (Integer,%) -> %
```

These exports come from (p1221) Field():

```
associates? : (%,%) -> Boolean
coerce : % -> %
coerce : Fraction Integer -> %
coerce : Fraction Integer -> %
coerce : Fraction Integer -> %
divide : (%,%) -> Record(quotient: %,remainder: %)
euclideanSize : % -> NonNegativeInteger
expressIdealMember : (List %,%) -> Union(List %,"failed")
```

```
exquo : (%,%) -> Union(%,"failed")
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
factor : % -> Factored %
gcd : (%,%) -> %
gcd : List % -> %
gcdPolynomial :
  (SparseUnivariatePolynomial %,
   SparseUnivariatePolynomial %) ->
      SparseUnivariatePolynomial %
inv : % -> %
lcm : (%,%) -> %
lcm : List % -> %
multiEuclidean : (List %,%) -> Union(List %,"failed")
prime? : % -> Boolean
principalIdeal : List % -> Record(coef: List %,generator: %)
sizeLess? : (%,%) -> Boolean
squareFree : % -> Factored %
squareFreePart : % -> %
unit? : % -> Boolean
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
?/? : (%,%) -> %
?*? : (Fraction Integer,%) -> %
?*? : (Fraction Integer,%) -> %
?*? : (%,Fraction Integer) -> %
?*? : (%,Fraction Integer) -> %
?**? : (%,Integer) -> %
?^? : (%,Integer) -> %
?quo? : (%,%) -> %
?rem? : (%,%) -> %
```

These exports come from (p87) FullyRetractableTo(Fraction(Integer)):

```
retract : % -> Fraction Integer
retract : % -> Fraction Integer
  if Fraction Integer has RETRACT FRAC INT
retract : % -> Integer if Fraction Integer has RETRACT INT
retractIfCan : % -> Union(Fraction Integer,"failed")
retractIfCan : % -> Union(Fraction Integer,"failed")
  if Fraction Integer has RETRACT FRAC INT
retractIfCan : % -> Union(Integer,"failed")
  if Fraction Integer has RETRACT INT
```

These exports come from (p947) Algebra(Integer):

```
?*? : (%,Integer) -> %
```

— category RCFIELD RealClosedField —

```
)abbrev category RCFIELD RealClosedField
++ Author: Renaud Rioboo
++ Date Created: may 1993
++ Date Last Updated: January 2004
++ Basic Functions: provides computations with generic real roots of
++                  polynomials
++ Related Constructors: SimpleOrderedAlgebraicExtension, RealClosure
++ Also See:
++ AMS Classifications:
++ Keywords: Real Algebraic Numbers
++ References:
++ Description:
++ \axiomType{RealClosedField} provides common acces
++ functions for all real closed fields.

RealClosedField : Category == PUB where

    E ==> OutputForm
    SUP ==> SparseUnivariatePolynomial
    OFIELD ==> Join(OrderedRing,Field)
    PME ==> SUP($)
    N ==> NonNegativeInteger
    PI ==> PositiveInteger
    RN ==> Fraction(Integer)
    Z  ==> Integer
    POLY ==> Polynomial
    PACK ==> SparseUnivariatePolynomialFunctions2

    PUB == Join(CharacteristicZero,
                OrderedRing,
                CommutativeRing,
                Field,
                FullyRetractableTo(Fraction(Integer)),
                Algebra Integer,
                Algebra(Fraction(Integer)),
                RadicalCategory) with

        mainForm :   $ -> Union(E,"failed")
             ++ \axiom{mainForm(x)} is the main algebraic quantity name of
             ++ \axiom{x}

        mainDefiningPolynomial :   $ -> Union(PME,"failed")
             ++ \axiom{mainDefiningPolynomial(x)} is the defining
             ++ polynomial for the main algebraic quantity of \axiom{x}

        mainValue :   $ -> Union(PME,"failed")
             ++ \axiom{mainValue(x)} is the expression of \axiom{x} in terms
             ++ of \axiom{SparseUnivariatePolynomial($)}

        rootOf:          (PME,PI,E)           -> Union($,"failed")
```

```
     ++ \axiom{rootOf(pol,n,name)} creates the nth root for the order
     ++ of \axiom{pol} and names it \axiom{name}

rootOf:          (PME,PI)             -> Union($,"failed")
     ++ \axiom{rootOf(pol,n)} creates the nth root for the order
     ++ of \axiom{pol} and gives it unique name

allRootsOf:       PME                ->  List $
     ++ \axiom{allRootsOf(pol)} creates all the roots
     ++ of \axiom{pol} naming each uniquely

allRootsOf:       (SUP(RN))          ->  List $
     ++ \axiom{allRootsOf(pol)} creates all the roots
     ++ of \axiom{pol} naming each uniquely

allRootsOf:       (SUP(Z))           ->  List $
     ++ \axiom{allRootsOf(pol)} creates all the roots
     ++ of \axiom{pol} naming each uniquely

allRootsOf:       (POLY($))          ->  List $
     ++ \axiom{allRootsOf(pol)} creates all the roots
     ++ of \axiom{pol} naming each uniquely

allRootsOf:       (POLY(RN))         ->  List $
     ++ \axiom{allRootsOf(pol)} creates all the roots
     ++ of \axiom{pol} naming each uniquely

allRootsOf:       (POLY(Z))          ->  List $
     ++ \axiom{allRootsOf(pol)} creates all the roots
     ++ of \axiom{pol} naming each uniquely

sqrt:            ($,N)                ->     $
     ++ \axiom{sqrt(x,n)} is \axiom{x ** (1/n)}

sqrt:             $                   ->     $
     ++ \axiom{sqrt(x)} is \axiom{x ** (1/2)}

sqrt:             RN                  ->     $
     ++ \axiom{sqrt(x)} is \axiom{x ** (1/2)}

sqrt:             Z                   ->     $
     ++ \axiom{sqrt(x)} is \axiom{x ** (1/2)}

rename! :        ($,E)                ->     $
     ++ \axiom{rename!(x,name)} changes the way \axiom{x} is printed

rename :         ($,E)                ->     $
     ++ \axiom{rename(x,name)} gives a new number that prints as name

approximate:      ($,$) -> RN
```

```
        ++ \axiom{approximate(n,p)} gives an approximation of \axiom{n}
        ++ that has precision \axiom{p}

    add

      sqrt(a:$):$ == sqrt(a,2)

      sqrt(a:RN):$ == sqrt(a::$,2)

      sqrt(a:Z):$ == sqrt(a::$,2)

      characteristic() == 0

      rootOf(pol,n,o) ==
        r := rootOf(pol,n)
        r case "failed" => "failed"
        rename!(r,o)

      rootOf(pol,n) ==
        liste:List($):= allRootsOf(pol)
        # liste > n => "failed"
        liste.n


      sqrt(x,n) ==
        n = 0 => 1
        n = 1 => x
        zero?(x) => 0
        one?(x) => 1
        if odd?(n)
        then
          r := rootOf(monomial(1,n) - (x :: PME), 1)
        else
          r := rootOf(monomial(1,n) - (x :: PME), 2)
        r case "failed" => error "no roots"
        n = 2 => rename(r,root(x::E)$E)
        rename(r,root(x :: E, n :: E)$E)

      (x : $) ** (rn : RN) == sqrt(x**numer(rn),denom(rn)::N)

      nthRoot(x, n) ==
        zero?(n) => x
        negative?(n) => inv(sqrt(x,(-n) :: N))
        sqrt(x,n :: N)

      allRootsOf(p:SUP(RN)) == allRootsOf(map(z +-> z::$ ,p)$PACK(RN,$))

      allRootsOf(p:SUP(Z)) == allRootsOf(map(z +-> z::$ ,p)$PACK(Z,$))

      allRootsOf(p:POLY($)) == allRootsOf(univariate(p))
```

```
        allRootsOf(p:POLY(RN)) == allRootsOf(univariate(p))

        allRootsOf(p:POLY(Z)) == allRootsOf(univariate(p))
```

———-

— **RCFIELD.dotabb** —

```
"RCFIELD"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RCFIELD"];
"RCFIELD" -> "ALGEBRA"
"RCFIELD" -> "CHARZ"
"RCFIELD" -> "COMRING"
"RCFIELD" -> "FIELD"
"RCFIELD" -> "FRETRCT"
"RCFIELD" -> "ORDRING"
"RCFIELD" -> "RADCAT"
```

———-

— **RCFIELD.dotfull** —

```
"RealClosedField()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RCFIELD"];
"RealClosedField()" -> "Algebra(Integer)"
"RealClosedField()" -> "Algebra(Fraction(Integer))"
"RealClosedField()" -> "CharacteristicZero()"
"RealClosedField()" -> "CommutativeRing()"
"RealClosedField()" -> "Field()"
"RealClosedField()" -> "FullyRetractableTo(Fraction(Integer))"
"RealClosedField()" -> "OrderedRing()"
"RealClosedField()" -> "RadicalCategory()"
```

———-

— **RCFIELD.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"RealClosedField()" [color=lightblue];
"RealClosedField()" -> "ALGEBRA..."
```

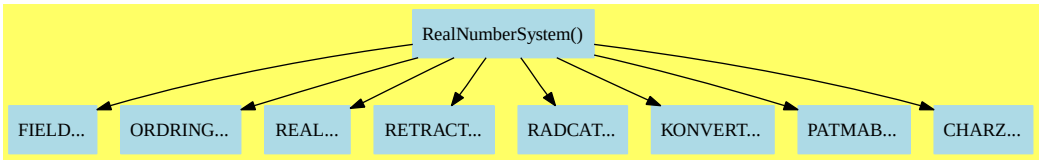
```
"RealClosedField()" -> "CHARZ..."
"RealClosedField()" -> "COMRING..."
"RealClosedField()" -> "FIELD..."
"RealClosedField()" -> "FRETRCT..."
"RealClosedField()" -> "ORDRING..."
"RealClosedField()" -> "RADCAT..."

"ALGEBRA..." [color=lightblue];
"CHARZ..." [color=lightblue];
"COMRING..." [color=lightblue];
"FIELD..." [color=lightblue];
"FRETRCT..." [color=lightblue];
"ORDRING..." [color=lightblue];
"RADCAT..." [color=lightblue];

}
```

———-

17.0.200 RealNumberSystem (RNS)



— RealNumberSystem.input —

```
)set break resume
)sys rm -f RealNumberSystem.output
)spool RealNumberSystem.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show RealNumberSystem
--R RealNumberSystem  is a category constructor
--R Abbreviation for RealNumberSystem is RNS
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for RNS
--R
--R------------------------------ Operations ------------------------------
```

```
--R ?*? : (Fraction Integer,%) -> %       ?*? : (%,Fraction Integer) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,Fraction Integer) -> %
--R ?**? : (%,Integer) -> %               ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?/? : (%,%) -> %
--R ?<? : (%,%) -> Boolean                ?<=? : (%,%) -> Boolean
--R ?=? : (%,%) -> Boolean                ?>? : (%,%) -> Boolean
--R ?>=? : (%,%) -> Boolean               1 : () -> %
--R 0 : () -> %                           ?^? : (%,Integer) -> %
--R ?^? : (%,PositiveInteger) -> %        abs : % -> %
--R associates? : (%,%) -> Boolean        ceiling : % -> %
--R coerce : Fraction Integer -> %        coerce : Integer -> %
--R coerce : Fraction Integer -> %        coerce : % -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R convert : % -> Pattern Float          convert : % -> DoubleFloat
--R convert : % -> Float                  factor : % -> Factored %
--R floor : % -> %                        fractionPart : % -> %
--R gcd : List % -> %                     gcd : (%,%) -> %
--R hash : % -> SingleInteger             inv : % -> %
--R latex : % -> String                   lcm : List % -> %
--R lcm : (%,%) -> %                      max : (%,%) -> %
--R min : (%,%) -> %                      negative? : % -> Boolean
--R norm : % -> %                         nthRoot : (%,Integer) -> %
--R one? : % -> Boolean                   positive? : % -> Boolean
--R prime? : % -> Boolean                 ?quo? : (%,%) -> %
--R recip : % -> Union(%,"failed")        ?rem? : (%,%) -> %
--R retract : % -> Fraction Integer       retract : % -> Integer
--R round : % -> %                        sample : () -> %
--R sign : % -> Integer                   sizeLess? : (%,%) -> Boolean
--R sqrt : % -> %                         squareFree : % -> Factored %
--R squareFreePart : % -> %               truncate : % -> %
--R unit? : % -> Boolean                  unitCanonical : % -> %
--R wholePart : % -> Integer              zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R divide : (%,%) -> Record(quotient: %,remainder: %)
--R euclideanSize : % -> NonNegativeInteger
--R expressIdealMember : (List %,%) -> Union(List %,"failed")
--R exquo : (%,%) -> Union(%,"failed")
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivariatePolynomia
--R multiEuclidean : (List %,%) -> Union(List %,"failed")
--R patternMatch : (%,Pattern Float,PatternMatchResult(Float,%)) -> PatternMatchResult(Float,%)
--R principalIdeal : List % -> Record(coef: List %,generator: %)
--R retractIfCan : % -> Union(Fraction Integer,"failed")
```

```
--R retractIfCan : % -> Union(Integer,"failed")
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R
--E 1

)spool
)lisp (bye)
```

———-

— **RealNumberSystem.help** —

```
=====================================================================
RealNumberSystem examples
=====================================================================

See Also:
o )show RealNumberSystem
```

———-

See:

⇒ “FloatingPointSystem” (FPS) 18.0.208 on page 1530
⇐ “CharacteristicZero” (CHARZ) 10.0.136 on page 838
⇐ “ConvertibleTo” (KONVERT) 2.0.8 on page 23
⇐ “Field” (FIELD) 16.0.185 on page 1221
⇐ “OrderedRing” (ORDRING) 10.0.144 on page 879
⇐ “PatternMatchable” (PATMAB) 4.0.58 on page 226
⇐ “RadicalCategory” (RADCAT) 2.0.17 on page 52
⇐ “RealConstant” (REAL) 3.0.32 on page 106
⇐ “RetractableTo” (RETRACT) 2.0.18 on page 55

Exports:

0	1	abs	associates?
ceiling	characteristic	coerce	convert
divide	euclideanSize	expressIdealMember	exquo
extendedEuclidean	factor	floor	fractionPart
gcd	gcdPolynomial	hash	inv
latex	lcm	max	min
multiEuclidean	negative?	norm	nthRoot
one?	patternMatch	positive?	prime?
principalIdeal	recip	retract	retractIfCan
round	sample	sign	sizeLess?
sqrt	squareFree	squareFreePart	subtractIfCan
truncate	unit?	unitCanonical	unitNormal
wholePart	zero?	?*?	?**?
?+?	?-?	-?	?/?
?<?	?<=?	?=?	?>?
?>=?	?^?	?quo?	?rem?
?~=?			

These are directly exported but not implemented:

```
abs : % -> %
wholePart : % -> Integer
```

These are implemented by this category:

```
characteristic : () -> NonNegativeInteger
ceiling : % -> %
coerce : Fraction Integer -> %
convert : % -> Pattern Float
floor : % -> %
fractionPart : % -> %
norm : % -> %
patternMatch :
  (%,Pattern Float,PatternMatchResult(Float,%)) ->
    PatternMatchResult(Float,%)
round : % -> %
truncate : % -> %
```

These exports come from (p1221) Field():

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean
coerce : % -> %
coerce : Integer -> %
coerce : Integer -> %
coerce : Fraction Integer -> %
coerce : % -> OutputForm
divide : (%,%) -> Record(quotient: %,remainder: %)
```

```
euclideanSize : % -> NonNegativeInteger
expressIdealMember : (List %,%) -> Union(List %,"failed")
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
exquo : (%,%) -> Union(%,"failed")
factor : % -> Factored %
gcd : List % -> %
gcd : (%,%) -> %
gcdPolynomial :
  (SparseUnivariatePolynomial %,
   SparseUnivariatePolynomial %) ->
      SparseUnivariatePolynomial %
hash : % -> SingleInteger
inv : % -> %
latex : % -> String
lcm : List % -> %
lcm : (%,%) -> %
multiEuclidean : (List %,%) -> Union(List %,"failed")
one? : % -> Boolean
prime? : % -> Boolean
principalIdeal : List % -> Record(coef: List %,generator: %)
recip : % -> Union(%,"failed")
sample : () -> %
sizeLess? : (%,%) -> Boolean
squareFree : % -> Factored %
squareFreePart : % -> %
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (Fraction Integer,%) -> %
?*? : (%,Fraction Integer) -> %
?**? : (%,Fraction Integer) -> %
?^? : (%,Integer) -> %
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?/? : (%,%) -> %
?quo? : (%,%) -> %
```

```
?rem? : (%,%) -> %
```

These exports come from (p879) OrderedRing():

```
negative? : % -> Boolean
positive? : % -> Boolean
sign : % -> Integer
max : (%,%) -> %
min : (%,%) -> %
?<? : (%,%) -> Boolean
?<=? : (%,%) -> Boolean
?>? : (%,%) -> Boolean
?>=? : (%,%) -> Boolean
```

These exports come from (p106) RealConstant():

```
convert : % -> DoubleFloat
convert : % -> Float
```

These exports come from (p55) RetractableTo(Integer):

```
retract : % -> Integer
retractIfCan : % -> Union(Integer,"failed")
```

These exports come from (p55) RetractableTo(Fraction(Integer)):

```
retract : % -> Fraction Integer
retractIfCan : % -> Union(Fraction Integer,"failed")
```

These exports come from (p52) RadicalCategory():

```
nthRoot : (%,Integer) -> %
sqrt : % -> %
```

These exports come from (p23) ConvertibleTo(Pattern(Float)):

These exports come from (p226) PatternMatchable(Float):

These exports come from (p838) CharacteristicZero():

— category RNS RealNumberSystem —

```
)abbrev category RNS RealNumberSystem
++ Author: Michael Monagan and Stephen M. Watt
++ Date Created:
++   January 1988
++ Change History:
++ Related Constructors:
++ Keywords: real numbers
++ Description:
++ The real number system category is intended as a model for the real
++ numbers.  The real numbers form an ordered normed field.  Note that
++ we have purposely not included \spadtype{DifferentialRing} or
++ the elementary functions (see \spadtype{TranscendentalFunctionCategory})
++ in the definition.

RealNumberSystem(): Category ==
  Join(Field, OrderedRing, RealConstant, RetractableTo Integer,
       RetractableTo Fraction Integer, RadicalCategory,
        ConvertibleTo Pattern Float, PatternMatchable Float,
          CharacteristicZero) with
    norm : % -> %
      ++ norm x returns the same as absolute value.
    ceiling : % -> %
      ++ ceiling x returns the small integer \spad{>= x}.
    floor: % -> %
      ++ floor x returns the largest integer \spad{<= x}.
    wholePart  : % -> Integer
      ++ wholePart x returns the integer part of x.
    fractionPart : % -> %
      ++ fractionPart x returns the fractional part of x.
    truncate: % -> %
      ++ truncate x returns the integer between x and 0 closest to x.
    round: % -> %
      ++ round x computes the integer closest to x.
    abs  : % -> %
      ++ abs x returns the absolute value of x.
 add
   characteristic() == 0

   fractionPart x == x - truncate x

   truncate x == (negative? x => -floor(-x); floor x)

   round x == (negative? x => truncate(x-1/2::%); truncate(x+1/2::%))

   norm x == abs x

   coerce(x:Fraction Integer):% == numer(x)::% / denom(x)::%

   convert(x:%):Pattern(Float)  == convert(x)@Float :: Pattern(Float)
```

```
   floor x ==
      x1 := (wholePart x) :: %
      x = x1 => x
      x < 0 => (x1 - 1)
      x1

   ceiling x ==
      x1 := (wholePart x)::%
      x = x1 => x
      x >= 0 => (x1 + 1)
      x1

   patternMatch(x, p, l) ==
     generic? p => addMatch(p, x, l)
     constant? p =>
       (r := retractIfCan(p)@Union(Float, "failed")) case Float =>
         convert(x)@Float = r::Float => l
         failed()
       failed()
     failed()
```

———-

— **RNS.dotabb** —

```
"RNS"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RNS"];
"RNS" -> "FIELD"
"RNS" -> "ORDRING"
"RNS" -> "REAL"
"RNS" -> "RETRACT"
"RNS" -> "RADCAT"
"RNS" -> "KONVERT"
"RNS" -> "PATMAB"
"RNS" -> "CHARZ"
```

———-

— **RNS.dotfull** —

```
"RealNumberSystem()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RNS"];
"RealNumberSystem()" -> "Field()"
"RealNumberSystem()" -> "OrderedRing()"
"RealNumberSystem()" -> "RealConstant()"
"RealNumberSystem()" -> "RetractableTo(Integer)"
```

```
"RealNumberSystem()" -> "RetractableTo(Fraction(Integer))"
"RealNumberSystem()" -> "RadicalCategory()"
"RealNumberSystem()" -> "ConvertibleTo(Pattern(Float))"
"RealNumberSystem()" -> "PatternMatchable(Float)"
"RealNumberSystem()" -> "CharacteristicZero()"
```

———-

— RNS.dotpic —

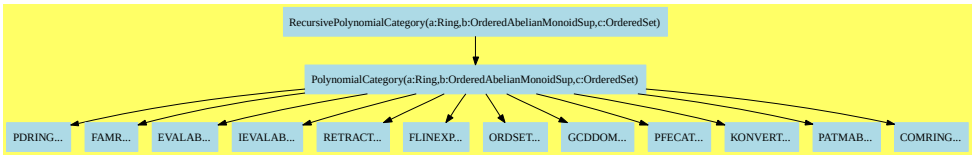
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"RealNumberSystem()" [color=lightblue];
"RealNumberSystem()" -> "FIELD..."
"RealNumberSystem()" -> "ORDRING..."
"RealNumberSystem()" -> "REAL..."
"RealNumberSystem()" -> "RETRACT..."
"RealNumberSystem()" -> "RADCAT..."
"RealNumberSystem()" -> "KONVERT..."
"RealNumberSystem()" -> "PATMAB..."
"RealNumberSystem()" -> "CHARZ..."

"FIELD..." [color=lightblue];
"ORDRING..." [color=lightblue];
"REAL..." [color=lightblue];
"RETRACT..." [color=lightblue];
"RADCAT..." [color=lightblue];
"KONVERT..." [color=lightblue];
"PATMAB..." [color=lightblue];
"CHARZ..." [color=lightblue];
}
```

———-

17.0.201 RecursivePolynomialCategory (RPOLCAT)



— RecursivePolynomialCategory.input —

```
)set break resume
)sys rm -f RecursivePolynomialCategory.output
)spool RecursivePolynomialCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show RecursivePolynomialCategory
--R RecursivePolynomialCategory(R: Ring,E: OrderedAbelianMonoidSup,V: OrderedSet)  is a category constructor
--R Abbreviation for RecursivePolynomialCategory is RPOLCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for RPOLCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,R) -> %                      ?*? : (R,%) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R D : (%,List V) -> %                   D : (%,V) -> %
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        coefficient : (%,E) -> R
--R coefficients : % -> List R            coerce : V -> %
--R coerce : R -> %                       coerce : Integer -> %
--R coerce : % -> OutputForm              deepestInitial : % -> %
--R deepestTail : % -> %                  degree : % -> E
--R differentiate : (%,List V) -> %       differentiate : (%,V) -> %
--R eval : (%,List V,List %) -> %         eval : (%,V,%) -> %
--R eval : (%,List V,List R) -> %         eval : (%,V,R) -> %
--R eval : (%,List %,List %) -> %         eval : (%,%,%) -> %
--R eval : (%,Equation %) -> %            eval : (%,List Equation %) -> %
--R ground : % -> R                       ground? : % -> Boolean
--R hash : % -> SingleInteger             head : % -> %
--R headReduce : (%,%) -> %               headReduced? : (%,%) -> Boolean
--R infRittWu? : (%,%) -> Boolean         init : % -> %
--R initiallyReduce : (%,%) -> %          iteratedInitials : % -> List %
--R latex : % -> String                   lazyPquo : (%,%,V) -> %
--R lazyPquo : (%,%) -> %                 lazyPrem : (%,%,V) -> %
--R lazyPrem : (%,%) -> %                 leadingCoefficient : (%,V) -> %
--R leadingCoefficient : % -> R           leadingMonomial : % -> %
--R leastMonomial : % -> %                mainCoefficients : % -> List %
--R mainMonomial : % -> %                 mainMonomials : % -> List %
--R map : ((R -> R),%) -> %               mapExponents : ((E -> E),%) -> %
--R mdeg : % -> NonNegativeInteger        minimumDegree : % -> E
--R monic? : % -> Boolean                 monicModulo : (%,%) -> %
--R monomial : (R,E) -> %                 monomial? : % -> Boolean
```

```
--R monomials : % -> List %                mvar : % -> V
--R normalized? : (%,%) -> Boolean         one? : % -> Boolean
--R pomopo! : (%,R,E,%) -> %               pquo : (%,%,V) -> %
--R pquo : (%,%) -> %                      prem : (%,%,V) -> %
--R prem : (%,%) -> %                      primitiveMonomials : % -> List %
--R quasiMonic? : % -> Boolean             recip : % -> Union(%,"failed")
--R reduced? : (%,List %) -> Boolean       reduced? : (%,%) -> Boolean
--R reductum : (%,V) -> %                  reductum : % -> %
--R retract : % -> V                       retract : % -> R
--R sample : () -> %                       supRittWu? : (%,%) -> Boolean
--R tail : % -> %                          variables : % -> List V
--R zero? : % -> Boolean                   ?~=? : (%,%) -> Boolean
--R ?*? : (Fraction Integer,%) -> % if R has ALGEBRA FRAC INT
--R ?*? : (%,Fraction Integer) -> % if R has ALGEBRA FRAC INT
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (%,R) -> % if R has FIELD
--R ?<? : (%,%) -> Boolean if R has ORDSET
--R ?<=? : (%,%) -> Boolean if R has ORDSET
--R ?>? : (%,%) -> Boolean if R has ORDSET
--R ?>=? : (%,%) -> Boolean if R has ORDSET
--R D : (%,List V,List NonNegativeInteger) -> %
--R D : (%,V,NonNegativeInteger) -> %
--R LazardQuotient : (%,%,NonNegativeInteger) -> % if R has INTDOM
--R LazardQuotient2 : (%,%,%,NonNegativeInteger) -> % if R has INTDOM
--R RittWuCompare : (%,%) -> Union(Boolean,"failed")
--R ?^? : (%,NonNegativeInteger) -> %
--R associates? : (%,%) -> Boolean if R has INTDOM
--R binomThmExpt : (%,%,NonNegativeInteger) -> % if R has COMRING
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if and(has($,CharacteristicNonZero),has(R,PolynomialFact
--R coefficient : (%,List V,List NonNegativeInteger) -> %
--R coefficient : (%,V,NonNegativeInteger) -> %
--R coerce : % -> % if R has INTDOM
--R coerce : Fraction Integer -> % if R has RETRACT FRAC INT or R has ALGEBRA FRAC INT
--R coerce : % -> Polynomial R if V has KONVERT SYMBOL
--R conditionP : Matrix % -> Union(Vector %,"failed") if and(has($,CharacteristicNonZero),has(R,
--R content : (%,V) -> % if R has GCDDOM
--R content : % -> R if R has GCDDOM
--R convert : % -> Polynomial R if V has KONVERT SYMBOL
--R convert : % -> String if R has RETRACT INT and V has KONVERT SYMBOL
--R convert : Polynomial R -> % if V has KONVERT SYMBOL
--R convert : Polynomial Integer -> % if not has(R,Algebra Fraction Integer) and R has ALGEBRA I
--R convert : Polynomial Fraction Integer -> % if R has ALGEBRA FRAC INT and V has KONVERT SYMBO
--R convert : % -> InputForm if V has KONVERT INFORM and R has KONVERT INFORM
--R convert : % -> Pattern Integer if V has KONVERT PATTERN INT and R has KONVERT PATTERN INT
--R convert : % -> Pattern Float if V has KONVERT PATTERN FLOAT and R has KONVERT PATTERN FLOAT
--R degree : (%,List V) -> List NonNegativeInteger
--R degree : (%,V) -> NonNegativeInteger
--R differentiate : (%,List V,List NonNegativeInteger) -> %
```

```
--R differentiate : (%,V,NonNegativeInteger) -> %
--R discriminant : (%,V) -> % if R has COMRING
--R exactQuotient : (%,%) -> % if R has INTDOM
--R exactQuotient : (%,R) -> % if R has INTDOM
--R exactQuotient! : (%,%) -> % if R has INTDOM
--R exactQuotient! : (%,R) -> % if R has INTDOM
--R exquo : (%,%) -> Union(%,"failed") if R has INTDOM
--R exquo : (%,R) -> Union(%,"failed") if R has INTDOM
--R extendedSubResultantGcd : (%,%) -> Record(gcd: %,coef1: %,coef2: %) if R has INTDOM
--R factor : % -> Factored % if R has PFECAT
--R factorPolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial % if R has PFECA
--R factorSquareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial % if R
--R gcd : (%,%) -> % if R has GCDDOM
--R gcd : List % -> % if R has GCDDOM
--R gcd : (R,%) -> R if R has GCDDOM
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivariatePolynomia
--R halfExtendedSubResultantGcd1 : (%,%) -> Record(gcd: %,coef1: %) if R has INTDOM
--R halfExtendedSubResultantGcd2 : (%,%) -> Record(gcd: %,coef2: %) if R has INTDOM
--R headReduced? : (%,List %) -> Boolean
--R initiallyReduced? : (%,List %) -> Boolean
--R initiallyReduced? : (%,%) -> Boolean
--R isExpt : % -> Union(Record(var: V,exponent: NonNegativeInteger),"failed")
--R isPlus : % -> Union(List %,"failed")
--R isTimes : % -> Union(List %,"failed")
--R lastSubResultant : (%,%) -> % if R has INTDOM
--R lazyPremWithDefault : (%,%,V) -> Record(coef: %,gap: NonNegativeInteger,remainder: %)
--R lazyPremWithDefault : (%,%) -> Record(coef: %,gap: NonNegativeInteger,remainder: %)
--R lazyPseudoDivide : (%,%,V) -> Record(coef: %,gap: NonNegativeInteger,quotient: %,remainder: %)
--R lazyPseudoDivide : (%,%) -> Record(coef: %,gap: NonNegativeInteger,quotient: %,remainder: %)
--R lazyResidueClass : (%,%) -> Record(polnum: %,polden: %,power: NonNegativeInteger)
--R lcm : (%,%) -> % if R has GCDDOM
--R lcm : List % -> % if R has GCDDOM
--R mainContent : % -> % if R has GCDDOM
--R mainPrimitivePart : % -> % if R has GCDDOM
--R mainSquareFreePart : % -> % if R has GCDDOM
--R mainVariable : % -> Union(V,"failed")
--R max : (%,%) -> % if R has ORDSET
--R min : (%,%) -> % if R has ORDSET
--R minimumDegree : (%,List V) -> List NonNegativeInteger
--R minimumDegree : (%,V) -> NonNegativeInteger
--R monicDivide : (%,%,V) -> Record(quotient: %,remainder: %)
--R monomial : (%,List V,List NonNegativeInteger) -> %
--R monomial : (%,V,NonNegativeInteger) -> %
--R multivariate : (SparseUnivariatePolynomial %,V) -> %
--R multivariate : (SparseUnivariatePolynomial R,V) -> %
--R nextsubResultant2 : (%,%,%,%) -> % if R has INTDOM
--R normalized? : (%,List %) -> Boolean
--R numberOfMonomials : % -> NonNegativeInteger
--R patternMatch : (%,Pattern Integer,PatternMatchResult(Integer,%)) -> PatternMatchResult(Integer,%) if V
--R patternMatch : (%,Pattern Float,PatternMatchResult(Float,%)) -> PatternMatchResult(Float,%) if V has PA
```

```
--R primPartElseUnitCanonical : % -> % if R has INTDOM
--R primPartElseUnitCanonical! : % -> % if R has INTDOM
--R prime? : % -> Boolean if R has PFECAT
--R primitivePart : (%,V) -> % if R has GCDDOM
--R primitivePart : % -> % if R has GCDDOM
--R primitivePart! : % -> % if R has GCDDOM
--R pseudoDivide : (%,%) -> Record(quotient: %,remainder: %)
--R reducedSystem : Matrix % -> Matrix R
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix R,vec: Vector R)
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer) if R
--R reducedSystem : Matrix % -> Matrix Integer if R has LINEXP INT
--R resultant : (%,%) -> % if R has INTDOM
--R resultant : (%,%,V) -> % if R has COMRING
--R retract : Polynomial R -> % if not has(R,Algebra Fraction Integer) and not has(R,Algebra Int
--R retract : Polynomial Integer -> % if not has(R,Algebra Fraction Integer) and R has ALGEBRA I
--R retract : Polynomial Fraction Integer -> % if R has ALGEBRA FRAC INT and V has KONVERT SYMBO
--R retract : % -> Integer if R has RETRACT INT
--R retract : % -> Fraction Integer if R has RETRACT FRAC INT
--R retractIfCan : Polynomial R -> Union(%,"failed") if not has(R,Algebra Fraction Integer) and
--R retractIfCan : Polynomial Integer -> Union(%,"failed") if not has(R,Algebra Fraction Integer
--R retractIfCan : Polynomial Fraction Integer -> Union(%,"failed") if R has ALGEBRA FRAC INT an
--R retractIfCan : % -> Union(V,"failed")
--R retractIfCan : % -> Union(Integer,"failed") if R has RETRACT INT
--R retractIfCan : % -> Union(Fraction Integer,"failed") if R has RETRACT FRAC INT
--R retractIfCan : % -> Union(R,"failed")
--R solveLinearPolynomialEquation : (List SparseUnivariatePolynomial %,SparseUnivariatePolynomia
--R squareFree : % -> Factored % if R has GCDDOM
--R squareFreePart : % -> % if R has GCDDOM
--R squareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial %
--R subResultantChain : (%,%) -> List % if R has INTDOM
--R subResultantGcd : (%,%) -> % if R has INTDOM
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R totalDegree : (%,List V) -> NonNegativeInteger
--R totalDegree : % -> NonNegativeInteger
--R unit? : % -> Boolean if R has INTDOM
--R unitCanonical : % -> % if R has INTDOM
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %) if R has INTDOM
--R univariate : % -> SparseUnivariatePolynomial R
--R univariate : (%,V) -> SparseUnivariatePolynomial %
--R
--E 1

)spool
)lisp (bye)
```

———-

— RecursivePolynomialCategory.help —

```
==========================================================================
RecursivePolynomialCategory examples
==========================================================================

See Also:
o )show RecursivePolynomialCategory
```

———-

See:

⇐ “PolynomialCategory” (POLYCAT) 16.0.189 on page 1255

Exports:

0	1	associates?
binomThmExpt	characteristic	charthRoot
coefficient	coefficients	coerce
conditionP	convert	D
deepestInitial	deepestTail	degree
differentiate	discriminant	eval
exactQuotient	exactQuotient!	exquo
extendedSubResultantGcd	factor	factorPolynomial
factorSquareFreePolynomial	gcd	gcdPolynomial
ground	ground?	halfExtendedSubResultantGcd1
halfExtendedSubResultantGcd2	hash	head
headReduce	headReduced?	infRittWu?
init	initiallyReduce	initiallyReduced?
isExpt	isPlus	isTimes
iteratedInitials	lastSubResultant	latex
LazardQuotient	LazardQuotient2	lazyPquo
lazyPrem	lazyPremWithDefault	lazyPseudoDivide
lazyResidueClass	lcm	leadingCoefficient
leadingMonomial	leastMonomial	mainCoefficients
mainContent	mainMonomial	mainPrimitivePart
mainSquareFreePart	mainVariable	map
mapExponents	max	mdeg
min	minimumDegree	monic?
monicDivide	monicModulo	monomial
monomial?	monomials	multivariate
mvar	nextsubResultant2	normalized?
numberOfMonomials	one?	patternMatch
pomopo!	pquo	prem
primPartElseUnitCanonical	primPartElseUnitCanonical!	prime?
primitiveMonomials	primitivePart	primitivePart!
pseudoDivide	quasiMonic?	recip
reduced?	reducedSystem	reductum
resultant	retract	retractIfCan
RittWuCompare	sample	solveLinearPolynomialEquation
squareFree	squareFreePart	squareFreePolynomial
subResultantChain	subResultantGcd	subtractIfCan
supRittWu?	tail	totalDegree
unit?	unitCanonical	unitNormal
univariate	variables	zero?
?*?	?**?	?+?
?-?	-?	?=?
?^?	?~=?	?/?
?<?	?<=?	?>?
?>=?		

Attributes Exported:

- if #1 has CommutativeRing then commutative("*") where **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- if #1 has IntegralDomain then noZeroDivisors where **noZeroDivisors** is true if $x*y \neq 0$ implies both x and y are non-zero.
- if #1 has canonicalUnitNormal then canonicalUnitNormal where **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
exactQuotient! : (%,R) -> % if R has INTDOM
extendedSubResultantGcd : (%,%) -> Record(gcd: %,coef1: %,coef2: %)
  if R has INTDOM
halfExtendedSubResultantGcd1 : (%,%) -> Record(gcd: %,coef1: %)
  if R has INTDOM
halfExtendedSubResultantGcd2 : (%,%) -> Record(gcd: %,coef2: %)
  if R has INTDOM
lastSubResultant : (%,%) -> % if R has INTDOM
LazardQuotient : (%,%,NonNegativeInteger) -> %
  if R has INTDOM
LazardQuotient2 : (%,%,%,NonNegativeInteger) -> %
  if R has INTDOM
nextsubResultant2 : (%,%,%,%) -> % if R has INTDOM
resultant : (%,%) -> % if R has INTDOM
subResultantChain : (%,%) -> List % if R has INTDOM
subResultantGcd : (%,%) -> % if R has INTDOM
```

These are implemented by this category:

```
coerce : % -> OutputForm
coerce : % -> Polynomial R if V has KONVERT SYMBOL
convert : % -> String
  if R has RETRACT INT
  and V has KONVERT SYMBOL
convert : % -> Polynomial R if V has KONVERT SYMBOL
convert : Polynomial R -> % if V has KONVERT SYMBOL
convert : Polynomial Integer -> %
  if not has(R,Algebra Fraction Integer)
  and R has ALGEBRA INT
  and V has KONVERT SYMBOL
```

```
    or R has ALGEBRA FRAC INT
    and V has KONVERT SYMBOL
  convert : Polynomial Fraction Integer -> %
    if R has ALGEBRA FRAC INT
    and V has KONVERT SYMBOL
  deepestInitial : % -> %
  deepestTail : % -> %
  exactQuotient : (%,R) -> % if R has INTDOM
  exactQuotient : (%,%) -> % if R has INTDOM
  exactQuotient! : (%,%) -> % if R has INTDOM
  gcd : (R,%) -> R if R has GCDDOM
  head : % -> %
  headReduce : (%,%) -> %
  headReduced? : (%,List %) -> Boolean
  headReduced? : (%,%) -> Boolean
  infRittWu? : (%,%) -> Boolean
  init : % -> %
  initiallyReduce : (%,%) -> %
  initiallyReduced? : (%,%) -> Boolean
  initiallyReduced? : (%,List %) -> Boolean
  iteratedInitials : % -> List %
  lazyPremWithDefault : (%,%) ->
    Record(coef: %,gap: NonNegativeInteger,remainder: %)
  lazyPremWithDefault : (%,%,V) ->
    Record(coef: %,gap: NonNegativeInteger,remainder: %)
  lazyPquo : (%,%) -> %
  lazyPquo : (%,%,V) -> %
  lazyPrem : (%,%,V) -> %
  lazyPrem : (%,%) -> %
  lazyPseudoDivide : (%,%) ->
    Record(coef: %,gap: NonNegativeInteger,quotient: %,remainder: %)
  lazyPseudoDivide : (%,%,V) ->
    Record(coef: %,gap: NonNegativeInteger,quotient: %,remainder: %)
  lazyResidueClass : (%,%) ->
    Record(polnum: %,polden: %,power: NonNegativeInteger)
  leadingCoefficient : (%,V) -> %
  leastMonomial : % -> %
  mainCoefficients : % -> List %
  mainContent : % -> % if R has GCDDOM
  mainMonomial : % -> %
  mainMonomials : % -> List %
  mainPrimitivePart : % -> % if R has GCDDOM
  mainSquareFreePart : % -> % if R has GCDDOM
  mdeg : % -> NonNegativeInteger
  monic? : % -> Boolean
  monicModulo : (%,%) -> %
  mvar : % -> V
  normalized? : (%,%) -> Boolean
  normalized? : (%,List %) -> Boolean
  pquo : (%,%) -> %
```

```
pquo : (%,%,V) -> %
prem : (%,%,V) -> %
prem : (%,%) -> %
primitivePart! : % -> % if R has GCDDOM
primPartElseUnitCanonical : % -> % if R has INTDOM
primPartElseUnitCanonical! : % -> % if R has INTDOM
pseudoDivide : (%,%) -> Record(quotient: %,remainder: %)
quasiMonic? : % -> Boolean
reduced? : (%,%) -> Boolean
reduced? : (%,List %) -> Boolean
reductum : (%,V) -> %
retract : Polynomial R -> %
  if not has(R,Algebra Fraction Integer)
  and not has(R,Algebra Integer)
  and V has KONVERT SYMBOL
  or not has(R,IntegerNumberSystem)
  and not has(R,Algebra Fraction Integer)
  and R has ALGEBRA INT
  and V has KONVERT SYMBOL
  or not has(R,QuotientFieldCategory Integer)
  and R has ALGEBRA FRAC INT
  and V has KONVERT SYMBOL
retract : Polynomial Integer -> %
  if not has(R,Algebra Fraction Integer)
  and R has ALGEBRA INT
  and V has KONVERT SYMBOL
  or R has ALGEBRA FRAC INT
  and V has KONVERT SYMBOL
retract : Polynomial Fraction Integer -> %
  if R has ALGEBRA FRAC INT and V has KONVERT SYMBOL
retractIfCan : Polynomial R -> Union(%,"failed")
  if not has(R,Algebra Fraction Integer)
  and not has(R,Algebra Integer)
  and V has KONVERT SYMBOL
  or not has(R,IntegerNumberSystem)
  and not has(R,Algebra Fraction Integer)
  and R has ALGEBRA INT
  and V has KONVERT SYMBOL
  or not has(R,QuotientFieldCategory Integer)
  and R has ALGEBRA FRAC INT
  and V has KONVERT SYMBOL
retractIfCan : Polynomial Fraction Integer -> Union(%,"failed")
  if R has ALGEBRA FRAC INT
  and V has KONVERT SYMBOL
retractIfCan : Polynomial Integer -> Union(%,"failed")
  if not has(R,Algebra Fraction Integer)
  and R has ALGEBRA INT
  and V has KONVERT SYMBOL
  or R has ALGEBRA FRAC INT
  and V has KONVERT SYMBOL
```

```
RittWuCompare : (%,%) -> Union(Boolean,"failed")
supRittWu? : (%,%) -> Boolean
tail : % -> %
```

These exports come from (p1255) PolynomialCategory(R,E,V)
where R:Ring, E:OrderedAbelianMonoidSup, V:OrderedSet:

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean
  if R has INTDOM
binomThmExpt : (%,%,NonNegativeInteger) -> %
  if R has COMRING
characteristic : () -> NonNegativeInteger
charthRoot : % -> Union(%,"failed")
  if and(has($,CharacteristicNonZero),
     has(R,PolynomialFactorizationExplicit))
  or R has CHARNZ
coefficient : (%,List V,List NonNegativeInteger) -> %
coefficient : (%,V,NonNegativeInteger) -> %
coefficient : (%,E) -> R
coefficients : % -> List R
coerce : R -> %
coerce : Fraction Integer -> %
  if R has RETRACT FRAC INT or R has ALGEBRA FRAC INT
coerce : V -> %
coerce : % -> % if R has INTDOM
coerce : Integer -> %
conditionP : Matrix % -> Union(Vector %,"failed")
  if and(has($,CharacteristicNonZero),
         has(R,PolynomialFactorizationExplicit))
content : % -> R if R has GCDDOM
content : (%,V) -> % if R has GCDDOM
convert : % -> Pattern Integer
  if V has KONVERT PATTERN INT and R has KONVERT PATTERN INT
convert : % -> Pattern Float
  if V has KONVERT PATTERN FLOAT and R has KONVERT PATTERN FLOAT
convert : % -> InputForm
  if V has KONVERT INFORM and R has KONVERT INFORM
D : (%,List V) -> %
D : (%,V) -> %
D : (%,List V,List NonNegativeInteger) -> %
D : (%,V,NonNegativeInteger) -> %
degree : % -> E
degree : (%,List V) -> List NonNegativeInteger
degree : (%,V) -> NonNegativeInteger
differentiate : (%,List V,List NonNegativeInteger) -> %
differentiate : (%,V,NonNegativeInteger) -> %
differentiate : (%,List V) -> %
differentiate : (%,V) -> %
```

```
discriminant : (%,V) -> % if R has COMRING
eval : (%,List Equation %) -> %
eval : (%,Equation %) -> %
eval : (%,List %,List %) -> %
eval : (%,%,%) -> %
eval : (%,V,R) -> %
eval : (%,List V,List R) -> %
eval : (%,V,%) -> %
eval : (%,List V,List %) -> %
exquo : (%,R) -> Union(%,"failed") if R has INTDOM
exquo : (%,%) -> Union(%,"failed") if R has INTDOM
factor : % -> Factored % if R has PFECAT
factorPolynomial :
  SparseUnivariatePolynomial % ->
    Factored SparseUnivariatePolynomial %
     if R has PFECAT
factorSquareFreePolynomial :
  SparseUnivariatePolynomial % ->
    Factored SparseUnivariatePolynomial %
      if R has PFECAT
gcd : (%,%) -> % if R has GCDDOM
gcd : List % -> % if R has GCDDOM
gcdPolynomial :
  (SparseUnivariatePolynomial %,
   SparseUnivariatePolynomial %) ->
     SparseUnivariatePolynomial %
       if R has GCDDOM
ground : % -> R
ground? : % -> Boolean
hash : % -> SingleInteger
isExpt : % -> Union(Record(var: V,exponent: NonNegativeInteger),"failed")
isPlus : % -> Union(List %,"failed")
isTimes : % -> Union(List %,"failed")
latex : % -> String
lcm : (%,%) -> % if R has GCDDOM
lcm : List % -> % if R has GCDDOM
leadingCoefficient : % -> R
leadingMonomial : % -> %
mainVariable : % -> Union(V,"failed")
map : ((R -> R),%) -> %
mapExponents : ((E -> E),%) -> %
max : (%,%) -> % if R has ORDSET
min : (%,%) -> % if R has ORDSET
minimumDegree : % -> E
minimumDegree : (%,List V) -> List NonNegativeInteger
minimumDegree : (%,V) -> NonNegativeInteger
monicDivide : (%,%,V) -> Record(quotient: %,remainder: %)
monomial : (%,V,NonNegativeInteger) -> %
monomial : (%,List V,List NonNegativeInteger) -> %
monomial : (R,E) -> %
```

```
monomial? : % -> Boolean
monomials : % -> List %
multivariate : (SparseUnivariatePolynomial %,V) -> %
multivariate : (SparseUnivariatePolynomial R,V) -> %
numberOfMonomials : % -> NonNegativeInteger
one? : % -> Boolean
patternMatch :
  (%,Pattern Integer,PatternMatchResult(Integer,%)) ->
    PatternMatchResult(Integer,%)
      if V has PATMAB INT and R has PATMAB INT
patternMatch :
  (%,Pattern Float,PatternMatchResult(Float,%)) ->
    PatternMatchResult(Float,%)
     if V has PATMAB FLOAT and R has PATMAB FLOAT
pomopo! : (%,R,E,%) -> %
prime? : % -> Boolean if R has PFECAT
primitiveMonomials : % -> List %
primitivePart : (%,V) -> % if R has GCDDOM
primitivePart : % -> % if R has GCDDOM
recip : % -> Union(%,"failed")
reducedSystem : Matrix % -> Matrix R
reducedSystem : (Matrix %,Vector %) ->
   Record(mat: Matrix R,vec: Vector R)
reducedSystem : (Matrix %,Vector %) ->
  Record(mat: Matrix Integer,vec: Vector Integer)
    if R has LINEXP INT
reducedSystem : Matrix % -> Matrix Integer
  if R has LINEXP INT
reductum : % -> %
resultant : (%,%,V) -> % if R has COMRING
retract : % -> R
retract : % -> Integer if R has RETRACT INT
retract : % -> Fraction Integer
  if R has RETRACT FRAC INT
retract : % -> V
retractIfCan : % -> Union(R,"failed")
retractIfCan : % -> Union(Integer,"failed")
  if R has RETRACT INT
retractIfCan : % -> Union(Fraction Integer,"failed")
  if R has RETRACT FRAC INT
retractIfCan : % -> Union(V,"failed")
sample : () -> %
solveLinearPolynomialEquation :
  (List SparseUnivariatePolynomial %,
   SparseUnivariatePolynomial %) ->
    Union(List SparseUnivariatePolynomial %,"failed")
      if R has PFECAT
squareFree : % -> Factored % if R has GCDDOM
squareFreePart : % -> % if R has GCDDOM
squareFreePolynomial :
```

```
  SparseUnivariatePolynomial % ->
    Factored SparseUnivariatePolynomial %
      if R has PFECAT
subtractIfCan : (%,%) -> Union(%,"failed")
totalDegree : (%,List V) -> NonNegativeInteger
totalDegree : % -> NonNegativeInteger
unit? : % -> Boolean if R has INTDOM
unitCanonical : % -> % if R has INTDOM
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
  if R has INTDOM
univariate : % -> SparseUnivariatePolynomial R
univariate : (%,V) -> SparseUnivariatePolynomial %
variables : % -> List V
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,R) -> %
?*? : (R,%) -> %
?*? : (Fraction Integer,%) -> % if R has ALGEBRA FRAC INT
?*? : (%,Fraction Integer) -> % if R has ALGEBRA FRAC INT
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?/? : (%,R) -> % if R has FIELD
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?<? : (%,%) -> Boolean if R has ORDSET
?<=? : (%,%) -> Boolean if R has ORDSET
?>? : (%,%) -> Boolean if R has ORDSET
?>=? : (%,%) -> Boolean if R has ORDSET
```

— category RPOLCAT RecursivePolynomialCategory —

```
)abbrev category RPOLCAT RecursivePolynomialCategory
++ Author: Marc Moreno Maza
++ Date Created: 04/22/1994
++ Date Last Updated: 14/12/1998
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords: polynomial, multivariate, ordered variables set
++ References:
++ Description:
```

```
++ A category for general multi-variate polynomials with coefficients
++ in a ring, variables in an ordered set, and exponents from an
++ ordered abelian monoid, with a \axiomOp{sup} operation.
++ When not constant, such a polynomial is viewed as a univariate polynomial
++ in its main variable w. r. t. to the total ordering on the elements in
++ the ordered set, so that some operations usually defined for univariate
++ polynomials make sense here.

RecursivePolynomialCategory(R:Ring, E:OrderedAbelianMonoidSup, V:OrderedSet):_
 Category ==
  PolynomialCategory(R, E, V) with
     mvar : $ -> V
         ++ \axiom{mvar(p)} returns an error if \axiom{p} belongs to
         ++ \axiom{R}, otherwise returns its main variable w. r. t. to the
         ++ total ordering on the elements in \axiom{V}.
     mdeg  : $ -> NonNegativeInteger
         ++ \axiom{mdeg(p)} returns an error if \axiom{p} is \axiom{0},
         ++ otherwise, if \axiom{p} belongs to \axiom{R} returns \axiom{0},
         ++ otherwise, returns the degree of \axiom{p} in its main variable.
     init : $ -> $
         ++ \axiom{init(p)} returns an error if \axiom{p} belongs to
         ++ \axiom{R}, otherwise returns its leading coefficient, where
         ++ \axiom{p} is viewed as a univariate polynomial in its main
         ++ variable.
     head  : $ -> $
         ++ \axiom{head(p)} returns \axiom{p} if \axiom{p} belongs to
         ++ \axiom{R}, otherwise returns its leading term (monomial in the
         ++ AXIOM sense), where \axiom{p} is viewed as a univariate polynomial
         ++  in its main variable.
     tail  : $ -> $
         ++ \axiom{tail(p)} returns its reductum, where \axiom{p} is viewed
         ++ as a univariate polynomial in its main variable.
     deepestTail : $ -> $
         ++ \axiom{deepestTail(p)} returns \axiom{0} if \axiom{p} belongs to
         ++ \axiom{R}, otherwise returns tail(p), if \axiom{tail(p)} belongs
         ++ to  \axiom{R} or \axiom{mvar(tail(p)) < mvar(p)}, otherwise
         ++ returns \axiom{deepestTail(tail(p))}.
     iteratedInitials : $ -> List $
         ++ \axiom{iteratedInitials(p)} returns \axiom{[]} if \axiom{p}
         ++ belongs to \axiom{R},
         ++ otherwise returns the list of the iterated initials of \axiom{p}.
     deepestInitial : $ -> $
         ++ \axiom{deepestInitial(p)} returns an error if \axiom{p} belongs
         ++ to \axiom{R},
         ++ otherwise returns the last term of \axiom{iteratedInitials(p)}.
     leadingCoefficient : ($,V) -> $
         ++ \axiom{leadingCoefficient(p,v)} returns the leading coefficient
         ++ of \axiom{p}, where \axiom{p} is viewed as A univariate
         ++ polynomial in \axiom{v}.
     reductum  : ($,V) -> $
```

```
    ++ \axiom{reductum(p,v)} returns the reductum of \axiom{p}, where
    ++ \axiom{p} is viewed as a univariate polynomial in \axiom{v}.
monic? : $ -> Boolean
    ++ \axiom{monic?(p)} returns false if \axiom{p} belongs to \axiom{R},
    ++ otherwise returns true iff \axiom{p} is monic as a univariate
    ++ polynomial in its main variable.
quasiMonic? : $ -> Boolean
    ++ \axiom{quasiMonic?(p)} returns false if \axiom{p} belongs to
    ++ \axiom{R}, otherwise returns true iff the initial of \axiom{p}
    ++ lies in the base ring \axiom{R}.
mainMonomial : $ -> $
    ++ \axiom{mainMonomial(p)} returns an error if \axiom{p} is
    ++ \axiom{0}, otherwise, if \axiom{p} belongs to \axiom{R} returns
    ++ \axiom{1}, otherwise, \axiom{mvar(p)} raised to the power
    ++ \axiom{mdeg(p)}.
leastMonomial : $ -> $
    ++ \axiom{leastMonomial(p)} returns an error if \axiom{p} is
    ++ \axiom{0}, otherwise, if \axiom{p} belongs to \axiom{R} returns
    ++ \axiom{1}, otherwise, the monomial of \axiom{p} with lowest
    ++ degree, where \axiom{p} is viewed as a univariate polynomial in
    ++ its main variable.
mainCoefficients : $ -> List $
    ++ \axiom{mainCoefficients(p)} returns an error if \axiom{p} is
    ++ \axiom{0}, otherwise, if \axiom{p} belongs to \axiom{R} returns
    ++ [p], otherwise returns the list of the coefficients of \axiom{p},
    ++ where \axiom{p} is viewed as a univariate polynomial in its main
    ++ variable.
mainMonomials : $ -> List $
    ++ \axiom{mainMonomials(p)} returns an error if \axiom{p} is
    ++ \axiom{0}, otherwise, if \axiom{p} belongs to \axiom{R} returns
    ++ [1], otherwise returns the list of the monomials of \axiom{p},
    ++ where \axiom{p} is viewed as a univariate polynomial in its main
    ++ variable.
RittWuCompare : ($, $) -> Union(Boolean,"failed")
    ++ \axiom{RittWuCompare(a,b)} returns \axiom{"failed"} if \axiom{a}
    ++ and \axiom{b} have same rank w.r.t.
    ++ Ritt and Wu Wen Tsun ordering using the refinement of Lazard,
    ++ otherwise returns \axiom{infRittWu?(a,b)}.
infRittWu?  : ($, $) -> Boolean
    ++ \axiom{infRittWu?(a,b)} returns true if \axiom{a} is less than
    ++ \axiom{b} w.r.t. the Ritt and Wu Wen Tsun ordering using the
    ++ refinement of Lazard.
supRittWu? : ($, $) -> Boolean
    ++ \axiom{supRittWu?(a,b)} returns true if \axiom{a} is greater
    ++ than \axiom{b} w.r.t. the Ritt and Wu Wen Tsun ordering using the
    ++ refinement of Lazard.
reduced? : ($,$) -> Boolean
    ++ \axiom{reduced?(a,b)} returns true iff
    ++ \axiom{degree(a,mvar(b)) < mdeg(b)}.
reduced? : ($,List($)) -> Boolean
```

```
    ++ \axiom{reduced?(q,lp)} returns true iff \axiom{reduced?(q,p)}
    ++ holds for every \axiom{p} in \axiom{lp}.
headReduced? : ($,$) -> Boolean
    ++ \axiom{headReduced?(a,b)} returns true iff
    ++ \axiom{degree(head(a),mvar(b)) < mdeg(b)}.
headReduced? : ($,List($)) -> Boolean
    ++ \axiom{headReduced?(q,lp)} returns true iff
    ++ \axiom{headReduced?(q,p)} holds for every \axiom{p} in \axiom{lp}.
initiallyReduced? : ($,$) -> Boolean
    ++ \axiom{initiallyReduced?(a,b)} returns false iff there exists an
    ++ iterated initial of \axiom{a} which is not reduced w.r.t \axiom{b}.
initiallyReduced? :  ($,List($)) -> Boolean
    ++ \axiom{initiallyReduced?(q,lp)} returns true iff
    ++ \axiom{initiallyReduced?(q,p)} holds for every \axiom{p} in
    ++ \axiom{lp}.
normalized? : ($,$) -> Boolean
    ++ \axiom{normalized?(a,b)} returns true iff \axiom{a} and its
    ++ iterated initials have degree zero w.r.t. the main variable of
    ++ \axiom{b}
normalized? : ($,List($)) -> Boolean
    ++ \axiom{normalized?(q,lp)} returns true iff
    ++ \axiom{normalized?(q,p)} holds
    ++ for every \axiom{p} in \axiom{lp}.
prem : ($, $) -> $
    ++ \axiom{prem(a,b)} computes the pseudo-remainder of \axiom{a} by
    ++ \axiom{b}, both viewed as univariate polynomials in the main
    ++ variable of \axiom{b}.
pquo : ($, $) -> $
    ++ \axiom{pquo(a,b)} computes the pseudo-quotient of \axiom{a} by
    ++ \axiom{b}, both viewed as univariate polynomials in the main
    ++ variable of \axiom{b}.
prem : ($, $, V) -> $
    ++ \axiom{prem(a,b,v)} computes the pseudo-remainder of \axiom{a}
    ++ by \axiom{b}, both viewed as univariate polynomials in \axiom{v}.
pquo : ($, $, V) -> $
    ++ \axiom{pquo(a,b,v)} computes the pseudo-quotient of \axiom{a} by
    ++ \axiom{b}, both viewed as univariate polynomials in \axiom{v}.
lazyPrem : ($, $) ->  $
    ++ \axiom{lazyPrem(a,b)} returns the polynomial \axiom{r} reduced
    ++ w.r.t. \axiom{b} and such that \axiom{b} divides
    ++ \axiom{init(b)^e a - r} where \axiom{e}
    ++ is the number of steps of this pseudo-division.
lazyPquo : ($, $) ->  $
    ++ \axiom{lazyPquo(a,b)} returns the polynomial \axiom{q} such that
    ++ \axiom{lazyPseudoDivide(a,b)} returns \axiom{[c,g,q,r]}.
lazyPrem : ($, $, V) -> $
    ++ \axiom{lazyPrem(a,b,v)} returns the polynomial \axiom{r}
    ++ reduced w.r.t. \axiom{b} viewed as univariate polynomials in the
    ++ variable \axiom{v} such that \axiom{b} divides
    ++ \axiom{init(b)^e a - r} where \axiom{e} is the number of steps of
```

```
    ++ this pseudo-division.
lazyPquo : ($, $, V) ->  $
    ++ \axiom{lazyPquo(a,b,v)} returns the polynomial \axiom{q} such that
    ++ \axiom{lazyPseudoDivide(a,b,v)} returns \axiom{[c,g,q,r]}.
lazyPremWithDefault : ($, $) -> _
  Record (coef : $, gap : NonNegativeInteger, remainder : $)
    ++ \axiom{lazyPremWithDefault(a,b)} returns \axiom{[c,g,r]}
    ++ such that \axiom{r = lazyPrem(a,b)} and
    ++ \axiom{(c**g)*r = prem(a,b)}.
lazyPremWithDefault : ($, $, V) -> _
  Record (coef : $, gap : NonNegativeInteger, remainder : $)
    ++ \axiom{lazyPremWithDefault(a,b,v)} returns \axiom{[c,g,r]}
    ++ such that \axiom{r = lazyPrem(a,b,v)} and
    ++ \axiom{(c**g)*r = prem(a,b,v)}.
lazyPseudoDivide : ($,$) -> _
  Record(coef:$, gap: NonNegativeInteger,quotient:$, remainder:$)
    ++ \axiom{lazyPseudoDivide(a,b)} returns \axiom{[c,g,q,r]}
    ++ such that \axiom{[c,g,r] = lazyPremWithDefault(a,b)} and
    ++ \axiom{q} is the pseudo-quotient computed in this lazy
    ++ pseudo-division.
lazyPseudoDivide : ($,$,V) -> _
  Record(coef:$, gap:NonNegativeInteger, quotient:$, remainder: $)
    ++ \axiom{lazyPseudoDivide(a,b,v)} returns \axiom{[c,g,q,r]} such
    ++ that  \axiom{r = lazyPrem(a,b,v)}, \axiom{(c**g)*r = prem(a,b,v)}
    ++ and \axiom{q} is the pseudo-quotient computed in this lazy
    ++ pseudo-division.
pseudoDivide : ($, $) -> Record (quotient : $, remainder : $)
    ++ \axiom{pseudoDivide(a,b)} computes \axiom{[pquo(a,b),prem(a,b)]},
    ++ both polynomials viewed as univariate polynomials in the main
    ++ variable of \axiom{b}, if \axiom{b} is not a constant polynomial.
monicModulo : ($, $) -> $
    ++ \axiom{monicModulo(a,b)} computes \axiom{a mod b}, if \axiom{b} is
    ++ monic as univariate polynomial in its main variable.
lazyResidueClass : ($,$) -> _
 Record(polnum:$, polden:$, power:NonNegativeInteger)
    ++ \axiom{lazyResidueClass(a,b)} returns \axiom{[p,q,n]} where
    ++ \axiom{p / q**n} represents the residue class of \axiom{a}
    ++ modulo \axiom{b} and \axiom{p} is reduced w.r.t. \axiom{b} and
    ++ \axiom{q} is \axiom{init(b)}.
headReduce: ($, $) ->  $
    ++ \axiom{headReduce(a,b)} returns a polynomial \axiom{r} such that
    ++ \axiom{headReduced?(r,b)} holds and there exists an integer
    ++ \axiom{e} such that \axiom{init(b)^e a - r} is zero modulo
    ++ \axiom{b}.
initiallyReduce: ($, $) ->  $
    ++ \axiom{initiallyReduce(a,b)} returns a polynomial \axiom{r} such
    ++ that \axiom{initiallyReduced?(r,b)} holds and there exists an
    ++ integer \axiom{e} such that \axiom{init(b)^e a - r} is zero
    ++ modulo \axiom{b}.
```

```
if (V has ConvertibleTo(Symbol))
then
  CoercibleTo(Polynomial R)
  ConvertibleTo(Polynomial R)
  if R has Algebra Fraction Integer
    then
      retractIfCan : Polynomial Fraction Integer -> Union($,"failed")
          ++ \axiom{retractIfCan(p)} returns \axiom{p} as an element of
          ++ the current domain if all its variables belong to \axiom{V}.
      retract : Polynomial Fraction Integer -> $
          ++ \axiom{retract(p)} returns \axiom{p} as an element of the
          ++ current domain if \axiom{retractIfCan(p)} does not return
          ++ "failed", otherwise an error is produced.
      convert : Polynomial Fraction Integer -> $
          ++ \axiom{convert(p)} returns the same as \axiom{retract(p)}.
      retractIfCan : Polynomial Integer -> Union($,"failed")
          ++ \axiom{retractIfCan(p)} returns \axiom{p} as an element of
          ++ the current domain if all its variables belong to \axiom{V}.
      retract : Polynomial Integer -> $
          ++ \axiom{retract(p)} returns \axiom{p} as an element of the
          ++ current domain if \axiom{retractIfCan(p)} does not return
          ++ "failed", otherwise an error is produced.
      convert : Polynomial Integer -> $
          ++ \axiom{convert(p)} returns the same as \axiom{retract(p)}
      if not (R has QuotientFieldCategory(Integer))
        then
          retractIfCan : Polynomial R -> Union($,"failed")
            ++ \axiom{retractIfCan(p)} returns \axiom{p} as an element
            ++ of the current domain if all its variables belong to
            ++ \axiom{V}.
          retract : Polynomial R -> $
            ++ \axiom{retract(p)} returns \axiom{p} as an element of the
            ++ current domain if \axiom{retractIfCan(p)} does not
            ++ return "failed", otherwise an error is produced.
  if (R has Algebra Integer) and not(R has Algebra Fraction Integer)
    then
      retractIfCan : Polynomial Integer -> Union($,"failed")
          ++ \axiom{retractIfCan(p)} returns \axiom{p} as an element of
          ++ the current domain if all its variables belong to \axiom{V}.
      retract : Polynomial Integer -> $
          ++ \axiom{retract(p)} returns \axiom{p} as an element of the
          ++ current domain if \axiom{retractIfCan(p)} does not return
          ++ "failed", otherwise an error is produced.
      convert : Polynomial Integer -> $
          ++ \axiom{convert(p)} returns the same as \axiom{retract(p)}.
      if not (R has IntegerNumberSystem)
        then
          retractIfCan : Polynomial R -> Union($,"failed")
            ++ \axiom{retractIfCan(p)} returns \axiom{p} as an element
            ++ of the current domain if all its variables belong to
```

```
          ++ \axiom{V}.
        retract : Polynomial R -> $
          ++ \axiom{retract(p)} returns \axiom{p} as an element of the
          ++ current domain if \axiom{retractIfCan(p)} does not
          ++ return "failed", otherwise an error is produced.
  if not(R has Algebra Integer) and not(R has Algebra Fraction Integer)
    then
      retractIfCan : Polynomial R -> Union($,"failed")
        ++ \axiom{retractIfCan(p)} returns \axiom{p} as an element of
        ++ the current domain if all its variables belong to \axiom{V}.
      retract : Polynomial R -> $
        ++ \axiom{retract(p)} returns \axiom{p} as an element of the
        ++ current domain if \axiom{retractIfCan(p)} does not return
        ++ "failed", otherwise an error is produced.
  convert : Polynomial R -> $
    ++ \axiom{convert(p)} returns \axiom{p} as an element of the current
    ++ domain if all its variables belong to \axiom{V}, otherwise an
    ++ error is produced.

  if R has RetractableTo(Integer)
  then
    ConvertibleTo(String)

if R has IntegralDomain
then
  primPartElseUnitCanonical : $ -> $
      ++ \axiom{primPartElseUnitCanonical(p)} returns
      ++ \axiom{primitivePart(p)} if \axiom{R} is a gcd-domain,
      ++ otherwise \axiom{unitCanonical(p)}.
  primPartElseUnitCanonical! : $ -> $
      ++ \axiom{primPartElseUnitCanonical!(p)} replaces  \axiom{p}
      ++ by \axiom{primPartElseUnitCanonical(p)}.
  exactQuotient : ($,R) -> $
      ++ \axiom{exactQuotient(p,r)} computes the exact quotient of
      ++ \axiom{p} by \axiom{r}, which is assumed to be a divisor of
      ++ \axiom{p}. No error is returned if this exact quotient fails!
  exactQuotient! : ($,R) -> $
      ++ \axiom{exactQuotient!(p,r)} replaces \axiom{p} by
      ++ \axiom{exactQuotient(p,r)}.
  exactQuotient : ($,$) -> $
      ++ \axiom{exactQuotient(a,b)} computes the exact quotient of
      ++ \axiom{a} by \axiom{b}, which is assumed to be a divisor of
      ++ \axiom{a}. No error is returned if this exact quotient fails!
  exactQuotient! : ($,$) -> $
      ++ \axiom{exactQuotient!(a,b)} replaces \axiom{a} by
      ++ \axiom{exactQuotient(a,b)}
  subResultantGcd : ($, $) -> $
      ++ \axiom{subResultantGcd(a,b)} computes a gcd of \axiom{a} and
      ++ \axiom{b} where \axiom{a} and \axiom{b} are assumed to have the
      ++ same main variable \axiom{v} and are viewed as univariate
```

```
        ++ polynomials in \axiom{v} with coefficients in the fraction
        ++ field of the polynomial ring generated by their other variables
        ++ over \axiom{R}.
    extendedSubResultantGcd : ($, $) -> _
     Record (gcd : $, coef1 : $, coef2 : $)
        ++ \axiom{extendedSubResultantGcd(a,b)} returns \axiom{[ca,cb,r]}
        ++ such that \axiom{r} is \axiom{subResultantGcd(a,b)} and we have
        ++ \axiom{ca * a + cb * cb = r} .
    halfExtendedSubResultantGcd1: ($, $) -> Record (gcd : $, coef1 : $)
        ++ \axiom{halfExtendedSubResultantGcd1(a,b)} returns \axiom{[g,ca]}
        ++ if \axiom{extendedSubResultantGcd(a,b)} returns \axiom{[g,ca,cb]}
        ++ otherwise produces an error.
    halfExtendedSubResultantGcd2: ($, $) -> Record (gcd : $, coef2 : $)
        ++ \axiom{halfExtendedSubResultantGcd2(a,b)} returns \axiom{[g,cb]}
        ++ if \axiom{extendedSubResultantGcd(a,b)} returns \axiom{[g,ca,cb]}
        ++ otherwise produces an error.
    resultant  : ($, $) -> $
        ++ \axiom{resultant(a,b)} computes the resultant of \axiom{a} and
        ++ \axiom{b} where \axiom{a} and \axiom{b} are assumed to have the
        ++ same main variable \axiom{v} and are viewed as univariate
        ++ polynomials in \axiom{v}.
    subResultantChain : ($, $) -> List $
        ++ \axiom{subResultantChain(a,b)}, where \axiom{a} and \axiom{b}
        ++ are not contant polynomials with the same main variable, returns
        ++ the subresultant chain of \axiom{a} and \axiom{b}.
    lastSubResultant: ($, $) -> $
        ++ \axiom{lastSubResultant(a,b)} returns the last non-zero
        ++ subresultant of \axiom{a} and \axiom{b} where \axiom{a} and
        ++ \axiom{b} are assumed to have the same main variable \axiom{v}
        ++ and are viewed as univariate polynomials in \axiom{v}.
    LazardQuotient: ($, $, NonNegativeInteger) -> $
        ++ \axiom{LazardQuotient(a,b,n)} returns \axiom{a**n exquo b**(n-1)}
        ++ assuming that this quotient does not fail.
    LazardQuotient2: ($, $, $, NonNegativeInteger) -> $
        ++ \axiom{LazardQuotient2(p,a,b,n)} returns
        ++ \axiom{(a**(n-1) * p) exquo b**(n-1)}
        ++ assuming that this quotient does not fail.
    next_subResultant2: ($, $, $, $) -> $
        ++ \axiom{nextsubResultant2(p,q,z,s)} is the multivariate version
        ++ of the operation
        ++ next_sousResultant2 from PseudoRemainderSequence from
        ++ the \axiomType{PseudoRemainderSequence} constructor.

  if R has GcdDomain
  then
    gcd : (R,$) -> R
        ++ \axiom{gcd(r,p)} returns the gcd of \axiom{r} and the content
        ++ of \axiom{p}.
    primitivePart! : $ -> $
        ++ \axiom{primitivePart!(p)} replaces \axiom{p}  by its primitive
```

```
          ++ part.
       mainContent : $ -> $
          ++ \axiom{mainContent(p)} returns the content of \axiom{p} viewed
          ++ as a univariate polynomial in its main variable and with
          ++ coefficients in the polynomial ring generated by its other
          ++ variables over \axiom{R}.
       mainPrimitivePart : $ -> $
          ++ \axiom{mainPrimitivePart(p)} returns the primitive part of
          ++ \axiom{p} viewed as a univariate polynomial in its main
          ++ variable and with coefficients in the polynomial ring generated
          ++ by its other variables over \axiom{R}.
       mainSquareFreePart : $ -> $
          ++ \axiom{mainSquareFreePart(p)} returns the square free part of
          ++ \axiom{p} viewed as a univariate polynomial in its main
          ++ variable and with coefficients in the polynomial ring
          ++ generated by its other variables over \axiom{R}.

 add
     O ==> OutputForm
     NNI ==> NonNegativeInteger
     INT ==> Integer

     exactQuo : (R,R) -> R

     coerce(p:$):O ==
       ground? (p) => (ground(p))::O
--       if one?((ip := init(p)))
       if (((ip := init(p))) = 1)
         then
           if zero?((tp := tail(p)))
             then
--               if one?((dp := mdeg(p)))
               if (((dp := mdeg(p))) = 1)
                 then
                   return((mvar(p))::O)
                 else
                   return(((mvar(p))::O **$O (dp::O)))
             else
--               if one?((dp := mdeg(p)))
               if (((dp := mdeg(p))) = 1)
                 then
                   return((mvar(p))::O +$O (tp::O))
                 else
                   return(((mvar(p))::O **$O (dp::O)) +$O (tp::O))
          else
           if zero?((tp := tail(p)))
             then
--               if one?((dp := mdeg(p)))
               if (((dp := mdeg(p))) = 1)
                 then
```

```
                  return((ip::O) *$O  (mvar(p))::O)
                else
                  return((ip::O) *$O ((mvar(p))::O **$O (dp::O)))
            else
--              if one?(mdeg(p))
              if ((mdeg(p)) = 1)
                then
                  return(((ip::O) *$O  (mvar(p))::O) +$O (tp::O))
      ((ip)::O *$O ((mvar(p))::O **$O ((mdeg(p)::O))) +$O (tail(p)::O))

    mvar p ==
      ground?(p) => error"Error in mvar from RPOLCAT : #1 is constant."
      mainVariable(p)::V

    mdeg p ==
      ground?(p) => 0$NNI
      degree(p,mainVariable(p)::V)

    init p ==
      ground?(p) => error"Error in mvar from RPOLCAT : #1 is constant."
      v := mainVariable(p)::V
      coefficient(p,v,degree(p,v))

    leadingCoefficient (p,v) ==
      zero? (d := degree(p,v)) => p
      coefficient(p,v,d)

    head p ==
      ground? p => p
      v := mainVariable(p)::V
      d := degree(p,v)
      monomial(coefficient(p,v,d),v,d)

    reductum(p,v) ==
      zero? (d := degree(p,v)) => 0$$
      p - monomial(coefficient(p,v,d),v,d)

    tail p ==
      ground? p => 0$$
      p - head(p)

    deepestTail p ==
      ground? p => 0$$
      ground? tail(p) => tail(p)
      mvar(p) > mvar(tail(p)) => tail(p)
      deepestTail(tail(p))

    iteratedInitials p ==
      ground? p => []
      p := init(p)
```

```
  cons(p,iteratedInitials(p))

localDeepestInitial (p : $) : $ ==
  ground? p => p
  localDeepestInitial init p

deepestInitial p ==
  ground? p => _
    error"Error in deepestInitial from RPOLCAT : #1 is constant."
  localDeepestInitial init p

monic? p ==
  ground? p => false
  (recip(init(p))$$ case $)@Boolean

quasiMonic?  p ==
  ground? p => false
  ground?(init(p))

mainMonomial p ==
  zero? p => error"Error in mainMonomial from RPOLCAT : #1 is zero"
  ground? p => 1$$
  v := mainVariable(p)::V
  monomial(1$$,v,degree(p,v))

leastMonomial p ==
  zero? p => error"Error in leastMonomial from RPOLCAT : #1 is zero"
  ground? p => 1$$
  v := mainVariable(p)::V
  monomial(1$$,v,minimumDegree(p,v))

mainCoefficients p ==
  zero? p => error"Error in mainCoefficients from RPOLCAT : #1 is zero"
  ground? p => [p]
  v := mainVariable(p)::V
  coefficients(univariate(p,v)@SparseUnivariatePolynomial($))

mainMonomials p ==
  zero? p => error"Error in mainMonomials from RPOLCAT : #1 is zero"
  ground? p => [1$$]
  v := mainVariable(p)::V
  lm := monomials(univariate(p,v)@SparseUnivariatePolynomial($))
  [monomial(1$$,v,degree(m)) for m in lm]

RittWuCompare (a,b) ==
  (ground? b and  ground? a) => "failed"::Union(Boolean,"failed")
  ground? b => false::Union(Boolean,"failed")
  ground? a => true::Union(Boolean,"failed")
  mvar(a) < mvar(b) => true::Union(Boolean,"failed")
  mvar(a) > mvar(b) => false::Union(Boolean,"failed")
```

```
    mdeg(a) < mdeg(b) => true::Union(Boolean,"failed")
    mdeg(a) > mdeg(b) => false::Union(Boolean,"failed")
    lc := RittWuCompare(init(a),init(b))
    lc case Boolean => lc
    RittWuCompare(tail(a),tail(b))

  infRittWu? (a,b) ==
    lc : Union(Boolean,"failed") := RittWuCompare(a,b)
    lc case Boolean => lc::Boolean
    false

  supRittWu? (a,b) ==
    infRittWu? (b,a)

  prem (a:$, b:$)  : $ ==
    cP := lazyPremWithDefault (a,b)
    ((cP.coef) ** (cP.gap)) * cP.remainder

  pquo (a:$, b:$)  : $ ==
    cPS := lazyPseudoDivide (a,b)
    c := (cPS.coef) ** (cPS.gap)
    c * cPS.quotient

  prem (a:$, b:$, v:V) : $ ==
    cP := lazyPremWithDefault (a,b,v)
    ((cP.coef) ** (cP.gap)) * cP.remainder

  pquo (a:$, b:$, v:V)  : $ ==
    cPS := lazyPseudoDivide (a,b,v)
    c := (cPS.coef) ** (cPS.gap)
    c * cPS.quotient

  lazyPrem (a:$, b:$) : $ ==
    (not ground?(b)) and (monic?(b)) => monicModulo(a,b)
    (lazyPremWithDefault (a,b)).remainder

  lazyPquo (a:$, b:$) : $ ==
    (lazyPseudoDivide (a,b)).quotient

  lazyPrem (a:$, b:$, v:V) : $ ==
    zero? b => _
      error"Error in lazyPrem : ($,$,V) -> $ from RPOLCAT : #2 is zero"
    ground?(b) => 0$$
    (v = mvar(b)) => lazyPrem(a,b)
    dbv : NNI := degree(b,v)
    zero? dbv => 0$$
    dav : NNI  := degree(a,v)
    zero? dav => a
    test : INT := dav::INT - dbv
    lcbv : $ := leadingCoefficient(b,v)
```

```
    while not zero?(a) and not negative?(test) repeat
      lcav := leadingCoefficient(a,v)
      term := monomial(lcav,v,test::NNI)
      a := lcbv * a - term * b
      test := degree(a,v)::INT - dbv
    a

  lazyPquo (a:$, b:$, v:V) : $ ==
    (lazyPseudoDivide (a,b,v)).quotient

  headReduce (a:$,b:$) ==
    ground? b => error _
     "Error in headReduce : ($,$) -> Boolean from TSETCAT : #2 is constant"
    ground? a => a
    mvar(a) = mvar(b) => lazyPrem(a,b)
    while not reduced?((ha := head a),b) repeat
      lrc := lazyResidueClass(ha,b)
      if zero? tail(a)
        then
          a := lrc.polnum
        else
          a := lrc.polnum +  (lrc.polden)**(lrc.power) * tail(a)
    a

  initiallyReduce(a:$,b:$) ==
    ground? b => error _
"Error in initiallyReduce : ($,$) -> Boolean from TSETCAT : #2 is constant"
    ground? a => a
    v := mvar(b)
    mvar(a) = v => lazyPrem(a,b)
    ia := a
    ma := 1$$
    ta := 0$$
    while (not ground?(ia)) and (mvar(ia) >= mvar(b)) repeat
      if (mvar(ia) = mvar(b)) and (mdeg(ia) >= mdeg(b))
        then
          iamodb := lazyResidueClass(ia,b)
          ia := iamodb.polnum
          if not zero? ta
            then
              ta :=  (iamodb.polden)**(iamodb.power) * ta
      if zero? ia
        then
          ia := ta
          ma := 1$$
          ta := 0$$
        else
          if not ground?(ia)
            then
              ta := tail(ia) * ma + ta
```

```
            ma := mainMonomial(ia) * ma
            ia := init(ia)
    ia * ma + ta

  lazyPremWithDefault (a,b) ==
    ground?(b) => error _
      "Error in lazyPremWithDefault from RPOLCAT : #2 is constant"
    ground?(a) => [1$$,0$NNI,a]
    xa := mvar a
    xb := mvar b
    xa < xb => [1$$,0$NNI,a]
    lcb : $ := init b
    db : NNI := mdeg b
    test : INT := degree(a,xb)::INT - db
    delta : INT := max(test + 1$INT, 0$INT)
    if xa = xb
      then
        b := tail b
        while not zero?(a) and not negative?(test) repeat
          term := monomial(init(a),xb,test::NNI)
          a := lcb * tail(a) - term * b
          delta := delta - 1$INT
          test := degree(a,xb)::INT - db
      else
        while not zero?(a) and not negative?(test) repeat
          term := monomial(leadingCoefficient(a,xb),xb,test::NNI)
          a := lcb * a - term * b
          delta := delta - 1$INT
          test := degree(a,xb)::INT - db
    [lcb, (delta::NNI), a]

  lazyPremWithDefault (a,b,v) ==
    zero? b =>  error _
     "Error in lazyPremWithDefault : ($,$,V) -> $ from RPOLCAT : #2 is zero"
    ground?(b) => [b,1$NNI,0$$]
    (v = mvar(b)) => lazyPremWithDefault(a,b)
    dbv : NNI := degree(b,v)
    zero? dbv => [b,1$NNI,0$$]
    dav : NNI  := degree(a,v)
    zero? dav => [1$$,0$NNI,a]
    test : INT := dav::INT - dbv
    delta : INT := max(test + 1$INT, 0$INT)
    lcbv : $ := leadingCoefficient(b,v)
    while not zero?(a) and not negative?(test) repeat
      lcav := leadingCoefficient(a,v)
      term := monomial(lcav,v,test::NNI)
      a := lcbv * a - term * b
      delta := delta - 1$INT
      test := degree(a,v)::INT - dbv
    [lcbv, (delta::NNI), a]
```

```
pseudoDivide (a,b) ==
  cPS := lazyPseudoDivide (a,b)
  c := (cPS.coef) ** (cPS.gap)
  [c * cPS.quotient, c * cPS.remainder]

lazyPseudoDivide (a,b) ==
  ground?(b) => error _
     "Error in lazyPseudoDivide from RPOLCAT : #2 is constant"
  ground?(a) => [1$$,0$NNI,0$$,a]
  xa := mvar a
  xb := mvar b
  xa < xb => [1$$,0$NNI,0$$, a]
  lcb : $ := init b
  db : NNI := mdeg b
  q := 0$$
  test : INT := degree(a,xb)::INT - db
  delta : INT := max(test + 1$INT, 0$INT)
  if xa = xb
    then
      b := tail b
      while not zero?(a) and not negative?(test) repeat
        term := monomial(init(a),xb,test::NNI)
        a := lcb * tail(a) - term * b
        q := lcb * q + term
        delta := delta - 1$INT
        test := degree(a,xb)::INT - db
    else
      while not zero?(a) and not negative?(test) repeat
        term := monomial(leadingCoefficient(a,xb),xb,test::NNI)
        a := lcb * a - term * b
        q := lcb * q + term
        delta := delta - 1$INT
        test := degree(a,xb)::INT - db
  [lcb, (delta::NNI), q, a]

lazyPseudoDivide (a,b,v) ==
  zero? b =>  error _
    "Error in lazyPseudoDivide : ($,$,V) -> $ from RPOLCAT : #2 is zero"
  ground?(b) => [b,1$NNI,a,0$$]
  (v = mvar(b)) => lazyPseudoDivide(a,b)
  dbv : NNI := degree(b,v)
  zero? dbv => [b,1$NNI,a,0$$]
  dav : NNI  := degree(a,v)
  zero? dav => [1$$,0$NNI,0$$, a]
  test : INT := dav::INT - dbv
  delta : INT := max(test + 1$INT, 0$INT)
  lcbv : $ := leadingCoefficient(b,v)
  q := 0$$
  while not zero?(a) and not negative?(test) repeat
```

```
    lcav := leadingCoefficient(a,v)
    term := monomial(lcav,v,test::NNI)
    a := lcbv * a - term * b
    q := lcbv * q + term
    delta := delta - 1$INT
    test := degree(a,v)::INT - dbv
  [lcbv, (delta::NNI), q, a]

monicModulo (a,b) ==
  ground?(b) => error"Error in monicModulo from RPOLCAT : #2 is constant"
  rec : Union($,"failed")
  rec := recip((ib := init(b)))$$
  (rec case "failed")@Boolean => error _
    "Error in monicModulo from RPOLCAT : #2 is not monic"
  ground? a => a
  ib * ((lazyPremWithDefault ((rec::$) * a,(rec::$) * b)).remainder)

lazyResidueClass(a,b) ==
  zero? b => [a,1$$,0$NNI]
  ground? b => [0$$,1$$,0$NNI]
  ground? a => [a,1$$,0$NNI]
  xa := mvar a
  xb := mvar b
  xa < xb => [a,1$$,0$NNI]
  monic?(b) => [monicModulo(a,b),1$$,0$NNI]
  lcb : $ := init b
  db : NNI := mdeg b
  test : INT := degree(a,xb)::INT - db
  pow : NNI := 0
  if xa = xb
    then
      b := tail b
      while not zero?(a) and not negative?(test) repeat
        term := monomial(init(a),xb,test::NNI)
        a := lcb * tail(a) - term * b
        pow := pow + 1$NNI
        test := degree(a,xb)::INT - db
    else
      while not zero?(a) and not negative?(test) repeat
        term := monomial(leadingCoefficient(a,xb),xb,test::NNI)
        a := lcb * a - term * b
        pow := pow + 1$NNI
        test := degree(a,xb)::INT - db
  [a,lcb,pow]

reduced? (a:$,b:$) : Boolean ==
  degree(a,mvar(b)) < mdeg(b)

reduced? (p:$, lq : List($)) : Boolean ==
  ground? p => true
```

```
    while (not empty? lq) and (reduced?(p, first lq)) repeat
      lq := rest lq
    empty? lq

  headReduced? (a:$,b:$) : Boolean ==
    reduced?(head(a),b)

  headReduced? (p:$, lq : List($)) : Boolean ==
    reduced?(head(p),lq)

  initiallyReduced? (a:$,b:$) : Boolean ==
    ground? b => error _
"Error in initiallyReduced? : ($,$) -> Bool. from RPOLCAT : #2 is constant"
    ground?(a) => true
    mvar(a) < mvar(b) => true
    (mvar(a) = mvar(b)) => reduced?(a,b)
    initiallyReduced?(init(a),b)

  initiallyReduced? (p:$, lq : List($)) : Boolean ==
    ground? p => true
    while (not empty? lq) and (initiallyReduced?(p, first lq)) repeat
      lq := rest lq
    empty? lq

  normalized?(a:$,b:$) : Boolean ==
    ground? b => error _
   "Error in  normalized? : ($,$) -> Boolean from TSETCAT : #2 is constant"
    ground? a => true
    mvar(a) < mvar(b) => true
    (mvar(a) = mvar(b)) => false
    normalized?(init(a),b)

  normalized? (p:$, lq : List($)) : Boolean ==
    while (not empty? lq) and (normalized?(p, first lq)) repeat
      lq := rest lq
    empty? lq

  if R has IntegralDomain
  then

    if R has EuclideanDomain
      then
        exactQuo(r:R,s:R):R ==
          r quo$R s
      else
        exactQuo(r:R,s:R):R ==
          (r exquo$R s)::R

    exactQuotient (p:$,r:R) ==
      (p exquo$$ r)::$
```

```
      exactQuotient (a:$,b:$) ==
        ground? b => exactQuotient(a,ground(b))
        (a exquo$$ b)::$

      exactQuotient! (a:$,b:$) ==
        ground? b => exactQuotient!(a,ground(b))
        a := (a exquo$$ b)::$

      if (R has GcdDomain) and not(R has Field)
      then

        primPartElseUnitCanonical p ==
          primitivePart p

        primitivePart! p ==
          zero? p => p
--          if one?(cp := content(p))
          if ((cp := content(p)) = 1)
            then
              p := unitCanonical p
            else
              p := unitCanonical exactQuotient!(p,cp)
          p

        primPartElseUnitCanonical! p ==
          primitivePart! p

      else
        primPartElseUnitCanonical p ==
          unitCanonical p

        primPartElseUnitCanonical! p ==
          p := unitCanonical p


    if R has GcdDomain
    then

      gcd(r:R,p:$):R ==
--          one? r => r
        (r = 1) => r
        zero? p => r
        ground? p => gcd(r,ground(p))$R
        gcd(gcd(r,init(p)),tail(p))

      mainContent p ==
        zero? p => p
        "gcd"/mainCoefficients(p)
```

```
  mainPrimitivePart p ==
    zero? p => p
    (unitNormal((p exquo$$ mainContent(p))::$)).canonical

  mainSquareFreePart p ==
    ground? p => p
    v := mainVariable(p)::V
    sfp : SparseUnivariatePolynomial($)
    sfp := squareFreePart(univariate(p,v)@SparseUnivariatePolynomial($))
    multivariate(sfp,v)

if (V has ConvertibleTo(Symbol))
  then

    PR ==> Polynomial R
    PQ ==> Polynomial Fraction Integer
    PZ ==> Polynomial Integer
    IES ==> IndexedExponents(Symbol)
    Q ==> Fraction Integer
    Z ==> Integer

    convert(p:$) : PR ==
      ground? p => (ground(p)$$)::PR
      v : V := mvar(p)
      d : NNI := mdeg(p)
      convert(init(p))@PR *$PR _
                  ((convert(v)@Symbol)::PR)**d +$PR convert(tail(p))@PR

    coerce(p:$) : PR ==
      convert(p)@PR

    localRetract : PR -> $
    localRetractPQ : PQ -> $
    localRetractPZ : PZ -> $
    localRetractIfCan : PR -> Union($,"failed")
    localRetractIfCanPQ : PQ -> Union($,"failed")
    localRetractIfCanPZ : PZ -> Union($,"failed")

    if V has Finite
      then

        sizeV : NNI := size()$V
        lv : List Symbol
        lv := _
          [convert(index(i::PositiveInteger)$V)@Symbol for i in 1..sizeV]

        localRetract(p : PR) : $ ==
          ground? p => (ground(p)$PR)::$
          mvp : Symbol := (mainVariable(p)$PR)::Symbol
          d : NNI
```

```
  imvp : PositiveInteger := _
              (position(mvp,lv)$(List Symbol))::PositiveInteger
  vimvp : V := index(imvp)$V
  xvimvp,c : $
  newp := 0$$
  while (not zero? (d := degree(p,mvp))) repeat
    c := localRetract(coefficient(p,mvp,d)$PR)
    xvimvp := monomial(c,vimvp,d)$$
    newp := newp +$$ xvimvp
    p := p -$PR monomial(coefficient(p,mvp,d)$PR,mvp,d)$PR
  newp +$$ localRetract(p)

if R has Algebra Fraction Integer
  then
    localRetractPQ(pq:PQ):$ ==
      ground? pq => ((ground(pq)$PQ)::R)::$
      mvp : Symbol := (mainVariable(pq)$PQ)::Symbol
      d : NNI
      imvp : PositiveInteger := _
              (position(mvp,lv)$(List Symbol))::PositiveInteger
      vimvp : V := index(imvp)$V
      xvimvp,c : $
      newp := 0$$
      while (not zero? (d := degree(pq,mvp))) repeat
        c := localRetractPQ(coefficient(pq,mvp,d)$PQ)
        xvimvp := monomial(c,vimvp,d)$$
        newp := newp +$$ xvimvp
        pq := pq -$PQ monomial(coefficient(pq,mvp,d)$PQ,mvp,d)$PQ
      newp +$$ localRetractPQ(pq)

if R has Algebra Integer
  then
    localRetractPZ(pz:PZ):$ ==
      ground? pz => ((ground(pz)$PZ)::R)::$
      mvp : Symbol := (mainVariable(pz)$PZ)::Symbol
      d : NNI
      imvp : PositiveInteger := _
              (position(mvp,lv)$(List Symbol))::PositiveInteger
      vimvp : V := index(imvp)$V
      xvimvp,c : $
      newp := 0$$
      while (not zero? (d := degree(pz,mvp))) repeat
        c := localRetractPZ(coefficient(pz,mvp,d)$PZ)
        xvimvp := monomial(c,vimvp,d)$$
        newp := newp +$$ xvimvp
        pz := pz -$PZ monomial(coefficient(pz,mvp,d)$PZ,mvp,d)$PZ
      newp +$$ localRetractPZ(pz)

retractable?(p:PR):Boolean ==
  lvp := variables(p)$PR
```

```
      while not empty? lvp and member?(first lvp,lv) repeat
        lvp := rest lvp
      empty? lvp

    retractablePQ?(p:PQ):Boolean ==
      lvp := variables(p)$PQ
      while not empty? lvp and member?(first lvp,lv) repeat
        lvp := rest lvp
      empty? lvp

    retractablePZ?(p:PZ):Boolean ==
      lvp := variables(p)$PZ
      while not empty? lvp and member?(first lvp,lv) repeat
        lvp := rest lvp
      empty? lvp

    localRetractIfCan(p : PR): Union($,"failed") ==
      not retractable?(p) => "failed"::Union($,"failed")
      localRetract(p)::Union($,"failed")

    localRetractIfCanPQ(p : PQ): Union($,"failed") ==
      not retractablePQ?(p) => "failed"::Union($,"failed")
      localRetractPQ(p)::Union($,"failed")

    localRetractIfCanPZ(p : PZ): Union($,"failed") ==
      not retractablePZ?(p) => "failed"::Union($,"failed")
      localRetractPZ(p)::Union($,"failed")

  if R has Algebra Fraction Integer
    then

    mpc2Z := MPolyCatFunctions2(Symbol,IES,IES,Z,R,PZ,PR)
    mpc2Q := MPolyCatFunctions2(Symbol,IES,IES,Q,R,PQ,PR)
    ZToR (z:Z):R == coerce(z)@R
    QToR (q:Q):R == coerce(q)@R
    PZToPR (pz:PZ):PR == map(ZToR,pz)$mpc2Z
    PQToPR (pq:PQ):PR == map(QToR,pq)$mpc2Q

    retract(pz:PZ) ==
      rif : Union($,"failed") := retractIfCan(pz)@Union($,"failed")
      (rif case "failed") => error _
                          "failed in retract: POLY Z -> $ from RPOLCAT"
      rif::$

    convert(pz:PZ) ==
      retract(pz)@$

    retract(pq:PQ) ==
      rif : Union($,"failed") := retractIfCan(pq)@Union($,"failed")
      (rif case "failed") => error _
```

```
                        "failed in retract: POLY Z -> $ from RPOLCAT"
  rif::$

convert(pq:PQ) ==
  retract(pq)@$

if not (R has QuotientFieldCategory(Integer))
  then
    -- the only operation to implement is
    -- retractIfCan : PR -> Union($,"failed")
    -- when V does not have Finite

    if V has Finite
      then
        retractIfCan(pr:PR) ==
          localRetractIfCan(pr)@Union($,"failed")

        retractIfCan(pq:PQ) ==
          localRetractIfCanPQ(pq)@Union($,"failed")
      else
        retractIfCan(pq:PQ) ==
          pr : PR := PQToPR(pq)
          retractIfCan(pr)@Union($,"failed")

    retractIfCan(pz:PZ) ==
      pr : PR := PZToPR(pz)
      retractIfCan(pr)@Union($,"failed")

    retract(pr:PR) ==
      rif : Union($,"failed") := _
                              retractIfCan(pr)@Union($,"failed")
      (rif case "failed") => error _
                   "failed in retract: POLY Z -> $ from RPOLCAT"
      rif::$

    convert(pr:PR) ==
      retract(pr)@$

  else
    -- the only operation to implement is
    -- retractIfCan : PQ -> Union($,"failed")
    -- when V does not have Finite
    mpc2ZQ := MPolyCatFunctions2(Symbol,IES,IES,Z,Q,PZ,PQ)
    mpc2RQ := MPolyCatFunctions2(Symbol,IES,IES,R,Q,PR,PQ)
    ZToQ(z:Z):Q == coerce(z)@Q
    RToQ(r:R):Q == retract(r)@Q

    PZToPQ (pz:PZ):PQ == map(ZToQ,pz)$mpc2ZQ
    PRToPQ (pr:PR):PQ == map(RToQ,pr)$mpc2RQ
```

```
        retractIfCan(pz:PZ) ==
          pq : PQ := PZToPQ(pz)
          retractIfCan(pq)@Union($,"failed")

        if V has Finite
          then
            retractIfCan(pq:PQ) ==
              localRetractIfCanPQ(pq)@Union($,"failed")

            convert(pr:PR) ==
              lrif : Union($,"failed") := _
                             localRetractIfCan(pr)@Union($,"failed")
              (lrif case "failed") => error _
                             "failed in convert: PR->$ from RPOLCAT"
              lrif::$
          else
            convert(pr:PR) ==
              pq : PQ := PRToPQ(pr)
              retract(pq)@$

if (R has Algebra Integer) and not(R has Algebra Fraction Integer)
  then

    mpc2Z := MPolyCatFunctions2(Symbol,IES,IES,Z,R,PZ,PR)
    ZToR (z:Z):R == coerce(z)@R
    PZToPR (pz:PZ):PR == map(ZToR,pz)$mpc2Z

    retract(pz:PZ) ==
      rif : Union($,"failed") := retractIfCan(pz)@Union($,"failed")
      (rif case "failed") => error _
                       "failed in retract: POLY Z -> $ from RPOLCAT"
      rif::$

    convert(pz:PZ) ==
      retract(pz)@$

    if not (R has IntegerNumberSystem)
      then
        -- the only operation to implement is
        -- retractIfCan : PR -> Union($,"failed")
        -- when V does not have Finite

        if V has Finite
          then
            retractIfCan(pr:PR) ==
              localRetractIfCan(pr)@Union($,"failed")

            retractIfCan(pz:PZ) ==
              localRetractIfCanPZ(pz)@Union($,"failed")
          else
```

```
            retractIfCan(pz:PZ) ==
              pr : PR := PZToPR(pz)
              retractIfCan(pr)@Union($,"failed")

          retract(pr:PR) ==
            rif : Union($,"failed"):=retractIfCan(pr)@Union($,"failed")
            (rif case "failed") => error _
                          "failed in retract: POLY Z -> $ from RPOLCAT"
            rif::$

          convert(pr:PR) ==
            retract(pr)@$

        else
          -- the only operation to implement is
          -- retractIfCan : PZ -> Union($,"failed")
          -- when V does not have Finite

          mpc2RZ := MPolyCatFunctions2(Symbol,IES,IES,R,Z,PR,PZ)
          RToZ(r:R):Z == retract(r)@Z
          PRToPZ (pr:PR):PZ == map(RToZ,pr)$mpc2RZ

          if V has Finite
            then
              convert(pr:PR) ==
                lrif : Union($,"failed") := _
                               localRetractIfCan(pr)@Union($,"failed")
                (lrif case "failed") => error _
                               "failed in convert: PR->$ from RPOLCAT"
                lrif::$
              retractIfCan(pz:PZ) ==
                localRetractIfCanPZ(pz)@Union($,"failed")
            else
              convert(pr:PR) ==
                pz : PZ := PRToPZ(pr)
                retract(pz)@$


    if not(R has Algebra Integer) and not(R has Algebra Fraction Integer)
      then
        -- the only operation to implement is
        -- retractIfCan : PR -> Union($,"failed")

        if V has Finite
          then
            retractIfCan(pr:PR) ==
              localRetractIfCan(pr)@Union($,"failed")

        retract(pr:PR) ==
          rif : Union($,"failed") := retractIfCan(pr)@Union($,"failed")
```

```
            (rif case "failed") => error _
                           "failed in retract: POLY Z -> $ from RPOLCAT"
            rif::$

          convert(pr:PR) ==
            retract(pr)@$

        if (R has RetractableTo(INT))
          then

            convert(pol:$):String ==
              ground?(pol) => convert(retract(ground(pol))@INT)@String
              ipol : $ := init(pol)
              vpol : V := mvar(pol)
              dpol : NNI := mdeg(pol)
              tpol: $  := tail(pol)
              sipol,svpol,sdpol,stpol : String
--              if one? ipol
              if (ipol = 1)
                then
                  sipol := empty()$String
                else
--                  if one?(-ipol)
                  if ((-ipol) = 1)
                    then
                      sipol := "-"
                    else
                      sipol := convert(ipol)@String
                      if not monomial?(ipol)
                        then
                          sipol := concat(["(",sipol,")*"])$String
                        else
                          sipol := concat(sipol,"*")$String
              svpol := string(convert(vpol)@Symbol)
--              if one? dpol
              if (dpol = 1)
                then
                  sdpol :=  empty()$String
                else
                  sdpol := _
                       concat("**",convert(convert(dpol)@INT)@String )$String
              if zero? tpol
                then
                  stpol :=  empty()$String
                else
                  if ground?(tpol)
                    then
                      n := retract(ground(tpol))@INT
                      if n > 0
                        then
```

```
                    stpol :=  concat(" +",convert(n)@String)$String
                  else
                    stpol := convert(n)@String
                else
                  stpol := convert(tpol)@String
                  if _
                    not member?((stpol.1)::String,["+","-"])$(List String)
                    then
                      stpol :=  concat(" + ",stpol)$String
            concat([sipol,svpol,sdpol,stpol])$String
```

———-

— RPOLCAT.dotabb —

```
"RPOLCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RPOLCAT"];
"RPOLCAT" -> "POLYCAT"
```

———-

— RPOLCAT.dotfull —

```
"RecursivePolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RPOLCAT"];
"RecursivePolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
 -> "PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
```

———-

— RPOLCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"RecursivePolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
 [color=lightblue];
"RecursivePolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
 -> "PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"

"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
 [color=lightblue];
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
```

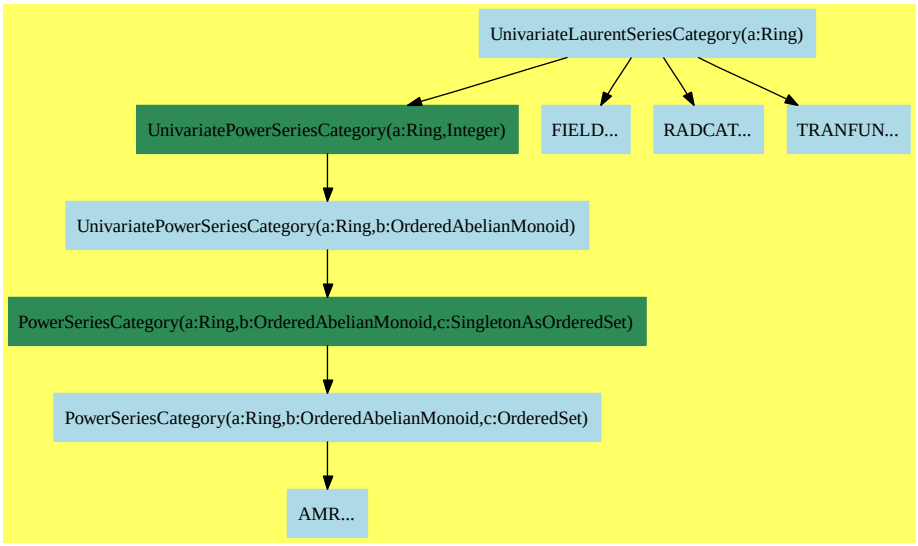
```
  -> "PDRING..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "FAMR..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "EVALAB..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "IEVALAB..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "RETRACT..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "FLINEXP..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "ORDSET..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "GCDDOM..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "PFECAT..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "KONVERT..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "PATMAB..."
"PolynomialCategory(a:Ring,b:OrderedAbelianMonoidSup,c:OrderedSet)"
  -> "COMRING..."

"PDRING..." [color=lightblue];
"FAMR..." [color=lightblue];
"EVALAB..." [color=lightblue];
"IEVALAB..." [color=lightblue];
"RETRACT..." [color=lightblue];
"FLINEXP..." [color=lightblue];
"ORDSET..." [color=lightblue];
"GCDDOM..." [color=lightblue];
"PFECAT..." [color=lightblue];
"KONVERT..." [color=lightblue];
"PATMAB..." [color=lightblue];
"COMRING..." [color=lightblue];

}
```

———————

17.0.202 UnivariateLaurentSeriesCategory (ULSCAT)



— **UnivariateLaurentSeriesCategory.input** —

```
)set break resume
)sys rm -f UnivariateLaurentSeriesCategory.output
)spool UnivariateLaurentSeriesCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show UnivariateLaurentSeriesCategory
--R UnivariateLaurentSeriesCategory Coef: Ring  is a category constructor
--R Abbreviation for UnivariateLaurentSeriesCategory is ULSCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for ULSCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (Coef,%) -> %                  ?*? : (%,Coef) -> %
--R ?*? : (%,%) -> %                     ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %       ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                     ?-? : (%,%) -> %
--R -? : % -> %                          ?=? : (%,%) -> Boolean
--R 1 : () -> %                          0 : () -> %
--R ?^? : (%,PositiveInteger) -> %       center : % -> Coef
--R coefficient : (%,Integer) -> Coef    coerce : Integer -> %
--R coerce : % -> OutputForm             complete : % -> %
```

```
--R degree : % -> Integer                ?.? : (%,Integer) -> Coef
--R extend : (%,Integer) -> %            hash : % -> SingleInteger
--R latex : % -> String                  leadingCoefficient : % -> Coef
--R leadingMonomial : % -> %             map : ((Coef -> Coef),%) -> %
--R monomial : (Coef,Integer) -> %       monomial? : % -> Boolean
--R one? : % -> Boolean                  order : (%,Integer) -> Integer
--R order : % -> Integer                 pole? : % -> Boolean
--R recip : % -> Union(%,"failed")       reductum : % -> %
--R sample : () -> %                     truncate : (%,Integer) -> %
--R variable : % -> Symbol               zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
--R ?*? : (Fraction Integer,%) -> % if Coef has ALGEBRA FRAC INT
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
--R ?**? : (%,%) -> % if Coef has ALGEBRA FRAC INT
--R ?**? : (%,Integer) -> % if Coef has FIELD
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (%,%) -> % if Coef has FIELD
--R ?/? : (%,Coef) -> % if Coef has FIELD
--R D : % -> % if Coef has *: (Integer,Coef) -> Coef
--R D : (%,NonNegativeInteger) -> % if Coef has *: (Integer,Coef) -> Coef
--R D : (%,Symbol) -> % if Coef has PDRING SYMBOL and Coef has *: (Integer,Coef) -> Coef
--R D : (%,List Symbol) -> % if Coef has PDRING SYMBOL and Coef has *: (Integer,Coef) -> Coef
--R D : (%,Symbol,NonNegativeInteger) -> % if Coef has PDRING SYMBOL and Coef has *: (Integer,Coef) -> Coef
--R D : (%,List Symbol,List NonNegativeInteger) -> % if Coef has PDRING SYMBOL and Coef has *: (Integer,Coe
--R ?^? : (%,Integer) -> % if Coef has FIELD
--R ?^? : (%,NonNegativeInteger) -> %
--R acos : % -> % if Coef has ALGEBRA FRAC INT
--R acosh : % -> % if Coef has ALGEBRA FRAC INT
--R acot : % -> % if Coef has ALGEBRA FRAC INT
--R acoth : % -> % if Coef has ALGEBRA FRAC INT
--R acsc : % -> % if Coef has ALGEBRA FRAC INT
--R acsch : % -> % if Coef has ALGEBRA FRAC INT
--R approximate : (%,Integer) -> Coef if Coef has **: (Coef,Integer) -> Coef and Coef has coerce: Symbol ->
--R asec : % -> % if Coef has ALGEBRA FRAC INT
--R asech : % -> % if Coef has ALGEBRA FRAC INT
--R asin : % -> % if Coef has ALGEBRA FRAC INT
--R asinh : % -> % if Coef has ALGEBRA FRAC INT
--R associates? : (%,%) -> Boolean if Coef has INTDOM
--R atan : % -> % if Coef has ALGEBRA FRAC INT
--R atanh : % -> % if Coef has ALGEBRA FRAC INT
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if Coef has CHARNZ
--R coerce : % -> % if Coef has INTDOM
--R coerce : Fraction Integer -> % if Coef has ALGEBRA FRAC INT
--R coerce : Coef -> % if Coef has COMRING
--R cos : % -> % if Coef has ALGEBRA FRAC INT
--R cosh : % -> % if Coef has ALGEBRA FRAC INT
--R cot : % -> % if Coef has ALGEBRA FRAC INT
```

```
--R coth : % -> % if Coef has ALGEBRA FRAC INT
--R csc : % -> % if Coef has ALGEBRA FRAC INT
--R csch : % -> % if Coef has ALGEBRA FRAC INT
--R differentiate : % -> % if Coef has *: (Integer,Coef) -> Coef
--R differentiate : (%,NonNegativeInteger) -> % if Coef has *: (Integer,Coef) -> Coef
--R differentiate : (%,Symbol) -> % if Coef has PDRING SYMBOL and Coef has *: (Integer,Coef) ->
--R differentiate : (%,List Symbol) -> % if Coef has PDRING SYMBOL and Coef has *: (Integer,Coef
--R differentiate : (%,Symbol,NonNegativeInteger) -> % if Coef has PDRING SYMBOL and Coef has *:
--R differentiate : (%,List Symbol,List NonNegativeInteger) -> % if Coef has PDRING SYMBOL and C
--R divide : (%,%) -> Record(quotient: %,remainder: %) if Coef has FIELD
--R ?.? : (%,%) -> % if Integer has SGROUP
--R euclideanSize : % -> NonNegativeInteger if Coef has FIELD
--R eval : (%,Coef) -> Stream Coef if Coef has **: (Coef,Integer) -> Coef
--R exp : % -> % if Coef has ALGEBRA FRAC INT
--R expressIdealMember : (List %,%) -> Union(List %,"failed") if Coef has FIELD
--R exquo : (%,%) -> Union(%,"failed") if Coef has INTDOM
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %) if Coef has FIELD
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed") if Coef has FIELD
--R factor : % -> Factored % if Coef has FIELD
--R gcd : (%,%) -> % if Coef has FIELD
--R gcd : List % -> % if Coef has FIELD
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivari
--R integrate : (%,Symbol) -> % if Coef has ACFS INT and Coef has PRIMCAT and Coef has TRANFUN a
--R integrate : % -> % if Coef has ALGEBRA FRAC INT
--R inv : % -> % if Coef has FIELD
--R lcm : (%,%) -> % if Coef has FIELD
--R lcm : List % -> % if Coef has FIELD
--R log : % -> % if Coef has ALGEBRA FRAC INT
--R monomial : (%,List SingletonAsOrderedSet,List Integer) -> %
--R monomial : (%,SingletonAsOrderedSet,Integer) -> %
--R multiEuclidean : (List %,%) -> Union(List %,"failed") if Coef has FIELD
--R multiplyCoefficients : ((Integer -> Coef),%) -> %
--R multiplyExponents : (%,PositiveInteger) -> %
--R nthRoot : (%,Integer) -> % if Coef has ALGEBRA FRAC INT
--R pi : () -> % if Coef has ALGEBRA FRAC INT
--R prime? : % -> Boolean if Coef has FIELD
--R principalIdeal : List % -> Record(coef: List %,generator: %) if Coef has FIELD
--R ?quo? : (%,%) -> % if Coef has FIELD
--R rationalFunction : (%,Integer,Integer) -> Fraction Polynomial Coef if Coef has INTDOM
--R rationalFunction : (%,Integer) -> Fraction Polynomial Coef if Coef has INTDOM
--R ?rem? : (%,%) -> % if Coef has FIELD
--R sec : % -> % if Coef has ALGEBRA FRAC INT
--R sech : % -> % if Coef has ALGEBRA FRAC INT
--R series : Stream Record(k: Integer,c: Coef) -> %
--R sin : % -> % if Coef has ALGEBRA FRAC INT
--R sinh : % -> % if Coef has ALGEBRA FRAC INT
--R sizeLess? : (%,%) -> Boolean if Coef has FIELD
--R sqrt : % -> % if Coef has ALGEBRA FRAC INT
--R squareFree : % -> Factored % if Coef has FIELD
--R squareFreePart : % -> % if Coef has FIELD
```

```
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R tan : % -> % if Coef has ALGEBRA FRAC INT
--R tanh : % -> % if Coef has ALGEBRA FRAC INT
--R terms : % -> Stream Record(k: Integer,c: Coef)
--R truncate : (%,Integer,Integer) -> %
--R unit? : % -> Boolean if Coef has INTDOM
--R unitCanonical : % -> % if Coef has INTDOM
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %) if Coef has INTDOM
--R variables : % -> List SingletonAsOrderedSet
--R
--E 1

)spool
)lisp (bye)
```

———-

— **UnivariateLaurentSeriesCategory.help** —

```
====================================================================
UnivariateLaurentSeriesCategory examples
====================================================================

See Also:
o )show UnivariateLaurentSeriesCategory
```

———-

See:

⇒ “UnivariateLaurentSeriesConstructorCategory” (ULSCCAT) 18.0.211 on page 1553
⇐ “Field” (FIELD) 16.0.185 on page 1221
⇐ “RadicalCategory” (RADCAT) 2.0.17 on page 52
⇐ “TranscendentalFunctionCategory” (TRANFUN) 3.0.35 on page 117
⇐ “UnivariatePowerSeriesCategory” (UPSCAT) 15.0.184 on page 1210

Exports:

0	1	acos	acosh
acot	acoth	acsc	acsch
approximate	asec	asech	asin
asinh	associates?	atan	atanh
center	characteristic	charthRoot	coefficient
coerce	complete	cos	cosh
cot	coth	csc	csch
D	degree	differentiate	divide
euclideanSize	eval	exp	expressIdealMember
exquo	extend	extendedEuclidean	factor
gcd	gcdPolynomial	hash	integrate
inv	latex	lcm	leadingCoefficient
leadingMonomial	log	map	monomial
monomial?	multiEuclidean	multiplyCoefficients	multiplyExponents
nthRoot	one?	order	pi
pole?	prime?	principalIdeal	rationalFunction
recip	reductum	sample	sec
sech	series	sin	sinh
sizeLess?	sqrt	squareFree	squareFreePart
subtractIfCan	tan	tanh	terms
truncate	unit?	unitCanonical	unitNormal
variable	variables	zero?	?*?
?**?	?+?	?-? * -?	
?=?	?^?	?~=?	?/?
?.?	?quo?	?rem?	

Attributes Exported:

- if #1 has Field then canonicalUnitNormal where **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- if #1 has Field then canonicalsClosed where **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.
- if #1 has IntegralDomain then noZeroDivisors where **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- if #1 has CommutativeRing then commutative("*") where **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.

These are directly exported but not implemented:

```
integrate : % -> % if Coef has ALGEBRA FRAC INT
integrate : (%,Symbol) -> %
  if Coef has ACFS INT
  and Coef has PRIMCAT
  and Coef has TRANFUN
  and Coef has ALGEBRA FRAC INT
  or Coef has variables: Coef -> List Symbol
  and Coef has integrate: (Coef,Symbol) -> Coef
  and Coef has ALGEBRA FRAC INT
multiplyCoefficients : ((Integer -> Coef),%) -> %
rationalFunction : (%,Integer) -> Fraction Polynomial Coef
  if Coef has INTDOM
rationalFunction : (%,Integer,Integer) -> Fraction Polynomial Coef
  if Coef has INTDOM
series : Stream Record(k: Integer,c: Coef) -> %
```

These exports come from (p1210) UnivariatePowerSeriesCategory(Coef,Integer) where Coef:Ring:

```
0 : () -> %
1 : () -> %
approximate : (%,Integer) -> Coef
  if Coef has **: (Coef,Integer) -> Coef
  and Coef has coerce: Symbol -> Coef
associates? : (%,%) -> Boolean if Coef has INTDOM
center : % -> Coef
characteristic : () -> NonNegativeInteger
charthRoot : % -> Union(%,"failed") if Coef has CHARNZ
coefficient : (%,Integer) -> Coef
coerce : % -> % if Coef has INTDOM
coerce : Fraction Integer -> % if Coef has ALGEBRA FRAC INT
coerce : Coef -> % if Coef has COMRING
coerce : Integer -> %
coerce : % -> OutputForm
complete : % -> %
D : % -> % if Coef has *: (Integer,Coef) -> Coef
D : (%,NonNegativeInteger) -> %
  if Coef has *: (Integer,Coef) -> Coef
D : (%,Symbol) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Integer,Coef) -> Coef
D : (%,List Symbol) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Integer,Coef) -> Coef
D : (%,Symbol,NonNegativeInteger) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Integer,Coef) -> Coef
D : (%,List Symbol,List NonNegativeInteger) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Integer,Coef) -> Coef
```

```
degree : % -> Integer
differentiate : (%,Symbol) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Integer,Coef) -> Coef
differentiate : (%,List Symbol) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Integer,Coef) -> Coef
differentiate : (%,Symbol,NonNegativeInteger) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Integer,Coef) -> Coef
differentiate : (%,List Symbol,List NonNegativeInteger) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Integer,Coef) -> Coef
differentiate : % -> %
  if Coef has *: (Integer,Coef) -> Coef
differentiate : (%,NonNegativeInteger) -> %
  if Coef has *: (Integer,Coef) -> Coef
eval : (%,Coef) -> Stream Coef
  if Coef has **: (Coef,Integer) -> Coef
exquo : (%,%) -> Union(%,"failed")
  if Coef has INTDOM
extend : (%,Integer) -> %
hash : % -> SingleInteger
latex : % -> String
leadingCoefficient : % -> Coef
leadingMonomial : % -> %
map : ((Coef -> Coef),%) -> %
monomial : (%,SingletonAsOrderedSet,Integer) -> %
monomial : (%,List SingletonAsOrderedSet,List Integer) -> %
monomial : (Coef,Integer) -> %
monomial? : % -> Boolean
multiplyExponents : (%,PositiveInteger) -> %
one? : % -> Boolean
order : (%,Integer) -> Integer
order : % -> Integer
pole? : % -> Boolean
recip : % -> Union(%,"failed")
reductum : % -> %
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
terms : % -> Stream Record(k: Integer,c: Coef)
truncate : (%,Integer,Integer) -> %
truncate : (%,Integer) -> %
unit? : % -> Boolean if Coef has INTDOM
unitCanonical : % -> % if Coef has INTDOM
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
  if Coef has INTDOM
variable : % -> Symbol
variables : % -> List SingletonAsOrderedSet
zero? : % -> Boolean
```

```
?.? : (%,Integer) -> Coef
?**? : (%,NonNegativeInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (%,%) -> %
?-? : (%,%) -> %
?**? : (%,PositiveInteger) -> %
?^? : (%,PositiveInteger) -> %
?*? : (Integer,%) -> %
?*? : (Coef,%) -> %
?*? : (%,Coef) -> %
?*? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
?*? : (Fraction Integer,%) -> % if Coef has ALGEBRA FRAC INT
?/? : (%,Coef) -> % if Coef has FIELD
-? : % -> %
?.? : (%,%) -> % if Integer has SGROUP
```

These exports come from (p117) TranscendentalFunctionCategory():

```
acos : % -> % if Coef has ALGEBRA FRAC INT
acosh : % -> % if Coef has ALGEBRA FRAC INT
acot : % -> % if Coef has ALGEBRA FRAC INT
acoth : % -> % if Coef has ALGEBRA FRAC INT
acsc : % -> % if Coef has ALGEBRA FRAC INT
acsch : % -> % if Coef has ALGEBRA FRAC INT
asec : % -> % if Coef has ALGEBRA FRAC INT
asech : % -> % if Coef has ALGEBRA FRAC INT
asin : % -> % if Coef has ALGEBRA FRAC INT
asinh : % -> % if Coef has ALGEBRA FRAC INT
atanh : % -> % if Coef has ALGEBRA FRAC INT
atan : % -> % if Coef has ALGEBRA FRAC INT
cos : % -> % if Coef has ALGEBRA FRAC INT
cosh : % -> % if Coef has ALGEBRA FRAC INT
cot : % -> % if Coef has ALGEBRA FRAC INT
coth : % -> % if Coef has ALGEBRA FRAC INT
csc : % -> % if Coef has ALGEBRA FRAC INT
csch : % -> % if Coef has ALGEBRA FRAC INT
exp : % -> % if Coef has ALGEBRA FRAC INT
log : % -> % if Coef has ALGEBRA FRAC INT
pi : () -> % if Coef has ALGEBRA FRAC INT
sec : % -> % if Coef has ALGEBRA FRAC INT
sech : % -> % if Coef has ALGEBRA FRAC INT
sin : % -> % if Coef has ALGEBRA FRAC INT
sinh : % -> % if Coef has ALGEBRA FRAC INT
tan : % -> % if Coef has ALGEBRA FRAC INT
tanh : % -> % if Coef has ALGEBRA FRAC INT
```

```
?**? : (%,%) -> % if Coef has ALGEBRA FRAC INT
```

These exports come from (p52) RadicalCategory():

```
nthRoot : (%,Integer) -> % if Coef has ALGEBRA FRAC INT
sqrt : % -> % if Coef has ALGEBRA FRAC INT
?**? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
```

These exports come from (p1221) Field():

```
divide : (%,%) -> Record(quotient: %,remainder: %)
  if Coef has FIELD
euclideanSize : % -> NonNegativeInteger
  if Coef has FIELD
expressIdealMember : (List %,%) -> Union(List %,"failed")
  if Coef has FIELD
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
  if Coef has FIELD
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
  if Coef has FIELD
factor : % -> Factored % if Coef has FIELD
gcd : (%,%) -> % if Coef has FIELD
gcd : List % -> % if Coef has FIELD
gcdPolynomial :
  (SparseUnivariatePolynomial %,
   SparseUnivariatePolynomial %) ->
     SparseUnivariatePolynomial %
       if Coef has FIELD
inv : % -> % if Coef has FIELD
lcm : (%,%) -> % if Coef has FIELD
lcm : List % -> % if Coef has FIELD
multiEuclidean : (List %,%) -> Union(List %,"failed")
  if Coef has FIELD
prime? : % -> Boolean if Coef has FIELD
principalIdeal : List % -> Record(coef: List %,generator: %)
  if Coef has FIELD
sizeLess? : (%,%) -> Boolean if Coef has FIELD
squareFree : % -> Factored % if Coef has FIELD
squareFreePart : % -> % if Coef has FIELD
?**? : (%,Integer) -> % if Coef has FIELD
?^? : (%,Integer) -> % if Coef has FIELD
?/? : (%,%) -> % if Coef has FIELD
?quo? : (%,%) -> % if Coef has FIELD
?rem? : (%,%) -> % if Coef has FIELD
```

— category ULSCAT UnivariateLaurentSeriesCategory —

```
)abbrev category ULSCAT UnivariateLaurentSeriesCategory
```

```
++ Author: Clifton J. Williamson
++ Date Created: 21 December 1989
++ Date Last Updated: 20 September 1993
++ Basic Operations:
++ Related Domains:
++ Also See:
++ AMS Classifications:
++ Keywords: series, Laurent
++ Examples:
++ References:
++ Description:
++ \spadtype{UnivariateLaurentSeriesCategory} is the category of
++ Laurent series in one variable.

UnivariateLaurentSeriesCategory(Coef): Category == Definition where
  Coef  : Ring
  I    ==> Integer
  NNI  ==> NonNegativeInteger
  Term ==> Record(k:I,c:Coef)

  Definition ==> UnivariatePowerSeriesCategory(Coef,Integer) with

    series: Stream Term -> %
      ++ \spad{series(st)} creates a series from a stream of non-zero terms,
      ++ where a term is an exponent-coefficient pair.  The terms in the
      ++ stream should be ordered by increasing order of exponents.
    multiplyCoefficients: (I -> Coef,%) -> %
      ++ \spad{multiplyCoefficients(f,sum(n = n0..infinity,a[n] * x**n)) =
      ++ sum(n = 0..infinity,f(n) * a[n] * x**n)}.
      ++ This function is used when Puiseux series are represented by
      ++ a Laurent series and an exponent.
    if Coef has IntegralDomain then
      rationalFunction: (%,I) -> Fraction Polynomial Coef
        ++ \spad{rationalFunction(f,k)} returns a rational function
        ++ consisting of the sum of all terms of f of degree <= k.
      rationalFunction: (%,I,I) -> Fraction Polynomial Coef
        ++ \spad{rationalFunction(f,k1,k2)} returns a rational function
        ++ consisting of the sum of all terms of f of degree d with
        ++ \spad{k1 <= d <= k2}.

    if Coef has Algebra Fraction Integer then
      integrate: % -> %
        ++ \spad{integrate(f(x))} returns an anti-derivative of the power
        ++ series \spad{f(x)} with constant coefficient 1.
        ++ We may integrate a series when we can divide coefficients
        ++ by integers.
      if Coef has integrate: (Coef,Symbol) -> Coef and _
         Coef has variables: Coef -> List Symbol then
        integrate: (%,Symbol) -> %
          ++ \spad{integrate(f(x),y)} returns an anti-derivative of the power
```

```
          ++ series \spad{f(x)} with respect to the variable \spad{y}.
      if Coef has TranscendentalFunctionCategory and _
         Coef has PrimitiveFunctionCategory and _
         Coef has AlgebraicallyClosedFunctionSpace Integer then
        integrate: (%,Symbol) -> %
          ++ \spad{integrate(f(x),y)} returns an anti-derivative of
          ++ the power series \spad{f(x)} with respect to the variable
          ++ \spad{y}.
      RadicalCategory
        --++ We provide rational powers when we can divide coefficients
        --++ by integers.
      TranscendentalFunctionCategory
        --++ We provide transcendental functions when we can divide
        --++ coefficients by integers.
    if Coef has Field then Field
        --++ Univariate Laurent series over a field form a field.
        --++ In fact, K((x)) is the quotient field of K[[x]].
```

———-

— ULSCAT.dotabb —

```
"ULSCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ULSCAT"];
"ULSCAT" -> "UPSCAT"
"ULSCAT" -> "FIELD"
"ULSCAT" -> "TRANFUN"
"ULSCAT" -> "RADCAT"
```

———-

— ULSCAT.dotfull —

```
"UnivariateLaurentSeriesCategory(a:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ULSCAT"];
"UnivariateLaurentSeriesCategory(a:Ring)" ->
    "Field()"
"UnivariateLaurentSeriesCategory(a:Ring)" ->
    "RadicalCategory()"
"UnivariateLaurentSeriesCategory(a:Ring)" ->
    "TranscendentalFunctionCategory()"
"UnivariateLaurentSeriesCategory(a:Ring)" ->
    "UnivariatePowerSeriesCategory(a:Ring,Integer)"
```

———-

— ULSCAT.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"UnivariateLaurentSeriesCategory(a:Ring)" [color=lightblue];
"UnivariateLaurentSeriesCategory(a:Ring)" ->
    "UnivariatePowerSeriesCategory(a:Ring,Integer)"
"UnivariateLaurentSeriesCategory(a:Ring)" ->
    "FIELD..."
"UnivariateLaurentSeriesCategory(a:Ring)" ->
    "RADCAT..."
"UnivariateLaurentSeriesCategory(a:Ring)" ->
    "TRANFUN..."

"UnivariatePowerSeriesCategory(a:Ring,Integer)" [color=seagreen];
"UnivariatePowerSeriesCategory(a:Ring,Integer)" ->
    "UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)"

"UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)"
 [color=lightblue];
"UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)" ->
 "PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:SingletonAsOrderedSet)"

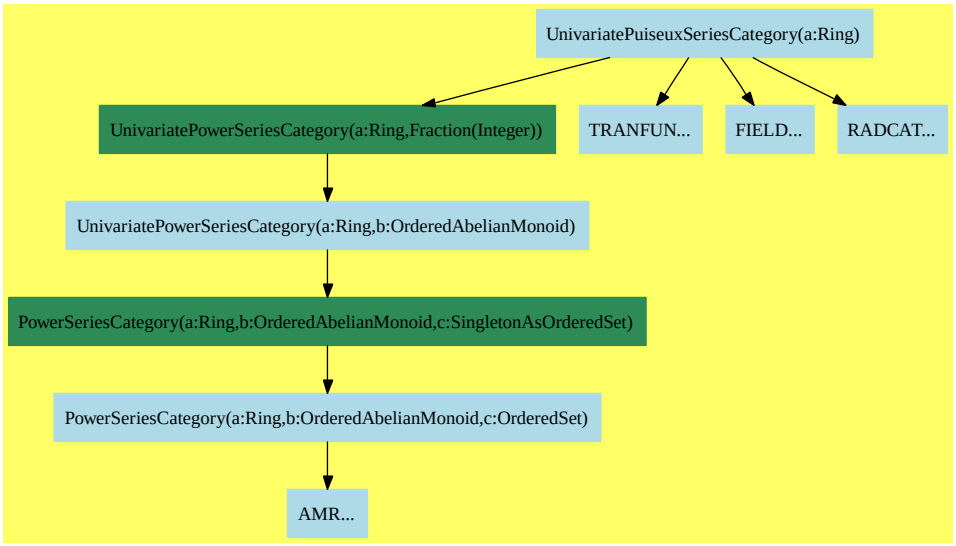
"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:SingletonAsOrderedSet)"
 [color=seagreen];
"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:SingletonAsOrderedSet)"
  -> "PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)"

"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)"
 [color=lightblue];
"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)" ->
    "AMR..."

"AMR..." [color=lightblue];
"FIELD..." [color=lightblue];
"TRANFUN..." [color=lightblue];
"RADCAT..." [color=lightblue];
}
```

———-

17.0.203 UnivariatePuiseuxSeriesCategory (UPXSCAT)



— UnivariatePuiseuxSeriesCategory.input —

```
)set break resume
)sys rm -f UnivariatePuiseuxSeriesCategory.output
)spool UnivariatePuiseuxSeriesCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show UnivariatePuiseuxSeriesCategory
--R UnivariatePuiseuxSeriesCategory Coef: Ring  is a category constructor
--R Abbreviation for UnivariatePuiseuxSeriesCategory is UPXSCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for UPXSCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (Coef,%) -> %                  ?*? : (%,Coef) -> %
--R ?*? : (%,%) -> %                     ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %       ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                     ?-? : (%,%) -> %
--R -? : % -> %                          ?=? : (%,%) -> Boolean
--R 1 : () -> %                          0 : () -> %
--R ?^? : (%,PositiveInteger) -> %       center : % -> Coef
--R coerce : Integer -> %                coerce : % -> OutputForm
--R complete : % -> %                    degree : % -> Fraction Integer
```

```
--R hash : % -> SingleInteger             latex : % -> String
--R leadingCoefficient : % -> Coef        leadingMonomial : % -> %
--R map : ((Coef -> Coef),%) -> %         monomial? : % -> Boolean
--R one? : % -> Boolean                   order : % -> Fraction Integer
--R pole? : % -> Boolean                  recip : % -> Union(%,"failed")
--R reductum : % -> %                     sample : () -> %
--R variable : % -> Symbol                zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
--R ?*? : (Fraction Integer,%) -> % if Coef has ALGEBRA FRAC INT
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
--R ?**? : (%,%) -> % if Coef has ALGEBRA FRAC INT
--R ?**? : (%,Integer) -> % if Coef has FIELD
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (%,%) -> % if Coef has FIELD
--R ?/? : (%,Coef) -> % if Coef has FIELD
--R D : % -> % if Coef has *: (Fraction Integer,Coef) -> Coef
--R D : (%,NonNegativeInteger) -> % if Coef has *: (Fraction Integer,Coef) -> Coef
--R D : (%,Symbol) -> % if Coef has PDRING SYMBOL and Coef has *: (Fraction Integer,Coef) -> Coef
--R D : (%,List Symbol) -> % if Coef has PDRING SYMBOL and Coef has *: (Fraction Integer,Coef) -> Coef
--R D : (%,Symbol,NonNegativeInteger) -> % if Coef has PDRING SYMBOL and Coef has *: (Fraction Integer,Coef)
--R D : (%,List Symbol,List NonNegativeInteger) -> % if Coef has PDRING SYMBOL and Coef has *: (Fraction Int
--R ?^? : (%,Integer) -> % if Coef has FIELD
--R ?^? : (%,NonNegativeInteger) -> %
--R acos : % -> % if Coef has ALGEBRA FRAC INT
--R acosh : % -> % if Coef has ALGEBRA FRAC INT
--R acot : % -> % if Coef has ALGEBRA FRAC INT
--R acoth : % -> % if Coef has ALGEBRA FRAC INT
--R acsc : % -> % if Coef has ALGEBRA FRAC INT
--R acsch : % -> % if Coef has ALGEBRA FRAC INT
--R approximate : (%,Fraction Integer) -> Coef if Coef has **: (Coef,Fraction Integer) -> Coef and Coef has
--R asec : % -> % if Coef has ALGEBRA FRAC INT
--R asech : % -> % if Coef has ALGEBRA FRAC INT
--R asin : % -> % if Coef has ALGEBRA FRAC INT
--R asinh : % -> % if Coef has ALGEBRA FRAC INT
--R associates? : (%,%) -> Boolean if Coef has INTDOM
--R atan : % -> % if Coef has ALGEBRA FRAC INT
--R atanh : % -> % if Coef has ALGEBRA FRAC INT
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if Coef has CHARNZ
--R coefficient : (%,Fraction Integer) -> Coef
--R coerce : % -> % if Coef has INTDOM
--R coerce : Fraction Integer -> % if Coef has ALGEBRA FRAC INT
--R coerce : Coef -> % if Coef has COMRING
--R cos : % -> % if Coef has ALGEBRA FRAC INT
--R cosh : % -> % if Coef has ALGEBRA FRAC INT
--R cot : % -> % if Coef has ALGEBRA FRAC INT
--R coth : % -> % if Coef has ALGEBRA FRAC INT
--R csc : % -> % if Coef has ALGEBRA FRAC INT
```

```
--R csch : % -> % if Coef has ALGEBRA FRAC INT
--R differentiate : % -> % if Coef has *: (Fraction Integer,Coef) -> Coef
--R differentiate : (%,NonNegativeInteger) -> % if Coef has *: (Fraction Integer,Coef) -> Coef
--R differentiate : (%,Symbol) -> % if Coef has PDRING SYMBOL and Coef has *: (Fraction Integer,
--R differentiate : (%,List Symbol) -> % if Coef has PDRING SYMBOL and Coef has *: (Fraction Int
--R differentiate : (%,Symbol,NonNegativeInteger) -> % if Coef has PDRING SYMBOL and Coef has *:
--R differentiate : (%,List Symbol,List NonNegativeInteger) -> % if Coef has PDRING SYMBOL and C
--R divide : (%,%) -> Record(quotient: %,remainder: %) if Coef has FIELD
--R ?.? : (%,%) -> % if Fraction Integer has SGROUP
--R ?.? : (%,Fraction Integer) -> Coef
--R euclideanSize : % -> NonNegativeInteger if Coef has FIELD
--R eval : (%,Coef) -> Stream Coef if Coef has **: (Coef,Fraction Integer) -> Coef
--R exp : % -> % if Coef has ALGEBRA FRAC INT
--R expressIdealMember : (List %,%) -> Union(List %,"failed") if Coef has FIELD
--R exquo : (%,%) -> Union(%,"failed") if Coef has INTDOM
--R extend : (%,Fraction Integer) -> %
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %) if Coef has FIELD
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed") if Coef has FIELD
--R factor : % -> Factored % if Coef has FIELD
--R gcd : (%,%) -> % if Coef has FIELD
--R gcd : List % -> % if Coef has FIELD
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivari
--R integrate : (%,Symbol) -> % if Coef has ACFS INT and Coef has PRIMCAT and Coef has TRANFUN a
--R integrate : % -> % if Coef has ALGEBRA FRAC INT
--R inv : % -> % if Coef has FIELD
--R lcm : (%,%) -> % if Coef has FIELD
--R lcm : List % -> % if Coef has FIELD
--R log : % -> % if Coef has ALGEBRA FRAC INT
--R monomial : (%,List SingletonAsOrderedSet,List Fraction Integer) -> %
--R monomial : (%,SingletonAsOrderedSet,Fraction Integer) -> %
--R monomial : (Coef,Fraction Integer) -> %
--R multiEuclidean : (List %,%) -> Union(List %,"failed") if Coef has FIELD
--R multiplyExponents : (%,Fraction Integer) -> %
--R multiplyExponents : (%,PositiveInteger) -> %
--R nthRoot : (%,Integer) -> % if Coef has ALGEBRA FRAC INT
--R order : (%,Fraction Integer) -> Fraction Integer
--R pi : () -> % if Coef has ALGEBRA FRAC INT
--R prime? : % -> Boolean if Coef has FIELD
--R principalIdeal : List % -> Record(coef: List %,generator: %) if Coef has FIELD
--R ?quo? : (%,%) -> % if Coef has FIELD
--R ?rem? : (%,%) -> % if Coef has FIELD
--R sec : % -> % if Coef has ALGEBRA FRAC INT
--R sech : % -> % if Coef has ALGEBRA FRAC INT
--R series : (NonNegativeInteger,Stream Record(k: Fraction Integer,c: Coef)) -> %
--R sin : % -> % if Coef has ALGEBRA FRAC INT
--R sinh : % -> % if Coef has ALGEBRA FRAC INT
--R sizeLess? : (%,%) -> Boolean if Coef has FIELD
--R sqrt : % -> % if Coef has ALGEBRA FRAC INT
--R squareFree : % -> Factored % if Coef has FIELD
--R squareFreePart : % -> % if Coef has FIELD
```

```
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R tan : % -> % if Coef has ALGEBRA FRAC INT
--R tanh : % -> % if Coef has ALGEBRA FRAC INT
--R terms : % -> Stream Record(k: Fraction Integer,c: Coef)
--R truncate : (%,Fraction Integer,Fraction Integer) -> %
--R truncate : (%,Fraction Integer) -> %
--R unit? : % -> Boolean if Coef has INTDOM
--R unitCanonical : % -> % if Coef has INTDOM
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %) if Coef has INTDOM
--R variables : % -> List SingletonAsOrderedSet
--R
--E 1

)spool
)lisp (bye)
```

———-

— **UnivariatePuiseuxSeriesCategory.help** —

```
====================================================================
UnivariatePuiseuxSeriesCategory examples
====================================================================

See Also:
o )show UnivariatePuiseuxSeriesCategory
```

———-

See:

⇒ “UnivariatePuiseuxSeriesConstructorCategory” (UPXSCCA) 18.0.212 on page 1570
⇐ “TranscendentalFunctionCategory” (TRANFUN) 3.0.35 on page 117
⇐ “Field” (FIELD) 16.0.185 on page 1221
⇐ “RadicalCategory” (RADCAT) 2.0.17 on page 52
⇐ “UnivariatePowerSeriesCategory” (UPSCAT) 15.0.184 on page 1210

Exports:

0	1	acos	acosh
acot	acoth	acsc	acsch
approximate	asec	asech	asin
asinh	associates?	atan	atanh
center	characteristic	charthRoot	coefficient
coerce	complete	cos	cosh
cot	coth	csc	csch
D	degree	differentiate	divide
euclideanSize	eval	exp	expressIdealMember
exquo	extend	extendedEuclidean	factor
gcd	gcdPolynomial	hash	integrate
inv	latex	lcm	leadingCoefficient
leadingMonomial	log	map	monomial
monomial?	multiEuclidean	multiplyExponents	nthRoot
one?	order	pi	pole?
prime?	principalIdeal	recip	reductum
sample	sec	sech	series
sin	sinh	sizeLess?	sqrt
squareFree	squareFreePart	subtractIfCan	tan
tanh	terms	truncate	unit?
unitCanonical	unitNormal	variable	variables
zero?	?*?	?**?	?+?
?-?	-?	?=?	?~=?
?/?	?^?	?.?	?quo?
?rem?			

Attributes Exported:

- if #1 has Field then canonicalUnitNormal where **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- if #1 has Field then canonicalsClosed where **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.
- if #1 has IntegralDomain then noZeroDivisors where **noZeroDivisors** is true if $x*y \neq 0$ implies both x and y are non-zero.
- if #1 has CommutativeRing then commutative("*") where **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.

These are directly exported but not implemented:

```
integrate : % -> % if Coef has ALGEBRA FRAC INT
integrate : (%,Symbol) -> %
  if Coef has ACFS INT
  and Coef has PRIMCAT
  and Coef has TRANFUN
  and Coef has ALGEBRA FRAC INT
  or Coef has variables: Coef -> List Symbol
  and Coef has integrate: (Coef,Symbol) -> Coef
  and Coef has ALGEBRA FRAC INT
multiplyExponents : (%,Fraction Integer) -> %
series : (NonNegativeInteger,Stream Record(k: Fraction Integer,c: Coef)) -> %
```

These exports come from (p1210) UnivariatePowerSeriesCategory(Coef,RN)
where Coef:Ring and RN:Fraction(Integer):

```
0 : () -> %
1 : () -> %
approximate : (%,Fraction Integer) -> Coef
  if Coef has **: (Coef,Fraction Integer) -> Coef
  and Coef has coerce: Symbol -> Coef
associates? : (%,%) -> Boolean if Coef has INTDOM
center : % -> Coef
characteristic : () -> NonNegativeInteger
charthRoot : % -> Union(%,"failed") if Coef has CHARNZ
coefficient : (%,Fraction Integer) -> Coef
coerce : % -> % if Coef has INTDOM
coerce : Fraction Integer -> % if Coef has ALGEBRA FRAC INT
coerce : Coef -> % if Coef has COMRING
coerce : Integer -> %
coerce : % -> OutputForm
complete : % -> %
D : % -> % if Coef has *: (Fraction Integer,Coef) -> Coef
D : (%,NonNegativeInteger) -> %
  if Coef has *: (Fraction Integer,Coef) -> Coef
D : (%,Symbol) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Fraction Integer,Coef) -> Coef
D : (%,List Symbol) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Fraction Integer,Coef) -> Coef
D : (%,Symbol,NonNegativeInteger) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Fraction Integer,Coef) -> Coef
D : (%,List Symbol,List NonNegativeInteger) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Fraction Integer,Coef) -> Coef
degree : % -> Fraction Integer
differentiate : (%,Symbol) -> %
```

```
  if Coef has PDRING SYMBOL
  and Coef has *: (Fraction Integer,Coef) -> Coef
differentiate : (%,List Symbol) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Fraction Integer,Coef) -> Coef
differentiate : (%,Symbol,NonNegativeInteger) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Fraction Integer,Coef) -> Coef
differentiate : (%,List Symbol,List NonNegativeInteger) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Fraction Integer,Coef) -> Coef
differentiate : % -> %
  if Coef has *: (Fraction Integer,Coef) -> Coef
differentiate : (%,NonNegativeInteger) -> %
  if Coef has *: (Fraction Integer,Coef) -> Coef
eval : (%,Coef) -> Stream Coef
  if Coef has **: (Coef,Fraction Integer) -> Coef
exquo : (%,%) -> Union(%,"failed") if Coef has INTDOM
extend : (%,Fraction Integer) -> %
hash : % -> SingleInteger
latex : % -> String
leadingCoefficient : % -> Coef
leadingMonomial : % -> %
map : ((Coef -> Coef),%) -> %
monomial : (%,List SingletonAsOrderedSet,List Fraction Integer) -> %
monomial : (Coef,Fraction Integer) -> %
monomial : (%,SingletonAsOrderedSet,Fraction Integer) -> %
monomial? : % -> Boolean
multiplyExponents : (%,PositiveInteger) -> %
one? : % -> Boolean
order : (%,Fraction Integer) -> Fraction Integer
order : % -> Fraction Integer
pole? : % -> Boolean
recip : % -> Union(%,"failed")
reductum : % -> %
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
terms : % -> Stream Record(k: Fraction Integer,c: Coef)
truncate : (%,Fraction Integer,Fraction Integer) -> %
truncate : (%,Fraction Integer) -> %
unit? : % -> Boolean if Coef has INTDOM
unitCanonical : % -> % if Coef has INTDOM
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
  if Coef has INTDOM
variable : % -> Symbol
variables : % -> List SingletonAsOrderedSet
zero? : % -> Boolean
?**? : (%,NonNegativeInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?+? : (%,%) -> %
```

```
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (%,%) -> %
?-? : (%,%) -> %
?**? : (%,PositiveInteger) -> %
?^? : (%,PositiveInteger) -> %
?*? : (Integer,%) -> %
?*? : (Coef,%) -> %
?*? : (%,Coef) -> %
?*? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
?*? : (Fraction Integer,%) -> % if Coef has ALGEBRA FRAC INT
?/? : (%,Coef) -> % if Coef has FIELD
-? : % -> %
?.? : (%,%) -> % if Fraction Integer has SGROUP
?.? : (%,Fraction Integer) -> Coef
```

These exports come from (p117) TranscendentalFunctionCategory():

```
acos : % -> % if Coef has ALGEBRA FRAC INT
acosh : % -> % if Coef has ALGEBRA FRAC INT
acot : % -> % if Coef has ALGEBRA FRAC INT
acoth : % -> % if Coef has ALGEBRA FRAC INT
acsc : % -> % if Coef has ALGEBRA FRAC INT
acsch : % -> % if Coef has ALGEBRA FRAC INT
asec : % -> % if Coef has ALGEBRA FRAC INT
asech : % -> % if Coef has ALGEBRA FRAC INT
asin : % -> % if Coef has ALGEBRA FRAC INT
asinh : % -> % if Coef has ALGEBRA FRAC INT
atan : % -> % if Coef has ALGEBRA FRAC INT
atanh : % -> % if Coef has ALGEBRA FRAC INT
cos : % -> % if Coef has ALGEBRA FRAC INT
cosh : % -> % if Coef has ALGEBRA FRAC INT
cot : % -> % if Coef has ALGEBRA FRAC INT
coth : % -> % if Coef has ALGEBRA FRAC INT
csc : % -> % if Coef has ALGEBRA FRAC INT
csch : % -> % if Coef has ALGEBRA FRAC INT
exp : % -> % if Coef has ALGEBRA FRAC INT
log : % -> % if Coef has ALGEBRA FRAC INT
pi : () -> % if Coef has ALGEBRA FRAC INT
sec : % -> % if Coef has ALGEBRA FRAC INT
sech : % -> % if Coef has ALGEBRA FRAC INT
sin : % -> % if Coef has ALGEBRA FRAC INT
sinh : % -> % if Coef has ALGEBRA FRAC INT
tan : % -> % if Coef has ALGEBRA FRAC INT
tanh : % -> % if Coef has ALGEBRA FRAC INT
?**? : (%,%) -> % if Coef has ALGEBRA FRAC INT
```

These exports come from (p1221) Field():

```
divide : (%,%) -> Record(quotient: %,remainder: %)
  if Coef has FIELD
euclideanSize : % -> NonNegativeInteger if Coef has FIELD
expressIdealMember : (List %,%) -> Union(List %,"failed")
  if Coef has FIELD
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
  if Coef has FIELD
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
  if Coef has FIELD
factor : % -> Factored % if Coef has FIELD
gcd : (%,%) -> % if Coef has FIELD
gcd : List % -> % if Coef has FIELD
gcdPolynomial :
  (SparseUnivariatePolynomial %,
   SparseUnivariatePolynomial %) ->
     SparseUnivariatePolynomial %
       if Coef has FIELD
inv : % -> % if Coef has FIELD
lcm : (%,%) -> % if Coef has FIELD
lcm : List % -> % if Coef has FIELD
multiEuclidean : (List %,%) -> Union(List %,"failed")
  if Coef has FIELD
prime? : % -> Boolean if Coef has FIELD
principalIdeal : List % -> Record(coef: List %,generator: %)
  if Coef has FIELD
sizeLess? : (%,%) -> Boolean if Coef has FIELD
squareFree : % -> Factored % if Coef has FIELD
squareFreePart : % -> % if Coef has FIELD
?**? : (%,Integer) -> % if Coef has FIELD
?^? : (%,Integer) -> % if Coef has FIELD
?/? : (%,%) -> % if Coef has FIELD
?quo? : (%,%) -> % if Coef has FIELD
?rem? : (%,%) -> % if Coef has FIELD
```

These exports come from (p52) RadicalCategory():

```
nthRoot : (%,Integer) -> % if Coef has ALGEBRA FRAC INT
sqrt : % -> % if Coef has ALGEBRA FRAC INT
?**? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
```

— category UPXSCAT UnivariatePuiseuxSeriesCategory —

```
)abbrev category UPXSCAT UnivariatePuiseuxSeriesCategory
++ Author: Clifton J. Williamson
++ Date Created: 21 December 1989
++ Date Last Updated: 20 September 1993
++ Basic Operations:
++ Related Domains:
++ Also See:
```

```
++ AMS Classifications:
++ Keywords: series, Puiseux
++ Examples:
++ References:
++ Description:
++ \spadtype{UnivariatePuiseuxSeriesCategory} is the category of Puiseux
++ series in one variable.

UnivariatePuiseuxSeriesCategory(Coef): Category == Definition where
  Coef : Ring
  NNI  ==> NonNegativeInteger
  RN   ==> Fraction Integer
  Term ==> Record(k:RN,c:Coef)

  Definition ==> UnivariatePowerSeriesCategory(Coef,RN) with

    series: (NNI,Stream Term) -> %
      ++ \spad{series(n,st)} creates a series from a common denomiator and
      ++ a stream of non-zero terms, where a term is an exponent-coefficient
      ++ pair.  The terms in the stream should be ordered by increasing order
      ++ of exponents and \spad{n} should be a common denominator for the
      ++ exponents in the stream of terms.
    multiplyExponents: (%,Fraction Integer) -> %
      ++ \spad{multiplyExponents(f,r)} multiplies all exponents of the power
      ++ series f by the positive rational number r.

    if Coef has Algebra Fraction Integer then
      integrate: % -> %
        ++ \spad{integrate(f(x))} returns an anti-derivative of the power
        ++ series \spad{f(x)} with constant coefficient 1.
        ++ We may integrate a series when we can divide coefficients
        ++ by rational numbers.
      if Coef has integrate: (Coef,Symbol) -> Coef and _
         Coef has variables: Coef -> List Symbol then
        integrate: (%,Symbol) -> %
          ++ \spad{integrate(f(x),var)} returns an anti-derivative of the power
          ++ series \spad{f(x)} with respect to the variable \spad{var}.
      if Coef has TranscendentalFunctionCategory and _
         Coef has PrimitiveFunctionCategory and _
         Coef has AlgebraicallyClosedFunctionSpace Integer then
        integrate: (%,Symbol) -> %
          ++ \spad{integrate(f(x),y)} returns an anti-derivative of
          ++ the power series \spad{f(x)} with respect to the variable
          ++ \spad{y}.
      RadicalCategory
        --++ We provide rational powers when we can divide coefficients
        --++ by integers.
      TranscendentalFunctionCategory
        --++ We provide transcendental functions when we can divide
        --++ coefficients by integers.
```

```
    if Coef has Field then Field
        --++ Univariate Puiseux series over a field form a field.
```

———-

— **UPXSCAT.dotabb** —

```
"UPXSCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=UPXSCAT"];
"UPXSCAT" -> "TRANFUN"
"UPXSCAT" -> "FIELD"
"UPXSCAT" -> "RADCAT"
"UPXSCAT" -> "UPSCAT"
```

———-

— **UPXSCAT.dotfull** —

```
"UnivariatePuiseuxSeriesCategory(a:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=UPXSCAT"];
"UnivariatePuiseuxSeriesCategory(a:Ring)" ->
    "UnivariatePowerSeriesCategory(a:Ring,Fraction(Integer))"
"UnivariatePuiseuxSeriesCategory(a:Ring)" ->
    "TranscendentalFunctionCategory()"
"UnivariatePuiseuxSeriesCategory(a:Ring)" ->
    "Field()"
"UnivariatePuiseuxSeriesCategory(a:Ring)" ->
    "RadicalCategory()"
```

———-

— **UPXSCAT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"UnivariatePuiseuxSeriesCategory(a:Ring)" [color=lightblue];
"UnivariatePuiseuxSeriesCategory(a:Ring)" ->
    "UnivariatePowerSeriesCategory(a:Ring,Fraction(Integer))"
"UnivariatePuiseuxSeriesCategory(a:Ring)" ->
    "TRANFUN..."
"UnivariatePuiseuxSeriesCategory(a:Ring)" ->
    "FIELD..."
```

```
"UnivariatePuiseuxSeriesCategory(a:Ring)" ->
    "RADCAT..."

"UnivariatePowerSeriesCategory(a:Ring,Fraction(Integer))" [color=seagreen];
"UnivariatePowerSeriesCategory(a:Ring,Fraction(Integer))" ->
    "UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)"

"UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)"
 [color=lightblue];
"UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)" ->
 "PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:SingletonAsOrderedSet)"

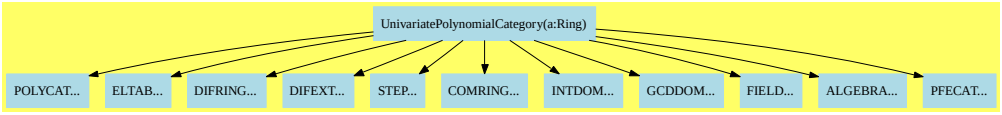
"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:SingletonAsOrderedSet)"
 [color=seagreen];
"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:SingletonAsOrderedSet)"
  -> "PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)"

"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)"
 [color=lightblue];
"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)" ->
    "AMR..."

"TRANFUN..." [color=lightblue];
"FIELD..." [color=lightblue];
"RADCAT..." [color=lightblue];
"AMR..." [color=lightblue];
}
```

———-

17.0.204 UnivariatePolynomialCategory (UPOLYC)



— **UnivariatePolynomialCategory.input** —

```
)set break resume
)sys rm -f UnivariatePolynomialCategory.output
)spool UnivariatePolynomialCategory.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 1
)show UnivariatePolynomialCategory
--R UnivariatePolynomialCategory R: Ring  is a category constructor
--R Abbreviation for UnivariatePolynomialCategory is UPOLYC
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for UPOLYC
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,R) -> %                      ?*? : (R,%) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R D : (%,(R -> R)) -> %                 D : % -> %
--R D : (%,NonNegativeInteger) -> %       1 : () -> %
--R 0 : () -> %                           ?^? : (%,PositiveInteger) -> %
--R coefficients : % -> List R            coerce : R -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R degree : % -> NonNegativeInteger      differentiate : % -> %
--R ?.? : (%,%) -> %                      ?.? : (%,R) -> R
--R eval : (%,List %,List %) -> %         eval : (%,%,%) -> %
--R eval : (%,Equation %) -> %            eval : (%,List Equation %) -> %
--R ground : % -> R                       ground? : % -> Boolean
--R hash : % -> SingleInteger             init : () -> % if R has STEP
--R latex : % -> String                   leadingCoefficient : % -> R
--R leadingMonomial : % -> %              map : ((R -> R),%) -> %
--R monomial? : % -> Boolean              monomials : % -> List %
--R one? : % -> Boolean                   primitiveMonomials : % -> List %
--R pseudoRemainder : (%,%) -> %          recip : % -> Union(%,"failed")
--R reductum : % -> %                     retract : % -> R
--R sample : () -> %                      zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (Fraction Integer,%) -> % if R has ALGEBRA FRAC INT
--R ?*? : (%,Fraction Integer) -> % if R has ALGEBRA FRAC INT
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (%,R) -> % if R has FIELD
--R ?<? : (%,%) -> Boolean if R has ORDSET
--R ?<=? : (%,%) -> Boolean if R has ORDSET
--R ?>? : (%,%) -> Boolean if R has ORDSET
--R ?>=? : (%,%) -> Boolean if R has ORDSET
--R D : (%,(R -> R),NonNegativeInteger) -> %
--R D : (%,List Symbol,List NonNegativeInteger) -> % if R has PDRING SYMBOL
--R D : (%,Symbol,NonNegativeInteger) -> % if R has PDRING SYMBOL
--R D : (%,List Symbol) -> % if R has PDRING SYMBOL
--R D : (%,Symbol) -> % if R has PDRING SYMBOL
--R D : (%,List SingletonAsOrderedSet,List NonNegativeInteger) -> %
--R D : (%,SingletonAsOrderedSet,NonNegativeInteger) -> %
--R D : (%,List SingletonAsOrderedSet) -> %
```

```
--R D : (%,SingletonAsOrderedSet) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R associates? : (%,%) -> Boolean if R has INTDOM
--R binomThmExpt : (%,%,NonNegativeInteger) -> % if R has COMRING
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if and(has($,CharacteristicNonZero),has(R,PolynomialFactorizationEx
--R coefficient : (%,List SingletonAsOrderedSet,List NonNegativeInteger) -> %
--R coefficient : (%,SingletonAsOrderedSet,NonNegativeInteger) -> %
--R coefficient : (%,NonNegativeInteger) -> R
--R coerce : % -> % if R has INTDOM
--R coerce : Fraction Integer -> % if R has RETRACT FRAC INT or R has ALGEBRA FRAC INT
--R coerce : SingletonAsOrderedSet -> %
--R composite : (Fraction %,%) -> Union(Fraction %,"failed") if R has INTDOM
--R composite : (%,%) -> Union(%,"failed") if R has INTDOM
--R conditionP : Matrix % -> Union(Vector %,"failed") if and(has($,CharacteristicNonZero),has(R,PolynomialF
--R content : (%,SingletonAsOrderedSet) -> % if R has GCDDOM
--R content : % -> R if R has GCDDOM
--R convert : % -> InputForm if SingletonAsOrderedSet has KONVERT INFORM and R has KONVERT INFORM
--R convert : % -> Pattern Integer if SingletonAsOrderedSet has KONVERT PATTERN INT and R has KONVERT PATTE
--R convert : % -> Pattern Float if SingletonAsOrderedSet has KONVERT PATTERN FLOAT and R has KONVERT PATTE
--R degree : (%,List SingletonAsOrderedSet) -> List NonNegativeInteger
--R degree : (%,SingletonAsOrderedSet) -> NonNegativeInteger
--R differentiate : (%,(R -> R),%) -> %
--R differentiate : (%,(R -> R)) -> %
--R differentiate : (%,(R -> R),NonNegativeInteger) -> %
--R differentiate : (%,List Symbol,List NonNegativeInteger) -> % if R has PDRING SYMBOL
--R differentiate : (%,Symbol,NonNegativeInteger) -> % if R has PDRING SYMBOL
--R differentiate : (%,List Symbol) -> % if R has PDRING SYMBOL
--R differentiate : (%,Symbol) -> % if R has PDRING SYMBOL
--R differentiate : (%,NonNegativeInteger) -> %
--R differentiate : (%,List SingletonAsOrderedSet,List NonNegativeInteger) -> %
--R differentiate : (%,SingletonAsOrderedSet,NonNegativeInteger) -> %
--R differentiate : (%,List SingletonAsOrderedSet) -> %
--R differentiate : (%,SingletonAsOrderedSet) -> %
--R discriminant : % -> R if R has COMRING
--R discriminant : (%,SingletonAsOrderedSet) -> % if R has COMRING
--R divide : (%,%) -> Record(quotient: %,remainder: %) if R has FIELD
--R divideExponents : (%,NonNegativeInteger) -> Union(%,"failed")
--R ?.? : (%,Fraction %) -> Fraction % if R has INTDOM
--R elt : (Fraction %,R) -> R if R has FIELD
--R elt : (Fraction %,Fraction %) -> Fraction % if R has INTDOM
--R euclideanSize : % -> NonNegativeInteger if R has FIELD
--R eval : (%,List SingletonAsOrderedSet,List %) -> %
--R eval : (%,SingletonAsOrderedSet,%) -> %
--R eval : (%,List SingletonAsOrderedSet,List R) -> %
--R eval : (%,SingletonAsOrderedSet,R) -> %
--R expressIdealMember : (List %,%) -> Union(List %,"failed") if R has FIELD
--R exquo : (%,%) -> Union(%,"failed") if R has INTDOM
--R exquo : (%,R) -> Union(%,"failed") if R has INTDOM
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %) if R has FIELD
```

```
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed") if R has FIELD
--R factor : % -> Factored % if R has PFECAT
--R factorPolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial % if
--R factorSquareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolyno
--R gcd : (%,%) -> % if R has GCDDOM
--R gcd : List % -> % if R has GCDDOM
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivari
--R integrate : % -> % if R has ALGEBRA FRAC INT
--R isExpt : % -> Union(Record(var: SingletonAsOrderedSet,exponent: NonNegativeInteger),"failed"
--R isPlus : % -> Union(List %,"failed")
--R isTimes : % -> Union(List %,"failed")
--R karatsubaDivide : (%,NonNegativeInteger) -> Record(quotient: %,remainder: %)
--R lcm : (%,%) -> % if R has GCDDOM
--R lcm : List % -> % if R has GCDDOM
--R mainVariable : % -> Union(SingletonAsOrderedSet,"failed")
--R makeSUP : % -> SparseUnivariatePolynomial R
--R mapExponents : ((NonNegativeInteger -> NonNegativeInteger),%) -> %
--R max : (%,%) -> % if R has ORDSET
--R min : (%,%) -> % if R has ORDSET
--R minimumDegree : (%,List SingletonAsOrderedSet) -> List NonNegativeInteger
--R minimumDegree : (%,SingletonAsOrderedSet) -> NonNegativeInteger
--R minimumDegree : % -> NonNegativeInteger
--R monicDivide : (%,%) -> Record(quotient: %,remainder: %)
--R monicDivide : (%,%,SingletonAsOrderedSet) -> Record(quotient: %,remainder: %)
--R monomial : (%,List SingletonAsOrderedSet,List NonNegativeInteger) -> %
--R monomial : (%,SingletonAsOrderedSet,NonNegativeInteger) -> %
--R monomial : (R,NonNegativeInteger) -> %
--R multiEuclidean : (List %,%) -> Union(List %,"failed") if R has FIELD
--R multiplyExponents : (%,NonNegativeInteger) -> %
--R multivariate : (SparseUnivariatePolynomial %,SingletonAsOrderedSet) -> %
--R multivariate : (SparseUnivariatePolynomial R,SingletonAsOrderedSet) -> %
--R nextItem : % -> Union(%,"failed") if R has STEP
--R numberOfMonomials : % -> NonNegativeInteger
--R order : (%,%) -> NonNegativeInteger if R has INTDOM
--R patternMatch : (%,Pattern Integer,PatternMatchResult(Integer,%)) -> PatternMatchResult(Integ
--R patternMatch : (%,Pattern Float,PatternMatchResult(Float,%)) -> PatternMatchResult(Float,%)
--R pomopo! : (%,R,NonNegativeInteger,%) -> %
--R prime? : % -> Boolean if R has PFECAT
--R primitivePart : (%,SingletonAsOrderedSet) -> % if R has GCDDOM
--R primitivePart : % -> % if R has GCDDOM
--R principalIdeal : List % -> Record(coef: List %,generator: %) if R has FIELD
--R pseudoDivide : (%,%) -> Record(coef: R,quotient: %,remainder: %) if R has INTDOM
--R pseudoQuotient : (%,%) -> % if R has INTDOM
--R ?quo? : (%,%) -> % if R has FIELD
--R reducedSystem : Matrix % -> Matrix R
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix R,vec: Vector R)
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer) if R
--R reducedSystem : Matrix % -> Matrix Integer if R has LINEXP INT
--R ?rem? : (%,%) -> % if R has FIELD
--R resultant : (%,%) -> R if R has COMRING
```

```
--R resultant : (%,%,SingletonAsOrderedSet) -> % if R has COMRING
--R retract : % -> SingletonAsOrderedSet
--R retract : % -> Integer if R has RETRACT INT
--R retract : % -> Fraction Integer if R has RETRACT FRAC INT
--R retractIfCan : % -> Union(SingletonAsOrderedSet,"failed")
--R retractIfCan : % -> Union(Integer,"failed") if R has RETRACT INT
--R retractIfCan : % -> Union(Fraction Integer,"failed") if R has RETRACT FRAC INT
--R retractIfCan : % -> Union(R,"failed")
--R separate : (%,%) -> Record(primePart: %,commonPart: %) if R has GCDDOM
--R shiftLeft : (%,NonNegativeInteger) -> %
--R shiftRight : (%,NonNegativeInteger) -> %
--R sizeLess? : (%,%) -> Boolean if R has FIELD
--R solveLinearPolynomialEquation : (List SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> Unio
--R squareFree : % -> Factored % if R has GCDDOM
--R squareFreePart : % -> % if R has GCDDOM
--R squareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial % if R has PI
--R subResultantGcd : (%,%) -> % if R has INTDOM
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R totalDegree : (%,List SingletonAsOrderedSet) -> NonNegativeInteger
--R totalDegree : % -> NonNegativeInteger
--R unit? : % -> Boolean if R has INTDOM
--R unitCanonical : % -> % if R has INTDOM
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %) if R has INTDOM
--R univariate : % -> SparseUnivariatePolynomial R
--R univariate : (%,SingletonAsOrderedSet) -> SparseUnivariatePolynomial %
--R unmakeSUP : SparseUnivariatePolynomial R -> %
--R variables : % -> List SingletonAsOrderedSet
--R vectorise : (%,NonNegativeInteger) -> Vector R
--R
--E 1

)spool
)lisp (bye)
```

———-

— UnivariatePolynomialCategory.help —

```
=====================================================================
UnivariatePolynomialCategory examples
=====================================================================

See Also:
o )show UnivariatePolynomialCategory
```

———-

See:

⇐ “Algebra” (ALGEBRA) 11.0.152 on page 947
⇐ “CommutativeRing” (COMRING) 10.0.137 on page 843
⇐ “DifferentialExtension” (DIFEXT) 11.0.153 on page 953
⇐ “DifferentialRing” (DIFRING) 10.0.138 on page 848
⇐ “Eltable” (ELTAB) 2.0.10 on page 29
⇐ “Field” (FIELD) 16.0.185 on page 1221
⇐ “GcdDomain” (GCDDOM) 13.0.173 on page 1129
⇐ “IntegralDomain” (INTDOM) 12.0.164 on page 1047
⇐ “PolynomialCategory” (POLYCAT) 16.0.189 on page 1255
⇐ “PolynomialFactorizationExplicit” (PFECAT) 15.0.183 on page 1202
⇐ “StepThrough” (STEP) 4.0.64 on page 252

Exports:

0	1	associates?
binomThmExpt	characteristic	charthRoot
coefficient	coefficients	coerce
composite	conditionP	content
convert	D	degree
differentiate	discriminant	divide
divideExponents	elt	euclideanSize
eval	expressIdealMember	exquo
extendedEuclidean	factor	factorPolynomial
factorSquareFreePolynomial	gcd	gcdPolynomial
ground	ground?	hash
init	integrate	isExpt
isPlus	isTimes	karatsubaDivide
latex	lcm	leadingCoefficient
leadingMonomial	mainVariable	makeSUP
map	mapExponents	max
min	minimumDegree	monicDivide
monomial	monomial?	monomials
multiEuclidean	multiplyExponents	multivariate
nextItem	numberOfMonomials	one?
order	patternMatch	pomopo!
prime?	primitiveMonomials	primitivePart
principalIdeal	pseudoDivide	pseudoQuotient
pseudoRemainder	recip	reducedSystem
reductum	resultant	retract
retractIfCan	sample	separate
shiftLeft	shiftRight	sizeLess?
solveLinearPolynomialEquation	squareFree	squareFreePart
squareFreePolynomial	subResultantGcd	subtractIfCan
totalDegree	unit?	unitCanonical
unitNormal	univariate	unmakeSUP
variables	vectorise	zero?
?*?	?**?	?+?
?-?	-?	?=?
?^?	?.?	?~=?
?/?	?<?	?<=?
?>?	?>=?	?quo?
?rem?		

Attributes exported:

- if $ has CommutativeRing then commutative("*") where **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- if $ has IntegralDomain then noZeroDivisors where **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- if R has Field then additiveValuation where **additiveValuation** implies

`euclideanSize(a*b)=euclideanSize(a)+euclideanSize(b)`.

- if $ has canonicalUnitNormal then canonicalUnitNormal where **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
discriminant : % -> R if R has COMRING
divideExponents : (%,NonNegativeInteger) -> Union(%,"failed")
monicDivide : (%,%) -> Record(quotient: %,remainder: %)
multiplyExponents : (%,NonNegativeInteger) -> %
pseudoRemainder : (%,%) -> %
resultant : (%,%) -> R if R has COMRING
subResultantGcd : (%,%) -> % if R has INTDOM
```

These are implemented by this category:

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean if R has INTDOM
binomThmExpt : (%,%,NonNegativeInteger) -> %
    if R has COMRING
characteristic : () -> NonNegativeInteger
coefficient : (%,NonNegativeInteger) -> R
coefficients : % -> List R
coerce : R -> %
coerce : Fraction Integer -> %
    if R has RETRACT FRAC INT
    or R has ALGEBRA FRAC INT
coerce : % -> % if R has INTDOM
content : % -> R if R has GCDDOM
coerce : Integer -> %
coerce : % -> OutputForm
coerce : SingletonAsOrderedSet -> %
composite : (Fraction %,%) -> Union(Fraction %,"failed")
    if R has INTDOM
composite : (%,%) -> Union(%,"failed")
    if R has INTDOM
content : (%,SingletonAsOrderedSet) -> %
    if R has GCDDOM
D : (%,List SingletonAsOrderedSet) -> %
```

```
D : (%,SingletonAsOrderedSet) -> %
D : (%,List SingletonAsOrderedSet,List NonNegativeInteger) -> %
D : (%,SingletonAsOrderedSet,NonNegativeInteger) -> %
degree : % -> NonNegativeInteger
degree :
  (%,List SingletonAsOrderedSet) -> List NonNegativeInteger
differentiate :
  (%,List SingletonAsOrderedSet,List NonNegativeInteger) -> %
differentiate :
  (%,SingletonAsOrderedSet,NonNegativeInteger) -> %
differentiate : (%,List SingletonAsOrderedSet) -> %
differentiate : (%,(R -> R),%) -> %
differentiate : (%,(R -> R)) -> %
differentiate : % -> %
differentiate : (%,SingletonAsOrderedSet) -> %
divide : (%,%) -> Record(quotient: %,remainder: %)
    if R has FIELD
elt : (Fraction %,Fraction %) -> Fraction %
    if R has INTDOM
elt : (Fraction %,R) -> R if R has FIELD
euclideanSize : % -> NonNegativeInteger
    if R has FIELD
eval : (%,List SingletonAsOrderedSet,List %) -> %
eval : (%,SingletonAsOrderedSet,%) -> %
eval : (%,List SingletonAsOrderedSet,List R) -> %
eval : (%,SingletonAsOrderedSet,R) -> %
eval : (%,List Equation %) -> %
eval : (%,List %,List %) -> %
eval : (%,%,%) -> %
eval : (%,Equation %) -> %
exquo : (%,R) -> Union(%,"failed")
    if R has INTDOM
exquo : (%,%) -> Union(%,"failed")
    if R has INTDOM
factor : % -> Factored % if R has PFECAT
factorPolynomial :
  SparseUnivariatePolynomial % ->
    Factored SparseUnivariatePolynomial %
      if R has PFECAT
factorSquareFreePolynomial :
  SparseUnivariatePolynomial % ->
    Factored SparseUnivariatePolynomial %
      if R has PFECAT
gcd : (%,%) -> % if R has GCDDOM
gcd : List % -> % if R has GCDDOM
gcdPolynomial :
  (SparseUnivariatePolynomial %,
   SparseUnivariatePolynomial %) ->
     SparseUnivariatePolynomial %
       if R has GCDDOM
```

```
ground : % -> R
ground? : % -> Boolean
hash : % -> SingleInteger
init : () -> % if R has STEP
integrate : % -> % if R has ALGEBRA FRAC INT
karatsubaDivide :
  (%,NonNegativeInteger) -> Record(quotient: %,remainder: %)
latex : % -> String
lcm : (%,%) -> % if R has GCDDOM
lcm : List % -> % if R has GCDDOM
leadingCoefficient : % -> R
leadingMonomial : % -> %
mainVariable : % -> Union(SingletonAsOrderedSet,"failed")
makeSUP : % -> SparseUnivariatePolynomial R
map : ((R -> R),%) -> %
mapExponents :
  ((NonNegativeInteger -> NonNegativeInteger),%) -> %
max : (%,%) -> % if R has ORDSET
min : (%,%) -> % if R has ORDSET
minimumDegree :
  (%,SingletonAsOrderedSet) -> NonNegativeInteger
minimumDegree :
  (%,List SingletonAsOrderedSet) -> List NonNegativeInteger
monomial : (R,NonNegativeInteger) -> %
monomial : (%,SingletonAsOrderedSet,NonNegativeInteger) -> %
monomial? : % -> Boolean
nextItem : % -> Union(%,"failed") if R has STEP
numberOfMonomials : % -> NonNegativeInteger
one? : % -> Boolean
order : (%,%) -> NonNegativeInteger
    if R has INTDOM
pomopo! : (%,R,NonNegativeInteger,%) -> %
prime? : % -> Boolean if R has PFECAT
pseudoDivide :
  (%,%) -> Record(coef: R,quotient: %,remainder: %)
    if R has INTDOM
pseudoQuotient : (%,%) -> % if R has INTDOM
recip : % -> Union(%,"failed")
reducedSystem : (Matrix %,Vector %) ->
  Record(mat: Matrix Integer,vec: Vector Integer)
    if R has LINEXP INT
reducedSystem : Matrix % -> Matrix Integer
    if R has LINEXP INT
reductum : % -> %
retract : % -> R
retract : % -> Integer if R has RETRACT INT
retract : % -> Fraction Integer
    if R has RETRACT FRAC INT
retractIfCan : % -> Union(Integer,"failed")
    if R has RETRACT INT
```

```
retractIfCan : % -> Union(Fraction Integer,"failed")
    if R has RETRACT FRAC INT
retractIfCan : % -> Union(R,"failed")
sample : () -> %
separate : (%,%) -> Record(primePart: %,commonPart: %)
    if R has GCDDOM
shiftLeft : (%,NonNegativeInteger) -> %
shiftRight : (%,NonNegativeInteger) -> %
solveLinearPolynomialEquation :
  (List SparseUnivariatePolynomial %,
   SparseUnivariatePolynomial %) ->
     Union(List SparseUnivariatePolynomial %,"failed")
       if R has PFECAT
squareFree : % -> Factored % if R has GCDDOM
squareFreePart : % -> % if R has GCDDOM
squareFreePolynomial :
  SparseUnivariatePolynomial % ->
    Factored SparseUnivariatePolynomial %
     if R has PFECAT
subtractIfCan : (%,%) -> Union(%,"failed")
totalDegree :
  (%,List SingletonAsOrderedSet) -> NonNegativeInteger
unit? : % -> Boolean if R has INTDOM
unitCanonical : % -> % if R has INTDOM
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
    if R has INTDOM
unmakeSUP : SparseUnivariatePolynomial R -> %
variables : % -> List SingletonAsOrderedSet
vectorise : (%,NonNegativeInteger) -> Vector R
zero? : % -> Boolean
?<=? : (%,%) -> Boolean if R has ORDSET
?>? : (%,%) -> Boolean if R has ORDSET
?>=? : (%,%) -> Boolean if R has ORDSET
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,R) -> %
?*? : (R,%) -> %
?*? : (Fraction Integer,%) -> %
    if R has ALGEBRA FRAC INT
?*? : (%,Fraction Integer) -> %
    if R has ALGEBRA FRAC INT
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?/? : (%,R) -> % if R has FIELD
?-? : (%,%) -> %
-? : % -> %
?^? : (%,PositiveInteger) -> %
```

```
?^? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?.? : (%,Fraction %) -> Fraction %
    if R has INTDOM
```

These exports come from (p1255) PolynomialCategory(R,N,S)
where R:Ring, N:NonNegativeInteger, S:SingletonAsOrderedSet:

```
charthRoot : % -> Union(%,"failed")
  if
   and(has($,CharacteristicNonZero),
       has(R,PolynomialFactorizationExplicit))
   or R has CHARNZ
coefficient :
  (%,SingletonAsOrderedSet,NonNegativeInteger) -> %
coefficient :
  (%,List SingletonAsOrderedSet,List NonNegativeInteger) -> %
conditionP : Matrix % -> Union(Vector %,"failed")
 if and(has($,CharacteristicNonZero),
        has(R,PolynomialFactorizationExplicit))
convert : % -> Pattern Integer
    if SingletonAsOrderedSet has KONVERT PATTERN INT
    and R has KONVERT PATTERN INT
convert : % -> Pattern Float
    if SingletonAsOrderedSet has KONVERT PATTERN FLOAT
    and R has KONVERT PATTERN FLOAT
convert : % -> InputForm
    if SingletonAsOrderedSet has KONVERT INFORM
    and R has KONVERT INFORM
degree : (%,SingletonAsOrderedSet) -> NonNegativeInteger
discriminant : (%,SingletonAsOrderedSet) -> %
    if R has COMRING
isExpt : % ->
  Union(
   Record(var: SingletonAsOrderedSet,exponent: NonNegativeInteger),
  "failed")
isPlus : % -> Union(List %,"failed")
isTimes : % -> Union(List %,"failed")
minimumDegree : % -> NonNegativeInteger
monicDivide :
  (%,%,SingletonAsOrderedSet) -> Record(quotient: %,remainder: %)
monomial :
  (%,List SingletonAsOrderedSet,List NonNegativeInteger) -> %
monomials : % -> List %
multivariate :
  (SparseUnivariatePolynomial %,SingletonAsOrderedSet) -> %
multivariate :
  (SparseUnivariatePolynomial R,SingletonAsOrderedSet) -> %
patternMatch :
```

```
 (%,Pattern Integer,PatternMatchResult(Integer,%)) ->
   PatternMatchResult(Integer,%)
    if SingletonAsOrderedSet has PATMAB INT
    and R has PATMAB INT
patternMatch :
 (%,Pattern Float,PatternMatchResult(Float,%)) ->
   PatternMatchResult(Float,%)
    if SingletonAsOrderedSet has PATMAB FLOAT
    and R has PATMAB FLOAT
primitiveMonomials : % -> List %
primitivePart : (%,SingletonAsOrderedSet) -> %
    if R has GCDDOM
primitivePart : % -> % if R has GCDDOM
reducedSystem : Matrix % -> Matrix R
reducedSystem : (Matrix %,Vector %) ->
  Record(mat: Matrix R,vec: Vector R)
resultant : (%,%,SingletonAsOrderedSet) -> %
    if R has COMRING
retract : % -> SingletonAsOrderedSet
retractIfCan : % -> Union(SingletonAsOrderedSet,"failed")
totalDegree : % -> NonNegativeInteger
univariate : % -> SparseUnivariatePolynomial R
univariate :
  (%,SingletonAsOrderedSet) -> SparseUnivariatePolynomial %
?<? : (%,%) -> Boolean if R has ORDSET
```

These exports come from (p29) Eltable(R:Ring,R:Ring):

```
?.? : (%,R) -> R
```

These exports come from (p29) Eltable(R:UPOLYC,R:UPOLYC):

```
?.? : (%,%) -> %
```

These exports come from (p848) DifferentialRing():

```
D : % -> %
D : (%,NonNegativeInteger) -> %
differentiate : (%,NonNegativeInteger) -> %
```

These exports come from (p953) DifferentialExtension(R:Ring):

```
D : (%,(R -> R)) -> %
D : (%,(R -> R),NonNegativeInteger) -> %
D : (%,Symbol) -> % if R has PDRING SYMBOL
D : (%,List Symbol) -> % if R has PDRING SYMBOL
D : (%,Symbol,NonNegativeInteger) -> %
    if R has PDRING SYMBOL
D : (%,List Symbol,List NonNegativeInteger) -> %
```

```
    if R has PDRING SYMBOL
differentiate : (%,(R -> R),NonNegativeInteger) -> %
differentiate : (%,Symbol) -> %
    if R has PDRING SYMBOL
differentiate : (%,List Symbol) -> %
    if R has PDRING SYMBOL
differentiate : (%,Symbol,NonNegativeInteger) -> %
    if R has PDRING SYMBOL
differentiate : (%,List Symbol,List NonNegativeInteger) -> %
    if R has PDRING SYMBOL
```

These exports come from (p252) StepThrough()

These exports come from (p843) CommutativeRing()

These exports come from (p1047) IntegralDomain()

These exports come from (p1129) GcdDomain()

These exports come from (p1221) Field()

```
expressIdealMember : (List %,%) -> Union(List %,"failed")
    if R has FIELD
extendedEuclidean : (%,%) ->
  Record(coef1: %,coef2: %,generator: %)
    if R has FIELD
extendedEuclidean :
  (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
    if R has FIELD
multiEuclidean : (List %,%) -> Union(List %,"failed")
    if R has FIELD
principalIdeal : List % -> Record(coef: List %,generator: %)
    if R has FIELD
sizeLess? : (%,%) -> Boolean if R has FIELD
?quo? : (%,%) -> % if R has FIELD
?rem? : (%,%) -> % if R has FIELD
```

These exports come from (p947) Algebra(Fraction(Integer))

These exports come from (p1202) PolynomialFactorizationExplicit()

— category UPOLYC UnivariatePolynomialCategory —

```
)abbrev category UPOLYC UnivariatePolynomialCategory
++ Author:
++ Date Created:
++ Date Last Updated:
++ Basic Functions: Ring, monomial, coefficient, reductum, differentiate,
++ elt, map, resultant, discriminant
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ The category of univariate polynomials over a ring R.
++ No particular model is assumed - implementations can be either
++ sparse or dense.

UnivariatePolynomialCategory(R:Ring): Category ==
 Join(PolynomialCategory(R, NonNegativeInteger, SingletonAsOrderedSet),
      Eltable(R, R), Eltable(%, %), DifferentialRing,
      DifferentialExtension R) with
    vectorise        : (%,NonNegativeInteger) -> Vector R
      ++ vectorise(p, n) returns \spad{[a0,...,a(n-1)]} where
      ++ \spad{p = a0 + a1*x + ... + a(n-1)*x**(n-1)} + higher order terms.
      ++ The degree of polynomial p can be different from \spad{n-1}.
    makeSUP: % -> SparseUnivariatePolynomial R
      ++ makeSUP(p) converts the polynomial p to be of type
      ++ SparseUnivariatePolynomial over the same coefficients.
    unmakeSUP: SparseUnivariatePolynomial R -> %
      ++ unmakeSUP(sup) converts sup of type
      ++ \spadtype{SparseUnivariatePolynomial(R)}
      ++ to be a member of the given type.
      ++ Note that converse of makeSUP.
    multiplyExponents: (%,NonNegativeInteger) -> %
      ++ multiplyExponents(p,n) returns a new polynomial resulting from
      ++ multiplying all exponents of the polynomial p by the non negative
      ++ integer n.
    divideExponents: (%,NonNegativeInteger) -> Union(%,"failed")
      ++ divideExponents(p,n) returns a new polynomial resulting from
      ++ dividing all exponents of the polynomial p by the non negative
      ++ integer n, or "failed" if some exponent is not exactly divisible
      ++ by n.
    monicDivide: (%,%) -> Record(quotient:%,remainder:%)
      ++ monicDivide(p,q) divide the polynomial p by the monic polynomial q,
```

```
      ++ returning the pair \spad{[quotient, remainder]}.
      ++ Error: if q isn't monic.
-- These three are for Karatsuba
    karatsubaDivide: (%,NonNegativeInteger) -> Record(quotient:%,remainder:%)
      ++ \spad{karatsubaDivide(p,n)} returns the same as
      ++ \spad{monicDivide(p,monomial(1,n))}
    shiftRight: (%,NonNegativeInteger) -> %
      ++ \spad{shiftRight(p,n)} returns
      ++ \spad{monicDivide(p,monomial(1,n)).quotient}
    shiftLeft: (%,NonNegativeInteger) -> %
      ++ \spad{shiftLeft(p,n)} returns \spad{p * monomial(1,n)}
    pseudoRemainder: (%,%) -> %
       ++ pseudoRemainder(p,q) = r, for polynomials p and q, returns the
       ++ remainder when
       ++ \spad{p' := p*lc(q)**(deg p - deg q + 1)}
       ++ is pseudo right-divided by q, i.e. \spad{p' = s q + r}.
    differentiate: (%, R -> R, %) -> %
       ++ differentiate(p, d, x') extends the R-derivation d to an
       ++ extension D in \spad{R[x]} where Dx is given by x', and
       ++ returns \spad{Dp}.
    if R has StepThrough then StepThrough
    if R has CommutativeRing then
      discriminant: % -> R
        ++ discriminant(p) returns the discriminant of the polynomial p.
      resultant: (%,%) -> R
        ++ resultant(p,q) returns the resultant of the polynomials p and q.
    if R has IntegralDomain then
        Eltable(Fraction %, Fraction %)
        elt  : (Fraction %, Fraction %) -> Fraction %
             ++ elt(a,b) evaluates the fraction of univariate polynomials
             ++ \spad{a} with the distinguished variable replaced by b.
        order: (%, %) -> NonNegativeInteger
             ++ order(p, q) returns the largest n such that \spad{q**n}
             ++ divides polynomial p
             ++ i.e. the order of \spad{p(x)} at \spad{q(x)=0}.
        subResultantGcd: (%,%) -> %
           ++ subResultantGcd(p,q) computes the gcd of the polynomials p
           ++ and q using the SubResultant GCD algorithm.
        composite: (%, %) -> Union(%, "failed")
           ++ composite(p, q) returns h if \spad{p = h(q)}, and "failed"
           ++ no such h exists.
        composite: (Fraction %, %) -> Union(Fraction %, "failed")
           ++ composite(f, q) returns h if f = h(q), and "failed" is
           ++ no such h exists.
        pseudoQuotient: (%,%) -> %
           ++ pseudoQuotient(p,q) returns r, the quotient when
           ++ \spad{p' := p*lc(q)**(deg p - deg q + 1)}
           ++ is pseudo right-divided by q, i.e. \spad{p' = s q + r}.
        pseudoDivide: (%, %) -> Record(coef:R, quotient: %, remainder:%)
           ++ pseudoDivide(p,q) returns \spad{[c, q, r]}, when
```

```
        ++ \spad{p' := p*lc(q)**(deg p - deg q + 1) = c * p}
        ++ is pseudo right-divided by q, i.e. \spad{p' = s q + r}.
  if R has GcdDomain then
      separate: (%, %) -> Record(primePart:%, commonPart: %)
        ++ separate(p, q) returns \spad{[a, b]} such that polynomial
        ++ \spad{p = a b} and \spad{a} is relatively prime to q.
  if R has Field then
      EuclideanDomain
      additiveValuation
        ++ euclideanSize(a*b) = euclideanSize(a) + euclideanSize(b)
      elt      : (Fraction %, R) -> R
        ++ elt(a,r) evaluates the fraction of univariate polynomials
        ++ \spad{a} with the distinguished variable replaced by the
        ++ constant r.
  if R has Algebra Fraction Integer then
    integrate: % -> %
      ++ integrate(p) integrates the univariate polynomial p with respect
      ++ to its distinguished variable.
add
  pp,qq: SparseUnivariatePolynomial %

  variables(p) ==
    zero? p or zero?(degree p) => []
    [create()]

  degree(p:%,v:SingletonAsOrderedSet) == degree p

  totalDegree(p:%,lv:List SingletonAsOrderedSet) ==
     empty? lv => 0
     totalDegree p

  degree(p:%,lv:List SingletonAsOrderedSet) ==
     empty? lv => []
     [degree p]

  eval(p:%,lv: List SingletonAsOrderedSet,lq: List %):% ==
    empty? lv => p
    not empty? rest lv => _
      error "can only eval a univariate polynomial once"
    eval(p,first lv,first lq)$%

  eval(p:%,v:SingletonAsOrderedSet,q:%):% == p(q)

  eval(p:%,lv: List SingletonAsOrderedSet,lr: List R):% ==
    empty? lv => p
    not empty? rest lv => _
       error "can only eval a univariate polynomial once"
    eval(p,first lv,first lr)$%

  eval(p:%,v:SingletonAsOrderedSet,r:R):% == p(r)::%
```

```
eval(p:%,le:List Equation %):% ==
  empty? le  => p
  not empty? rest le => _
     error "can only eval a univariate polynomial once"
  mainVariable(lhs first le) case "failed" => p
  p(rhs first le)

mainVariable(p:%) ==
  zero? degree p =>  "failed"
  create()$SingletonAsOrderedSet

minimumDegree(p:%,v:SingletonAsOrderedSet) == minimumDegree p

minimumDegree(p:%,lv:List SingletonAsOrderedSet) ==
   empty? lv => []
   [minimumDegree p]

monomial(p:%,v:SingletonAsOrderedSet,n:NonNegativeInteger) ==
   mapExponents(x1+->x1+n,p)

coerce(v:SingletonAsOrderedSet):% == monomial(1,1)

makeSUP p ==
  zero? p => 0
  monomial(leadingCoefficient p,degree p) + makeSUP reductum p

unmakeSUP sp ==
  zero? sp => 0
  monomial(leadingCoefficient sp,degree sp) + unmakeSUP reductum sp

karatsubaDivide(p:%,n:NonNegativeInteger) == monicDivide(p,monomial(1,n))

shiftRight(p:%,n:NonNegativeInteger) ==
   monicDivide(p,monomial(1,n)).quotient

shiftLeft(p:%,n:NonNegativeInteger) == p * monomial(1,n)

if R has PolynomialFactorizationExplicit then
   PFBRU ==> PolynomialFactorizationByRecursionUnivariate(R,%)
   pp,qq:SparseUnivariatePolynomial %
   lpp:List SparseUnivariatePolynomial %
   SupR ==> SparseUnivariatePolynomial R
   sp:SupR

   solveLinearPolynomialEquation(lpp,pp) ==
     solveLinearPolynomialEquationByRecursion(lpp,pp)$PFBRU

   factorPolynomial(pp) ==
     factorByRecursion(pp)$PFBRU
```

```
   factorSquareFreePolynomial(pp) ==
     factorSquareFreeByRecursion(pp)$PFBRU

   import FactoredFunctions2(SupR,S)

   factor p ==
     zero? degree p  =>
       ansR:=factor leadingCoefficient p
       makeFR(unit(ansR)::%,
              [[w.flg,w.fctr::%,w.xpnt] for w in factorList ansR])
     map(unmakeSUP,factorPolynomial(makeSUP p)$R)

vectorise(p, n) ==
  m := minIndex(v := new(n, 0)$Vector(R))
  for i in minIndex v .. maxIndex v repeat
    qsetelt_!(v, i, coefficient(p, (i - m)::NonNegativeInteger))
  v

retract(p:%):R ==
  zero? p => 0
  zero? degree p => leadingCoefficient p
  error "Polynomial is not of degree 0"

retractIfCan(p:%):Union(R, "failed") ==
  zero? p => 0
  zero? degree p => leadingCoefficient p
  "failed"

if R has StepThrough then
   init() == init()$R::%

   nextItemInner: % -> Union(%,"failed")

   nextItemInner(n) ==
     zero? n => nextItem(0$R)::R::% -- assumed not to fail
     zero? degree n =>
       nn:=nextItem leadingCoefficient n
       nn case "failed" => "failed"
       nn::R::%
     n1:=reductum n
     n2:=nextItemInner n1 -- try stepping the reductum
     n2 case % => monomial(leadingCoefficient n,degree n) + n2
     1+degree n1 < degree n => -- there was a hole between lt n and n1
       monomial(leadingCoefficient n,degree n)+
         monomial(nextItem(init()$R)::R,1+degree n1)
     n3:=nextItem leadingCoefficient n
     n3 case "failed" => "failed"
     monomial(n3,degree n)
```

```
     nextItem(n) ==
       n1:=nextItemInner n
       n1 case "failed" => monomial(nextItem(init()$R)::R,1+degree(n))
       n1

  if R has GcdDomain then

    content(p:%,v:SingletonAsOrderedSet) == content(p)::%

    primeFactor: (%, %) -> %

    primeFactor(p, q) ==
      (p1 := (p exquo gcd(p, q))::%) = p => p
      primeFactor(p1, q)

    separate(p, q) ==
      a := primeFactor(p, q)
      [a, (p exquo a)::%]

  if R has CommutativeRing then
    differentiate(x:%, deriv:R -> R, x':%) ==
      d:% := 0
      while (dg := degree x) > 0 repeat
        lc := leadingCoefficient x
        d := d + x' * monomial(dg * lc, (dg - 1)::NonNegativeInteger)
               + monomial(deriv lc, dg)
        x := reductum x
      d + deriv(leadingCoefficient x)::%
  else
    ncdiff: (NonNegativeInteger, %) -> %
    -- computes d(x**n) given dx = x', non-commutative case
    ncdiff(n, x') ==
      zero? n => 0
      zero?(n1 := (n - 1)::NonNegativeInteger) => x'
      x' * monomial(1, n1) + monomial(1, 1) * ncdiff(n1, x')

    differentiate(x:%, deriv:R -> R, x':%) ==
      d:% := 0
      while (dg := degree x) > 0 repeat
        lc := leadingCoefficient x
        d := d + monomial(deriv lc, dg) + lc * ncdiff(dg, x')
        x := reductum x
      d + deriv(leadingCoefficient x)::%

  differentiate(x:%, deriv:R -> R) == differentiate(x, deriv, 1$%)$%

  differentiate(x:%) ==
      d:% := 0
      while (dg := degree x) > 0 repeat
        d:=d+monomial(dg*leadingCoefficient x,(dg-1)::NonNegativeInteger)
```

```
      x := reductum x
    d

differentiate(x:%,v:SingletonAsOrderedSet) == differentiate x

if R has IntegralDomain then

  elt(g:Fraction %, f:Fraction %) == ((numer g) f) / ((denom g) f)

  pseudoQuotient(p, q) ==
    (n := degree(p)::Integer - degree q + 1) < 1 => 0
    ((leadingCoefficient(q)**(n::NonNegativeInteger) * p
      - pseudoRemainder(p, q)) exquo q)::%

  pseudoDivide(p, q) ==
    (n := degree(p)::Integer - degree q + 1) < 1 => [1, 0, p]
    prem := pseudoRemainder(p, q)
    lc   := leadingCoefficient(q)**(n::NonNegativeInteger)
    [lc,((lc*p - prem) exquo q)::%, prem]

  composite(f:Fraction %, q:%) ==
    (n := composite(numer f, q)) case "failed" => "failed"
    (d := composite(denom f, q)) case "failed" => "failed"
    n::% / d::%

  composite(p:%, q:%) ==
    ground? p => p
    cqr := pseudoDivide(p, q)
    ground?(cqr.remainder) and
      ((v := cqr.remainder exquo cqr.coef) case %) and
        ((u := composite(cqr.quotient, q)) case %) and
          ((w := (u::%) exquo cqr.coef) case %) =>
            v::% + monomial(1, 1) * w::%
    "failed"

  elt(p:%, f:Fraction %) ==
    zero? p => 0
    ans:Fraction(%) := (leadingCoefficient p)::%::Fraction(%)
    n := degree p
    while not zero?(p:=reductum p) repeat
      ans := ans * f ** (n - (n := degree p))::NonNegativeInteger +
                 (leadingCoefficient p)::%::Fraction(%)
    zero? n => ans
    ans * f ** n

  order(p, q) ==
    zero? p => error "order: arguments must be nonzero"
    degree(q) < 1 => error "order: place must be non-trivial"
    ans:NonNegativeInteger := 0
    repeat
```

```
        (u  := p exquo q) case "failed" => return ans
        p   := u::%
        ans := ans + 1

    if R has GcdDomain then
      squareFree(p:%) ==
        squareFree(p)$UnivariatePolynomialSquareFree(R, %)

      squareFreePart(p:%) ==
        squareFreePart(p)$UnivariatePolynomialSquareFree(R, %)

    if R has PolynomialFactorizationExplicit then

      gcdPolynomial(pp,qq) ==
            zero? pp => unitCanonical qq  -- subResultantGcd can't handle 0
            zero? qq => unitCanonical pp
            unitCanonical(gcd(content (pp),content(qq))*
                   primitivePart
                      subResultantGcd(primitivePart pp,primitivePart qq))

      squareFreePolynomial pp ==
         squareFree(pp)$UnivariatePolynomialSquareFree(%,
                                    SparseUnivariatePolynomial %)

    if R has Field then
      elt(f:Fraction %, r:R) == ((numer f) r) / ((denom f) r)

      euclideanSize x ==
            zero? x =>
              error "euclideanSize called on 0 in Univariate Polynomial"
            degree x

      divide(x,y) ==
            zero? y => error "division by 0 in Univariate Polynomials"
            quot:=0
            lc := inv leadingCoefficient y
            while not zero?(x) and (degree x >= degree y) repeat
               f:=lc*leadingCoefficient x
               n:=(degree x - degree y)::NonNegativeInteger
               quot:=quot+monomial(f,n)
               x:=x-monomial(f,n)*y
            [quot,x]

    if R has Algebra Fraction Integer then

      integrate p ==
        ans:% := 0
        while p ^= 0 repeat
          l := leadingCoefficient p
          d := 1 + degree p
```

```
          ans := ans + inv(d::Fraction(Integer)) * monomial(l, d)
          p := reductum p
        ans
```

———-

— **UPOLYC.dotabb** —

```
"UPOLYC"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=UPOLYC"];
"UPOLYC" -> "POLYCAT"
"UPOLYC" -> "ELTAB"
"UPOLYC" -> "DIFRING"
"UPOLYC" -> "DIFEXT"
"UPOLYC" -> "STEP"
"UPOLYC" -> "COMRING"
"UPOLYC" -> "INTDOM"
"UPOLYC" -> "GCDDOM"
"UPOLYC" -> "FIELD"
"UPOLYC" -> "ALGEBRA"
"UPOLYC" -> "PFECAT"
```

———-

— **UPOLYC.dotfull** —

```
"UnivariatePolynomialCategory(a:Ring)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=UPOLYC"];
"UnivariatePolynomialCategory(a:Ring)" ->
   "PolynomialCategory(a:Ring,b:NonNegativeInteger,c:SingletonAsOrderedSet)"
"UnivariatePolynomialCategory(a:Ring)" ->
   "Eltable(a:Ring,b:Ring)"
"UnivariatePolynomialCategory(a:Ring)" ->
   "Eltable(a:UnivariatePolynomialCategory(a:Ring),b:UnivariatePolynomialCategory(a:Ring))"
"UnivariatePolynomialCategory(a:Ring)" ->
   "DifferentialRing()"
"UnivariatePolynomialCategory(a:Ring)" ->
   "DifferentialExtension(a:Ring)"
"UnivariatePolynomialCategory(a:Ring)" ->
   "StepThrough()"
"UnivariatePolynomialCategory(a:Ring)" ->
   "CommutativeRing()"
"UnivariatePolynomialCategory(a:Ring)" ->
   "IntegralDomain()"
"UnivariatePolynomialCategory(a:Ring)" ->
   "GcdDomain()"
```

```
"UnivariatePolynomialCategory(a:Ring)" ->
   "Field()"
"UnivariatePolynomialCategory(a:Ring)" ->
   "Algebra(Fraction(Integer))"
"UnivariatePolynomialCategory(a:Ring)" ->
   "PolynomialFactorizationExplicit()"
```

———-

— UPOLYC.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"UnivariatePolynomialCategory(a:Ring)" [color=lightblue];
"UnivariatePolynomialCategory(a:Ring)" -> "POLYCAT..."
"UnivariatePolynomialCategory(a:Ring)" -> "ELTAB..."
"UnivariatePolynomialCategory(a:Ring)" -> "DIFRING..."
"UnivariatePolynomialCategory(a:Ring)" -> "DIFEXT..."
"UnivariatePolynomialCategory(a:Ring)" -> "STEP..."
"UnivariatePolynomialCategory(a:Ring)" -> "COMRING..."
"UnivariatePolynomialCategory(a:Ring)" -> "INTDOM..."
"UnivariatePolynomialCategory(a:Ring)" -> "GCDDOM..."
"UnivariatePolynomialCategory(a:Ring)" -> "FIELD..."
"UnivariatePolynomialCategory(a:Ring)" -> "ALGEBRA..."
"UnivariatePolynomialCategory(a:Ring)" -> "PFECAT..."

"POLYCAT..." [color=lightblue];
"ELTAB..." [color=lightblue];
"DIFRING..." [color=lightblue];
"DIFEXT..." [color=lightblue];
"STEP..." [color=lightblue];
"COMRING..." [color=lightblue];
"INTDOM..." [color=lightblue];
"GCDDOM..." [color=lightblue];
"FIELD..." [color=lightblue];
"ALGEBRA..." [color=lightblue];
"PFECAT..." [color=lightblue];

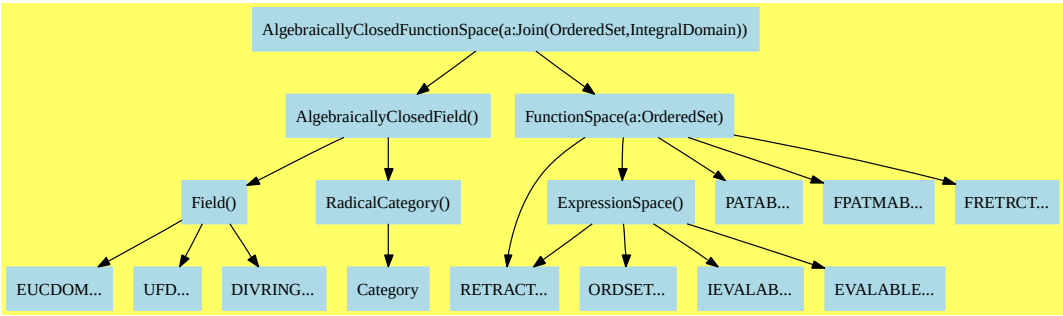
}
```

———-

Chapter 18

Category Layer 17

18.0.205 AlgebraicallyClosedFunctionSpace (ACFS)



— AlgebraicallyClosedFunctionSpace.input —

```
)set break resume
)sys rm -f AlgebraicallyClosedFunctionSpace.output
)spool AlgebraicallyClosedFunctionSpace.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show AlgebraicallyClosedFunctionSpace
--R AlgebraicallyClosedFunctionSpace R: Join(OrderedSet,IntegralDomain)  is a category constructor
--R Abbreviation for AlgebraicallyClosedFunctionSpace is ACFS
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for ACFS
--R
--R------------------------------ Operations --------------------------------
```

```
--R ?*? : (Fraction Integer,%) -> %
--R ?*? : (%,%) -> %
--R ?*? : (PositiveInteger,%) -> %
--R ?**? : (%,Integer) -> %
--R ?+? : (%,%) -> %
--R -? : % -> %
--R ?<? : (%,%) -> Boolean
--R ?=? : (%,%) -> Boolean
--R ?>=? : (%,%) -> Boolean
--R 0 : () -> %
--R ?^? : (%,PositiveInteger) -> %
--R applyQuote : (Symbol,%,%) -> %
--R belong? : BasicOperator -> Boolean
--R box : List % -> %
--R coerce : Symbol -> %
--R coerce : Fraction Integer -> %
--R coerce : Integer -> %
--R distribute : % -> %
--R elt : (BasicOperator,%) -> %
--R eval : (%,Kernel %,%) -> %
--R eval : (%,Equation %) -> %
--R eval : (%,List %,List %) -> %
--R freeOf? : (%,%) -> Boolean
--R gcd : List % -> %
--R ground : % -> R
--R hash : % -> SingleInteger
--R inv : % -> %
--R kernel : (BasicOperator,%) -> %
--R latex : % -> String
--R lcm : (%,%) -> %
--R max : (%,%) -> %
--R nthRoot : (%,Integer) -> %
--R paren : % -> %
--R prime? : % -> Boolean
--R recip : % -> Union(%,"failed")
--R retract : % -> Kernel %
--R retract : % -> R
--R rootOf : % -> %
--R rootsOf : (%,Symbol) -> List %
--R rootsOf : Polynomial % -> List %
--R sizeLess? : (%,%) -> Boolean
--R squareFree : % -> Factored %
--R subst : (%,Equation %) -> %
--R unit? : % -> Boolean
--R variables : % -> List Symbol
--R zeroOf : (%,Symbol) -> %
--R zeroOf : Polynomial % -> %
--R zerosOf : % -> List %
--R ?~=? : (%,%) -> Boolean
--R ?*? : (R,%) -> % if R has COMRING
?*? : (%,Fraction Integer) -> %
?*? : (Integer,%) -> %
?**? : (%,Fraction Integer) -> %
?**? : (%,PositiveInteger) -> %
?-? : (%,%) -> %
?/? : (%,%) -> %
?<=? : (%,%) -> Boolean
?>? : (%,%) -> Boolean
1 : () -> %
?^? : (%,Integer) -> %
applyQuote : (Symbol,%) -> %
associates? : (%,%) -> Boolean
box : % -> %
coerce : Kernel % -> %
coerce : R -> %
coerce : % -> %
coerce : % -> OutputForm
distribute : (%,%) -> %
elt : (BasicOperator,%,%) -> %
eval : (%,List Equation %) -> %
eval : (%,%,%) -> %
factor : % -> Factored %
freeOf? : (%,Symbol) -> Boolean
gcd : (%,%) -> %
ground? : % -> Boolean
height : % -> NonNegativeInteger
is? : (%,Symbol) -> Boolean
kernels : % -> List Kernel %
lcm : List % -> %
map : ((% -> %),Kernel %) -> %
min : (%,%) -> %
one? : % -> Boolean
paren : List % -> %
?quo? : (%,%) -> %
?rem? : (%,%) -> %
retract : % -> Symbol
rootOf : (%,Symbol) -> %
rootOf : Polynomial % -> %
rootsOf : % -> List %
sample : () -> %
sqrt : % -> %
squareFreePart : % -> %
tower : % -> List Kernel %
unitCanonical : % -> %
zero? : % -> Boolean
zeroOf : % -> %
zerosOf : (%,Symbol) -> List %
zerosOf : Polynomial % -> List %
```

```
--R ?*? : (%,R) -> % if R has COMRING
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (SparseMultivariatePolynomial(R,Kernel %),SparseMultivariatePolynomial(R,Kernel %)) -> % if R has
--R D : (%,List Symbol,List NonNegativeInteger) -> % if R has RING
--R D : (%,Symbol,NonNegativeInteger) -> % if R has RING
--R D : (%,List Symbol) -> % if R has RING
--R D : (%,Symbol) -> % if R has RING
--R ?^? : (%,NonNegativeInteger) -> %
--R applyQuote : (Symbol,%,%,%) -> %
--R applyQuote : (Symbol,%,%,%,%) -> %
--R applyQuote : (Symbol,List %) -> %
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if R has CHARNZ
--R coerce : SparseMultivariatePolynomial(R,Kernel %) -> % if R has RING
--R coerce : Fraction R -> % if R has INTDOM
--R coerce : Polynomial Fraction R -> % if R has INTDOM
--R coerce : Fraction Polynomial Fraction R -> % if R has INTDOM
--R coerce : Fraction Polynomial R -> % if R has INTDOM
--R coerce : Polynomial R -> % if R has RING
--R commutator : (%,%) -> % if R has GROUP
--R conjugate : (%,%) -> % if R has GROUP
--R convert : % -> Pattern Integer if R has KONVERT PATTERN INT
--R convert : % -> Pattern Float if R has KONVERT PATTERN FLOAT
--R convert : Factored % -> % if R has INTDOM
--R convert : % -> InputForm if R has KONVERT INFORM
--R definingPolynomial : % -> % if $ has RING
--R denom : % -> SparseMultivariatePolynomial(R,Kernel %) if R has INTDOM
--R denominator : % -> % if R has INTDOM
--R differentiate : (%,List Symbol,List NonNegativeInteger) -> % if R has RING
--R differentiate : (%,Symbol,NonNegativeInteger) -> % if R has RING
--R differentiate : (%,List Symbol) -> % if R has RING
--R differentiate : (%,Symbol) -> % if R has RING
--R divide : (%,%) -> Record(quotient: %,remainder: %)
--R elt : (BasicOperator,%,%,%) -> %
--R elt : (BasicOperator,%,%,%,%) -> %
--R elt : (BasicOperator,List %) -> %
--R euclideanSize : % -> NonNegativeInteger
--R eval : (%,List Kernel %,List %) -> %
--R eval : (%,List Symbol,List (% -> %)) -> %
--R eval : (%,List Symbol,List (List % -> %)) -> %
--R eval : (%,Symbol,(List % -> %)) -> %
--R eval : (%,Symbol,(% -> %)) -> %
--R eval : (%,List BasicOperator,List (% -> %)) -> %
--R eval : (%,List BasicOperator,List (List % -> %)) -> %
--R eval : (%,BasicOperator,(List % -> %)) -> %
--R eval : (%,BasicOperator,(% -> %)) -> %
--R eval : (%,Symbol) -> % if R has KONVERT INFORM
--R eval : (%,List Symbol) -> % if R has KONVERT INFORM
--R eval : % -> % if R has KONVERT INFORM
```

```
--R eval : (%,BasicOperator,%,Symbol) -> % if R has KONVERT INFORM
--R eval : (%,List BasicOperator,List %,Symbol) -> % if R has KONVERT INFORM
--R eval : (%,List Symbol,List NonNegativeInteger,List (% -> %)) -> % if R has RING
--R eval : (%,List Symbol,List NonNegativeInteger,List (List % -> %)) -> % if R has RING
--R eval : (%,Symbol,NonNegativeInteger,(List % -> %)) -> % if R has RING
--R eval : (%,Symbol,NonNegativeInteger,(% -> %)) -> % if R has RING
--R even? : % -> Boolean if $ has RETRACT INT
--R expressIdealMember : (List %,%) -> Union(List %,"failed")
--R exquo : (%,%) -> Union(%,"failed")
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivari
--R is? : (%,BasicOperator) -> Boolean
--R isExpt : % -> Union(Record(var: Kernel %,exponent: Integer),"failed") if R has SGROUP
--R isExpt : (%,BasicOperator) -> Union(Record(var: Kernel %,exponent: Integer),"failed") if R h
--R isExpt : (%,Symbol) -> Union(Record(var: Kernel %,exponent: Integer),"failed") if R has RING
--R isMult : % -> Union(Record(coef: Integer,var: Kernel %),"failed") if R has ABELSG
--R isPlus : % -> Union(List %,"failed") if R has ABELSG
--R isPower : % -> Union(Record(val: %,exponent: Integer),"failed") if R has RING
--R isTimes : % -> Union(List %,"failed") if R has SGROUP
--R kernel : (BasicOperator,List %) -> %
--R mainKernel : % -> Union(Kernel %,"failed")
--R minPoly : Kernel % -> SparseUnivariatePolynomial % if $ has RING
--R multiEuclidean : (List %,%) -> Union(List %,"failed")
--R numer : % -> SparseMultivariatePolynomial(R,Kernel %) if R has RING
--R numerator : % -> % if R has RING
--R odd? : % -> Boolean if $ has RETRACT INT
--R operator : BasicOperator -> BasicOperator
--R operators : % -> List BasicOperator
--R patternMatch : (%,Pattern Integer,PatternMatchResult(Integer,%)) -> PatternMatchResult(Integ
--R patternMatch : (%,Pattern Float,PatternMatchResult(Float,%)) -> PatternMatchResult(Float,%)
--R principalIdeal : List % -> Record(coef: List %,generator: %)
--R reducedSystem : Matrix % -> Matrix Integer if and(has(R,Ring),has(R,LinearlyExplicitRingOver
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer) if an
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix R,vec: Vector R) if R has RING
--R reducedSystem : Matrix % -> Matrix R if R has RING
--R retract : % -> Fraction Integer if R has RETRACT INT and R has INTDOM or R has RETRACT FRAC
--R retract : % -> Integer if R has RETRACT INT
--R retract : % -> Fraction Polynomial R if R has INTDOM
--R retract : % -> Polynomial R if R has RING
--R retractIfCan : % -> Union(Fraction Integer,"failed") if R has RETRACT INT and R has INTDOM o
--R retractIfCan : % -> Union(Kernel %,"failed")
--R retractIfCan : % -> Union(Symbol,"failed")
--R retractIfCan : % -> Union(Integer,"failed") if R has RETRACT INT
--R retractIfCan : % -> Union(R,"failed")
--R retractIfCan : % -> Union(Fraction Polynomial R,"failed") if R has INTDOM
--R retractIfCan : % -> Union(Polynomial R,"failed") if R has RING
--R rootOf : (SparseUnivariatePolynomial %,Symbol) -> %
--R rootOf : SparseUnivariatePolynomial % -> %
--R rootsOf : (SparseUnivariatePolynomial %,Symbol) -> List %
```

```
--R rootsOf : SparseUnivariatePolynomial % -> List %
--R subst : (%,List Equation %) -> %
--R subst : (%,List Kernel %,List %) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R univariate : (%,Kernel %) -> Fraction SparseUnivariatePolynomial % if R has INTDOM
--R zeroOf : (SparseUnivariatePolynomial %,Symbol) -> %
--R zeroOf : SparseUnivariatePolynomial % -> %
--R zerosOf : (SparseUnivariatePolynomial %,Symbol) -> List %
--R zerosOf : SparseUnivariatePolynomial % -> List %
--R
--E 1

)spool
)lisp (bye)
```

———-

— **AlgebraicallyClosedFunctionSpace.help** —

```
=====================================================================
AlgebraicallyClosedFunctionSpace examples
=====================================================================

See Also:
o )show AlgebraicallyClosedFunctionSpace
```

———-

See:

⇐ “AlgebraicallyClosedField” (ACF) 17.0.191 on page 1293
⇐ “FunctionSpace” (FS) 17.0.195 on page 1336

Exports:

0	1	applyQuote	associates?
belong?	box	characteristic	charthRoot
coerce	commutator	conjugate	convert
D	definingPolynomial	denom	denominator
differentiate	distribute	divide	elt
euclideanSize	eval	even?	expressIdealMember
exquo	extendedEuclidean	factor	freeOf?
gcd	gcdPolynomial	ground	ground?
hash	height	inv	is?
isExpt	isMult	isPlus	isPower
isTimes	kernel	kernels	latex
lcm	mainKernel	map	max
min	minPoly	multiEuclidean	nthRoot
numer	numerator	odd?	one?
operator	operators	paren	patternMatch
prime?	principalIdeal	recip	reducedSystem
retract	retractIfCan	rootOf	rootsOf
sample	sizeLess?	sqrt	squareFree
squareFreePart	subst	subtractIfCan	tower
unit?	unitCanonical	unitNormal	univariate
variables	zero?	zeroOf	zerosOf
?*?	?**?	?+?	?-?
-?	?/?	?<?	?<=?
?=?	?>?	?>=?	?^?
?~=?	?quo?	?rem?	

Attributes Exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.
- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **nil**

These are implemented by this category:

```
rootOf : % -> %
rootOf : (%,Symbol) -> %
rootsOf : % -> List %
rootsOf : (%,Symbol) -> List %
rootsOf : (SparseUnivariatePolynomial %,Symbol) -> List %
zeroOf : % -> %
zeroOf : (%,Symbol) -> %
zeroOf : (SparseUnivariatePolynomial %,Symbol) -> %
zerosOf : % -> List %
zerosOf : (%,Symbol) -> List %
zerosOf : (SparseUnivariatePolynomial %,Symbol) -> List %
```

These exports come from (p1293) AlgebraicallyClosedField():

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean
characteristic : () -> NonNegativeInteger
coerce : % -> %
coerce : Integer -> %
coerce : % -> OutputForm
coerce : Fraction Integer -> %
divide : (%,%) -> Record(quotient: %,remainder: %)
euclideanSize : % -> NonNegativeInteger
expressIdealMember : (List %,%) -> Union(List %,"failed")
exquo : (%,%) -> Union(%,"failed")
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
factor : % -> Factored %
gcd : List % -> %
gcd : (%,%) -> %
gcdPolynomial :
  (SparseUnivariatePolynomial %,
   SparseUnivariatePolynomial %) ->
     SparseUnivariatePolynomial %
hash : % -> SingleInteger
inv : % -> %
latex : % -> String
lcm : List % -> %
lcm : (%,%) -> %
multiEuclidean : (List %,%) -> Union(List %,"failed")
nthRoot : (%,Integer) -> %
one? : % -> Boolean
prime? : % -> Boolean
principalIdeal : List % -> Record(coef: List %,generator: %)
recip : % -> Union(%,"failed")
rootOf : SparseUnivariatePolynomial % -> %
rootOf : Polynomial % -> %
```

```
rootOf : (SparseUnivariatePolynomial %,Symbol) -> %
rootsOf : SparseUnivariatePolynomial % -> List %
rootsOf : Polynomial % -> List %
sample : () -> %
sizeLess? : (%,%) -> Boolean
sqrt : % -> %
squareFree : % -> Factored %
squareFreePart : % -> %
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
zero? : % -> Boolean
zeroOf : Polynomial % -> %
zeroOf : SparseUnivariatePolynomial % -> %
zerosOf : Polynomial % -> List %
zerosOf : SparseUnivariatePolynomial % -> List %
?*? : (Fraction Integer,%) -> %
?*? : (%,Fraction Integer) -> %
?**? : (%,Fraction Integer) -> %
?**? : (%,Integer) -> %
?^? : (%,Integer) -> %
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?/? : (%,%) -> %
?quo? : (%,%) -> %
?rem? : (%,%) -> %
```

These exports come from (p1336) FunctionSpace(R)
where R:Join(OrderedSet, IntegralDomain)):

```
applyQuote : (Symbol,%) -> %
applyQuote : (Symbol,%,%) -> %
applyQuote : (Symbol,%,%,%) -> %
applyQuote : (Symbol,%,%,%,%) -> %
applyQuote : (Symbol,List %) -> %
belong? : BasicOperator -> Boolean
box : % -> %
box : List % -> %
```

```
charthRoot : % -> Union(%,"failed") if R has CHARNZ
coerce : R -> %
coerce : Symbol -> %
coerce : Kernel % -> %
coerce : Fraction Polynomial R -> % if R has INTDOM
coerce : Fraction Polynomial Fraction R -> % if R has INTDOM
coerce : SparseMultivariatePolynomial(R,Kernel %) -> %
  if R has RING
coerce : Fraction R -> % if R has INTDOM
coerce : Polynomial Fraction R -> % if R has INTDOM
coerce : Polynomial R -> % if R has RING
commutator : (%,%) -> % if R has GROUP
conjugate : (%,%) -> % if R has GROUP
convert : % -> InputForm if R has KONVERT INFORM
convert : % -> Pattern Integer if R has KONVERT PATTERN INT
convert : % -> Pattern Float if R has KONVERT PATTERN FLOAT
convert : Factored % -> % if R has INTDOM
D : (%,List Symbol,List NonNegativeInteger) -> % if R has RING
D : (%,Symbol,NonNegativeInteger) -> % if R has RING
D : (%,List Symbol) -> % if R has RING
D : (%,Symbol) -> % if R has RING
definingPolynomial : % -> % if $ has RING
denom : % -> SparseMultivariatePolynomial(R,Kernel %) if R has INTDOM
denominator : % -> % if R has INTDOM
differentiate : (%,List Symbol,List NonNegativeInteger) -> %
  if R has RING
differentiate : (%,Symbol,NonNegativeInteger) -> % if R has RING
differentiate : (%,List Symbol) -> % if R has RING
distribute : % -> %
distribute : (%,%) -> %
differentiate : (%,Symbol) -> % if R has RING
elt : (BasicOperator,%,%,%) -> %
elt : (BasicOperator,%,%,%,%) -> %
elt : (BasicOperator,%) -> %
elt : (BasicOperator,%,%) -> %
elt : (BasicOperator,List %) -> %
eval : (%,List BasicOperator,List (% -> %)) -> %
eval : (%,List Equation %) -> %
eval : (%,Symbol,(% -> %)) -> %
eval : (%,Symbol,(List % -> %)) -> %
eval : (%,BasicOperator,%,Symbol) -> % if R has KONVERT INFORM
eval : (%,BasicOperator,(List % -> %)) -> %
eval : (%,BasicOperator,(% -> %)) -> %
eval : (%,List Symbol,List (% -> %)) -> %
eval : (%,List BasicOperator,List (List % -> %)) -> %
eval : (%,List Symbol,List (List % -> %)) -> %
eval : (%,List %,List %) -> %
eval : (%,%,%) -> %
eval : (%,Equation %) -> %
eval : (%,Kernel %,%) -> %
```

```
eval : (%,List Symbol) -> % if R has KONVERT INFORM
eval : % -> % if R has KONVERT INFORM
eval : (%,Symbol) -> % if R has KONVERT INFORM
eval : (%,Symbol,NonNegativeInteger,(% -> %)) -> % if R has RING
eval : (%,Symbol,NonNegativeInteger,(List % -> %)) -> % if R has RING
eval :
  (%,List Symbol,List NonNegativeInteger,List (List % -> %)) -> %
    if R has RING
eval :
  (%,List Symbol,List NonNegativeInteger,List (% -> %)) -> %
    if R has RING
eval : (%,List Kernel %,List %) -> %
eval : (%,List BasicOperator,List %,Symbol) -> %
  if R has KONVERT INFORM
even? : % -> Boolean if $ has RETRACT INT
freeOf? : (%,%) -> Boolean
freeOf? : (%,Symbol) -> Boolean
ground : % -> R
ground? : % -> Boolean
height : % -> NonNegativeInteger
is? : (%,BasicOperator) -> Boolean
is? : (%,Symbol) -> Boolean
isExpt : % ->
  Union(Record(var: Kernel %,exponent: Integer),"failed")
    if R has SGROUP
isExpt :
  (%,BasicOperator) ->
    Union(Record(var: Kernel %,exponent: Integer),"failed")
      if R has RING
isExpt :
  (%,Symbol) ->
    Union(Record(var: Kernel %,exponent: Integer),"failed")
      if R has RING
isMult : % ->
  Union(Record(coef: Integer,var: Kernel %),"failed")
    if R has ABELSG
isPlus : % -> Union(List %,"failed") if R has ABELSG
isPower : % -> Union(Record(val: %,exponent: Integer),"failed")
    if R has RING
isTimes : % -> Union(List %,"failed") if R has SGROUP
kernel : (BasicOperator,List %) -> %
kernel : (BasicOperator,%) -> %
kernels : % -> List Kernel %
mainKernel : % -> Union(Kernel %,"failed")
map : ((% -> %),Kernel %) -> %
max : (%,%) -> %
min : (%,%) -> %
minPoly : Kernel % -> SparseUnivariatePolynomial % if $ has RING
numer : % -> SparseMultivariatePolynomial(R,Kernel %) if R has RING
numerator : % -> % if R has RING
```

```
odd? : % -> Boolean if $ has RETRACT INT
operator : BasicOperator -> BasicOperator
operators : % -> List BasicOperator
paren : % -> %
paren : List % -> %
patternMatch :
  (%,Pattern Integer,PatternMatchResult(Integer,%)) ->
     PatternMatchResult(Integer,%)
       if R has PATMAB INT
patternMatch :
  (%,Pattern Float,PatternMatchResult(Float,%)) ->
     PatternMatchResult(Float,%)
       if R has PATMAB FLOAT
reducedSystem : Matrix % -> Matrix Integer
  if and(has(R,Ring),has(R,LinearlyExplicitRingOver Integer))
  or and(has(R,LinearlyExplicitRingOver Integer),has(R,Ring))
reducedSystem :
 (Matrix %,Vector %) ->
   Record(mat: Matrix Integer,vec: Vector Integer)
     if and(has(R,Ring),has(R,LinearlyExplicitRingOver Integer))
     or and(has(R,LinearlyExplicitRingOver Integer),has(R,Ring))
reducedSystem :
 (Matrix %,Vector %) -> Record(mat: Matrix R,vec: Vector R)
   if R has RING
reducedSystem : Matrix % -> Matrix R if R has RING
retract : % -> Kernel %
retract : % -> Fraction Polynomial R if R has INTDOM
retract : % -> Polynomial R if R has RING
retract : % -> R
retract : % -> Symbol
retract : % -> Integer if R has RETRACT INT
retract : % -> Fraction Integer
  if R has RETRACT INT
  and R has INTDOM
  or R has RETRACT FRAC INT
retractIfCan : % -> Union(Integer,"failed") if R has RETRACT INT
retractIfCan : % -> Union(Fraction Integer,"failed")
  if R has RETRACT INT
  and R has INTDOM
  or R has RETRACT FRAC INT
retractIfCan : % -> Union(Kernel %,"failed")
retractIfCan : % -> Union(R,"failed")
retractIfCan : % -> Union(Fraction Polynomial R,"failed")
  if R has INTDOM
retractIfCan : % -> Union(Polynomial R,"failed") if R has RING
retractIfCan : % -> Union(Symbol,"failed")
subst : (%,List Kernel %,List %) -> %
subst : (%,List Equation %) -> %
subst : (%,Equation %) -> %
tower : % -> List Kernel %
```

```
 univariate : (%,Kernel %) -> Fraction SparseUnivariatePolynomial %
   if R has INTDOM
 variables : % -> List Symbol
 ?<? : (%,%) -> Boolean
 ?<=? : (%,%) -> Boolean
 ?>? : (%,%) -> Boolean
 ?>=? : (%,%) -> Boolean
 ?*? : (R,%) -> % if R has COMRING
 ?*? : (%,R) -> % if R has COMRING
 ?/? :
  (SparseMultivariatePolynomial(R,Kernel %),
   SparseMultivariatePolynomial(R,Kernel %)) -> %
     if R has INTDOM
```

— category ACFS AlgebraicallyClosedFunctionSpace —

```
)abbrev category ACFS AlgebraicallyClosedFunctionSpace
++ Author: Manuel Bronstein
++ Date Created: 31 October 1988
++ Date Last Updated: 7 October 1991
++ Keywords: algebraic, closure, field.
++ Description:
++ Model for algebraically closed function spaces.

AlgebraicallyClosedFunctionSpace(R:Join(OrderedSet, IntegralDomain)):
 Category == Join(AlgebraicallyClosedField, FunctionSpace R) with
   rootOf : $ -> $
     ++ rootOf(p) returns y such that \spad{p(y) = 0}.
     ++ Error: if p has more than one variable y.
   rootsOf: $ -> List $
     ++ rootsOf(p, y) returns \spad{[y1,...,yn]} such that \spad{p(yi) = 0};
     ++ Note that the returned symbols y1,...,yn are bound in the interpreter
     ++ to respective root values.
     ++ Error: if p has more than one variable y.
   rootOf : ($, Symbol) -> $
     ++ rootOf(p,y) returns y such that \spad{p(y) = 0}.
     ++ The object returned displays as \spad{'y}.
   rootsOf: ($, Symbol) -> List $
     ++ rootsOf(p, y) returns \spad{[y1,...,yn]} such that \spad{p(yi) = 0};
     ++ The returned roots display as \spad{'y1},...,\spad{'yn}.
     ++ Note that the returned symbols y1,...,yn are bound in the interpreter
     ++ to respective root values.
   zeroOf : $ -> $
     ++ zeroOf(p) returns y such that \spad{p(y) = 0}.
     ++ The value y is expressed in terms of radicals if possible,and otherwise
     ++ as an implicit algebraic quantity.
     ++ Error: if p has more than one variable.
   zerosOf: $ -> List $
     ++ zerosOf(p) returns \spad{[y1,...,yn]} such that \spad{p(yi) = 0}.
```

```
   ++ The yi's are expressed in radicals if possible.
   ++ The returned symbols y1,...,yn are bound in the interpreter
   ++ to respective root values.
   ++ Error: if p has more than one variable.
 zeroOf : ($, Symbol) -> $
   ++ zeroOf(p, y) returns y such that \spad{p(y) = 0}.
   ++ The value y is expressed in terms of radicals if possible,and otherwise
   ++ as an implicit algebraic quantity
   ++ which displays as \spad{'y}.
 zerosOf: ($, Symbol) -> List $
   ++ zerosOf(p, y) returns \spad{[y1,...,yn]} such that \spad{p(yi) = 0}.
   ++ The yi's are expressed in radicals if possible, and otherwise
   ++ as implicit algebraic quantities
   ++ which display as \spad{'yi}.
   ++ The returned symbols y1,...,yn are bound in the interpreter
   ++ to respective root values.
add
  rootOf(p:$) ==
    empty?(l := variables p) => error "rootOf: constant expression"
    rootOf(p, first l)

  rootsOf(p:$) ==
    empty?(l := variables p) => error "rootsOf: constant expression"
    rootsOf(p, first l)

  zeroOf(p:$) ==
    empty?(l := variables p) => error "zeroOf: constant expression"
    zeroOf(p, first l)

  zerosOf(p:$) ==
    empty?(l := variables p) => error "zerosOf: constant expression"
    zerosOf(p, first l)

  zeroOf(p:$, x:Symbol) ==
    n := numer(f := univariate(p, kernel(x)$Kernel($)))
    degree denom f > 0 => error "zeroOf: variable appears in denom"
    degree n = 0 => error "zeroOf: constant expression"
    zeroOf(n, x)

  rootOf(p:$, x:Symbol) ==
    n := numer(f := univariate(p, kernel(x)$Kernel($)))
    degree denom f > 0 => error "roofOf: variable appears in denom"
    degree n = 0 => error "rootOf: constant expression"
    rootOf(n, x)

  zerosOf(p:$, x:Symbol) ==
    n := numer(f := univariate(p, kernel(x)$Kernel($)))
    degree denom f > 0 => error "zerosOf: variable appears in denom"
    degree n = 0 => empty()
    zerosOf(n, x)
```

```
    rootsOf(p:$, x:Symbol) ==
      n := numer(f := univariate(p, kernel(x)$Kernel($)))
      degree denom f > 0 => error "roofsOf: variable appears in denom"
      degree n = 0 => empty()
      rootsOf(n, x)

    rootsOf(p:SparseUnivariatePolynomial $, y:Symbol) ==
      (r := retractIfCan(p)@Union($,"failed")) case $ => rootsOf(r::$,y)
      rootsOf(p, y)$AlgebraicallyClosedField_&($)

    zerosOf(p:SparseUnivariatePolynomial $, y:Symbol) ==
      (r := retractIfCan(p)@Union($,"failed")) case $ => zerosOf(r::$,y)
      zerosOf(p, y)$AlgebraicallyClosedField_&($)

    zeroOf(p:SparseUnivariatePolynomial $, y:Symbol) ==
      (r := retractIfCan(p)@Union($,"failed")) case $ => zeroOf(r::$, y)
      zeroOf(p, y)$AlgebraicallyClosedField_&($)
```

———-

— ACFS.dotabb —

```
"ACFS"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ACFS"];
"ACFS" -> "ACF"
"ACFS" -> "FS"
```

———-

— ACFS.dotfull —

```
"AlgebraicallyClosedFunctionSpace(a:Join(OrderedSet,IntegralDomain))"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ACFS"];
"AlgebraicallyClosedFunctionSpace(a:Join(OrderedSet,IntegralDomain))"
  -> "AlgebraicallyClosedField()"
"AlgebraicallyClosedFunctionSpace(a:Join(OrderedSet,IntegralDomain))"
  -> "FunctionSpace(a:OrderedSet)"
```

———-

— ACFS.dotpic —

```
digraph pic {
 fontsize=10;
```

```
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"AlgebraicallyClosedFunctionSpace(a:Join(OrderedSet,IntegralDomain))"
 [color=lightblue];
"AlgebraicallyClosedFunctionSpace(a:Join(OrderedSet,IntegralDomain))"
  -> "AlgebraicallyClosedField()"
"AlgebraicallyClosedFunctionSpace(a:Join(OrderedSet,IntegralDomain))"
  -> "FunctionSpace(a:OrderedSet)"

"AlgebraicallyClosedField()" [color=lightblue];
"AlgebraicallyClosedField()" -> "Field()"
"AlgebraicallyClosedField()" -> "RadicalCategory()"

"Field()" [color=lightblue];
"Field()" -> "EUCDOM..."
"Field()" -> "UFD..."
"Field()" -> "DIVRING..."

"RadicalCategory()" [color=lightblue];
"RadicalCategory()" -> "Category"

"Category" [color=lightblue];

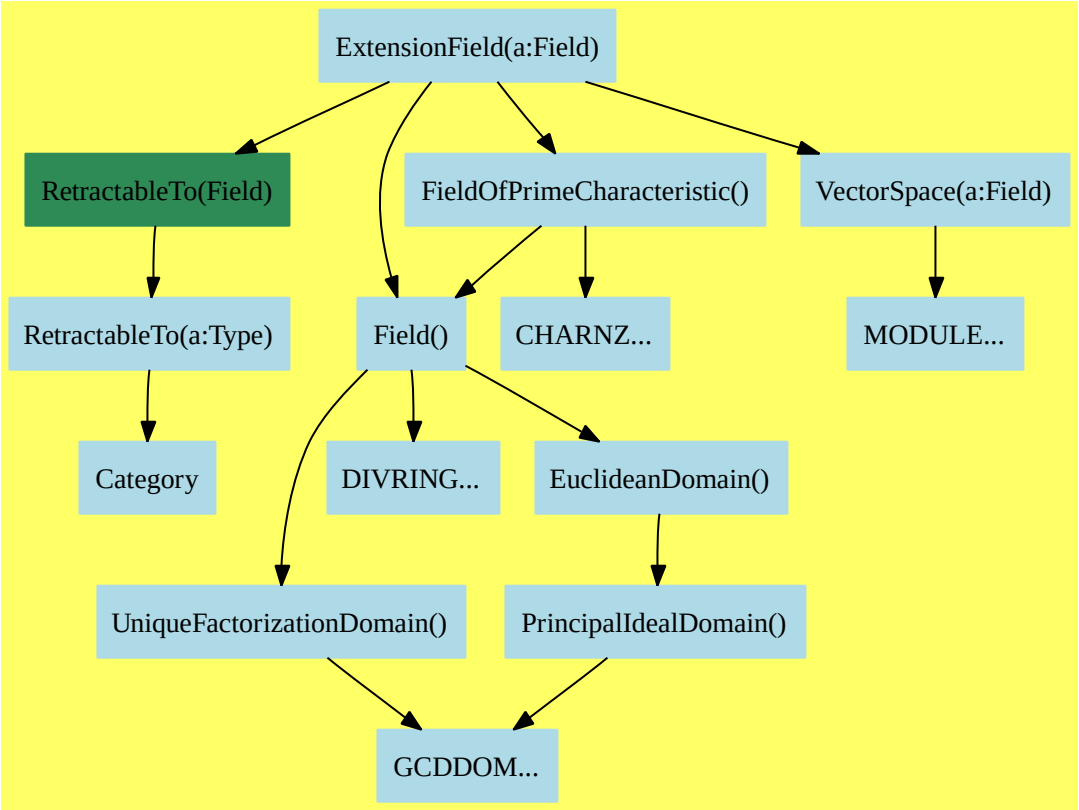
"FunctionSpace(a:OrderedSet)" [color=lightblue];
"FunctionSpace(a:OrderedSet)" -> "ExpressionSpace()"
"FunctionSpace(a:OrderedSet)" -> "RETRACT..."
"FunctionSpace(a:OrderedSet)" -> "PATAB..."
"FunctionSpace(a:OrderedSet)" -> "FPATMAB..."
"FunctionSpace(a:OrderedSet)" -> "FRETRCT..."

"ExpressionSpace()" [color=lightblue];
"ExpressionSpace()" -> "ORDSET..."
"ExpressionSpace()" -> "RETRACT..."
"ExpressionSpace()" -> "IEVALAB..."
"ExpressionSpace()" -> "EVALABLE..."

"UFD..." [color=lightblue];
"EUCDOM..." [color=lightblue];
"DIVRING..." [color=lightblue];
"EVALABLE..." [color=lightblue];
"FRETRCT..." [color=lightblue];
"FPATMAB..." [color=lightblue];
"IEVALAB..." [color=lightblue];
"ORDSET..." [color=lightblue];
"PATAB..." [color=lightblue];
"RETRACT..." [color=lightblue];
}
```

———-

18.0.206 ExtensionField (XF)



— **ExtensionField.input** —

```
)set break resume
)sys rm -f ExtensionField.output
)spool ExtensionField.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show ExtensionField
--R ExtensionField F: Field  is a category constructor
--R Abbreviation for ExtensionField is XF
```

```
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for XF
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (F,%) -> %                      ?*? : (%,F) -> %
--R ?*? : (Fraction Integer,%) -> %       ?*? : (%,Fraction Integer) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,Integer) -> %
--R ?**? : (%,PositiveInteger) -> %       ?+? : (%,%) -> %
--R ?-? : (%,%) -> %                      -? : % -> %
--R ?/? : (%,F) -> %                      ?/? : (%,%) -> %
--R ?=? : (%,%) -> Boolean                1 : () -> %
--R 0 : () -> %                           ?^? : (%,Integer) -> %
--R ?^? : (%,PositiveInteger) -> %        algebraic? : % -> Boolean
--R associates? : (%,%) -> Boolean        coerce : F -> %
--R coerce : Fraction Integer -> %        coerce : % -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R dimension : () -> CardinalNumber      factor : % -> Factored %
--R gcd : List % -> %                     gcd : (%,%) -> %
--R hash : % -> SingleInteger             inGroundField? : % -> Boolean
--R inv : % -> %                          latex : % -> String
--R lcm : List % -> %                     lcm : (%,%) -> %
--R one? : % -> Boolean                   prime? : % -> Boolean
--R ?quo? : (%,%) -> %                    recip : % -> Union(%,"failed")
--R ?rem? : (%,%) -> %                    retract : % -> F
--R sample : () -> %                      sizeLess? : (%,%) -> Boolean
--R squareFree : % -> Factored %          squareFreePart : % -> %
--R transcendent? : % -> Boolean          unit? : % -> Boolean
--R unitCanonical : % -> %                zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R Frobenius : (%,NonNegativeInteger) -> % if F has FINITE
--R Frobenius : % -> % if F has FINITE
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if F has CHARNZ or F has FINITE
--R degree : % -> OnePointCompletion PositiveInteger
--R discreteLog : (%,%) -> Union(NonNegativeInteger,"failed") if F has CHARNZ or F has FINITE
--R divide : (%,%) -> Record(quotient: %,remainder: %)
--R euclideanSize : % -> NonNegativeInteger
--R expressIdealMember : (List %,%) -> Union(List %,"failed")
--R exquo : (%,%) -> Union(%,"failed")
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
--R extensionDegree : () -> OnePointCompletion PositiveInteger
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivariatePolynomia
--R multiEuclidean : (List %,%) -> Union(List %,"failed")
--R order : % -> OnePointCompletion PositiveInteger if F has CHARNZ or F has FINITE
--R primeFrobenius : % -> % if F has CHARNZ or F has FINITE
```

```
--R primeFrobenius : (%,NonNegativeInteger) -> % if F has CHARNZ or F has FINITE
--R principalIdeal : List % -> Record(coef: List %,generator: %)
--R retractIfCan : % -> Union(F,"failed")
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R transcendenceDegree : () -> NonNegativeInteger
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R
--E 1

)spool
)lisp (bye)
```

———-

— **ExtensionField.help** —

```
=====================================================================
ExtensionField examples
=====================================================================

See Also:
o )show ExtensionField
```

———-

See:

⇒ “FiniteAlgebraicExtensionField” (FAXF) 19.0.213 on page 1583
⇐ “Field” (FIELD) 16.0.185 on page 1221
⇐ “RetractableTo” (RETRACT) 2.0.18 on page 55
⇐ “VectorSpace” (VSPACE) 11.0.158 on page 987

Exports:

0	1	algebraic?	associates?
characteristic	charthRoot	coerce	degree
discreteLog	divide	euclideanSize	expressIdealMember
exquo	extendedEuclidean	extensionDegree	dimension
factor	Frobenius	gcd	gcdPolynomial
hash	inGroundField?	inv	latex
lcm	multiEuclidean	one?	order
prime?	primeFrbenius	principalIdeal	recip
retract	retractIfCan	sample	sizeLess?
squareFree	squareFreePart	subtractIfCan	transcendenceDegree
transcendent?	unit?	unitCanonical	unitNormal
zero?	?*?	?**?	?+?
?-?	-?	?/?	?=?
?^?	?quo?	?rem?	?~=?

Attributes Exported:

- **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.
- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
inGroundField? : % -> Boolean
degree : % -> OnePointCompletion PositiveInteger
extensionDegree : () -> OnePointCompletion PositiveInteger
transcendenceDegree : () -> NonNegativeInteger
```

These are implemented by this category:

```
algebraic? : % -> Boolean
Frobenius : % -> % if F has FINITE
Frobenius : (%,NonNegativeInteger) -> % if F has FINITE
transcendent? : % -> Boolean
```

These exports come from (p1221) Field():

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean
characteristic : () -> NonNegativeInteger
coerce : % -> %
coerce : Integer -> %
coerce : % -> OutputForm
coerce : Fraction Integer -> %
divide : (%,%) -> Record(quotient: %,remainder: %)
euclideanSize : % -> NonNegativeInteger
expressIdealMember : (List %,%) -> Union(List %,"failed")
exquo : (%,%) -> Union(%,"failed")
extendedEuclidean : (%,%,%) ->
  Union(Record(coef1: %,coef2: %),"failed")
extendedEuclidean : (%,%) ->
  Record(coef1: %,coef2: %,generator: %)
factor : % -> Factored %
gcd : List % -> %
gcd : (%,%) -> %
gcdPolynomial : (SparseUnivariatePolynomial %,
                 SparseUnivariatePolynomial %) ->
                     SparseUnivariatePolynomial %
hash : % -> SingleInteger
inv : % -> %
latex : % -> String
lcm : List % -> %
lcm : (%,%) -> %
multiEuclidean : (List %,%) -> Union(List %,"failed")
one? : % -> Boolean
prime? : % -> Boolean
principalIdeal : List % -> Record(coef: List %,generator: %)
recip : % -> Union(%,"failed")
sample : () -> %
squareFree : % -> Factored %
squareFreePart : % -> %
sizeLess? : (%,%) -> Boolean
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
zero? : % -> Boolean
?/? : (%,%) -> %
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
```

```
?*? : (NonNegativeInteger,%) -> %
?*? : (Fraction Integer,%) -> %
?*? : (%,Fraction Integer) -> %
?*? : (F,%) -> %
?*? : (%,F) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,Integer) -> %
?^? : (%,Integer) -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?quo? : (%,%) -> %
?rem? : (%,%) -> %
```

These exports come from (p55) RetractableTo(F:Field):

```
coerce : F -> %
retract : % -> F
retractIfCan : % -> Union(F,"failed")
```

These exports come from (p987) VectorSpace(F:Field):

```
dimension : () -> CardinalNumber
?/? : (%,F) -> %
```

These exports come from (p1323) FieldOfPrimeCharacteristic():

```
charthRoot : % -> Union(%,"failed")
    if F has CHARNZ or F has FINITE
discreteLog : (%,%) -> Union(NonNegativeInteger,"failed")
    if F has CHARNZ or F has FINITE
order : % -> OnePointCompletion PositiveInteger
    if F has CHARNZ or F has FINITE
primeFrobenius : % -> %
    if F has CHARNZ or F has FINITE
primeFrobenius : (%,NonNegativeInteger) -> %
    if F has CHARNZ or F has FINITE
```

— category XF ExtensionField —

```
)abbrev category XF ExtensionField
++ Author: J. Grabmeier, A. Scheerhorn
++ Date Created: 10 March 1991
++ Date Last Updated: 31 March 1991
++ Basic Operations: _+, _*, extensionDegree, algebraic?, transcendent?
++ Related Constructors:
```

```
++ Also See:
++ AMS Classifications:
++ Keywords: field, extension field
++ References:
++  J. Grabmeier, A. Scheerhorn: Finite Fields in AXIOM.
++  AXIOM Technical Report Series, ATR/5 NP2522.
++ Description:
++ ExtensionField F is the category of fields which extend the field F

ExtensionField(F:Field) : Category  == _
  Join(Field,RetractableTo F,VectorSpace F) with
    if F has CharacteristicZero then CharacteristicZero
    if F has CharacteristicNonZero then FieldOfPrimeCharacteristic
    algebraic? : $ -> Boolean
      ++ algebraic?(a) tests whether an element \spad{a} is algebraic with
      ++ respect to the ground field F.
    transcendent? : $ -> Boolean
      ++ transcendent?(a) tests whether an element \spad{a} is transcendent
      ++ with respect to the ground field F.
    inGroundField?: $ -> Boolean
      ++ inGroundField?(a) tests whether an element \spad{a}
      ++ is already in the ground field F.
    degree : $ -> OnePointCompletion PositiveInteger
      ++ degree(a) returns the degree of minimal polynomial of an element
      ++ \spad{a} if \spad{a} is algebraic
      ++ with respect to the ground field F, and \spad{infinity} otherwise.
    extensionDegree : () -> OnePointCompletion PositiveInteger
      ++ extensionDegree() returns the degree of the field extension if the
      ++ extension is algebraic, and \spad{infinity} if it is not.
    transcendenceDegree : () -> NonNegativeInteger
      ++ transcendenceDegree() returns the transcendence degree of the
      ++ field extension, 0 if the extension is algebraic.
    -- perhaps more absolute degree functions
    if F has Finite then
      FieldOfPrimeCharacteristic
      Frobenius: $ -> $
        ++ Frobenius(a) returns \spad{a ** q} where q is the \spad{size()$F}.
      Frobenius:   ($,NonNegativeInteger) -> $
        ++ Frobenius(a,s) returns \spad{a**(q**s)} where q is the size()$F.
  add
    algebraic?(a) == not infinite? (degree(a)@OnePointCompletion_
      (PositiveInteger))$OnePointCompletion(PositiveInteger)
    transcendent? a == infinite?(degree(a)@OnePointCompletion _
      (PositiveInteger))$OnePointCompletion(PositiveInteger)
    if F has Finite then
      Frobenius(a) == a ** size()$F
      Frobenius(a,s) == a ** (size()$F ** s)
```

— **XF.dotabb** —

```
"XF"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=XF"];
"XF" -> "FIELD"
"XF" -> "RETRACT"
"XF" -> "VSPACE"
"XF" -> "FPC"
```

———-

— **XF.dotfull** —

```
"ExtensionField(a:Field)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=XF"];
"ExtensionField(a:Field)" -> "Field()"
"ExtensionField(a:Field)" -> "RetractableTo(Field)"
"ExtensionField(a:Field)" -> "VectorSpace(a:Field)"
"ExtensionField(a:Field)" -> "FieldOfPrimeCharacteristic()"
```

———-

— **XF.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"ExtensionField(a:Field)" [color=lightblue];
"ExtensionField(a:Field)" -> "Field()"
"ExtensionField(a:Field)" -> "RetractableTo(Field)"
"ExtensionField(a:Field)" -> "VectorSpace(a:Field)"

"FieldOfPrimeCharacteristic()" [color=lightblue];
"FieldOfPrimeCharacteristic()" -> "CHARNZ..."
"FieldOfPrimeCharacteristic()" -> "Field()"

"Field()" [color=lightblue];
"Field()" -> "EuclideanDomain()"
"Field()" -> "UniqueFactorizationDomain()"
"Field()" -> "DIVRING..."

"EuclideanDomain()" [color=lightblue];
"EuclideanDomain()" -> "PrincipalIdealDomain()"
```

```
"UniqueFactorizationDomain()" [color=lightblue];
"UniqueFactorizationDomain()" -> "GCDDOM..."

"PrincipalIdealDomain()" [color=lightblue];
"PrincipalIdealDomain()" -> "GCDDOM..."

"RetractableTo(Field)" [color=seagreen];
"RetractableTo(Field)" -> "RetractableTo(a:Type)"

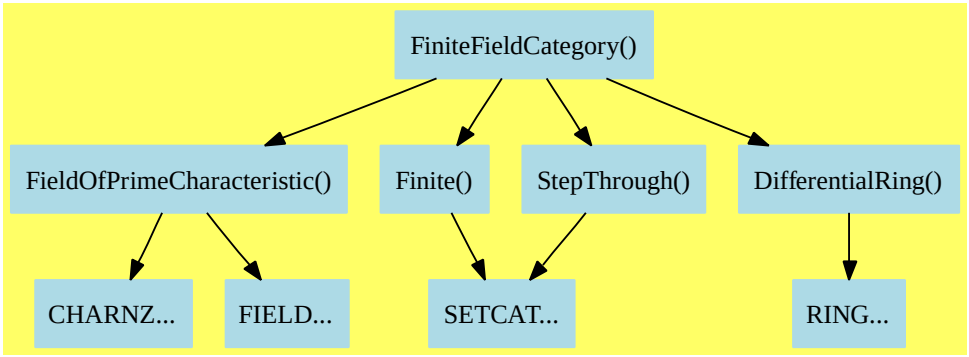
"RetractableTo(a:Type)" [color=lightblue];
"RetractableTo(a:Type)" -> "Category"

"VectorSpace(a:Field)" [color=lightblue];
"VectorSpace(a:Field)" -> "MODULE..."

"MODULE..." [color=lightblue];
"DIVRING..." [color=lightblue];
"GCDDOM..." [color=lightblue];
"CHARNZ..." [color=lightblue];
"Category" [color=lightblue];
}
```

———-

18.0.207 FiniteFieldCategory (FFIELDC)



— FiniteFieldCategory.input —

```
)set break resume
)sys rm -f FiniteFieldCategory.output
```

```
)spool FiniteFieldCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FiniteFieldCategory
--R FiniteFieldCategory  is a category constructor
--R Abbreviation for FiniteFieldCategory is FFIELDC
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FFIELDC
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (Fraction Integer,%) -> %       ?*? : (%,Fraction Integer) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,Integer) -> %
--R ?**? : (%,PositiveInteger) -> %       ?+? : (%,%) -> %
--R ?-? : (%,%) -> %                      -? : % -> %
--R ?/? : (%,%) -> %                      ?=? : (%,%) -> Boolean
--R D : % -> %                            D : (%,NonNegativeInteger) -> %
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,Integer) -> %                ?^? : (%,PositiveInteger) -> %
--R associates? : (%,%) -> Boolean        charthRoot : % -> %
--R coerce : Fraction Integer -> %        coerce : % -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R createPrimitiveElement : () -> %      differentiate : % -> %
--R factor : % -> Factored %              gcd : List % -> %
--R gcd : (%,%) -> %                      hash : % -> SingleInteger
--R index : PositiveInteger -> %          init : () -> %
--R inv : % -> %                          latex : % -> String
--R lcm : List % -> %                     lcm : (%,%) -> %
--R lookup : % -> PositiveInteger         one? : % -> Boolean
--R order : % -> PositiveInteger          prime? : % -> Boolean
--R primeFrobenius : % -> %               primitive? : % -> Boolean
--R primitiveElement : () -> %            ?quo? : (%,%) -> %
--R random : () -> %                      recip : % -> Union(%,"failed")
--R ?rem? : (%,%) -> %                    sample : () -> %
--R size : () -> NonNegativeInteger       sizeLess? : (%,%) -> Boolean
--R squareFree : % -> Factored %          squareFreePart : % -> %
--R unit? : % -> Boolean                  unitCanonical : % -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed")
--R conditionP : Matrix % -> Union(Vector %,"failed")
--R differentiate : (%,NonNegativeInteger) -> %
--R discreteLog : % -> NonNegativeInteger
--R discreteLog : (%,%) -> Union(NonNegativeInteger,"failed")
```

```
--R divide : (%,%) -> Record(quotient: %,remainder: %)
--R euclideanSize : % -> NonNegativeInteger
--R expressIdealMember : (List %,%) -> Union(List %,"failed")
--R exquo : (%,%) -> Union(%,"failed")
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
--R factorsOfCyclicGroupSize : () -> List Record(factor: Integer,exponent: Integer)
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivari
--R multiEuclidean : (List %,%) -> Union(List %,"failed")
--R nextItem : % -> Union(%,"failed")
--R order : % -> OnePointCompletion PositiveInteger
--R primeFrobenius : (%,NonNegativeInteger) -> %
--R principalIdeal : List % -> Record(coef: List %,generator: %)
--R representationType : () -> Union("prime",polynomial,normal,cyclic)
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R tableForDiscreteLogarithm : Integer -> Table(PositiveInteger,NonNegativeInteger)
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R
--E 1

)spool
)lisp (bye)
```

———-

— FiniteFieldCategory.help —

```
=====================================================================
FiniteFieldCategory examples
=====================================================================

See Also:
o )show FiniteFieldCategory
```

———-

See:

⇐ “DifferentialRing” (DIFRING) 10.0.138 on page 848
⇐ “FieldOfPrimeCharacteristic” (FPC) 17.0.193 on page 1323
⇐ “Finite” (FINITE) 4.0.46 on page 168
⇐ “StepThrough” (STEP) 4.0.64 on page 252

Exports:

0	1	associates?
characteristic	charthRoot	coerce
conditionP	createPrimitiveElement	D
differentiate	discreteLog	divide
euclideanSize	expressIdealMember	exquo
extendedEuclidean	factor	factorsOfCyclicGroupSize
gcd	gcdPolynomial	hash
index	init	inv
latex	lcm	lookup
multiEuclidean	nextItem	one?
order	prime?	primeFrobenius
primitive?	primitiveElement	principalIdeal
random	recip	representationType
sample	size	sizeLess?
squareFree	squareFreePart	subtractIfCan
tableForDiscreteLogarithm	unit?	unitCanonical
unitNormal	zero?	?*?
?**?	?+?	?-?
-?	?/?	?=?
?^?	?quo?	?rem?
?~=?		

Attributes Exported:

- **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.
- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
factorsOfCyclicGroupSize : () ->
   List Record(factor: Integer,exponent: Integer)
primitiveElement : () -> %
representationType : () -> Union("prime",polynomial,normal,cyclic)
```

```
tableForDiscreteLogarithm : Integer ->
   Table(PositiveInteger,NonNegativeInteger)
```

These are implemented by this category:

```
charthRoot : % -> %
charthRoot : % -> Union(%,"failed")
conditionP : Matrix % -> Union(Vector %,"failed")
createPrimitiveElement : () -> %
differentiate : % -> %
discreteLog : % -> NonNegativeInteger
discreteLog : (%,%) -> Union(NonNegativeInteger,"failed")
gcdPolynomial : (SparseUnivariatePolynomial %,
                 SparseUnivariatePolynomial %) ->
                     SparseUnivariatePolynomial %
init : () -> %
nextItem : % -> Union(%,"failed")
order : % -> OnePointCompletion PositiveInteger
order : % -> PositiveInteger
primitive? : % -> Boolean
```

These exports come from (p1323) FieldOfPrimeCharacteristic():

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean
characteristic : () -> NonNegativeInteger
coerce : % -> %
coerce : Integer -> %
coerce : % -> OutputForm
coerce : Fraction Integer -> %
divide : (%,%) -> Record(quotient: %,remainder: %)
euclideanSize : % -> NonNegativeInteger
expressIdealMember : (List %,%) -> Union(List %,"failed")
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
exquo : (%,%) -> Union(%,"failed")
factor : % -> Factored %
gcd : List % -> %
gcd : (%,%) -> %
hash : % -> SingleInteger
inv : % -> %
latex : % -> String
lcm : List % -> %
lcm : (%,%) -> %
multiEuclidean : (List %,%) -> Union(List %,"failed")
one? : % -> Boolean
prime? : % -> Boolean
primeFrobenius : % -> %
primeFrobenius : (%,NonNegativeInteger) -> %
```

```
principalIdeal : List % -> Record(coef: List %,generator: %)
recip : % -> Union(%,"failed")
sample : () -> %
sizeLess? : (%,%) -> Boolean
squareFree : % -> Factored %
squareFreePart : % -> %
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (Fraction Integer,%) -> %
?*? : (%,Fraction Integer) -> %
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,Integer) -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?^? : (%,Integer) -> %
?/? : (%,%) -> %
?quo? : (%,%) -> %
?rem? : (%,%) -> %
```

These exports come from (p168) Finite():

```
index : PositiveInteger -> %
lookup : % -> PositiveInteger
random : () -> %
size : () -> NonNegativeInteger
```

These exports come from (p252) StepThrough():

These exports come from (p848) DifferentialRing():

```
D : % -> %
D : (%,NonNegativeInteger) -> %
differentiate : (%,NonNegativeInteger) -> %
```

— category FFIELDC FiniteFieldCategory —

```
)abbrev category FFIELDC FiniteFieldCategory
++ Author: J. Grabmeier, A. Scheerhorn
++ Date Created: 11 March 1991
++ Date Last Updated: 31 March 1991
++ Basic Operations: _+, _*, extensionDegree, order, primitiveElement
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords: field, extension field, algebraic extension, finite field
++  Galois field
++ References:
++  D.Lipson, Elements of Algebra and Algebraic Computing, The
++  Benjamin/Cummings Publishing Company, Inc.-Menlo Park, California, 1981.
++  J. Grabmeier, A. Scheerhorn: Finite Fields in AXIOM.
++  AXIOM Technical Report Series, ATR/5 NP2522.
++ Description:
++ FiniteFieldCategory is the category of finite fields

FiniteFieldCategory() : Category ==_
  Join(FieldOfPrimeCharacteristic,Finite,StepThrough,DifferentialRing) with
--                ,PolynomialFactorizationExplicit) with
    charthRoot: $ -> $
      ++ charthRoot(a) takes the characteristic'th root of a.
      ++ Note that such a root is alway defined in finite fields.
    conditionP: Matrix $ -> Union(Vector $,"failed")
      ++ conditionP(mat), given a matrix representing a homogeneous system
      ++ of equations, returns a vector whose characteristic'th powers
      ++ is a non-trivial solution, or "failed" if no such vector exists.
    -- the reason for implementing the following function is that we
    -- can implement the functions order, getGenerator and primitive? on
    -- category level without computing the, may be time intensive,
    -- factorization of size()-1 at every function call again.
    factorsOfCyclicGroupSize:_
      () -> List Record(factor:Integer,exponent:Integer)
      ++ factorsOfCyclicGroupSize() returns the factorization of size()-1
    -- the reason for implementing the function tableForDiscreteLogarithm
    -- is that we can implement the functions discreteLog and
    -- shanksDiscLogAlgorithm on category level
    -- computing the necessary exponentiation tables in the respective
    -- domains once and for all
    -- absoluteDegree : $ -> PositiveInteger
    --  ++ degree of minimal polynomial, if algebraic with respect
    --  ++ to the prime subfield
    tableForDiscreteLogarithm: Integer -> _
             Table(PositiveInteger,NonNegativeInteger)
      ++ tableForDiscreteLogarithm(a,n) returns a table of the discrete
      ++ logarithms of \spad{a**0} up to \spad{a**(n-1)} which, called with
      ++ key \spad{lookup(a**i)} returns i for i in \spad{0..n-1}.
      ++ Error: if not called for prime divisors of order of
      ++        multiplicative group.
```

```
  createPrimitiveElement: () -> $
    ++ createPrimitiveElement() computes a generator of the (cyclic)
    ++ multiplicative group of the field.
    -- RDJ: Are these next lines to be included?
    -- we run through the field and test, algorithms which construct
    -- elements of larger order were found to be too slow
  primitiveElement: () -> $
    ++ primitiveElement() returns a primitive element stored in a global
    ++ variable in the domain.
    ++ At first call, the primitive element is computed
    ++ by calling \spadfun{createPrimitiveElement}.
  primitive?: $ -> Boolean
    ++ primitive?(b) tests whether the element b is a generator of the
    ++ (cyclic) multiplicative group of the field, i.e. is a primitive
    ++ element.
    ++ Implementation Note that see ch.IX.1.3, th.2 in D. Lipson.
  discreteLog: $ -> NonNegativeInteger
    ++ discreteLog(a) computes the discrete logarithm of \spad{a}
    ++ with respect to \spad{primitiveElement()} of the field.
  order: $ -> PositiveInteger
    ++ order(b) computes the order of an element b in the multiplicative
    ++ group of the field.
    ++ Error: if b equals 0.
  representationType: () -> Union("prime","polynomial","normal","cyclic")
    ++ representationType() returns the type of the representation, one of:
    ++ \spad{prime}, \spad{polynomial}, \spad{normal}, or \spad{cyclic}.
add
  I   ==> Integer
  PI  ==> PositiveInteger
  NNI ==> NonNegativeInteger
  SUP ==> SparseUnivariatePolynomial
  DLP ==> DiscreteLogarithmPackage

  -- exported functions

  differentiate x          == 0

  init() == 0

  nextItem(a) ==
    zero?(a:=index(lookup(a)+1)) => "failed"
    a

  order(e):OnePointCompletion(PositiveInteger) ==
    (order(e)@PI)::OnePointCompletion(PositiveInteger)

  conditionP(mat:Matrix $) ==
    l:=nullSpace mat
    empty? l or every?(zero?, first l) => "failed"
    map(charthRoot,first l)
```

```
    charthRoot(x:$):$ == x**(size() quo characteristic())

    charthRoot(x:%):Union($,"failed") ==
        (charthRoot(x)@$)::Union($,"failed")

    createPrimitiveElement() ==
      sm1  : PositiveInteger := (size()$$-1) pretend PositiveInteger
      start : Integer :=
        -- in the polynomial case, index from 1 to characteristic-1
        -- gives prime field elements
        representationType = "polynomial" => characteristic()::Integer
        1
      found : Boolean := false
      for i in start..  while not found repeat
        e : $ := index(i::PositiveInteger)
        found := (order(e) = sm1)
      e

    primitive? a ==
      -- add special implementation for prime field case
      zero?(a) => false
      explist := factorsOfCyclicGroupSize()
      q:=(size()-1)@Integer
      equalone : Boolean := false
      for exp in explist while not equalone repeat
--        equalone := one?(a**(q quo exp.factor))
        equalone := ((a**(q quo exp.factor)) = 1)
      not equalone

    order e ==
      e = 0 => error "order(0) is not defined "
      ord:Integer:= size()-1 -- order e divides ord
      a:Integer:= 0
      lof:=factorsOfCyclicGroupSize()
      for rec in lof repeat -- run through prime divisors
        a := ord quo (primeDivisor := rec.factor)
--        goon := one?(e**a)
        goon := ((e**a) = 1)
        -- run through exponents of the prime divisors
        for j in 0..(rec.exponent)-2 while goon repeat
          -- as long as we get (e**ord = 1) we
          -- continue dividing by primeDivisor
          ord := a
          a := ord quo primeDivisor
--          goon := one?(e**a)
          goon := ((e**a) = 1)
        if goon then ord := a
        -- as we do a top down search we have found the
        -- correct exponent of primeDivisor in order e
```

```
      -- and continue with next prime divisor
    ord pretend PositiveInteger

  discreteLog(b) ==
    zero?(b) => error "discreteLog: logarithm of zero"
    faclist:=factorsOfCyclicGroupSize()
    a:=b
    gen:=primitiveElement()
    -- in GF(2) its necessary to have discreteLog(1) = 1
    b = gen => 1
    disclog:Integer:=0
    mult:Integer:=1
    groupord := (size() - 1)@Integer
    exp:Integer:=groupord
    for f in faclist repeat
      fac:=f.factor
      for t in 0..f.exponent-1 repeat
        exp:=exp quo fac
        -- shanks discrete logarithm algorithm
        exptable:=tableForDiscreteLogarithm(fac)
        n:=#exptable
        c:=a**exp
        end:=(fac - 1) quo n
        found:=false
        disc1:Integer:=0
        for i in 0..end while not found repeat
          rho:= search(lookup(c),exptable)_
                $Table(PositiveInteger,NNI)
          rho case NNI =>
            found := true
            disc1:=((n * i + rho)@Integer) * mult
          c:=c* gen**((groupord quo fac) * (-n))
        not found => error "discreteLog: ?? discrete logarithm"
        -- end of shanks discrete logarithm algorithm
        mult := mult * fac
        disclog:=disclog+disc1
        a:=a * (gen ** (-disc1))
    disclog pretend NonNegativeInteger

  discreteLog(logbase,b) ==
    zero?(b) =>
      messagePrint("discreteLog: logarithm of zero")$OutputForm
      "failed"
    zero?(logbase) =>
      messagePrint("discreteLog: logarithm to base zero")$OutputForm
      "failed"
    b = logbase => 1
    not zero?((groupord:=order(logbase)@PI) rem order(b)@PI) =>
       messagePrint("discreteLog: second argument not in cyclic group_
generated by first argument")$OutputForm
```

```
        "failed"
      faclist:=factors factor groupord
      a:=b
      disclog:Integer:=0
      mult:Integer:=1
      exp:Integer:= groupord
      for f in faclist repeat
        fac:=f.factor
        primroot:= logbase ** (groupord quo fac)
        for t in 0..f.exponent-1 repeat
          exp:=exp quo fac
          rhoHelp:= shanksDiscLogAlgorithm(primroot,_
                a**exp,fac pretend NonNegativeInteger)$DLP($)
          rhoHelp case "failed" => return "failed"
          rho := (rhoHelp :: NNI) * mult
          disclog := disclog + rho
          mult := mult * fac
          a:=a * (logbase ** (-rho))
        disclog pretend NonNegativeInteger

    FP ==> SparseUnivariatePolynomial($)
    FRP ==> Factored FP
    f,g:FP

    squareFreePolynomial(f:FP):FRP ==
          squareFree(f)$UnivariatePolynomialSquareFree($,FP)

    factorPolynomial(f:FP):FRP == factor(f)$DistinctDegreeFactorize($,FP)

    factorSquareFreePolynomial(f:FP):FRP ==
        f = 0 => 0
        flist := distdfact(f,true)$DistinctDegreeFactorize($,FP)
        (flist.cont :: FP) *
            (*/[primeFactor(u.irr,u.pow) for u in flist.factors])

    gcdPolynomial(f:FP,g:FP):FP ==
         gcd(f,g)$EuclideanDomain_&(FP)
```

———-

— FFIELDC.dotabb —

```
"FFIELDC"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FFIELDC"];
"FFIELDC" -> "FPC"
"FFIELDC" -> "FINITE"
"FFIELDC" -> "STEP"
"FFIELDC" -> "DIFRING"
```

———-

— **FFIELDC.dotfull** —

```
"FiniteFieldCategory()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FFIELDC"];
"FiniteFieldCategory()" -> "FieldOfPrimeCharacteristic()"
"FiniteFieldCategory()" -> "Finite()"
"FiniteFieldCategory()" -> "StepThrough()"
"FiniteFieldCategory()" -> "DifferentialRing()"
```

———-

— **FFIELDC.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FiniteFieldCategory()" [color=lightblue];
"FiniteFieldCategory()" -> "FieldOfPrimeCharacteristic()"
"FiniteFieldCategory()" -> "Finite()"
"FiniteFieldCategory()" -> "StepThrough()"
"FiniteFieldCategory()" -> "DifferentialRing()"

"FieldOfPrimeCharacteristic()" [color=lightblue];
"FieldOfPrimeCharacteristic()" -> "CHARNZ..."
"FieldOfPrimeCharacteristic()" -> "FIELD..."

"Finite()" [color=lightblue];
"Finite()" -> "SETCAT..."

"StepThrough()" [color=lightblue];
"StepThrough()" -> "SETCAT..."

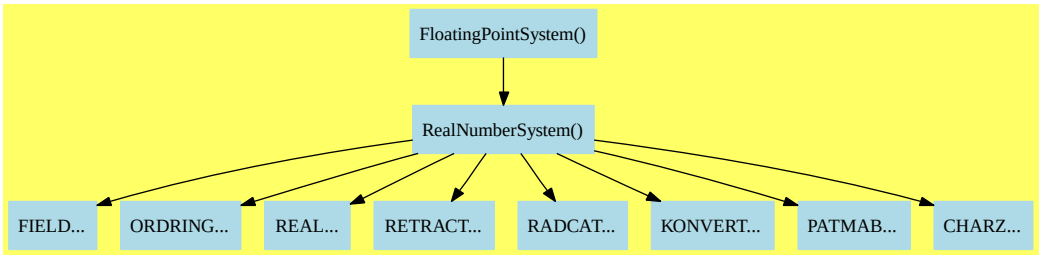
"DifferentialRing()" [color=lightblue];
"DifferentialRing()" -> "RING..."

"RING..." [color=lightblue];
"FIELD..." [color=lightblue];
"CHARNZ..." [color=lightblue];
"SETCAT..." [color=lightblue];

}
```

———-

18.0.208 FloatingPointSystem (FPS)



— FloatingPointSystem.input —

```
)set break resume
)sys rm -f FloatingPointSystem.output
)spool FloatingPointSystem.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FloatingPointSystem
--R FloatingPointSystem  is a category constructor
--R Abbreviation for FloatingPointSystem is FPS
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FPS
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (Fraction Integer,%) -> %       ?*? : (%,Fraction Integer) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,Fraction Integer) -> %
--R ?**? : (%,Integer) -> %               ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?/? : (%,%) -> %
--R ?<? : (%,%) -> Boolean                ?<=? : (%,%) -> Boolean
--R ?=? : (%,%) -> Boolean                ?>? : (%,%) -> Boolean
--R ?>=? : (%,%) -> Boolean               1 : () -> %
--R 0 : () -> %                           ?^? : (%,Integer) -> %
--R ?^? : (%,PositiveInteger) -> %        abs : % -> %
--R associates? : (%,%) -> Boolean        base : () -> PositiveInteger
--R bits : () -> PositiveInteger          ceiling : % -> %
--R coerce : Fraction Integer -> %        coerce : Integer -> %
--R coerce : Fraction Integer -> %        coerce : % -> %
```

```
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R convert : % -> Pattern Float          convert : % -> DoubleFloat
--R convert : % -> Float                  digits : () -> PositiveInteger
--R exponent : % -> Integer               factor : % -> Factored %
--R float : (Integer,Integer) -> %        floor : % -> %
--R fractionPart : % -> %                 gcd : List % -> %
--R gcd : (%,%) -> %                      hash : % -> SingleInteger
--R inv : % -> %                          latex : % -> String
--R lcm : List % -> %                     lcm : (%,%) -> %
--R mantissa : % -> Integer               max : (%,%) -> %
--R min : (%,%) -> %                      negative? : % -> Boolean
--R norm : % -> %                         nthRoot : (%,Integer) -> %
--R one? : % -> Boolean                   order : % -> Integer
--R positive? : % -> Boolean              precision : () -> PositiveInteger
--R prime? : % -> Boolean                 ?quo? : (%,%) -> %
--R recip : % -> Union(%,"failed")        ?rem? : (%,%) -> %
--R retract : % -> Fraction Integer       retract : % -> Integer
--R round : % -> %                        sample : () -> %
--R sign : % -> Integer                   sizeLess? : (%,%) -> Boolean
--R sqrt : % -> %                         squareFree : % -> Factored %
--R squareFreePart : % -> %               truncate : % -> %
--R unit? : % -> Boolean                  unitCanonical : % -> %
--R wholePart : % -> Integer              zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R bits : PositiveInteger -> PositiveInteger if $ has arbitraryPrecision
--R characteristic : () -> NonNegativeInteger
--R decreasePrecision : Integer -> PositiveInteger if $ has arbitraryPrecision
--R digits : PositiveInteger -> PositiveInteger if $ has arbitraryPrecision
--R divide : (%,%) -> Record(quotient: %,remainder: %)
--R euclideanSize : % -> NonNegativeInteger
--R expressIdealMember : (List %,%) -> Union(List %,"failed")
--R exquo : (%,%) -> Union(%,"failed")
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
--R float : (Integer,Integer,PositiveInteger) -> %
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivariatePolynomia
--R increasePrecision : Integer -> PositiveInteger if $ has arbitraryPrecision
--R max : () -> % if not has($,arbitraryPrecision) and not has($,arbitraryExponent)
--R min : () -> % if not has($,arbitraryPrecision) and not has($,arbitraryExponent)
--R multiEuclidean : (List %,%) -> Union(List %,"failed")
--R patternMatch : (%,Pattern Float,PatternMatchResult(Float,%)) -> PatternMatchResult(Float,%)
--R precision : PositiveInteger -> PositiveInteger if $ has arbitraryPrecision
--R principalIdeal : List % -> Record(coef: List %,generator: %)
--R retractIfCan : % -> Union(Fraction Integer,"failed")
--R retractIfCan : % -> Union(Integer,"failed")
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
```

```
--R
--E 1

)spool
)lisp (bye)
```

———-

— **FloatingPointSystem.help** —

```
=====================================================================
FloatingPointSystem examples
=====================================================================

See Also:
o )show FloatingPointSystem
```

———-

See:

⇐ "RealNumberSystem" (RNS) 17.0.200 on page 1400

Exports:

0	1	abs	associates?
base	bits	characteristic	ceiling
coerce	convert	decreasePrecision	digits
divide	euclideanSize	exponent	expressIdealMember
exquo	extendedEuclidean	factor	float
floor	fractionPart	gcd	gcdPolynomial
hash	increasePrecision	inv	latex
lcm	mantissa	max	min
multiEuclidean	negative?	norm	nthRoot
one?	order	patternMatch	positive?
precision	prime?	principalIdeal	recip
retract	retractIfCan	round	sample
sign	sizeLess?	sqrt	squareFree
squareFreePart	subtractIfCan	truncate	unit?
unitCanonical	unitNormal	wholePart	zero?
?*?	?**?	?+?	?-?
-?	?/?	?<?	?<=?
?=?	?>?	?>=?	?^?
?~=?	?quo?	?rem?	

Attributes Exported:

- **approximate** means "is an approximation to the real numbers".
- **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.
- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
base : () -> PositiveInteger
bits : () -> PositiveInteger
bits : PositiveInteger -> PositiveInteger
   if $ has arbitraryPrecision
decreasePrecision : Integer -> PositiveInteger
   if $ has arbitraryPrecision
digits : PositiveInteger -> PositiveInteger
   if $ has arbitraryPrecision
exponent : % -> Integer
float : (Integer,Integer,PositiveInteger) -> %
increasePrecision : Integer -> PositiveInteger
   if $ has arbitraryPrecision
mantissa : % -> Integer
max : () -> %
   if not has($,arbitraryPrecision)
   and not has($,arbitraryExponent)
min : () -> %
   if not has($,arbitraryPrecision)
   and not has($,arbitraryExponent)
order : % -> Integer
precision : () -> PositiveInteger
precision : PositiveInteger -> PositiveInteger
   if $ has arbitraryPrecision
```

These are implemented by this category:

```
digits : () -> PositiveInteger
float : (Integer,Integer) -> %
```

These exports come from (p1400) RealNumberSystem():

```
0 : () -> %
1 : () -> %
abs : % -> %
associates? : (%,%) -> Boolean
characteristic : () -> NonNegativeInteger
ceiling : % -> %
coerce : Fraction Integer -> %
coerce : Integer -> %
coerce : Fraction Integer -> %
coerce : % -> %
coerce : Integer -> %
coerce : % -> OutputForm
convert : % -> Pattern Float
convert : % -> DoubleFloat
convert : % -> Float
divide : (%,%) -> Record(quotient: %,remainder: %)
euclideanSize : % -> NonNegativeInteger
expressIdealMember : (List %,%) -> Union(List %,"failed")
exquo : (%,%) -> Union(%,"failed")
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
factor : % -> Factored %
floor : % -> %
fractionPart : % -> %
gcd : List % -> %
gcd : (%,%) -> %
gcdPolynomial :
  (SparseUnivariatePolynomial %,
   SparseUnivariatePolynomial %) ->
      SparseUnivariatePolynomial %
hash : % -> SingleInteger
inv : % -> %
latex : % -> String
lcm : List % -> %
lcm : (%,%) -> %
max : (%,%) -> %
min : (%,%) -> %
multiEuclidean : (List %,%) -> Union(List %,"failed")
negative? : % -> Boolean
norm : % -> %
nthRoot : (%,Integer) -> %
one? : % -> Boolean
patternMatch :
  (%,Pattern Float,PatternMatchResult(Float,%)) ->
     PatternMatchResult(Float,%)
positive? : % -> Boolean
prime? : % -> Boolean
principalIdeal : List % -> Record(coef: List %,generator: %)
```

```
recip : % -> Union(%,"failed")
retract : % -> Fraction Integer
retract : % -> Integer
retractIfCan : % -> Union(Fraction Integer,"failed")
retractIfCan : % -> Union(Integer,"failed")
round : % -> %
sample : () -> %
sign : % -> Integer
sizeLess? : (%,%) -> Boolean
sqrt : % -> %
squareFree : % -> Factored %
squareFreePart : % -> %
subtractIfCan : (%,%) -> Union(%,"failed")
truncate : % -> %
unit? : % -> Boolean
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
wholePart : % -> Integer
zero? : % -> Boolean
?*? : (Fraction Integer,%) -> %
?*? : (%,Fraction Integer) -> %
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?**? : (%,Fraction Integer) -> %
?**? : (%,Integer) -> %
?**? : (%,PositiveInteger) -> %
?+? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
?/? : (%,%) -> %
?<? : (%,%) -> Boolean
?<=? : (%,%) -> Boolean
?=? : (%,%) -> Boolean
?>? : (%,%) -> Boolean
?>=? : (%,%) -> Boolean
?^? : (%,Integer) -> %
?^? : (%,PositiveInteger) -> %
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?quo? : (%,%) -> %
?rem? : (%,%) -> %
```

— category FPS FloatingPointSystem —

```
)abbrev category FPS FloatingPointSystem
++ Author:
```

```
++ Date Created:
++ Change History:
++ Basic Operations: approximate, base, bits, digits, exponent, float,
++    mantissa, order, precision, round?
++ Related Constructors:
++ Keywords: float, floating point
++ Description:
++ This category is intended as a model for floating point systems.
++ A floating point system is a model for the real numbers.  In fact,
++ it is an approximation in the sense that not all real numbers are
++ exactly representable by floating point numbers.
++ A floating point system is characterized by the following:
++
++ 1: base of the exponent where the actual implemenations are
++ usually binary or decimal)\br
++ 2: precision of the mantissa (arbitrary or fixed)\br
++ 3: rounding error for operations
--++   4:  when, and what happens if exponent overflow/underflow occurs
++
++ Because a Float is an approximation to the real numbers, even though
++ it is defined to be a join of a Field and OrderedRing, some of
++ the attributes do not hold.  In particular associative("+")
++ does not hold.  Algorithms defined over a field need special
++ considerations when the field is a floating point system.

FloatingPointSystem(): Category == RealNumberSystem() with
   approximate
      ++ \spad{approximate} means "is an approximation to the real numbers".
   float: (Integer,Integer) -> %
      ++ float(a,e) returns \spad{a * base() ** e}.
   float: (Integer,Integer,PositiveInteger) -> %
      ++ float(a,e,b) returns \spad{a * b ** e}.
   order: % -> Integer
      ++ order x is the order of magnitude of x.
      ++ Note that \spad{base ** order x <= |x| < base ** (1 + order x)}.
   base: () -> PositiveInteger
      ++ base() returns the base of the
      ++\spadfunFrom{exponent}{FloatingPointSystem}.

   exponent: % -> Integer
      ++ exponent(x) returns the
      ++ \spadfunFrom{exponent}{FloatingPointSystem} part of x.

   mantissa: % -> Integer
      ++ mantissa(x) returns the mantissa part of x.
  -- round?: () -> B
  --    ++ round?() returns the rounding or chopping.

   bits: () -> PositiveInteger
      ++ bits() returns ceiling's precision in bits.
```

```
  digits: () -> PositiveInteger
     ++ digits() returns ceiling's precision in decimal digits.
  precision: () -> PositiveInteger
     ++ precision() returns the precision in digits base.

  if % has arbitraryPrecision then
     bits: PositiveInteger -> PositiveInteger
       ++ bits(n) set the \spadfunFrom{precision}{FloatingPointSystem}
       ++ to n bits.

     digits: PositiveInteger -> PositiveInteger
       ++ digits(d) set the \spadfunFrom{precision}{FloatingPointSystem}
       ++ to d digits.

     precision: PositiveInteger -> PositiveInteger
       ++ precision(n) set the precision in the base to n decimal digits.

     increasePrecision: Integer -> PositiveInteger
       ++ increasePrecision(n) increases the current
       ++ \spadfunFrom{precision}{FloatingPointSystem} by n decimal digits.

     decreasePrecision: Integer -> PositiveInteger
       ++ decreasePrecision(n) decreases the current
       ++ \spadfunFrom{precision}{FloatingPointSystem} precision
       ++ by n decimal digits.

  if not (% has arbitraryExponent) then
   --  overflow: (()->Exit) -> Void
   --    ++ overflow() returns the Exponent overflow of float
   --  underflow: (()->Exit) -> Void
   --    ++ underflow() returns the Exponent underflow of float
   --  maxExponent: () -> Integer
   --    ++ maxExponent() returns the max Exponent of float
     if not (% has arbitraryPrecision) then
        min: () -> %
         ++ min() returns the minimum floating point number.
        max: () -> %
         ++ max() returns the maximum floating point number.
add
  float(ma, ex) == float(ma, ex, base())
  digits() == max(1,4004 * (bits()-1) quo 13301)::PositiveInteger
```

———-

— FPS.dotabb —

```
"FPS"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FPS"];
```

```
"FPS" -> "RNS"
```

———-

— FPS.dotfull —

```
"FloatingPointSystem()"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FPS"];
"FloatingPointSystem()" -> "RealNumberSystem()"
```

———-

— FPS.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

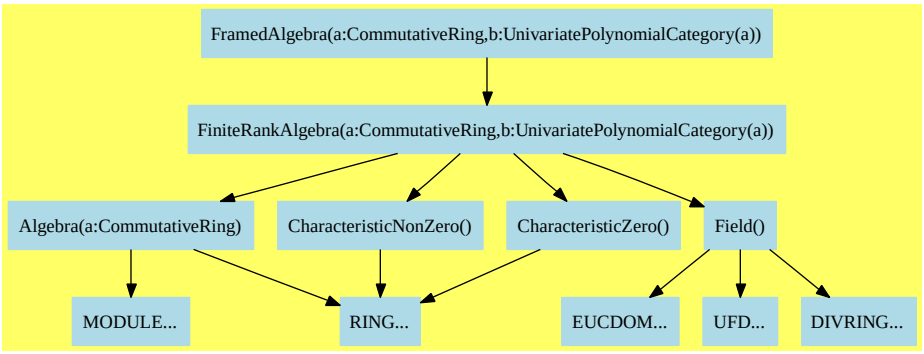
"FloatingPointSystem()" [color=lightblue];
"FloatingPointSystem()" -> "RealNumberSystem()"

"RealNumberSystem()" [color=lightblue];
"RealNumberSystem()" -> "FIELD..."
"RealNumberSystem()" -> "ORDRING..."
"RealNumberSystem()" -> "REAL..."
"RealNumberSystem()" -> "RETRACT..."
"RealNumberSystem()" -> "RADCAT..."
"RealNumberSystem()" -> "KONVERT..."
"RealNumberSystem()" -> "PATMAB..."
"RealNumberSystem()" -> "CHARZ..."

"FIELD..." [color=lightblue];
"ORDRING..." [color=lightblue];
"REAL..." [color=lightblue];
"RETRACT..." [color=lightblue];
"RADCAT..." [color=lightblue];
"KONVERT..." [color=lightblue];
"PATMAB..." [color=lightblue];
"CHARZ..." [color=lightblue];
}
```

———-

18.0.209 FramedAlgebra (FRAMALG)



— FramedAlgebra.input —

```
)set break resume
)sys rm -f FramedAlgebra.output
)spool FramedAlgebra.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FramedAlgebra
--R FramedAlgebra(R: CommutativeRing,UP: UnivariatePolynomialCategory t#1)  is a category constructor
--R Abbreviation for FramedAlgebra is FRAMALG
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FRAMALG
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (R,%) -> %                      ?*? : (%,R) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        basis : () -> Vector %
--R coerce : R -> %                       coerce : Integer -> %
--R coerce : % -> OutputForm              convert : Vector R -> %
--R convert : % -> Vector R               coordinates : % -> Vector R
--R discriminant : () -> R                discriminant : Vector % -> R
--R hash : % -> SingleInteger             latex : % -> String
--R norm : % -> R                         one? : % -> Boolean
--R rank : () -> PositiveInteger          recip : % -> Union(%,"failed")
--R represents : Vector R -> %            sample : () -> %
--R trace : % -> R                        traceMatrix : () -> Matrix R
```

```
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R characteristicPolynomial : % -> UP
--R charthRoot : % -> Union(%,"failed") if R has CHARNZ
--R coordinates : Vector % -> Matrix R
--R coordinates : (Vector %,Vector %) -> Matrix R
--R coordinates : (%,Vector %) -> Vector R
--R minimalPolynomial : % -> UP if R has FIELD
--R regularRepresentation : % -> Matrix R
--R regularRepresentation : (%,Vector %) -> Matrix R
--R represents : (Vector R,Vector %) -> %
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R traceMatrix : Vector % -> Matrix R
--R
--E 1

)spool
)lisp (bye)
```

———-

— **FramedAlgebra.help** —

```
=====================================================================
FramedAlgebra examples
=====================================================================

See Also:
o )show FramedAlgebra
```

———-

See:

⇒ “MonogenicAlgebra” (MONOGEN) 19.0.214 on page 1598
⇐ “FiniteRankAlgebra” (FINRALG) 17.0.194 on page 1329

Exports:

0	1	basis
characteristic	characteristicPolynomial	charthRoot
coerce	convert	coordinates
discriminant	hash	latex
minimalPolynomial	norm	one?
rank	recip	regularRepresentation
represents	sample	subtractIfCan
trace	traceMatrix	zero?
?*?	?**?	?+?
?-?	-?	?=?
?^?	?~=?	

Attributes Exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
basis : () -> Vector %
represents : Vector R -> %
```

These are implemented by this category:

```
convert : Vector R -> %
convert : % -> Vector R
coordinates : Vector % -> Matrix R
coordinates : % -> Vector R
discriminant : () -> R
regularRepresentation : % -> Matrix R
traceMatrix : () -> Matrix R
```

These exports come from (p1329) FiniteRankAlgebra(R, UP)
where R:CommutativeRing and UP:UnivariatePolynomialCategory R):

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
characteristicPolynomial : % -> UP
charthRoot : % -> Union(%,"failed") if R has CHARNZ
coerce : R -> %
coerce : Integer -> %
coerce : % -> OutputForm
coordinates : (%,Vector %) -> Vector R
coordinates : (Vector %,Vector %) -> Matrix R
```

```
 discriminant : Vector % -> R
 hash : % -> SingleInteger
 latex : % -> String
 minimalPolynomial : % -> UP if R has FIELD
 norm : % -> R
 one? : % -> Boolean
 rank : () -> PositiveInteger
 recip : % -> Union(%,"failed")
 regularRepresentation : (%,Vector %) -> Matrix R
 represents : (Vector R,Vector %) -> %
 sample : () -> %
 subtractIfCan : (%,%) -> Union(%,"failed")
 trace : % -> R
 traceMatrix : Vector % -> Matrix R
 zero? : % -> Boolean
 ?+? : (%,%) -> %
 ?=? : (%,%) -> Boolean
 ?~=? : (%,%) -> Boolean
 ?*? : (%,%) -> %
 ?*? : (Integer,%) -> %
 ?*? : (PositiveInteger,%) -> %
 ?*? : (NonNegativeInteger,%) -> %
 ?*? : (R,%) -> %
 ?*? : (%,R) -> %
 ?-? : (%,%) -> %
 -? : % -> %
 ?**? : (%,PositiveInteger) -> %
 ?**? : (%,NonNegativeInteger) -> %
 ?^? : (%,PositiveInteger) -> %
 ?^? : (%,NonNegativeInteger) -> %
```

— category FRAMALG FramedAlgebra —

```
)abbrev category FRAMALG FramedAlgebra
++ Author: Barry Trager
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A \spadtype{FramedAlgebra} is a \spadtype{FiniteRankAlgebra} together
++ with a fixed R-module basis.

FramedAlgebra(R:CommutativeRing, UP:UnivariatePolynomialCategory R):
 Category == FiniteRankAlgebra(R, UP) with
```

```
    basis                 : () -> Vector %
      ++ basis() returns the fixed R-module basis.
    coordinates           : % -> Vector R
      ++ coordinates(a) returns the coordinates of \spad{a} with
      ++ respect to the fixed R-module basis.
    coordinates           : Vector % -> Matrix R
      ++ coordinates([v1,...,vm]) returns the coordinates of the
      ++ vi's with to the fixed basis.  The coordinates of vi are
      ++ contained in the ith row of the matrix returned by this
      ++ function.
    represents            : Vector R -> %
      ++ represents([a1,..,an]) returns \spad{a1*v1 + ... + an*vn}, where
      ++ v1, ..., vn are the elements of the fixed basis.
    convert               : % -> Vector R
      ++ convert(a) returns the coordinates of \spad{a} with respect to the
      ++ fixed R-module basis.
    convert               : Vector R -> %
      ++ convert([a1,..,an]) returns \spad{a1*v1 + ... + an*vn}, where
      ++ v1, ..., vn are the elements of the fixed basis.
    traceMatrix           : () -> Matrix R
      ++ traceMatrix() is the n-by-n matrix ( \spad{Tr(vi * vj)} ), where
      ++ v1, ..., vn are the elements of the fixed basis.
    discriminant          : () -> R
      ++ discriminant() = determinant(traceMatrix()).
    regularRepresentation : % -> Matrix R
      ++ regularRepresentation(a) returns the matrix of the linear
      ++ map defined by left multiplication by \spad{a} with respect
      ++ to the fixed basis.
  --attributes
    --separable <=> discriminant() ^= 0
add
 convert(x:%):Vector(R)  == coordinates(x)
 convert(v:Vector R):%   == represents(v)
 traceMatrix()           == traceMatrix basis()
 discriminant()          == discriminant basis()
 regularRepresentation x == regularRepresentation(x, basis())
 coordinates x           == coordinates(x, basis())
 represents x            == represents(x, basis())

 coordinates(v:Vector %) ==
   m := new(#v, rank(), 0)$Matrix(R)
   for i in minIndex v .. maxIndex v for j in minRowIndex m .. repeat
     setRow_!(m, j, coordinates qelt(v, i))
   m

 regularRepresentation x ==
   m := new(n := rank(), n, 0)$Matrix(R)
   b := basis()
   for i in minIndex b .. maxIndex b for j in minRowIndex m .. repeat
     setRow_!(m, j, coordinates(x * qelt(b, i)))
```

```
     m

   characteristicPolynomial x ==
      mat00 := (regularRepresentation x)
      mat0 := map(y+->y::UP,mat00)$MatrixCategoryFunctions2(R, Vector R,
                  Vector R, Matrix R, UP, Vector UP,Vector UP, Matrix UP)
      mat1 : Matrix UP := scalarMatrix(rank(),monomial(1,1)$UP)
      determinant(mat1 - mat0)

   if R has Field then
    -- depends on the ordering of results from nullSpace, also see FFP
      minimalPolynomial(x:%):UP ==
        y:%:=1
        n:=rank()
        m:Matrix R:=zero(n,n+1)
        for i in 1..n+1 repeat
          setColumn_!(m,i,coordinates(y))
          y:=y*x
        v:=first nullSpace(m)
        +/[monomial(v.(i+1),i) for i in 0..#v-1]
```

———-

— FRAMALG.dotabb —

```
"FRAMALG"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FRAMALG"];
"FRAMALG" -> "FINRALG"
```

———-

— FRAMALG.dotfull —

```
"FramedAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FRAMALG"];
"FramedAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
   "FiniteRankAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))"
```

———-

— FRAMALG.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
```

```
 node [shape=box, color=white, style=filled];

"FramedAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))"
 [color=lightblue];
"FramedAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
   "FiniteRankAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))"

"FiniteRankAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))"
 [color=lightblue];
"FiniteRankAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "Algebra(a:CommutativeRing)"
"FiniteRankAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "Field()"
"FiniteRankAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "CharacteristicNonZero()"
"FiniteRankAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "CharacteristicZero()"

"Algebra(a:CommutativeRing)" [color=lightblue];
"Algebra(a:CommutativeRing)" -> "RING..."
"Algebra(a:CommutativeRing)" -> "MODULE..."

"Field()" [color=lightblue];
"Field()" -> "EUCDOM..."
"Field()" -> "UFD..."
"Field()" -> "DIVRING..."

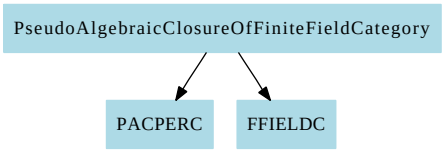
"CharacteristicNonZero()" [color=lightblue];
"CharacteristicNonZero()" -> "RING..."

"CharacteristicZero()" [color=lightblue];
"CharacteristicZero()" -> "RING..."

"EUCDOM..." [color=lightblue];
"UFD..." [color=lightblue];
"DIVRING..." [color=lightblue];
"RING..." [color=lightblue];
"MODULE..." [color=lightblue];
}
```

———-

18.0.210 PseudoAlgebraicClosureOfFiniteFieldCategory (PACFFC)



— PseudoAlgebraicClosureOfFiniteFieldCategory.input —

```
)set break resume
)sys rm -f PseudoAlgebraicClosureOfFiniteFieldCategory.output
)spool PseudoAlgebraicClosureOfFiniteFieldCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show PseudoAlgebraicClosureOfFiniteFieldCategory
--R
--R PseudoAlgebraicClosureOfFiniteFieldCategory  is a category constructor
--R Abbreviation for PseudoAlgebraicClosureOfFiniteFieldCategory is PACFFC
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PACFFC
--R
--R------------------------------ Operations ------------------------------
--R ?*? : (Fraction Integer,%) -> %       ?*? : (%,Fraction Integer) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,Integer) -> %
--R ?**? : (%,PositiveInteger) -> %       ?+? : (%,%) -> %
--R ?-? : (%,%) -> %                      -? : % -> %
--R ?/? : (%,%) -> %                      ?=? : (%,%) -> Boolean
--R D : % -> %                            D : (%,NonNegativeInteger) -> %
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,Integer) -> %                ?^? : (%,PositiveInteger) -> %
--R associates? : (%,%) -> Boolean        charthRoot : % -> %
--R coerce : Fraction Integer -> %        coerce : % -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R conjugate : % -> %                    createPrimitiveElement : () -> %
--R differentiate : % -> %                extDegree : % -> PositiveInteger
--R factor : % -> Factored %              fullOutput : % -> OutputForm
--R gcd : List % -> %                     gcd : (%,%) -> %
--R ground? : % -> Boolean                hash : % -> SingleInteger
--R index : PositiveInteger -> %          init : () -> %
--R inv : % -> %                          latex : % -> String
--R lcm : List % -> %                     lcm : (%,%) -> %
--R lookup : % -> PositiveInteger         maxTower : List % -> %
--R one? : % -> Boolean                   order : % -> PositiveInteger
```

```
--R previousTower : % -> %                prime? : % -> Boolean
--R primeFrobenius : % -> %               primitive? : % -> Boolean
--R primitiveElement : () -> %            ?quo? : (%,%) -> %
--R random : () -> %                      recip : % -> Union(%,"failed")
--R ?rem? : (%,%) -> %                    sample : () -> %
--R setTower! : % -> Void                 size : () -> NonNegativeInteger
--R sizeLess? : (%,%) -> Boolean          squareFree : % -> Factored %
--R squareFreePart : % -> %               unit? : % -> Boolean
--R unitCanonical : % -> %                vectorise : (%,%) -> Vector %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed")
--R conditionP : Matrix % -> Union(Vector %,"failed")
--R definingPolynomial : () -> SparseUnivariatePolynomial %
--R definingPolynomial : % -> SparseUnivariatePolynomial %
--R differentiate : (%,NonNegativeInteger) -> %
--R discreteLog : % -> NonNegativeInteger
--R discreteLog : (%,%) -> Union(NonNegativeInteger,"failed")
--R distinguishedRootsOf : (SparseUnivariatePolynomial %,%) -> List %
--R divide : (%,%) -> Record(quotient: %,remainder: %)
--R euclideanSize : % -> NonNegativeInteger
--R expressIdealMember : (List %,%) -> Union(List %,"failed")
--R exquo : (%,%) -> Union(%,"failed")
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
--R factorsOfCyclicGroupSize : () -> List Record(factor: Integer,exponent: Integer)
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivariatePolynomia
--R lift : % -> SparseUnivariatePolynomial %
--R lift : (%,%) -> SparseUnivariatePolynomial %
--R multiEuclidean : (List %,%) -> Union(List %,"failed")
--R newElement : (SparseUnivariatePolynomial %,%,Symbol) -> %
--R newElement : (SparseUnivariatePolynomial %,Symbol) -> %
--R nextItem : % -> Union(%,"failed")
--R order : % -> OnePointCompletion PositiveInteger
--R primeFrobenius : (%,NonNegativeInteger) -> %
--R principalIdeal : List % -> Record(coef: List %,generator: %)
--R reduce : SparseUnivariatePolynomial % -> %
--R representationType : () -> Union("prime",polynomial,normal,cyclic)
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R tableForDiscreteLogarithm : Integer -> Table(PositiveInteger,NonNegativeInteger)
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R
--E 1

)spool
)lisp (bye)
```

———-

— **PseudoAlgebraicClosureOfFiniteFieldCategory.help** —

```
==========================================================================
PseudoAlgebraicClosureOfFiniteFieldCategory examples
==========================================================================

See Also:
o )show PseudoAlgebraicClosureOfFiniteFieldCategory
```

———-

⇐ "FieldOfPrimeCharacteristic" (FPC) 17.0.193 on page 1323
⇐ "PseudoAlgebraicClosureOfPerfectFieldCategory" (PACPERC) 17.0.197 on page 1369

Exports:

0	1	associates?
characteristic	charthRoot	conditionP
coerce	conjugate	createPrimitiveElement
D	definingPolynomial	differentiate
discreteLog	distinguishedRootsOf	divide
euclideanSize	extendedEuclidean	expressIdealMember
exquo	extDegree	factor
factorsOfCyclicGroupSize	fullOutput	gcd
gcdPolynomial	ground?	hash
index	init	inv
latex	lcm	lift
lookup	maxTower	multiEuclidean
newElement	nextItem	one?
order	previousTower	prime?
primeFrobenius	primitive?	primitiveElement
principalIdeal	?quo?	random
recip	reduce	?rem?
representationType	sample	setTower!
size	sizeLess?	squareFree
squareFreePart	subtractIfCan	tableForDiscreteLogarithm
unit?	unitCanonical	unitNormal
vectorise	zero?	?*?
?**?	?+?	?-?
-?	?/?	?=?
?^?	?~=?	

Attributes Exported:

- **canonicalUnitNormal** is true if we can choose a canonical representative for each

class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.

- **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.
- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These exports come from (p1369) PseudoAlgebraicClosureOfPerfectFieldCategory():

```
?*? : (Fraction Integer,%) -> %
?*? : (%,Fraction Integer) -> %
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?**? : (%,Integer) -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?+? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
?/? : (%,%) -> %
?=? : (%,%) -> Boolean
1 : () -> %
0 : () -> %
?^? : (%,Integer) -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?~=? : (%,%) -> Boolean
associates? : (%,%) -> Boolean
characteristic : () -> NonNegativeInteger
coerce : Fraction Integer -> %
coerce : % -> %
coerce : Integer -> %
coerce : % -> OutputForm
conjugate : % -> %
definingPolynomial : () -> SUP %
definingPolynomial : % -> SUP %
distinguishedRootsOf : (SparseUnivariatePolynomial %,%) -> List %
```

```
divide : (%,%) -> Record(quotient: %,remainder: %)
euclideanSize : % -> NonNegativeInteger
expressIdealMember : (List %,%) -> Union(List %,"failed")
exquo : (%,%) -> Union(%,"failed")
extDegree : % -> PI
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
factor : % -> Factored %
fullOutput : % -> OutputForm
gcd : List % -> %
gcd : (%,%) -> %
gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivariate
ground? : % -> Boolean
hash : % -> SingleInteger
inv : % -> %
latex : % -> String
lcm : List % -> %
lcm : (%,%) -> %
lift : % -> SUP %
lift : (%,%) -> SUP %
maxTower : List % -> %
multiEuclidean : (List %,%) -> Union(List %,"failed")
newElement : (SUP %,Symbol) -> %
newElement : (SUP %,%,Symbol) -> %
one? : % -> Boolean
previousTower : % -> %
prime? : % -> Boolean
principalIdeal : List % -> Record(coef: List %,generator: %)
?quo? : (%,%) -> %
recip : % -> Union(%,"failed")
reduce : SUP % -> %
?rem? : (%,%) -> %
sample : () -> %
setTower! : % -> Void
sizeLess? : (%,%) -> Boolean
squareFree : % -> Factored %
squareFreePart : % -> %
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
vectorise : (%,%) -> Vector %
zero? : % -> Boolean
```

These exports come from (p1518) FiniteFieldCategory():

```
charthRoot : % -> %
charthRoot : % -> Union(%,"failed")
conditionP : Matrix % -> Union(Vector %,"failed")
createPrimitiveElement : () -> %
```

```
 D : % -> %
 D : (%,NonNegativeInteger) -> %
 differentiate : % -> %
 differentiate : (%,NonNegativeInteger) -> %
 discreteLog : % -> NonNegativeInteger
 discreteLog : (%,%) -> Union(NonNegativeInteger,"failed")
 factorsOfCyclicGroupSize : () ->
     List Record(factor: Integer,exponent: Integer)
 index : PositiveInteger -> %
 init : () -> %
 lookup : % -> PositiveInteger
 nextItem : % -> Union(%,"failed")
 order : % -> OnePointCompletion PositiveInteger
 order : % -> PositiveInteger
 primeFrobenius : (%,NonNegativeInteger) -> %
 primeFrobenius : % -> %
 primitive? : % -> Boolean
 primitiveElement : () -> %
 random : () -> %
 representationType : () -> Union("prime",polynomial,normal,cyclic)
 size : () -> NonNegativeInteger
 tableForDiscreteLogarithm : Integer ->
     Table(PositiveInteger,NonNegativeInteger)
```

— category PACFFC PseudoAlgebraicClosureOfFiniteFieldCategory —

```
)abbrev category  PACFFC PseudoAlgebraicClosureOfFiniteFieldCategory
-- PseudoAlgebraicClosureOfFiniteFieldCategory
++ Authors: Gaetan Hache
++ Date Created: june 1996
++ Date Last Updated:
++ References:
++ Description: This category exports the function for the domain
++ PseudoAlgebraicClosureOfFiniteField which implement dynamic extension
++ using the simple notion of tower extensions.
++ A tower extension T  of the ground
++ field K is any sequence of field extension (T : K_0, K_1, ..., K_i...,K_n)
++ where K_0 = K and for i =1,2,...,n, K_i is an extension
++ of K_{i-1} of degree > 1 and defined by an irreducible polynomial
++ p(Z) in K_{i-1}.
++ Two towers (T_1: K_01, K_11,...,K_i1,...,K_n1)
++ and (T_2: K_02, K_12,...,K_i2,...,K_n2)
++ are said to be related if T_1 <= T_2 (or T_1 >= T_2),
++ that is if K_i1 = K_i2 for i=1,2,...,n1
++ (or i=1,2,...,n2). Any algebraic operations defined for several elements
++ are only defined if all of the concerned elements are comming from
++ a set of related tour extensions.
PseudoAlgebraicClosureOfFiniteFieldCategory:Category ==
```

```
  Join(FiniteFieldCategory, PseudoAlgebraicClosureOfPerfectFieldCategory)
```

———-

— PACFFC.dotabb —

```
"PACFFC" [color=lightblue,href="bookvol10.2.pdf#nameddest=PACFFC"];
"PACFFC" -> "PACPERC"
```

———-

— PACFFC.dotfull —

```
"PseudoAlgebraicClosureOfFiniteFieldCategory"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PACFFC"];
"PseudoAlgebraicClosureOfFiniteFieldCategory" ->
  "PseudoAlgebraicClosureOfPerfectFieldCategory()"
"PseudoAlgebraicClosureOfFiniteFieldCategory" ->
  "FiniteFieldCategory()"
```

———-

— PACFFC.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

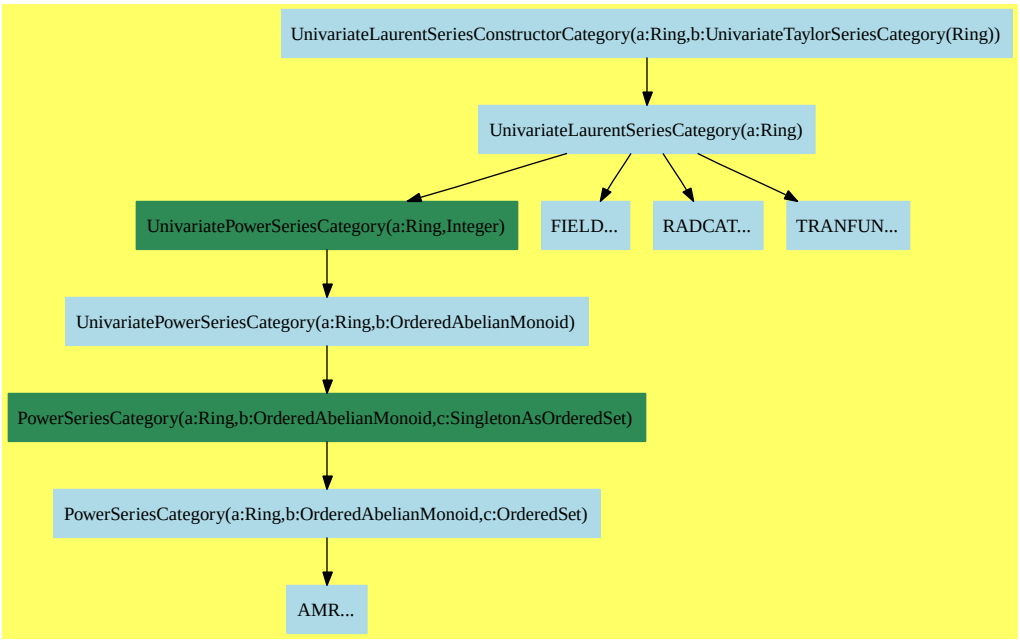
"PseudoAlgebraicClosureOfFiniteFieldCategory" [color=lightblue];
"PseudoAlgebraicClosureOfFiniteFieldCategory" -> "PACPERC"
"PseudoAlgebraicClosureOfFiniteFieldCategory" -> "FFIELDC"

"PACPERC" [color=lightblue];
"FFIELDC" [color=lightblue];

}
```

———-

18.0.211 UnivariateLaurentSeriesConstructorCategory (ULSCCAT)



— UnivariateLaurentSeriesConstructorCategory.input —

```
)set break resume
)sys rm -f UnivariateLaurentSeriesConstructorCategory.output
)spool UnivariateLaurentSeriesConstructorCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show UnivariateLaurentSeriesConstructorCategory
--R UnivariateLaurentSeriesConstructorCategory(Coef: Ring,UTS: UnivariateTaylorSeriesCategory t#1)  is a cat
--R Abbreviation for UnivariateLaurentSeriesConstructorCategory is ULSCCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for ULSCCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (Coef,%) -> %                  ?*? : (%,Coef) -> %
--R ?*? : (%,%) -> %                     ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %       ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                     ?-? : (%,%) -> %
--R -? : % -> %                          ?=? : (%,%) -> Boolean
--R 1 : () -> %                          0 : () -> %
```

```
--R ?^? : (%,PositiveInteger) -> %        center : % -> Coef
--R coefficient : (%,Integer) -> Coef     coerce : UTS -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R complete : % -> %                     degree : % -> Integer
--R ?.? : (%,Integer) -> Coef             extend : (%,Integer) -> %
--R hash : % -> SingleInteger             latex : % -> String
--R laurent : (Integer,UTS) -> %          leadingCoefficient : % -> Coef
--R leadingMonomial : % -> %              map : ((Coef -> Coef),%) -> %
--R monomial : (Coef,Integer) -> %        monomial? : % -> Boolean
--R one? : % -> Boolean                   order : (%,Integer) -> Integer
--R order : % -> Integer                  pole? : % -> Boolean
--R recip : % -> Union(%,"failed")        reductum : % -> %
--R removeZeroes : (Integer,%) -> %       removeZeroes : % -> %
--R retract : % -> UTS                    sample : () -> %
--R taylor : % -> UTS                     taylorRep : % -> UTS
--R truncate : (%,Integer) -> %           variable : % -> Symbol
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
--R ?*? : (Fraction Integer,%) -> % if Coef has ALGEBRA FRAC INT
--R ?*? : (UTS,%) -> % if Coef has FIELD
--R ?*? : (%,UTS) -> % if Coef has FIELD
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
--R ?**? : (%,%) -> % if Coef has ALGEBRA FRAC INT
--R ?**? : (%,Integer) -> % if Coef has FIELD
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (UTS,UTS) -> % if Coef has FIELD
--R ?/? : (%,%) -> % if Coef has FIELD
--R ?/? : (%,Coef) -> % if Coef has FIELD
--R ?<? : (%,%) -> Boolean if and(has(UTS,OrderedSet),has(Coef,Field))
--R ?<=? : (%,%) -> Boolean if and(has(UTS,OrderedSet),has(Coef,Field))
--R ?>? : (%,%) -> Boolean if and(has(UTS,OrderedSet),has(Coef,Field))
--R ?>=? : (%,%) -> Boolean if and(has(UTS,OrderedSet),has(Coef,Field))
--R D : (%,Symbol) -> % if and(has(UTS,PartialDifferentialRing Symbol),has(Coef,Field)) or Coef
--R D : (%,List Symbol) -> % if and(has(UTS,PartialDifferentialRing Symbol),has(Coef,Field)) or
--R D : (%,Symbol,NonNegativeInteger) -> % if and(has(UTS,PartialDifferentialRing Symbol),has(Co
--R D : (%,List Symbol,List NonNegativeInteger) -> % if and(has(UTS,PartialDifferentialRing Symb
--R D : % -> % if and(has(UTS,DifferentialRing),has(Coef,Field)) or Coef has *: (Integer,Coef) -
--R D : (%,NonNegativeInteger) -> % if and(has(UTS,DifferentialRing),has(Coef,Field)) or Coef ha
--R D : (%,(UTS -> UTS),NonNegativeInteger) -> % if Coef has FIELD
--R D : (%,(UTS -> UTS)) -> % if Coef has FIELD
--R ?^? : (%,Integer) -> % if Coef has FIELD
--R ?^? : (%,NonNegativeInteger) -> %
--R abs : % -> % if and(has(UTS,OrderedIntegralDomain),has(Coef,Field))
--R acos : % -> % if Coef has ALGEBRA FRAC INT
--R acosh : % -> % if Coef has ALGEBRA FRAC INT
--R acot : % -> % if Coef has ALGEBRA FRAC INT
--R acoth : % -> % if Coef has ALGEBRA FRAC INT
--R acsc : % -> % if Coef has ALGEBRA FRAC INT
--R acsch : % -> % if Coef has ALGEBRA FRAC INT
```

```
--R approximate : (%,Integer) -> Coef if Coef has **: (Coef,Integer) -> Coef and Coef has coerce: Symbol ->
--R asec : % -> % if Coef has ALGEBRA FRAC INT
--R asech : % -> % if Coef has ALGEBRA FRAC INT
--R asin : % -> % if Coef has ALGEBRA FRAC INT
--R asinh : % -> % if Coef has ALGEBRA FRAC INT
--R associates? : (%,%) -> Boolean if Coef has INTDOM
--R atan : % -> % if Coef has ALGEBRA FRAC INT
--R atanh : % -> % if Coef has ALGEBRA FRAC INT
--R ceiling : % -> UTS if and(has(UTS,IntegerNumberSystem),has(Coef,Field))
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if and(OR(has(UTS,CharacteristicNonZero),and(has($,CharacteristicNo
--R coerce : % -> % if Coef has INTDOM
--R coerce : Fraction Integer -> % if Coef has ALGEBRA FRAC INT
--R coerce : Symbol -> % if and(has(UTS,RetractableTo Symbol),has(Coef,Field))
--R coerce : Coef -> % if Coef has COMRING
--R conditionP : Matrix % -> Union(Vector %,"failed") if and(and(has($,CharacteristicNonZero),has(UTS,Polyn
--R convert : % -> Pattern Integer if and(has(UTS,ConvertibleTo Pattern Integer),has(Coef,Field))
--R convert : % -> Pattern Float if and(has(UTS,ConvertibleTo Pattern Float),has(Coef,Field))
--R convert : % -> InputForm if and(has(UTS,ConvertibleTo InputForm),has(Coef,Field))
--R convert : % -> Float if and(has(UTS,RealConstant),has(Coef,Field))
--R convert : % -> DoubleFloat if and(has(UTS,RealConstant),has(Coef,Field))
--R cos : % -> % if Coef has ALGEBRA FRAC INT
--R cosh : % -> % if Coef has ALGEBRA FRAC INT
--R cot : % -> % if Coef has ALGEBRA FRAC INT
--R coth : % -> % if Coef has ALGEBRA FRAC INT
--R csc : % -> % if Coef has ALGEBRA FRAC INT
--R csch : % -> % if Coef has ALGEBRA FRAC INT
--R denom : % -> UTS if Coef has FIELD
--R denominator : % -> % if Coef has FIELD
--R differentiate : (%,Symbol) -> % if and(has(UTS,PartialDifferentialRing Symbol),has(Coef,Field)) or Coef
--R differentiate : (%,List Symbol) -> % if and(has(UTS,PartialDifferentialRing Symbol),has(Coef,Field)) or
--R differentiate : (%,Symbol,NonNegativeInteger) -> % if and(has(UTS,PartialDifferentialRing Symbol),has(C
--R differentiate : (%,List Symbol,List NonNegativeInteger) -> % if and(has(UTS,PartialDifferentialRing Syml
--R differentiate : % -> % if and(has(UTS,DifferentialRing),has(Coef,Field)) or Coef has *: (Integer,Coef) -
--R differentiate : (%,NonNegativeInteger) -> % if and(has(UTS,DifferentialRing),has(Coef,Field)) or Coef ha
--R differentiate : (%,(UTS -> UTS),NonNegativeInteger) -> % if Coef has FIELD
--R differentiate : (%,(UTS -> UTS)) -> % if Coef has FIELD
--R divide : (%,%) -> Record(quotient: %,remainder: %) if Coef has FIELD
--R ?.? : (%,UTS) -> % if and(has(UTS,Eltable(UTS,UTS)),has(Coef,Field))
--R ?.? : (%,%) -> % if Integer has SGROUP
--R euclideanSize : % -> NonNegativeInteger if Coef has FIELD
--R eval : (%,List UTS,List UTS) -> % if and(has(UTS,Evalable UTS),has(Coef,Field))
--R eval : (%,UTS,UTS) -> % if and(has(UTS,Evalable UTS),has(Coef,Field))
--R eval : (%,Equation UTS) -> % if and(has(UTS,Evalable UTS),has(Coef,Field))
--R eval : (%,List Equation UTS) -> % if and(has(UTS,Evalable UTS),has(Coef,Field))
--R eval : (%,List Symbol,List UTS) -> % if and(has(UTS,InnerEvalable(Symbol,UTS)),has(Coef,Field))
--R eval : (%,Symbol,UTS) -> % if and(has(UTS,InnerEvalable(Symbol,UTS)),has(Coef,Field))
--R eval : (%,Coef) -> Stream Coef if Coef has **: (Coef,Integer) -> Coef
--R exp : % -> % if Coef has ALGEBRA FRAC INT
--R expressIdealMember : (List %,%) -> Union(List %,"failed") if Coef has FIELD
```

```
--R exquo : (%,%) -> Union(%,"failed") if Coef has INTDOM
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %) if Coef has FIELD
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed") if Coef has FIELD
--R factor : % -> Factored % if Coef has FIELD
--R factorPolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial % if
--R factorSquareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolyno
--R floor : % -> UTS if and(has(UTS,IntegerNumberSystem),has(Coef,Field))
--R fractionPart : % -> % if and(has(UTS,EuclideanDomain),has(Coef,Field))
--R gcd : (%,%) -> % if Coef has FIELD
--R gcd : List % -> % if Coef has FIELD
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivari
--R init : () -> % if and(has(UTS,StepThrough),has(Coef,Field))
--R integrate : (%,Symbol) -> % if Coef has ACFS INT and Coef has PRIMCAT and Coef has TRANFUN a
--R integrate : % -> % if Coef has ALGEBRA FRAC INT
--R inv : % -> % if Coef has FIELD
--R lcm : (%,%) -> % if Coef has FIELD
--R lcm : List % -> % if Coef has FIELD
--R log : % -> % if Coef has ALGEBRA FRAC INT
--R map : ((UTS -> UTS),%) -> % if Coef has FIELD
--R max : (%,%) -> % if and(has(UTS,OrderedSet),has(Coef,Field))
--R min : (%,%) -> % if and(has(UTS,OrderedSet),has(Coef,Field))
--R monomial : (%,List SingletonAsOrderedSet,List Integer) -> %
--R monomial : (%,SingletonAsOrderedSet,Integer) -> %
--R multiEuclidean : (List %,%) -> Union(List %,"failed") if Coef has FIELD
--R multiplyCoefficients : ((Integer -> Coef),%) -> %
--R multiplyExponents : (%,PositiveInteger) -> %
--R negative? : % -> Boolean if and(has(UTS,OrderedIntegralDomain),has(Coef,Field))
--R nextItem : % -> Union(%,"failed") if and(has(UTS,StepThrough),has(Coef,Field))
--R nthRoot : (%,Integer) -> % if Coef has ALGEBRA FRAC INT
--R numer : % -> UTS if Coef has FIELD
--R numerator : % -> % if Coef has FIELD
--R patternMatch : (%,Pattern Integer,PatternMatchResult(Integer,%)) -> PatternMatchResult(Integ
--R patternMatch : (%,Pattern Float,PatternMatchResult(Float,%)) -> PatternMatchResult(Float,%)
--R pi : () -> % if Coef has ALGEBRA FRAC INT
--R positive? : % -> Boolean if and(has(UTS,OrderedIntegralDomain),has(Coef,Field))
--R prime? : % -> Boolean if Coef has FIELD
--R principalIdeal : List % -> Record(coef: List %,generator: %) if Coef has FIELD
--R ?quo? : (%,%) -> % if Coef has FIELD
--R random : () -> % if and(has(UTS,IntegerNumberSystem),has(Coef,Field))
--R rationalFunction : (%,Integer,Integer) -> Fraction Polynomial Coef if Coef has INTDOM
--R rationalFunction : (%,Integer) -> Fraction Polynomial Coef if Coef has INTDOM
--R reducedSystem : Matrix % -> Matrix Integer if and(has(UTS,LinearlyExplicitRingOver Integer),
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer) if an
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix UTS,vec: Vector UTS) if Coef has F
--R reducedSystem : Matrix % -> Matrix UTS if Coef has FIELD
--R ?rem? : (%,%) -> % if Coef has FIELD
--R retract : % -> Symbol if and(has(UTS,RetractableTo Symbol),has(Coef,Field))
--R retract : % -> Fraction Integer if and(has(UTS,RetractableTo Integer),has(Coef,Field))
--R retract : % -> Integer if and(has(UTS,RetractableTo Integer),has(Coef,Field))
--R retractIfCan : % -> Union(Symbol,"failed") if and(has(UTS,RetractableTo Symbol),has(Coef,Fie
```

```
--R retractIfCan : % -> Union(Fraction Integer,"failed") if and(has(UTS,RetractableTo Integer),has(Coef,Fiel
--R retractIfCan : % -> Union(Integer,"failed") if and(has(UTS,RetractableTo Integer),has(Coef,Field))
--R retractIfCan : % -> Union(UTS,"failed")
--R sec : % -> % if Coef has ALGEBRA FRAC INT
--R sech : % -> % if Coef has ALGEBRA FRAC INT
--R series : Stream Record(k: Integer,c: Coef) -> %
--R sign : % -> Integer if and(has(UTS,OrderedIntegralDomain),has(Coef,Field))
--R sin : % -> % if Coef has ALGEBRA FRAC INT
--R sinh : % -> % if Coef has ALGEBRA FRAC INT
--R sizeLess? : (%,%) -> Boolean if Coef has FIELD
--R solveLinearPolynomialEquation : (List SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> Unio
--R sqrt : % -> % if Coef has ALGEBRA FRAC INT
--R squareFree : % -> Factored % if Coef has FIELD
--R squareFreePart : % -> % if Coef has FIELD
--R squareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial % if and(has
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R tan : % -> % if Coef has ALGEBRA FRAC INT
--R tanh : % -> % if Coef has ALGEBRA FRAC INT
--R taylorIfCan : % -> Union(UTS,"failed")
--R terms : % -> Stream Record(k: Integer,c: Coef)
--R truncate : (%,Integer,Integer) -> %
--R unit? : % -> Boolean if Coef has INTDOM
--R unitCanonical : % -> % if Coef has INTDOM
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %) if Coef has INTDOM
--R variables : % -> List SingletonAsOrderedSet
--R wholePart : % -> UTS if and(has(UTS,EuclideanDomain),has(Coef,Field))
--R
--E 1

)spool
)lisp (bye)
```

———-

— UnivariateLaurentSeriesConstructorCategory.help —

```
=====================================================================
UnivariateLaurentSeriesConstructorCategory examples
=====================================================================

See Also:
o )show UnivariateLaurentSeriesConstructorCategory
```

———-

See:

⇐ “RetractableTo” (RETRACT) 2.0.18 on page 55

⇐ “UnivariateLaurentSeriesCategory” (ULSCAT) 17.0.202 on page 1448

Exports:

0	1	abs
acos	acosh	acot
acoth	acsc	acsch
approximate	asec	asech
asin	asinh	associates?
atan	atanh	ceiling
center	characteristic	charthRoot
coefficient	coerce	complete
conditionP	convert	cos
cosh	cot	coth
csc	csch	D
degree	denom	denominator
differentiate	divide	euclideanSize
eval	exp	expressIdealMember
exquo	extend	extendedEuclidean
factor	factorPolynomial	factorSquareFreePolynomial
floor	fractionPart	gcd
gcdPolynomial	hash	init
integrate	inv	latex
laurent	lcm	leadingCoefficient
leadingMonomial	log	map
max	min	monomial
monomial?	multiEuclidean	multiplyCoefficients
multiplyExponents	negative?	nextItem
nthRoot	numer	numerator
one?	order	patternMatch
pi	pole?	positive?
prime?	principalIdeal	random
rationalFunction	recip	reducedSystem
reductum	removeZeroes	retract
retractIfCan	sample	sec
sech	series	sign
sin	sinh	sizeLess?
solveLinearPolynomialEquation	sqrt	squareFree
squareFreePart	squareFreePolynomial	subtractIfCan
tan	tanh	taylor
taylorIfCan	taylorRep	terms
truncate	unit?	unitCanonical
unitNormal	variable	variables
wholePart	zero?	?*?
?**?	?+?	?-?
-?	?=?	?^?
?~=?	?/?	?<?
?<=?	?>?	?>=?
?.?	?quo?	?rem?

Attributes Exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.
- if #1 has Field then canonicalClosed where **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.
- if #1 has Field then canonicalUnitNormal where **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- if #1 has CommutativeRing then commutative("*") where **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- if #1 has IntegralDomain then noZeroDivisors where **noZeroDivisors** is true if $x*y \neq 0$ implies both x and y are non-zero.
- if #1 has Field then nil

These are directly exported but not implemented:

```
coerce : UTS -> %
degree : % -> Integer
laurent : (Integer,UTS) -> %
removeZeroes : % -> %
removeZeroes : (Integer,%) -> %
taylor : % -> UTS
taylorIfCan : % -> Union(UTS,"failed")
taylorRep : % -> UTS
```

These are implemented by this category:

```
retract : % -> UTS
retractIfCan : % -> Union(UTS,"failed")
zero? : % -> Boolean
```

These exports come from (p1448) UnivariateLaurentSeriesCategory(Coef:Ring)

```
0 : () -> %
1 : () -> %
acos : % -> % if Coef has ALGEBRA FRAC INT
acosh : % -> % if Coef has ALGEBRA FRAC INT
acot : % -> % if Coef has ALGEBRA FRAC INT
acoth : % -> % if Coef has ALGEBRA FRAC INT
acsc : % -> % if Coef has ALGEBRA FRAC INT
```

```
acsch : % -> % if Coef has ALGEBRA FRAC INT
approximate : (%,Integer) -> Coef
  if Coef has **: (Coef,Integer) -> Coef
  and Coef has coerce: Symbol -> Coef
asec : % -> % if Coef has ALGEBRA FRAC INT
asech : % -> % if Coef has ALGEBRA FRAC INT
asin : % -> % if Coef has ALGEBRA FRAC INT
asinh : % -> % if Coef has ALGEBRA FRAC INT
associates? : (%,%) -> Boolean if Coef has INTDOM
atan : % -> % if Coef has ALGEBRA FRAC INT
atanh : % -> % if Coef has ALGEBRA FRAC INT
center : % -> Coef
characteristic : () -> NonNegativeInteger
charthRoot : % -> Union(%,"failed")
  if and(OR(has(UTS,CharacteristicNonZero),
            and(has($,CharacteristicNonZero),
                has(UTS,PolynomialFactorizationExplicit))),
         has(Coef,Field))
  or Coef has CHARNZ
coefficient : (%,Integer) -> Coef
coerce : % -> % if Coef has INTDOM
coerce : Fraction Integer -> % if Coef has ALGEBRA FRAC INT
coerce : Coef -> % if Coef has COMRING
coerce : Integer -> %
coerce : % -> OutputForm
complete : % -> %
cos : % -> % if Coef has ALGEBRA FRAC INT
cosh : % -> % if Coef has ALGEBRA FRAC INT
cot : % -> % if Coef has ALGEBRA FRAC INT
coth : % -> % if Coef has ALGEBRA FRAC INT
csc : % -> % if Coef has ALGEBRA FRAC INT
csch : % -> % if Coef has ALGEBRA FRAC INT
D : % -> %
  if and(has(UTS,DifferentialRing),has(Coef,Field))
  or Coef has *: (Integer,Coef) -> Coef
D : (%,NonNegativeInteger) -> %
  if and(has(UTS,DifferentialRing),has(Coef,Field))
  or Coef has *: (Integer,Coef) -> Coef
D : (%,Symbol) -> %
  if and(has(UTS,PartialDifferentialRing Symbol),has(Coef,Field))
  or Coef has PDRING SYMBOL
  and Coef has *: (Integer,Coef) -> Coef
D : (%,List Symbol) -> %
  if and(has(UTS,PartialDifferentialRing Symbol),has(Coef,Field))
  or Coef has PDRING SYMBOL
  and Coef has *: (Integer,Coef) -> Coef
D : (%,Symbol,NonNegativeInteger) -> %
  if and(has(UTS,PartialDifferentialRing Symbol),has(Coef,Field))
  or Coef has PDRING SYMBOL
  and Coef has *: (Integer,Coef) -> Coef
```

```
D : (%,List Symbol,List NonNegativeInteger) -> %
  if and(has(UTS,PartialDifferentialRing Symbol),has(Coef,Field))
  or Coef has PDRING SYMBOL
  and Coef has *: (Integer,Coef) -> Coef
differentiate : (%,Symbol) -> %
  if and(has(UTS,PartialDifferentialRing Symbol),has(Coef,Field))
  or Coef has PDRING SYMBOL
  and Coef has *: (Integer,Coef) -> Coef
differentiate : (%,List Symbol) -> %
  if and(has(UTS,PartialDifferentialRing Symbol),has(Coef,Field))
  or Coef has PDRING SYMBOL
  and Coef has *: (Integer,Coef) -> Coef
differentiate : (%,Symbol,NonNegativeInteger) -> %
  if and(has(UTS,PartialDifferentialRing Symbol),has(Coef,Field))
  or Coef has PDRING SYMBOL
  and Coef has *: (Integer,Coef) -> Coef
differentiate : (%,List Symbol,List NonNegativeInteger) -> %
  if and(has(UTS,PartialDifferentialRing Symbol),has(Coef,Field))
  or Coef has PDRING SYMBOL
  and Coef has *: (Integer,Coef) -> Coef
differentiate : % -> %
  if and(has(UTS,DifferentialRing),has(Coef,Field))
  or Coef has *: (Integer,Coef) -> Coef
differentiate : (%,NonNegativeInteger) -> %
  if and(has(UTS,DifferentialRing),has(Coef,Field))
  or Coef has *: (Integer,Coef) -> Coef
divide : (%,%) -> Record(quotient: %,remainder: %)
  if Coef has FIELD
euclideanSize : % -> NonNegativeInteger if Coef has FIELD
eval : (%,Coef) -> Stream Coef
  if Coef has **: (Coef,Integer) -> Coef
exp : % -> % if Coef has ALGEBRA FRAC INT
expressIdealMember : (List %,%) -> Union(List %,"failed")
  if Coef has FIELD
exquo : (%,%) -> Union(%,"failed") if Coef has INTDOM
extend : (%,Integer) -> %
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
  if Coef has FIELD
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
  if Coef has FIELD
factor : % -> Factored % if Coef has FIELD
gcd : (%,%) -> % if Coef has FIELD
gcd : List % -> % if Coef has FIELD
gcdPolynomial :
 (SparseUnivariatePolynomial %,
  SparseUnivariatePolynomial %) ->
    SparseUnivariatePolynomial %
      if Coef has FIELD
hash : % -> SingleInteger
integrate : (%,Symbol) -> %
```

```
  if Coef has ACFS INT
  and Coef has PRIMCAT
  and Coef has TRANFUN
  and Coef has ALGEBRA FRAC INT
  or Coef has variables: Coef -> List Symbol
  and Coef has integrate: (Coef,Symbol) -> Coef
  and Coef has ALGEBRA FRAC INT
integrate : % -> % if Coef has ALGEBRA FRAC INT
inv : % -> % if Coef has FIELD
latex : % -> String
lcm : (%,%) -> % if Coef has FIELD
lcm : List % -> % if Coef has FIELD
leadingCoefficient : % -> Coef
leadingMonomial : % -> %
log : % -> % if Coef has ALGEBRA FRAC INT
map : ((Coef -> Coef),%) -> %
monomial : (%,List SingletonAsOrderedSet,List Integer) -> %
monomial : (%,SingletonAsOrderedSet,Integer) -> %
monomial : (Coef,Integer) -> %
monomial? : % -> Boolean
multiEuclidean : (List %,%) -> Union(List %,"failed")
  if Coef has FIELD
multiplyCoefficients : ((Integer -> Coef),%) -> %
multiplyExponents : (%,PositiveInteger) -> %
nthRoot : (%,Integer) -> % if Coef has ALGEBRA FRAC INT
one? : % -> Boolean
order : (%,Integer) -> Integer
order : % -> Integer
pi : () -> % if Coef has ALGEBRA FRAC INT
pole? : % -> Boolean
prime? : % -> Boolean if Coef has FIELD
principalIdeal : List % -> Record(coef: List %,generator: %)
  if Coef has FIELD
rationalFunction : (%,Integer) -> Fraction Polynomial Coef
  if Coef has INTDOM
rationalFunction : (%,Integer,Integer) -> Fraction Polynomial Coef
  if Coef has INTDOM
recip : % -> Union(%,"failed")
reductum : % -> %
sample : () -> %
sec : % -> % if Coef has ALGEBRA FRAC INT
sech : % -> % if Coef has ALGEBRA FRAC INT
series : Stream Record(k: Integer,c: Coef) -> %
sin : % -> % if Coef has ALGEBRA FRAC INT
sinh : % -> % if Coef has ALGEBRA FRAC INT
sizeLess? : (%,%) -> Boolean if Coef has FIELD
squareFree : % -> Factored % if Coef has FIELD
squareFreePart : % -> % if Coef has FIELD
sqrt : % -> % if Coef has ALGEBRA FRAC INT
subtractIfCan : (%,%) -> Union(%,"failed")
```

```
tan : % -> % if Coef has ALGEBRA FRAC INT
tanh : % -> % if Coef has ALGEBRA FRAC INT
terms : % -> Stream Record(k: Integer,c: Coef)
truncate : (%,Integer,Integer) -> %
truncate : (%,Integer) -> %
unit? : % -> Boolean if Coef has INTDOM
unitCanonical : % -> % if Coef has INTDOM
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
  if Coef has INTDOM
variable : % -> Symbol
variables : % -> List SingletonAsOrderedSet
?.? : (%,Integer) -> Coef
?**? : (%,Integer) -> % if Coef has FIELD
?**? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
?**? : (%,NonNegativeInteger) -> %
?^? : (%,Integer) -> % if Coef has FIELD
?^? : (%,NonNegativeInteger) -> %
?/? : (%,%) -> % if Coef has FIELD
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,%) -> % if Coef has ALGEBRA FRAC INT
?^? : (%,PositiveInteger) -> %
?*? : (Integer,%) -> %
?*? : (Coef,%) -> %
?*? : (%,Coef) -> %
?*? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
?*? : (Fraction Integer,%) -> % if Coef has ALGEBRA FRAC INT
?/? : (%,Coef) -> % if Coef has FIELD
?.? : (%,%) -> % if Integer has SGROUP
?quo? : (%,%) -> % if Coef has FIELD
?rem? : (%,%) -> % if Coef has FIELD
```

These exports come from (p1376) QuotientFieldCategory(UTS)
where UTS:UnivariateLaurentSeriesCategory(Coef:Ring)

```
abs : % -> %
  if and(has(UTS,OrderedIntegralDomain),has(Coef,Field))
ceiling : % -> UTS
  if and(has(UTS,IntegerNumberSystem),has(Coef,Field))
conditionP : Matrix % -> Union(Vector %,"failed")
  if and(and(has($,CharacteristicNonZero),
             has(UTS,PolynomialFactorizationExplicit)),
         has(Coef,Field))
```

```
coerce : Symbol -> %
  if and(has(UTS,RetractableTo Symbol),has(Coef,Field))
convert : % -> Pattern Integer
  if and(has(UTS,ConvertibleTo Pattern Integer),has(Coef,Field))
convert : % -> Pattern Float
  if and(has(UTS,ConvertibleTo Pattern Float),has(Coef,Field)\hfill\\
convert : % -> InputForm
  if and(has(UTS,ConvertibleTo InputForm),has(Coef,Field))
convert : % -> Float
  if and(has(UTS,RealConstant),has(Coef,Field))
convert : % -> DoubleFloat
  if and(has(UTS,RealConstant),has(Coef,Field))
D : (%,(UTS -> UTS),NonNegativeInteger) -> %
  if Coef has FIELD
D : (%,(UTS -> UTS)) -> % if Coef has FIELD
denom : % -> UTS if Coef has FIELD
denominator : % -> % if Coef has FIELD
differentiate : (%,(UTS -> UTS)) -> % if Coef has FIELD
differentiate : (%,(UTS -> UTS),NonNegativeInteger) -> %
  if Coef has FIELD
eval : (%,Equation UTS) -> %
  if and(has(UTS,Evalable UTS),has(Coef,Field))
eval : (%,List Symbol,List UTS) -> %
  if and(has(UTS,InnerEvalable(Symbol,UTS)),has(Coef,Field))
eval : (%,List Equation UTS) -> %
  if and(has(UTS,Evalable UTS),has(Coef,Field))
eval : (%,UTS,UTS) -> %
  if and(has(UTS,Evalable UTS),has(Coef,Field))
eval : (%,List UTS,List UTS) -> %
  if and(has(UTS,Evalable UTS),has(Coef,Field))
eval : (%,Symbol,UTS) -> %
  if and(has(UTS,InnerEvalable(Symbol,UTS)),has(Coef,Field))
factorPolynomial :
  SparseUnivariatePolynomial % ->
    Factored SparseUnivariatePolynomial %
     if and(has(UTS,PolynomialFactorizationExplicit),has(Coef,Field))
factorSquareFreePolynomial :
  SparseUnivariatePolynomial % ->
    Factored SparseUnivariatePolynomial %
     if and(has(UTS,PolynomialFactorizationExplicit),has(Coef,Field))
floor : % -> UTS
  if and(has(UTS,IntegerNumberSystem),has(Coef,Field))
fractionPart : % -> %
  if and(has(UTS,EuclideanDomain),has(Coef,Field))
init : () -> % if and(has(UTS,StepThrough),has(Coef,Field))
map : ((UTS -> UTS),%) -> % if Coef has FIELD
max : (%,%) -> % if and(has(UTS,OrderedSet),has(Coef,Field))
min : (%,%) -> % if and(has(UTS,OrderedSet),has(Coef,Field))
negative? : % -> Boolean
  if and(has(UTS,OrderedIntegralDomain),has(Coef,Field))
```

```
nextItem : % -> Union(%,"failed")
  if and(has(UTS,StepThrough),has(Coef,Field))
numer : % -> UTS if Coef has FIELD
numerator : % -> % if Coef has FIELD
patternMatch :
 (%,Pattern Integer,PatternMatchResult(Integer,%)) ->
   PatternMatchResult(Integer,%)
     if and(has(UTS,PatternMatchable Integer),has(Coef,Field))
patternMatch :
 (%,Pattern Float,PatternMatchResult(Float,%)) ->
   PatternMatchResult(Float,%)
     if and(has(UTS,PatternMatchable Float),has(Coef,Field))
positive? : % -> Boolean
  if and(has(UTS,OrderedIntegralDomain),has(Coef,Field))
random : () -> %
  if and(has(UTS,IntegerNumberSystem),has(Coef,Field))
reducedSystem :
 (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer)
  if and(has(UTS,LinearlyExplicitRingOver Integer),has(Coef,Field))
reducedSystem : Matrix % -> Matrix Integer
  if and(has(UTS,LinearlyExplicitRingOver Integer),has(Coef,Field))
reducedSystem :
 (Matrix %,Vector %) -> Record(mat: Matrix UTS,vec: Vector UTS)
   if Coef has FIELD
reducedSystem : Matrix % -> Matrix UTS if Coef has FIELD
retract : % -> Symbol
  if and(has(UTS,RetractableTo Symbol),has(Coef,Field))
retract : % -> Integer
  if and(has(UTS,RetractableTo Integer),has(Coef,Field))
retract : % -> Fraction Integer
  if and(has(UTS,RetractableTo Integer),has(Coef,Field))
retractIfCan : % -> Union(Fraction Integer,"failed")
  if and(has(UTS,RetractableTo Integer),has(Coef,Field))
retractIfCan : % -> Union(Symbol,"failed")
  if and(has(UTS,RetractableTo Symbol),has(Coef,Field))
retractIfCan : % -> Union(Integer,"failed")
  if and(has(UTS,RetractableTo Integer),has(Coef,Field))
sign : % -> Integer
  if and(has(UTS,OrderedIntegralDomain),has(Coef,Field))
solveLinearPolynomialEquation :
  (List SparseUnivariatePolynomial %,
   SparseUnivariatePolynomial %) ->
     Union(List SparseUnivariatePolynomial %,"failed")
      if and(has(UTS,PolynomialFactorizationExplicit),has(Coef,Field))
squareFreePolynomial :
  SparseUnivariatePolynomial % ->
     Factored SparseUnivariatePolynomial %
       if and(has(UTS,PolynomialFactorizationExplicit),has(Coef,Field))
wholePart : % -> UTS
  if and(has(UTS,EuclideanDomain),has(Coef,Field))
```

```
 ?*? : (UTS,%) -> % if Coef has FIELD
 ?*? : (%,UTS) -> % if Coef has FIELD
 ?<? : (%,%) -> Boolean if and(has(UTS,OrderedSet),has(Coef,Field))
 ?/? : (UTS,UTS) -> % if Coef has FIELD
 ?.? : (%,UTS) -> % if and(has(UTS,Eltable(UTS,UTS)),has(Coef,Field))
 ?<=? : (%,%) -> Boolean if and(has(UTS,OrderedSet),has(Coef,Field))
 ?>? : (%,%) -> Boolean if and(has(UTS,OrderedSet),has(Coef,Field))
 ?>=? : (%,%) -> Boolean if and(has(UTS,OrderedSet),has(Coef,Field))
```

— category ULSCCAT UnivariateLaurentSeriesConstructorCategory —

```
)abbrev category ULSCCAT UnivariateLaurentSeriesConstructorCategory
++ Author: Clifton J. Williamson
++ Date Created: 6 February 1990
++ Date Last Updated: 10 May 1990
++ Basic Operations:
++ Related Domains:
++ Also See:
++ AMS Classifications:
++ Keywords: series, Laurent, Taylor
++ Examples:
++ References:
++ Description:
++ This is a category of univariate Laurent series constructed from
++ univariate Taylor series.  A Laurent series is represented by a pair
++ \spad{[n,f(x)]}, where n is an arbitrary integer and \spad{f(x)}
++ is a Taylor series.  This pair represents the Laurent series
++ \spad{x**n * f(x)}.

UnivariateLaurentSeriesConstructorCategory(Coef,UTS):_
 Category == Definition where
  Coef: Ring
  UTS : UnivariateTaylorSeriesCategory Coef
  I ==> Integer

  Definition ==> Join(UnivariateLaurentSeriesCategory(Coef),_
                      RetractableTo UTS) with
    laurent: (I,UTS) -> %
      ++ \spad{laurent(n,f(x))} returns \spad{x**n * f(x)}.
    degree: % -> I
      ++ \spad{degree(f(x))} returns the degree of the lowest order term of
      ++ \spad{f(x)}, which may have zero as a coefficient.
    taylorRep: % -> UTS
      ++ \spad{taylorRep(f(x))} returns \spad{g(x)}, where
      ++ \spad{f = x**n * g(x)} is represented by \spad{[n,g(x)]}.
    removeZeroes: % -> %
      ++ \spad{removeZeroes(f(x))} removes leading zeroes from the
      ++ representation of the Laurent series \spad{f(x)}.
```

```
      ++ A Laurent series is represented by (1) an exponent and
      ++ (2) a Taylor series which may have leading zero coefficients.
      ++ When the Taylor series has a leading zero coefficient, the
      ++ 'leading zero' is removed from the Laurent series as follows:
      ++ the series is rewritten by increasing the exponent by 1 and
      ++ dividing the Taylor series by its variable.
      ++ Note that \spad{removeZeroes(f)} removes all leading zeroes from f
    removeZeroes: (I,%) -> %
      ++ \spad{removeZeroes(n,f(x))} removes up to n leading zeroes from
      ++ the Laurent series \spad{f(x)}.
      ++ A Laurent series is represented by (1) an exponent and
      ++ (2) a Taylor series which may have leading zero coefficients.
      ++ When the Taylor series has a leading zero coefficient, the
      ++ 'leading zero' is removed from the Laurent series as follows:
      ++ the series is rewritten by increasing the exponent by 1 and
      ++ dividing the Taylor series by its variable.
    coerce: UTS -> %
      ++ \spad{coerce(f(x))} converts the Taylor series \spad{f(x)} to a
      ++ Laurent series.
    taylor: % -> UTS
      ++ taylor(f(x)) converts the Laurent series f(x) to a Taylor series,
      ++ if possible.  Error: if this is not possible.
    taylorIfCan: % -> Union(UTS,"failed")
      ++ \spad{taylorIfCan(f(x))} converts the Laurent series \spad{f(x)}
      ++ to a Taylor series, if possible. If this is not possible,
      ++ "failed" is returned.
    if Coef has Field then QuotientFieldCategory(UTS)
      --++ the quotient field of univariate Taylor series over a field is
      --++ the field of Laurent series

   add

    zero? x == zero? taylorRep x

    retract(x:%):UTS == taylor x

    retractIfCan(x:%):Union(UTS,"failed") == taylorIfCan x
```

———-

— ULSCCAT.dotabb —

```
"ULSCCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ULSCCAT"];
"ULSCCAT" -> "ULSCAT"
```

———-

— **ULSCCAT.dotfull** —

```
"UnivariateLaurentSeriesConstructorCategory(a:Ring,b:UnivariateTaylorSeriesCategory(Ring))"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=ULSCCAT"];
"UnivariateLaurentSeriesConstructorCategory(a:Ring,b:UnivariateTaylorSeriesCategory(Ring))"
  -> "UnivariateLaurentSeriesCategory(a:Ring)"
```

————-

— **ULSCCAT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"UnivariateLaurentSeriesConstructorCategory(a:Ring,b:UnivariateTaylorSeriesCategory(Ring))"
 [color=lightblue];
"UnivariateLaurentSeriesConstructorCategory(a:Ring,b:UnivariateTaylorSeriesCategory(Ring))"
  -> "UnivariateLaurentSeriesCategory(a:Ring)"

"UnivariateLaurentSeriesCategory(a:Ring)" [color=lightblue];
"UnivariateLaurentSeriesCategory(a:Ring)" ->
    "UnivariatePowerSeriesCategory(a:Ring,Integer)"
"UnivariateLaurentSeriesCategory(a:Ring)" ->
    "FIELD..."
"UnivariateLaurentSeriesCategory(a:Ring)" ->
    "RADCAT..."
"UnivariateLaurentSeriesCategory(a:Ring)" ->
    "TRANFUN..."

"UnivariatePowerSeriesCategory(a:Ring,Integer)" [color=seagreen];
"UnivariatePowerSeriesCategory(a:Ring,Integer)" ->
    "UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)"

"UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)"
 [color=lightblue];
"UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)" ->
 "PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:SingletonAsOrderedSet)"

"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:SingletonAsOrderedSet)"
 [color=seagreen];
"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:SingletonAsOrderedSet)"
  -> "PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)"

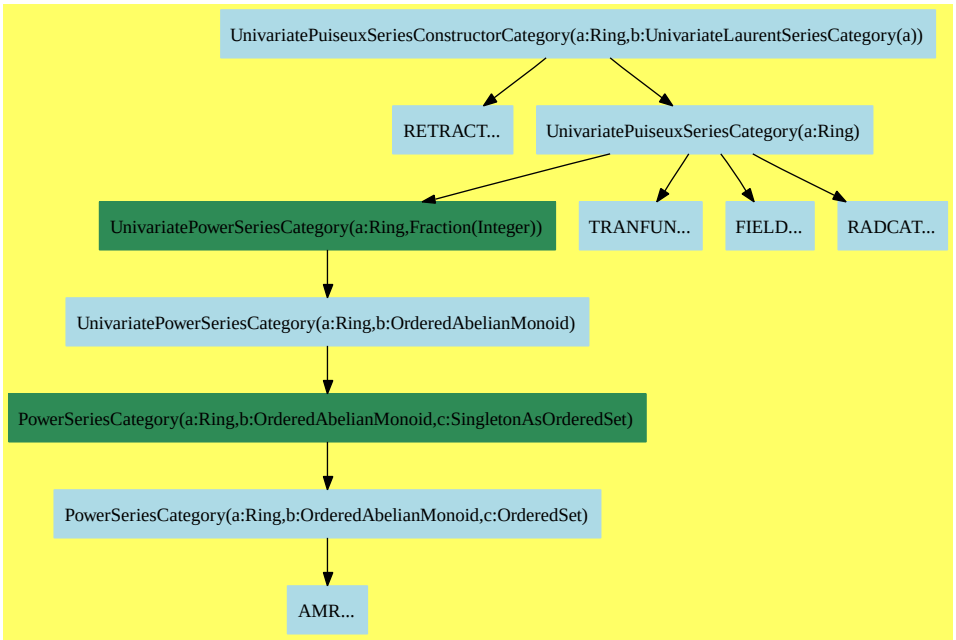
"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)"
 [color=lightblue];
```

```
"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)" ->
    "AMR..."

"AMR..." [color=lightblue];
"FIELD..." [color=lightblue];
"TRANFUN..." [color=lightblue];
"RADCAT..." [color=lightblue];
}
```

———-

18.0.212 UnivariatePuiseuxSeriesConstructorCategory (UPXSCCA)



— UnivariatePuiseuxSeriesConstructorCategory.input —

```
)set break resume
)sys rm -f UnivariatePuiseuxSeriesConstructorCategory.output
)spool UnivariatePuiseuxSeriesConstructorCategory.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 1
)show UnivariatePuiseuxSeriesConstructorCategory
--R UnivariatePuiseuxSeriesConstructorCategory(Coef: Ring,ULS: UnivariateLaurentSeriesCategory t#1)  is a ca
--R Abbreviation for UnivariatePuiseuxSeriesConstructorCategory is UPXSCCA
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for UPXSCCA
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (Coef,%) -> %                  ?*? : (%,Coef) -> %
--R ?*? : (%,%) -> %                     ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %       ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                     ?-? : (%,%) -> %
--R -? : % -> %                          ?=? : (%,%) -> Boolean
--R 1 : () -> %                          0 : () -> %
--R ?^? : (%,PositiveInteger) -> %       center : % -> Coef
--R coerce : ULS -> %                    coerce : Integer -> %
--R coerce : % -> OutputForm             complete : % -> %
--R degree : % -> Fraction Integer       hash : % -> SingleInteger
--R latex : % -> String                  laurent : % -> ULS
--R laurentRep : % -> ULS                leadingCoefficient : % -> Coef
--R leadingMonomial : % -> %             map : ((Coef -> Coef),%) -> %
--R monomial? : % -> Boolean             one? : % -> Boolean
--R order : % -> Fraction Integer        pole? : % -> Boolean
--R recip : % -> Union(%,"failed")       reductum : % -> %
--R retract : % -> ULS                   sample : () -> %
--R variable : % -> Symbol               zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
--R ?*? : (Fraction Integer,%) -> % if Coef has ALGEBRA FRAC INT
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
--R ?**? : (%,%) -> % if Coef has ALGEBRA FRAC INT
--R ?**? : (%,Integer) -> % if Coef has FIELD
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (%,%) -> % if Coef has FIELD
--R ?/? : (%,Coef) -> % if Coef has FIELD
--R D : % -> % if Coef has *: (Fraction Integer,Coef) -> Coef
--R D : (%,NonNegativeInteger) -> % if Coef has *: (Fraction Integer,Coef) -> Coef
--R D : (%,Symbol) -> % if Coef has PDRING SYMBOL and Coef has *: (Fraction Integer,Coef) -> Coef
--R D : (%,List Symbol) -> % if Coef has PDRING SYMBOL and Coef has *: (Fraction Integer,Coef) -> Coef
--R D : (%,Symbol,NonNegativeInteger) -> % if Coef has PDRING SYMBOL and Coef has *: (Fraction Integer,Coef
--R D : (%,List Symbol,List NonNegativeInteger) -> % if Coef has PDRING SYMBOL and Coef has *: (Fraction Int
--R ?^? : (%,Integer) -> % if Coef has FIELD
--R ?^? : (%,NonNegativeInteger) -> %
--R acos : % -> % if Coef has ALGEBRA FRAC INT
--R acosh : % -> % if Coef has ALGEBRA FRAC INT
--R acot : % -> % if Coef has ALGEBRA FRAC INT
--R acoth : % -> % if Coef has ALGEBRA FRAC INT
--R acsc : % -> % if Coef has ALGEBRA FRAC INT
--R acsch : % -> % if Coef has ALGEBRA FRAC INT
```

```
--R approximate : (%,Fraction Integer) -> Coef if Coef has **: (Coef,Fraction Integer) -> Coef a
--R asec : % -> % if Coef has ALGEBRA FRAC INT
--R asech : % -> % if Coef has ALGEBRA FRAC INT
--R asin : % -> % if Coef has ALGEBRA FRAC INT
--R asinh : % -> % if Coef has ALGEBRA FRAC INT
--R associates? : (%,%) -> Boolean if Coef has INTDOM
--R atan : % -> % if Coef has ALGEBRA FRAC INT
--R atanh : % -> % if Coef has ALGEBRA FRAC INT
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if Coef has CHARNZ
--R coefficient : (%,Fraction Integer) -> Coef
--R coerce : % -> % if Coef has INTDOM
--R coerce : Fraction Integer -> % if Coef has ALGEBRA FRAC INT
--R coerce : Coef -> % if Coef has COMRING
--R cos : % -> % if Coef has ALGEBRA FRAC INT
--R cosh : % -> % if Coef has ALGEBRA FRAC INT
--R cot : % -> % if Coef has ALGEBRA FRAC INT
--R coth : % -> % if Coef has ALGEBRA FRAC INT
--R csc : % -> % if Coef has ALGEBRA FRAC INT
--R csch : % -> % if Coef has ALGEBRA FRAC INT
--R differentiate : % -> % if Coef has *: (Fraction Integer,Coef) -> Coef
--R differentiate : (%,NonNegativeInteger) -> % if Coef has *: (Fraction Integer,Coef) -> Coef
--R differentiate : (%,Symbol) -> % if Coef has PDRING SYMBOL and Coef has *: (Fraction Integer,
--R differentiate : (%,List Symbol) -> % if Coef has PDRING SYMBOL and Coef has *: (Fraction Int
--R differentiate : (%,Symbol,NonNegativeInteger) -> % if Coef has PDRING SYMBOL and Coef has *:
--R differentiate : (%,List Symbol,List NonNegativeInteger) -> % if Coef has PDRING SYMBOL and C
--R divide : (%,%) -> Record(quotient: %,remainder: %) if Coef has FIELD
--R ?.? : (%,%) -> % if Fraction Integer has SGROUP
--R ?.? : (%,Fraction Integer) -> Coef
--R euclideanSize : % -> NonNegativeInteger if Coef has FIELD
--R eval : (%,Coef) -> Stream Coef if Coef has **: (Coef,Fraction Integer) -> Coef
--R exp : % -> % if Coef has ALGEBRA FRAC INT
--R expressIdealMember : (List %,%) -> Union(List %,"failed") if Coef has FIELD
--R exquo : (%,%) -> Union(%,"failed") if Coef has INTDOM
--R extend : (%,Fraction Integer) -> %
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %) if Coef has FIELD
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed") if Coef has FIELD
--R factor : % -> Factored % if Coef has FIELD
--R gcd : (%,%) -> % if Coef has FIELD
--R gcd : List % -> % if Coef has FIELD
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivari
--R integrate : (%,Symbol) -> % if Coef has ACFS INT and Coef has PRIMCAT and Coef has TRANFUN a
--R integrate : % -> % if Coef has ALGEBRA FRAC INT
--R inv : % -> % if Coef has FIELD
--R laurentIfCan : % -> Union(ULS,"failed")
--R lcm : (%,%) -> % if Coef has FIELD
--R lcm : List % -> % if Coef has FIELD
--R log : % -> % if Coef has ALGEBRA FRAC INT
--R monomial : (%,List SingletonAsOrderedSet,List Fraction Integer) -> %
--R monomial : (%,SingletonAsOrderedSet,Fraction Integer) -> %
```

```
--R monomial : (Coef,Fraction Integer) -> %
--R multiEuclidean : (List %,%) -> Union(List %,"failed") if Coef has FIELD
--R multiplyExponents : (%,Fraction Integer) -> %
--R multiplyExponents : (%,PositiveInteger) -> %
--R nthRoot : (%,Integer) -> % if Coef has ALGEBRA FRAC INT
--R order : (%,Fraction Integer) -> Fraction Integer
--R pi : () -> % if Coef has ALGEBRA FRAC INT
--R prime? : % -> Boolean if Coef has FIELD
--R principalIdeal : List % -> Record(coef: List %,generator: %) if Coef has FIELD
--R puiseux : (Fraction Integer,ULS) -> %
--R ?quo? : (%,%) -> % if Coef has FIELD
--R rationalPower : % -> Fraction Integer
--R ?rem? : (%,%) -> % if Coef has FIELD
--R retractIfCan : % -> Union(ULS,"failed")
--R sec : % -> % if Coef has ALGEBRA FRAC INT
--R sech : % -> % if Coef has ALGEBRA FRAC INT
--R series : (NonNegativeInteger,Stream Record(k: Fraction Integer,c: Coef)) -> %
--R sin : % -> % if Coef has ALGEBRA FRAC INT
--R sinh : % -> % if Coef has ALGEBRA FRAC INT
--R sizeLess? : (%,%) -> Boolean if Coef has FIELD
--R sqrt : % -> % if Coef has ALGEBRA FRAC INT
--R squareFree : % -> Factored % if Coef has FIELD
--R squareFreePart : % -> % if Coef has FIELD
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R tan : % -> % if Coef has ALGEBRA FRAC INT
--R tanh : % -> % if Coef has ALGEBRA FRAC INT
--R terms : % -> Stream Record(k: Fraction Integer,c: Coef)
--R truncate : (%,Fraction Integer,Fraction Integer) -> %
--R truncate : (%,Fraction Integer) -> %
--R unit? : % -> Boolean if Coef has INTDOM
--R unitCanonical : % -> % if Coef has INTDOM
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %) if Coef has INTDOM
--R variables : % -> List SingletonAsOrderedSet
--R
--E 1

)spool
)lisp (bye)
```

———-

— UnivariatePuiseuxSeriesConstructorCategory.help —

```
=====================================================================
UnivariatePuiseuxSeriesConstructorCategory examples
=====================================================================

See Also:
o )show UnivariatePuiseuxSeriesConstructorCategory
```

———-

See:

⇐ "RetractableTo" (RETRACT) 2.0.18 on page 55
⇐ "UnivariatePuiseuxSeriesCategory" (UPXSCAT) 17.0.203 on page 1460

Exports:

0	1	acos	acosh
acot	acoth	acsc	acsch
approximate	asec	asech	asin
asinh	associates?	atan	atanh
center	characteristic	charthRoot	coefficient
coerce	complete	cos	cosh
cot	coth	csc	csch
D	degree	differentiate	divide
euclideanSize	eval	exp	expressIdealMember
exquo	extend	extendedEuclidean	factor
gcd	gcdPolynomial	hash	integrate
inv	latex	laurent	laurentIfCan
laurentRep	lcm	leadingCoefficient	leadingMonomial
log	map	monomial	monomial?
multiEuclidean	multiplyExponents	nthRoot	one?
order	pi	pole?	prime?
principalIdeal	puiseux	rationalPower	recip
reductum	retract	retractIfCan	sample
sec	sech	series	sin
sinh	sizeLess?	sqrt	squareFree
squareFreePart	subtractIfCan	tan	tanh
terms	truncate	unit?	unitCanonical
unitNormal	variable	variables	zero?
?.?	?*?	?**?	?+?
?-?	-?	?=?	?^?
?~=?	?/?	?quo?	?rem?

Attributes Exported:

- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.
- if #1 has Field then canonicalsClosed where

- **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.
- if #1 has Field then canonicalUnitNormal where **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- if #1 has IntegralDomain then noZeroDivisors where **noZeroDivisors** is true if $x*y \neq 0$ implies both x and y are non-zero.
- if #1 has CommutativeRing then commututive("*") where **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.

These are directly exported but not implemented:

```
coerce : ULS -> %
degree : % -> Fraction Integer
laurent : % -> ULS
laurentIfCan : % -> Union(ULS,"failed")
laurentRep : % -> ULS
puiseux : (Fraction Integer,ULS) -> %
rationalPower : % -> Fraction Integer
```

These are implemented by this category:

```
retract : % -> ULS
retractIfCan : % -> Union(ULS,"failed")
zero? : % -> Boolean
```

These exports come from (p1460) UnivariatePuiseuxSeriesCategory(Coef:Ring):

```
0 : () -> %
1 : () -> %
approximate : (%,Fraction Integer) -> Coef
  if Coef has **: (Coef,Fraction Integer) -> Coef
  and Coef has coerce: Symbol -> Coef
associates? : (%,%) -> Boolean if Coef has INTDOM
acos : % -> % if Coef has ALGEBRA FRAC INT
acosh : % -> % if Coef has ALGEBRA FRAC INT
acot : % -> % if Coef has ALGEBRA FRAC INT
acoth : % -> % if Coef has ALGEBRA FRAC INT
acsc : % -> % if Coef has ALGEBRA FRAC INT
acsch : % -> % if Coef has ALGEBRA FRAC INT
asec : % -> % if Coef has ALGEBRA FRAC INT
asech : % -> % if Coef has ALGEBRA FRAC INT
asin : % -> % if Coef has ALGEBRA FRAC INT
asinh : % -> % if Coef has ALGEBRA FRAC INT
atan : % -> % if Coef has ALGEBRA FRAC INT
atanh : % -> % if Coef has ALGEBRA FRAC INT
```

```
center : % -> Coef
characteristic : () -> NonNegativeInteger
charthRoot : % -> Union(%,"failed") if Coef has CHARNZ
coefficient : (%,Fraction Integer) -> Coef
coerce : % -> % if Coef has INTDOM
coerce : Fraction Integer -> % if Coef has ALGEBRA FRAC INT
coerce : Coef -> % if Coef has COMRING
coerce : Integer -> %
coerce : % -> OutputForm
complete : % -> %
cos : % -> % if Coef has ALGEBRA FRAC INT
cosh : % -> % if Coef has ALGEBRA FRAC INT
cot : % -> % if Coef has ALGEBRA FRAC INT
coth : % -> % if Coef has ALGEBRA FRAC INT
csc : % -> % if Coef has ALGEBRA FRAC INT
csch : % -> % if Coef has ALGEBRA FRAC INT
D : % -> % if Coef has *: (Fraction Integer,Coef) -> Coef
D : (%,NonNegativeInteger) -> %
  if Coef has *: (Fraction Integer,Coef) -> Coef
D : (%,Symbol) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Fraction Integer,Coef) -> Coef
D : (%,List Symbol) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Fraction Integer,Coef) -> Coef
D : (%,Symbol,NonNegativeInteger) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Fraction Integer,Coef) -> Coef
D : (%,List Symbol,List NonNegativeInteger) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Fraction Integer,Coef) -> Coef
differentiate : (%,Symbol) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Fraction Integer,Coef) -> Coef
differentiate : (%,List Symbol) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Fraction Integer,Coef) -> Coef
differentiate : (%,Symbol,NonNegativeInteger) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Fraction Integer,Coef) -> Coef
differentiate : (%,List Symbol,List NonNegativeInteger) -> %
  if Coef has PDRING SYMBOL
  and Coef has *: (Fraction Integer,Coef) -> Coef
differentiate : % -> % if Coef has *: (Fraction Integer,Coef) -> Coef
differentiate : (%,NonNegativeInteger) -> %
  if Coef has *: (Fraction Integer,Coef) -> Coef
divide : (%,%) -> Record(quotient: %,remainder: %) if Coef has FIELD
euclideanSize : % -> NonNegativeInteger if Coef has FIELD
eval : (%,Coef) -> Stream Coef
  if Coef has **: (Coef,Fraction Integer) -> Coef
```

```
exp : % -> % if Coef has ALGEBRA FRAC INT
expressIdealMember : (List %,%) -> Union(List %,"failed")
  if Coef has FIELD
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
  if Coef has FIELD
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
  if Coef has FIELD
exquo : (%,%) -> Union(%,"failed") if Coef has INTDOM
extend : (%,Fraction Integer) -> %
factor : % -> Factored % if Coef has FIELD
gcd : (%,%) -> % if Coef has FIELD
gcd : List % -> % if Coef has FIELD
gcdPolynomial :
  (SparseUnivariatePolynomial %,
   SparseUnivariatePolynomial %) ->
     SparseUnivariatePolynomial %
       if Coef has FIELD
hash : % -> SingleInteger
integrate : (%,Symbol) -> %
  if Coef has ACFS INT
  and Coef has PRIMCAT
  and Coef has TRANFUN
  and Coef has ALGEBRA FRAC INT
  or Coef has variables: Coef -> List Symbol
  and Coef has integrate: (Coef,Symbol) -> Coef
  and Coef has ALGEBRA FRAC INT
integrate : % -> % if Coef has ALGEBRA FRAC INT
inv : % -> % if Coef has FIELD
latex : % -> String
lcm : (%,%) -> % if Coef has FIELD
lcm : List % -> % if Coef has FIELD
leadingCoefficient : % -> Coef
leadingMonomial : % -> %
log : % -> % if Coef has ALGEBRA FRAC INT
map : ((Coef -> Coef),%) -> %
monomial : (%,List SingletonAsOrderedSet,List Fraction Integer) -> %
monomial : (%,SingletonAsOrderedSet,Fraction Integer) -> %
monomial : (Coef,Fraction Integer) -> %
monomial? : % -> Boolean
multiEuclidean : (List %,%) -> Union(List %,"failed") if Coef has FIELD
multiplyExponents : (%,PositiveInteger) -> %
multiplyExponents : (%,Fraction Integer) -> %
nthRoot : (%,Integer) -> % if Coef has ALGEBRA FRAC INT
one? : % -> Boolean
order : (%,Fraction Integer) -> Fraction Integer
order : % -> Fraction Integer
pi : () -> % if Coef has ALGEBRA FRAC INT
pole? : % -> Boolean
prime? : % -> Boolean if Coef has FIELD
principalIdeal : List % -> Record(coef: List %,generator: %)
```

```
  if Coef has FIELD
recip : % -> Union(%,"failed")
reductum : % -> %
sample : () -> %
sec : % -> % if Coef has ALGEBRA FRAC INT
sech : % -> % if Coef has ALGEBRA FRAC INT
series :
  (NonNegativeInteger,Stream Record(k: Fraction Integer,c: Coef)) -> %
sin : % -> % if Coef has ALGEBRA FRAC INT
sinh : % -> % if Coef has ALGEBRA FRAC INT
sizeLess? : (%,%) -> Boolean if Coef has FIELD
sqrt : % -> % if Coef has ALGEBRA FRAC INT
squareFree : % -> Factored % if Coef has FIELD
squareFreePart : % -> % if Coef has FIELD
subtractIfCan : (%,%) -> Union(%,"failed")
tan : % -> % if Coef has ALGEBRA FRAC INT
tanh : % -> % if Coef has ALGEBRA FRAC INT
terms : % -> Stream Record(k: Fraction Integer,c: Coef)
truncate : (%,Fraction Integer,Fraction Integer) -> %
truncate : (%,Fraction Integer) -> %
unit? : % -> Boolean if Coef has INTDOM
unitCanonical : % -> % if Coef has INTDOM
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
  if Coef has INTDOM
variable : % -> Symbol
variables : % -> List SingletonAsOrderedSet
?**? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
?**? : (%,Integer) -> % if Coef has FIELD
?^? : (%,Integer) -> % if Coef has FIELD
?/? : (%,%) -> % if Coef has FIELD
?**? : (%,%) -> % if Coef has ALGEBRA FRAC INT
?**? : (%,NonNegativeInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (%,%) -> %
?-? : (%,%) -> %
?**? : (%,PositiveInteger) -> %
?^? : (%,PositiveInteger) -> %
?*? : (Integer,%) -> %
?*? : (Coef,%) -> %
?*? : (%,Coef) -> %
?*? : (%,Fraction Integer) -> % if Coef has ALGEBRA FRAC INT
?*? : (Fraction Integer,%) -> % if Coef has ALGEBRA FRAC INT
?/? : (%,Coef) -> % if Coef has FIELD
-? : % -> %
?.? : (%,%) -> % if Fraction Integer has SGROUP
```

```
 ?.? : (%,Fraction Integer) -> Coef
 ?quo? : (%,%) -> % if Coef has FIELD
 ?rem? : (%,%) -> % if Coef has FIELD
```

— category UPXSCCA UnivariatePuiseuxSeriesConstructorCategory

—

```
)abbrev category UPXSCCA UnivariatePuiseuxSeriesConstructorCategory
++ Author: Clifton J. Williamson
++ Date Created: 6 February 1990
++ Date Last Updated: 22 March 1990
++ Basic Operations:
++ Related Domains:
++ Also See:
++ AMS Classifications:
++ Keywords: series, Puiseux, Laurent
++ Examples:
++ References:
++ Description:
++ This is a category of univariate Puiseux series constructed
++ from univariate Laurent series.  A Puiseux series is represented
++ by a pair \spad{[r,f(x)]}, where r is a positive rational number and
++ \spad{f(x)} is a Laurent series.  This pair represents the Puiseux
++ series \spad{f(x^r)}.

UnivariatePuiseuxSeriesConstructorCategory(Coef,ULS):_
 Category == Definition where
  Coef : Ring
  ULS  : UnivariateLaurentSeriesCategory Coef
  I  ==> Integer
  RN ==> Fraction Integer

  Definition ==> Join(UnivariatePuiseuxSeriesCategory(Coef),_
                      RetractableTo ULS) with
    puiseux: (RN,ULS) -> %
      ++ \spad{puiseux(r,f(x))} returns \spad{f(x^r)}.
    rationalPower: % -> RN
      ++ \spad{rationalPower(f(x))} returns r where the Puiseux series
      ++ \spad{f(x) = g(x^r)}.
    laurentRep  : % -> ULS
      ++ \spad{laurentRep(f(x))} returns \spad{g(x)} where the Puiseux series
      ++ \spad{f(x) = g(x^r)} is represented by \spad{[r,g(x)]}.
    degree: % -> RN
      ++ \spad{degree(f(x))} returns the degree of the leading term of the
      ++ Puiseux series \spad{f(x)}, which may have zero as a coefficient.
    coerce: ULS -> %
      ++ \spad{coerce(f(x))} converts the Laurent series \spad{f(x)} to a
      ++ Puiseux series.
    laurent: % -> ULS
```

```
      ++ \spad{laurent(f(x))} converts the Puiseux series \spad{f(x)} to a
      ++ Laurent series if possible. Error: if this is not possible.
    laurentIfCan: % -> Union(ULS,"failed")
      ++ \spad{laurentIfCan(f(x))} converts the Puiseux series \spad{f(x)}
      ++ to a Laurent series if possible.
      ++ If this is not possible, "failed" is returned.

   add

     zero? x == zero? laurentRep x

     retract(x:%):ULS == laurent x

     retractIfCan(x:%):Union(ULS,"failed") == laurentIfCan x
```

———-

— UPXSCCA.dotabb —

```
"UPXSCCA"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=UPXSCCA"];
"UPXSCCA" -> "RETRACT"
"UPXSCCA" -> "UPXSCAT"
```

———-

— UPXSCCA.dotfull —

```
"UnivariatePuiseuxSeriesConstructorCategory(a:Ring,b:UnivariateLaurentSeriesCategory(a))"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=UPXSCCA"];
"UnivariatePuiseuxSeriesConstructorCategory(a:Ring,b:UnivariateLaurentSeriesCategory(a))"
  -> "RetractableTo(UnivariatePuiseuxSeriesCategory(Ring))"
"UnivariatePuiseuxSeriesConstructorCategory(a:Ring,b:UnivariateLaurentSeriesCategory(a))"
  -> "UnivariatePuiseuxSeriesCategory(a:Ring)"
```

———-

— UPXSCCA.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"UnivariatePuiseuxSeriesConstructorCategory(a:Ring,b:UnivariateLaurentSeriesCategory(a))"
```

```
 [color=lightblue,href="bookvol10.2.pdf#nameddest=UPXSCCA"];
"UnivariatePuiseuxSeriesConstructorCategory(a:Ring,b:UnivariateLaurentSeriesCategory(a))"
  -> "RETRACT..."
"UnivariatePuiseuxSeriesConstructorCategory(a:Ring,b:UnivariateLaurentSeriesCategory(a))"
  -> "UnivariatePuiseuxSeriesCategory(a:Ring)"

"UnivariatePuiseuxSeriesCategory(a:Ring)" [color=lightblue];
"UnivariatePuiseuxSeriesCategory(a:Ring)" ->
    "UnivariatePowerSeriesCategory(a:Ring,Fraction(Integer))"
"UnivariatePuiseuxSeriesCategory(a:Ring)" ->
    "TRANFUN..."
"UnivariatePuiseuxSeriesCategory(a:Ring)" ->
    "FIELD..."
"UnivariatePuiseuxSeriesCategory(a:Ring)" ->
    "RADCAT..."

"UnivariatePowerSeriesCategory(a:Ring,Fraction(Integer))" [color=seagreen];
"UnivariatePowerSeriesCategory(a:Ring,Fraction(Integer))" ->
    "UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)"

"UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)"
 [color=lightblue];
"UnivariatePowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid)" ->
 "PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:SingletonAsOrderedSet)"

"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:SingletonAsOrderedSet)"
 [color=seagreen];
"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:SingletonAsOrderedSet)"
  -> "PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)"

"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)"
 [color=lightblue];
"PowerSeriesCategory(a:Ring,b:OrderedAbelianMonoid,c:OrderedSet)" ->
    "AMR..."

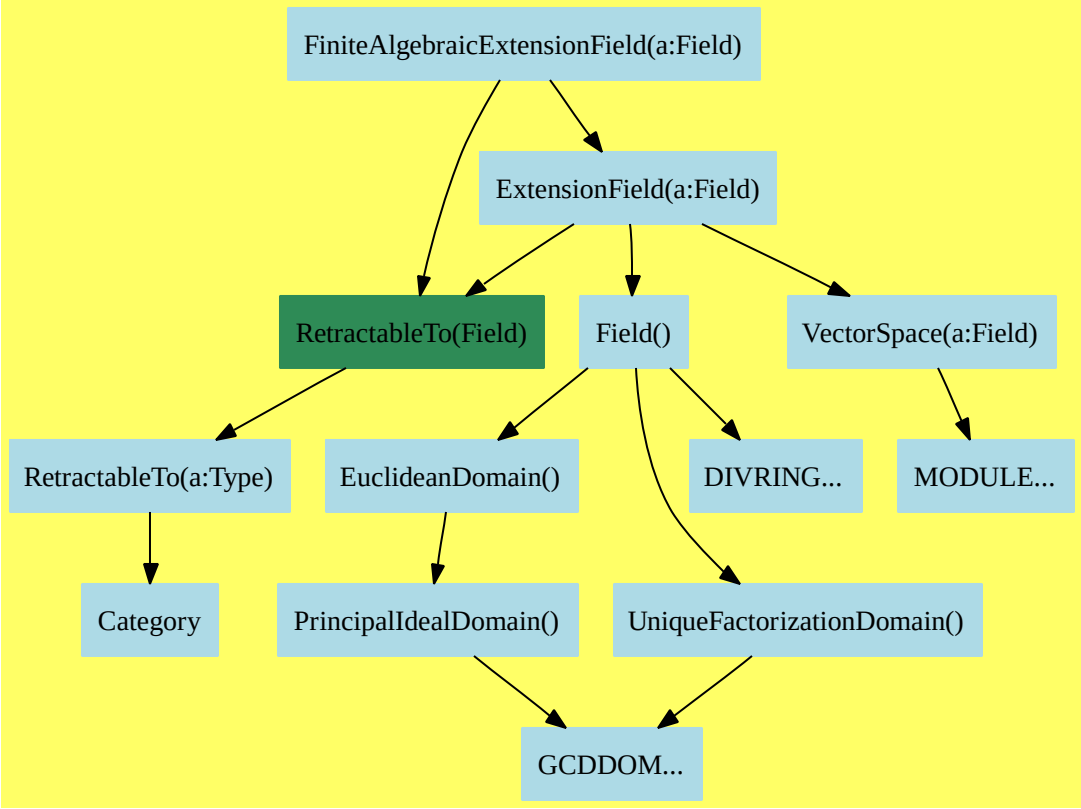
"RETRACT..." [color=lightblue];
"TRANFUN..." [color=lightblue];
"FIELD..." [color=lightblue];
"RADCAT..." [color=lightblue];
"AMR..." [color=lightblue];
}
```

———-

Chapter 19

Category Layer 18

19.0.213 FiniteAlgebraicExtensionField (FAXF)



— FiniteAlgebraicExtensionField.input —

```
)set break resume
)sys rm -f FiniteAlgebraicExtensionField.output
)spool FiniteAlgebraicExtensionField.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FiniteAlgebraicExtensionField
--R FiniteAlgebraicExtensionField F: Field  is a category constructor
--R Abbreviation for FiniteAlgebraicExtensionField is FAXF
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FAXF
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (F,%) -> %                      ?*? : (%,F) -> %
--R ?*? : (Fraction Integer,%) -> %       ?*? : (%,Fraction Integer) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,Integer) -> %
--R ?**? : (%,PositiveInteger) -> %       ?+? : (%,%) -> %
--R ?-? : (%,%) -> %                      -? : % -> %
--R ?/? : (%,F) -> %                      ?/? : (%,%) -> %
--R ?=? : (%,%) -> Boolean                D : % -> % if F has FINITE
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,Integer) -> %                ?^? : (%,PositiveInteger) -> %
--R algebraic? : % -> Boolean             associates? : (%,%) -> Boolean
--R basis : () -> Vector %                coerce : F -> %
--R coerce : Fraction Integer -> %        coerce : % -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R coordinates : % -> Vector F           degree : % -> PositiveInteger
--R dimension : () -> CardinalNumber      factor : % -> Factored %
--R gcd : List % -> %                     gcd : (%,%) -> %
--R hash : % -> SingleInteger             inGroundField? : % -> Boolean
--R inv : % -> %                          latex : % -> String
--R lcm : List % -> %                     lcm : (%,%) -> %
--R norm : % -> F                         one? : % -> Boolean
--R prime? : % -> Boolean                 ?quo? : (%,%) -> %
--R recip : % -> Union(%,"failed")        ?rem? : (%,%) -> %
--R represents : Vector F -> %            retract : % -> F
--R sample : () -> %                      sizeLess? : (%,%) -> Boolean
--R squareFree : % -> Factored %          squareFreePart : % -> %
--R trace : % -> F                        transcendent? : % -> Boolean
--R unit? : % -> Boolean                  unitCanonical : % -> %
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R D : (%,NonNegativeInteger) -> % if F has FINITE
--R Frobenius : (%,NonNegativeInteger) -> % if F has FINITE
```

```
--R Frobenius : % -> % if F has FINITE
--R ?^? : (%,NonNegativeInteger) -> %
--R basis : PositiveInteger -> Vector %
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if F has CHARNZ or F has FINITE
--R charthRoot : % -> % if F has FINITE
--R conditionP : Matrix % -> Union(Vector %,"failed") if F has FINITE
--R coordinates : Vector % -> Matrix F
--R createNormalElement : () -> % if F has FINITE
--R createPrimitiveElement : () -> % if F has FINITE
--R definingPolynomial : () -> SparseUnivariatePolynomial F
--R degree : % -> OnePointCompletion PositiveInteger
--R differentiate : (%,NonNegativeInteger) -> % if F has FINITE
--R differentiate : % -> % if F has FINITE
--R discreteLog : (%,%) -> Union(NonNegativeInteger,"failed") if F has CHARNZ or F has FINITE
--R discreteLog : % -> NonNegativeInteger if F has FINITE
--R divide : (%,%) -> Record(quotient: %,remainder: %)
--R euclideanSize : % -> NonNegativeInteger
--R expressIdealMember : (List %,%) -> Union(List %,"failed")
--R exquo : (%,%) -> Union(%,"failed")
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
--R extensionDegree : () -> PositiveInteger
--R extensionDegree : () -> OnePointCompletion PositiveInteger
--R factorsOfCyclicGroupSize : () -> List Record(factor: Integer,exponent: Integer) if F has FINITE
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivariatePolynomia
--R generator : () -> % if F has FINITE
--R index : PositiveInteger -> % if F has FINITE
--R init : () -> % if F has FINITE
--R linearAssociatedExp : (%,SparseUnivariatePolynomial F) -> % if F has FINITE
--R linearAssociatedLog : (%,%) -> Union(SparseUnivariatePolynomial F,"failed") if F has FINITE
--R linearAssociatedLog : % -> SparseUnivariatePolynomial F if F has FINITE
--R linearAssociatedOrder : % -> SparseUnivariatePolynomial F if F has FINITE
--R lookup : % -> PositiveInteger if F has FINITE
--R minimalPolynomial : (%,PositiveInteger) -> SparseUnivariatePolynomial % if F has FINITE
--R minimalPolynomial : % -> SparseUnivariatePolynomial F
--R multiEuclidean : (List %,%) -> Union(List %,"failed")
--R nextItem : % -> Union(%,"failed") if F has FINITE
--R norm : (%,PositiveInteger) -> % if F has FINITE
--R normal? : % -> Boolean if F has FINITE
--R normalElement : () -> % if F has FINITE
--R order : % -> OnePointCompletion PositiveInteger if F has CHARNZ or F has FINITE
--R order : % -> PositiveInteger if F has FINITE
--R primeFrobenius : % -> % if F has CHARNZ or F has FINITE
--R primeFrobenius : (%,NonNegativeInteger) -> % if F has CHARNZ or F has FINITE
--R primitive? : % -> Boolean if F has FINITE
--R primitiveElement : () -> % if F has FINITE
--R principalIdeal : List % -> Record(coef: List %,generator: %)
--R random : () -> % if F has FINITE
--R representationType : () -> Union("prime",polynomial,normal,cyclic) if F has FINITE
```

```
--R retractIfCan : % -> Union(F,"failed")
--R size : () -> NonNegativeInteger if F has FINITE
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R tableForDiscreteLogarithm : Integer -> Table(PositiveInteger,NonNegativeInteger) if F has FI
--R trace : (%,PositiveInteger) -> % if F has FINITE
--R transcendenceDegree : () -> NonNegativeInteger
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R
--E 1

)spool
)lisp (bye)
```

———-

— FiniteAlgebraicExtensionField.help —

```
=====================================================================
FiniteAlgebraicExtensionField examples
=====================================================================

See Also:
o )show FiniteAlgebraicExtensionField
```

———-

See:

⇐ “ExtensionField” (XF) 18.0.206 on page 1510
⇐ “RetractableTo” (RETRACT) 2.0.18 on page 55

Exports:

0	1	algebraic?
associates?	basis	characteristic
charthRoot	coerce	conditionP
coordinates	createNormalElement	createPrimitiveElement
D	definingPolynomial	degree
differentiate	dimension	discreteLog
divide	euclideanSize	expressIdealMember
exquo	extendedEuclidean	extensionDegree
factor	factorsOfCyclicGroupSize	Frobenius
gcd	gcdPolynomial	generator
hash	index	inGroundField?
init	inv	latex
lcm	linearAssociatedExp	linearAssociatedLog
linearAssociatedOrder	lookup	minimalPolynomial
multiEuclidean	nextItem	norm
normal?	normalElement	one?
order	prime?	primeFrobenius
primitive?	primitiveElement	principalIdeal
random	recip	representationType
represents	retract	retractIfCan
sample	size	sizeLess?
squareFree	squareFreePart	subtractIfCan
tableForDiscreteLogarithm	trace	transcendenceDegree
transcendent?	unit?	unitCanonical
unitNormal	zero?	?*?
?**?	?+?	?-?
-?	?/?	?=?
?^?	?quo?	?rem?
?~=?		

Attributes Exported:

- **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.
- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.

- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
basis : () -> Vector %
basis : PositiveInteger -> Vector %
coordinates : % -> Vector F
definingPolynomial : () -> SparseUnivariatePolynomial F
generator : () -> % if F has FINITE
minimalPolynomial : (%,PositiveInteger) ->
   SparseUnivariatePolynomial %
     if F has FINITE
normalElement : () -> % if F has FINITE
```

These are implemented by this category:

```
algebraic? : % -> Boolean
charthRoot : % -> Union(%,"failed")
    if F has CHARNZ or F has FINITE
coordinates : Vector % -> Matrix F
createNormalElement : () -> % if F has FINITE
degree : % -> PositiveInteger
dimension : () -> CardinalNumber
extensionDegree : () -> PositiveInteger
linearAssociatedExp : (%,SparseUnivariatePolynomial F) -> %
    if F has FINITE
linearAssociatedLog : (%,%) ->
   Union(SparseUnivariatePolynomial F,"failed")
     if F has FINITE
linearAssociatedLog : % -> SparseUnivariatePolynomial F
    if F has FINITE
linearAssociatedOrder : % -> SparseUnivariatePolynomial F
    if F has FINITE
minimalPolynomial : % -> SparseUnivariatePolynomial F
norm : % -> F
norm : (%,PositiveInteger) -> % if F has FINITE
normal? : % -> Boolean if F has FINITE
represents : Vector F -> %
size : () -> NonNegativeInteger if F has FINITE
trace : % -> F
trace : (%,PositiveInteger) -> % if F has FINITE
transcendenceDegree : () -> NonNegativeInteger
transcendent? : % -> Boolean
```

These exports come from (p1510) ExtensionField(F:Field):

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean
```

```
characteristic : () -> NonNegativeInteger
coerce : F -> %
coerce : % -> %
coerce : Integer -> %
coerce : % -> OutputForm
coerce : Fraction Integer -> %
discreteLog : (%,%) ->
   Union(NonNegativeInteger,"failed")
     if F has CHARNZ or F has FINITE
divide : (%,%) -> Record(quotient: %,remainder: %)
euclideanSize : % -> NonNegativeInteger
expressIdealMember : (List %,%) -> Union(List %,"failed")
exquo : (%,%) -> Union(%,"failed")
extendedEuclidean : (%,%,%) ->
   Union(Record(coef1: %,coef2: %),"failed")
extendedEuclidean : (%,%) ->
   Record(coef1: %,coef2: %,generator: %)
factor : % -> Factored %
Frobenius : (%,NonNegativeInteger) -> % if F has FINITE
Frobenius : % -> % if F has FINITE
gcd : List % -> %
gcd : (%,%) -> %
gcdPolynomial : (SparseUnivariatePolynomial %,
                 SparseUnivariatePolynomial %) ->
                    SparseUnivariatePolynomial %
hash : % -> SingleInteger
inGroundField? : % -> Boolean
inv : % -> %
latex : % -> String
lcm : List % -> %
lcm : (%,%) -> %
multiEuclidean : (List %,%) -> Union(List %,"failed")
one? : % -> Boolean
order : % -> OnePointCompletion PositiveInteger
     if F has CHARNZ or F has FINITE
prime? : % -> Boolean
primeFrobenius : % -> %
    if F has CHARNZ or F has FINITE
primeFrobenius : (%,NonNegativeInteger) -> %
    if F has CHARNZ or F has FINITE
principalIdeal : List % -> Record(coef: List %,generator: %)
recip : % -> Union(%,"failed")
retract : % -> F
retractIfCan : % -> Union(F,"failed")
sample : () -> %
squareFree : % -> Factored %
squareFreePart : % -> %
sizeLess? : (%,%) -> Boolean
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean
```

```
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
zero? : % -> Boolean
?/? : (%,%) -> %
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?*? : (Fraction Integer,%) -> %
?*? : (%,Fraction Integer) -> %
?*? : (F,%) -> %
?*? : (%,F) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,Integer) -> %
?^? : (%,Integer) -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?quo? : (%,%) -> %
?rem? : (%,%) -> %
?/? : (%,F) -> %
```

These exports come from (p55) RetractableTo(F:Field):

These exports come from (p1518) FiniteFieldCategory():

```
charthRoot : % -> % if F has FINITE
conditionP : Matrix % -> Union(Vector %,"failed")
    if F has FINITE
createPrimitiveElement : () -> % if F has FINITE
D : % -> % if F has FINITE
D : (%,NonNegativeInteger) -> % if F has FINITE
differentiate : % -> % if F has FINITE
differentiate : (%,NonNegativeInteger) -> %
    if F has FINITE
discreteLog : % -> NonNegativeInteger if F has FINITE
factorsOfCyclicGroupSize : () ->
   List Record(factor: Integer,exponent: Integer)
     if F has FINITE
index : PositiveInteger -> % if F has FINITE
init : () -> % if F has FINITE
lookup : % -> PositiveInteger if F has FINITE
nextItem : % -> Union(%,"failed") if F has FINITE
```

```
 order : % -> PositiveInteger if F has FINITE
 primitive? : % -> Boolean if F has FINITE
 primitiveElement : () -> % if F has FINITE
 random : () -> % if F has FINITE
 representationType : () ->
    Union("prime",polynomial,normal,cyclic)
      if F has FINITE
 tableForDiscreteLogarithm : Integer ->
    Table(PositiveInteger,NonNegativeInteger)
      if F has FINITE
```

— category FAXF FiniteAlgebraicExtensionField —

```
)abbrev category FAXF FiniteAlgebraicExtensionField
++ Author: J. Grabmeier, A. Scheerhorn
++ Date Created: 11 March 1991
++ Date Last Updated: 31 March 1991
++ Basic Operations: _+, _*, extensionDegree,
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords: field, extension field, algebraic extension, finite extension
++ References:
++  R.Lidl, H.Niederreiter: Finite Field, Encycoldia of Mathematics and
++  Its Applications, Vol. 20, Cambridge Univ. Press, 1983,
++  ISBN 0 521 30240 4 J. Grabmeier, A. Scheerhorn: Finite Fields in AXIOM.
++  AXIOM Technical Report Series, ATR/5 NP2522.
++ Description:
++ FiniteAlgebraicExtensionField F is the category of fields
++ which are finite algebraic extensions of the field F.
++ If F is finite then any finite algebraic extension of F
++ is finite, too. Let K be a finite algebraic extension of the
++ finite field F. The exponentiation of elements of K
++ defines a Z-module structure on the multiplicative group of K.
++ The additive group of K becomes a module over the ring of
++ polynomials over F via the operation
++ \spadfun{linearAssociatedExp}(a:K,f:SparseUnivariatePolynomial F)
++ which is linear over F, i.e. for elements a from K,
++ c,d from F and f,g univariate polynomials over F
++ we have \spadfun{linearAssociatedExp}(a,cf+dg) equals c times
++ \spadfun{linearAssociatedExp}(a,f) plus d times
++ \spadfun{linearAssociatedExp}(a,g).
++ Therefore \spadfun{linearAssociatedExp} is defined completely by
++ its action on  monomials from F[X]:
++ \spadfun{linearAssociatedExp}(a,monomial(1,k)\$SUP(F)) is defined to be
++ \spadfun{Frobenius}(a,k) which is a**(q**k) where q=size()\$F.
++ The operations order and discreteLog associated with the multiplicative
++ exponentiation have additive analogues associated to the operation
++ \spadfun{linearAssociatedExp}. These are the functions
```

```
++ \spadfun{linearAssociatedOrder} and \spadfun{linearAssociatedLog},
++ respectively.

FiniteAlgebraicExtensionField(F : Field) : Category == _
  Join(ExtensionField F, RetractableTo F) with
  -- should be unified with algebras
  -- Join(ExtensionField F, FramedAlgebra F, RetractableTo F) with
    basis : () -> Vector $
      ++ basis() returns a fixed basis of \$ as \spad{F}-vectorspace.
    basis : PositiveInteger -> Vector $
      ++ basis(n) returns a fixed basis of a subfield of \$ as
      ++ \spad{F}-vectorspace.
    coordinates : $ -> Vector F
      ++ coordinates(a) returns the coordinates of \spad{a} with respect
      ++ to the fixed \spad{F}-vectorspace basis.
    coordinates : Vector $ -> Matrix F
      ++ coordinates([v1,...,vm]) returns the coordinates of the
      ++ vi's with to the fixed basis.  The coordinates of vi are
      ++ contained in the ith row of the matrix returned by this
      ++ function.
    represents:  Vector F -> $
      ++ represents([a1,..,an]) returns \spad{a1*v1 + ... + an*vn}, where
      ++ v1,...,vn are the elements of the fixed basis.
    minimalPolynomial: $ -> SparseUnivariatePolynomial F
      ++ minimalPolynomial(a) returns the minimal polynomial of an
      ++ element \spad{a} over the ground field F.
    definingPolynomial: () -> SparseUnivariatePolynomial F
      ++ definingPolynomial() returns the polynomial used to define
      ++ the field extension.
    extensionDegree : () ->  PositiveInteger
      ++ extensionDegree() returns the degree of field extension.
    degree : $ -> PositiveInteger
      ++ degree(a) returns the degree of the minimal polynomial of an
      ++ element \spad{a} over the ground field F.
    norm: $  -> F
      ++ norm(a) computes the norm of \spad{a} with respect to the
      ++ field considered as an algebra with 1 over the ground field F.
    trace: $ -> F
      ++ trace(a) computes the trace of \spad{a} with respect to
      ++ the field considered as an algebra with 1 over the ground field F.
    if F has Finite then
      FiniteFieldCategory
      minimalPolynomial: ($,PositiveInteger) -> SparseUnivariatePolynomial $
        ++ minimalPolynomial(x,n) computes the minimal polynomial of x over
        ++ the field of extension degree n over the ground field F.
      norm: ($,PositiveInteger)  -> $
        ++ norm(a,d) computes the norm of \spad{a} with respect to the field
        ++ of extension degree d over the ground field of size.
        ++ Error: if d does not divide the extension degree of \spad{a}.
        ++ Note that norm(a,d) = reduce(*,[a**(q**(d*i)) for i in 0..n/d])
```

```
trace: ($,PositiveInteger)   -> $
  ++ trace(a,d) computes the trace of \spad{a} with respect to the
  ++ field of extension degree d over the ground field of size q.
  ++ Error: if d does not divide the extension degree of \spad{a}.
  ++ Note that
  ++ \spad{trace(a,d)=reduce(+,[a**(q**(d*i)) for i in 0..n/d])}.
createNormalElement: () -> $
  ++ createNormalElement() computes a normal element over the ground
  ++ field F, that is,
  ++ \spad{a**(q**i), 0 <= i < extensionDegree()} is an F-basis,
  ++ where \spad{q = size()\$F}.
  ++ Reference: Such an element exists Lidl/Niederreiter: Theorem 2.35.
normalElement: () -> $
  ++ normalElement() returns a element, normal over the ground field F,
  ++ i.e. \spad{a**(q**i), 0 <= i < extensionDegree()} is an F-basis,
  ++ where \spad{q = size()\$F}.
  ++ At the first call, the element is computed by
  ++ \spadfunFrom{createNormalElement}{FiniteAlgebraicExtensionField}
  ++ then cached in a global variable.
  ++ On subsequent calls, the element is retrieved by referencing the
  ++ global variable.
normal?: $ -> Boolean
  ++ normal?(a) tests whether the element \spad{a} is normal over the
  ++ ground field F, i.e.
  ++ \spad{a**(q**i), 0 <= i <= extensionDegree()-1} is an F-basis,
  ++ where \spad{q = size()\$F}.
  ++ Implementation according to Lidl/Niederreiter: Theorem 2.39.
generator: () -> $
  ++ generator() returns a root of the defining polynomial.
  ++ This element generates the field as an algebra over the ground
  ++ field.
linearAssociatedExp:($,SparseUnivariatePolynomial F) -> $
  ++ linearAssociatedExp(a,f) is linear over F, i.e.
  ++ for elements a from \$, c,d form F and
  ++ f,g univariate polynomials over F we have
  ++ \spadfun{linearAssociatedExp}(a,cf+dg) equals c times
  ++ \spadfun{linearAssociatedExp}(a,f) plus d times
  ++ \spadfun{linearAssociatedExp}(a,g). Therefore
  ++ \spadfun{linearAssociatedExp} is defined completely by its
  ++ action on monomials from F[X]:
  ++ \spadfun{linearAssociatedExp}(a,monomial(1,k)\$SUP(F)) is
  ++ defined to be \spadfun{Frobenius}(a,k) which is a**(q**k),
  ++ where q=size()\$F.
linearAssociatedOrder: $ -> SparseUnivariatePolynomial F
  ++ linearAssociatedOrder(a) retruns the monic polynomial g of
  ++ least degree, such that \spadfun{linearAssociatedExp}(a,g) is 0.
linearAssociatedLog: $ -> SparseUnivariatePolynomial F
  ++ linearAssociatedLog(a) returns a polynomial g, such that
  ++ \spadfun{linearAssociatedExp}(normalElement(),g) equals a.
linearAssociatedLog: ($,$) -> _
```

```
        Union(SparseUnivariatePolynomial F,"failed")
        ++ linearAssociatedLog(b,a) returns a polynomial g, such
        ++ that the \spadfun{linearAssociatedExp}(b,g) equals a.
        ++ If there is no such polynomial g, then
        ++ \spadfun{linearAssociatedLog} fails.
  add
    I   ==> Integer
    PI  ==> PositiveInteger
    NNI ==> NonNegativeInteger
    SUP ==> SparseUnivariatePolynomial
    DLP ==> DiscreteLogarithmPackage

    represents(v) ==
      a:$:=0
      b:=basis()
      for i in 1..extensionDegree()@PI repeat
        a:=a+(v.i)*(b.i)
      a

    transcendenceDegree() == 0$NNI

    dimension() == (#basis()) ::NonNegativeInteger::CardinalNumber

    coordinates(v:Vector $) ==
      m := new(#v, extensionDegree(), 0)$Matrix(F)
      for i in minIndex v .. maxIndex v for j in minRowIndex m .. repeat
        setRow_!(m, j, coordinates qelt(v, i))
      m

    algebraic? a == true

    transcendent? a == false

-- This definition is a duplicate and has been removed
--    extensionDegree():OnePointCompletion(PositiveInteger) ==
--      (#basis()) :: PositiveInteger::OnePointCompletion(PositiveInteger)

    extensionDegree() == (#basis()) :: PositiveInteger

-- These definitions are duplicates and have been removed
--    degree(a):OnePointCompletion(PositiveInteger) ==
--      degree(a)@PI::OnePointCompletion(PositiveInteger)

    -- degree a == degree(minimalPolynomial a)$SUP(F) :: PI

    trace a ==
      b := basis()
      abs : F := 0
      for i in 1..#b repeat
        abs := abs + coordinates(a*b.i).i
```

```
  abs

norm a ==
  b := basis()
  m := new(#b,#b, 0)$Matrix(F)
  for i in 1..#b repeat
    setRow_!(m,i, coordinates(a*b.i))
  determinant(m)

if F has Finite then
  linearAssociatedExp(x,f) ==
    erg:$:=0
    y:=x
    for i in 0..degree(f) repeat
      erg:=erg + coefficient(f,i) * y
      y:=Frobenius(y)
    erg

  linearAssociatedLog(b,x) ==
    x=0 => 0
    l:List List F:=[entries coordinates b]
    a:$:=b
    extdeg:NNI:=extensionDegree()@PI
    for i in 2..extdeg repeat
      a:=Frobenius(a)
      l:=concat(l,entries coordinates a)$(List List F)
    l:=concat(l,entries coordinates x)$(List List F)
    m1:=rowEchelon transpose matrix(l)$(Matrix F)
    v:=zero(extdeg)$(Vector F)
    rown:I:=1
    for i in 1..extdeg repeat
      if qelt(m1,rown,i) = 1$F then
        v.i:=qelt(m1,rown,extdeg+1)
        rown:=rown+1
    p:=+/[monomial(v.(i+1),i::NNI) for i in 0..(#v-1)]
    p=0 =>
     messagePrint("linearAssociatedLog: second argument not in_
                   group generated by first argument")$OutputForm
     "failed"
    p

  linearAssociatedLog(x) == linearAssociatedLog(normalElement(),x) ::
                          SparseUnivariatePolynomial(F)

  linearAssociatedOrder(x) ==
    x=0 => 0
    l:List List F:=[entries coordinates x]
    a:$:=x
    for i in 1..extensionDegree()@PI repeat
      a:=Frobenius(a)
```

```
    l:=concat(l,entries coordinates a)$(List List F)
  v:=first nullSpace transpose matrix(l)$(Matrix F)
  +/[monomial(v.(i+1),i::NNI) for i in 0..(#v-1)]

charthRoot(x):Union($,"failed") ==
  (charthRoot(x)@$)::Union($,"failed")
-- norm(e) == norm(e,1) pretend F
-- trace(e) == trace(e,1) pretend F

minimalPolynomial(a,n) ==
  extensionDegree()@PI rem n ^= 0 =>
    error "minimalPolynomial: 2. argument must divide extension degree"
  f:SUP $:=monomial(1,1)$(SUP $) - monomial(a,0)$(SUP $)
  u:$:=Frobenius(a,n)
  while not(u = a) repeat
    f:=f * (monomial(1,1)$(SUP $) - monomial(u,0)$(SUP $))
    u:=Frobenius(u,n)
  f

norm(e,s) ==
  qr := divide(extensionDegree(), s)
  zero?(qr.remainder) =>
    pow := (size()-1) quo (size()$F ** s - 1)
    e ** (pow::NonNegativeInteger)
  error "norm: second argument must divide degree of extension"

trace(e,s) ==
  qr:=divide(extensionDegree(),s)
  q:=size()$F
  zero?(qr.remainder) =>
    a:$:=0
    for i in 0..qr.quotient-1 repeat
      a:=a + e**(q**(s*i))
    a
  error "trace: second argument must divide degree of extension"

size() == size()$F ** extensionDegree()

createNormalElement() ==
  characteristic() = size() => 1
  res : $
  for i in 1.. repeat
    res := index(i :: PI)
    not inGroundField? res =>
      normal? res => return res
  -- theorem: there exists a normal element, this theorem is
  -- unknown to the compiler
  res

normal?(x:$) ==
```

```
        p:SUP $:=(monomial(1,extensionDegree()) - monomial(1,0))@(SUP $)
        f:SUP $:= +/[monomial(Frobenius(x,i),i)$(SUP $) _
                    for i in 0..extensionDegree()-1]
        gcd(p,f) = 1 => true
        false

      degree a ==
        y:$:=Frobenius a
        deg:PI:=1
        while y^=a repeat
          y := Frobenius(y)
          deg:=deg+1
        deg
```

———-

— **FAXF.dotabb** —

```
"FAXF"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FAXF"];
"FAXF" -> "XF"
"FAXF" -> "RETRACT"
```

———-

— **FAXF.dotfull** —

```
"FiniteAlgebraicExtensionField(a:Field)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FAXF"];
"FiniteAlgebraicExtensionField(a:Field)" -> "ExtensionField(a:Field)"
"FiniteAlgebraicExtensionField(a:Field)" -> "RetractableTo(a:Field)"
```

———-

— **FAXF.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FiniteAlgebraicExtensionField(a:Field)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FAXF"];
"FiniteAlgebraicExtensionField(a:Field)" -> "ExtensionField(a:Field)"
"FiniteAlgebraicExtensionField(a:Field)" -> "RetractableTo(Field)"
```

```
"ExtensionField(a:Field)" [color=lightblue];
"ExtensionField(a:Field)" -> "Field()"
"ExtensionField(a:Field)" -> "RetractableTo(Field)"
"ExtensionField(a:Field)" -> "VectorSpace(a:Field)"

"Field()" [color=lightblue];
"Field()" -> "EuclideanDomain()"
"Field()" -> "UniqueFactorizationDomain()"
"Field()" -> "DIVRING..."

"EuclideanDomain()" [color=lightblue];
"EuclideanDomain()" -> "PrincipalIdealDomain()"

"UniqueFactorizationDomain()" [color=lightblue];
"UniqueFactorizationDomain()" -> "GCDDOM..."

"PrincipalIdealDomain()" [color=lightblue];
"PrincipalIdealDomain()" -> "GCDDOM..."

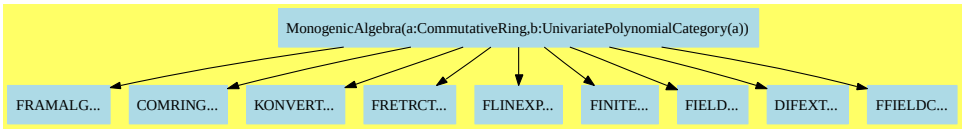
"RetractableTo(Field)" [color=seagreen];
"RetractableTo(Field)" -> "RetractableTo(a:Type)"

"RetractableTo(a:Type)" [color=lightblue];
"RetractableTo(a:Type)" -> "Category"

"VectorSpace(a:Field)" [color=lightblue];
"VectorSpace(a:Field)" -> "MODULE..."

"MODULE..." [color=lightblue];
"DIVRING..." [color=lightblue];
"GCDDOM..." [color=lightblue];
"Category" [color=lightblue];
}
```

19.0.214 MonogenicAlgebra (MONOGEN)



— MonogenicAlgebra.input —

```
)set break resume
)sys rm -f MonogenicAlgebra.output
)spool MonogenicAlgebra.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show MonogenicAlgebra
--R MonogenicAlgebra(R: CommutativeRing,UP: UnivariatePolynomialCategory t#1)  is a category constructor
--R Abbreviation for MonogenicAlgebra is MONOGEN
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for MONOGEN
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (R,%) -> %                      ?*? : (%,R) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        basis : () -> Vector %
--R coerce : R -> %                       coerce : Integer -> %
--R coerce : % -> OutputForm              convert : UP -> %
--R convert : % -> UP                     convert : Vector R -> %
--R convert : % -> Vector R               coordinates : % -> Vector R
--R definingPolynomial : () -> UP         discriminant : () -> R
--R discriminant : Vector % -> R          generator : () -> %
--R hash : % -> SingleInteger             inv : % -> % if R has FIELD
--R latex : % -> String                   lift : % -> UP
--R norm : % -> R                         one? : % -> Boolean
--R rank : () -> PositiveInteger          recip : % -> Union(%,"failed")
--R reduce : UP -> %                      represents : Vector R -> %
--R retract : % -> R                      sample : () -> %
--R trace : % -> R                        traceMatrix : () -> Matrix R
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (%,Fraction Integer) -> % if R has FIELD
--R ?*? : (Fraction Integer,%) -> % if R has FIELD
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,Integer) -> % if R has FIELD
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (%,%) -> % if R has FIELD
--R D : (%,(R -> R)) -> % if R has FIELD
--R D : (%,(R -> R),NonNegativeInteger) -> % if R has FIELD
--R D : (%,List Symbol,List NonNegativeInteger) -> % if and(has(R,PartialDifferentialRing Symbol),has(R,Fie
--R D : (%,Symbol,NonNegativeInteger) -> % if and(has(R,PartialDifferentialRing Symbol),has(R,Field))
--R D : (%,List Symbol) -> % if and(has(R,PartialDifferentialRing Symbol),has(R,Field))
--R D : (%,Symbol) -> % if and(has(R,PartialDifferentialRing Symbol),has(R,Field))
```

```
--R D : (%,NonNegativeInteger) -> % if and(has(R,DifferentialRing),has(R,Field)) or R has FFIELD
--R D : % -> % if and(has(R,DifferentialRing),has(R,Field)) or R has FFIELDC
--R ?^? : (%,Integer) -> % if R has FIELD
--R ?^? : (%,NonNegativeInteger) -> %
--R associates? : (%,%) -> Boolean if R has FIELD
--R characteristic : () -> NonNegativeInteger
--R characteristicPolynomial : % -> UP
--R charthRoot : % -> Union(%,"failed") if R has CHARNZ
--R charthRoot : % -> % if R has FFIELDC
--R coerce : Fraction Integer -> % if R has FIELD or R has RETRACT FRAC INT
--R coerce : % -> % if R has FIELD
--R conditionP : Matrix % -> Union(Vector %,"failed") if R has FFIELDC
--R coordinates : Vector % -> Matrix R
--R coordinates : (Vector %,Vector %) -> Matrix R
--R coordinates : (%,Vector %) -> Vector R
--R createPrimitiveElement : () -> % if R has FFIELDC
--R derivationCoordinates : (Vector %,(R -> R)) -> Matrix R if R has FIELD
--R differentiate : (%,(R -> R)) -> % if R has FIELD
--R differentiate : (%,(R -> R),NonNegativeInteger) -> % if R has FIELD
--R differentiate : (%,List Symbol,List NonNegativeInteger) -> % if and(has(R,PartialDifferentia
--R differentiate : (%,Symbol,NonNegativeInteger) -> % if and(has(R,PartialDifferentialRing Symb
--R differentiate : (%,List Symbol) -> % if and(has(R,PartialDifferentialRing Symbol),has(R,Fiel
--R differentiate : (%,Symbol) -> % if and(has(R,PartialDifferentialRing Symbol),has(R,Field))
--R differentiate : (%,NonNegativeInteger) -> % if and(has(R,DifferentialRing),has(R,Field)) or
--R differentiate : % -> % if and(has(R,DifferentialRing),has(R,Field)) or R has FFIELDC
--R discreteLog : (%,%) -> Union(NonNegativeInteger,"failed") if R has FFIELDC
--R discreteLog : % -> NonNegativeInteger if R has FFIELDC
--R divide : (%,%) -> Record(quotient: %,remainder: %) if R has FIELD
--R euclideanSize : % -> NonNegativeInteger if R has FIELD
--R expressIdealMember : (List %,%) -> Union(List %,"failed") if R has FIELD
--R exquo : (%,%) -> Union(%,"failed") if R has FIELD
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %) if R has FIELD
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed") if R has FIELD
--R factor : % -> Factored % if R has FIELD
--R factorsOfCyclicGroupSize : () -> List Record(factor: Integer,exponent: Integer) if R has FFI
--R gcd : (%,%) -> % if R has FIELD
--R gcd : List % -> % if R has FIELD
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivari
--R index : PositiveInteger -> % if R has FINITE
--R init : () -> % if R has FFIELDC
--R lcm : (%,%) -> % if R has FIELD
--R lcm : List % -> % if R has FIELD
--R lookup : % -> PositiveInteger if R has FINITE
--R minimalPolynomial : % -> UP if R has FIELD
--R multiEuclidean : (List %,%) -> Union(List %,"failed") if R has FIELD
--R nextItem : % -> Union(%,"failed") if R has FFIELDC
--R order : % -> OnePointCompletion PositiveInteger if R has FFIELDC
--R order : % -> PositiveInteger if R has FFIELDC
--R prime? : % -> Boolean if R has FIELD
--R primeFrobenius : % -> % if R has FFIELDC
```

```
--R primeFrobenius : (%,NonNegativeInteger) -> % if R has FFIELDC
--R primitive? : % -> Boolean if R has FFIELDC
--R primitiveElement : () -> % if R has FFIELDC
--R principalIdeal : List % -> Record(coef: List %,generator: %) if R has FIELD
--R ?quo? : (%,%) -> % if R has FIELD
--R random : () -> % if R has FINITE
--R reduce : Fraction UP -> Union(%,"failed") if R has FIELD
--R reducedSystem : Matrix % -> Matrix R
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix R,vec: Vector R)
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer) if R has LINEXP I
--R reducedSystem : Matrix % -> Matrix Integer if R has LINEXP INT
--R regularRepresentation : % -> Matrix R
--R regularRepresentation : (%,Vector %) -> Matrix R
--R ?rem? : (%,%) -> % if R has FIELD
--R representationType : () -> Union("prime",polynomial,normal,cyclic) if R has FFIELDC
--R represents : (Vector R,Vector %) -> %
--R retract : % -> Fraction Integer if R has RETRACT FRAC INT
--R retract : % -> Integer if R has RETRACT INT
--R retractIfCan : % -> Union(R,"failed")
--R retractIfCan : % -> Union(Fraction Integer,"failed") if R has RETRACT FRAC INT
--R retractIfCan : % -> Union(Integer,"failed") if R has RETRACT INT
--R size : () -> NonNegativeInteger if R has FINITE
--R sizeLess? : (%,%) -> Boolean if R has FIELD
--R squareFree : % -> Factored % if R has FIELD
--R squareFreePart : % -> % if R has FIELD
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R tableForDiscreteLogarithm : Integer -> Table(PositiveInteger,NonNegativeInteger) if R has FFIELDC
--R traceMatrix : Vector % -> Matrix R
--R unit? : % -> Boolean if R has FIELD
--R unitCanonical : % -> % if R has FIELD
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %) if R has FIELD
--R
--E 1

)spool
)lisp (bye)
```

———-

— MonogenicAlgebra.help —

```
=====================================================================
MonogenicAlgebra examples
=====================================================================

See Also:
o )show MonogenicAlgebra
```

———-

See:

⇒ "FunctionFieldCategory" (FFCAT) 20.0.217 on page 1642
⇐ "CommutativeRing" (COMRING) 10.0.137 on page 843
⇐ "ConvertibleTo" (KONVERT) 2.0.8 on page 23
⇐ "FramedAlgebra" (FRAMALG) 18.0.209 on page 1539
⇐ "FullyRetractableTo" (FRETRCT) 3.0.27 on page 87
⇐ "LinearlyExplicitRingOver" (LINEXP) 10.0.142 on page 869

Exports:

0	1	associates?
basis	characteristic	characteristicPolynomial
charthRoot	coerce	conditionP
convert	coordinates	createPrimitiveElement
D	definingPolynomial	derivationCoordinates
differentiate	discreteLog	discriminant
divide	euclideanSize	expressIdealMember
exquo	extendedEuclidean	factor
factorsOfCyclicGroupSize	generator	gcd
gcdPolynomial	hash	index
init	inv	latex
lcm	lift	lookup
minimalPolynomial	multiEuclidean	nextItem
norm	one?	order
prime?	primeFrobenius	primitive?
primitiveElement	principalIdeal	random
rank	recip	reduce
reducedSystem	regularRepresentation	represents
representationType	retract	retractIfCan
sample	size	sizeLess?
squareFree	squareFreePart	subtractIfCan
tableForDiscreteLogarithm	trace	traceMatrix
unit?	unitCanonical	unitNormal
zero?	?*?	?**?
?+?	?-?	-?
?=?	?^?	?~=?
?/?	?quo?	?rem?

Attributes Exported:

- if $ has Field then canonicalUnitNormal where **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- if $ has Field then canonicalClosed where **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.

- if $ has Field then noZeroDivisors where **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
definingPolynomial : () -> UP
lift : % -> UP
reduce : UP -> %
```

These are implemented by this category:

```
basis : () -> Vector %
characteristicPolynomial : % -> UP
convert : % -> UP
convert : UP -> %
derivationCoordinates : (Vector %,(R -> R)) -> Matrix R
    if R has FIELD
differentiate : (%,(R -> R)) -> % if R has FIELD
generator : () -> %
norm : % -> R
random : () -> % if R has FINITE
recip : % -> Union(%,"failed")
reduce : Fraction UP -> Union(%,"failed") if R has FIELD
retract : % -> R
retractIfCan : % -> Union(R,"failed")
size : () -> NonNegativeInteger if R has FINITE
```

These exports come from (p1539) FramedAlgebra(R,UP)
where R:CommutativeRing and UP:UnivariatePolynomialCategory(a)

```
0 : () -> %
1 : () -> %
characteristic : () -> NonNegativeInteger
charthRoot : % -> Union(%,"failed") if R has CHARNZ
coerce : R -> %
coerce : Integer -> %
coerce : % -> OutputForm
convert : Vector R -> %
convert : % -> Vector R
coordinates : (%,Vector %) -> Vector R
```

```
coordinates : (Vector %,Vector %) -> Matrix R
coordinates : Vector % -> Matrix R
coordinates : % -> Vector R
discriminant : Vector % -> R
discriminant : () -> R
hash : % -> SingleInteger
latex : % -> String
minimalPolynomial : % -> UP if R has FIELD
one? : % -> Boolean
rank : () -> PositiveInteger
regularRepresentation : (%,Vector %) -> Matrix R
regularRepresentation : % -> Matrix R
represents : (Vector R,Vector %) -> %
represents : Vector R -> %
sample : () -> %
subtractIfCan : (%,%) -> Union(%,"failed")
trace : % -> R
traceMatrix : Vector % -> Matrix R
traceMatrix : () -> Matrix R
zero? : % -> Boolean
?+? : (%,%) -> %
?=? : (%,%) -> Boolean
?~=? : (%,%) -> Boolean
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?*? : (R,%) -> %
?*? : (%,R) -> %
?-? : (%,%) -> %
-? : % -> %
?**? : (%,PositiveInteger) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
```

These exports come from (p87) FullyRetractableTo(R)
where R:CommutativeRing

```
coerce : Fraction Integer -> %
    if R has FIELD or R has RETRACT FRAC INT
retract : % -> Integer if R has RETRACT INT
retract : % -> Fraction Integer
    if R has RETRACT FRAC INT
retractIfCan : % -> Union(Fraction Integer,"failed")
    if R has RETRACT FRAC INT
retractIfCan : % -> Union(Integer,"failed")
    if R has RETRACT INT
```

These exports come from (p960) FullyLinearlyExplicitRingOver(R)

where R:CommutativeRing

```
reducedSystem : Matrix % -> Matrix R
reducedSystem :
  (Matrix %,Vector %) -> Record(mat: Matrix R,vec: Vector R)
reducedSystem :
  (Matrix %,Vector %) ->
     Record(mat: Matrix Integer,vec: Vector Integer)
       if R has LINEXP INT
reducedSystem : Matrix % -> Matrix Integer if R has LINEXP INT
```

These exports come from (p168) Finite()

```
index : PositiveInteger -> % if R has FINITE
lookup : % -> PositiveInteger if R has FINITE
```

These exports come from (p1221) Field()

```
associates? : (%,%) -> Boolean if R has FIELD
coerce : % -> % if R has FIELD
divide : (%,%) -> Record(quotient: %,remainder: %)
    if R has FIELD
euclideanSize : % -> NonNegativeInteger if R has FIELD
expressIdealMember : (List %,%) -> Union(List %,"failed")
    if R has FIELD
exquo : (%,%) -> Union(%,"failed") if R has FIELD
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
    if R has FIELD
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
    if R has FIELD
factor : % -> Factored % if R has FIELD
gcd : (%,%) -> % if R has FIELD
gcd : List % -> % if R has FIELD
gcdPolynomial :
   (SparseUnivariatePolynomial %,
    SparseUnivariatePolynomial %) ->
       SparseUnivariatePolynomial % if R has FIELD
inv : % -> % if R has FIELD
lcm : (%,%) -> % if R has FIELD
lcm : List % -> % if R has FIELD
multiEuclidean : (List %,%) -> Union(List %,"failed")
    if R has FIELD
prime? : % -> Boolean if R has FIELD
principalIdeal : List % -> Record(coef: List %,generator: %)
    if R has FIELD
sizeLess? : (%,%) -> Boolean if R has FIELD
squareFree : % -> Factored % if R has FIELD
squareFreePart : % -> % if R has FIELD
unit? : % -> Boolean if R has FIELD
```

```
unitCanonical : % -> % if R has FIELD
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
    if R has FIELD
?/? : (%,%) -> % if R has FIELD
?*? : (%,Fraction Integer) -> % if R has FIELD
?*? : (Fraction Integer,%) -> % if R has FIELD
?**? : (%,Integer) -> % if R has FIELD
?^? : (%,Integer) -> % if R has FIELD
?quo? : (%,%) -> % if R has FIELD
?rem? : (%,%) -> % if R has FIELD
```

These exports come from (p953) DifferentialExtension(R)
where R:CommutativeRing

```
D : (%,(R -> R)) -> % if R has FIELD
D : (%,(R -> R),NonNegativeInteger) -> % if R has FIELD
D : % -> %
    if and(has(R,DifferentialRing),has(R,Field))
    or R has FFIELDC
D : (%,NonNegativeInteger) -> %
    if and(has(R,DifferentialRing),has(R,Field))
    or R has FFIELDC
D : (%,List Symbol,List NonNegativeInteger) -> %
    if and(has(R,PartialDifferentialRing Symbol),has(R,Field))
D : (%,Symbol,NonNegativeInteger) -> %
    if and(has(R,PartialDifferentialRing Symbol),has(R,Field))
D : (%,List Symbol) -> %
    if and(has(R,PartialDifferentialRing Symbol),has(R,Field))
D : (%,Symbol) -> %
    if and(has(R,PartialDifferentialRing Symbol),has(R,Field))
differentiate : % -> %
    if and(has(R,DifferentialRing),has(R,Field))
    or R has FFIELDC
differentiate : (%,NonNegativeInteger) -> %
    if and(has(R,DifferentialRing),has(R,Field))
    or R has FFIELDC
differentiate : (%,List Symbol) -> %
    if and(has(R,PartialDifferentialRing Symbol),has(R,Field))
differentiate : (%,Symbol,NonNegativeInteger) -> %
    if and(has(R,PartialDifferentialRing Symbol),has(R,Field))
differentiate : (%,List Symbol,List NonNegativeInteger) -> %
    if and(has(R,PartialDifferentialRing Symbol),has(R,Field))
differentiate : (%,(R -> R),NonNegativeInteger) -> %
    if R has FIELD
differentiate : (%,Symbol) -> %
    if and(has(R,PartialDifferentialRing Symbol),has(R,Field))
```

These exports come from (p1518) FiniteFieldCategory():

```
charthRoot : % -> % if R has FFIELDC
```

```
 conditionP : Matrix % -> Union(Vector %,"failed")
     if R has FFIELDC
 createPrimitiveElement : () -> % if R has FFIELDC
 discreteLog : % -> NonNegativeInteger if R has FFIELDC
 discreteLog : (%,%) -> Union(NonNegativeInteger,"failed")
     if R has FFIELDC
 factorsOfCyclicGroupSize : () ->
    List Record(factor: Integer,exponent: Integer)
      if R has FFIELDC
 init : () -> % if R has FFIELDC
 nextItem : % -> Union(%,"failed") if R has FFIELDC
 order : % -> OnePointCompletion PositiveInteger
     if R has FFIELDC
 order : % -> PositiveInteger if R has FFIELDC
 primeFrobenius : % -> % if R has FFIELDC
 primeFrobenius : (%,NonNegativeInteger) -> % if R has FFIELDC
 primitive? : % -> Boolean if R has FFIELDC
 primitiveElement : () -> % if R has FFIELDC
 representationType : () ->
    Union("prime",polynomial,normal,cyclic)
      if R has FFIELDC
 tableForDiscreteLogarithm :
    Integer -> Table(PositiveInteger,NonNegativeInteger)
      if R has FFIELDC
```

— category MONOGEN MonogenicAlgebra —

```
)abbrev category MONOGEN MonogenicAlgebra
++ Author: Barry Trager
++ Date Created:
++ Date Last Updated:
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A \spadtype{MonogenicAlgebra} is an algebra of finite rank which
++ can be generated by a single element.

MonogenicAlgebra(R:CommutativeRing, UP:UnivariatePolynomialCategory R):
 Category ==
    Join(FramedAlgebra(R, UP), CommutativeRing, ConvertibleTo UP,
              FullyRetractableTo R, FullyLinearlyExplicitRingOver R) with
      generator         : () -> %
        ++ generator() returns the generator for this domain.
      definingPolynomial: () -> UP
        ++ definingPolynomial() returns the minimal polynomial which
```

```
      ++ \spad{generator()} satisfies.
    reduce            : UP -> %
      ++ reduce(up) converts the univariate polynomial up to an algebra
      ++ element, reducing by the \spad{definingPolynomial()} if necessary.
    convert           : UP -> %
      ++ convert(up) converts the univariate polynomial up to an algebra
      ++ element, reducing by the \spad{definingPolynomial()} if necessary.
    lift              : % -> UP
      ++ lift(z) returns a minimal degree univariate polynomial up such that
      ++ \spad{z=reduce up}.
    if R has Finite then Finite
    if R has Field then
      Field
      DifferentialExtension R
      reduce              : Fraction UP -> Union(%, "failed")
        ++ reduce(frac) converts the fraction frac to an algebra element.
      derivationCoordinates: (Vector %, R -> R) -> Matrix R
        ++ derivationCoordinates(b, ') returns M such that \spad{b' = M b}.
    if R has FiniteFieldCategory then FiniteFieldCategory
add
 convert(x:%):UP == lift x
 convert(p:UP):% == reduce p
 generator()     == reduce monomial(1, 1)$UP
 norm x          == resultant(definingPolynomial(), lift x)
 retract(x:%):R  == retract lift x
 retractIfCan(x:%):Union(R, "failed") == retractIfCan lift x

 basis() ==
   [reduce monomial(1,i)$UP for i in 0..(rank()-1)::NonNegativeInteger]

 characteristicPolynomial(x:%):UP ==
   characteristicPolynomial(x)$CharacteristicPolynomialInMonogenicalAlgebra(R,UP,%)

 if R has Finite then
   size()   == size()$R ** rank()
   random() == represents [random()$R for i in 1..rank()]$Vector(R)

 if R has Field then
   reduce(x:Fraction UP) == reduce(numer x) exquo reduce(denom x)

   differentiate(x:%, d:R -> R) ==
     p := definingPolynomial()
     yprime := - reduce(map(d, p)) / reduce(differentiate p)
     reduce(map(d, lift x)) + yprime * reduce differentiate lift x

   derivationCoordinates(b, d) ==
     coordinates(map(x +-> differentiate(x, d), b), b)

   recip x ==
     (bc := extendedEuclidean(lift x, definingPolynomial(), 1))
```

```
                                                case "failed" => "failed"
      reduce(bc.coef1)
```

———-

— **MONOGEN.dotabb** —

```
"MONOGEN"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=MONOGEN"];
"MONOGEN" -> "FRAMALG"
"MONOGEN" -> "COMRING"
"MONOGEN" -> "KONVERT"
"MONOGEN" -> "FRETRCT"
"MONOGEN" -> "FLINEXP"
"MONOGEN" -> "FINITE"
"MONOGEN" -> "FIELD"
"MONOGEN" -> "DIFEXT"
"MONOGEN" -> "FFIELDC"
```

———-

— **MONOGEN.dotfull** —

```
"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=MONOGEN"];
"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "FramedAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))"
"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "CommutativeRing()"
"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "ConvertibleTo(UnivariatePolynomialCategory(CommutativeRing))"
"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "FullyRetractableTo(a:CommutativeRing)"
"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "FullyLinearlyExplicitRingOver(a:CommutativeRing)"

"MonogenicAlgebra(a:FRAC(UPOLYC(UFD)),b:UPOLYC(FRAC(UPOLYC(UFD))))"
 [color=seagreen,href="bookvol10.2.pdf#nameddest=MONOGEN"];
"MonogenicAlgebra(a:FRAC(UPOLYC(UFD)),b:UPOLYC(FRAC(UPOLYC(UFD))))" ->
    "MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))"
```

———-

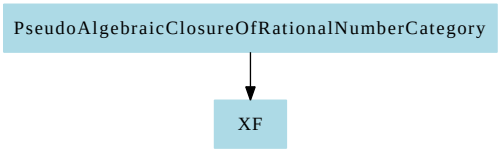
— **MONOGEN.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))"
 [color=lightblue];
"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "FRAMALG..."
"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "COMRING..."
"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "KONVERT..."
"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "FRETRCT..."
"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "FLINEXP..."
"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "FINITE..."
"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "FIELD..."
"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "DIFEXT..."
"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "FFIELDC..."

"FRAMALG..." [color=lightblue];
"COMRING..." [color=lightblue];
"KONVERT..." [color=lightblue];
"FRETRCT..." [color=lightblue];
"FLINEXP..." [color=lightblue];
"FINITE..." [color=lightblue];
"FIELD..." [color=lightblue];
"DIFEXT..." [color=lightblue];
"FFIELDC..." [color=lightblue];
}
```

19.0.215 PseudoAlgebraicClosureOfRationalNumberCategory (PACRATC)



— PseudoAlgebraicClosureOfRationalNumberCategory.input —

```
)set break resume
)sys rm -f PseudoAlgebraicClosureOfRationalNumberCategory.output
)spool PseudoAlgebraicClosureOfRationalNumberCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show PseudoAlgebraicClosureOfRationalNumberCategory
--R
--R PseudoAlgebraicClosureOfRationalNumberCategory  is a category constructor
--R Abbreviation for PseudoAlgebraicClosureOfRationalNumberCategory is PACRATC
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PACRATC
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (%,Fraction Integer) -> %       ?*? : (Fraction Integer,%) -> %
--R ?*? : (Fraction Integer,%) -> %       ?*? : (%,Fraction Integer) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,Integer) -> %
--R ?**? : (%,PositiveInteger) -> %       ?+? : (%,%) -> %
--R ?-? : (%,%) -> %                      -? : % -> %
--R ?/? : (%,Fraction Integer) -> %       ?/? : (%,%) -> %
--R ?=? : (%,%) -> Boolean                1 : () -> %
--R 0 : () -> %                           ?^? : (%,Integer) -> %
--R ?^? : (%,PositiveInteger) -> %        algebraic? : % -> Boolean
--R associates? : (%,%) -> Boolean        coerce : Fraction Integer -> %
--R coerce : Fraction Integer -> %        coerce : Integer -> %
--R coerce : Fraction Integer -> %        coerce : % -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R conjugate : % -> %                    dimension : () -> CardinalNumber
--R extDegree : % -> PositiveInteger      factor : % -> Factored %
--R fullOutput : % -> OutputForm          gcd : List % -> %
--R gcd : (%,%) -> %                      ground? : % -> Boolean
--R hash : % -> SingleInteger             inGroundField? : % -> Boolean
--R inv : % -> %                          latex : % -> String
--R lcm : List % -> %                     lcm : (%,%) -> %
--R maxTower : List % -> %                one? : % -> Boolean
--R previousTower : % -> %                prime? : % -> Boolean
--R ?quo? : (%,%) -> %                   recip : % -> Union(%,"failed")
--R ?rem? : (%,%) -> %                   retract : % -> Fraction Integer
--R retract : % -> Fraction Integer       retract : % -> Integer
--R sample : () -> %                      setTower! : % -> Void
--R sizeLess? : (%,%) -> Boolean          squareFree : % -> Factored %
--R squareFreePart : % -> %               transcendent? : % -> Boolean
```

```
--R unit? : % -> Boolean                  unitCanonical : % -> %
--R vectorise : (%,%) -> Vector %         zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R Frobenius : % -> % if Fraction Integer has FINITE
--R Frobenius : (%,NonNegativeInteger) -> % if Fraction Integer has FINITE
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if Fraction Integer has CHARNZ or Fraction Integer has F
--R definingPolynomial : % -> SparseUnivariatePolynomial %
--R definingPolynomial : () -> SparseUnivariatePolynomial %
--R degree : % -> OnePointCompletion PositiveInteger
--R discreteLog : (%,%) -> Union(NonNegativeInteger,"failed") if Fraction Integer has CHARNZ or
--R distinguishedRootsOf : (SparseUnivariatePolynomial %,%) -> List %
--R divide : (%,%) -> Record(quotient: %,remainder: %)
--R euclideanSize : % -> NonNegativeInteger
--R expressIdealMember : (List %,%) -> Union(List %,"failed")
--R exquo : (%,%) -> Union(%,"failed")
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
--R extensionDegree : () -> OnePointCompletion PositiveInteger
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivari
--R lift : (%,%) -> SparseUnivariatePolynomial %
--R lift : % -> SparseUnivariatePolynomial %
--R multiEuclidean : (List %,%) -> Union(List %,"failed")
--R newElement : (SparseUnivariatePolynomial %,Symbol) -> %
--R newElement : (SparseUnivariatePolynomial %,%,Symbol) -> %
--R order : % -> OnePointCompletion PositiveInteger if Fraction Integer has CHARNZ or Fraction I
--R primeFrobenius : % -> % if Fraction Integer has CHARNZ or Fraction Integer has FINITE
--R primeFrobenius : (%,NonNegativeInteger) -> % if Fraction Integer has CHARNZ or Fraction Inte
--R principalIdeal : List % -> Record(coef: List %,generator: %)
--R reduce : SparseUnivariatePolynomial % -> %
--R retractIfCan : % -> Union(Fraction Integer,"failed")
--R retractIfCan : % -> Union(Fraction Integer,"failed")
--R retractIfCan : % -> Union(Integer,"failed")
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R transcendenceDegree : () -> NonNegativeInteger
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R
--E 1

)spool
)lisp (bye)
```

———-

— PseudoAlgebraicClosureOfRationalNumberCategory.help —

```
=====================================================================
PseudoAlgebraicClosureOfRationalNumberCategory examples
=====================================================================

See Also:
o )show PseudoAlgebraicClosureOfRationalNumberCategory
```

———-

See:

⇒ "PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory" (PACEXTC) 20.0.218 on page 1665
⇐ "ExtensionField" (XF) 18.0.206 on page 1510

Exports:

0	1	algebraic?
associates?	characteristic	charthRoot
coerce	conjugate	definingPolynomial
degree	dimension	discreteLog
distinguishedRootsOf	divide	euclideanSize
expressIdealMember	exquo	extDegree
extendedEuclidean	extensionDegree	factor
Frobenius	fullOutput	gcd
gcdPolynomial	ground?	hash
inGroundField?	inv	latex
lcm	lift	maxTower
multiEuclidean	newElement	one?
order	previousTower	prime?
primeFrobenius	principalIdeal	?quo?
recip	reduce	?rem?
retract	retractIfCan	sample
setTower!	sizeLess?	squareFree
squareFreePart	subtractIfCan	transcendenceDegree
transcendent?	unit?	unitCanonical
unitNormal	vectorise	zero?
?*?	?**?	?+?
?-?	-?	?/?
?=?	?~=?	?^?

Attributes Exported:

- **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- **canonicalsClosed** is true if

`unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.

- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These exports come from (p1369) PseudoAlgebraicClosureOfPerfectFieldCategory():

```
?*? : (Fraction Integer,%) -> %
?*? : (%,Fraction Integer) -> %
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?**? : (%,Integer) -> %
?**? : (%,PositiveInteger) -> %
?+? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
?/? : (%,%) -> %
?=? : (%,%) -> Boolean
1 : () -> %
0 : () -> %
?^? : (%,Integer) -> %
?^? : (%,PositiveInteger) -> %
associates? : (%,%) -> Boolean
characteristic : () -> NonNegativeInteger
coerce : Fraction Integer -> %
coerce : Integer -> %
coerce : % -> %
coerce : % -> OutputForm
conjugate : % -> %
definingPolynomial : % -> SUP %
definingPolynomial : () -> SUP %
distinguishedRootsOf : (SparseUnivariatePolynomial %,%) -> List %
divide : (%,%) -> Record(quotient: %,remainder: %)
euclideanSize : % -> NonNegativeInteger
expressIdealMember : (List %,%) -> Union(List %,"failed")
exquo : (%,%) -> Union(%,"failed")
extDegree : % -> PI
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
factor : % -> Factored %
```

```
fullOutput : % -> OutputForm
gcd : List % -> %
gcd : (%,%) -> %
gcdPolynomial :
  (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) ->
     SparseUnivariatePolynomial %
ground? : % -> Boolean
hash : % -> SingleInteger
inv : % -> %
latex : % -> String
lcm : List % -> %
lcm : (%,%) -> %
lift : (%,%) -> SUP %
lift : % -> SUP %
maxTower : List % -> %
multiEuclidean : (List %,%) -> Union(List %,"failed")
newElement : (SUP %,Symbol) -> %
newElement : (SUP %,%,Symbol) -> %
one? : % -> Boolean
previousTower : % -> %
prime? : % -> Boolean
principalIdeal : List % -> Record(coef: List %,generator: %)
?quo? : (%,%) -> %
recip : % -> Union(%,"failed")
reduce : SUP % -> %
?rem? : (%,%) -> %
sample : () -> %
setTower! : % -> Void
sizeLess? : (%,%) -> Boolean
squareFree : % -> Factored %
squareFreePart : % -> %
subtractIfCan : (%,%) -> Union(%,"failed")
unit? : % -> Boolean
unitCanonical : % -> %
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
vectorise : (%,%) -> Vector %
zero? : % -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,NonNegativeInteger) -> %
```

These exports come from (p838) CharacteristicZero():

These exports come from (p55) RetractableTo(Integer):

```
retract : % -> Integer
retractIfCan : % -> Union(Integer,"failed")
```

These exports come from (p55) RetractableTo(Fraction(Integer)):

```
retract : % -> Fraction Integer
retractIfCan : % -> Union(Fraction Integer,"failed")
```

These exports come from (p1510) ExtensionField(Fraction(Integer)):

```
?/? : (%,Fraction Integer) -> %
algebraic? : % -> Boolean
charthRoot : % -> Union(%,"failed")
  if Fraction Integer has CHARNZ or Fraction Integer has FINITE
degree : % -> OnePointCompletion PositiveInteger
dimension : () -> CardinalNumber
discreteLog : (%,%) -> Union(NonNegativeInteger,"failed")
  if Fraction Integer has CHARNZ or Fraction Integer has FINITE
extensionDegree : () -> OnePointCompletion PositiveInteger
Frobenius : % -> % if Fraction Integer has FINITE
Frobenius : (%,NonNegativeInteger) -> % if Fraction Integer has FINITE
inGroundField? : % -> Boolean
order : % -> OnePointCompletion PositiveInteger
  if Fraction Integer has CHARNZ or Fraction Integer has FINITE
primeFrobenius : % -> %
  if Fraction Integer has CHARNZ or Fraction Integer has FINITE
transcendent? : % -> Boolean
primeFrobenius : (%,NonNegativeInteger) -> %
  if Fraction Integer has CHARNZ or Fraction Integer has FINITE
retractIfCan : % -> Union(Fraction Integer,"failed")
transcendenceDegree : () -> NonNegativeInteger
```

— category PACRATC PseudoAlgebraicClosureOfRationalNumberCategory —

```
)abbrev category  PACRATC PseudoAlgebraicClosureOfRationalNumberCategory
++ Authors: Gaetan Hache
++ Date Created: feb 1997
++ Date Last Updated:
++ Description: This category exports the function for the domain
++ PseudoAlgebraicClosureOfRationalNumber
++ which implement dynamic extension using the simple notion of tower
++ extensions. A tower extension T  of the ground
++ field K is any sequence of field extension (T : K_0, K_1, ..., K_i...,K_n)
++ where K_0 = K and for i =1,2,...,n, K_i is an extension
++ of K_{i-1} of degree > 1 and defined by an irreducible polynomial
++ p(Z) in K_{i-1}.
++ Two towers (T_1: K_01, K_11,...,K_i1,...,K_n1)
++ and (T_2: K_02, K_12,...,K_i2,...,K_n2)
++ are said to be related if T_1 <= T_2 (or T_1 >= T_2),
++ that is if K_i1 = K_i2 for i=1,2,...,n1
```

```
++ (or i=1,2,...,n2). Any algebraic operations defined for several elements
++ are only defined if all of the concerned elements are comming from
++ a set of related tour extensions.
PseudoAlgebraicClosureOfRationalNumberCategory:Category ==
  Join(PseudoAlgebraicClosureOfPerfectFieldCategory,
       CharacteristicZero,RetractableTo(Integer),
        RetractableTo(Fraction(Integer)),ExtensionField(Fraction(Integer)))
```

———-

— PACRATC.dotabb —

```
"PACRATC" [color=lightblue,href="bookvol10.2.pdf#nameddest=PACRATC"];
"PACRATC" -> "XF"
```

———-

— PACRATC.dotfull —

```
"PseudoAlgebraicClosureOfRationalNumberCategory"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PACRATC"];
"PseudoAlgebraicClosureOfRationalNumberCategory" -> "ExtensionField(F:Field)"
```

———-

— PACRATC.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"PseudoAlgebraicClosureOfRationalNumberCategory" [color=lightblue];
"PseudoAlgebraicClosureOfRationalNumberCategory" -> "XF"

"XF" [color=lightblue];

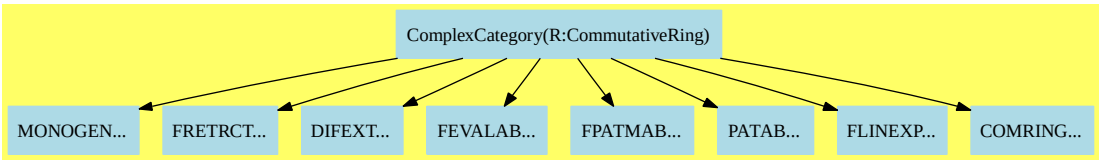
}
```

———-

Chapter 20

Category Layer 19

20.0.216 ComplexCategory (COMPCAT)



— ComplexCategory.input —

```
)set break resume
)sys rm -f ComplexCategory.output
)spool ComplexCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show ComplexCategory
--R ComplexCategory R: CommutativeRing  is a category constructor
--R Abbreviation for ComplexCategory is COMPCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for COMPCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (R,%) -> %                      ?*? : (%,R) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
```

```
--R D : (%,(R -> R)) -> %                --R 1 : () -> %
--R 0 : () -> %                          --R ?^? : (%,PositiveInteger) -> %
--R abs : % -> % if R has RNS            --R basis : () -> Vector %
--R coerce : R -> %                      --R coerce : Integer -> %
--R coerce : % -> OutputForm             --R complex : (R,R) -> %
--R conjugate : % -> %                   --R convert : Vector R -> %
--R convert : % -> Vector R              --R coordinates : % -> Vector R
--R discriminant : () -> R               --R discriminant : Vector % -> R
--R generator : () -> %                  --R hash : % -> SingleInteger
--R imag : % -> R                        --R imaginary : () -> %
--R inv : % -> % if R has FIELD          --R latex : % -> String
--R map : ((R -> R),%) -> %              --R norm : % -> R
--R one? : % -> Boolean                  --R pi : () -> % if R has TRANFUN
--R rank : () -> PositiveInteger         --R real : % -> R
--R recip : % -> Union(%,"failed")       --R represents : Vector R -> %
--R retract : % -> R                     --R sample : () -> %
--R trace : % -> R                       --R traceMatrix : () -> Matrix R
--R zero? : % -> Boolean                 --R ?~=? : (%,%) -> Boolean
--R ?*? : (%,Fraction Integer) -> % if R has FIELD
--R ?*? : (Fraction Integer,%) -> % if R has FIELD
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,Integer) -> % if R has FIELD
--R ?**? : (%,%) -> % if R has TRANFUN
--R ?**? : (%,Fraction Integer) -> % if R has RADCAT and R has TRANFUN
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (%,%) -> % if R has FIELD
--R ?<? : (%,%) -> Boolean if R has ORDSET
--R ?<=? : (%,%) -> Boolean if R has ORDSET
--R ?>? : (%,%) -> Boolean if R has ORDSET
--R ?>=? : (%,%) -> Boolean if R has ORDSET
--R D : % -> % if and(has(R,Field),has(R,DifferentialRing)) or R has DIFRING or and(has(R,Differ
--R D : (%,NonNegativeInteger) -> % if and(has(R,Field),has(R,DifferentialRing)) or R has DIFRIN
--R D : (%,Symbol) -> % if R has PDRING SYMBOL
--R D : (%,List Symbol) -> % if R has PDRING SYMBOL
--R D : (%,Symbol,NonNegativeInteger) -> % if R has PDRING SYMBOL
--R D : (%,List Symbol,List NonNegativeInteger) -> % if R has PDRING SYMBOL
--R D : (%,(R -> R),NonNegativeInteger) -> %
--R ?^? : (%,Integer) -> % if R has FIELD
--R ?^? : (%,NonNegativeInteger) -> %
--R acos : % -> % if R has TRANFUN
--R acosh : % -> % if R has TRANFUN
--R acot : % -> % if R has TRANFUN
--R acoth : % -> % if R has TRANFUN
--R acsc : % -> % if R has TRANFUN
--R acsch : % -> % if R has TRANFUN
--R argument : % -> R if R has TRANFUN
--R asec : % -> % if R has TRANFUN
--R asech : % -> % if R has TRANFUN
--R asin : % -> % if R has TRANFUN
--R asinh : % -> % if R has TRANFUN
```

```
--R associates? : (%,%) -> Boolean if R has INTDOM or R has EUCDOM and R has PFECAT
--R atan : % -> % if R has TRANFUN
--R atanh : % -> % if R has TRANFUN
--R characteristic : () -> NonNegativeInteger
--R characteristicPolynomial : % -> SparseUnivariatePolynomial R
--R charthRoot : % -> Union(%,"failed") if and(has($,CharacteristicNonZero),AND(has(R,EuclideanDomain),has(
--R charthRoot : % -> % if R has FFIELDC
--R coerce : % -> % if R has INTDOM or R has EUCDOM and R has PFECAT
--R coerce : Fraction Integer -> % if R has FIELD or R has RETRACT FRAC INT
--R conditionP : Matrix % -> Union(Vector %,"failed") if and(has($,CharacteristicNonZero),AND(has(R,Euclide
--R convert : % -> InputForm if R has KONVERT INFORM
--R convert : % -> Complex DoubleFloat if R has REAL
--R convert : % -> Complex Float if R has REAL
--R convert : % -> Pattern Float if R has KONVERT PATTERN FLOAT
--R convert : % -> Pattern Integer if R has KONVERT PATTERN INT
--R convert : SparseUnivariatePolynomial R -> %
--R convert : % -> SparseUnivariatePolynomial R
--R coordinates : Vector % -> Matrix R
--R coordinates : (Vector %,Vector %) -> Matrix R
--R coordinates : (%,Vector %) -> Vector R
--R cos : % -> % if R has TRANFUN
--R cosh : % -> % if R has TRANFUN
--R cot : % -> % if R has TRANFUN
--R coth : % -> % if R has TRANFUN
--R createPrimitiveElement : () -> % if R has FFIELDC
--R csc : % -> % if R has TRANFUN
--R csch : % -> % if R has TRANFUN
--R definingPolynomial : () -> SparseUnivariatePolynomial R
--R derivationCoordinates : (Vector %,(R -> R)) -> Matrix R if R has FIELD
--R differentiate : % -> % if and(has(R,Field),has(R,DifferentialRing)) or R has DIFRING or and(has(R,Diffe
--R differentiate : (%,NonNegativeInteger) -> % if and(has(R,Field),has(R,DifferentialRing)) or R has DIFRI
--R differentiate : (%,Symbol) -> % if R has PDRING SYMBOL
--R differentiate : (%,List Symbol) -> % if R has PDRING SYMBOL
--R differentiate : (%,Symbol,NonNegativeInteger) -> % if R has PDRING SYMBOL
--R differentiate : (%,List Symbol,List NonNegativeInteger) -> % if R has PDRING SYMBOL
--R differentiate : (%,(R -> R)) -> %
--R differentiate : (%,(R -> R),NonNegativeInteger) -> %
--R discreteLog : (%,%) -> Union(NonNegativeInteger,"failed") if R has FFIELDC
--R discreteLog : % -> NonNegativeInteger if R has FFIELDC
--R divide : (%,%) -> Record(quotient: %,remainder: %) if R has EUCDOM
--R ?.? : (%,R) -> % if R has ELTAB(R,R)
--R euclideanSize : % -> NonNegativeInteger if R has EUCDOM
--R eval : (%,Symbol,R) -> % if R has IEVALAB(SYMBOL,R)
--R eval : (%,List Symbol,List R) -> % if R has IEVALAB(SYMBOL,R)
--R eval : (%,List Equation R) -> % if R has EVALAB R
--R eval : (%,Equation R) -> % if R has EVALAB R
--R eval : (%,R,R) -> % if R has EVALAB R
--R eval : (%,List R,List R) -> % if R has EVALAB R
--R exp : % -> % if R has TRANFUN
--R expressIdealMember : (List %,%) -> Union(List %,"failed") if R has EUCDOM
```

```
--R exquo : (%,%) -> Union(%,"failed") if R has INTDOM or R has EUCDOM and R has PFECAT
--R exquo : (%,R) -> Union(%,"failed") if R has INTDOM
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %) if R has EUCDOM
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed") if R has EUCDOM
--R factor : % -> Factored % if R has EUCDOM and R has PFECAT or R has FIELD
--R factorPolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial % if
--R factorSquareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolyno
--R factorsOfCyclicGroupSize : () -> List Record(factor: Integer,exponent: Integer) if R has FFI
--R gcd : (%,%) -> % if R has EUCDOM or R has EUCDOM and R has PFECAT
--R gcd : List % -> % if R has EUCDOM or R has EUCDOM and R has PFECAT
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivari
--R index : PositiveInteger -> % if R has FINITE
--R init : () -> % if R has FFIELDC
--R lcm : (%,%) -> % if R has EUCDOM or R has EUCDOM and R has PFECAT
--R lcm : List % -> % if R has EUCDOM or R has EUCDOM and R has PFECAT
--R lift : % -> SparseUnivariatePolynomial R
--R log : % -> % if R has TRANFUN
--R lookup : % -> PositiveInteger if R has FINITE
--R max : (%,%) -> % if R has ORDSET
--R min : (%,%) -> % if R has ORDSET
--R minimalPolynomial : % -> SparseUnivariatePolynomial R if R has FIELD
--R multiEuclidean : (List %,%) -> Union(List %,"failed") if R has EUCDOM
--R nextItem : % -> Union(%,"failed") if R has FFIELDC
--R nthRoot : (%,Integer) -> % if R has RADCAT and R has TRANFUN
--R order : % -> OnePointCompletion PositiveInteger if R has FFIELDC
--R order : % -> PositiveInteger if R has FFIELDC
--R patternMatch : (%,Pattern Float,PatternMatchResult(Float,%)) -> PatternMatchResult(Float,%)
--R patternMatch : (%,Pattern Integer,PatternMatchResult(Integer,%)) -> PatternMatchResult(Integ
--R polarCoordinates : % -> Record(r: R,phi: R) if R has RNS and R has TRANFUN
--R prime? : % -> Boolean if R has EUCDOM and R has PFECAT or R has FIELD
--R primeFrobenius : % -> % if R has FFIELDC
--R primeFrobenius : (%,NonNegativeInteger) -> % if R has FFIELDC
--R primitive? : % -> Boolean if R has FFIELDC
--R primitiveElement : () -> % if R has FFIELDC
--R principalIdeal : List % -> Record(coef: List %,generator: %) if R has EUCDOM
--R ?quo? : (%,%) -> % if R has EUCDOM
--R random : () -> % if R has FINITE
--R rational : % -> Fraction Integer if R has INS
--R rational? : % -> Boolean if R has INS
--R rationalIfCan : % -> Union(Fraction Integer,"failed") if R has INS
--R reduce : Fraction SparseUnivariatePolynomial R -> Union(%,"failed") if R has FIELD
--R reduce : SparseUnivariatePolynomial R -> %
--R reducedSystem : Matrix % -> Matrix R
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix R,vec: Vector R)
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer) if R
--R reducedSystem : Matrix % -> Matrix Integer if R has LINEXP INT
--R regularRepresentation : % -> Matrix R
--R regularRepresentation : (%,Vector %) -> Matrix R
--R ?rem? : (%,%) -> % if R has EUCDOM
--R representationType : () -> Union("prime",polynomial,normal,cyclic) if R has FFIELDC
```

```
--R represents : (Vector R,Vector %) -> %
--R retract : % -> Fraction Integer if R has RETRACT FRAC INT
--R retract : % -> Integer if R has RETRACT INT
--R retractIfCan : % -> Union(R,"failed")
--R retractIfCan : % -> Union(Fraction Integer,"failed") if R has RETRACT FRAC INT
--R retractIfCan : % -> Union(Integer,"failed") if R has RETRACT INT
--R sec : % -> % if R has TRANFUN
--R sech : % -> % if R has TRANFUN
--R sin : % -> % if R has TRANFUN
--R sinh : % -> % if R has TRANFUN
--R size : () -> NonNegativeInteger if R has FINITE
--R sizeLess? : (%,%) -> Boolean if R has EUCDOM
--R solveLinearPolynomialEquation : (List SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> Unio
--R sqrt : % -> % if R has RADCAT and R has TRANFUN
--R squareFree : % -> Factored % if R has EUCDOM and R has PFECAT or R has FIELD
--R squareFreePart : % -> % if R has EUCDOM and R has PFECAT or R has FIELD
--R squareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial % if R has EU
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R tableForDiscreteLogarithm : Integer -> Table(PositiveInteger,NonNegativeInteger) if R has FFIELDC
--R tan : % -> % if R has TRANFUN
--R tanh : % -> % if R has TRANFUN
--R traceMatrix : Vector % -> Matrix R
--R unit? : % -> Boolean if R has INTDOM or R has EUCDOM and R has PFECAT
--R unitCanonical : % -> % if R has INTDOM or R has EUCDOM and R has PFECAT
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %) if R has INTDOM or R has EUCDOM and R has PF
--R
--E 1

)spool
)lisp (bye)
```

———-

— ComplexCategory.help —

```
=====================================================================
ComplexCategory examples
=====================================================================

See Also:
o )show ComplexCategory
```

———-

See:

⇐ “CommutativeRing” (COMRING) 10.0.137 on page 843
⇐ “DifferentialExtension” (DIFEXT) 11.0.153 on page 953

⇐ "FullyEvalableOver" (FEVALAB) 4.0.44 on page 159
⇐ "FullyPatternMatchable" (FPATMAB) 3.0.28 on page 91
⇐ "FullyRetractableTo" (FRETRCT) 3.0.27 on page 87
⇐ "MonogenicAlgebra" (MONOGEN) 19.0.214 on page 1598
⇐ "Patternable" (PATAB) 2.0.15 on page 47
⇐ "FullyLinearlyExplicitRingOver" (FLINEXP) 11.0.154 on page 960

Exports:

0	1	abs
acos	acosh	acot
acoth	acsc	acsch
argument	asec	asech
asin	asinh	associates?
atan	atanh	basis
characteristic	characteristicPolynomial	charthRoot
coerce	complex	conditionP
conjugate	convert	coordinates
cos	cosh	cot
coth	createPrimitiveElement	csc
csch	D	definingPolynomial
derivationCoordinates	differentiate	discreteLog
discriminant	divide	euclideanSize
eval	exp	expressIdealMember
exquo	extendedEuclidean	factor
factorPolynomial	factorSquareFreePolynomial	factorsOfCyclicGroupSize
gcd	gcdPolynomial	generator
hash	imag	imaginary
index	init	inv
latex	lcm	lift
log	lookup	map
max	min	minimalPolynomial
multiEuclidean	nextItem	norm
nthRoot	one?	order
patternMatch	pi	polarCoordinates
prime?	primeFrobenius	primitive?
primitiveElement	principalIdeal	random
rank	rational	rational?
rationalIfCan	real	recip
reduce	reducedSystem	regularRepresentation
represents	representationType	retract
retractIfCan	sample	sec
sech	sin	sinh
size	sizeLess?	solveLinearPolynomialEquation
sqrt	squareFree	squareFreePart
squareFreePolynomial	subtractIfCan	tableForDiscreteLogarithm
tan	tanh	trace
traceMatrix	unit?	unitCanonical
unitNormal	zero?	?*?
?**?	?+?	?-?
-?	?=?	?^?
?~=?	?/?	?<?
?<=?	?>?	?>=?
?.?	?quo?	?rem?

Attributes Exported:

- if #1 has multiplicativeValuation then multiplicativeValuation where **multiplicativeValuation** implies `euclideanSize(a*b)=euclideanSize(a)*euclideanSize(b)`.
- if #1 has additiveValuation then additiveValuation where **additiveValuation** implies `euclideanSize(a*b)=euclideanSize(a)+euclideanSize(b)`.
- if #1 has Field then canonicalsClosed where **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.
- if #1 has Field then canonicalUnitNormal where **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- if #1 has IntegralDomain or #1 has both EuclideanDomain and PolynomialFactorizationExplicit then noZeroDivisors where **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.
- **complex** means that this domain has $\sqrt{-1}$
- **nil**

These are directly exported but not implemented:

```
abs : % -> % if R has RNS
argument : % -> R if R has TRANFUN
complex : (R,R) -> %
conjugate : % -> %
exquo : (%,R) -> Union(%,"failed") if R has INTDOM
imag : % -> R
imaginary : () -> %
norm : % -> R
polarCoordinates : % -> Record(r: R,phi: R) if R has RNS and R has TRANFUN
rational : % -> Fraction Integer if R has INS
rational? : % -> Boolean if R has INS
rationalIfCan : % -> Union(Fraction Integer,"failed") if R has INS
real : % -> R
```

These are implemented by this category:

```
acos : % -> % if R has TRANFUN
acosh : % -> % if R has TRANFUN
asin : % -> % if R has TRANFUN
asinh : % -> % if R has TRANFUN
atan : % -> % if R has TRANFUN
atanh : % -> % if R has TRANFUN
characteristic : () -> NonNegativeInteger
characteristicPolynomial : % -> SparseUnivariatePolynomial R
coerce : % -> OutputForm
convert : % -> Complex DoubleFloat if R has REAL
convert : % -> Complex Float if R has REAL
convert : % -> InputForm if R has KONVERT INFORM
convert : % -> Pattern Float if R has KONVERT PATTERN FLOAT
convert : % -> Pattern Integer if R has KONVERT PATTERN INT
coordinates : % -> Vector R
coordinates : (%,Vector %) -> Vector R
cos : % -> % if R has TRANFUN
cosh : % -> % if R has TRANFUN
definingPolynomial : () -> SparseUnivariatePolynomial R
differentiate : (%,(R -> R)) -> %
discriminant : () -> R
divide : (%,%) -> Record(quotient: %,remainder: %) if R has EUCDOM
euclideanSize : % -> NonNegativeInteger if R has EUCDOM
exp : % -> % if R has TRANFUN
exquo : (%,%) -> Union(%,"failed") if R has INTDOM or R has EUCDOM and R has PFECAT
factorPolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial % if R has EUCDOM ar
factorSquareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial % if R has
inv : % -> % if R has FIELD
lift : % -> SparseUnivariatePolynomial R
log : % -> % if R has TRANFUN
map : ((R -> R),%) -> %
minimalPolynomial : % -> SparseUnivariatePolynomial R if R has FIELD
patternMatch : (%,Pattern Integer,PatternMatchResult(Integer,%)) -> PatternMatchResult(Integer,%) if R has
patternMatch : (%,Pattern Float,PatternMatchResult(Float,%)) -> PatternMatchResult(Float,%) if R has PATMA
pi : () -> % if R has TRANFUN
rank : () -> PositiveInteger
recip : % -> Union(%,"failed")
reduce : SparseUnivariatePolynomial R -> %
reducedSystem : Matrix % -> Matrix R
reducedSystem : Matrix % -> Matrix Integer if R has LINEXP INT
retract : % -> R
retractIfCan : % -> Union(R,"failed")
sin : % -> % if R has TRANFUN
sinh : % -> % if R has TRANFUN
solveLinearPolynomialEquation : (List SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> Union(
tan : % -> % if R has TRANFUN
tanh : % -> % if R has TRANFUN
trace : % -> R
unitNormal : % -> Record(unit: %,canonical: %,associate: %) if R has INTDOM or R has EUCDOM and R has PFEC
?=? : (%,%) -> Boolean
```

```
?+? : (%,%) -> %
-? : % -> %
?*? : (%,%) -> %
?*? : (R,%) -> %
?*? : (Integer,%) -> %
?<? : (%,%) -> Boolean if R has ORDSET
?**? : (%,Fraction Integer) -> % if R has RADCAT and R has TRANFUN
?rem? : (%,%) -> % if R has EUCDOM
?quo? : (%,%) -> % if R has EUCDOM
```

These exports come from (p1598) MonogenicAlgebra(R,S)
where R:CommutativeRing and S:SparseUnivariatePolynomial(R):

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean if R has INTDOM or R has EUCDOM and R has PFECAT
basis : () -> Vector %
charthRoot : % -> Union(%,"failed") if and(has($,CharacteristicNonZero),AND(has(R,EuclideanDoma
charthRoot : % -> % if R has FFIELDC
coerce : R -> %
coerce : % -> % if R has INTDOM or R has EUCDOM and R has PFECAT
coerce : Integer -> %
coerce : Fraction Integer -> % if R has FIELD or R has RETRACT FRAC INT
conditionP : Matrix % -> Union(Vector %,"failed") if and(has($,CharacteristicNonZero),AND(has(R
convert : SparseUnivariatePolynomial R -> %
convert : Vector R -> %
convert : % -> Vector R
convert : % -> SparseUnivariatePolynomial R
coordinates : Vector % -> Matrix R
coordinates : (Vector %,Vector %) -> Matrix R
createPrimitiveElement : () -> % if R has FFIELDC
D : (%,(R -> R)) -> %
D : (%,(R -> R),NonNegativeInteger) -> %
D : % -> % if and(has(R,Field),has(R,DifferentialRing)) or R has DIFRING or and(has(R,Different
D : (%,NonNegativeInteger) -> % if and(has(R,Field),has(R,DifferentialRing)) or R has DIFRING o
D : (%,List Symbol,List NonNegativeInteger) -> % if R has PDRING SYMBOL
D : (%,Symbol,NonNegativeInteger) -> % if R has PDRING SYMBOL
D : (%,List Symbol) -> % if R has PDRING SYMBOL
D : (%,Symbol) -> % if R has PDRING SYMBOL
derivationCoordinates : (Vector %,(R -> R)) -> Matrix R if R has FIELD
differentiate : (%,NonNegativeInteger) -> % if and(has(R,Field),has(R,DifferentialRing)) or R h
differentiate : (%,List Symbol) -> % if R has PDRING SYMBOL
differentiate : (%,Symbol,NonNegativeInteger) -> % if R has PDRING SYMBOL
differentiate : (%,List Symbol,List NonNegativeInteger) -> % if R has PDRING SYMBOL
differentiate : (%,(R -> R),NonNegativeInteger) -> %
differentiate : % -> % if and(has(R,Field),has(R,DifferentialRing)) or R has DIFRING or and(has
differentiate : (%,Symbol) -> % if R has PDRING SYMBOL
discreteLog : (%,%) -> Union(NonNegativeInteger,"failed") if R has FFIELDC
discreteLog : % -> NonNegativeInteger if R has FFIELDC
discriminant : Vector % -> R
```

```
expressIdealMember : (List %,%) -> Union(List %,"failed") if R has EUCDOM
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %) if R has EUCDOM
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed") if R has EUCDOM
factor : % -> Factored % if R has EUCDOM and R has PFECAT or R has FIELD
factorsOfCyclicGroupSize : () -> List Record(factor: Integer,exponent: Integer) if R has FFIELDC
gcd : (%,%) -> % if R has EUCDOM or R has EUCDOM and R has PFECAT
gcd : List % -> % if R has EUCDOM or R has EUCDOM and R has PFECAT
gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivariatePolynomial %
generator : () -> %
hash : % -> SingleInteger
index : PositiveInteger -> % if R has FINITE
init : () -> % if R has FFIELDC
latex : % -> String
lcm : (%,%) -> % if R has EUCDOM or R has EUCDOM and R has PFECAT
lcm : List % -> % if R has EUCDOM or R has EUCDOM and R has PFECAT
lookup : % -> PositiveInteger if R has FINITE
multiEuclidean : (List %,%) -> Union(List %,"failed") if R has EUCDOM
nextItem : % -> Union(%,"failed") if R has FFIELDC
one? : % -> Boolean
order : % -> OnePointCompletion PositiveInteger if R has FFIELDC
order : % -> PositiveInteger if R has FFIELDC
prime? : % -> Boolean if R has EUCDOM and R has PFECAT or R has FIELD
primeFrobenius : % -> % if R has FFIELDC
primeFrobenius : (%,NonNegativeInteger) -> % if R has FFIELDC
primitive? : % -> Boolean if R has FFIELDC
primitiveElement : () -> % if R has FFIELDC
principalIdeal : List % -> Record(coef: List %,generator: %) if R has EUCDOM
random : () -> % if R has FINITE
reduce : Fraction SparseUnivariatePolynomial R -> Union(%,"failed") if R has FIELD
reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix R,vec: Vector R)
reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer) if R has LINEXP INT
regularRepresentation : % -> Matrix R
regularRepresentation : (%,Vector %) -> Matrix R
representationType : () -> Union("prime",polynomial,normal,cyclic) if R has FFIELDC
represents : (Vector R,Vector %) -> %
represents : Vector R -> %
retract : % -> Fraction Integer if R has RETRACT FRAC INT
retract : % -> Integer if R has RETRACT INT
retractIfCan : % -> Union(Fraction Integer,"failed") if R has RETRACT FRAC INT
retractIfCan : % -> Union(Integer,"failed") if R has RETRACT INT
sample : () -> %
size : () -> NonNegativeInteger if R has FINITE
sizeLess? : (%,%) -> Boolean if R has EUCDOM
subtractIfCan : (%,%) -> Union(%,"failed")
squareFree : % -> Factored % if R has EUCDOM and R has PFECAT or R has FIELD
squareFreePart : % -> % if R has EUCDOM and R has PFECAT or R has FIELD
tableForDiscreteLogarithm : Integer -> Table(PositiveInteger,NonNegativeInteger) if R has FFIELDC
traceMatrix : () -> Matrix R
traceMatrix : Vector % -> Matrix R
unit? : % -> Boolean if R has INTDOM or R has EUCDOM and R has PFECAT
```

```
unitCanonical : % -> % if R has INTDOM or R has EUCDOM and R has PFECAT
zero? : % -> Boolean
?/? : (%,%) -> % if R has FIELD
?*? : (%,Fraction Integer) -> % if R has FIELD
?*? : (Fraction Integer,%) -> % if R has FIELD
?**? : (%,Integer) -> % if R has FIELD
?^? : (%,Integer) -> % if R has FIELD
?*? : (PositiveInteger,%) -> %
?*? : (NonNegativeInteger,%) -> %
?*? : (%,R) -> %
?-? : (%,%) -> %
?**? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
?^? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
```

These exports come from (p159) FullyEvalableOver(R:CommutativeRing)

```
?.? : (%,R) -> % if R has ELTAB(R,R)
eval : (%,Equation R) -> % if R has EVALAB R
eval : (%,List Symbol,List R) -> % if R has IEVALAB(SYMBOL,R)
eval : (%,List Equation R) -> % if R has EVALAB R
eval : (%,R,R) -> % if R has EVALAB R
eval : (%,List R,List R) -> % if R has EVALAB R
eval : (%,Symbol,R) -> % if R has IEVALAB(SYMBOL,R)
```

These exports come from (p843) CommutativeRing():

```
?~=? : (%,%) -> Boolean
```

These exports come from (p216) OrderedSet():

```
max : (%,%) -> % if R has ORDSET
min : (%,%) -> % if R has ORDSET
?<=? : (%,%) -> Boolean if R has ORDSET
?>? : (%,%) -> Boolean if R has ORDSET
?>=? : (%,%) -> Boolean if R has ORDSET
```

These exports come from (p52) RadicalCategory():

```
nthRoot : (%,Integer) -> % if R has RADCAT and R has TRANFUN
sqrt : % -> % if R has RADCAT and R has TRANFUN
```

These exports come from (p117) TranscendentalFunctionCategory():

```
acot : % -> % if R has TRANFUN
acoth : % -> % if R has TRANFUN
acsc : % -> % if R has TRANFUN
acsch : % -> % if R has TRANFUN
```

```
 asec : % -> % if R has TRANFUN
 asech : % -> % if R has TRANFUN
 cot : % -> % if R has TRANFUN
 coth : % -> % if R has TRANFUN
 csc : % -> % if R has TRANFUN
 csch : % -> % if R has TRANFUN
 sec : % -> % if R has TRANFUN
 sech : % -> % if R has TRANFUN
 ?**? : (%,%) -> % if R has TRANFUN
```

These exports come from (p1202) PolynomialFactorizationExplicit():

```
 squareFreePolynomial : SparseUnivariatePolynomial % ->
   Factored SparseUnivariatePolynomial % if R has EUCDOM and R has PFECAT
```

— category COMPCAT ComplexCategory —

```
)abbrev category COMPCAT ComplexCategory
++ Author:
++ Date Created:
++ Date Last Updated: 18 March 1994
++ Basic Functions:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords: complex, gaussian
++ References:
++ Description:
++ This category represents the extension of a ring by a square root of -1.

ComplexCategory(R:CommutativeRing): Category ==
  Join(MonogenicAlgebra(R, SparseUnivariatePolynomial R), FullyRetractableTo R,
   DifferentialExtension R, FullyEvalableOver R, FullyPatternMatchable(R),
    Patternable(R), FullyLinearlyExplicitRingOver R, CommutativeRing) with
     complex                      ++ indicates that % has sqrt(-1)
     imaginary:   () -> %         ++ imaginary() = sqrt(-1) = %i.
     conjugate:   % -> %          ++ conjugate(x + %i y) returns x - %i y.
     complex  :   (R, R) -> %     ++ complex(x,y) constructs x + %i*y.
     imag     :   % -> R          ++ imag(x) returns imaginary part of x.
     real     :   % -> R          ++ real(x) returns real part of x.
     norm     :   % -> R          ++ norm(x) returns x * conjugate(x)
     if R has OrderedSet then OrderedSet
     if R has IntegralDomain then
       IntegralDomain
       _exquo : (%,R) -> Union(%,"failed")
         ++ exquo(x, r) returns the exact quotient of x by r, or
         ++ "failed" if r does not divide x exactly.
     if R has EuclideanDomain then EuclideanDomain
     if R has multiplicativeValuation then multiplicativeValuation
```

```
    if R has additiveValuation then additiveValuation
    if R has Field then        -- this is a lie; we must know that
      Field                    -- x**2+1 is irreducible in R
    if R has ConvertibleTo InputForm then ConvertibleTo InputForm
    if R has CharacteristicZero then CharacteristicZero
    if R has CharacteristicNonZero then CharacteristicNonZero
    if R has RealConstant then
      ConvertibleTo Complex DoubleFloat
      ConvertibleTo Complex Float
    if R has RealNumberSystem then
      abs: % -> %
        ++ abs(x) returns the absolute value of x = sqrt(norm(x)).
    if R has TranscendentalFunctionCategory then
      TranscendentalFunctionCategory
      argument: % -> R
        ++ argument(x) returns the angle made by (0,1) and (0,x).
      if R has RadicalCategory then RadicalCategory
      if R has RealNumberSystem then
        polarCoordinates: % -> Record(r:R, phi:R)
          ++ polarCoordinates(x) returns (r, phi) such that
          ++ x = r * exp(%i * phi).
    if R has IntegerNumberSystem then
      rational?    : % -> Boolean
        ++ rational?(x) tests if x is a rational number.
      rational     : % -> Fraction Integer
        ++ rational(x) returns x as a rational number.
        ++ Error: if x is not a rational number.
      rationalIfCan: % -> Union(Fraction Integer, "failed")
        ++ rationalIfCan(x) returns x as a rational number, or
        ++ "failed" if x is not a rational number.
    if R has PolynomialFactorizationExplicit and R has EuclideanDomain then
       PolynomialFactorizationExplicit
add
      import MatrixCategoryFunctions2(%, Vector %, Vector %, Matrix %,
                                      R, Vector R, Vector R, Matrix R)
      SUP ==> SparseUnivariatePolynomial
      characteristicPolynomial x ==
         v := monomial(1,1)$SUP(R)
         v**2 - trace(x)*v**1 + norm(x)*v**0
      if R has PolynomialFactorizationExplicit and R has EuclideanDomain then
         SupR ==> SparseUnivariatePolynomial R
         Sup ==> SparseUnivariatePolynomial %
         import FactoredFunctionUtilities Sup
         import UnivariatePolynomialCategoryFunctions2(R,SupR,%,Sup)
         import UnivariatePolynomialCategoryFunctions2(%,Sup,R,SupR)
         pp,qq:Sup
         if R has IntegerNumberSystem then
            myNextPrime: (%,NonNegativeInteger) -> %
            myNextPrime(x,n ) == -- prime is actually in R, and = 3(mod 4)
               xr:=real(x)-4::R
```

```
          while not prime? xr repeat
             xr:=xr-4::R
          complex(xr,0)
       --!TT:=InnerModularGcd(%,Sup,32719 :: %,myNextPrime)
       --!gcdPolynomial(pp,qq) == modularGcd(pp,qq)$TT
       solveLinearPolynomialEquation(lp:List Sup,p:Sup) ==
         solveLinearPolynomialEquation(lp,p)$ComplexIntegerSolveLinearPolynomialEquation(R,%)
   normPolynomial: Sup -> SupR
   normPolynomial pp ==
       map(z+->retract(z@%)::R,pp * map(conjugate,pp))
   factorPolynomial pp ==
       refine(squareFree pp,factorSquareFreePolynomial)
   factorSquareFreePolynomial pp ==
       pnorm:=normPolynomial pp
       k:R:=0
       while degree gcd(pnorm,differentiate pnorm)>0 repeat
          k:=k+1
          pnorm:=normPolynomial
                   elt(pp,monomial(1,1)-monomial(complex(0,k),0))
       fR:=factorSquareFreePolynomial pnorm
       numberOfFactors fR = 1 =>
           makeFR(1,[["irred",pp,1]])
       lF:List Record(flg:Union("nil", "sqfr", "irred", "prime"),
                      fctr:Sup, xpnt:Integer):=[]
       for u in factorList fR repeat
           p1:=map((z:R):%+->z::%,u.fctr)
           if not zero? k then
              p1:=elt(p1,monomial(1,1)+monomial(complex(0,k),0))
           p2:=gcd(p1,pp)
           lF:=cons(["irred",p2,1],lF)
           pp:=(pp exquo p2)::Sup
       makeFR(pp,lF)
rank()           == 2
discriminant()   == -4 :: R
norm x           == real(x)**2 + imag(x)**2
trace x          == 2 * real x
imaginary()      == complex(0, 1)
conjugate x      == complex(real x, - imag x)
characteristic() == characteristic()$R
map(fn, x)       == complex(fn real x, fn imag x)
x = y            == real(x) = real(y) and imag(x) = imag(y)
x + y            == complex(real x + real y, imag x + imag y)
- x              == complex(- real x, - imag x)
r:R * x:%        == complex(r * real x, r * imag x)
coordinates(x:%) == [real x, imag x]
n:Integer * x:%  == complex(n * real x, n * imag x)
differentiate(x:%, d:R -> R) == complex(d real x, d imag x)

definingPolynomial() ==
  monomial(1,2)$(SUP R) + monomial(1,0)$(SUP R)
```

```
        reduce(pol:SUP R) ==
          part:= (monicDivide(pol,definingPolynomial())).remainder
          complex(coefficient(part,0),coefficient(part,1))

        lift(x) == monomial(real x,0)$(SUP R)+monomial(imag x,1)$(SUP R)

        minimalPolynomial x ==
          zero? imag x =>
            monomial(1, 1)$(SUP R) - monomial(real x, 0)$(SUP R)
          monomial(1, 2)$(SUP R) - monomial(trace x, 1)$(SUP R)
            + monomial(norm x, 0)$(SUP R)

        coordinates(x:%, v:Vector %):Vector(R) ==
          ra := real(a := v(minIndex v))
          rb := real(b := v(maxIndex v))
          (#v ^= 2) or
            ((d := recip(ra * (ib := imag b) - (ia := imag a) * rb))
              case "failed") =>error "coordinates: vector is not a basis"
          rx := real x
          ix := imag x
          [d::R * (rx * ib - ix * rb), d::R * (ra * ix - ia * rx)]

        coerce(x:%):OutputForm ==
          re := (r := real x)::OutputForm
          ie := (i := imag x)::OutputForm
          zero? i => re
          outi := "%i"::Symbol::OutputForm
          ip :=
--            one? i => outi
            (i = 1) => outi
--            one?(-i) => -outi
            ((-i) = 1) => -outi
            ie * outi
          zero? r => ip
          re + ip

        retract(x:%):R ==
          not zero?(imag x) =>
            error "Imaginary part is nonzero. Cannot retract."
          real x

        retractIfCan(x:%):Union(R, "failed") ==
          not zero?(imag x) => "failed"
          real x

        x:% * y:% ==
          complex(real x * real y - imag x * imag y,
                    imag x * real y + imag y * real x)
```

```
reducedSystem(m:Matrix %):Matrix R ==
  vertConcat(map(real, m), map(imag, m))

reducedSystem(m:Matrix %, v:Vector %):
 Record(mat:Matrix R, vec:Vector R) ==
  rh := reducedSystem(v::Matrix %)@Matrix(R)
  [reducedSystem(m)@Matrix(R), column(rh, minColIndex rh)]

if R has RealNumberSystem then
  abs(x:%):%        == (sqrt norm x)::%

if R has RealConstant then
  convert(x:%):Complex(DoubleFloat) ==
   complex(convert(real x)@DoubleFloat,convert(imag x)@DoubleFloat)

  convert(x:%):Complex(Float) ==
    complex(convert(real x)@Float, convert(imag x)@Float)

if R has ConvertibleTo InputForm then
  convert(x:%):InputForm ==
    convert([convert("complex"::Symbol), convert real x,
             convert imag x]$List(InputForm))@InputForm

if R has ConvertibleTo Pattern Integer then
  convert(x:%):Pattern Integer ==
     convert(x)$ComplexPattern(Integer, R, %)
if R has ConvertibleTo Pattern Float then
  convert(x:%):Pattern Float ==
     convert(x)$ComplexPattern(Float, R, %)

if R has PatternMatchable Integer then
  patternMatch(x:%, p:Pattern Integer,
   l:PatternMatchResult(Integer, %)) ==
    patternMatch(x, p, l)$ComplexPatternMatch(Integer, R, %)

if R has PatternMatchable Float then
  patternMatch(x:%, p:Pattern Float,
   l:PatternMatchResult(Float, %)) ==
    patternMatch(x, p, l)$ComplexPatternMatch(Float, R, %)


if R has OrderedSet then
  x < y ==
    real x = real y => imag x < imag y
    real x < real y

if R has IntegerNumberSystem then
  rational? x == zero? imag x

  rational x ==
```

```
          zero? imag x => rational real x
          error "Not a rational number"

        rationalIfCan x ==
          zero? imag x => rational real x
          "failed"

      if R has Field then
        inv x ==
          zero? imag x => (inv real x)::%
          r := norm x
          complex(real(x) / r, - imag(x) / r)

      if R has IntegralDomain then
        _exquo(x:%, r:R) ==
--          one? r => x
          (r = 1) => x
          (r1 := real(x) exquo r) case "failed" => "failed"
          (r2 := imag(x) exquo r) case "failed" => "failed"
          complex(r1, r2)

        _exquo(x:%, y:%) ==
          zero? imag y => x exquo real y
          x * conjugate(y) exquo norm(y)

        recip(x:%) == 1 exquo x

        if R has OrderedRing then
          unitNormal x ==
            zero? x => [1,x,1]
            (u := recip x) case % => [x, 1, u]
            zero? real x =>
              c := unitNormal imag x
              [complex(0, c.unit), (c.associate * imag x)::%,
                                          complex(0, - c.associate)]
            c := unitNormal real x
            x := c.associate * x
            imag x < 0 =>
              x := complex(- imag x, real x)
              [- c.unit * imaginary(), x, c.associate * imaginary()]
            [c.unit ::%, x, c.associate ::%]
          else
            unitNormal x ==
              zero? x => [1,x,1]
              (u := recip x) case % => [x, 1, u]
              zero? real x =>
                c := unitNormal imag x
                [complex(0, c.unit), (c.associate * imag x)::%,
                                          complex(0, - c.associate)]
              c := unitNormal real x
```

```
      x := c.associate * x
      [c.unit ::%, x, c.associate ::%]

if R has EuclideanDomain then
   if R has additiveValuation then
       euclideanSize x == max(euclideanSize real x,
                              euclideanSize imag x)
   else
       euclideanSize x == euclideanSize(real(x)**2 + imag(x)**2)$R
   if R has IntegerNumberSystem then
     x rem y ==
       zero? imag y =>
         yr:=real y
         complex(symmetricRemainder(real(x), yr),
                 symmetricRemainder(imag(x), yr))
       divide(x, y).remainder
     x quo y ==
       zero? imag y =>
         yr:= real y
         xr:= real x
         xi:= imag x
         complex((xr-symmetricRemainder(xr,yr)) quo yr,
                 (xi-symmetricRemainder(xi,yr)) quo yr)
       divide(x, y).quotient

   else
     x rem y ==
       zero? imag y =>
         yr:=real y
         complex(real(x) rem yr,imag(x) rem yr)
       divide(x, y).remainder
     x quo y ==
       zero? imag y => complex(real x quo real y,imag x quo real y)
       divide(x, y).quotient

   divide(x, y) ==
     r := norm y
     y1 := conjugate y
     xx := x * y1
     x1 := real(xx) rem r
     a  := x1
     if x1^=0 and sizeLess?(r, 2 * x1) then
       a := x1 - r
       if sizeLess?(x1, a) then a := x1 + r
     x2 := imag(xx) rem r
     b  := x2
     if x2^=0 and sizeLess?(r, 2 * x2) then
       b := x2 - r
       if sizeLess?(x2, b) then b := x2 + r
     y1 := (complex(a, b) exquo y1)::%
```

```
      [((x - y1) exquo y)::%, y1]

  if R has TranscendentalFunctionCategory then
    half := recip(2::R)::R

    if R has RealNumberSystem then
      atan2loc(y: R, x: R): R ==
          pi1 := pi()$R
          pi2 := pi1 * half
          x = 0 => if y >= 0 then pi2 else -pi2

          -- Atan in (-pi/2,pi/2]
          theta := atan(y * recip(x)::R)
          while theta <= -pi2 repeat theta := theta + pi1
          while theta >   pi2 repeat theta := theta - pi1

          x >= 0 => theta      -- I or IV

          if y >= 0 then
              theta + pi1      -- II
          else
              theta - pi1      -- III

      argument x == atan2loc(imag x, real x)

    else
      -- Not ordered so dictate two quadrants
      argument x ==
        zero? real x => pi()$R * half
        atan(imag(x) * recip(real x)::R)

    pi()  == pi()$R :: %

    if R is DoubleFloat then
       stoc ==> S_-TO_-C$Lisp
       ctos ==> C_-TO_-S$Lisp

       exp   x == ctos EXP(stoc x)$Lisp
       log   x == ctos LOG(stoc x)$Lisp

       sin   x == ctos SIN(stoc x)$Lisp
       cos   x == ctos COS(stoc x)$Lisp
       tan   x == ctos TAN(stoc x)$Lisp
       asin  x == ctos ASIN(stoc x)$Lisp
       acos  x == ctos ACOS(stoc x)$Lisp
       atan  x == ctos ATAN(stoc x)$Lisp

       sinh  x == ctos SINH(stoc x)$Lisp
       cosh  x == ctos COSH(stoc x)$Lisp
       tanh  x == ctos TANH(stoc x)$Lisp
```

```
   asinh x == ctos ASINH(stoc x)$Lisp
   acosh x == ctos ACOSH(stoc x)$Lisp
   atanh x == ctos ATANH(stoc x)$Lisp

else
  atan x ==
    ix := imaginary()*x
    - imaginary() * half * (log(1 + ix) - log(1 - ix))

  log x ==
    complex(log(norm x) * half, argument x)

  exp x ==
    e := exp real x
    complex(e * cos imag x, e * sin imag x)

  cos x ==
    e := exp(imaginary() * x)
    half * (e + recip(e)::%)

  sin x ==
    e := exp(imaginary() * x)
    - imaginary() * half * (e - recip(e)::%)

if R has RealNumberSystem then
  polarCoordinates x ==
    [sqrt norm x, (negative?(t := argument x) => t + 2 * pi(); t)]

  x:% ** q:Fraction(Integer) ==
    zero? q =>
      zero? x => error "0 ** 0 is undefined"
      1
    zero? x => 0
    rx := real x
    zero? imag x and positive? rx => (rx ** q)::%
    zero? imag x and denom q = 2 => complex(0, (-rx)**q)
    ax := sqrt(norm x) ** q
    tx := q::R * argument x
    complex(ax * cos tx, ax * sin tx)

else if R has RadicalCategory then
  x:% ** q:Fraction(Integer) ==
    zero? q =>
      zero? x => error "0 ** 0 is undefined"
      1
    r := real x
    zero?(i := imag x) => (r ** q)::%
    t := numer(q) * recip(denom(q)::R)::R * argument x
    e:R :=
      zero? r => i ** q
```

```
      norm(x) ** (q / (2::Fraction(Integer)))
    complex(e * cos t, e * sin t)
```

— COMPCAT.dotabb —

```
"COMPCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=COMPCAT"];
"COMPCAT" -> "COMRING"
"COMPCAT" -> "DIFEXT"
"COMPCAT" -> "FEVALAB"
"COMPCAT" -> "FPATMAB"
"COMPCAT" -> "FRETRCT"
"COMPCAT" -> "MONOGEN"
"COMPCAT" -> "PATAB"
"COMPCAT" -> "FLINEXP"
"COMPCAT" -> "ORDSET"
```

— COMPCAT.dotfull —

```
"ComplexCategory(R:CommutativeRing)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=COMPCAT"];
"ComplexCategory(R:CommutativeRing)" ->
  "MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))"
"ComplexCategory(R:CommutativeRing)" ->
  "FullyRetractableTo(a:CommutativeRing)"
"ComplexCategory(R:CommutativeRing)" ->
  "DifferentialExtension(CommutativeRing)"
"ComplexCategory(R:CommutativeRing)" ->
  "FullyEvalableOver(CommutativeRing)"
"ComplexCategory(R:CommutativeRing)" ->
  "FullyPatternMatchable(CommutativeRing)"
"ComplexCategory(R:CommutativeRing)" ->
  "Patternable(CommutativeRing)"
"ComplexCategory(R:CommutativeRing)" ->
  "FullyLinearlyExplicitRingOver(a:CommutativeRing)"
"ComplexCategory(R:CommutativeRing)" ->
  "CommutativeRing()"
"ComplexCategory(R:CommutativeRing)" ->
  "OrderedSet()"
```

— COMPCAT.dotpic —

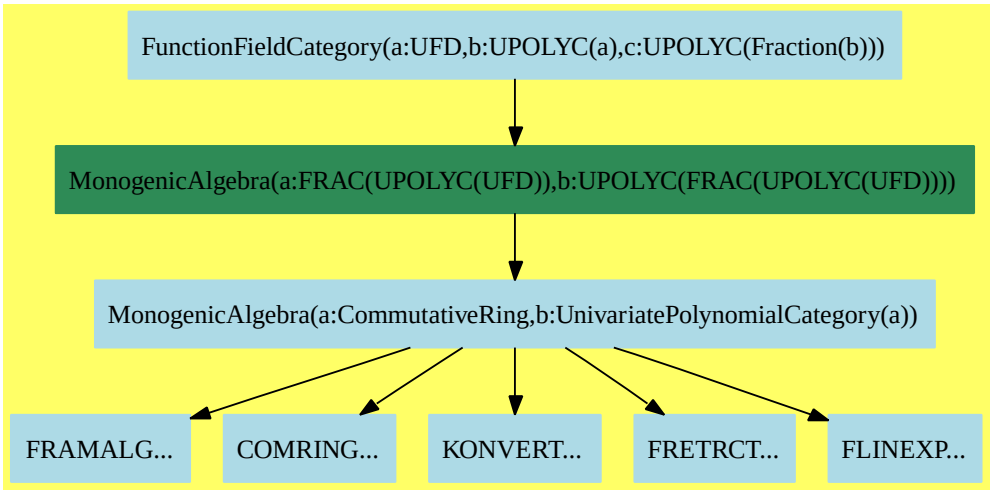
```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"ComplexCategory(R:CommutativeRing)" [color=lightblue];
"ComplexCategory(R:CommutativeRing)" -> "MONOGEN..."
"ComplexCategory(R:CommutativeRing)" -> "FRETRCT..."
"ComplexCategory(R:CommutativeRing)" -> "DIFEXT..."
"ComplexCategory(R:CommutativeRing)" -> "FEVALAB..."
"ComplexCategory(R:CommutativeRing)" -> "FPATMAB..."
"ComplexCategory(R:CommutativeRing)" -> "PATAB..."
"ComplexCategory(R:CommutativeRing)" -> "FLINEXP..."
"ComplexCategory(R:CommutativeRing)" -> "COMRING..."
"ComplexCategory(R:CommutativeRing)" -> "ORDSET..."

"MONOGEN..." [color=lightblue];
"FRETRCT..." [color=lightblue];
"DIFEXT..." [color=lightblue];
"FEVALAB..." [color=lightblue];
"COMRING..." [color=lightblue];
"FPATMAB..." [color=lightblue];
"PATAB..." [color=lightblue];
"FLINEXP..." [color=lightblue];
"ORDSET..." [color=lightblue];
}
```

———-

20.0.217 FunctionFieldCategory (FFCAT)



— **FunctionFieldCategory.input** —

```
)set break resume
)sys rm -f FunctionFieldCategory.output
)spool FunctionFieldCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show FunctionFieldCategory
--R FunctionFieldCategory(F: UniqueFactorizationDomain,UP: UnivariatePolynomialCategory t#1,UPUP
--R Abbreviation for FunctionFieldCategory is FFCAT
--R This constructor is exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for FFCAT
--R
--R------------------------------ Operations --------------------------------
--R ?*? : (Fraction UP,%) -> %            ?*? : (%,Fraction UP) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
--R ?+? : (%,%) -> %                      ?-? : (%,%) -> %
--R -? : % -> %                           ?=? : (%,%) -> Boolean
--R 1 : () -> %                           0 : () -> %
--R ?^? : (%,PositiveInteger) -> %        basis : () -> Vector %
--R branchPoint? : UP -> Boolean          branchPoint? : F -> Boolean
--R coerce : Fraction UP -> %             coerce : Integer -> %
--R coerce : % -> OutputForm              convert : UPUP -> %
--R convert : % -> UPUP                   convert : Vector Fraction UP -> %
```

```
--R convert : % -> Vector Fraction UP     definingPolynomial : () -> UPUP
--R discriminant : () -> Fraction UP      elt : (%,F,F) -> F
--R generator : () -> %                   genus : () -> NonNegativeInteger
--R hash : % -> SingleInteger             integral? : (%,UP) -> Boolean
--R integral? : (%,F) -> Boolean          integral? : % -> Boolean
--R integralBasis : () -> Vector %        latex : % -> String
--R lift : % -> UPUP                      norm : % -> Fraction UP
--R one? : % -> Boolean                   primitivePart : % -> %
--R ramified? : UP -> Boolean             ramified? : F -> Boolean
--R rank : () -> PositiveInteger          rationalPoint? : (F,F) -> Boolean
--R recip : % -> Union(%,"failed")        reduce : UPUP -> %
--R represents : (Vector UP,UP) -> %      retract : % -> Fraction UP
--R sample : () -> %                      singular? : UP -> Boolean
--R singular? : F -> Boolean              trace : % -> Fraction UP
--R zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
--R ?*? : (%,Fraction Integer) -> % if Fraction UP has FIELD
--R ?*? : (Fraction Integer,%) -> % if Fraction UP has FIELD
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,Integer) -> % if Fraction UP has FIELD
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (%,%) -> % if Fraction UP has FIELD
--R D : % -> % if and(has(Fraction UP,Field),has(Fraction UP,DifferentialRing)) or and(has(Fraction UP,Diffe
--R D : (%,NonNegativeInteger) -> % if and(has(Fraction UP,Field),has(Fraction UP,DifferentialRing)) or and
--R D : (%,Symbol) -> % if and(has(Fraction UP,Field),has(Fraction UP,PartialDifferentialRing Symbol)) or an
--R D : (%,List Symbol) -> % if and(has(Fraction UP,Field),has(Fraction UP,PartialDifferentialRing Symbol))
--R D : (%,Symbol,NonNegativeInteger) -> % if and(has(Fraction UP,Field),has(Fraction UP,PartialDifferential
--R D : (%,List Symbol,List NonNegativeInteger) -> % if and(has(Fraction UP,Field),has(Fraction UP,PartialDi
--R D : (%,(Fraction UP -> Fraction UP)) -> % if Fraction UP has FIELD
--R D : (%,(Fraction UP -> Fraction UP),NonNegativeInteger) -> % if Fraction UP has FIELD
--R ?^? : (%,Integer) -> % if Fraction UP has FIELD
--R ?^? : (%,NonNegativeInteger) -> %
--R absolutelyIrreducible? : () -> Boolean
--R algSplitSimple : (%,(UP -> UP)) -> Record(num: %,den: UP,derivden: UP,gd: UP)
--R associates? : (%,%) -> Boolean if Fraction UP has FIELD
--R branchPointAtInfinity? : () -> Boolean
--R characteristic : () -> NonNegativeInteger
--R characteristicPolynomial : % -> UPUP
--R charthRoot : % -> Union(%,"failed") if Fraction UP has CHARNZ
--R charthRoot : % -> % if Fraction UP has FFIELDC
--R coerce : % -> % if Fraction UP has FIELD
--R coerce : Fraction Integer -> % if Fraction UP has FIELD or Fraction UP has RETRACT FRAC INT
--R complementaryBasis : Vector % -> Vector %
--R conditionP : Matrix % -> Union(Vector %,"failed") if Fraction UP has FFIELDC
--R coordinates : Vector % -> Matrix Fraction UP
--R coordinates : % -> Vector Fraction UP
--R coordinates : (Vector %,Vector %) -> Matrix Fraction UP
--R coordinates : (%,Vector %) -> Vector Fraction UP
--R createPrimitiveElement : () -> % if Fraction UP has FFIELDC
--R derivationCoordinates : (Vector %,(Fraction UP -> Fraction UP)) -> Matrix Fraction UP if Fraction UP has
--R differentiate : % -> % if and(has(Fraction UP,Field),has(Fraction UP,DifferentialRing)) or and(has(Fract
```

```
--R differentiate : (%,NonNegativeInteger) -> % if and(has(Fraction UP,Field),has(Fraction UP,Di
--R differentiate : (%,Symbol) -> % if and(has(Fraction UP,Field),has(Fraction UP,PartialDiffere
--R differentiate : (%,List Symbol) -> % if and(has(Fraction UP,Field),has(Fraction UP,PartialDi
--R differentiate : (%,Symbol,NonNegativeInteger) -> % if and(has(Fraction UP,Field),has(Fractio
--R differentiate : (%,List Symbol,List NonNegativeInteger) -> % if and(has(Fraction UP,Field),h
--R differentiate : (%,(UP -> UP)) -> %
--R differentiate : (%,(Fraction UP -> Fraction UP)) -> % if Fraction UP has FIELD
--R differentiate : (%,(Fraction UP -> Fraction UP),NonNegativeInteger) -> % if Fraction UP has
--R discreteLog : (%,%) -> Union(NonNegativeInteger,"failed") if Fraction UP has FFIELDC
--R discreteLog : % -> NonNegativeInteger if Fraction UP has FFIELDC
--R discriminant : Vector % -> Fraction UP
--R divide : (%,%) -> Record(quotient: %,remainder: %) if Fraction UP has FIELD
--R elliptic : () -> Union(UP,"failed")
--R euclideanSize : % -> NonNegativeInteger if Fraction UP has FIELD
--R expressIdealMember : (List %,%) -> Union(List %,"failed") if Fraction UP has FIELD
--R exquo : (%,%) -> Union(%,"failed") if Fraction UP has FIELD
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %) if Fraction UP has FIELD
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed") if Fraction UP has
--R factor : % -> Factored % if Fraction UP has FIELD
--R factorsOfCyclicGroupSize : () -> List Record(factor: Integer,exponent: Integer) if Fraction
--R gcd : (%,%) -> % if Fraction UP has FIELD
--R gcd : List % -> % if Fraction UP has FIELD
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivari
--R hyperelliptic : () -> Union(UP,"failed")
--R index : PositiveInteger -> % if Fraction UP has FINITE
--R init : () -> % if Fraction UP has FFIELDC
--R integralAtInfinity? : % -> Boolean
--R integralBasisAtInfinity : () -> Vector %
--R integralCoordinates : % -> Record(num: Vector UP,den: UP)
--R integralDerivationMatrix : (UP -> UP) -> Record(num: Matrix UP,den: UP)
--R integralMatrix : () -> Matrix Fraction UP
--R integralMatrixAtInfinity : () -> Matrix Fraction UP
--R integralRepresents : (Vector UP,UP) -> %
--R inv : % -> % if Fraction UP has FIELD
--R inverseIntegralMatrix : () -> Matrix Fraction UP
--R inverseIntegralMatrixAtInfinity : () -> Matrix Fraction UP
--R lcm : (%,%) -> % if Fraction UP has FIELD
--R lcm : List % -> % if Fraction UP has FIELD
--R lookup : % -> PositiveInteger if Fraction UP has FINITE
--R minimalPolynomial : % -> UPUP if Fraction UP has FIELD
--R multiEuclidean : (List %,%) -> Union(List %,"failed") if Fraction UP has FIELD
--R nextItem : % -> Union(%,"failed") if Fraction UP has FFIELDC
--R nonSingularModel : Symbol -> List Polynomial F if F has FIELD
--R normalizeAtInfinity : Vector % -> Vector %
--R numberOfComponents : () -> NonNegativeInteger
--R order : % -> OnePointCompletion PositiveInteger if Fraction UP has FFIELDC
--R order : % -> PositiveInteger if Fraction UP has FFIELDC
--R prime? : % -> Boolean if Fraction UP has FIELD
--R primeFrobenius : % -> % if Fraction UP has FFIELDC
--R primeFrobenius : (%,NonNegativeInteger) -> % if Fraction UP has FFIELDC
```

```
--R primitive? : % -> Boolean if Fraction UP has FFIELDC
--R primitiveElement : () -> % if Fraction UP has FFIELDC
--R principalIdeal : List % -> Record(coef: List %,generator: %) if Fraction UP has FIELD
--R ?quo? : (%,%) -> % if Fraction UP has FIELD
--R ramifiedAtInfinity? : () -> Boolean
--R random : () -> % if Fraction UP has FINITE
--R rationalPoints : () -> List List F if F has FINITE
--R reduce : Fraction UPUP -> Union(%,"failed") if Fraction UP has FIELD
--R reduceBasisAtInfinity : Vector % -> Vector %
--R reducedSystem : Matrix % -> Matrix Fraction UP
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Fraction UP,vec: Vector Fraction UP)
--R reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer) if Fraction UP ha
--R reducedSystem : Matrix % -> Matrix Integer if Fraction UP has LINEXP INT
--R regularRepresentation : % -> Matrix Fraction UP
--R regularRepresentation : (%,Vector %) -> Matrix Fraction UP
--R ?rem? : (%,%) -> % if Fraction UP has FIELD
--R representationType : () -> Union("prime",polynomial,normal,cyclic) if Fraction UP has FFIELDC
--R represents : Vector Fraction UP -> %
--R represents : (Vector Fraction UP,Vector %) -> %
--R retract : % -> Fraction Integer if Fraction UP has RETRACT FRAC INT
--R retract : % -> Integer if Fraction UP has RETRACT INT
--R retractIfCan : % -> Union(Fraction UP,"failed")
--R retractIfCan : % -> Union(Fraction Integer,"failed") if Fraction UP has RETRACT FRAC INT
--R retractIfCan : % -> Union(Integer,"failed") if Fraction UP has RETRACT INT
--R singularAtInfinity? : () -> Boolean
--R size : () -> NonNegativeInteger if Fraction UP has FINITE
--R sizeLess? : (%,%) -> Boolean if Fraction UP has FIELD
--R squareFree : % -> Factored % if Fraction UP has FIELD
--R squareFreePart : % -> % if Fraction UP has FIELD
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R tableForDiscreteLogarithm : Integer -> Table(PositiveInteger,NonNegativeInteger) if Fraction UP has FFI
--R traceMatrix : () -> Matrix Fraction UP
--R traceMatrix : Vector % -> Matrix Fraction UP
--R unit? : % -> Boolean if Fraction UP has FIELD
--R unitCanonical : % -> % if Fraction UP has FIELD
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %) if Fraction UP has FIELD
--R yCoordinates : % -> Record(num: Vector UP,den: UP)
--R
--E 1

)spool
)lisp (bye)
```

———-

— FunctionFieldCategory.help —

```
=====================================================================
FunctionFieldCategory examples
```

```
=====================================================================

See Also:
o )show FunctionFieldCategory
```

See:

⇐ “MonogenicAlgebra” (MONOGEN) 19.0.214 on page 1598

Exports:

0	1
absolutelyIrreducible?	algSplitSimple
associates?	basis
branchPoint?	branchPointAtInfinity?
characteristic	characteristicPolynomial
charthRoot	coerce
complementaryBasis	conditionP
convert	coordinates
createPrimitiveElement	D
definingPolynomial	derivationCoordinates
differentiate	discreteLog
discriminant	divide
elliptic	elt
euclideanSize	expressIdealMember
exquo	extendedEuclidean
factor	factorsOfCyclicGroupSize
gcd	gcdPolynomial
generator	genus
hash	hyperelliptic
index	init
integral?	integralAtInfinity?
integralBasis	integralBasisAtInfinity
integralCoordinates	integralDerivationMatrix
integralMatrix	integralMatrixAtInfinity
integralRepresents	inv
inverseIntegralMatrix	inverseIntegralMatrixAtInfinity
latex	lcm
lift	lookup
minimalPolynomial	multiEuclidean
nextItem	nonSingularModel
norm	normalizeAtInfinity
numberOfComponents	one?
order	prime?
primeFrobenius	primitive?
primitiveElement	primitivePart
principalIdeal	ramified?
ramifiedAtInfinity?	rank
random	rationalPoints
rationalPoint?	recip
reduce	reduceBasisAtInfinity
reducedSystem	regularRepresentation
representationType	represents
retract	retractIfCan
sample	singular?
singularAtInfinity?	size
sizeLess?	squareFree
squareFreePart	subtractIfCan
tableForDiscreteLogarithm	trace
traceMatrix	unit?
unitCanonical	unitNormal
yCoordinates	zero?
?*?	?**?
?+?	?-?
-?	?=?
?^?	?~=?
?/?	?quo?

Attributes Exported:

- if $ has Fraction(UPOLYC(UFD)) and Field then noZeroDivisors where **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- if $ has Fraction(UPOLYC(UFD)) and Field then canonicalUnitNormal where **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- if $ has Fraction(UPOLYC(UFD)) and Field then canonicalsClosed where **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.
- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.

These are directly exported but not implemented:

```
branchPointAtInfinity? : () -> Boolean
branchPoint? : UP -> Boolean
branchPoint? : F -> Boolean
integralBasis : () -> Vector %
integralBasisAtInfinity : () -> Vector %
ramifiedAtInfinity? : () -> Boolean
ramified? : UP -> Boolean
ramified? : F -> Boolean
singularAtInfinity? : () -> Boolean
singular? : F -> Boolean
singular? : UP -> Boolean
```

These are implemented by this category:

```
absolutelyIrreducible? : () -> Boolean
algSplitSimple : (%,(UP -> UP)) ->
     Record(num: %,den: UP,derivden: UP,gd: UP)
complementaryBasis : Vector % -> Vector %
differentiate : (%,(UP -> UP)) -> %
elliptic : () -> Union(UP,"failed")
elt : (%,F,F) -> F
genus : () -> NonNegativeInteger
hyperelliptic : () -> Union(UP,"failed")
integral? : % -> Boolean
```

```
integral? : (%,F) -> Boolean
integral? : (%,UP) -> Boolean
integralAtInfinity? : % -> Boolean
normalizeAtInfinity : Vector % -> Vector %
numberOfComponents : () -> NonNegativeInteger
primitivePart : % -> %
rationalPoint? : (F,F) -> Boolean
rationalPoints : () -> List List F if F has FINITE
reduceBasisAtInfinity : Vector % -> Vector %
represents : (Vector UP,UP) -> %
yCoordinates : % -> Record(num: Vector UP,den: UP)
```

These exports come from (p1598) MonogenicAlgebra(RF, UPUP)
where RF:Fraction UP, UP:UnivariatePolynomialCategory F
F:UniqueFactorizationDomain, and
UPUP:UnivariatePolynomialCategory Fraction UP

```
0 : () -> %
1 : () -> %
associates? : (%,%) -> Boolean
    if Fraction UP has FIELD
basis : () -> Vector %
characteristic : () -> NonNegativeInteger
characteristicPolynomial : % -> UPUP
charthRoot : % -> Union(%,"failed")
    if Fraction UP has CHARNZ
charthRoot : % -> %
    if Fraction UP has FFIELDC
coerce : % -> %
    if Fraction UP has FIELD
coerce : Fraction Integer -> %
    if Fraction UP has FIELD
    or Fraction UP has RETRACT FRAC INT
coerce : Fraction UP -> %
coerce : Integer -> %
coerce : % -> OutputForm
conditionP : Matrix % -> Union(Vector %,"failed")
    if Fraction UP has FFIELDC
convert : UPUP -> %
convert : % -> UPUP
convert : Vector Fraction UP -> %
convert : % -> Vector Fraction UP
coordinates : Vector % -> Matrix Fraction UP
coordinates : % -> Vector Fraction UP
coordinates : (Vector %,Vector %) -> Matrix Fraction UP
coordinates : (%,Vector %) -> Vector Fraction UP
createPrimitiveElement : () -> %
    if Fraction UP has FFIELDC
D : % -> %
    if
```

```
        and(
          has(Fraction UP,Field),
          has(Fraction UP,DifferentialRing))
      or
        and(
          has(Fraction UP,DifferentialRing),
          has(Fraction UP,Field))
      or Fraction UP has FFIELDC
D : (%,NonNegativeInteger) -> %
    if
       and(
         has(Fraction UP,Field),
         has(Fraction UP,DifferentialRing))
      or
        and(
          has(Fraction UP,DifferentialRing),
          has(Fraction UP,Field))
      or Fraction UP has FFIELDC
D : (%,Symbol) -> %
    if
       and(
         has(Fraction UP,Field),
         has(Fraction UP,PartialDifferentialRing Symbol))
      or
        and(
          has(Fraction UP,PartialDifferentialRing Symbol),
          has(Fraction UP,Field))
D : (%,List Symbol) -> %
    if
       and(
         has(Fraction UP,Field),
         has(Fraction UP,PartialDifferentialRing Symbol))
      or
        and(
          has(Fraction UP,PartialDifferentialRing Symbol),
          has(Fraction UP,Field))
D : (%,Symbol,NonNegativeInteger) -> %
    if
       and(
         has(Fraction UP,Field),
         has(Fraction UP,PartialDifferentialRing Symbol))
      or
        and(
          has(Fraction UP,PartialDifferentialRing Symbol),
          has(Fraction UP,Field))
D : (%,List Symbol,List NonNegativeInteger) -> %
    if
       and(
         has(Fraction UP,Field),
         has(Fraction UP,PartialDifferentialRing Symbol))
```

```
     or
       and(
         has(Fraction UP,PartialDifferentialRing Symbol),
         has(Fraction UP,Field))
D : (%,(Fraction UP -> Fraction UP)) -> %
    if Fraction UP has FIELD
D : (%,(Fraction UP -> Fraction UP),NonNegativeInteger) -> %
    if Fraction UP has FIELD
definingPolynomial : () -> UPUP
derivationCoordinates : (Vector %,(Fraction UP -> Fraction UP))
  -> Matrix Fraction UP
    if Fraction UP has FIELD
differentiate : % -> %
    if
       and(
         has(Fraction UP,Field),
         has(Fraction UP,DifferentialRing))
     or
       and(
         has(Fraction UP,DifferentialRing),
         has(Fraction UP,Field))
     or Fraction UP has FFIELDC
differentiate : (%,NonNegativeInteger) -> %
    if
       and(
         has(Fraction UP,Field),
         has(Fraction UP,DifferentialRing))
     or
       and(
         has(Fraction UP,DifferentialRing),
         has(Fraction UP,Field))
     or Fraction UP has FFIELDC
differentiate : (%,Symbol) -> %
    if
      and(
        has(Fraction UP,Field),
        has(Fraction UP,PartialDifferentialRing Symbol))
    or
     and(
       has(Fraction UP,PartialDifferentialRing Symbol),
       has(Fraction UP,Field))
differentiate : (%,List Symbol) -> %
    if
      and(
        has(Fraction UP,Field),
        has(Fraction UP,PartialDifferentialRing Symbol))
     or
      and(
        has(Fraction UP,PartialDifferentialRing Symbol),
        has(Fraction UP,Field))
```

```
differentiate : (%,Symbol,NonNegativeInteger) -> %
    if
       and(
         has(Fraction UP,Field),
         has(Fraction UP,PartialDifferentialRing Symbol))
     or
       and(
         has(Fraction UP,PartialDifferentialRing Symbol),
         has(Fraction UP,Field))
differentiate : (%,List Symbol,List NonNegativeInteger) -> %
    if
       and(
         has(Fraction UP,Field),
         has(Fraction UP,PartialDifferentialRing Symbol))
      or
       and(
         has(Fraction UP,PartialDifferentialRing Symbol),
         has(Fraction UP,Field))
differentiate : (%,(Fraction UP -> Fraction UP)) -> %
    if Fraction UP has FIELD
differentiate :
  (%,(Fraction UP -> Fraction UP),NonNegativeInteger) -> %
    if Fraction UP has FIELD
discreteLog : (%,%) -> Union(NonNegativeInteger,"failed")
    if Fraction UP has FFIELDC
discreteLog : % -> NonNegativeInteger
    if Fraction UP has FFIELDC
discriminant : Vector % -> Fraction UP
discriminant : () -> Fraction UP
divide : (%,%) -> Record(quotient: %,remainder: %)
    if Fraction UP has FIELD
euclideanSize : % -> NonNegativeInteger
    if Fraction UP has FIELD
expressIdealMember : (List %,%) -> Union(List %,"failed")
    if Fraction UP has FIELD
exquo : (%,%) -> Union(%,"failed")
    if Fraction UP has FIELD
extendedEuclidean : (%,%) ->
   Record(coef1: %,coef2: %,generator: %)
    if Fraction UP has FIELD
extendedEuclidean : (%,%,%) ->
   Union(Record(coef1: %,coef2: %),"failed")
    if Fraction UP has FIELD
factor : % -> Factored %
    if Fraction UP has FIELD
factorsOfCyclicGroupSize : () ->
   List Record(factor: Integer,exponent: Integer)
    if Fraction UP has FFIELDC
gcd : (%,%) -> %
    if Fraction UP has FIELD
```

```
gcd : List % -> %
    if Fraction UP has FIELD
gcdPolynomial : (SparseUnivariatePolynomial %,
                 SparseUnivariatePolynomial %) ->
                    SparseUnivariatePolynomial %
    if Fraction UP has FIELD
generator : () -> %
hash : % -> SingleInteger
index : PositiveInteger -> %
    if Fraction UP has FINITE
init : () -> %
    if Fraction UP has FFIELDC
integralCoordinates : % ->
   Record(num: Vector UP,den: UP)
integralDerivationMatrix : (UP -> UP) ->
   Record(num: Matrix UP,den: UP)
integralMatrix : () -> Matrix Fraction UP
integralMatrixAtInfinity : () -> Matrix Fraction UP
integralRepresents : (Vector UP,UP) -> %
inv : % -> %
    if Fraction UP has FIELD
inverseIntegralMatrix : () -> Matrix Fraction UP
inverseIntegralMatrixAtInfinity : () ->
   Matrix Fraction UP
latex : % -> String
lcm : (%,%) -> %
    if Fraction UP has FIELD
lcm : List % -> %
    if Fraction UP has FIELD
lift : % -> UPUP
lookup : % -> PositiveInteger
    if Fraction UP has FINITE
minimalPolynomial : % -> UPUP
    if Fraction UP has FIELD
multiEuclidean : (List %,%) -> Union(List %,"failed")
    if Fraction UP has FIELD
nextItem : % -> Union(%,"failed")
    if Fraction UP has FFIELDC
nonSingularModel : Symbol -> List Polynomial F
    if F has FIELD
norm : % -> Fraction UP
one? : % -> Boolean
order : % -> OnePointCompletion PositiveInteger
    if Fraction UP has FFIELDC
order : % -> PositiveInteger
    if Fraction UP has FFIELDC
prime? : % -> Boolean
    if Fraction UP has FIELD
primeFrobenius : % -> %
    if Fraction UP has FFIELDC
```

```
primeFrobenius : (%,NonNegativeInteger) -> %
    if Fraction UP has FFIELDC
primitive? : % -> Boolean
    if Fraction UP has FFIELDC
primitiveElement : () -> %
    if Fraction UP has FFIELDC
principalIdeal : List % ->
  Record(coef: List %,generator: %)
    if Fraction UP has FIELD
rank : () -> PositiveInteger
random : () -> %
    if Fraction UP has FINITE
recip : % -> Union(%,"failed")
reduce : UPUP -> %
reduce : Fraction UPUP -> Union(%,"failed")
    if Fraction UP has FIELD
reducedSystem : Matrix % -> Matrix Fraction UP
reducedSystem : (Matrix %,Vector %) ->
  Record(mat: Matrix Fraction UP,vec: Vector Fraction UP)
reducedSystem : (Matrix %,Vector %) ->
  Record(mat: Matrix Integer,vec: Vector Integer)
    if Fraction UP has LINEXP INT
reducedSystem : Matrix % -> Matrix Integer
    if Fraction UP has LINEXP INT
regularRepresentation : % -> Matrix Fraction UP
regularRepresentation : (%,Vector %) -> Matrix Fraction UP
representationType : () ->
  Union("prime",polynomial,normal,cyclic)
    if Fraction UP has FFIELDC
represents : Vector Fraction UP -> %
represents : (Vector Fraction UP,Vector %) -> %
retract : % -> Fraction Integer
    if Fraction UP has RETRACT FRAC INT
retract : % -> Integer
    if Fraction UP has RETRACT INT
retract : % -> Fraction UP
retractIfCan : % -> Union(Fraction UP,"failed")
retractIfCan : % -> Union(Fraction Integer,"failed")
    if Fraction UP has RETRACT FRAC INT
retractIfCan : % -> Union(Integer,"failed")
    if Fraction UP has RETRACT INT
sample : () -> %
size : () -> NonNegativeInteger
    if Fraction UP has FINITE
sizeLess? : (%,%) -> Boolean
    if Fraction UP has FIELD
squareFree : % -> Factored %
    if Fraction UP has FIELD
squareFreePart : % -> %
    if Fraction UP has FIELD
```

```
subtractIfCan : (%,%) -> Union(%,"failed")
tableForDiscreteLogarithm : Integer ->
  Table(PositiveInteger,NonNegativeInteger)
    if Fraction UP has FFIELDC
trace : % -> Fraction UP
traceMatrix : () -> Matrix Fraction UP
traceMatrix : Vector % -> Matrix Fraction UP
unit? : % -> Boolean
    if Fraction UP has FIELD
unitCanonical : % -> %
    if Fraction UP has FIELD
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
    if Fraction UP has FIELD
zero? : % -> Boolean
?*? : (Fraction UP,%) -> %
?*? : (%,Fraction UP) -> %
?*? : (%,%) -> %
?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %
?**? : (%,PositiveInteger) -> %
?+? : (%,%) -> %
?-? : (%,%) -> %
-? : % -> %
?=? : (%,%) -> Boolean
?^? : (%,PositiveInteger) -> %
?~=? : (%,%) -> Boolean
?*? : (%,Fraction Integer) -> % if Fraction UP has FIELD
?*? : (Fraction Integer,%) -> % if Fraction UP has FIELD
?*? : (NonNegativeInteger,%) -> %
?**? : (%,Integer) -> % if Fraction UP has FIELD
?**? : (%,NonNegativeInteger) -> %
?/? : (%,%) -> % if Fraction UP has FIELD
?^? : (%,Integer) -> % if Fraction UP has FIELD
?^? : (%,NonNegativeInteger) -> %
?quo? : (%,%) -> % if Fraction UP has FIELD
?rem? : (%,%) -> % if Fraction UP has FIELD
```

— category FFCAT FunctionFieldCategory —

```
)abbrev category FFCAT FunctionFieldCategory
++ Function field of a curve
++ Author: Manuel Bronstein
++ Date Created: 1987
++ Date Last Updated: 19 Mai 1993
++ Keywords: algebraic, curve, function, field.
++ Description:
++ This category is a model for the function field of a
++ plane algebraic curve.
```

```
FunctionFieldCategory(F, UP, UPUP): Category == Definition where
  F   : UniqueFactorizationDomain
  UP  : UnivariatePolynomialCategory F
  UPUP: UnivariatePolynomialCategory Fraction UP

  Z   ==> Integer
  Q   ==> Fraction F
  P   ==> Polynomial F
  RF  ==> Fraction UP
  QF  ==> Fraction UPUP
  SY  ==> Symbol
  REC ==> Record(num:$, den:UP, derivden:UP, gd:UP)

  Definition ==> MonogenicAlgebra(RF, UPUP) with
    numberOfComponents     : () -> NonNegativeInteger
      ++ numberOfComponents() returns the number of absolutely irreducible
      ++ components.
      ++
      ++X P0 := UnivariatePolynomial(x, Integer)
      ++X P1 := UnivariatePolynomial(y, Fraction P0)
      ++X R := RadicalFunctionField(INT, P0, P1, 1 - x**20, 20)
      ++X numberOfComponents()$R
    genus                  : () -> NonNegativeInteger
      ++ genus() returns the genus of one absolutely irreducible component
      ++
      ++X P0 := UnivariatePolynomial(x, Integer)
      ++X P1 := UnivariatePolynomial(y, Fraction P0)
      ++X R := RadicalFunctionField(INT, P0, P1, 1 - x**20, 20)
      ++X genus()$R
    absolutelyIrreducible? : () -> Boolean
      ++ absolutelyIrreducible?() tests if the curve absolutely irreducible?
      ++
      ++X P0 := UnivariatePolynomial(x, Integer)
      ++X P1 := UnivariatePolynomial(y, Fraction P0)
      ++X R2 := RadicalFunctionField(INT, P0, P1, 2 * x**2, 4)
      ++X absolutelyIrreducible?()$R2
    rationalPoint?         : (F, F) -> Boolean
      ++ rationalPoint?(a, b) tests if \spad{(x=a,y=b)} is on the curve.
      ++
      ++X P0 := UnivariatePolynomial(x, Integer)
      ++X P1 := UnivariatePolynomial(y, Fraction P0)
      ++X R := RadicalFunctionField(INT, P0, P1, 1 - x**20, 20)
      ++X rationalPoint?(0,0)$R
      ++X R2 := RadicalFunctionField(INT, P0, P1, 2 * x**2, 4)
      ++X rationalPoint?(0,0)$R2
    branchPointAtInfinity? : () -> Boolean
      ++ branchPointAtInfinity?() tests if there is a branch point
      ++ at infinity.
      ++
      ++X P0 := UnivariatePolynomial(x, Integer)
```

```
  ++X P1 := UnivariatePolynomial(y, Fraction P0)
  ++X R := RadicalFunctionField(INT, P0, P1, 1 - x**20, 20)
  ++X branchPointAtInfinity?()$R
  ++X R2 := RadicalFunctionField(INT, P0, P1, 2 * x**2, 4)
  ++X branchPointAtInfinity?()$R
branchPoint?           : F -> Boolean
  ++ branchPoint?(a) tests whether \spad{x = a} is a branch point.
branchPoint?           : UP -> Boolean
  ++ branchPoint?(p) tests whether \spad{p(x) = 0} is a branch point.
singularAtInfinity?    : () -> Boolean
  ++ singularAtInfinity?() tests if there is a singularity at infinity.
singular?              : F -> Boolean
  ++ singular?(a) tests whether \spad{x = a} is singular.
singular?              : UP -> Boolean
  ++ singular?(p) tests whether \spad{p(x) = 0} is singular.
ramifiedAtInfinity?    : () -> Boolean
  ++ ramifiedAtInfinity?() tests if infinity is ramified.
ramified?              : F -> Boolean
  ++ ramified?(a) tests whether \spad{x = a} is ramified.
ramified?              : UP -> Boolean
  ++ ramified?(p) tests whether \spad{p(x) = 0} is ramified.
integralBasis          : () -> Vector $
  ++ integralBasis() returns the integral basis for the curve.
  ++
  ++X P0 := UnivariatePolynomial(x, Integer)
  ++X P1 := UnivariatePolynomial(y, Fraction P0)
  ++X R := RadicalFunctionField(INT, P0, P1, 1 - x**20, 20)
  ++X integralBasis()$R
integralBasisAtInfinity: () -> Vector $
  ++ integralBasisAtInfinity() returns the local integral basis
  ++ at infinity
  ++
  ++X P0 := UnivariatePolynomial(x, Integer)
  ++X P1 := UnivariatePolynomial(y, Fraction P0)
  ++X R := RadicalFunctionField(INT, P0, P1, 1 - x**20, 20)
  ++X integralBasisAtInfinity()$R
integralAtInfinity?    : $  -> Boolean
  ++ integralAtInfinity?() tests if f is locally integral at infinity.
integral?              : $  -> Boolean
  ++ integral?() tests if f is integral over \spad{k[x]}.
complementaryBasis     : Vector $ -> Vector $
  ++ complementaryBasis(b1,...,bn) returns the complementary basis
  ++ \spad{(b1',...,bn')} of \spad{(b1,...,bn)}.
normalizeAtInfinity    : Vector $ -> Vector $
  ++ normalizeAtInfinity(v) makes v normal at infinity.
reduceBasisAtInfinity  : Vector $ -> Vector $
  ++ reduceBasisAtInfinity(b1,...,bn) returns \spad{(x**i * bj)}
  ++ for all i,j such that \spad{x**i*bj} is locally integral
  ++ at infinity.
integralMatrix         : () -> Matrix RF
```

```
  ++ integralMatrix() returns M such that
  ++ \spad{(w1,...,wn) = M (1, y, ..., y**(n-1))},
  ++ where \spad{(w1,...,wn)} is the integral basis of
  ++ \spadfunFrom{integralBasis}{FunctionFieldCategory}.
  ++
  ++X P0 := UnivariatePolynomial(x, Integer)
  ++X P1 := UnivariatePolynomial(y, Fraction P0)
  ++X R := RadicalFunctionField(INT, P0, P1, 1 - x**20, 20)
  ++X integralMatrix()$R
inverseIntegralMatrix  : () -> Matrix RF
  ++ inverseIntegralMatrix() returns M such that
  ++ \spad{M (w1,...,wn) = (1, y, ..., y**(n-1))}
  ++ where \spad{(w1,...,wn)} is the integral basis of
  ++ \spadfunFrom{integralBasis}{FunctionFieldCategory}.
  ++
  ++X P0 := UnivariatePolynomial(x, Integer)
  ++X P1 := UnivariatePolynomial(y, Fraction P0)
  ++X R := RadicalFunctionField(INT, P0, P1, 1 - x**20, 20)
  ++X inverseIntegralMatrix()$R
integralMatrixAtInfinity       : () -> Matrix RF
  ++ integralMatrixAtInfinity() returns M such that
  ++ \spad{(v1,...,vn) = M (1, y, ..., y**(n-1))}
  ++ where \spad{(v1,...,vn)} is the local integral basis at infinity
  ++ returned by \spad{infIntBasis()}.
  ++
  ++X P0 := UnivariatePolynomial(x, Integer)
  ++X P1 := UnivariatePolynomial(y, Fraction P0)
  ++X R := RadicalFunctionField(INT, P0, P1, 1 - x**20, 20)
  ++X integralMatrixAtInfinity()$R
inverseIntegralMatrixAtInfinity: () -> Matrix RF
  ++ inverseIntegralMatrixAtInfinity() returns M such
  ++ that \spad{M (v1,...,vn) = (1, y, ..., y**(n-1))}
  ++ where \spad{(v1,...,vn)} is the local integral basis at infinity
  ++ returned by \spad{infIntBasis()}.
  ++
  ++X P0 := UnivariatePolynomial(x, Integer)
  ++X P1 := UnivariatePolynomial(y, Fraction P0)
  ++X R := RadicalFunctionField(INT, P0, P1, 1 - x**20, 20)
  ++X inverseIntegralMatrixAtInfinity()$R
yCoordinates           : $ -> Record(num:Vector(UP), den:UP)
  ++ yCoordinates(f) returns \spad{[[A1,...,An], D]} such that
  ++ \spad{f = (A1 + A2 y +...+ An y**(n-1)) / D}.
represents             : (Vector UP, UP) -> $
  ++ represents([A0,...,A(n-1)],D) returns
  ++ \spad{(A0 + A1 y +...+ A(n-1)*y**(n-1))/D}.
integralCoordinates    : $ -> Record(num:Vector(UP), den:UP)
  ++ integralCoordinates(f) returns \spad{[[A1,...,An], D]} such that
  ++ \spad{f = (A1 w1 +...+ An wn) / D}  where \spad{(w1,...,wn)} is the
  ++ integral basis returned by \spad{integralBasis()}.
integralRepresents     : (Vector UP, UP) -> $
```

```
  ++ integralRepresents([A1,...,An], D) returns
  ++ \spad{(A1 w1+...+An wn)/D}
  ++ where \spad{(w1,...,wn)} is the integral
  ++ basis of \spad{integralBasis()}.
integralDerivationMatrix:(UP -> UP) -> Record(num:Matrix(UP),den:UP)
  ++ integralDerivationMatrix(d) extends the derivation d from UP to $
  ++ and returns (M, Q) such that the i^th row of M divided by Q form
  ++ the coordinates of \spad{d(wi)} with respect to \spad{(w1,...,wn)}
  ++ where \spad{(w1,...,wn)} is the integral basis returned
  ++ by integralBasis().
integral?              : ($,  F) -> Boolean
  ++ integral?(f, a) tests whether f is locally integral at \spad{x = a}.
integral?              : ($, UP) -> Boolean
  ++ integral?(f, p) tests whether f is locally integral at
  ++ \spad{p(x) = 0}
differentiate          : ($, UP -> UP) -> $
  ++ differentiate(x, d) extends the derivation d from UP to $ and
  ++ applies it to x.
represents             : (Vector UP, UP) -> $
  ++ represents([A0,...,A(n-1)],D) returns
  ++ \spad{(A0 + A1 y +...+ A(n-1)*y**(n-1))/D}.
primitivePart          : $ -> $
  ++ primitivePart(f) removes the content of the denominator and
  ++ the common content of the numerator of f.
elt                    : ($, F, F) -> F
  ++ elt(f,a,b) or f(a, b) returns the value of f
  ++ at the point \spad{(x = a, y = b)}
  ++ if it is not singular.
elliptic               : () -> Union(UP, "failed")
  ++ elliptic() returns \spad{p(x)} if the curve is the elliptic
  ++ defined by \spad{y**2 = p(x)}, "failed" otherwise.
hyperelliptic          : () -> Union(UP, "failed")
  ++ hyperelliptic() returns \spad{p(x)} if the curve is the
  ++ hyperelliptic
  ++ defined by \spad{y**2 = p(x)}, "failed" otherwise.
algSplitSimple         : ($, UP -> UP) -> REC
  ++ algSplitSimple(f, D) returns \spad{[h,d,d',g]} such that
  ++ \spad{f=h/d},
  ++ \spad{h} is integral at all the normal places w.r.t. \spad{D},
  ++ \spad{d' = Dd}, \spad{g = gcd(d, discriminant())} and \spad{D}
  ++ is the derivation to use. \spad{f} must have at most simple finite
  ++ poles.
if F has Field then
  nonSingularModel: SY -> List Polynomial F
    ++ nonSingularModel(u) returns the equations in u1,...,un of
    ++ an affine non-singular model for the curve.
if F has Finite then
  rationalPoints: () -> List List F
    ++ rationalPoints() returns the list of all the affine
    ++rational points.
```

```
   add
    import InnerCommonDenominator(UP, RF, Vector UP, Vector RF)
    import UnivariatePolynomialCommonDenominator(UP, RF, UPUP)

    repOrder: (Matrix RF, Z) -> Z
    Q2RF    : Q  -> RF
    infOrder: RF -> Z
    infValue: RF -> Fraction F
    intvalue: (Vector UP, F, F) -> F
    rfmonom : Z  -> RF
    kmin    : (Matrix RF,Vector Q) -> Union(Record(pos:Z,km:Z),"failed")

    Q2RF q                == numer(q)::UP / denom(q)::UP
    infOrder f            == (degree denom f)::Z - (degree numer f)::Z
    integral? f           == ground?(integralCoordinates(f).den)
    integral?(f:$, a:F)   == (integralCoordinates(f).den)(a) ^= 0
--    absolutelyIrreducible? == one? numberOfComponents()
    absolutelyIrreducible? == numberOfComponents() = 1
    yCoordinates f        == splitDenominator coordinates f

    hyperelliptic() ==
      degree(f := definingPolynomial()) ^= 2 => "failed"
      (u:=retractIfCan(reductum f)@Union(RF,"failed"))
        case "failed" => "failed"
      (v:=retractIfCan(-(u::RF) / leadingCoefficient f)@Union(UP, "failed"))
        case "failed" => "failed"
      odd? degree(p := v::UP) => p
      "failed"

    algSplitSimple(f, derivation) ==
      cd := splitDenominator lift f
      dd := (cd.den exquo (g := gcd(cd.den, derivation(cd.den))))::UP
      [reduce(inv(g::RF) * cd.num), dd, derivation dd,
                                  gcd(dd, retract(discriminant())@UP)]

    elliptic() ==
      (u := hyperelliptic()) case "failed" => "failed"
      degree(p := u::UP) = 3 => p
      "failed"

    rationalPoint?(x, y)   ==
      zero?((definingPolynomial() (y::UP::RF)) (x::UP::RF))

    if F has Field then
      import PolyGroebner(F)
      import MatrixCommonDenominator(UP, RF)

      UP2P  : (UP,   P)    -> P
      UPUP2P: (UPUP, P, P) -> P
```

```
      UP2P(p, x) ==
        (map((s:F):P +-> s::P, p)_
          $UnivariatePolynomialCategoryFunctions2(F, UP,
                                  P, SparseUnivariatePolynomial P)) x

      UPUP2P(p, x, y) ==
        (map((s:RF):P +-> UP2P(retract(s)@UP, x),p)_
          $UnivariatePolynomialCategoryFunctions2(RF, UPUP,
                                  P, SparseUnivariatePolynomial P)) y

      nonSingularModel u ==
        d    := commonDenominator(coordinates(w := integralBasis()))::RF
        vars := [concat(string u, string i)::SY for i in 1..(n := #w)]
        x    := "%%dummy1"::SY
        y    := "%%dummy2"::SY
        select_!(s+->zero?(degree(s, x)) and zero?(degree(s, y)),
                 lexGroebner([v::P - UPUP2P(lift(d * w.i), x::P, y::P)
                    for v in vars for i in 1..n], concat([x, y], vars)))

    if F has Finite then
      ispoint: (UPUP, F, F) -> List F

-- must use the 'elt function explicitely or the compiler takes 45 mins
-- on that function    MB 5/90
-- still takes ages : I split the expression up. JHD 6/Aug/90
      ispoint(p, x, y) ==
        jhd:RF:=p(y::UP::RF)
        zero?(jhd (x::UP::RF)) => [x, y]
        empty()

      rationalPoints() ==
        p := definingPolynomial()
        concat [[pt for y in 1..size()$F | not empty?(pt :=
          ispoint(p, index(x::PositiveInteger)$F,
                     index(y::PositiveInteger)$F))]$List(List F)
                               for x in 1..size()$F]$List(List(List F))

    intvalue(v, x, y) ==
      singular? x => error "Point is singular"
      mini := minIndex(w := integralBasis())
      rec := yCoordinates(+/[qelt(v, i)::RF * qelt(w, i)
                          for i in mini .. maxIndex w])
      n   := +/[(qelt(rec.num, i) x) *
                (y ** ((i - mini)::NonNegativeInteger))
                          for i in mini .. maxIndex w]
      zero?(d := (rec.den) x) =>
        zero? n => error "0/0 -- cannot compute value yet"
        error "Shouldn't happen"
      (n exquo d)::F
```

```
elt(f, x, y) ==
  rec := integralCoordinates f
  n   := intvalue(rec.num, x, y)
  zero?(d := (rec.den) x) =>
    zero? n => error "0/0 -- cannot compute value yet"
    error "Function has a pole at the given point"
  (n exquo d)::F

primitivePart f ==
  cd := yCoordinates f
  d  := gcd([content qelt(cd.num, i)
             for i in minIndex(cd.num) .. maxIndex(cd.num)]$List(F))
               * primitivePart(cd.den)
  represents [qelt(cd.num, i) / d
           for i in minIndex(cd.num) .. maxIndex(cd.num)]$Vector(RF)

reduceBasisAtInfinity b ==
  x := monomial(1, 1)$UP ::RF
  concat([[f for j in 0.. while
            integralAtInfinity?(f := x**j * qelt(b, i))]$Vector($)
                 for i in minIndex b .. maxIndex b]$List(Vector $))

complementaryBasis b ==
  m := inverse(traceMatrix b)::Matrix(RF)
  [represents row(m, i) for i in minRowIndex m .. maxRowIndex m]

integralAtInfinity? f ==
  not any?(s +-> infOrder(s) < 0,
     coordinates(f) * inverseIntegralMatrixAtInfinity())$Vector(RF)

numberOfComponents() ==
  count(integralAtInfinity?, integralBasis())$Vector($)

represents(v:Vector UP, d:UP) ==
  represents
    [qelt(v, i) / d for i in minIndex v .. maxIndex v]$Vector(RF)

genus() ==
  ds := discriminant()
  d  := degree(retract(ds)@UP) + infOrder(ds * determinant(
        integralMatrixAtInfinity() * inverseIntegralMatrix()) ** 2)
  dd := (((d exquo 2)::Z - rank()) exquo numberOfComponents())::Z
  (dd + 1)::NonNegativeInteger

repOrder(m, i) ==
  nostart:Boolean := true
  ans:Z := 0
  r := row(m, i)
  for j in minIndex r .. maxIndex r | qelt(r, j) ^= 0 repeat
    ans :=
```

```
      nostart => (nostart := false; infOrder qelt(r, j))
      min(ans, infOrder qelt(r,j))
  nostart => error "Null row"
  ans

infValue f ==
  zero? f => 0
  (n := infOrder f) > 0 => 0
  zero? n =>
    (leadingCoefficient numer f) / (leadingCoefficient denom f)
  error "f not locally integral at infinity"

rfmonom n ==
  n < 0 => inv(monomial(1, (-n)::NonNegativeInteger)$UP :: RF)
  monomial(1, n::NonNegativeInteger)$UP :: RF

kmin(m, v) ==
  nostart:Boolean := true
  k:Z := 0
  ii  := minRowIndex m - (i0  := minIndex v)
  for i in minIndex v .. maxIndex v | qelt(v, i) ^= 0 repeat
    nk := repOrder(m, i + ii)
    if nostart then (nostart := false; k := nk; i0 := i)
    else
      if nk < k then (k := nk; i0 := i)
  nostart => "failed"
  [i0, k]

normalizeAtInfinity w ==
  ans   := copy w
  infm  := inverseIntegralMatrixAtInfinity()
  mhat  := zero(rank(), rank())$Matrix(RF)
  ii    := minIndex w - minRowIndex mhat
  repeat
    m := coordinates(ans) * infm
    r := [rfmonom repOrder(m, i)
                for i in minRowIndex m .. maxRowIndex m]$Vector(RF)
    for i in minRowIndex m .. maxRowIndex m repeat
      for j in minColIndex m .. maxColIndex m repeat
        qsetelt_!(mhat, i, j, qelt(r, i + ii) * qelt(m, i, j))
    sol := first nullSpace transpose map(infValue,
           mhat)$MatrixCategoryFunctions2(RF, Vector RF, Vector RF,
                        Matrix RF, Q, Vector Q, Vector Q, Matrix Q)
    (pr := kmin(m, sol)) case "failed" => return ans
    qsetelt_!(ans, pr.pos,
     +/[Q2RF(qelt(sol, i)) * rfmonom(repOrder(m, i - ii) - pr.km)
              * qelt(ans, i) for i in minIndex sol .. maxIndex sol])

integral?(f:$, p:UP) ==
  (r:=retractIfCan(p)@Union(F,"failed")) case F => integral?(f,r::F)
```

```
      (integralCoordinates(f).den exquo p) case "failed"

    differentiate(f:$, d:UP -> UP) ==
      differentiate(f, x +-> differentiate(x, d)$RF)
```

———-

— **FFCAT.dotabb** —

```
"FFCAT"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FFCAT"];
"FFCAT" -> "MONOGEN"
```

———-

— **FFCAT.dotfull** —

```
"FunctionFieldCategory(a:UFD,b:UPOLYC(a),c:UPOLYC(Fraction(b)))"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=FFCAT"];
"FunctionFieldCategory(a:UFD,b:UPOLYC(a),c:UPOLYC(Fraction(b)))"
   -> "MonogenicAlgebra(a:FRAC(UPOLYC(UFD)),b:UPOLYC(FRAC(UPOLYC(UFD))))"
```

———-

— **FFCAT.dotpic** —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"FunctionFieldCategory(a:UFD,b:UPOLYC(a),c:UPOLYC(Fraction(b)))"
 [color=lightblue];
"FunctionFieldCategory(a:UFD,b:UPOLYC(a),c:UPOLYC(Fraction(b)))"
   -> "MonogenicAlgebra(a:FRAC(UPOLYC(UFD)),b:UPOLYC(FRAC(UPOLYC(UFD))))"

"MonogenicAlgebra(a:FRAC(UPOLYC(UFD)),b:UPOLYC(FRAC(UPOLYC(UFD))))"
 [color=seagreen];
"MonogenicAlgebra(a:FRAC(UPOLYC(UFD)),b:UPOLYC(FRAC(UPOLYC(UFD))))" ->
    "MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))"

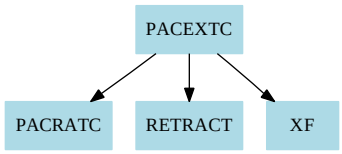
"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))"
 [color=lightblue];
"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "FRAMALG..."
```

```
"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "COMRING..."
"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "KONVERT..."
"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "FRETRCT..."
"MonogenicAlgebra(a:CommutativeRing,b:UnivariatePolynomialCategory(a))" ->
    "FLINEXP..."

"FRAMALG..." [color=lightblue];
"COMRING..." [color=lightblue];
"KONVERT..." [color=lightblue];
"FRETRCT..." [color=lightblue];
"FLINEXP..." [color=lightblue];
}
```

———-

20.0.218 PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory (PACEXTC)



— **PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory.input** —

```
)set break resume
)sys rm -f PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory.output
)spool PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory.output
)set message test on
)set message auto off
)clear all

--S 1 of 1
)show PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory
--R PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory  is a category constructor
--R Abbreviation for PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory is PACEXTC
--R This constructor is not exposed in this frame.
--R Issue )edit bookvol10.2.pamphlet to see algebra source code for PACEXTC
--R
```

```
--R------------------------------ Operations ------------------------------
--R ?*? : (%,Fraction Integer) -> %       ?*? : (Fraction Integer,%) -> %
--R ?*? : (Fraction Integer,%) -> %       ?*? : (%,Fraction Integer) -> %
--R ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
--R ?*? : (PositiveInteger,%) -> %        ?**? : (%,Integer) -> %
--R ?**? : (%,PositiveInteger) -> %       ?+? : (%,%) -> %
--R ?-? : (%,%) -> %                      -? : % -> %
--R ?/? : (%,Fraction Integer) -> %       ?/? : (%,%) -> %
--R ?=? : (%,%) -> Boolean                1 : () -> %
--R 0 : () -> %                           ?^? : (%,Integer) -> %
--R ?^? : (%,PositiveInteger) -> %        algebraic? : % -> Boolean
--R associates? : (%,%) -> Boolean        coerce : Fraction Integer -> %
--R coerce : Fraction Integer -> %        coerce : Integer -> %
--R coerce : Fraction Integer -> %        coerce : % -> %
--R coerce : Integer -> %                 coerce : % -> OutputForm
--R conjugate : % -> %                    dimension : () -> CardinalNumber
--R extDegree : % -> PositiveInteger      factor : % -> Factored %
--R fullOutput : % -> OutputForm          gcd : List % -> %
--R gcd : (%,%) -> %                      ground? : % -> Boolean
--R hash : % -> SingleInteger             inGroundField? : % -> Boolean
--R inv : % -> %                          latex : % -> String
--R lcm : List % -> %                     lcm : (%,%) -> %
--R maxTower : List % -> %                one? : % -> Boolean
--R previousTower : % -> %                prime? : % -> Boolean
--R ?quo? : (%,%) -> %                    recip : % -> Union(%,"failed")
--R ?rem? : (%,%) -> %                    retract : % -> Fraction Integer
--R retract : % -> Fraction Integer       retract : % -> Integer
--R sample : () -> %                      setTower! : % -> Void
--R sizeLess? : (%,%) -> Boolean          squareFree : % -> Factored %
--R squareFreePart : % -> %               transcendent? : % -> Boolean
--R unit? : % -> Boolean                  unitCanonical : % -> %
--R vectorise : (%,%) -> Vector %         zero? : % -> Boolean
--R ?~=? : (%,%) -> Boolean
--R ?*? : (%,PseudoAlgebraicClosureOfRationalNumber) -> %
--R ?*? : (PseudoAlgebraicClosureOfRationalNumber,%) -> %
--R ?*? : (NonNegativeInteger,%) -> %
--R ?**? : (%,NonNegativeInteger) -> %
--R ?/? : (%,PseudoAlgebraicClosureOfRationalNumber) -> %
--R Frobenius : % -> % if PseudoAlgebraicClosureOfRationalNumber has FINITE or Fraction Integer
--R Frobenius : (%,NonNegativeInteger) -> % if PseudoAlgebraicClosureOfRationalNumber has FINITE
--R ?^? : (%,NonNegativeInteger) -> %
--R characteristic : () -> NonNegativeInteger
--R charthRoot : % -> Union(%,"failed") if PseudoAlgebraicClosureOfRationalNumber has CHARNZ or
--R coerce : PseudoAlgebraicClosureOfRationalNumber -> %
--R coerce : PseudoAlgebraicClosureOfRationalNumber -> %
--R definingPolynomial : % -> SparseUnivariatePolynomial %
--R definingPolynomial : () -> SparseUnivariatePolynomial %
--R degree : % -> OnePointCompletion PositiveInteger
--R discreteLog : (%,%) -> Union(NonNegativeInteger,"failed") if PseudoAlgebraicClosureOfRationa
--R distinguishedRootsOf : (SparseUnivariatePolynomial %,%) -> List %
```

```
--R divide : (%,%) -> Record(quotient: %,remainder: %)
--R euclideanSize : % -> NonNegativeInteger
--R expressIdealMember : (List %,%) -> Union(List %,"failed")
--R exquo : (%,%) -> Union(%,"failed")
--R extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
--R extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
--R extensionDegree : () -> OnePointCompletion PositiveInteger
--R gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUnivariatePolynomia
--R lift : (%,%) -> SparseUnivariatePolynomial %
--R lift : % -> SparseUnivariatePolynomial %
--R multiEuclidean : (List %,%) -> Union(List %,"failed")
--R newElement : (SparseUnivariatePolynomial %,Symbol) -> %
--R newElement : (SparseUnivariatePolynomial %,%,Symbol) -> %
--R order : % -> OnePointCompletion PositiveInteger if PseudoAlgebraicClosureOfRationalNumber has CHARNZ or
--R primeFrobenius : % -> % if PseudoAlgebraicClosureOfRationalNumber has CHARNZ or PseudoAlgebraicClosureO
--R primeFrobenius : (%,NonNegativeInteger) -> % if PseudoAlgebraicClosureOfRationalNumber has CHARNZ or Ps
--R principalIdeal : List % -> Record(coef: List %,generator: %)
--R reduce : SparseUnivariatePolynomial % -> %
--R retract : % -> PseudoAlgebraicClosureOfRationalNumber
--R retract : % -> PseudoAlgebraicClosureOfRationalNumber
--R retractIfCan : % -> Union(PseudoAlgebraicClosureOfRationalNumber,"failed")
--R retractIfCan : % -> Union(PseudoAlgebraicClosureOfRationalNumber,"failed")
--R retractIfCan : % -> Union(Fraction Integer,"failed")
--R retractIfCan : % -> Union(Fraction Integer,"failed")
--R retractIfCan : % -> Union(Integer,"failed")
--R subtractIfCan : (%,%) -> Union(%,"failed")
--R transcendenceDegree : () -> NonNegativeInteger
--R unitNormal : % -> Record(unit: %,canonical: %,associate: %)
--R
--E 1

)spool
)lisp (bye)
```

———-

— PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory.help

—

```
====================================================================
PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory examples
====================================================================

See Also:
o )show PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory
```

———-

See:

⇐ "PseudoAlgebraicClosureOfRationalNumberCategory" (PACRATC) 19.0.215 on page 1610

Exports:

0	1	-?	?**?
?*?	?+?	?-?	?/?
?=?	?^?	?~=?	?quo?
?rem?	algebraic?	associates?	characteristic
charthRoot	coerce	conjugate	definingPolynomial
degree	dimension	discreteLog	distinguishedRootsOf
divide	euclideanSize	expressIdealMember	exquo
extDegree	extendedEuclidean	extensionDegree	factor
Frobenius	fullOutput	gcd	gcdPolynomial
ground?	hash	inGroundField?	inv
latex	lcm	lift	maxTower
multiEuclidean	newElement	one?	order
previousTower	prime?	primeFrobenius	principalIdeal
recip	reduce	retract	retractIfCan
sample	setTower!	sizeLess?	squareFree
squareFreePart	subtractIfCan	transcendenceDegree	transcendent?
unit?	unitCanonical	unitNormal	vectorise
zero?			

Attributes Exported:

- **commutative("*")** is true if it has an operation " $*$ " : $(D, D) -> D$ which is commutative.
- **unitsKnown** is true if a monoid (a multiplicative semigroup with a 1) has unitsKnown means that the operation `recip` can only return "failed" if its argument is not a unit.
- **leftUnitary** is true if $1 * x = x$ for all x.
- **rightUnitary** is true if $x * 1 = x$ for all x.
- **noZeroDivisors** is true if $x * y \neq 0$ implies both x and y are non-zero.
- **canonicalUnitNormal** is true if we can choose a canonical representative for each class of associate elements, that is `associates?(a,b)` returns true if and only if `unitCanonical(a) = unitCanonical(b)`.
- **canonicalsClosed** is true if `unitCanonical(a)*unitCanonical(b) = unitCanonical(a*b)`.

These exports come from (p1610) PseudoAlgebraicClosureOfRationalNumberCategory

```
-? : % -> %
0 : () -> %
```

```
1 : () -> %
?**? : (%,Integer) -> %
?**? : (%,NonNegativeInteger) -> %
?**? : (%,PositiveInteger) -> %
?*? : (%,%) -> %
?*? : (%,Fraction Integer) -> %
?*? : (%,Fraction Integer) -> %
?*? : (Fraction Integer,%) -> %
?*? : (Fraction Integer,%) -> %
?*? : (Integer,%) -> %
?*? : (NonNegativeInteger,%) -> %
?*? : (PositiveInteger,%) -> %
?+? : (%,%) -> %
?-? : (%,%) -> %
?/? : (%,%) -> %
?/? : (%,Fraction Integer) -> %
?=? : (%,%) -> Boolean
?^? : (%,Integer) -> %
?^? : (%,NonNegativeInteger) -> %
?^? : (%,PositiveInteger) -> %
?quo? : (%,%) -> %
?rem? : (%,%) -> %
?~=? : (%,%) -> Boolean
Frobenius : % -> % if Fraction Integer has FINITE
Frobenius : (%,NonNegativeInteger) -> % if Fraction Integer has FINITE
algebraic? : % -> Boolean
associates? : (%,%) -> Boolean
characteristic : () -> NonNegativeInteger
charthRoot : % -> Union(%,"failed")
   if Fraction Integer has CHARNZ or Fraction Integer has FINITE
coerce : % -> %
coerce : % -> OutputForm
coerce : Fraction Integer -> %
coerce : Fraction Integer -> %
coerce : Fraction Integer -> %
coerce : Integer -> %
coerce : Integer -> %
conjugate : % -> %
definingPolynomial : % -> SparseUnivariatePolynomial %
definingPolynomial : () -> SparseUnivariatePolynomial %
degree : % -> OnePointCompletion PositiveInteger
dimension : () -> CardinalNumber
discreteLog : (%,%) -> Union(NonNegativeInteger,"failed")
   if Fraction Integer has CHARNZ or Fraction Integer has FINITE
distinguishedRootsOf : (SparseUnivariatePolynomial %,%) -> List %
divide : (%,%) -> Record(quotient: %,remainder: %)
euclideanSize : % -> NonNegativeInteger
expressIdealMember : (List %,%) -> Union(List %,"failed")
exquo : (%,%) -> Union(%,"failed")
extDegree : % -> PositiveInteger
```

```
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
extensionDegree : () -> OnePointCompletion PositiveInteger
factor : % -> Factored %
fullOutput : % -> OutputForm
gcd : (%,%) -> %
gcd : List % -> %
gcdPolynomial : (SparseUnivariatePolynomial %,
             SparseUnivariatePolynomial %) -> SparseUnivariatePolynomial %
ground? : % -> Boolean
hash : % -> SingleInteger
inGroundField? : % -> Boolean
inv : % -> %
latex : % -> String
lcm : (%,%) -> %
lcm : List % -> %
lift : % -> SparseUnivariatePolynomial %
lift : (%,%) -> SparseUnivariatePolynomial %
maxTower : List % -> %
multiEuclidean : (List %,%) -> Union(List %,"failed")
newElement : (SparseUnivariatePolynomial %,%,Symbol) -> %
newElement : (SparseUnivariatePolynomial %,Symbol) -> %
one? : % -> Boolean
order : % -> OnePointCompletion PositiveInteger
   if Fraction Integer has CHARNZ or Fraction Integer has FINITE
previousTower : % -> %
prime? : % -> Boolean
primeFrobenius : % -> %
   if Fraction Integer has CHARNZ or Fraction Integer has FINITE
primeFrobenius : (%,NonNegativeInteger) -> %
   if Fraction Integer has CHARNZ or Fraction Integer has FINITE
principalIdeal : List % -> Record(coef: List %,generator: %)
recip : % -> Union(%,"failed")
reduce : SparseUnivariatePolynomial % -> %
retract : % -> Fraction Integer
retract : % -> Fraction Integer
retract : % -> Integer
retractIfCan : % -> Union(Fraction Integer,"failed")
retractIfCan : % -> Union(Fraction Integer,"failed")
retractIfCan : % -> Union(Integer,"failed")
sample : () -> %
setTower! : % -> Void
sizeLess? : (%,%) -> Boolean
squareFree : % -> Factored %
squareFreePart : % -> %
subtractIfCan : (%,%) -> Union(%,"failed")
transcendenceDegree : () -> NonNegativeInteger
transcendent? : % -> Boolean
unit? : % -> Boolean
unitCanonical : % -> %
```

```
 unitNormal : % -> Record(unit: %,canonical: %,associate: %)
 vectorise : (%,%) -> Vector %
 zero? : % -> Boolean

These exports come from \refto{RetractableTo}
\begin{verbatim}
 coerce : PseudoAlgebraicClosureOfRationalNumber -> %
 coerce : PseudoAlgebraicClosureOfRationalNumber -> %
 retract : % -> PseudoAlgebraicClosureOfRationalNumber
 retract : % -> PseudoAlgebraicClosureOfRationalNumber
 retractIfCan : % -> Union(PseudoAlgebraicClosureOfRationalNumber,"failed")
 retractIfCan : % -> Union(PseudoAlgebraicClosureOfRationalNumber,"failed")
```

These exports come from (p1510) ExtensionField

```
 ?*? : (%,PseudoAlgebraicClosureOfRationalNumber) -> %
 ?*? : (PseudoAlgebraicClosureOfRationalNumber,%) -> %
 ?/? : (%,PseudoAlgebraicClosureOfRationalNumber) -> %
 charthRoot : % -> Union(%,"failed") if PseudoAlgebraicClosureOfRationalNumber has CHARNZ or PseudoAlgebrai
 discreteLog : (%,%) -> Union(NonNegativeInteger,"failed") if PseudoAlgebraicClosureOfRationalNumber has CH
 Frobenius : % -> % if PseudoAlgebraicClosureOfRationalNumber has FINITE or Fraction Integer has FINITE
 Frobenius : (%,NonNegativeInteger) -> % if PseudoAlgebraicClosureOfRationalNumber has FINITE or Fraction In
 order : % -> OnePointCompletion PositiveInteger if PseudoAlgebraicClosureOfRationalNumber has CHARNZ or Ps
 primeFrobenius : % -> % if PseudoAlgebraicClosureOfRationalNumber has CHARNZ or PseudoAlgebraicClosureOfRat
 primeFrobenius : (%,NonNegativeInteger) -> % if PseudoAlgebraicClosureOfRationalNumber has CHARNZ or Pseudo
```

— category PACEXTC PseudoAlgebraicClosureOfAlgExtOfRational-NumberCategory —

```
)abbrev category PACEXTC PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory
++ Authors: Gaetan Hache
++ Date Created: jan 1998
++ Date Last Updated: May 2010 by Tim Daly
++ Description:
++ This category exports the function for the domain
++ PseudoAlgebraicClosureOfAlgExtOfRationalNumber which implement dynamic
++ extension using the simple notion of tower extensions. A tower extension
++ T of the ground field K is any sequence of field extension
++ (T : K_0, K_1, ..., K_i...,K_n) where K_0 = K and for i =1,2,...,n,
++ K_i is an extension of K_{i-1} of degree > 1 and defined by an
++ irreducible polynomial p(Z) in K_{i-1}.
++ Two towers (T_1: K_01, K_11,...,K_i1,...,K_n1) and
++ (T_2: K_02, K_12,...,K_i2,...,K_n2)
++ are said to be related if T_1 <= T_2 (or T_1 >= T_2),
++ that is if K_i1 = K_i2 for i=1,2,...,n1 (or i=1,2,...,n2).
++ Any algebraic operations defined for several elements
++ are only defined if all of the concerned elements are comming from
++ a set of related tour extensions.
```

```
PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory:Category == Impl where
  Q    ==> PseudoAlgebraicClosureOfRationalNumber
  Impl ==>
    Join(PseudoAlgebraicClosureOfRationalNumberCategory,_
         RetractableTo(Q),ExtensionField(Q))
```

———-

— PACEXTC.dotabb —

```
"PACEXTC" [color=lightblue,href="bookvol10.2.pdf#nameddest=PACEXTC"];
"PACEXTC" -> "PACRATC"
"PACEXTC" -> "RETRACT"
"PACEXTC" -> "XF"
```

———-

— PACEXTC.dotfull —

```
"PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PACEXTC"];
   -> "PseudoAlgebraicClosureOfRationalNumberCategory"
"PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory"
   -> "RetractableTo(PseudoAlgebraicClosureOfRationalNumber)"
"PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory"
   -> "ExtensionField(PseudoAlgebraicClosureOfRationalNumber)"

"PseudoAlgebraicClosureOfRationalNumberCategory"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PACRATC"];

"RetractableTo(PseudoAlgebraicClosureOfRationalNumber)"
  [color=lightblue,href="bookvol10.2.pdf#nameddest=RETRACT"];
 ->  "RetractableTo(a:Type)"

"RetractableTo(a:Type)"
  [color=lightblue,href="bookvol10.2.pdf#nameddest=RETRACT"];

"ExtensionField(PseudoAlgebraicClosureOfRationalNumber)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=XF"];
"ExtensionField(PseudoAlgebraicClosureOfRationalNumber)" ->
 ->  "ExtensionField(a:Field)"

"ExtensionField(a:Field)"
  [color=lightblue,href="bookvol10.2.pdf#nameddest=XF"];
```

———-

— PACEXTC.dotpic —

```
digraph pic {
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

"PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=PACEXTC"];
   -> "PseudoAlgebraicClosureOfRationalNumberCategory"
"PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory"
   -> "RetractableTo(PseudoAlgebraicClosureOfRationalNumber)"
"PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory"
   -> "ExtensionField(PseudoAlgebraicClosureOfRationalNumber)"

"RetractableTo(PseudoAlgebraicClosureOfRationalNumber)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=RETRACT"];
"RetractableTo(PseudoAlgebraicClosureOfRationalNumber)"
  -> "RETRACT..."

"ExtensionField(PseudoAlgebraicClosureOfRationalNumber)"
 [color=lightblue,href="bookvol10.2.pdf#nameddest=XF"];
"ExtensionField(PseudoAlgebraicClosureOfRationalNumber)" ->
 ->  "XF..."

"XF..." [color=lightblue];
"RETRACT..." [color=lightblue];

}
```

———-

Chapter 21

The bootstrap code

21.1 ABELGRP.lsp BOOTSTRAP

ABELGRP depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **ABELGRP** category which we can write into the **MID** directory. We compile the lisp code and copy the **ABELGRP.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— ABELGRP.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |AbelianGroup;AL| (QUOTE NIL))

(DEFUN |AbelianGroup| NIL
  (LET (#:G82664)
    (COND
      (|AbelianGroup;AL|)
      (T (SETQ |AbelianGroup;AL| (|AbelianGroup;|))))))

(DEFUN |AbelianGroup;| NIL
  (PROG (#1=#:G82662)
    (RETURN
      (PROG1
        (LETT #1#
          (|Join|
            (|CancellationAbelianMonoid|)
            (|mkCategory|
              (QUOTE |domain|)
```

```
             (QUOTE (
               ((|-| (|$| |$|)) T)
               ((|-| (|$| |$| |$|)) T)
               ((|*| (|$| (|Integer|) |$|)) T)))
             NIL
             (QUOTE ((|Integer|)))
             NIL))
          |AbelianGroup|)
        (SETELT #1# 0 (QUOTE (|AbelianGroup|)))))))

(setf (get (quote |AbelianGroup|) (quote niladic)) t)
```

———-

21.2 ABELGRP-.lsp BOOTSTRAP

ABELGRP- depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **ABELGRP-** category which we can write into the **MID** directory. We compile the lisp code and copy the **ABELGRP-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— ABELGRP-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |ABELGRP-;-;3S;1| (|x| |y| |$|)
  (SPADCALL |x| (SPADCALL |y| (QREFELT |$| 7)) (QREFELT |$| 8)))

(DEFUN |ABELGRP-;subtractIfCan;2SU;2| (|x| |y| |$|)
  (CONS 0 (SPADCALL |x| |y| (QREFELT |$| 10))))

(DEFUN |ABELGRP-;*;Nni2S;3| (|n| |x| |$|)
  (SPADCALL |n| |x| (QREFELT |$| 14)))

(DEFUN |ABELGRP-;*;I2S;4| (|n| |x| |$|)
  (COND
    ((ZEROP |n|) (|spadConstant| |$| 17))
    ((|<| 0 |n|) (SPADCALL |n| |x| (QREFELT |$| 20)))
    ((QUOTE T)
      (SPADCALL (|-| |n|) (SPADCALL |x| (QREFELT |$| 7)) (QREFELT |$| 20)))))

(DEFUN |AbelianGroup&| (|#1|)
  (PROG (|DV$1| |dv$| |$| |pv$|)
    (RETURN
```

```
      (PROGN
        (LETT |DV$1| (|devaluate| |#1|) . #1=(|AbelianGroup&|))
        (LETT |dv$| (LIST (QUOTE |AbelianGroup&|) |DV$1|) . #1#)
        (LETT |$| (make-array 22) . #1#)
        (QSETREFV |$| 0 |dv$|)
        (QSETREFV |$| 3 (LETT |pv$| (|buildPredVector| 0 0 NIL) . #1#))
        (|stuffDomainSlots| |$|)
        (QSETREFV |$| 6 |#1|)
        (COND
          ((|HasCategory| |#1| (QUOTE (|Ring|))))
          ((QUOTE T)
            (QSETREFV |$| 21
              (CONS (|dispatchFunction| |ABELGRP-;*;I2S;4|) |$|))))
        |$|))))

(setf (get
  (QUOTE |AbelianGroup&|)
  (QUOTE |infovec|))
  (LIST
    (QUOTE
      #(NIL NIL NIL NIL NIL NIL
        (|local| |#1|)
        (0 . |-|)
        (5 . |+|)
        |ABELGRP-;-;3S;1|
        (11 . |-|)
        (|Union| |$| (QUOTE "failed"))
        |ABELGRP-;subtractIfCan;2SU;2|
        (|Integer|)
        (17 . |*|)
        (|NonNegativeInteger|)
        |ABELGRP-;*;Nni2S;3|
        (23 . |Zero|)
        (|PositiveInteger|)
        (|RepeatedDoubling| 6)
        (27 . |double|)
        (33 . |*|)))
    (QUOTE #(|subtractIfCan| 39 |-| 45 |*| 51))
    (QUOTE NIL)
    (CONS
      (|makeByteWordVec2| 1 (QUOTE NIL))
      (CONS
        (QUOTE #())
        (CONS
          (QUOTE #())
          (|makeByteWordVec2| 21
            (QUOTE (1 6 0 0 7 2 6 0 0 0 8 2 6 0 0 0 10 2 6 0 13 0 14 0 6 0 17
                    2 19 6 18 6 20 2 0 0 13 0 21 2 0 11 0 0 12 2 0 0 0 0 9 2
                    0 0 13 0 21 2 0 0 15 0 16))))))
    (QUOTE |lookupComplete|)))
```

21.3 ABELMON.lsp BOOTSTRAP

ABELMON which needs **ABELSG** which needs **SETCAT** which needs **SINT** which needs **UFD** which needs **GCDDOM** which needs **COMRING** which needs **RING** which needs **RNG** which needs **ABELGRP** which needs **CABMON** which needs **ABELMON**. We break this chain with **ABELMON.lsp** which we cache here. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **ABELMON** category which we can write into the **MID** directory. We compile the lisp code and copy the **ABELMON.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— **ABELMON.lsp BOOTSTRAP** —

```
(|/VERSIONCHECK| 2)

(SETQ |AbelianMonoid;AL| (QUOTE NIL))

(DEFUN |AbelianMonoid| NIL
  (LET (#:G82597)
    (COND
      (|AbelianMonoid;AL|)
      (T (SETQ |AbelianMonoid;AL| (|AbelianMonoid;|))))))

(DEFUN |AbelianMonoid;| NIL
  (PROG (#1=#:G82595)
    (RETURN
      (PROG1
        (LETT #1#
          (|Join|
            (|AbelianSemiGroup|)
            (|mkCategory|
              (QUOTE |domain|)
              (QUOTE (
                ((|Zero| (|$|) |constant|) T)
                ((|sample| (|$|) |constant|) T)
                ((|zero?| ((|Boolean|) |$|)) T)
                ((|*| (|$| (|NonNegativeInteger|) |$|)) T)))
              NIL
              (QUOTE ((|NonNegativeInteger|) (|Boolean|)))
              NIL))
            |AbelianMonoid|)
```

```
        (SETELT #1# 0 (QUOTE (|AbelianMonoid|)))))))

(setf (get (QUOTE |AbelianMonoid|) (QUOTE NILADIC)) T)
```

———-

21.4 ABELMON-.lsp BOOTSTRAP

ABELMON- which needs **ABELSG** which needs **SETCAT** which needs **SINT** which needs **UFD** which needs **GCDDOM** which needs **COMRING** which needs **RING** which needs **RNG** which needs **ABELGRP** which needs **CABMON** which needs **ABELMON-**. We break this chain with **ABELMON-.lsp** which we cache here. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **ABELMON-** category which we can write into the **MID** directory. We compile the lisp code and copy the **ABELMON-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— ABELMON-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |ABELMON-;zero?;SB;1| (|x| |$|)
  (SPADCALL |x| (|spadConstant| |$| 7) (QREFELT |$| 9)))

(DEFUN |ABELMON-;*;Pi2S;2| (|n| |x| |$|)
  (SPADCALL |n| |x| (QREFELT |$| 12)))

(DEFUN |ABELMON-;sample;S;3| (|$|)
  (|spadConstant| |$| 7))

(DEFUN |ABELMON-;*;Nni2S;4| (|n| |x| |$|)
  (COND
    ((ZEROP |n|) (|spadConstant| |$| 7))
    ((QUOTE T) (SPADCALL |n| |x| (QREFELT |$| 17)))))

(DEFUN |AbelianMonoid&| (|#1|)
  (PROG (|DV$1| |dv$| |$| |pv$|)
    (RETURN
      (PROGN
        (LETT |DV$1| (|devaluate| |#1|) . #1=(|AbelianMonoid&|))
        (LETT |dv$| (LIST (QUOTE |AbelianMonoid&|) |DV$1|) . #1#)
        (LETT |$| (make-array 19) . #1#)
        (QSETREFV |$| 0 |dv$|)
```

```
        (QSETREFV |$| 3 (LETT |pv$| (|buildPredVector| 0 0 NIL) . #1#))
        (|stuffDomainSlots| |$|)
        (QSETREFV |$| 6 |#1|)
        (COND
          ((|HasCategory| |#1| (QUOTE (|Ring|))))
          ((QUOTE T)
            (QSETREFV |$| 18
              (CONS (|dispatchFunction| |ABELMON-;*;Nni2S;4|) |$|)))) |$|))))

(setf (get
  (QUOTE |AbelianMonoid&|)
  (QUOTE |infovec|))
  (LIST
    (QUOTE
      #(NIL NIL NIL NIL NIL NIL
        (|local| |#1|)
        (0 . |Zero|)
        (|Boolean|)
        (4 . |=|)
        |ABELMON-;zero?;SB;1|
        (|NonNegativeInteger|)
        (10 . |*|)
        (|PositiveInteger|)
        |ABELMON-;*;Pi2S;2|
        |ABELMON-;sample;S;3|
        (|RepeatedDoubling| 6)
        (16 . |double|)
        (22 . |*|)))
    (QUOTE #(|zero?| 28 |sample| 33 |*| 37))
    (QUOTE NIL)
    (CONS
      (|makeByteWordVec2| 1 (QUOTE NIL))
      (CONS
        (QUOTE #())
        (CONS
          (QUOTE #())
          (|makeByteWordVec2| 18
            (QUOTE (0 6 0 7 2 6 8 0 0 9 2 6 0 11 0 12 2 16 6 13 6 17 2 0 0 11
                    0 18 1 0 8 0 10 0 0 0 15 2 0 0 11 0 18 2 0 0 13 0 14))))))
   (QUOTE |lookupComplete|)))
```

———-

21.5 ABELSG.lsp BOOTSTRAP

ABELSG needs **SETCAT** which needs **SINT** which needs **UFD** which needs **GCDDOM** which needs **COMRING** which needs **RING** which needs **RNG** which needs **ABELGRP**

which needs **CABMON** which needs **ABELMON** which needs **ABELSG**. We break this chain with **ABELSG.lsp** which we cache here. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **ABELSG** category which we can write into the **MID** directory. We compile the lisp code and copy the **ABELSG.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— **ABELSG.lsp BOOTSTRAP** —

```
(|/VERSIONCHECK| 2)

(SETQ |AbelianSemiGroup;AL| (QUOTE NIL))

(DEFUN |AbelianSemiGroup| NIL
  (LET (#:G82568)
    (COND
      (|AbelianSemiGroup;AL|)
      (T (SETQ |AbelianSemiGroup;AL| (|AbelianSemiGroup;|))))))

(DEFUN |AbelianSemiGroup;| NIL
  (PROG (#1=#:G82566)
    (RETURN
      (PROG1
        (LETT #1#
          (|Join|
            (|SetCategory|)
            (|mkCategory|
              (QUOTE |domain|)
              (QUOTE (
                ((|+| (|$| |$| |$|)) T)
                ((|*| (|$| (|PositiveInteger|) |$|)) T)))
              NIL
              (QUOTE ((|PositiveInteger|)))
              NIL))
            |AbelianSemiGroup|)
        (SETELT #1# 0 (QUOTE (|AbelianSemiGroup|)))))))

(setf (get (QUOTE |AbelianSemiGroup|) (QUOTE NILADIC)) T)
```

———-

21.6 ABELSG-.lsp BOOTSTRAP

ABELSG- needs **SETCAT** which needs **SINT** which needs **UFD** which needs **GCDDOM** which needs **COMRING** which needs **RING** which needs **RNG** which needs **ABELGRP**

which needs **CABMON** which needs **ABELMON** which needs **ABELSG-**. We break this chain with **ABELSG-.lsp** which we cache here. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **ABELSG-** category which we can write into the **MID** directory. We compile the lisp code and copy the **ABELSG-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— ABELSG-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |ABELSG-;*;Pi2S;1| (|n| |x| |$|) (SPADCALL |n| |x| (QREFELT |$| 9)))

(DEFUN |AbelianSemiGroup&| (|#1|)
  (PROG (|DV$1| |dv$| |$| |pv$|)
    (RETURN
      (PROGN
        (LETT |DV$1| (|devaluate| |#1|) . #1=(|AbelianSemiGroup&|))
        (LETT |dv$| (LIST (QUOTE |AbelianSemiGroup&|) |DV$1|) . #1#)
        (LETT |$| (make-array 11) . #1#)
        (QSETREFV |$| 0 |dv$|)
        (QSETREFV |$| 3 (LETT |pv$| (|buildPredVector| 0 0 NIL) . #1#))
        (|stuffDomainSlots| |$|)
        (QSETREFV |$| 6 |#1|)
        (COND
          ((|HasCategory| |#1| (QUOTE (|Ring|))))
          ((QUOTE T)
            (QSETREFV |$| 10
              (CONS (|dispatchFunction| |ABELSG-;*;Pi2S;1|) |$|))))
        |$|))))

(setf (get
  (QUOTE |AbelianSemiGroup&|)
  (QUOTE |infovec|))
  (LIST
    (QUOTE
      #(NIL NIL NIL NIL NIL NIL
        (|local| |#1|)
        (|PositiveInteger|)
        (|RepeatedDoubling| 6)
        (0 . |double|)
        (6 . |*|)))
    (QUOTE #(|*| 12))
    (QUOTE NIL)
    (CONS
      (|makeByteWordVec2| 1 (QUOTE NIL))
      (CONS
```

```
        (QUOTE #())
        (CONS
          (QUOTE #())
          (|makeByteWordVec2| 10
            (QUOTE (2 8 6 7 6 9 2 0 0 7 0 10 2 0 0 7 0 10))))))
    (QUOTE |lookupComplete|)))
```

———-

21.7 ALAGG.lsp BOOTSTRAP

ALAGG depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **ALAGG** category which we can write into the **MID** directory. We compile the lisp code and copy the **ALAGG.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— ALAGG.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |AssociationListAggregate;CAT| (QUOTE NIL))

(SETQ |AssociationListAggregate;AL| (QUOTE NIL))

(DEFUN |AssociationListAggregate|
 (|&REST| #1=#:G88404 |&AUX| #2=#:G88402)
  (DSETQ #2# #1#)
  (LET (#3=#:G88403)
   (COND
    ((SETQ #3# (|assoc| (|devaluateList| #2#) |AssociationListAggregate;AL|))
      (CDR #3#))
    (T
      (SETQ |AssociationListAggregate;AL|
       (|cons5|
        (CONS
         (|devaluateList| #2#)
         (SETQ #3# (APPLY (FUNCTION |AssociationListAggregate;|) #2#)))
        |AssociationListAggregate;AL|)) #3#))))

(DEFUN |AssociationListAggregate;| (|t#1| |t#2|)
 (PROG (#1=#:G88401)
  (RETURN
   (PROG1
    (LETT #1#
     (|sublisV|
```

```
    (PAIR
     (QUOTE (|t#1| |t#2|)) (LIST (|devaluate| |t#1|) (|devaluate| |t#2|)))
    (|sublisV|
     (PAIR
      (QUOTE (#2=#:G88400))
      (LIST (QUOTE (|Record| (|:| |key| |t#1|) (|:| |entry| |t#2|)))))
     (COND
      (|AssociationListAggregate;CAT|)
      ((QUOTE T)
       (LETT |AssociationListAggregate;CAT|
        (|Join|
         (|TableAggregate| (QUOTE |t#1|) (QUOTE |t#2|))
         (|ListAggregate| (QUOTE #2#))
         (|mkCategory|
          (QUOTE |domain|)
          (QUOTE
           ((((|assoc|
              ((|Union|
                (|Record| (|:| |key| |t#1|) (|:| |entry| |t#2|)) "failed")
               |t#1| |$|))
              T)))
          NIL (QUOTE NIL) NIL))
         . #3=(|AssociationListAggregate|))))))
      . #3#)
   (SETELT #1# 0
    (LIST
     (QUOTE |AssociationListAggregate|)
     (|devaluate| |t#1|)
     (|devaluate| |t#2|)))))))
```

———-

21.8 CABMON.lsp BOOTSTRAP

CABMON which needs **ABELMON** which needs **ABELSG** which needs **SETCAT** which needs **SINT** which needs **UFD** which needs **GCDDOM** which needs **COMRING** which needs **RING** which needs **RNG** which needs **ABELGRP** which needs **CABMON**. We break this chain with **CABMON.lsp** which we cache here. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **CABMON** category which we can write into the **MID** directory. We compile the lisp code and copy the **CABMON.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— CABMON.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |CancellationAbelianMonoid;AL| (QUOTE NIL))

(DEFUN |CancellationAbelianMonoid| NIL
  (LET (#:G82646)
    (COND
      (|CancellationAbelianMonoid;AL|)
      (T
        (SETQ
          |CancellationAbelianMonoid;AL|
          (|CancellationAbelianMonoid;|))))))

(DEFUN |CancellationAbelianMonoid;| NIL
  (PROG (#1=#:G82644)
    (RETURN
      (PROG1
        (LETT #1#
          (|Join|
            (|AbelianMonoid|)
            (|mkCategory|
              (QUOTE |domain|)
              (QUOTE
               ((((|subtractIfCan| ((|Union| |$| "failed") |$| |$|)) T)))
              NIL
             (QUOTE NIL)
             NIL))
           |CancellationAbelianMonoid|)
        (SETELT #1# 0 (QUOTE (|CancellationAbelianMonoid|)))))))

(setf (get (QUOTE |CancellationAbelianMonoid|) (QUOTE NILADIC)) T)
```

———-

21.9 CLAGG.lsp BOOTSTRAP

CLAGG depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **CLAGG** category which we can write into the **MID** directory. We compile the lisp code and copy the **CLAGG.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— CLAGG.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)
```

```
(SETQ |Collection;CAT| (QUOTE NIL))

(SETQ |Collection;AL| (QUOTE NIL))

(DEFUN |Collection| (#1=#:G82618)
 (LET (#2=#:G82619)
  (COND
   ((SETQ #2# (|assoc| (|devaluate| #1#) |Collection;AL|)) (CDR #2#))
   (T
    (SETQ |Collection;AL|
     (|cons5|
      (CONS
       (|devaluate| #1#)
       (SETQ #2# (|Collection;| #1#)))
      |Collection;AL|))
     #2#))))

(DEFUN |Collection;| (|t#1|)
 (PROG (#1=#:G82617)
  (RETURN
   (PROG1
    (LETT #1#
     (|sublisV|
      (PAIR (QUOTE (|t#1|)) (LIST (|devaluate| |t#1|)))
      (COND
       (|Collection;CAT|)
       ((QUOTE T)
        (LETT |Collection;CAT|
         (|Join|
          (|HomogeneousAggregate| (QUOTE |t#1|))
          (|mkCategory|
           (QUOTE |domain|)
           (QUOTE (
            ((|construct| (|$| (|List| |t#1|))) T)
            ((|find| ((|Union| |t#1| "failed")
                      (|Mapping| (|Boolean|) |t#1|) |$|))
              T)
            ((|reduce| (|t#1| (|Mapping| |t#1| |t#1| |t#1|) |$|))
             (|has| |$| (ATTRIBUTE |finiteAggregate|)))
            ((|reduce| (|t#1| (|Mapping| |t#1| |t#1| |t#1|) |$| |t#1|))
             (|has| |$| (ATTRIBUTE |finiteAggregate|)))
            ((|remove| (|$| (|Mapping| (|Boolean|) |t#1|) |$|))
             (|has| |$| (ATTRIBUTE |finiteAggregate|)))
            ((|select| (|$| (|Mapping| (|Boolean|) |t#1|) |$|))
             (|has| |$| (ATTRIBUTE |finiteAggregate|)))
            ((|reduce| (|t#1| (|Mapping| |t#1| |t#1| |t#1|) |$| |t#1| |t#1|))
             (AND
              (|has| |t#1| (|SetCategory|))
              (|has| |$| (ATTRIBUTE |finiteAggregate|))))
```

```
           ((|remove| (|$| |t#1| |$|))
            (AND
             (|has| |t#1| (|SetCategory|))
             (|has| |$| (ATTRIBUTE |finiteAggregate|))))
           ((|removeDuplicates| (|$| |$|))
            (AND
             (|has| |t#1| (|SetCategory|))
             (|has| |$| (ATTRIBUTE |finiteAggregate|))))))
          (QUOTE ((((|ConvertibleTo| (|InputForm|))
                    (|has| |t#1| (|ConvertibleTo| (|InputForm|))))))
          (QUOTE ((|List| |t#1|))) NIL))
         . #2=(|Collection|)))))
     . #2#)
   (SETELT #1# 0 (LIST (QUOTE |Collection|) (|devaluate| |t#1|)))))))
```

———-

21.10 CLAGG-.lsp BOOTSTRAP

CLAGG- depends on **CLAGG**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **CLAGG-** category which we can write into the **MID** directory. We compile the lisp code and copy the **CLAGG-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— CLAGG-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |CLAGG-;#;ANni;1| (|c| |$|) (LENGTH (SPADCALL |c| (QREFELT |$| 9))))

(DEFUN |CLAGG-;count;MANni;2| (|f| |c| |$|)
 (PROG (|x| #1=#:G82637 #2=#:G82634 #3=#:G82632 #4=#:G82633)
  (RETURN
   (SEQ
    (PROGN
     (LETT #4# NIL |CLAGG-;count;MANni;2|)
     (SEQ
      (LETT |x| NIL |CLAGG-;count;MANni;2|)
      (LETT #1# (SPADCALL |c| (QREFELT |$| 9)) |CLAGG-;count;MANni;2|)
      G190
      (COND
       ((OR (ATOM #1#) (PROGN (LETT |x| (CAR #1#) |CLAGG-;count;MANni;2|) NIL))
        (GO G191)))
      (SEQ
       (EXIT
```

```
        (COND
         ((SPADCALL |x| |f|)
          (PROGN
           (LETT #2# 1 |CLAGG-;count;MANni;2|)
           (COND
            (#4# (LETT #3# (|+| #3# #2#) |CLAGG-;count;MANni;2|))
            ((QUOTE T)
             (PROGN
              (LETT #3# #2# |CLAGG-;count;MANni;2|)
              (LETT #4# (QUOTE T) |CLAGG-;count;MANni;2|)))))))
      (LETT #1# (CDR #1#) |CLAGG-;count;MANni;2|)
      (GO G190)
      G191
      (EXIT NIL))
     (COND (#4# #3#) ((QUOTE T) 0)))))))

(DEFUN |CLAGG-;any?;MAB;3| (|f| |c| |$|)
 (PROG (|x| #1=#:G82642 #2=#:G82640 #3=#:G82638 #4=#:G82639)
  (RETURN
   (SEQ
    (PROGN
     (LETT #4# NIL |CLAGG-;any?;MAB;3|)
     (SEQ
      (LETT |x| NIL |CLAGG-;any?;MAB;3|)
      (LETT #1# (SPADCALL |c| (QREFELT |$| 9)) |CLAGG-;any?;MAB;3|)
      G190
      (COND
       ((OR (ATOM #1#) (PROGN (LETT |x| (CAR #1#) |CLAGG-;any?;MAB;3|) NIL))
        (GO G191)))
      (SEQ
       (EXIT
        (PROGN
         (LETT #2# (SPADCALL |x| |f|) |CLAGG-;any?;MAB;3|)
         (COND
          (#4#
           (LETT #3#
            (COND (#3# (QUOTE T)) ((QUOTE T) #2#))
            |CLAGG-;any?;MAB;3|))
          ((QUOTE T)
           (PROGN
            (LETT #3# #2# |CLAGG-;any?;MAB;3|)
            (LETT #4# (QUOTE T) |CLAGG-;any?;MAB;3|)))))))
      (LETT #1# (CDR #1#) |CLAGG-;any?;MAB;3|)
      (GO G190)
      G191
      (EXIT NIL))
     (COND (#4# #3#) ((QUOTE T) (QUOTE NIL)))))))))

(DEFUN |CLAGG-;every?;MAB;4| (|f| |c| |$|)
 (PROG (|x| #1=#:G82647 #2=#:G82645 #3=#:G82643 #4=#:G82644)
```

```
  (RETURN
   (SEQ
    (PROGN
     (LETT #4# NIL |CLAGG-;every?;MAB;4|)
     (SEQ
      (LETT |x| NIL |CLAGG-;every?;MAB;4|)
      (LETT #1# (SPADCALL |c| (QREFELT |$| 9)) |CLAGG-;every?;MAB;4|)
      G190
      (COND
       ((OR (ATOM #1#) (PROGN (LETT |x| (CAR #1#) |CLAGG-;every?;MAB;4|) NIL))
        (GO G191)))
      (SEQ
       (EXIT
        (PROGN
         (LETT #2# (SPADCALL |x| |f|) |CLAGG-;every?;MAB;4|)
         (COND
          (#4#
           (LETT #3#
            (COND (#3# #2#) ((QUOTE T) (QUOTE NIL)))
            |CLAGG-;every?;MAB;4|))
          ((QUOTE T)
           (PROGN
            (LETT #3# #2# |CLAGG-;every?;MAB;4|)
            (LETT #4# (QUOTE T) |CLAGG-;every?;MAB;4|)))))))
      (LETT #1# (CDR #1#) |CLAGG-;every?;MAB;4|)
      (GO G190)
      G191
      (EXIT NIL))
     (COND (#4# #3#) ((QUOTE T) (QUOTE T))))))))

(DEFUN |CLAGG-;find;MAU;5| (|f| |c| |$|)
 (SPADCALL |f| (SPADCALL |c| (QREFELT |$| 9)) (QREFELT |$| 18)))

(DEFUN |CLAGG-;reduce;MAS;6| (|f| |x| |$|)
 (SPADCALL |f| (SPADCALL |x| (QREFELT |$| 9)) (QREFELT |$| 21)))

(DEFUN |CLAGG-;reduce;MA2S;7| (|f| |x| |s| |$|)
 (SPADCALL |f| (SPADCALL |x| (QREFELT |$| 9)) |s| (QREFELT |$| 23)))

(DEFUN |CLAGG-;remove;M2A;8| (|f| |x| |$|)
 (SPADCALL
  (SPADCALL |f| (SPADCALL |x| (QREFELT |$| 9)) (QREFELT |$| 25))
  (QREFELT |$| 26)))

(DEFUN |CLAGG-;select;M2A;9| (|f| |x| |$|)
 (SPADCALL
  (SPADCALL |f| (SPADCALL |x| (QREFELT |$| 9)) (QREFELT |$| 28))
  (QREFELT |$| 26)))

(DEFUN |CLAGG-;remove;S2A;10| (|s| |x| |$|)
```

```
 (SPADCALL
  (CONS (FUNCTION |CLAGG-;remove;S2A;10!0|) (VECTOR |$| |s|))
  |x|
  (QREFELT |$| 31)))

(DEFUN |CLAGG-;remove;S2A;10!0| (|#1| |$$|)
 (SPADCALL |#1| (QREFELT |$$| 1) (QREFELT (QREFELT |$$| 0) 30)))

(DEFUN |CLAGG-;reduce;MA3S;11| (|f| |x| |s1| |s2| |$|)
 (SPADCALL |f| (SPADCALL |x| (QREFELT |$| 9)) |s1| |s2| (QREFELT |$| 33)))

(DEFUN |CLAGG-;removeDuplicates;2A;12| (|x| |$|)
 (SPADCALL
  (SPADCALL (SPADCALL |x| (QREFELT |$| 9)) (QREFELT |$| 35))
  (QREFELT |$| 26)))

(DEFUN |Collection&| (|#1| |#2|)
 (PROG (|DV$1| |DV$2| |dv$| |$| |pv$|)
  (RETURN
   (PROGN
    (LETT |DV$1| (|devaluate| |#1|) . #1=(|Collection&|))
    (LETT |DV$2| (|devaluate| |#2|) . #1#)
    (LETT |dv$| (LIST (QUOTE |Collection&|) |DV$1| |DV$2|) . #1#)
    (LETT |$| (make-array 37) . #1#)
    (QSETREFV |$| 0 |dv$|)
    (QSETREFV |$| 3
     (LETT |pv$|
      (|buildPredVector| 0 0
       (LIST
        (|HasCategory| |#2| (QUOTE (|ConvertibleTo| (|InputForm|))))
        (|HasCategory| |#2| (QUOTE (|SetCategory|)))
        (|HasAttribute| |#1| (QUOTE |finiteAggregate|))))
      . #1#))
    (|stuffDomainSlots| |$|)
    (QSETREFV |$| 6 |#1|)
    (QSETREFV |$| 7 |#2|)
    (COND
     ((|testBitVector| |pv$| 3)
      (PROGN
       (QSETREFV |$| 11 (CONS (|dispatchFunction| |CLAGG-;#;ANni;1|) |$|))
       (QSETREFV |$| 13 (CONS (|dispatchFunction| |CLAGG-;count;MANni;2|) |$|))
       (QSETREFV |$| 15 (CONS (|dispatchFunction| |CLAGG-;any?;MAB;3|) |$|))
       (QSETREFV |$| 16 (CONS (|dispatchFunction| |CLAGG-;every?;MAB;4|) |$|))
       (QSETREFV |$| 19 (CONS (|dispatchFunction| |CLAGG-;find;MAU;5|) |$|))
       (QSETREFV |$| 22 (CONS (|dispatchFunction| |CLAGG-;reduce;MAS;6|) |$|))
       (QSETREFV |$| 24 (CONS (|dispatchFunction| |CLAGG-;reduce;MA2S;7|) |$|))
       (QSETREFV |$| 27 (CONS (|dispatchFunction| |CLAGG-;remove;M2A;8|) |$|))
       (QSETREFV |$| 29 (CONS (|dispatchFunction| |CLAGG-;select;M2A;9|) |$|))
       (COND
        ((|testBitVector| |pv$| 2)
```

```
         (PROGN
          (QSETREFV |$| 32
           (CONS (|dispatchFunction| |CLAGG-;remove;S2A;10|) |$|))
          (QSETREFV |$| 34
           (CONS (|dispatchFunction| |CLAGG-;reduce;MA3S;11|) |$|))
          (QSETREFV |$| 36
           (CONS (|dispatchFunction| |CLAGG-;removeDuplicates;2A;12|)
                 |$|)))))))))
    |$|))))

(setf (get
 (QUOTE |Collection&|)
 (QUOTE |infovec|))
 (LIST (QUOTE
  #(NIL NIL NIL NIL NIL NIL (|local| |#1|) (|local| |#2|) (|List| 7)
   (0 . |parts|) (|NonNegativeInteger|) (5 . |#|) (|Mapping| 14 7)
   (10 . |count|) (|Boolean|) (16 . |any?|) (22 . |every?|)
   (|Union| 7 (QUOTE "failed")) (28 . |find|) (34 . |find|)
   (|Mapping| 7 7 7) (40 . |reduce|) (46 . |reduce|) (52 . |reduce|)
   (59 . |reduce|) (66 . |remove|) (72 . |construct|) (77 . |remove|)
   (83 . |select|) (89 . |select|) (95 . |=|) (101 . |remove|)
   (107 . |remove|) (113 . |reduce|) (121 . |reduce|)
   (129 . |removeDuplicates|) (134 . |removeDuplicates|)))
   (QUOTE #(|select| 139 |removeDuplicates| 145 |remove| 150 |reduce|
   162 |find| 183 |every?| 189 |count| 195 |any?| 201 |#| 207))
   (QUOTE NIL)
   (CONS
    (|makeByteWordVec2| 1 (QUOTE NIL))
    (CONS
     (QUOTE #())
     (CONS
      (QUOTE #())
      (|makeByteWordVec2| 36
       (QUOTE (1 6 8 0 9 1 0 10 0 11 2 0 10 12 0 13 2 0 14 12 0 15 2 0 14
        12 0 16 2 8 17 12 0 18 2 0 17 12 0 19 2 8 7 20 0 21 2 0 7 20 0 22
        3 8 7 20 0 7 23 3 0 7 20 0 7 24 2 8 0 12 0 25 1 6 0 8 26 2 0 0 12
        0 27 2 8 0 12 0 28 2 0 0 12 0 29 2 7 14 0 0 30 2 6 0 12 0 31 2 0 0
        7 0 32 4 8 7 20 0 7 7 33 4 0 7 20 0 7 7 34 1 8 0 0 35 1 0 0 0 36 2
        0 0 12 0 29 1 0 0 0 36 2 0 0 7 0 32 2 0 0 12 0 27 4 0 7 20 0 7 7 34
        3 0 7 20 0 7 24 2 0 7 20 0 22 2 0 17 12 0 19 2 0 14 12 0 16 2 0 10
        12 0 13 2 0 14 12 0 15 1 0 10 0 11))))))
    (QUOTE |lookupComplete|)))
```

21.11 COMRING.lsp BOOTSTRAP

COMRING depends on itself. We need to break this cycle to build the algebra. So we keep

a cached copy of the translated **COMRING** category which we can write into the **MID** directory. We compile the lisp code and copy the **COMRING.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— COMRING.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |CommutativeRing;AL| (QUOTE NIL))

(DEFUN |CommutativeRing| NIL
  (LET (#:G82892)
    (COND
      (|CommutativeRing;AL|)
      (T (SETQ |CommutativeRing;AL| (|CommutativeRing;|))))))

(DEFUN |CommutativeRing;| NIL
  (PROG (#1=#:G82890)
    (RETURN
      (PROG1
        (LETT #1#
          (|Join|
            (|Ring|)
            (|BiModule| (QUOTE |$|) (QUOTE |$|))
            (|mkCategory|
              (QUOTE |package|)
              NIL
              (QUOTE ((((|commutative| "*") T)))
              (QUOTE NIL)
              NIL))
           |CommutativeRing|)
        (SETELT #1# 0 (QUOTE (|CommutativeRing|)))))))

(setf (get (QUOTE |CommutativeRing|) (QUOTE NILADIC)) T)
```

———-

21.12 DIFRING.lsp BOOTSTRAP

DIFRING needs **INT** which needs **DIFRING**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **DIFRING** category which we can write into the **MID** directory. We compile the lisp code and copy the **DIFRING.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— **DIFRING.lsp BOOTSTRAP** —

```
(|/VERSIONCHECK| 2)

(SETQ |DifferentialRing;AL| (QUOTE NIL))

(DEFUN |DifferentialRing| NIL
  (LET (#:G84565)
    (COND
      (|DifferentialRing;AL|)
      (T (SETQ |DifferentialRing;AL| (|DifferentialRing;|))))))

(DEFUN |DifferentialRing;| NIL
  (PROG (#1=#:G84563)
    (RETURN
      (PROG1
        (LETT #1#
          (|Join|
            (|Ring|)
            (|mkCategory|
              (QUOTE |domain|)
              (QUOTE
                ((((|differentiate| (|$| |$|)) T)
                  ((D (|$| |$|)) T)
                  ((|differentiate| (|$| |$| (|NonNegativeInteger|))) T)
                  ((D (|$| |$| (|NonNegativeInteger|))) T)))
              NIL
              (QUOTE ((|NonNegativeInteger|)))
              NIL))
          |DifferentialRing|)
        (SETELT #1# 0 (QUOTE (|DifferentialRing|)))))))

(setf (get (QUOTE |DifferentialRing|) (QUOTE NILADIC)) T)
```

———-

21.13 DIFRING-.lsp BOOTSTRAP

DIFRING- needs **DIFRING**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **DIFRING-** category which we can write into the **MID** directory. We compile the lisp code and copy the **DIFRING-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— DIFRING-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |DIFRING-;D;2S;1| (|r| |$|)
  (SPADCALL |r| (QREFELT |$| 7)))

(DEFUN |DIFRING-;differentiate;SNniS;2| (|r| |n| |$|)
  (PROG (|i|)
    (RETURN
      (SEQ
        (SEQ
          (LETT |i| 1 |DIFRING-;differentiate;SNniS;2|)
          G190
          (COND ((QSGREATERP |i| |n|) (GO G191)))
          (SEQ
            (EXIT
              (LETT |r|
                (SPADCALL |r| (QREFELT |$| 7))
                |DIFRING-;differentiate;SNniS;2|)))
          (LETT |i| (QSADD1 |i|) |DIFRING-;differentiate;SNniS;2|)
          (GO G190)
          G191
          (EXIT NIL))
        (EXIT |r|)))))

(DEFUN |DIFRING-;D;SNniS;3| (|r| |n| |$|)
  (SPADCALL |r| |n| (QREFELT |$| 11)))

(DEFUN |DifferentialRing&| (|#1|)
  (PROG (|DV$1| |dv$| |$| |pv$|)
    (RETURN
      (PROGN
        (LETT |DV$1| (|devaluate| |#1|) . #1=(|DifferentialRing&|))
        (LETT |dv$| (LIST (QUOTE |DifferentialRing&|) |DV$1|) . #1#)
        (LETT |$| (make-array 13) . #1#)
        (QSETREFV |$| 0 |dv$|)
        (QSETREFV |$| 3 (LETT |pv$| (|buildPredVector| 0 0 NIL) . #1#))
        (|stuffDomainSlots| |$|)
        (QSETREFV |$| 6 |#1|)
        |$|))))

(setf (get
  (QUOTE |DifferentialRing&|)
  (QUOTE |infovec|))
  (LIST
    (QUOTE
      #(NIL NIL NIL NIL NIL NIL
```

```
        (|local| |#1|)
        (0 . |differentiate|)
        |DIFRING-;D;2S;1|
        (|NonNegativeInteger|)
        |DIFRING-;differentiate;SNniS;2|
        (5 . |differentiate|)
        |DIFRING-;D;SNniS;3|))
    (QUOTE #(|differentiate| 11 D 17))
    (QUOTE NIL)
    (CONS
      (|makeByteWordVec2| 1 (QUOTE NIL))
      (CONS
        (QUOTE #())
        (CONS
          (QUOTE #())
          (|makeByteWordVec2| 12
            (QUOTE
             (1 6 0 0 7 2 6 0 0 9 11 2 0 0 0 9 10 2 0 0 0 9 12 1 0 0 0 8))))))
    (QUOTE |lookupComplete|)))
```

———-

21.14 DIVRING.lsp BOOTSTRAP

DIVRING depends on **QFCAT** which eventually depends on **DIVRING**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **DIVRING** category which we can write into the **MID** directory. We compile the lisp code and copy the **DIVRING.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— **DIVRING.lsp BOOTSTRAP** —

```
(|/VERSIONCHECK| 2)

(SETQ |DivisionRing;AL| (QUOTE NIL))

(DEFUN |DivisionRing| NIL
  (LET (#:G84035)
    (COND
      (|DivisionRing;AL|)
      (T (SETQ |DivisionRing;AL| (|DivisionRing;|))))))

(DEFUN |DivisionRing;| NIL
  (PROG (#1=#:G84033)
```

```
    (RETURN
      (PROG1
        (LETT #1#
          (|sublisV|
            (PAIR
              (QUOTE (#2=#:G84032))
              (LIST (QUOTE (|Fraction| (|Integer|)))))
            (|Join|
              (|EntireRing|)
              (|Algebra| (QUOTE #2#))
              (|mkCategory|
                (QUOTE |domain|)
                (QUOTE (
                  ((|**| (|$| |$| (|Integer|))) T)
                  ((|^| (|$| |$| (|Integer|))) T)
                  ((|inv| (|$| |$|)) T)))
                NIL
                (QUOTE ((|Integer|)))
                NIL)))
          |DivisionRing|)
        (SETELT #1# 0 (QUOTE (|DivisionRing|)))))))

(setf (get (QUOTE |DivisionRing|) (QUOTE NILADIC)) T)
```

———-

21.15 DIVRING-.lsp BOOTSTRAP

DIVRING- depends on **DIVRING**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **DIVRING-** category which we can write into the **MID** directory. We compile the lisp code and copy the **DIVRING-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— DIVRING-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |DIVRING-;^;SIS;1| (|x| |n| |$|)
  (SPADCALL |x| |n| (QREFELT |$| 8)))

(DEFUN |DIVRING-;**;SIS;2| (|x| |n| |$|)
  (COND
    ((ZEROP |n|) (|spadConstant| |$| 10))
    ((SPADCALL |x| (QREFELT |$| 12))
```

```
      (COND
        ((|<| |n| 0) (|error| "division by zero"))
        ((QUOTE T) |x|)))
    ((|<| |n| 0)
      (SPADCALL (SPADCALL |x| (QREFELT |$| 14)) (|-| |n|) (QREFELT |$| 17)))
    ((QUOTE T) (SPADCALL |x| |n| (QREFELT |$| 17)))))

(DEFUN |DIVRING-;*;F2S;3| (|q| |x| |$|)
  (SPADCALL
    (SPADCALL
      (SPADCALL |q| (QREFELT |$| 20))
      (SPADCALL
        (SPADCALL (SPADCALL |q| (QREFELT |$| 21)) (QREFELT |$| 22))
        (QREFELT |$| 14))
      (QREFELT |$| 23))
    |x|
    (QREFELT |$| 24)))

(DEFUN |DivisionRing&| (|#1|)
  (PROG (|DV$1| |dv$| |$| |pv$|)
    (RETURN
      (PROGN
        (LETT |DV$1| (|devaluate| |#1|) . #1=(|DivisionRing&|))
        (LETT |dv$| (LIST (QUOTE |DivisionRing&|) |DV$1|) . #1#)
        (LETT |$| (make-array 27) . #1#)
        (QSETREFV |$| 0 |dv$|)
        (QSETREFV |$| 3 (LETT |pv$| (|buildPredVector| 0 0 NIL) . #1#))
        (|stuffDomainSlots| |$|)
        (QSETREFV |$| 6 |#1|)
        |$|))))

(setf (get
  (QUOTE |DivisionRing&|)
  (QUOTE |infovec|))
  (LIST
    (QUOTE
      #(NIL NIL NIL NIL NIL NIL
        (|local| |#1|)
        (|Integer|)
        (0 . |**|)
        |DIVRING-;^;SIS;1|
        (6 . |One|)
        (|Boolean|)
        (10 . |zero?|)
        (15 . |Zero|)
        (19 . |inv|)
        (|PositiveInteger|)
        (|RepeatedSquaring| 6)
        (24 . |expt|)
        |DIVRING-;**;SIS;2|
```

```
      (|Fraction| 7)
      (30 . |numer|)
      (35 . |denom|)
      (40 . |coerce|)
      (45 . |*|)
      (51 . |*|)
      |DIVRING-;*;F2S;3|
      (|NonNegativeInteger|)))
   (QUOTE #(|^| 57 |**| 63 |*| 69))
   (QUOTE NIL)
   (CONS
     (|makeByteWordVec2| 1 (QUOTE NIL))
     (CONS
       (QUOTE #())
       (CONS
         (QUOTE #())
           (|makeByteWordVec2| 25
             (QUOTE
               (2 6 0 0 7 8 0 6 0 10 1 6 11 0 12 0 6 0 13 1 6 0 0 14 2 16 6
                6 15 17 1 19 7 0 20 1 19 7 0 21 1 6 0 7 22 2 6 0 7 0 23 2 6
                0 0 0 24 2 0 0 0 7 9 2 0 0 0 7 18 2 0 0 19 0 25))))))
   (QUOTE |lookupComplete|)))
```

———-

21.16 ES.lsp BOOTSTRAP

ES depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **ES** category which we can write into the **MID** directory. We compile the lisp code and copy the **ES.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— ES.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |ExpressionSpace;AL| (QUOTE NIL))

(DEFUN |ExpressionSpace| NIL
 (LET (#:G82344)
  (COND
   (|ExpressionSpace;AL|)
   (T (SETQ |ExpressionSpace;AL| (|ExpressionSpace;|))))))
```

```
(DEFUN |ExpressionSpace;| NIL
 (PROG (#1=#:G82342)
  (RETURN
   (PROG1
    (LETT #1#
     (|sublisV|
      (PAIR
       (QUOTE (#2=#:G82340 #3=#:G82341))
       (LIST (QUOTE (|Kernel| |$|)) (QUOTE (|Kernel| |$|))))
      (|Join|
       (|OrderedSet|)
       (|RetractableTo| (QUOTE #2#))
       (|InnerEvalable| (QUOTE #3#) (QUOTE |$|))
       (|Evalable| (QUOTE |$|))
       (|mkCategory|
        (QUOTE |domain|)
        (QUOTE (
         ((|elt| (|$| (|BasicOperator|) |$|)) T)
         ((|elt| (|$| (|BasicOperator|) |$| |$|)) T)
         ((|elt| (|$| (|BasicOperator|) |$| |$| |$|)) T)
         ((|elt| (|$| (|BasicOperator|) |$| |$| |$| |$|)) T)
         ((|elt| (|$| (|BasicOperator|) (|List| |$|))) T)
         ((|subst| (|$| |$| (|Equation| |$|))) T)
         ((|subst| (|$| |$| (|List| (|Equation| |$|)))) T)
         ((|subst| (|$| |$| (|List| (|Kernel| |$|)) (|List| |$|))) T)
         ((|box| (|$| |$|)) T)
         ((|box| (|$| (|List| |$|))) T)
         ((|paren| (|$| |$|)) T)
         ((|paren| (|$| (|List| |$|))) T)
         ((|distribute| (|$| |$|)) T)
         ((|distribute| (|$| |$| |$|)) T)
         ((|height| ((|NonNegativeInteger|) |$|)) T)
         ((|mainKernel| ((|Union| (|Kernel| |$|) "failed") |$|)) T)
         ((|kernels| ((|List| (|Kernel| |$|)) |$|)) T)
         ((|tower| ((|List| (|Kernel| |$|)) |$|)) T)
         ((|operators| ((|List| (|BasicOperator|)) |$|)) T)
         ((|operator| ((|BasicOperator|) (|BasicOperator|))) T)
         ((|belong?| ((|Boolean|) (|BasicOperator|))) T)
         ((|is?| ((|Boolean|) |$| (|BasicOperator|))) T)
         ((|is?| ((|Boolean|) |$| (|Symbol|))) T)
         ((|kernel| (|$| (|BasicOperator|) |$|)) T)
         ((|kernel| (|$| (|BasicOperator|) (|List| |$|))) T)
         ((|map| (|$| (|Mapping| |$| |$|) (|Kernel| |$|))) T)
         ((|freeOf?| ((|Boolean|) |$| |$|)) T)
         ((|freeOf?| ((|Boolean|) |$| (|Symbol|))) T)
         ((|eval| (|$| |$| (|List| (|Symbol|)) (|List| (|Mapping| |$| |$|))))
            T)
         ((|eval|
           (|$| |$| (|List| (|Symbol|)) (|List| (|Mapping| |$| (|List| |$|)))))
           T)
```

```
        ((|eval| (|$| |$| (|Symbol|) (|Mapping| |$| (|List| |$|)))) T)
        ((|eval| (|$| |$| (|Symbol|) (|Mapping| |$| |$|))) T)
        ((|eval|
          (|$| |$| (|List| (|BasicOperator|)) (|List| (|Mapping| |$| |$|))))
          T)
        ((|eval|
          (|$| |$| (|List| (|BasicOperator|))
            (|List| (|Mapping| |$| (|List| |$|)))))
          T)
        ((|eval| (|$| |$| (|BasicOperator|) (|Mapping| |$| (|List| |$|)))) T)
        ((|eval| (|$| |$| (|BasicOperator|) (|Mapping| |$| |$|))) T)
        ((|minPoly|
          ((|SparseUnivariatePolynomial| |$|) (|Kernel| |$|)))
          (|has| |$| (|Ring|)))
        ((|definingPolynomial| (|$| |$|)) (|has| |$| (|Ring|)))
        ((|even?|
          ((|Boolean|) |$|)) (|has| |$| (|RetractableTo| (|Integer|))))
        ((|odd?|
          ((|Boolean|) |$|)) (|has| |$| (|RetractableTo| (|Integer|))))))
       NIL
       (QUOTE (
        (|Boolean|)
        (|SparseUnivariatePolynomial| |$|)
        (|Kernel| |$|)
        (|BasicOperator|)
        (|List| (|BasicOperator|))
        (|List| (|Mapping| |$| (|List| |$|)))
        (|List| (|Mapping| |$| |$|))
        (|Symbol|)
        (|List| (|Symbol|))
        (|List| |$|)
        (|List| (|Kernel| |$|))
        (|NonNegativeInteger|)
        (|List| (|Equation| |$|))
        (|Equation| |$|)))
       NIL)))
    |ExpressionSpace|)
   (SETELT #1# 0 (QUOTE (|ExpressionSpace|)))))))

(setf (get (QUOTE |ExpressionSpace|) (QUOTE NILADIC)) T)
```

———-

21.17 ES-.lsp BOOTSTRAP

ES- depends on **ES**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **ES-** category which we can write into the **MID** directory. We compile the lisp code and copy the **ES-.o** file to the **OUT** directory. This is eventually forcibly

replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— ES-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |ES-;box;2S;1| (|x| |$|)
 (SPADCALL (LIST |x|) (QREFELT |$| 16)))

(DEFUN |ES-;paren;2S;2| (|x| |$|)
 (SPADCALL (LIST |x|) (QREFELT |$| 18)))

(DEFUN |ES-;belong?;BoB;3| (|op| |$|)
 (COND
  ((SPADCALL |op| (QREFELT |$| 13) (QREFELT |$| 21)) (QUOTE T))
  ((QUOTE T) (SPADCALL |op| (QREFELT |$| 14) (QREFELT |$| 21)))))

(DEFUN |ES-;listk| (|f| |$|)
 (SPADCALL (|ES-;allKernels| |f| |$|) (QREFELT |$| 25)))

(DEFUN |ES-;tower;SL;5| (|f| |$|)
 (SPADCALL (|ES-;listk| |f| |$|) (QREFELT |$| 26)))

(DEFUN |ES-;allk| (|l| |$|)
 (PROG (#1=#:G82361 |f| #2=#:G82362)
  (RETURN
   (SEQ
    (SPADCALL
     (ELT |$| 30)
     (PROGN
      (LETT #1# NIL |ES-;allk|)
      (SEQ
       (LETT |f| NIL |ES-;allk|)
       (LETT #2# |l| |ES-;allk|)
       G190
       (COND
        ((OR (ATOM #2#)
             (PROGN (LETT |f| (CAR #2#) |ES-;allk|) NIL))
         (GO G191)))
       (SEQ (EXIT (LETT #1# (CONS (|ES-;allKernels| |f| |$|) #1#) |ES-;allk|)))
       (LETT #2# (CDR #2#) |ES-;allk|) (GO G190) G191 (EXIT (NREVERSE0 #1#))))
     (SPADCALL NIL (QREFELT |$| 29))
     (QREFELT |$| 33))))))

(DEFUN |ES-;operators;SL;7| (|f| |$|)
 (PROG (#1=#:G82365 |k| #2=#:G82366)
  (RETURN
```

```
   (SEQ
    (PROGN
     (LETT #1# NIL |ES-;operators;SL;7|)
     (SEQ
      (LETT |k| NIL |ES-;operators;SL;7|)
      (LETT #2# (|ES-;listk| |f| |$|) |ES-;operators;SL;7|)
      G190
      (COND
       ((OR (ATOM #2#) (PROGN (LETT |k| (CAR #2#) |ES-;operators;SL;7|) NIL))
         (GO G191)))
      (SEQ
       (EXIT
        (LETT #1#
         (CONS (SPADCALL |k| (QREFELT |$| 35)) #1#) |ES-;operators;SL;7|)))
      (LETT #2# (CDR #2#) |ES-;operators;SL;7|)
      (GO G190)
      G191
      (EXIT (NREVERSE0 #1#))))))))

(DEFUN |ES-;height;SNni;8| (|f| |$|)
 (PROG (#1=#:G82371 |k| #2=#:G82372)
  (RETURN
   (SEQ
    (SPADCALL
     (ELT |$| 41)
     (PROGN
      (LETT #1# NIL |ES-;height;SNni;8|)
      (SEQ
       (LETT |k| NIL |ES-;height;SNni;8|)
       (LETT #2# (SPADCALL |f| (QREFELT |$| 38)) |ES-;height;SNni;8|)
       G190
       (COND
        ((OR (ATOM #2#) (PROGN (LETT |k| (CAR #2#) |ES-;height;SNni;8|) NIL))
          (GO G191)))
       (SEQ
        (EXIT
         (LETT #1#
          (CONS (SPADCALL |k| (QREFELT |$| 40)) #1#) |ES-;height;SNni;8|)))
       (LETT #2# (CDR #2#) |ES-;height;SNni;8|)
       (GO G190)
       G191
       (EXIT (NREVERSE0 #1#))))
     0
     (QREFELT |$| 44))))))

(DEFUN |ES-;freeOf?;SSB;9| (|x| |s| |$|)
 (PROG (#1=#:G82377 |k| #2=#:G82378)
  (RETURN
   (SEQ
    (COND
```

```
     ((SPADCALL |s|
       (PROGN
        (LETT #1# NIL |ES-;freeOf?;SSB;9|)
        (SEQ
         (LETT |k| NIL |ES-;freeOf?;SSB;9|)
         (LETT #2# (|ES-;listk| |x| |$|) |ES-;freeOf?;SSB;9|)
         G190
         (COND
          ((OR (ATOM #2#) (PROGN (LETT |k| (CAR #2#) |ES-;freeOf?;SSB;9|) NIL))
            (GO G191)))
         (SEQ
          (EXIT
           (LETT #1#
            (CONS (SPADCALL |k| (QREFELT |$| 46)) #1#) |ES-;freeOf?;SSB;9|)))
         (LETT #2# (CDR #2#) |ES-;freeOf?;SSB;9|)
         (GO G190)
         G191
         (EXIT (NREVERSE0 #1#))))
        (QREFELT |$| 48))
       (QUOTE NIL))
     ((QUOTE T) (QUOTE T)))))))

(DEFUN |ES-;distribute;2S;10| (|x| |$|)
 (PROG (#1=#:G82381 |k| #2=#:G82382)
  (RETURN
   (SEQ
    (|ES-;unwrap|
     (PROGN (LETT #1# NIL |ES-;distribute;2S;10|)
      (SEQ
       (LETT |k| NIL |ES-;distribute;2S;10|)
       (LETT #2# (|ES-;listk| |x| |$|) |ES-;distribute;2S;10|)
       G190
       (COND
        ((OR
          (ATOM #2#)
          (PROGN (LETT |k| (CAR #2#) |ES-;distribute;2S;10|) NIL))
           (GO G191)))
       (SEQ
        (EXIT
         (COND
          ((SPADCALL |k| (QREFELT |$| 13) (QREFELT |$| 50))
            (LETT #1# (CONS |k| #1#) |ES-;distribute;2S;10|)))))
       (LETT #2# (CDR #2#) |ES-;distribute;2S;10|)
       (GO G190)
       G191
       (EXIT (NREVERSE0 #1#))))
     |x|
     |$|)))))

(DEFUN |ES-;box;LS;11| (|l| |$|)
```

```
 (SPADCALL (QREFELT |$| 14) |l| (QREFELT |$| 52)))

(DEFUN |ES-;paren;LS;12| (|l| |$|)
 (SPADCALL (QREFELT |$| 13) |l| (QREFELT |$| 52)))

(DEFUN |ES-;freeOf?;2SB;13| (|x| |k| |$|)
 (COND
  ((SPADCALL
    (SPADCALL |k| (QREFELT |$| 56))
    (|ES-;listk| |x| |$|)
    (QREFELT |$| 57))
     (QUOTE NIL))
  ((QUOTE T) (QUOTE T))))

(DEFUN |ES-;kernel;Bo2S;14| (|op| |arg| |$|)
 (SPADCALL |op| (LIST |arg|) (QREFELT |$| 59)))

(DEFUN |ES-;elt;Bo2S;15| (|op| |x| |$|)
 (SPADCALL |op| (LIST |x|) (QREFELT |$| 52)))

(DEFUN |ES-;elt;Bo3S;16| (|op| |x| |y| |$|)
 (SPADCALL |op| (LIST |x| |y|) (QREFELT |$| 52)))

(DEFUN |ES-;elt;Bo4S;17| (|op| |x| |y| |z| |$|)
 (SPADCALL |op| (LIST |x| |y| |z|) (QREFELT |$| 52)))

(DEFUN |ES-;elt;Bo5S;18| (|op| |x| |y| |z| |t| |$|)
 (SPADCALL |op| (LIST |x| |y| |z| |t|) (QREFELT |$| 52)))

(DEFUN |ES-;eval;SSMS;19| (|x| |s| |f| |$|)
 (SPADCALL |x| (LIST |s|) (LIST |f|) (QREFELT |$| 67)))

(DEFUN |ES-;eval;SBoMS;20| (|x| |s| |f| |$|)
 (SPADCALL
  |x|
  (LIST (SPADCALL |s| (QREFELT |$| 69)))
  (LIST |f|)
  (QREFELT |$| 67)))

(DEFUN |ES-;eval;SSMS;21| (|x| |s| |f| |$|)
 (SPADCALL
  |x|
  (LIST |s|)
  (LIST (CONS (FUNCTION |ES-;eval;SSMS;21!0|) (VECTOR |f| |$|)))
  (QREFELT |$| 67)))

(DEFUN |ES-;eval;SSMS;21!0| (|#1| |$$|)
 (SPADCALL (SPADCALL |#1| (QREFELT (QREFELT |$$| 1) 72)) (QREFELT |$$| 0)))

(DEFUN |ES-;eval;SBoMS;22| (|x| |s| |f| |$|)
```

```
 (SPADCALL
  |x|
  (LIST |s|)
  (LIST (CONS (FUNCTION |ES-;eval;SBoMS;22!0|) (VECTOR |f| |$|)))
  (QREFELT |$| 75)))

(DEFUN |ES-;eval;SBoMS;22!0| (|#1| |$$|)
 (SPADCALL (SPADCALL |#1| (QREFELT (QREFELT |$$| 1) 72)) (QREFELT |$$| 0)))

(DEFUN |ES-;subst;SES;23| (|x| |e| |$|)
 (SPADCALL |x| (LIST |e|) (QREFELT |$| 78)))

(DEFUN |ES-;eval;SLLS;24| (|x| |ls| |lf| |$|)
 (PROG (#1=#:G82403 |f| #2=#:G82404)
  (RETURN
   (SEQ
    (SPADCALL
     |x|
     |ls|
     (PROGN
      (LETT #1# NIL |ES-;eval;SLLS;24|)
      (SEQ
       (LETT |f| NIL |ES-;eval;SLLS;24|)
       (LETT #2# |lf| |ES-;eval;SLLS;24|)
       G190
       (COND
        ((OR (ATOM #2#) (PROGN (LETT |f| (CAR #2#) |ES-;eval;SLLS;24|) NIL))
          (GO G191)))
       (SEQ
        (EXIT
         (LETT #1#
          (CONS (CONS (FUNCTION |ES-;eval;SLLS;24!0|) (VECTOR |f| |$|)) #1#)
          |ES-;eval;SLLS;24|)))
       (LETT #2# (CDR #2#) |ES-;eval;SLLS;24|)
       (GO G190)
       G191
       (EXIT (NREVERSE0 #1#))))
     (QREFELT |$| 75))))))

(DEFUN |ES-;eval;SLLS;24!0| (|#1| |$$|)
 (SPADCALL (SPADCALL |#1| (QREFELT (QREFELT |$$| 1) 72)) (QREFELT |$$| 0)))

(DEFUN |ES-;eval;SLLS;25| (|x| |ls| |lf| |$|)
 (PROG (#1=#:G82407 |f| #2=#:G82408)
  (RETURN
   (SEQ
    (SPADCALL
     |x|
     |ls|
     (PROGN
```

```
      (LETT #1# NIL |ES-;eval;SLLS;25|)
      (SEQ
       (LETT |f| NIL |ES-;eval;SLLS;25|)
       (LETT #2# |lf| |ES-;eval;SLLS;25|)
       G190
       (COND
        ((OR (ATOM #2#) (PROGN (LETT |f| (CAR #2#) |ES-;eval;SLLS;25|) NIL))
          (GO G191)))
       (SEQ
        (EXIT
         (LETT #1#
          (CONS (CONS (FUNCTION |ES-;eval;SLLS;25!0|) (VECTOR |f| |$|)) #1#)
          |ES-;eval;SLLS;25|)))
       (LETT #2# (CDR #2#) |ES-;eval;SLLS;25|)
       (GO G190)
       G191
       (EXIT (NREVERSE0 #1#))))
     (QREFELT |$| 67))))))

(DEFUN |ES-;eval;SLLS;25!0| (|#1| |$$|)
 (SPADCALL (SPADCALL |#1| (QREFELT (QREFELT |$$| 1) 72)) (QREFELT |$$| 0)))

(DEFUN |ES-;eval;SLLS;26| (|x| |ls| |lf| |$|)
 (PROG (#1=#:G82412 |s| #2=#:G82413)
  (RETURN
   (SEQ
    (SPADCALL
     |x|
     (PROGN
      (LETT #1# NIL |ES-;eval;SLLS;26|)
      (SEQ
       (LETT |s| NIL |ES-;eval;SLLS;26|)
       (LETT #2# |ls| |ES-;eval;SLLS;26|)
       G190
       (COND
        ((OR (ATOM #2#) (PROGN (LETT |s| (CAR #2#) |ES-;eval;SLLS;26|) NIL))
          (GO G191)))
       (SEQ
        (EXIT
         (LETT #1#
          (CONS (SPADCALL |s| (QREFELT |$| 69)) #1#) |ES-;eval;SLLS;26|)))
       (LETT #2# (CDR #2#) |ES-;eval;SLLS;26|)
       (GO G190)
       G191
       (EXIT (NREVERSE0 #1#))))
     |lf|
     (QREFELT |$| 67))))))

(DEFUN |ES-;map;MKS;27| (|fn| |k| |$|)
 (PROG (#1=#:G82428 |x| #2=#:G82429 |l|)
```

```
  (RETURN
   (SEQ
    (COND
     ((SPADCALL
       (LETT |l|
        (PROGN
         (LETT #1# NIL |ES-;map;MKS;27|)
         (SEQ
          (LETT |x| NIL |ES-;map;MKS;27|)
          (LETT #2# (SPADCALL |k| (QREFELT |$| 85)) |ES-;map;MKS;27|)
          G190
          (COND
           ((OR (ATOM #2#) (PROGN (LETT |x| (CAR #2#) |ES-;map;MKS;27|) NIL))
             (GO G191)))
          (SEQ
           (EXIT
            (LETT #1# (CONS (SPADCALL |x| |fn|) #1#) |ES-;map;MKS;27|)))
          (LETT #2# (CDR #2#) |ES-;map;MKS;27|)
          (GO G190)
          G191
          (EXIT (NREVERSE0 #1#))))
        |ES-;map;MKS;27|)
       (SPADCALL |k| (QREFELT |$| 85)) (QREFELT |$| 86))
      (SPADCALL |k| (QREFELT |$| 87)))
     ((QUOTE T)
       (SPADCALL (SPADCALL |k| (QREFELT |$| 35)) |l| (QREFELT |$| 52)))))))

(DEFUN |ES-;operator;2Bo;28| (|op| |$|)
 (COND
  ((SPADCALL |op| (SPADCALL "%paren" (QREFELT |$| 9)) (QREFELT |$| 89))
    (QREFELT |$| 13))
  ((SPADCALL |op| (SPADCALL "%box" (QREFELT |$| 9)) (QREFELT |$| 89))
    (QREFELT |$| 14))
  ((QUOTE T) (|error| "Unknown operator"))))

(DEFUN |ES-;mainKernel;SU;29| (|x| |$|)
 (PROG (|l| |kk| #1=#:G82445 |n| |k|)
  (RETURN
   (SEQ
    (COND
     ((NULL (LETT |l| (SPADCALL |x| (QREFELT |$| 38)) |ES-;mainKernel;SU;29|))
       (CONS 1 "failed"))
     ((QUOTE T)
      (SEQ
       (LETT |n|
        (SPADCALL
         (LETT |k| (|SPADfirst| |l|) |ES-;mainKernel;SU;29|) (QREFELT |$| 40))
        |ES-;mainKernel;SU;29|)
       (SEQ
        (LETT |kk| NIL |ES-;mainKernel;SU;29|)
```

```
        (LETT #1# (CDR |l|) |ES-;mainKernel;SU;29|)
        G190
        (COND
         ((OR
           (ATOM #1#)
           (PROGN (LETT |kk| (CAR #1#) |ES-;mainKernel;SU;29|) NIL))
            (GO G191)))
        (SEQ
         (EXIT
          (COND
           ((|<| |n| (SPADCALL |kk| (QREFELT |$| 40)))
            (SEQ
             (LETT |n| (SPADCALL |kk| (QREFELT |$| 40)) |ES-;mainKernel;SU;29|)
             (EXIT (LETT |k| |kk| |ES-;mainKernel;SU;29|)))))))
        (LETT #1# (CDR #1#) |ES-;mainKernel;SU;29|) (GO G190) G191 (EXIT NIL))
       (EXIT (CONS 0 |k|)))))))))

(DEFUN |ES-;allKernels| (|f| |$|)
 (PROG (|l| |k| #1=#:G82458 |u| |s0| |n| |arg| |t| |s|)
  (RETURN
   (SEQ
    (LETT |s|
     (SPADCALL
      (LETT |l|
       (SPADCALL |f| (QREFELT |$| 38))
       |ES-;allKernels|)
      (QREFELT |$| 29))
     |ES-;allKernels|)
    (SEQ
     (LETT |k| NIL |ES-;allKernels|)
     (LETT #1# |l| |ES-;allKernels|)
     G190
     (COND
      ((OR (ATOM #1#) (PROGN (LETT |k| (CAR #1#) |ES-;allKernels|) NIL))
       (GO G191)))
     (SEQ
      (LETT |t|
       (SEQ
        (LETT |u|
         (SPADCALL
          (SPADCALL |k| (QREFELT |$| 35))
           "%dummyVar"
           (QREFELT |$| 94))
          |ES-;allKernels|)
        (EXIT
         (COND
          ((QEQCAR |u| 0)
           (SEQ
            (LETT |arg| (SPADCALL |k| (QREFELT |$| 85)) |ES-;allKernels|)
            (LETT |s0|
```

```
             (SPADCALL
              (SPADCALL
               (SPADCALL |arg| (QREFELT |$| 95))
               (QREFELT |$| 56))
              (|ES-;allKernels| (|SPADfirst| |arg|) |$|)
              (QREFELT |$| 96))
             |ES-;allKernels|)
            (LETT |arg| (CDR (CDR |arg|)) |ES-;allKernels|)
            (LETT |n| (QCDR |u|) |ES-;allKernels|)
            (COND ((|<| 1 |n|) (LETT |arg| (CDR |arg|) |ES-;allKernels|)))
            (EXIT (SPADCALL |s0| (|ES-;allk| |arg| |$|) (QREFELT |$| 30)))))
          ((QUOTE T) (|ES-;allk| (SPADCALL |k| (QREFELT |$| 85)) |$|)))))
       |ES-;allKernels|)
      (EXIT
       (LETT |s| (SPADCALL |s| |t| (QREFELT |$| 30)) |ES-;allKernels|)))
     (LETT #1# (CDR #1#) |ES-;allKernels|)
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT |s|)))))

(DEFUN |ES-;kernel;BoLS;31| (|op| |args| |$|)
 (COND
  ((NULL (SPADCALL |op| (QREFELT |$| 97))) (|error| "Unknown operator"))
  ((QUOTE T) (|ES-;okkernel| |op| |args| |$|))))

(DEFUN |ES-;okkernel| (|op| |l| |$|)
 (PROG (#1=#:G82465 |f| #2=#:G82466)
  (RETURN
   (SEQ
    (SPADCALL
     (SPADCALL |op| |l|
      (|+| 1
       (SPADCALL
        (ELT |$| 41)
        (PROGN
         (LETT #1# NIL |ES-;okkernel|)
         (SEQ
          (LETT |f| NIL |ES-;okkernel|)
          (LETT #2# |l| |ES-;okkernel|)
          G190
          (COND
           ((OR (ATOM #2#) (PROGN (LETT |f| (CAR #2#) |ES-;okkernel|) NIL))
             (GO G191)))
          (SEQ
           (EXIT
            (LETT #1#
             (CONS (SPADCALL |f| (QREFELT |$| 99)) #1#) |ES-;okkernel|)))
          (LETT #2# (CDR #2#) |ES-;okkernel|)
          (GO G190)
```

```
          G191
          (EXIT (NREVERSE0 #1#))))
        0
        (QREFELT |$| 44)))
      (QREFELT |$| 100))
     (QREFELT |$| 87))))))

(DEFUN |ES-;elt;BoLS;33| (|op| |args| |$|)
 (PROG (|u| #1=#:G82482 |v|)
  (RETURN
   (SEQ
    (EXIT
     (COND
      ((NULL (SPADCALL |op| (QREFELT |$| 97))) (|error| "Unknown operator"))
      ((QUOTE T)
       (SEQ
        (SEQ
         (LETT |u| (SPADCALL |op| (QREFELT |$| 102)) |ES-;elt;BoLS;33|)
         (EXIT
          (COND
           ((QEQCAR |u| 0)
            (COND
             ((NULL (EQL (LENGTH |args|) (QCDR |u|)))
              (PROGN
               (LETT #1#
                (|error| "Wrong number of arguments")
                |ES-;elt;BoLS;33|)
                (GO #1#))))))))
        (LETT |v| (SPADCALL |op| |args| (QREFELT |$| 105)) |ES-;elt;BoLS;33|)
        (EXIT
         (COND
          ((QEQCAR |v| 0) (QCDR |v|))
          ((QUOTE T) (|ES-;okkernel| |op| |args| |$|)))))))))
    #1#
    (EXIT #1#)))))

(DEFUN |ES-;retract;SK;34| (|f| |$|)
 (PROG (|k|)
  (RETURN
   (SEQ
    (LETT |k| (SPADCALL |f| (QREFELT |$| 107)) |ES-;retract;SK;34|)
    (EXIT
     (COND
      ((OR
        (QEQCAR |k| 1)
        (NULL
         (SPADCALL
          (SPADCALL (QCDR |k|) (QREFELT |$| 87))
          |f|
          (QREFELT |$| 108))))
```

```
        (|error| "not a kernel"))
      ((QUOTE T) (QCDR |k|)))))))))

(DEFUN |ES-;retractIfCan;SU;35| (|f| |$|)
 (PROG (|k|)
  (RETURN
   (SEQ
    (LETT |k| (SPADCALL |f| (QREFELT |$| 107)) |ES-;retractIfCan;SU;35|)
    (EXIT
     (COND
      ((OR
        (QEQCAR |k| 1)
        (NULL
         (SPADCALL
          (SPADCALL (QCDR |k|) (QREFELT |$| 87))
          |f|
          (QREFELT |$| 108))))
        (CONS 1 "failed"))
       ((QUOTE T) |k|))))))))

(DEFUN |ES-;is?;SSB;36| (|f| |s| |$|)
 (PROG (|k|)
  (RETURN
   (SEQ
    (LETT |k| (SPADCALL |f| (QREFELT |$| 111)) |ES-;is?;SSB;36|)
    (EXIT
     (COND
      ((QEQCAR |k| 1) (QUOTE NIL))
      ((QUOTE T) (SPADCALL (QCDR |k|) |s| (QREFELT |$| 112)))))))))

(DEFUN |ES-;is?;SBoB;37| (|f| |op| |$|)
 (PROG (|k|)
  (RETURN
   (SEQ
    (LETT |k| (SPADCALL |f| (QREFELT |$| 111)) |ES-;is?;SBoB;37|)
    (EXIT
     (COND
      ((QEQCAR |k| 1) (QUOTE NIL))
      ((QUOTE T) (SPADCALL (QCDR |k|) |op| (QREFELT |$| 50)))))))))

(DEFUN |ES-;unwrap| (|l| |x| |$|)
 (PROG (|k| #1=#:G82507)
  (RETURN
   (SEQ
    (SEQ
     (LETT |k| NIL |ES-;unwrap|)
     (LETT #1# (NREVERSE |l|) |ES-;unwrap|)
     G190
     (COND
      ((OR (ATOM #1#) (PROGN (LETT |k| (CAR #1#) |ES-;unwrap|) NIL))
```

```
        (GO G191)))
      (SEQ
       (EXIT
        (LETT |x|
         (SPADCALL |x| |k|
          (|SPADfirst| (SPADCALL |k| (QREFELT |$| 85)))
          (QREFELT |$| 115))
         |ES-;unwrap|)))
      (LETT #1# (CDR #1#) |ES-;unwrap|)
      (GO G190)
      G191
      (EXIT NIL))
     (EXIT |x|)))))

(DEFUN |ES-;distribute;3S;39| (|x| |y| |$|)
 (PROG (|ky| #1=#:G82512 |k| #2=#:G82513)
  (RETURN
   (SEQ
    (LETT |ky| (SPADCALL |y| (QREFELT |$| 56)) |ES-;distribute;3S;39|)
    (EXIT
     (|ES-;unwrap|
      (PROGN
       (LETT #1# NIL |ES-;distribute;3S;39|)
       (SEQ
        (LETT |k| NIL |ES-;distribute;3S;39|)
        (LETT #2# (|ES-;listk| |x| |$|) |ES-;distribute;3S;39|)
        G190
        (COND
         ((OR
           (ATOM #2#)
           (PROGN (LETT |k| (CAR #2#) |ES-;distribute;3S;39|) NIL))
          (GO G191)))
        (SEQ
         (EXIT
          (COND
           ((COND
             ((SPADCALL |k|
               (SPADCALL "%paren" (QREFELT |$| 9))
                (QREFELT |$| 112))
               (SPADCALL |ky|
                (|ES-;listk| (SPADCALL |k| (QREFELT |$| 87)) |$|)
                (QREFELT |$| 57)))
              ((QUOTE T) (QUOTE NIL)))
             (LETT #1# (CONS |k| #1#) |ES-;distribute;3S;39|)))))
        (LETT #2# (CDR #2#) |ES-;distribute;3S;39|)
        (GO G190)
        G191
        (EXIT (NREVERSE0 #1#))))
      |x|
      |$|))))))
```

```
(DEFUN |ES-;eval;SLS;40| (|f| |leq| |$|)
 (PROG (|rec|)
  (RETURN
   (SEQ
    (LETT |rec| (|ES-;mkKerLists| |leq| |$|) |ES-;eval;SLS;40|)
    (EXIT (SPADCALL |f| (QCAR |rec|) (QCDR |rec|) (QREFELT |$| 117)))))))

(DEFUN |ES-;subst;SLS;41| (|f| |leq| |$|)
 (PROG (|rec|)
  (RETURN
   (SEQ
    (LETT |rec| (|ES-;mkKerLists| |leq| |$|) |ES-;subst;SLS;41|)
    (EXIT (SPADCALL |f| (QCAR |rec|) (QCDR |rec|) (QREFELT |$| 119)))))))

(DEFUN |ES-;mkKerLists| (|leq| |$|)
 (PROG (|eq| #1=#:G82530 |k| |lk| |lv|)
  (RETURN
   (SEQ
    (LETT |lk| NIL |ES-;mkKerLists|)
    (LETT |lv| NIL |ES-;mkKerLists|)
    (SEQ
     (LETT |eq| NIL |ES-;mkKerLists|)
     (LETT #1# |leq| |ES-;mkKerLists|)
     G190
     (COND
      ((OR (ATOM #1#) (PROGN (LETT |eq| (CAR #1#) |ES-;mkKerLists|) NIL))
       (GO G191)))
     (SEQ
      (LETT |k|
       (SPADCALL (SPADCALL |eq| (QREFELT |$| 122)) (QREFELT |$| 111))
       |ES-;mkKerLists|)
      (EXIT
       (COND
        ((QEQCAR |k| 1) (|error| "left hand side must be a single kernel"))
        ((NULL (SPADCALL (QCDR |k|) |lk| (QREFELT |$| 57)))
         (SEQ
          (LETT |lk| (CONS (QCDR |k|) |lk|) |ES-;mkKerLists|)
          (EXIT
           (LETT |lv|
            (CONS (SPADCALL |eq| (QREFELT |$| 123)) |lv|)
            |ES-;mkKerLists|)))))))
     (LETT #1# (CDR #1#) |ES-;mkKerLists|)
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT (CONS |lk| |lv|))))))

(DEFUN |ES-;even?;SB;43| (|x| |$|)
 (|ES-;intpred?| |x| (ELT |$| 125) |$|))
```

```
(DEFUN |ES-;odd?;SB;44| (|x| |$|)
 (|ES-;intpred?| |x| (ELT |$| 127) |$|))

(DEFUN |ES-;intpred?| (|x| |pred?| |$|)
 (PROG (|u|)
  (RETURN
   (SEQ
    (LETT |u| (SPADCALL |x| (QREFELT |$| 130)) |ES-;intpred?|)
    (EXIT
     (COND
      ((QEQCAR |u| 0) (SPADCALL (QCDR |u|) |pred?|))
      ((QUOTE T) (QUOTE NIL))))))))

(DEFUN |ExpressionSpace&| (|#1|)
 (PROG (|DV$1| |dv$| |$| |pv$|)
  (RETURN
   (PROGN
    (LETT |DV$1| (|devaluate| |#1|) . #1=(|ExpressionSpace&|))
    (LETT |dv$| (LIST (QUOTE |ExpressionSpace&|) |DV$1|) . #1#)
    (LETT |$| (make-array 131) . #1#)
    (QSETREFV |$| 0 |dv$|)
    (QSETREFV |$| 3
     (LETT |pv$|
      (|buildPredVector| 0 0
       (LIST
        (|HasCategory| |#1| (QUOTE (|RetractableTo| (|Integer|))))
        (|HasCategory| |#1| (QUOTE (|Ring|))))) . #1#))
    (|stuffDomainSlots| |$|)
    (QSETREFV |$| 6 |#1|)
    (QSETREFV |$| 13
     (SPADCALL (SPADCALL "%paren" (QREFELT |$| 9)) (QREFELT |$| 12)))
    (QSETREFV |$| 14
     (SPADCALL (SPADCALL "%box" (QREFELT |$| 9)) (QREFELT |$| 12)))
    (COND
     ((|testBitVector| |pv$| 1)
      (PROGN
       (QSETREFV |$| 126 (CONS (|dispatchFunction| |ES-;even?;SB;43|) |$|))
       (QSETREFV |$| 128 (CONS (|dispatchFunction| |ES-;odd?;SB;44|) |$|)))))
    |$|))))

(setf (get
 (QUOTE |ExpressionSpace&|)
 (QUOTE |infovec|))
 (LIST
  (QUOTE #(
   NIL NIL NIL NIL NIL NIL (|local| |#1|) (|String|) (|Symbol|)
   (0 . |coerce|) (|BasicOperator|) (|CommonOperators|) (5 . |operator|)
   (QUOTE |oppren|) (QUOTE |opbox|) (|List| |$|) (10 . |box|)
   |ES-;box;2S;1| (15 . |paren|) |ES-;paren;2S;2| (|Boolean|)
```

```
 (20 . |=|) |ES-;belong?;BoB;3| (|List| 34) (|Set| 34)
 (26 . |parts|) (31 . |sort!|) (|List| 55) |ES-;tower;SL;5|
 (36 . |brace|) (41 . |union|) (|Mapping| 24 24 24) (|List| 24)
 (47 . |reduce|) (|Kernel| 6) (54 . |operator|) (|List| 10)
 |ES-;operators;SL;7| (59 . |kernels|) (|NonNegativeInteger|)
 (64 . |height|) (69 . |max|) (|Mapping| 39 39 39) (|List| 39)
 (75 . |reduce|) |ES-;height;SNni;8| (82 . |name|) (|List| 8)
 (87 . |member?|) |ES-;freeOf?;SSB;9| (93 . |is?|)
 |ES-;distribute;2S;10| (99 . |elt|) |ES-;box;LS;11|
 |ES-;paren;LS;12| (|Kernel| |$|) (105 . |retract|)
 (110 . |member?|) |ES-;freeOf?;2SB;13| (116 . |kernel|)
 |ES-;kernel;Bo2S;14| |ES-;elt;Bo2S;15| |ES-;elt;Bo3S;16|
 |ES-;elt;Bo4S;17| |ES-;elt;Bo5S;18| (|Mapping| |$| 15)
 (|List| 65) (122 . |eval|) |ES-;eval;SSMS;19| (129 . |name|)
 |ES-;eval;SBoMS;20| (|List| 6) (134 . |first|)
 (|Mapping| |$| |$|) |ES-;eval;SSMS;21| (139 . |eval|)
 |ES-;eval;SBoMS;22| (|List| 79) (146 . |subst|) (|Equation| |$|)
 |ES-;subst;SES;23| (|List| 73) |ES-;eval;SLLS;24|
 |ES-;eval;SLLS;25| |ES-;eval;SLLS;26| (152 . |argument|)
 (157 . |=|) (163 . |coerce|) |ES-;map;MKS;27| (168 . |is?|)
 |ES-;operator;2Bo;28| (|Union| 55 (QUOTE "failed"))
 |ES-;mainKernel;SU;29| (|Union| (|None|) (QUOTE "failed"))
 (174 . |property|) (180 . |second|) (185 . |remove!|)
 (191 . |belong?|) |ES-;kernel;BoLS;31| (196 . |height|)
 (201 . |kernel|) (|Union| 39 (QUOTE "failed")) (208 . |arity|)
 (|Union| 6 (QUOTE "failed")) (|BasicOperatorFunctions1| 6)
 (213 . |evaluate|) |ES-;elt;BoLS;33| (219 . |mainKernel|)
 (224 . |=|) |ES-;retract;SK;34| |ES-;retractIfCan;SU;35|
 (230 . |retractIfCan|) (235 . |is?|) |ES-;is?;SSB;36|
 |ES-;is?;SBoB;37| (241 . |eval|) |ES-;distribute;3S;39|
 (248 . |eval|) |ES-;eval;SLS;40| (255 . |subst|)
 |ES-;subst;SLS;41| (|Equation| 6) (262 . |lhs|) (267 . |rhs|)
 (|Integer|) (272 . |even?|) (277 . |even?|) (282 . |odd?|)
 (287 . |odd?|) (|Union| 124 (QUOTE "failed")) (292 . |retractIfCan|)))
(QUOTE #(
 |tower| 297 |subst| 302 |retractIfCan| 314 |retract| 319 |paren| 324
 |operators| 334 |operator| 339 |odd?| 344 |map| 349 |mainKernel| 355
 |kernel| 360 |is?| 372 |height| 384 |freeOf?| 389 |even?| 401 |eval| 406
 |elt| 461 |distribute| 497 |box| 508 |belong?| 518))
(QUOTE NIL)
(CONS
 (|makeByteWordVec2| 1 (QUOTE NIL))
 (CONS
  (QUOTE #())
  (CONS
   (QUOTE #())
   (|makeByteWordVec2| 130 (QUOTE
    (1 8 0 7 9 1 11 10 8 12 1 6 0 15 16 1 6 0 15 18 2 10 20 0 0 21 1 24
     23 0 25 1 23 0 0 26 1 24 0 23 29 2 24 0 0 0 30 3 32 24 31 0 24 33
     1 34 10 0 35 1 6 27 0 38 1 34 39 0 40 2 39 0 0 0 41 3 43 39 42 0
```

```
      39 44 1 34 8 0 46 2 47 20 8 0 48 2 34 20 0 10 50 2 6 0 10 15 52 1
      6 55 0 56 2 23 20 34 0 57 2 6 0 10 15 59 3 6 0 0 47 66 67 1 10 8
      0 69 1 71 6 0 72 3 6 0 0 36 66 75 2 6 0 0 77 78 1 34 71 0 85 2 71
      20 0 0 86 1 6 0 55 87 2 10 20 0 8 89 2 10 93 0 7 94 1 71 6 0 95
      2 24 0 34 0 96 1 6 20 10 97 1 6 39 0 99 3 34 0 10 71 39 100 1 10
      101 0 102 2 104 103 10 71 105 1 6 91 0 107 2 6 20 0 0 108 1 6 91
      0 111 2 34 20 0 8 112 3 6 0 0 55 0 115 3 6 0 0 27 15 117 3 6 0 0
      27 15 119 1 121 6 0 122 1 121 6 0 123 1 124 20 0 125 1 0 20 0 126
      1 124 20 0 127 1 0 20 0 128 1 6 129 0 130 1 0 27 0 28 2 0 0 0 77
      120 2 0 0 0 79 80 1 0 91 0 110 1 0 55 0 109 1 0 0 0 19 1 0 0 15
      54 1 0 36 0 37 1 0 10 10 90 1 0 20 0 128 2 0 0 73 55 88 1 0 91 0
      92 2 0 0 10 15 98 2 0 0 10 0 60 2 0 20 0 8 113 2 0 20 0 10 114 1
      0 39 0 45 2 0 20 0 8 49 2 0 20 0 0 58 1 0 20 0 126 3 0 0 0 10 73
      76 3 0 0 0 36 66 84 3 0 0 0 10 65 70 3 0 0 0 36 81 82 3 0 0 0 8
      65 68 3 0 0 0 8 73 74 3 0 0 0 47 81 83 2 0 0 0 77 118 2 0 0 10
      15 106 5 0 0 10 0 0 0 0 64 3 0 0 10 0 0 62 4 0 0 10 0 0 0 63 2
      0 0 10 0 61 2 0 0 0 0 116 1 0 0 0 51 1 0 0 15 53 1 0 0 0 17 1 0
      20 10 22))))))
  (QUOTE |lookupComplete|)))
```

———-

21.18 EUCDOM.lsp BOOTSTRAP

EUCDOM depends on **INT** which depends on **EUCDOM**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **EUCDOM** category which we can write into the **MID** directory. We compile the lisp code and copy the **EUCDOM.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

21.18.1 The Lisp Implementation

EUCDOM;VersionCheck

This implements the bootstrap code for **EuclideanDomain**. The call to **VERSIONCHECK** is a legacy check to ensure that we did not load algebra code from a previous system version (which would not run due to major surgical changes in the system) without recompiling.

— **EUCDOM;VersionCheck** —

```
(|/VERSIONCHECK| 2)
```

———-

The Domain Cache Variable

We create a variable which is formed by concatenating the string "**;AL**" to the domain name forming, in this case, "**EuclideanDomain;AL**". The variable has the initial value at load time of a list of one element, **NIL**. This list is a data structure that will be modified to hold an executable function. This function is created the first time the domain is used which it replaces the **NIL**.

— **EuclideanDomain;AL** —

```
(SETQ |EuclideanDomain;AL| (QUOTE NIL))
```

———-

The Domain Function

When you call a domain the code is pretty simple at the top level. This code will check to see if this domain has ever been used. It does this by checking the value of the cached domain variable (which is the domain name **EuclideanDomain** concatenated with the string "**;AL**" to form the cache variable name which is **EuclideanDomain;AL**).

If this value is NIL we have never executed this function before. If it is not NIL we have executed this function before and we need only return the cached function which was stored in the cache variable.

If this is the first time this function is called, the cache variable is NIL and we execute the other branch of the conditional. This calls a function which

1. creates a procedure
2. returns the procedure as a value.

This procedure replaces the cached variable **EuclideanDomain;AL** value so it will be non-NIL the second time this domain is used. Thus the work of building the domain only happens once.

If this function has never been called before we call the

— **EuclideanDomain** —

```
(DEFUN |EuclideanDomain| NIL
  (LET (#:G83585)
    (COND
      (|EuclideanDomain;AL|)
      (T (SETQ |EuclideanDomain;AL| (|EuclideanDomain;|))))))
```

———-

The First Call Domain Function

— **EuclideanDomain;** —

```
(DEFUN |EuclideanDomain;| NIL
  (PROG (#1=#:G83583)
    (RETURN
      (PROG1
        (LETT #1#
          (|Join|
            (|PrincipalIdealDomain|)
            (|mkCategory|
              (QUOTE |domain|)
              (QUOTE (
                ((|sizeLess?| ((|Boolean|) |$| |$|)) T)
                ((|euclideanSize| ((|NonNegativeInteger|) |$|)) T)
                ((|divide|
                  ((|Record|
                    (|:| |quotient| |$|)
                    (|:| |remainder| |$|))
                  |$| |$|)) T)
                ((|quo| (|$| |$| |$|)) T)
                ((|rem| (|$| |$| |$|)) T)
                ((|extendedEuclidean|
                  ((|Record|
                    (|:| |coef1| |$|)
                    (|:| |coef2| |$|)
                    (|:| |generator| |$|))
                  |$| |$|)) T)
                ((|extendedEuclidean|
                  ((|Union|
                      (|Record| (|:| |coef1| |$|) (|:| |coef2| |$|))
                      "failed")
                    |$| |$| |$|)) T)
                ((|multiEuclidean|
                  ((|Union|
                      (|List| |$|)
                      "failed")
                   (|List| |$|) |$|)) T)))
              NIL
              (QUOTE ((|List| |$|) (|NonNegativeInteger|) (|Boolean|)))
              NIL))
            |EuclideanDomain|)
        (SETELT #1# 0 (QUOTE (|EuclideanDomain|)))))))
```

———-

EUCDOM;NILADIC

— **EUCDOM;NILADIC** —

```
(setf (get (QUOTE |EuclideanDomain|) (QUOTE NILADIC)) T)
```

————-

— **EUCDOM.lsp BOOTSTRAP** —

```
\getchunk{EUCDOM;VersionCheck}
\getchunk{EuclideanDomain;AL}
\getchunk{EuclideanDomain}
\getchunk{EuclideanDomain;}
\getchunk{EUCDOM;NILADIC}
```

————-

21.19 EUCDOM-.lsp BOOTSTRAP

EUCDOM- depends on **EUCDOM**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **EUCDOM-** category which we can write into the **MID** directory. We compile the lisp code and copy the **EUCDOM-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

21.19.1 The Lisp Implementation

EUCDOM-;VersionCheck

This implements the bootstrap code for **EuclideanDomain**. The call to **VERSIONCHECK** is a legacy check to ensure that we did not load algebra code from a previous system version (which would not run due to major surgical changes in the system) without recompiling.

— **EUCDOM-;VersionCheck** —

```
(|/VERSIONCHECK| 2)
```

————-

EUCDOM-;sizeLess?;2SB;1

— **EUCDOM-;sizeLess?;2SB;1** —

```
(DEFUN |EUCDOM-;sizeLess?;2SB;1| (|x| |y| $)
 (COND
  ((SPADCALL |y| (QREFELT $ 8)) (QUOTE NIL))
  ((SPADCALL |x| (QREFELT $ 8)) (QUOTE T))
  ((QUOTE T)
   (< (SPADCALL |x| (QREFELT $ 10)) (SPADCALL |y| (QREFELT $ 10))))))
```

————-

EUCDOM-;quo;3S;2

— **EUCDOM-;quo;3S;2** —

```
(DEFUN |EUCDOM-;quo;3S;2| (|x| |y| $)
 (QCAR (SPADCALL |x| |y| (QREFELT $ 13))))
```

————-

EUCDOM-;rem;3S;3

— **EUCDOM-;rem;3S;3** —

```
(DEFUN |EUCDOM-;rem;3S;3| (|x| |y| $)
 (QCDR (SPADCALL |x| |y| (QREFELT $ 13))))
```

————-

EUCDOM-;exquo;2SU;4

— **EUCDOM-;exquo;2SU;4** —

```
(DEFUN |EUCDOM-;exquo;2SU;4| (|x| |y| $)
 (PROG (|qr|)
  (RETURN
   (SEQ
```

```
    (COND
     ((SPADCALL |x| (QREFELT $ 8)) (CONS 0 (|spadConstant| $ 16)))
     ((SPADCALL |y| (QREFELT $ 8)) (CONS 1 "failed"))
     ((QUOTE T)
      (SEQ
       (LETT |qr|
        (SPADCALL |x| |y| (QREFELT $ 13))
        |EUCDOM-;exquo;2SU;4|)
       (EXIT
        (COND
         ((SPADCALL (QCDR |qr|) (QREFELT $ 8)) (CONS 0 (QCAR |qr|)))
         ((QUOTE T) (CONS 1 "failed")))))))))
```

———-

EUCDOM-;gcd;3S;5

— **EUCDOM-;gcd;3S;5** —

```
(DEFUN |EUCDOM-;gcd;3S;5| (|x| |y| $)
 (PROG (|#G13| |#G14|)
  (RETURN
   (SEQ
    (LETT |x| (SPADCALL |x| (QREFELT $ 19)) |EUCDOM-;gcd;3S;5|)
    (LETT |y| (SPADCALL |y| (QREFELT $ 19)) |EUCDOM-;gcd;3S;5|)
    (SEQ G190
     (COND
      ((NULL
        (COND
         ((SPADCALL |y| (QREFELT $ 8)) (QUOTE NIL))
         ((QUOTE T) (QUOTE T))))
       (GO G191)))
     (SEQ
      (PROGN
       (LETT |#G13| |y| |EUCDOM-;gcd;3S;5|)
       (LETT |#G14| (SPADCALL |x| |y| (QREFELT $ 20)) |EUCDOM-;gcd;3S;5|)
       (LETT |x| |#G13| |EUCDOM-;gcd;3S;5|)
       (LETT |y| |#G14| |EUCDOM-;gcd;3S;5|))
      (EXIT
       (LETT |y| (SPADCALL |y| (QREFELT $ 19)) |EUCDOM-;gcd;3S;5|)))
     NIL
     (GO G190)
    G191
    (EXIT NIL))
   (EXIT |x|)))))
```

———-

EUCDOM-;unitNormalizeIdealElt

— **EUCDOM-;unitNormalizeIdealElt** —

```
(DEFUN |EUCDOM-;unitNormalizeIdealElt| (|s| $)
 (PROG (|#G16| |u| |c| |a|)
  (RETURN
   (SEQ
    (PROGN
     (LETT |#G16|
       (SPADCALL (QVELT |s| 2) (QREFELT $ 23))
       |EUCDOM-;unitNormalizeIdealElt|)
     (LETT |u| (QVELT |#G16| 0) |EUCDOM-;unitNormalizeIdealElt|)
     (LETT |c| (QVELT |#G16| 1) |EUCDOM-;unitNormalizeIdealElt|)
     (LETT |a| (QVELT |#G16| 2) |EUCDOM-;unitNormalizeIdealElt|)
     |#G16|)
    (EXIT
     (COND
      ((SPADCALL |a| (|spadConstant| $ 24) (QREFELT $ 25)) |s|)
      ((QUOTE T)
       (VECTOR
        (SPADCALL |a| (QVELT |s| 0) (QREFELT $ 26))
        (SPADCALL |a| (QVELT |s| 1) (QREFELT $ 26))
        |c|))))))))
```

———-

EUCDOM-;extendedEuclidean;2SR;7

— **EUCDOM-;extendedEuclidean;2SR;7** —

```
(DEFUN |EUCDOM-;extendedEuclidean;2SR;7| (|x| |y| $)
 (PROG (|s3| |s2| |qr| |s1|)
  (RETURN
   (SEQ
    (LETT |s1|
     (|EUCDOM-;unitNormalizeIdealElt|
      (VECTOR (|spadConstant| $ 24) (|spadConstant| $ 16) |x|)
       $)
     |EUCDOM-;extendedEuclidean;2SR;7|)
    (LETT |s2|
     (|EUCDOM-;unitNormalizeIdealElt|
      (VECTOR (|spadConstant| $ 16) (|spadConstant| $ 24) |y|)
```

```
  $)
  |EUCDOM-;extendedEuclidean;2SR;7|)
 (EXIT
  (COND
   ((SPADCALL |y| (QREFELT $ 8)) |s1|)
   ((SPADCALL |x| (QREFELT $ 8)) |s2|)
   ((QUOTE T)
    (SEQ
     (SEQ
      G190
      (COND
       ((NULL
         (COND
          ((SPADCALL (QVELT |s2| 2) (QREFELT $ 8)) (QUOTE NIL))
          ((QUOTE T) (QUOTE T))))
        (GO G191)))
      (SEQ
       (LETT |qr|
        (SPADCALL (QVELT |s1| 2) (QVELT |s2| 2) (QREFELT $ 13))
        |EUCDOM-;extendedEuclidean;2SR;7|)
       (LETT |s3|
        (VECTOR
         (SPADCALL (QVELT |s1| 0)
          (SPADCALL (QCAR |qr|) (QVELT |s2| 0) (QREFELT $ 26))
          (QREFELT $ 27))
         (SPADCALL (QVELT |s1| 1)
          (SPADCALL (QCAR |qr|) (QVELT |s2| 1) (QREFELT $ 26))
          (QREFELT $ 27))
         (QCDR |qr|))
        |EUCDOM-;extendedEuclidean;2SR;7|)
       (LETT |s1| |s2| |EUCDOM-;extendedEuclidean;2SR;7|)
       (EXIT
        (LETT |s2|
         (|EUCDOM-;unitNormalizeIdealElt| |s3| $)
         |EUCDOM-;extendedEuclidean;2SR;7|)))
      NIL
      (GO G190)
      G191
      (EXIT NIL))
     (COND
      ((NULL (SPADCALL (QVELT |s1| 0) (QREFELT $ 8)))
       (COND
        ((NULL (SPADCALL (QVELT |s1| 0) |y| (QREFELT $ 28)))
         (SEQ
          (LETT |qr|
           (SPADCALL (QVELT |s1| 0) |y| (QREFELT $ 13))
           |EUCDOM-;extendedEuclidean;2SR;7|)
          (QSETVELT |s1| 0 (QCDR |qr|))
          (QSETVELT |s1| 1
           (SPADCALL (QVELT |s1| 1)
```

```
              (SPADCALL (QCAR |qr|) |x| (QREFELT $ 26)) (QREFELT $ 29)))
             (EXIT
              (LETT |s1|
               (|EUCDOM-;unitNormalizeIdealElt| |s1| $)
               |EUCDOM-;extendedEuclidean;2SR;7|)))))))
       (EXIT |s1|)))))))))
```

———-

EUCDOM-;extendedEuclidean;3SU;8

— **EUCDOM-;extendedEuclidean;3SU;8** —

```
(DEFUN |EUCDOM-;extendedEuclidean;3SU;8| (|x| |y| |z| $)
 (PROG (|s| |w| |qr|)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |z| (QREFELT $ 8))
       (CONS 0 (CONS (|spadConstant| $ 16) (|spadConstant| $ 16))))
     ((QUOTE T)
       (SEQ
        (LETT |s|
         (SPADCALL |x| |y| (QREFELT $ 32))
         |EUCDOM-;extendedEuclidean;3SU;8|)
        (LETT |w|
         (SPADCALL |z| (QVELT |s| 2) (QREFELT $ 33))
         |EUCDOM-;extendedEuclidean;3SU;8|)
        (EXIT
         (COND
          ((QEQCAR |w| 1) (CONS 1 "failed"))
          ((SPADCALL |y| (QREFELT $ 8))
            (CONS 0
             (CONS (SPADCALL (QVELT |s| 0) (QCDR |w|) (QREFELT $ 26))
              (SPADCALL (QVELT |s| 1) (QCDR |w|) (QREFELT $ 26)))))
          ((QUOTE T)
           (SEQ
            (LETT |qr|
             (SPADCALL
              (SPADCALL (QVELT |s| 0) (QCDR |w|) (QREFELT $ 26))
              |y|
              (QREFELT $ 13))
             |EUCDOM-;extendedEuclidean;3SU;8|)
            (EXIT
             (CONS 0
              (CONS (QCDR |qr|)
               (SPADCALL
```

```
            (SPADCALL (QVELT |s| 1) (QCDR |w|) (QREFELT $ 26))
            (SPADCALL (QCAR |qr|) |x| (QREFELT $ 26))
            (QREFELT $ 29)))))))))))))))
```

———-

EUCDOM-;principalIdeal;LR;9

— EUCDOM-;principalIdeal;LR;9 —

```
(DEFUN |EUCDOM-;principalIdeal;LR;9| (|l| $)
 (PROG (|uca| |v| |u| #0=#:G1497 |vv| #1=#:G1498)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |l| NIL (QREFELT $ 38))
       (|error| "empty list passed to principalIdeal"))
     ((SPADCALL (CDR |l|) NIL (QREFELT $ 38))
       (SEQ
        (LETT |uca|
         (SPADCALL (|SPADfirst| |l|) (QREFELT $ 23))
         |EUCDOM-;principalIdeal;LR;9|)
        (EXIT (CONS (LIST (QVELT |uca| 0)) (QVELT |uca| 1)))))
     ((SPADCALL (CDR (CDR |l|)) NIL (QREFELT $ 38))
       (SEQ
        (LETT |u|
         (SPADCALL (|SPADfirst| |l|)
          (SPADCALL |l| (QREFELT $ 39)) (QREFELT $ 32))
         |EUCDOM-;principalIdeal;LR;9|)
        (EXIT (CONS (LIST (QVELT |u| 0) (QVELT |u| 1)) (QVELT |u| 2)))))
     ((QUOTE T)
       (SEQ
        (LETT |v|
         (SPADCALL (CDR |l|) (QREFELT $ 42))
         |EUCDOM-;principalIdeal;LR;9|)
        (LETT |u|
         (SPADCALL (|SPADfirst| |l|) (QCDR |v|) (QREFELT $ 32))
         |EUCDOM-;principalIdeal;LR;9|)
        (EXIT
         (CONS
          (CONS (QVELT |u| 0)
           (PROGN
            (LETT #0# NIL |EUCDOM-;principalIdeal;LR;9|)
            (SEQ
             (LETT |vv| NIL |EUCDOM-;principalIdeal;LR;9|)
             (LETT #1# (QCAR |v|) |EUCDOM-;principalIdeal;LR;9|)
             G190
```

```
            (COND
             ((OR (ATOM #1#)
                  (PROGN
                   (LETT |vv| (CAR #1#) |EUCDOM-;principalIdeal;LR;9|) NIL))
              (GO G191)))
            (SEQ
             (EXIT
              (LETT #0#
               (CONS (SPADCALL (QVELT |u| 1) |vv| (QREFELT $ 26))
                  #0#)
               |EUCDOM-;principalIdeal;LR;9|)))
            (LETT #1# (CDR #1#)
             |EUCDOM-;principalIdeal;LR;9|)
            (GO G190)
            G191
            (EXIT (NREVERSE0 #0#)))))
          (QVELT |u| 2))))))))))
```

———-

EUCDOM-;expressIdealMember;LSU;10

— EUCDOM-;expressIdealMember;LSU;10 —

```
(DEFUN |EUCDOM-;expressIdealMember;LSU;10| (|l| |z| $)
 (PROG (#0=#:G1513 #1=#:G1514 |pid| |q| #2=#:G1515 |v| #3=#:G1516)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |z| (|spadConstant| $ 16) (QREFELT $ 25))
      (CONS 0
       (PROGN
        (LETT #0# NIL |EUCDOM-;expressIdealMember;LSU;10|)
        (SEQ
         (LETT |v| NIL |EUCDOM-;expressIdealMember;LSU;10|)
         (LETT #1# |l| |EUCDOM-;expressIdealMember;LSU;10|)
         G190
         (COND
          ((OR (ATOM #1#)
               (PROGN
                (LETT |v| (CAR #1#) |EUCDOM-;expressIdealMember;LSU;10|) NIL))
           (GO G191)))
         (SEQ
          (EXIT
           (LETT #0#
            (CONS (|spadConstant| $ 16) #0#)
            |EUCDOM-;expressIdealMember;LSU;10|)))
```

```
         (LETT #1# (CDR #1#) |EUCDOM-;expressIdealMember;LSU;10|)
         (GO G190)
         G191
        (EXIT (NREVERSE0 #0#))))))
     ((QUOTE T)
      (SEQ
       (LETT |pid|
        (SPADCALL |l| (QREFELT $ 42))
        |EUCDOM-;expressIdealMember;LSU;10|)
       (LETT |q|
        (SPADCALL |z| (QCDR |pid|) (QREFELT $ 33))
        |EUCDOM-;expressIdealMember;LSU;10|)
       (EXIT
        (COND
         ((QEQCAR |q| 1) (CONS 1 "failed"))
         ((QUOTE T)
          (CONS 0
           (PROGN
            (LETT #2# NIL |EUCDOM-;expressIdealMember;LSU;10|)
            (SEQ
             (LETT |v| NIL |EUCDOM-;expressIdealMember;LSU;10|)
             (LETT #3# (QCAR |pid|) |EUCDOM-;expressIdealMember;LSU;10|)
             G190
             (COND
              ((OR (ATOM #3#)
                   (PROGN
                    (LETT |v| (CAR #3#) |EUCDOM-;expressIdealMember;LSU;10|)
                    NIL))
               (GO G191)))
             (SEQ
              (EXIT
              (LETT #2#
               (CONS (SPADCALL (QCDR |q|) |v| (QREFELT $ 26))
                  #2#)
               |EUCDOM-;expressIdealMember;LSU;10|)))
             (LETT #3# (CDR #3#) |EUCDOM-;expressIdealMember;LSU;10|)
             (GO G190)
             G191
             (EXIT (NREVERSE0 #2#)))))))))))))))
```

———-

EUCDOM-;multiEuclidean;LSU;11

— **EUCDOM-;multiEuclidean;LSU;11** —

```
(DEFUN |EUCDOM-;multiEuclidean;LSU;11| (|l| |z| $)
```

```
(PROG (|n| |l1| |l2| #0=#:G1405 #1=#:G1535 #2=#:G1522 #3=#:G1520
       #4=#:G1521 #5=#:G1406 #6=#:G1536 #7=#:G1525 #8=#:G1523 #9=#:G1524
       |u| |v1| |v2|)
 (RETURN
  (SEQ
   (LETT |n| (LENGTH |l|) |EUCDOM-;multiEuclidean;LSU;11|)
   (EXIT
    (COND
     ((ZEROP |n|) (|error| "empty list passed to multiEuclidean"))
     ((EQL |n| 1) (CONS 0 (LIST |z|)))
     ((QUOTE T)
      (SEQ
       (LETT |l1|
        (SPADCALL |l| (QREFELT $ 46)) |EUCDOM-;multiEuclidean;LSU;11|)
       (LETT |l2|
        (SPADCALL |l1| (QUOTIENT2 |n| 2) (QREFELT $ 48))
        |EUCDOM-;multiEuclidean;LSU;11|)
       (LETT |u|
        (SPADCALL
         (PROGN
          (LETT #4# NIL |EUCDOM-;multiEuclidean;LSU;11|)
          (SEQ
           (LETT #0# NIL |EUCDOM-;multiEuclidean;LSU;11|)
           (LETT #1# |l1| |EUCDOM-;multiEuclidean;LSU;11|)
           G190
           (COND
            ((OR (ATOM #1#)
                 (PROGN
                  (LETT #0# (CAR #1#) |EUCDOM-;multiEuclidean;LSU;11|)
                  NIL))
             (GO G191)))
           (SEQ
            (EXIT
             (PROGN
              (LETT #2# #0# |EUCDOM-;multiEuclidean;LSU;11|)
              (COND
               (#4#
                (LETT #3#
                 (SPADCALL #3# #2# (QREFELT $ 26))
                 |EUCDOM-;multiEuclidean;LSU;11|))
               ((QUOTE T)
                (PROGN
                 (LETT #3# #2# |EUCDOM-;multiEuclidean;LSU;11|)
                 (LETT #4# (QUOTE T) |EUCDOM-;multiEuclidean;LSU;11|)))))))
           (LETT #1# (CDR #1#) |EUCDOM-;multiEuclidean;LSU;11|)
           (GO G190)
           G191
           (EXIT NIL))
          (COND (#4# #3#) ((QUOTE T) (|spadConstant| $ 24))))
         (PROGN
```

```
    (LETT #9# NIL |EUCDOM-;multiEuclidean;LSU;11|)
    (SEQ
     (LETT #5# NIL |EUCDOM-;multiEuclidean;LSU;11|)
     (LETT #6# |l2| |EUCDOM-;multiEuclidean;LSU;11|)
     G190
     (COND
      ((OR (ATOM #6#)
           (PROGN
            (LETT #5# (CAR #6#) |EUCDOM-;multiEuclidean;LSU;11|)
            NIL))
       (GO G191)))
     (SEQ
      (EXIT
       (PROGN
        (LETT #7# #5# |EUCDOM-;multiEuclidean;LSU;11|)
        (COND
         (#9#
          (LETT #8#
           (SPADCALL #8# #7# (QREFELT $ 26))
           |EUCDOM-;multiEuclidean;LSU;11|))
         ((QUOTE T)
          (PROGN
           (LETT #8# #7# |EUCDOM-;multiEuclidean;LSU;11|)
           (LETT #9# (QUOTE T) |EUCDOM-;multiEuclidean;LSU;11|)))))))
     (LETT #6# (CDR #6#) |EUCDOM-;multiEuclidean;LSU;11|)
     (GO G190)
     G191
     (EXIT NIL))
    (COND (#9# #8#) ((QUOTE T) (|spadConstant| $ 24))))
   |z| (QREFELT $ 49))
  |EUCDOM-;multiEuclidean;LSU;11|)
 (EXIT
  (COND
   ((QEQCAR |u| 1) (CONS 1 "failed"))
   ((QUOTE T)
    (SEQ
     (LETT |v1|
      (SPADCALL |l1| (QCDR (QCDR |u|)) (QREFELT $ 50))
      |EUCDOM-;multiEuclidean;LSU;11|)
     (EXIT
      (COND
       ((QEQCAR |v1| 1) (CONS 1 "failed"))
       ((QUOTE T)
        (SEQ
         (LETT |v2|
          (SPADCALL |l2| (QCAR (QCDR |u|)) (QREFELT $ 50))
          |EUCDOM-;multiEuclidean;LSU;11|)
         (EXIT
          (COND
           ((QEQCAR |v2| 1) (CONS 1 "failed"))
```

```
                ((QUOTE T)
                 (CONS 0
                  (SPADCALL
                   (QCDR |v1|)
                   (QCDR |v2|)
                   (QREFELT $ 51))))))))))))))))))))))))
```

———-

EuclideanDomain&

— **EuclideanDomainAmp** —

```
(DEFUN |EuclideanDomain&| (|#1|)
 (PROG (DV$1 |dv$| $ |pv$|)
  (RETURN
   (PROGN
    (LETT DV$1 (|devaluate| |#1|) . #0=(|EuclideanDomain&|))
    (LETT |dv$| (LIST (QUOTE |EuclideanDomain&|) DV$1) . #0#)
    (LETT $ (make-array 53) . #0#)
    (QSETREFV $ 0 |dv$|)
    (QSETREFV $ 3 (LETT |pv$| (|buildPredVector| 0 0 NIL) . #0#))
    (|stuffDomainSlots| $)
    (QSETREFV $ 6 |#1|)
    $))))
```

———-

EUCDOM-;INFOVEC

— **EUCDOM-;INFOVEC** —

```
(setf (get
 (QUOTE |EuclideanDomain&|)
 (QUOTE |infovec|))
 (LIST
  (QUOTE #(NIL NIL NIL NIL NIL NIL (|local| |#1|) (|Boolean|) (0 . |zero?|)
  (|NonNegativeInteger|) (5 . |euclideanSize|) |EUCDOM-;sizeLess?;2SB;1|
  (|Record| (|:| |quotient| $) (|:| |remainder| $)) (10 . |divide|)
  |EUCDOM-;quo;3S;2| |EUCDOM-;rem;3S;3| (16 . |Zero|)
  (|Union| $ (QUOTE "failed")) |EUCDOM-;exquo;2SU;4| (20 . |unitCanonical|)
  (25 . |rem|) |EUCDOM-;gcd;3S;5|
  (|Record| (|:| |unit| $) (|:| |canonical| $) (|:| |associate| $))
  (31 . |unitNormal|) (36 . |One|) (40 . =) (46 . *) (52 . -)
```

```
  (58 . |sizeLess?|) (64 . +)
  (|Record| (|:| |coef1| $) (|:| |coef2| $) (|:| |generator| $))
  |EUCDOM-;extendedEuclidean;2SR;7|
  (70 . |extendedEuclidean|) (76 . |exquo|)
  (|Record| (|:| |coef1| $) (|:| |coef2| $))
  (|Union| 34 (QUOTE "failed")) |EUCDOM-;extendedEuclidean;3SU;8|
  (|List| 6) (82 . =) (88 . |second|)
  (|Record| (|:| |coef| 41) (|:| |generator| $))
  (|List| $) (93 . |principalIdeal|) |EUCDOM-;principalIdeal;LR;9|
  (|Union| 41 (QUOTE "failed")) |EUCDOM-;expressIdealMember;LSU;10|
  (98 . |copy|) (|Integer|) (103 . |split!|) (109 . |extendedEuclidean|)
  (116 . |multiEuclidean|) (122 . |concat|) |EUCDOM-;multiEuclidean;LSU;11|))
  (QUOTE
   #(|sizeLess?| 128 |rem| 134 |quo| 140 |principalIdeal| 146
     |multiEuclidean| 151 |gcd| 157 |extendedEuclidean| 163
     |exquo| 176 |expressIdealMember| 182))
  (QUOTE NIL)
  (CONS (|makeByteWordVec2| 1 (QUOTE NIL))
   (CONS (QUOTE #())
    (CONS (QUOTE #())
     (|makeByteWordVec2| 52 (QUOTE (1 6 7 0 8 1 6 9 0 10 2 6 12 0 0 13 0
       6 0 16 1 6 0 0 19 2 6 0 0 0 20 1 6 22 0 23 0 6 0 24 2 6 7 0 0 25 2 6 0
       0 0 26 2 6 0 0 0 27 2 6 7 0 0 28 2 6 0 0 0 29 2 6 30 0 0 32 2 6 17 0 0
       33 2 37 7 0 0 38 1 37 6 0 39 1 6 40 41 42 1 37 0 0 46 2 37 0 0 47 48 3
       6 35 0 0 0 49 2 6 44 41 0 50 2 37 0 0 0 51 2 0 7 0 0 11 2 0 0 0 0 15 2
       0 0 0 0 14 1 0 40 41 43 2 0 44 41 0 52 2 0 0 0 0 21 3 0 35 0 0 0 36 2 0
       30 0 0 31 2 0 17 0 0 18 2 0 44 41 0 45))))))
  (QUOTE |lookupComplete|)))
```

———-

— EUCDOM-.lsp BOOTSTRAP —

```
\getchunk{EUCDOM-;VersionCheck}
\getchunk{EUCDOM-;sizeLess?;2SB;1}
\getchunk{EUCDOM-;quo;3S;2}
\getchunk{EUCDOM-;rem;3S;3}
\getchunk{EUCDOM-;exquo;2SU;4}
\getchunk{EUCDOM-;gcd;3S;5}
\getchunk{EUCDOM-;unitNormalizeIdealElt}
\getchunk{EUCDOM-;extendedEuclidean;2SR;7}
\getchunk{EUCDOM-;extendedEuclidean;3SU;8}
\getchunk{EUCDOM-;principalIdeal;LR;9}
\getchunk{EUCDOM-;expressIdealMember;LSU;10}
\getchunk{EUCDOM-;multiEuclidean;LSU;11}
\getchunk{EuclideanDomainAmp}
\getchunk{EUCDOM-;INFOVEC}
```

21.20 ENTIRER.lsp BOOTSTRAP

ENTIRER depends on itself. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **ENTIRER** category which we can write into the **MID** directory. We compile the lisp code and copy the **ENTIRER.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— ENTIRER.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |EntireRing;AL| (QUOTE NIL))

(DEFUN |EntireRing| NIL
  (LET (#:G82841)
    (COND
      (|EntireRing;AL|)
      (T (SETQ |EntireRing;AL| (|EntireRing;|))))))

(DEFUN |EntireRing;| NIL
  (PROG (#1=#:G82839)
    (RETURN
      (PROG1
        (LETT #1#
          (|Join|
            (|Ring|)
            (|BiModule| (QUOTE |$|) (QUOTE |$|))
            (|mkCategory|
              (QUOTE |package|)
              NIL
              (QUOTE ((|noZeroDivisors| T)))
              (QUOTE NIL)
              NIL))
          |EntireRing|)
        (SETELT #1# 0 (QUOTE (|EntireRing|)))))))

(setf (get (QUOTE |EntireRing|) (QUOTE NILADIC)) T)
```

21.21 FFIELDC.lsp BOOTSTRAP

FFIELDC depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **FFIELDC** category which we can write into the **MID** directory. We compile the lisp code and copy the **FFIELDC.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— FFIELDC.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |FiniteFieldCategory;AL| (QUOTE NIL))

(DEFUN |FiniteFieldCategory| NIL
 (LET (#:G83129)
  (COND
   (|FiniteFieldCategory;AL|)
   (T (SETQ |FiniteFieldCategory;AL| (|FiniteFieldCategory;|))))))

(DEFUN |FiniteFieldCategory;| NIL
 (PROG (#1=#:G83127)
  (RETURN
   (PROG1
    (LETT #1#
     (|Join|
      (|FieldOfPrimeCharacteristic|)
      (|Finite|)
      (|StepThrough|)
      (|DifferentialRing|)
      (|mkCategory|
       (QUOTE |domain|)
       (QUOTE (
        ((|charthRoot| (|$| |$|)) T)
        ((|conditionP| ((|Union| (|Vector| |$|) "failed") (|Matrix| |$|))) T)
        ((|factorsOfCyclicGroupSize|
         ((|List| (|Record|
                   (|:| |factor| (|Integer|))
                   (|:| |exponent| (|Integer|))))))
          T)
        ((|tableForDiscreteLogarithm|
          ((|Table| (|PositiveInteger|) (|NonNegativeInteger|))
           (|Integer|))) T)
        ((|createPrimitiveElement| (|$|)) T)
        ((|primitiveElement| (|$|)) T)
        ((|primitive?| ((|Boolean|) |$|)) T)
        ((|discreteLog| ((|NonNegativeInteger|) |$|)) T)
```

```
        ((|order| ((|PositiveInteger|) |$|)) T)
        ((|representationType|
          ((|Union| "prime" "polynomial" "normal" "cyclic"))) T)))
       NIL
       (QUOTE (
        (|PositiveInteger|)
        (|NonNegativeInteger|)
        (|Boolean|)
        (|Table| (|PositiveInteger|) (|NonNegativeInteger|))
        (|Integer|)
        (|List|
         (|Record| (|:| |factor| (|Integer|)) (|:| |exponent| (|Integer|))))
        (|Matrix| |$|)))
       NIL))
     |FiniteFieldCategory|)
    (SETELT #1# 0 (QUOTE (|FiniteFieldCategory|)))))))

(setf (get (QUOTE |FiniteFieldCategory|) (QUOTE NILADIC)) T)
```

———-

21.22 FFIELDC-.lsp BOOTSTRAP

FFIELDC- depends on **FFIELDC**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **FFIELDC-** category which we can write into the **MID** directory. We compile the lisp code and copy the **FFIELDC-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— FFIELDC-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |FFIELDC-;differentiate;2S;1| (|x| |$|) (|spadConstant| |$| 7))

(DEFUN |FFIELDC-;init;S;2| (|$|) (|spadConstant| |$| 7))

(DEFUN |FFIELDC-;nextItem;SU;3| (|a| |$|)
 (COND
  ((SPADCALL
    (LETT |a|
     (SPADCALL (|+| (SPADCALL |a| (QREFELT |$| 11)) 1) (QREFELT |$| 12))
     |FFIELDC-;nextItem;SU;3|)
    (QREFELT |$| 14))
    (CONS 1 "failed"))
```

```
  ((QUOTE T) (CONS 0 |a|))))

(DEFUN |FFIELDC-;order;SOpc;4| (|e| |$|)
 (SPADCALL (SPADCALL |e| (QREFELT |$| 17)) (QREFELT |$| 20)))

(DEFUN |FFIELDC-;conditionP;MU;5| (|mat| |$|)
 (PROG (|l|)
  (RETURN
   (SEQ
    (LETT |l| (SPADCALL |mat| (QREFELT |$| 24)) |FFIELDC-;conditionP;MU;5|)
    (COND
     ((OR
       (NULL |l|)
       (SPADCALL (ELT |$| 14) (|SPADfirst| |l|) (QREFELT |$| 27)))
        (EXIT (CONS 1 "failed"))))
    (EXIT
     (CONS 0
      (SPADCALL (ELT |$| 28) (|SPADfirst| |l|) (QREFELT |$| 30)))))))))

(DEFUN |FFIELDC-;charthRoot;2S;6| (|x| |$|)
 (SPADCALL |x|
  (QUOTIENT2 (SPADCALL (QREFELT |$| 35)) (SPADCALL (QREFELT |$| 36)))
  (QREFELT |$| 37)))

(DEFUN |FFIELDC-;charthRoot;SU;7| (|x| |$|)
 (CONS 0 (SPADCALL |x| (QREFELT |$| 28))))

(DEFUN |FFIELDC-;createPrimitiveElement;S;8| (|$|)
 (PROG (|sm1| |start| |i| #1=#:G83175 |e| |found|)
  (RETURN
   (SEQ
    (LETT |sm1|
     (|-| (SPADCALL (QREFELT |$| 35)) 1)
     |FFIELDC-;createPrimitiveElement;S;8|)
    (LETT |start|
     (COND
      ((SPADCALL
        (SPADCALL (QREFELT |$| 42))
        (CONS 1 "polynomial")
        (QREFELT |$| 43))
       (SPADCALL (QREFELT |$| 36)))
      ((QUOTE T) 1))
     |FFIELDC-;createPrimitiveElement;S;8|)
    (LETT |found| (QUOTE NIL) |FFIELDC-;createPrimitiveElement;S;8|)
    (SEQ
     (LETT |i| |start| |FFIELDC-;createPrimitiveElement;S;8|)
     G190
     (COND
      ((NULL (COND (|found| (QUOTE NIL)) ((QUOTE T) (QUOTE T))))
        (GO G191)))
```

```
     (SEQ
      (LETT |e|
       (SPADCALL
        (PROG1
         (LETT #1# |i| |FFIELDC-;createPrimitiveElement;S;8|)
         (|check-subtype| (|>| #1# 0) (QUOTE (|PositiveInteger|)) #1#))
        (QREFELT |$| 12))
       |FFIELDC-;createPrimitiveElement;S;8|)
      (EXIT
       (LETT |found|
        (EQL (SPADCALL |e| (QREFELT |$| 17)) |sm1|)
        |FFIELDC-;createPrimitiveElement;S;8|)))
     (LETT |i| (|+| |i| 1) |FFIELDC-;createPrimitiveElement;S;8|)
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT |e|)))))

(DEFUN |FFIELDC-;primitive?;SB;9| (|a| |$|)
 (PROG (|explist| |q| |exp| #1=#:G83187 |equalone|)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |a| (QREFELT |$| 14)) (QUOTE NIL))
     ((QUOTE T)
      (SEQ
       (LETT |explist|
        (SPADCALL (QREFELT |$| 47)) |FFIELDC-;primitive?;SB;9|)
       (LETT |q|
        (|-| (SPADCALL (QREFELT |$| 35)) 1) |FFIELDC-;primitive?;SB;9|)
       (LETT |equalone| (QUOTE NIL) |FFIELDC-;primitive?;SB;9|)
       (SEQ
        (LETT |exp| NIL |FFIELDC-;primitive?;SB;9|)
        (LETT #1# |explist| |FFIELDC-;primitive?;SB;9|)
        G190
        (COND
         ((OR
           (ATOM #1#)
           (PROGN (LETT |exp| (CAR #1#) |FFIELDC-;primitive?;SB;9|) NIL)
           (NULL (COND (|equalone| (QUOTE NIL)) ((QUOTE T) (QUOTE T)))))
          (GO G191)))
        (SEQ
         (EXIT
          (LETT |equalone|
           (SPADCALL
            (SPADCALL |a| (QUOTIENT2 |q| (QCAR |exp|)) (QREFELT |$| 48))
            (QREFELT |$| 49))
           |FFIELDC-;primitive?;SB;9|)))
        (LETT #1# (CDR #1#) |FFIELDC-;primitive?;SB;9|)
        (GO G190)
```

```
        G191
        (EXIT NIL))
       (EXIT (COND (|equalone| (QUOTE NIL)) ((QUOTE T) (QUOTE T)))))))))))

(DEFUN |FFIELDC-;order;SPi;10| (|e| |$|)
 (PROG (|lof| |rec| #1=#:G83195 |primeDivisor|
        |j| #2=#:G83196 |a| |goon| |ord|)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |e| (|spadConstant| |$| 7) (QREFELT |$| 51))
       (|error| "order(0) is not defined "))
     ((QUOTE T)
      (SEQ
       (LETT |ord|
        (|-| (SPADCALL (QREFELT |$| 35)) 1) |FFIELDC-;order;SPi;10|)
       (LETT |a| 0 |FFIELDC-;order;SPi;10|)
       (LETT |lof| (SPADCALL (QREFELT |$| 47)) |FFIELDC-;order;SPi;10|)
       (SEQ
        (LETT |rec| NIL |FFIELDC-;order;SPi;10|)
        (LETT #1# |lof| |FFIELDC-;order;SPi;10|)
        G190
        (COND
         ((OR
           (ATOM #1#)
           (PROGN (LETT |rec| (CAR #1#) |FFIELDC-;order;SPi;10|) NIL))
          (GO G191)))
        (SEQ
         (LETT |a|
          (QUOTIENT2 |ord|
           (LETT |primeDivisor| (QCAR |rec|) |FFIELDC-;order;SPi;10|))
          |FFIELDC-;order;SPi;10|)
         (LETT |goon|
          (SPADCALL (SPADCALL |e| |a| (QREFELT |$| 48)) (QREFELT |$| 49))
          |FFIELDC-;order;SPi;10|)
         (SEQ
          (LETT |j| 0 |FFIELDC-;order;SPi;10|)
          (LETT #2# (|-| (QCDR |rec|) 2) |FFIELDC-;order;SPi;10|)
          G190
          (COND ((OR (QSGREATERP |j| #2#) (NULL |goon|)) (GO G191)))
          (SEQ
           (LETT |ord| |a| |FFIELDC-;order;SPi;10|)
           (LETT |a|
            (QUOTIENT2 |ord| |primeDivisor|)
            |FFIELDC-;order;SPi;10|)
           (EXIT
            (LETT |goon|
             (SPADCALL (SPADCALL |e| |a| (QREFELT |$| 48)) (QREFELT |$| 49))
             |FFIELDC-;order;SPi;10|)))
          (LETT |j| (QSADD1 |j|) |FFIELDC-;order;SPi;10|)
```

```
          (GO G190)
          G191
          (EXIT NIL))
         (EXIT (COND (|goon| (LETT |ord| |a| |FFIELDC-;order;SPi;10|)))))
        (LETT #1# (CDR #1#) |FFIELDC-;order;SPi;10|)
        (GO G190)
        G191
        (EXIT NIL))
       (EXIT |ord|))))))))

(DEFUN |FFIELDC-;discreteLog;SNni;11| (|b| |$|)
 (PROG (|faclist| |gen| |groupord| |f| #1=#:G83216 |fac| |t| #2=#:G83217
        |exp| |exptable| |n| |end| |i| |rho| |found| |disc1| |c| |mult|
        |disclog| |a|)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |b| (QREFELT |$| 14))
       (|error| "discreteLog: logarithm of zero"))
     ((QUOTE T)
      (SEQ
       (LETT |faclist|
        (SPADCALL (QREFELT |$| 47))
        |FFIELDC-;discreteLog;SNni;11|)
       (LETT |a| |b| |FFIELDC-;discreteLog;SNni;11|)
       (LETT |gen|
        (SPADCALL (QREFELT |$| 53))
        |FFIELDC-;discreteLog;SNni;11|)
       (EXIT
        (COND
         ((SPADCALL |b| |gen| (QREFELT |$| 51)) 1)
         ((QUOTE T)
          (SEQ
           (LETT |disclog| 0 |FFIELDC-;discreteLog;SNni;11|)
           (LETT |mult| 1 |FFIELDC-;discreteLog;SNni;11|)
           (LETT |groupord|
            (|-| (SPADCALL (QREFELT |$| 35)) 1)
            |FFIELDC-;discreteLog;SNni;11|)
           (LETT |exp| |groupord| |FFIELDC-;discreteLog;SNni;11|)
           (SEQ
            (LETT |f| NIL |FFIELDC-;discreteLog;SNni;11|)
            (LETT #1# |faclist| |FFIELDC-;discreteLog;SNni;11|)
            G190
            (COND
             ((OR
               (ATOM #1#)
               (PROGN
                (LETT |f| (CAR #1#) |FFIELDC-;discreteLog;SNni;11|)
                NIL))
              (GO G191)))
```

```
(SEQ
 (LETT |fac| (QCAR |f|) |FFIELDC-;discreteLog;SNni;11|)
 (EXIT
  (SEQ
   (LETT |t| 0 |FFIELDC-;discreteLog;SNni;11|)
   (LETT #2# (|-| (QCDR |f|) 1) |FFIELDC-;discreteLog;SNni;11|)
   G190
   (COND ((QSGREATERP |t| #2#) (GO G191)))
   (SEQ
    (LETT |exp|
     (QUOTIENT2 |exp| |fac|)
     |FFIELDC-;discreteLog;SNni;11|)
    (LETT |exptable|
     (SPADCALL |fac| (QREFELT |$| 55))
     |FFIELDC-;discreteLog;SNni;11|)
    (LETT |n|
     (SPADCALL |exptable| (QREFELT |$| 56))
     |FFIELDC-;discreteLog;SNni;11|)
    (LETT |c|
     (SPADCALL |a| |exp| (QREFELT |$| 48))
     |FFIELDC-;discreteLog;SNni;11|)
    (LETT |end|
     (QUOTIENT2 (|-| |fac| 1) |n|)
     |FFIELDC-;discreteLog;SNni;11|)
    (LETT |found| (QUOTE NIL) |FFIELDC-;discreteLog;SNni;11|)
    (LETT |disc1| 0 |FFIELDC-;discreteLog;SNni;11|)
    (SEQ
     (LETT |i| 0 |FFIELDC-;discreteLog;SNni;11|)
     G190
     (COND
      ((OR
        (QSGREATERP |i| |end|)
        (NULL
         (COND (|found| (QUOTE NIL)) ((QUOTE T) (QUOTE T)))))
       (GO G191)))
      (SEQ
       (LETT |rho|
        (SPADCALL
         (SPADCALL |c| (QREFELT |$| 11))
         |exptable|
         (QREFELT |$| 58))
        |FFIELDC-;discreteLog;SNni;11|)
       (EXIT
        (COND
         ((QEQCAR |rho| 0)
          (SEQ
           (LETT |found| (QUOTE T) |FFIELDC-;discreteLog;SNni;11|)
           (EXIT
            (LETT |disc1|
             (|*| (|+| (|*| |n| |i|) (QCDR |rho|)) |mult|)
```

```
                    |FFIELDC-;discreteLog;SNni;11|))))
                  ((QUOTE T)
                   (LETT |c|
                    (SPADCALL |c|
                     (SPADCALL |gen|
                      (|*| (QUOTIENT2 |groupord| |fac|) (|-| |n|))
                      (QREFELT |$| 48))
                     (QREFELT |$| 59))
                    |FFIELDC-;discreteLog;SNni;11|)))))
                 (LETT |i| (QSADD1 |i|) |FFIELDC-;discreteLog;SNni;11|)
                 (GO G190)
                 G191
                 (EXIT NIL))
                (EXIT
                 (COND
                  (|found|
                   (SEQ
                    (LETT |mult|
                     (|*| |mult| |fac|)
                     |FFIELDC-;discreteLog;SNni;11|)
                    (LETT |disclog|
                     (|+| |disclog| |disc1|)
                     |FFIELDC-;discreteLog;SNni;11|)
                    (EXIT
                     (LETT |a|
                      (SPADCALL |a|
                       (SPADCALL |gen| (|-| |disc1|) (QREFELT |$| 48))
                       (QREFELT |$| 59))
                      |FFIELDC-;discreteLog;SNni;11|))))
                   ((QUOTE T)
                    (|error| "discreteLog: ?? discrete logarithm")))))
                (LETT |t|
                 (QSADD1 |t|)
                 |FFIELDC-;discreteLog;SNni;11|)
                (GO G190)
                G191
                (EXIT NIL))))
              (LETT #1#
               (CDR #1#)
               |FFIELDC-;discreteLog;SNni;11|)
              (GO G190)
              G191
              (EXIT NIL))
             (EXIT |disclog|))))))))))))

(DEFUN |FFIELDC-;discreteLog;2SU;12| (|logbase| |b| |$|)
 (PROG (|groupord| |faclist| |f| #1=#:G83235 |fac| |primroot|
        |t| #2=#:G83236 |exp| |rhoHelp| #3=#:G83234 |rho| |disclog|
        |mult| |a|)
  (RETURN
```

```
   (SEQ
    (EXIT
     (COND
      ((SPADCALL |b| (QREFELT |$| 14))
       (SEQ
        (SPADCALL "discreteLog: logarithm of zero" (QREFELT |$| 64))
        (EXIT (CONS 1 "failed"))))
      ((SPADCALL |logbase| (QREFELT |$| 14))
       (SEQ
        (SPADCALL "discreteLog: logarithm to base zero" (QREFELT |$| 64))
        (EXIT (CONS 1 "failed"))))
      ((SPADCALL |b| |logbase| (QREFELT |$| 51)) (CONS 0 1))
      ((QUOTE T)
       (COND
        ((NULL
          (ZEROP
           (REMAINDER2
            (LETT |groupord|
             (SPADCALL |logbase| (QREFELT |$| 17))
             |FFIELDC-;discreteLog;2SU;12|)
            (SPADCALL |b| (QREFELT |$| 17)))))
         (SEQ
          (SPADCALL
"discreteLog: second argument not in cyclic group generated by first argument"
           (QREFELT |$| 64))
          (EXIT (CONS 1 "failed"))))
        ((QUOTE T)
         (SEQ
          (LETT |faclist|
           (SPADCALL (SPADCALL |groupord| (QREFELT |$| 66)) (QREFELT |$| 68))
           |FFIELDC-;discreteLog;2SU;12|)
          (LETT |a| |b| |FFIELDC-;discreteLog;2SU;12|)
          (LETT |disclog| 0 |FFIELDC-;discreteLog;2SU;12|)
          (LETT |mult| 1 |FFIELDC-;discreteLog;2SU;12|)
          (LETT |exp| |groupord| |FFIELDC-;discreteLog;2SU;12|)
          (SEQ
           (LETT |f| NIL |FFIELDC-;discreteLog;2SU;12|)
           (LETT #1# |faclist| |FFIELDC-;discreteLog;2SU;12|)
           G190
           (COND
            ((OR
              (ATOM #1#)
              (PROGN (LETT |f| (CAR #1#) |FFIELDC-;discreteLog;2SU;12|) NIL))
             (GO G191)))
           (SEQ
            (LETT |fac| (QCAR |f|) |FFIELDC-;discreteLog;2SU;12|)
            (LETT |primroot|
             (SPADCALL |logbase|
              (QUOTIENT2 |groupord| |fac|)
              (QREFELT |$| 48))
```

```
           |FFIELDC-;discreteLog;2SU;12|)
          (EXIT
           (SEQ
            (LETT |t| 0 |FFIELDC-;discreteLog;2SU;12|)
            (LETT #2# (|-| (QCDR |f|) 1) |FFIELDC-;discreteLog;2SU;12|)
            G190
            (COND ((QSGREATERP |t| #2#) (GO G191)))
            (SEQ
             (LETT |exp|
              (QUOTIENT2 |exp| |fac|)
              |FFIELDC-;discreteLog;2SU;12|)
             (LETT |rhoHelp|
              (SPADCALL |primroot|
               (SPADCALL |a| |exp| (QREFELT |$| 48))
               |fac|
               (QREFELT |$| 70))
               |FFIELDC-;discreteLog;2SU;12|)
             (EXIT
              (COND
               ((QEQCAR |rhoHelp| 1)
                (PROGN
                 (LETT #3# (CONS 1 "failed") |FFIELDC-;discreteLog;2SU;12|)
                 (GO #3#)))
               ((QUOTE T)
                (SEQ
                 (LETT |rho|
                  (|*| (QCDR |rhoHelp|) |mult|)
                  |FFIELDC-;discreteLog;2SU;12|)
                 (LETT |disclog|
                  (|+| |disclog| |rho|)
                  |FFIELDC-;discreteLog;2SU;12|)
                 (LETT |mult|
                  (|*| |mult| |fac|)
                  |FFIELDC-;discreteLog;2SU;12|)
                 (EXIT
                  (LETT |a|
                   (SPADCALL |a|
                    (SPADCALL |logbase| (|-| |rho|) (QREFELT |$| 48))
                    (QREFELT |$| 59))
                   |FFIELDC-;discreteLog;2SU;12|)))))))
             (LETT |t| (QSADD1 |t|) |FFIELDC-;discreteLog;2SU;12|)
             (GO G190)
             G191
             (EXIT NIL))))
           (LETT #1# (CDR #1#) |FFIELDC-;discreteLog;2SU;12|)
           (GO G190)
           G191
           (EXIT NIL))
          (EXIT (CONS 0 |disclog|)))))))
#3#
```

```
    (EXIT #3#)))))

(DEFUN |FFIELDC-;squareFreePolynomial| (|f| |$|)
 (SPADCALL |f| (QREFELT |$| 75)))

(DEFUN |FFIELDC-;factorPolynomial| (|f| |$|)
 (SPADCALL |f| (QREFELT |$| 77)))

(DEFUN |FFIELDC-;factorSquareFreePolynomial| (|f| |$|)
 (PROG (|flist| |u| #1=#:G83248 #2=#:G83245 #3=#:G83243 #4=#:G83244)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |f| (|spadConstant| |$| 78) (QREFELT |$| 79))
       (|spadConstant| |$| 80))
     ((QUOTE T)
      (SEQ
       (LETT |flist|
        (SPADCALL |f| (QUOTE T) (QREFELT |$| 83))
        |FFIELDC-;factorSquareFreePolynomial|)
       (EXIT
        (SPADCALL
         (SPADCALL (QCAR |flist|) (QREFELT |$| 84))
         (PROGN
          (LETT #4# NIL |FFIELDC-;factorSquareFreePolynomial|)
          (SEQ
           (LETT |u| NIL |FFIELDC-;factorSquareFreePolynomial|)
           (LETT #1# (QCDR |flist|) |FFIELDC-;factorSquareFreePolynomial|)
           G190
           (COND
            ((OR
              (ATOM #1#)
              (PROGN
               (LETT |u| (CAR #1#) |FFIELDC-;factorSquareFreePolynomial|)
               NIL))
             (GO G191)))
           (SEQ
            (EXIT
             (PROGN
              (LETT #2#
               (SPADCALL (QCAR |u|) (QCDR |u|) (QREFELT |$| 85))
               |FFIELDC-;factorSquareFreePolynomial|)
              (COND
               (#4#
                (LETT #3#
                 (SPADCALL #3# #2# (QREFELT |$| 86))
                 |FFIELDC-;factorSquareFreePolynomial|))
               ((QUOTE T)
                (PROGN
                 (LETT #3# #2# |FFIELDC-;factorSquareFreePolynomial|)
```

```
                (LETT #4#
                 (QUOTE T)
                 |FFIELDC-;factorSquareFreePolynomial|)))))))
           (LETT #1# (CDR #1#) |FFIELDC-;factorSquareFreePolynomial|)
           (GO G190)
           G191
          (EXIT NIL))
          (COND (#4# #3#) ((QUOTE T) (|spadConstant| |$| 87))))
         (QREFELT |$| 88)))))))))))

(DEFUN |FFIELDC-;gcdPolynomial;3Sup;16| (|f| |g| |$|)
 (SPADCALL |f| |g| (QREFELT |$| 90)))

(DEFUN |FiniteFieldCategory&| (|#1|)
 (PROG (|DV$1| |dv$| |$| |pv$|)
  (RETURN
   (PROGN
    (LETT |DV$1| (|devaluate| |#1|) . #1=(|FiniteFieldCategory&|))
    (LETT |dv$| (LIST (QUOTE |FiniteFieldCategory&|) |DV$1|) . #1#)
    (LETT |$| (make-array 93) . #1#)
    (QSETREFV |$| 0 |dv$|)
    (QSETREFV |$| 3 (LETT |pv$| (|buildPredVector| 0 0 NIL) . #1#))
    (|stuffDomainSlots| |$|)
    (QSETREFV |$| 6 |#1|) |$|))))

(setf (get
 (QUOTE |FiniteFieldCategory&|)
 (QUOTE |infovec|))
 (LIST
  (QUOTE
  #(NIL NIL NIL NIL NIL NIL (|local| |#1|) (0 . |Zero|)
    |FFIELDC-;differentiate;2S;1| |FFIELDC-;init;S;2| (|PositiveInteger|)
    (4 . |lookup|) (9 . |index|) (|Boolean|) (14 . |zero?|)
    (|Union| |$| (QUOTE "failed")) |FFIELDC-;nextItem;SU;3| (19 . |order|)
    (|Integer|) (|OnePointCompletion| 10) (24 . |coerce|)
    |FFIELDC-;order;SOpc;4| (|List| 26) (|Matrix| 6) (29 . |nullSpace|)
    (|Mapping| 13 6) (|Vector| 6) (34 . |every?|) (40 . |charthRoot|)
    (|Mapping| 6 6) (45 . |map|) (|Union| (|Vector| |$|) (QUOTE "failed"))
    (|Matrix| |$|) |FFIELDC-;conditionP;MU;5| (|NonNegativeInteger|)
    (51 . |size|) (55 . |characteristic|) (59 . |**|)
    |FFIELDC-;charthRoot;2S;6| |FFIELDC-;charthRoot;SU;7| (65 . |One|)
    (|Union| (QUOTE "prime") (QUOTE "polynomial") (QUOTE "normal")
    (QUOTE "cyclic")) (69 . |representationType|) (73 . |=|)
    |FFIELDC-;createPrimitiveElement;S;8| (|Record| (|:| |factor| 18)
    (|:| |exponent| 18)) (|List| 45) (79 . |factorsOfCyclicGroupSize|)
    (83 . |**|) (89 . |one?|) |FFIELDC-;primitive?;SB;9| (94 . |=|)
    |FFIELDC-;order;SPi;10| (100 . |primitiveElement|) (|Table| 10 34)
    (104 . |tableForDiscreteLogarithm|) (109 . |#|)
    (|Union| 34 (QUOTE "failed")) (114 . |search|) (120 . |*|)
    |FFIELDC-;discreteLog;SNni;11| (|Void|) (|String|) (|OutputForm|)
```

```
   (126 . |messagePrint|) (|Factored| |$|) (131 . |factor|)
   (|Factored| 18) (136 . |factors|) (|DiscreteLogarithmPackage| 6)
   (141 . |shanksDiscLogAlgorithm|) |FFIELDC-;discreteLog;2SU;12|
   (|Factored| 73) (|SparseUnivariatePolynomial| 6)
   (|UnivariatePolynomialSquareFree| 6 73) (148 . |squareFree|)
   (|DistinctDegreeFactorize| 6 73) (153 . |factor|) (158 . |Zero|)
   (162 . |=|) (168 . |Zero|) (|Record| (|:| |irr| 73) (|:| |pow| 18))
   (|Record| (|:| |cont| 6) (|:| |factors| (|List| 81)))
   (172 . |distdfact|) (178 . |coerce|) (183 . |primeFactor|)
   (189 . |*|) (195 . |One|) (199 . |*|) (|EuclideanDomain&| 73)
   (205 . |gcd|) (|SparseUnivariatePolynomial| |$|)
   |FFIELDC-;gcdPolynomial;3Sup;16|))
  (QUOTE
   #(|primitive?| 211 |order| 216 |nextItem| 226 |init| 231
     |gcdPolynomial| 235 |discreteLog| 241 |differentiate| 252
     |createPrimitiveElement| 257 |conditionP| 261 |charthRoot| 266))
 (QUOTE NIL)
 (CONS
  (|makeByteWordVec2| 1 (QUOTE NIL))
  (CONS
   (QUOTE #())
   (CONS
    (QUOTE #())
    (|makeByteWordVec2| 92
     (QUOTE
      (0 6 0 7 1 6 10 0 11 1 6 0 10 12 1 6 13 0 14 1 6 10 0 17 1 19
       0 18 20 1 23 22 0 24 2 26 13 25 0 27 1 6 0 0 28 2 26 0 29 0 30
       0 6 34 35 0 6 34 36 2 6 0 0 34 37 0 6 0 40 0 6 41 42 2 41 13
       0 0 43 0 6 46 47 2 6 0 0 18 48 1 6 13 0 49 2 6 13 0 0 51 0 6 0
       53 1 6 54 18 55 1 54 34 0 56 2 54 57 10 0 58 2 6 0 0 0 59 1
       63 61 62 64 1 18 65 0 66 1 67 46 0 68 3 69 57 6 6 34 70 1 74
       72 73 75 1 76 72 73 77 0 73 0 78 2 73 13 0 0 79 0 72 0 80 2
       76 82 73 13 83 1 73 0 6 84 2 72 0 73 18 85 2 72 0 0 0 86 0 72
       0 87 2 72 0 73 0 88 2 89 0 0 0 90 1 0 13 0 50 1 0 10 0 52 1 0
       19 0 21 1 0 15 0 16 0 0 0 9 2 0 91 91 91 92 1 0 34 0 60 2 0 57
       0 0 71 1 0 0 0 8 0 0 0 44 1 0 31 32 33 1 0 0 0 38 1 0 15 0 39))))))
 (QUOTE |lookupComplete|)))
```

———-

21.23 FPS.lsp BOOTSTRAP

FPS depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **FPS** category which we can write into the **MID** directory. We compile the lisp code and copy the **FPS.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— FPS.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |FloatingPointSystem;AL| (QUOTE NIL))

(DEFUN |FloatingPointSystem| NIL
  (LET (#:G105645)
    (COND
      (|FloatingPointSystem;AL|)
      (T (SETQ |FloatingPointSystem;AL| (|FloatingPointSystem;|))))))

(DEFUN |FloatingPointSystem;| NIL
  (PROG (#1=#:G105643)
    (RETURN
      (PROG1
        (LETT #1#
          (|Join|
            (|RealNumberSystem|)
            (|mkCategory|
              (QUOTE |domain|)
              (QUOTE (
                ((|float| (|$| (|Integer|) (|Integer|))) T)
                ((|float| (|$| (|Integer|) (|Integer|) (|PositiveInteger|))) T)
                ((|order| ((|Integer|) |$|)) T)
                ((|base| ((|PositiveInteger|))) T)
                ((|exponent| ((|Integer|) |$|)) T)
                ((|mantissa| ((|Integer|) |$|)) T)
                ((|bits| ((|PositiveInteger|))) T)
                ((|digits| ((|PositiveInteger|))) T)
                ((|precision| ((|PositiveInteger|))) T)
                ((|bits| ((|PositiveInteger|) (|PositiveInteger|)))
                  (|has| |$| (ATTRIBUTE |arbitraryPrecision|)))
                ((|digits| ((|PositiveInteger|) (|PositiveInteger|)))
                  (|has| |$| (ATTRIBUTE |arbitraryPrecision|)))
                ((|precision| ((|PositiveInteger|) (|PositiveInteger|)))
                  (|has| |$| (ATTRIBUTE |arbitraryPrecision|)))
                ((|increasePrecision| ((|PositiveInteger|) (|Integer|)))
                  (|has| |$| (ATTRIBUTE |arbitraryPrecision|)))
                ((|decreasePrecision| ((|PositiveInteger|) (|Integer|)))
                  (|has| |$| (ATTRIBUTE |arbitraryPrecision|)))
                ((|min| (|$|))
                  (AND
                    (|not| (|has| |$| (ATTRIBUTE |arbitraryPrecision|)))
                    (|not| (|has| |$| (ATTRIBUTE |arbitraryExponent|)))))
                ((|max| (|$|))
                  (AND
                    (|not| (|has| |$| (ATTRIBUTE |arbitraryPrecision|)))
```

```
                (|not| (|has| |$| (ATTRIBUTE |arbitraryExponent|)))))))
            (QUOTE ((|approximate| T)))
            (QUOTE ((|PositiveInteger|) (|Integer|)))
            NIL))
          |FloatingPointSystem|)
        (SETELT #1# 0 (QUOTE (|FloatingPointSystem|)))))))

(setf (get (QUOTE |FloatingPointSystem|) (QUOTE NILADIC)) T)
```

———-

21.24 FPS-.lsp BOOTSTRAP

FPS- depends **FPS**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **FPS-** category which we can write into the **MID** directory. We compile the lisp code and copy the **FPS-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— **FPS-.lsp BOOTSTRAP** —

```
(|/VERSIONCHECK| 2)

(DEFUN |FPS-;float;2IS;1| (|ma| |ex| |$|)
  (SPADCALL |ma| |ex| (SPADCALL (QREFELT |$| 8)) (QREFELT |$| 10)))

(DEFUN |FPS-;digits;Pi;2| (|$|)
  (PROG (#1=#:G105654)
    (RETURN
      (PROG1
        (LETT #1#
          (MAX 1
            (QUOTIENT2
              (SPADCALL 4004
                (|-| (SPADCALL (QREFELT |$| 13)) 1)
                (QREFELT |$| 14))
              13301))
          |FPS-;digits;Pi;2|)
        (|check-subtype| (|>| #1# 0) (QUOTE (|PositiveInteger|)) #1#)))))

(DEFUN |FloatingPointSystem&| (|#1|)
  (PROG (|DV$1| |dv$| |$| |pv$|)
    (RETURN
      (PROGN
        (LETT |DV$1| (|devaluate| |#1|) . #1=(|FloatingPointSystem&|))
```

```
        (LETT |dv$| (LIST (QUOTE |FloatingPointSystem&|) |DV$1|) . #1#)
        (LETT |$| (make-array 17) . #1#)
        (QSETREFV |$| 0 |dv$|)
        (QSETREFV |$| 3
          (LETT |pv$|
            (|buildPredVector| 0 0
              (LIST
                (|HasAttribute| |#1| (QUOTE |arbitraryExponent|))
                (|HasAttribute| |#1| (QUOTE |arbitraryPrecision|)))) . #1#))
        (|stuffDomainSlots| |$|)
        (QSETREFV |$| 6 |#1|)
        |$|))))

(setf (get
  (QUOTE |FloatingPointSystem&|)
  (QUOTE |infovec|))
  (LIST
    (QUOTE
      #(NIL NIL NIL NIL NIL NIL
        (|local| |#1|)
        (|PositiveInteger|)
        (0 . |base|)
        (|Integer|)
        (4 . |float|)
        |FPS-;float;2IS;1|
        (11 . |One|)
        (15 . |bits|)
        (19 . |*|)
        (25 . |max|)
        |FPS-;digits;Pi;2|))
    (QUOTE #(|float| 29 |digits| 35))
    (QUOTE NIL)
    (CONS
      (|makeByteWordVec2| 1 (QUOTE NIL))
      (CONS
        (QUOTE #())
        (CONS
          (QUOTE #())
          (|makeByteWordVec2| 16
            (QUOTE
              (0 6 7 8 3 6 0 9 9 7 10 0 6 0 12 0 6 7 13 2 9 0 7 0 14 0 6 0 15
               2 0 0 9 9 11 0 0 7 16))))))
     (QUOTE |lookupComplete|)))
```

———-

21.25 GCDDOM.lsp BOOTSTRAP

GCDDOM needs **COMRING** which needs **RING** which needs **RNG** which needs **ABELGRP** which needs **CABMON** which needs **ABELMON** which needs **ABELSG** which needs **SETCAT** which needs **SINT** which needs **UFD** which needs **GCDDOM**. We break this chain with **GCDDOM.lsp** which we cache here. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **GCDDOM** category which we can write into the **MID** directory. We compile the lisp code and copy the **GCDDOM.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— GCDDOM.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |GcdDomain;AL| (QUOTE NIL))

(DEFUN |GcdDomain| NIL
  (LET (#:G83171)
    (COND
      (|GcdDomain;AL|)
      (T (SETQ |GcdDomain;AL| (|GcdDomain;|))))))

(DEFUN |GcdDomain;| NIL
  (PROG (#1=#:G83169)
    (RETURN
      (PROG1
        (LETT #1#
          (|Join|
            (|IntegralDomain|)
            (|mkCategory|
              (QUOTE |domain|)
              (QUOTE (
                ((|gcd| (|$| |$| |$|)) T)
                ((|gcd| (|$| (|List| |$|))) T)
                ((|lcm| (|$| |$| |$|)) T)
                ((|lcm| (|$| (|List| |$|))) T)
                ((|gcdPolynomial|
                  ((|SparseUnivariatePolynomial| |$|)
                   (|SparseUnivariatePolynomial| |$|)
                   (|SparseUnivariatePolynomial| |$|)))
                  T)))
              NIL
              (QUOTE ((|SparseUnivariatePolynomial| |$|) (|List| |$|)))
              NIL))
            |GcdDomain|)
        (SETELT #1# 0 (QUOTE (|GcdDomain|)))))))
```

```
(setf (get (QUOTE |GcdDomain|) (QUOTE NILADIC)) T)
```

———-

21.26 GCDDOM-.lsp BOOTSTRAP

GCDDOM- depends on **GCDDOM**. We break this chain with **GCDDOM-.lsp** which we cache here. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **GCDDOM-** category which we can write into the **MID** directory. We compile the lisp code and copy the **GCDDOM-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— GCDDOM-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |GCDDOM-;lcm;3S;1| (|x| |y| |$|)
  (PROG (LCM)
    (RETURN
      (SEQ
        (COND
          ((OR
              (SPADCALL |y| (|spadConstant| |$| 7) (QREFELT |$| 9))
              (SPADCALL |x| (|spadConstant| |$| 7) (QREFELT |$| 9)))
            (|spadConstant| |$| 7))
          ((QUOTE T)
            (SEQ
              (LETT LCM
                (SPADCALL |y|
                  (SPADCALL |x| |y| (QREFELT |$| 10))
                  (QREFELT |$| 12))
                |GCDDOM-;lcm;3S;1|)
              (EXIT
                (COND
                  ((QEQCAR LCM 0) (SPADCALL |x| (QCDR LCM) (QREFELT |$| 13)))
                  ((QUOTE T) (|error| "bad gcd in lcm computation")))))))))))

(DEFUN |GCDDOM-;lcm;LS;2| (|l| |$|)
  (SPADCALL
    (ELT |$| 15)
    |l|
    (|spadConstant| |$| 16)
    (|spadConstant| |$| 7)
```

```
    (QREFELT |$| 19)))

(DEFUN |GCDDOM-;gcd;LS;3| (|l| |$|)
  (SPADCALL
    (ELT |$| 10)
    |l|
    (|spadConstant| |$| 7)
    (|spadConstant| |$| 16)
    (QREFELT |$| 19)))

(DEFUN |GCDDOM-;gcdPolynomial;3Sup;4| (|p1| |p2| |$|)
  (PROG (|e2| |e1| |c1| |p| |c2| #1=#:G83191)
    (RETURN
      (SEQ
        (COND
          ((SPADCALL |p1| (QREFELT |$| 24)) (SPADCALL |p2| (QREFELT |$| 25)))
          ((SPADCALL |p2| (QREFELT |$| 24)) (SPADCALL |p1| (QREFELT |$| 25)))
          ((QUOTE T)
            (SEQ
              (LETT |c1|
                (SPADCALL |p1| (QREFELT |$| 26))
                |GCDDOM-;gcdPolynomial;3Sup;4|)
              (LETT |c2|
                (SPADCALL |p2| (QREFELT |$| 26))
                |GCDDOM-;gcdPolynomial;3Sup;4|)
              (LETT |p1|
                (PROG2
                  (LETT #1#
                    (SPADCALL |p1| |c1| (QREFELT |$| 27))
                    |GCDDOM-;gcdPolynomial;3Sup;4|)
                  (QCDR #1#)
                  (|check-union|
                    (QEQCAR #1# 0)
                    (|SparseUnivariatePolynomial| (QREFELT |$| 6))
                    #1#))
                |GCDDOM-;gcdPolynomial;3Sup;4|)
              (LETT |p2|
                (PROG2
                  (LETT #1#
                    (SPADCALL |p2| |c2| (QREFELT |$| 27))
                    |GCDDOM-;gcdPolynomial;3Sup;4|)
                  (QCDR #1#)
                  (|check-union|
                    (QEQCAR #1# 0)
                    (|SparseUnivariatePolynomial| (QREFELT |$| 6))
                    #1#))
                |GCDDOM-;gcdPolynomial;3Sup;4|)
              (SEQ
                (LETT |e1|
                  (SPADCALL |p1| (QREFELT |$| 29))
```

```
    |GCDDOM-;gcdPolynomial;3Sup;4|)
  (EXIT
    (COND
      ((|<| 0 |e1|)
        (LETT |p1|
          (PROG2
            (LETT #1#
              (SPADCALL |p1|
                (SPADCALL
                 (|spadConstant| |$| 16) |e1| (QREFELT |$| 32))
                (QREFELT |$| 33))
              |GCDDOM-;gcdPolynomial;3Sup;4|)
            (QCDR #1#)
            (|check-union|
              (QEQCAR #1# 0)
              (|SparseUnivariatePolynomial| (QREFELT |$| 6))
              #1#))
          |GCDDOM-;gcdPolynomial;3Sup;4|))))))
(SEQ
  (LETT |e2|
    (SPADCALL |p2| (QREFELT |$| 29))
    |GCDDOM-;gcdPolynomial;3Sup;4|)
  (EXIT
    (COND
      ((|<| 0 |e2|)
        (LETT |p2|
          (PROG2
            (LETT #1#
              (SPADCALL |p2|
                (SPADCALL
                  (|spadConstant| |$| 16)
                  |e2|
                  (QREFELT |$| 32))
                (QREFELT |$| 33))
              |GCDDOM-;gcdPolynomial;3Sup;4|)
            (QCDR #1#)
            (|check-union|
              (QEQCAR #1# 0)
              (|SparseUnivariatePolynomial| (QREFELT |$| 6))
              #1#))
          |GCDDOM-;gcdPolynomial;3Sup;4|))))))
(LETT |e1|
  (MIN |e1| |e2|)
  |GCDDOM-;gcdPolynomial;3Sup;4|)
(LETT |c1|
  (SPADCALL |c1| |c2| (QREFELT |$| 10))
  |GCDDOM-;gcdPolynomial;3Sup;4|)
(LETT |p1|
  (COND
    ((OR
```

```
            (EQL (SPADCALL |p1| (QREFELT |$| 34)) 0)
            (EQL (SPADCALL |p2| (QREFELT |$| 34)) 0))
           (SPADCALL |c1| 0 (QREFELT |$| 32)))
         ((QUOTE T)
           (SEQ
            (LETT |p|
              (SPADCALL |p1| |p2| (QREFELT |$| 35))
              |GCDDOM-;gcdPolynomial;3Sup;4|)
            (EXIT
              (COND
                ((EQL (SPADCALL |p| (QREFELT |$| 34)) 0)
                  (SPADCALL |c1| 0 (QREFELT |$| 32)))
                ((QUOTE T)
                  (SEQ
                    (LETT |c2|
                      (SPADCALL
                        (SPADCALL |p1| (QREFELT |$| 36))
                        (SPADCALL |p2| (QREFELT |$| 36))
                        (QREFELT |$| 10))
                      |GCDDOM-;gcdPolynomial;3Sup;4|)
                    (EXIT
                      (SPADCALL
                        (SPADCALL |c1|
                          (SPADCALL
                            (PROG2
                              (LETT #1#
                                (SPADCALL
                                  (SPADCALL
                                    |c2|
                                    |p|
                                    (QREFELT |$| 37))
                                  (SPADCALL |p| (QREFELT |$| 36))
                                  (QREFELT |$| 27))
                                |GCDDOM-;gcdPolynomial;3Sup;4|)
                              (QCDR #1#)
                              (|check-union|
                                (QEQCAR #1# 0)
                                (|SparseUnivariatePolynomial|
                                  (QREFELT |$| 6))
                                #1#))
                            (QREFELT |$| 38))
                          (QREFELT |$| 37))
                        (QREFELT |$| 25)))))))))
    |GCDDOM-;gcdPolynomial;3Sup;4|)
  (EXIT
   (COND
    ((ZEROP |e1|) |p1|)
    ((QUOTE T)
      (SPADCALL
       (SPADCALL (|spadConstant| |$| 16) |e1| (QREFELT |$| 32))
```

```
                  |p1| (QREFELT |$| 39))))))))))))

(DEFUN |GcdDomain&| (|#1|)
  (PROG (|DV$1| |dv$| |$| |pv$|)
    (RETURN
      (PROGN
        (LETT |DV$1| (|devaluate| |#1|) . #1=(|GcdDomain&|))
        (LETT |dv$| (LIST (QUOTE |GcdDomain&|) |DV$1|) . #1#)
        (LETT |$| (make-array 42) . #1#)
        (QSETREFV |$| 0 |dv$|)
        (QSETREFV |$| 3 (LETT |pv$| (|buildPredVector| 0 0 NIL) . #1#))
        (|stuffDomainSlots| |$|)
        (QSETREFV |$| 6 |#1|)
        |$|))))

(setf (get
  (QUOTE |GcdDomain&|)
  (QUOTE |infovec|))
  (LIST
    (QUOTE
      #(NIL NIL NIL NIL NIL NIL
        (|local| |#1|)
        (0 . |Zero|)
        (|Boolean|)
        (4 . |=|)
        (10 . |gcd|)
        (|Union| |$| (QUOTE "failed"))
        (16 . |exquo|)
        (22 . |*|)
        |GCDDOM-;lcm;3S;1|
        (28 . |lcm|)
        (34 . |One|)
        (|Mapping| 6 6 6)
        (|List| 6)
        (38 . |reduce|)
        (|List| |$|)
        |GCDDOM-;lcm;LS;2|
        |GCDDOM-;gcd;LS;3|
        (|SparseUnivariatePolynomial| 6)
        (46 . |zero?|)
        (51 . |unitCanonical|)
        (56 . |content|)
        (61 . |exquo|)
        (|NonNegativeInteger|)
        (67 . |minimumDegree|)
        (72 . |Zero|)
        (76 . |One|)
        (80 . |monomial|)
        (86 . |exquo|)
        (92 . |degree|)
```

```
        (97 . |subResultantGcd|)
        (103 . |leadingCoefficient|)
        (108 . |*|)
        (114 . |primitivePart|)
        (119 . |*|)
        (|SparseUnivariatePolynomial| |$|)
        |GCDDOM-;gcdPolynomial;3Sup;4|))
    (QUOTE #(|lcm| 125 |gcdPolynomial| 136 |gcd| 142))
    (QUOTE NIL)
    (CONS
      (|makeByteWordVec2| 1 (QUOTE NIL))
      (CONS
        (QUOTE #())
        (CONS
          (QUOTE #())
          (|makeByteWordVec2| 41
           (QUOTE (0 6 0 7 2 6 8 0 0 9 2 6 0 0 0 10 2 6 11 0 0 12 2 6 0 0 0
                   13 2 6 0 0 0 15 0 6 0 16 4 18 6 17 0 6 6 19 1 23 8 0 24
                   1 23 0 0 25 1 23 6 0 26 2 23 11 0 6 27 1 23 28 0 29 0 23
                   0 30 0 23 0 31 2 23 0 6 28 32 2 23 11 0 0 33 1 23 28 0
                   34 2 23 0 0 0 35 1 23 6 0 36 2 23 0 6 0 37 1 23 0 0 38 2
                   23 0 0 0 39 1 0 0 20 21 2 0 0 0 0 14 2 0 40 40 40 41 1 0
                   0 20 22))))))
   (QUOTE |lookupComplete|)))
```

———-

21.27 HOAGG.lsp BOOTSTRAP

HOAGG depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **HOAGG** category which we can write into the **MID** directory. We compile the lisp code and copy the **HOAGG.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— HOAGG.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |HomogeneousAggregate;CAT| (QUOTE NIL))

(SETQ |HomogeneousAggregate;AL| (QUOTE NIL))

(DEFUN |HomogeneousAggregate| (#1=#:G82375)
  (LET (#2=#:G82376)
```

```
    (COND
      ((SETQ #2# (|assoc| (|devaluate| #1#) |HomogeneousAggregate;AL|))
        (CDR #2#))
      (T
        (SETQ |HomogeneousAggregate;AL|
          (|cons5|
            (CONS (|devaluate| #1#) (SETQ #2# (|HomogeneousAggregate;| #1#)))
            |HomogeneousAggregate;AL|))
        #2#))))

(DEFUN |HomogeneousAggregate;| (|t#1|)
  (PROG (#1=#:G82374)
    (RETURN
      (PROG1
        (LETT #1#
          (|sublisV|
            (PAIR (QUOTE (|t#1|)) (LIST (|devaluate| |t#1|)))
            (COND
              (|HomogeneousAggregate;CAT|)
              ((QUOTE T)
                (LETT |HomogeneousAggregate;CAT|
                  (|Join|
                    (|Aggregate|)
                    (|mkCategory|
                      (QUOTE |domain|)
                      (QUOTE (
                        ((|map| (|$| (|Mapping| |t#1| |t#1|) |$|)) T)
                        ((|map!| (|$| (|Mapping| |t#1| |t#1|) |$|))
                          (|has| |$| (ATTRIBUTE |shallowlyMutable|)))
                        ((|any?|
                           ((|Boolean|) (|Mapping| (|Boolean|) |t#1|) |$|))
                          (|has| |$| (ATTRIBUTE |finiteAggregate|)))
                        ((|every?|
                           ((|Boolean|) (|Mapping| (|Boolean|) |t#1|) |$|))
                          (|has| |$| (ATTRIBUTE |finiteAggregate|)))
                        ((|count|
                           ((|NonNegativeInteger|)
                            (|Mapping| (|Boolean|) |t#1|) |$|))
                          (|has| |$| (ATTRIBUTE |finiteAggregate|)))
                        ((|parts| ((|List| |t#1|) |$|))
                          (|has| |$| (ATTRIBUTE |finiteAggregate|)))
                        ((|members| ((|List| |t#1|) |$|))
                          (|has| |$| (ATTRIBUTE |finiteAggregate|)))
                        ((|count| ((|NonNegativeInteger|) |t#1| |$|))
                          (AND
                            (|has| |t#1| (|SetCategory|))
                            (|has| |$| (ATTRIBUTE |finiteAggregate|))))
                        ((|member?| ((|Boolean|) |t#1| |$|))
                          (AND
                            (|has| |t#1| (|SetCategory|))
```

```
                    (|has| |$| (ATTRIBUTE |finiteAggregate|))))))
                (QUOTE (
                 ((|SetCategory|) (|has| |t#1| (|SetCategory|)))
                 ((|Evalable| |t#1|)
                   (AND
                     (|has| |t#1| (|Evalable| |t#1|))
                     (|has| |t#1| (|SetCategory|))))))
               (QUOTE (
                 (|Boolean|)
                 (|NonNegativeInteger|)
                 (|List| |t#1|)))
               NIL))
          . #2=(|HomogeneousAggregate|))))) . #2#)
    (SETELT #1# 0
      (LIST (QUOTE |HomogeneousAggregate|) (|devaluate| |t#1|)))))))
```

———-

21.28 HOAGG-.lsp BOOTSTRAP

HOAGG- depends on **HOAGG**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **HOAGG-** category which we can write into the **MID** directory. We compile the lisp code and copy the **HOAGG-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— HOAGG-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |HOAGG-;eval;ALA;1| (|u| |l| |$|)
 (SPADCALL
  (CONS (FUNCTION |HOAGG-;eval;ALA;1!0|) (VECTOR |$| |l|))
  |u|
  (QREFELT |$| 11)))

(DEFUN |HOAGG-;eval;ALA;1!0| (|#1| |$$|)
 (SPADCALL |#1| (QREFELT |$$| 1) (QREFELT (QREFELT |$$| 0) 9)))

(DEFUN |HOAGG-;#;ANni;2| (|c| |$|)
 (LENGTH (SPADCALL |c| (QREFELT |$| 14))))

(DEFUN |HOAGG-;any?;MAB;3| (|f| |c| |$|)
 (PROG (|x| #1=#:G82396 #2=#:G82393 #3=#:G82391 #4=#:G82392)
  (RETURN
```

```
   (SEQ
    (PROGN
     (LETT #4# NIL |HOAGG-;any?;MAB;3|)
     (SEQ
      (LETT |x| NIL |HOAGG-;any?;MAB;3|)
      (LETT #1# (SPADCALL |c| (QREFELT |$| 14)) |HOAGG-;any?;MAB;3|)
      G190
      (COND
       ((OR (ATOM #1#) (PROGN (LETT |x| (CAR #1#) |HOAGG-;any?;MAB;3|) NIL))
        (GO G191)))
      (SEQ
       (EXIT
        (PROGN
         (LETT #2# (SPADCALL |x| |f|) |HOAGG-;any?;MAB;3|)
         (COND
          (#4#
           (LETT #3#
             (COND
              (#3# (QUOTE T))
              ((QUOTE T) #2#))
             |HOAGG-;any?;MAB;3|))
          ((QUOTE T)
           (PROGN
            (LETT #3# #2# |HOAGG-;any?;MAB;3|)
            (LETT #4# (QUOTE T) |HOAGG-;any?;MAB;3|)))))))
      (LETT #1# (CDR #1#) |HOAGG-;any?;MAB;3|) (GO G190) G191 (EXIT NIL))
     (COND (#4# #3#) ((QUOTE T) (QUOTE NIL))))))))

(DEFUN |HOAGG-;every?;MAB;4| (|f| |c| |$|)
 (PROG (|x| #1=#:G82401 #2=#:G82399 #3=#:G82397 #4=#:G82398)
  (RETURN
   (SEQ
    (PROGN
     (LETT #4# NIL |HOAGG-;every?;MAB;4|)
     (SEQ
      (LETT |x| NIL |HOAGG-;every?;MAB;4|)
      (LETT #1# (SPADCALL |c| (QREFELT |$| 14)) |HOAGG-;every?;MAB;4|)
      G190
      (COND
        ((OR (ATOM #1#) (PROGN (LETT |x| (CAR #1#) |HOAGG-;every?;MAB;4|) NIL))
         (GO G191)))
      (SEQ
       (EXIT
        (PROGN
         (LETT #2# (SPADCALL |x| |f|) |HOAGG-;every?;MAB;4|)
         (COND
          (#4#
           (LETT #3#
            (COND (#3# #2#) ((QUOTE T) (QUOTE NIL)))
            |HOAGG-;every?;MAB;4|))
```

```
          ((QUOTE T)
           (PROGN
            (LETT #3# #2# |HOAGG-;every?;MAB;4|)
            (LETT #4# (QUOTE T) |HOAGG-;every?;MAB;4|)))))))
      (LETT #1# (CDR #1#) |HOAGG-;every?;MAB;4|)
      (GO G190)
      G191
      (EXIT NIL))
     (COND (#4# #3#) ((QUOTE T) (QUOTE T)))))))

(DEFUN |HOAGG-;count;MANni;5| (|f| |c| |$|)
 (PROG (|x| #1=#:G82406 #2=#:G82404 #3=#:G82402 #4=#:G82403)
  (RETURN
   (SEQ
    (PROGN
     (LETT #4# NIL |HOAGG-;count;MANni;5|)
     (SEQ
      (LETT |x| NIL |HOAGG-;count;MANni;5|)
      (LETT #1# (SPADCALL |c| (QREFELT |$| 14)) |HOAGG-;count;MANni;5|)
      G190
      (COND
       ((OR (ATOM #1#) (PROGN (LETT |x| (CAR #1#) |HOAGG-;count;MANni;5|) NIL))
        (GO G191)))
      (SEQ
       (EXIT
        (COND
         ((SPADCALL |x| |f|)
          (PROGN
           (LETT #2# 1 |HOAGG-;count;MANni;5|)
           (COND
            (#4# (LETT #3# (|+| #3# #2#) |HOAGG-;count;MANni;5|))
            ((QUOTE T)
             (PROGN
              (LETT #3# #2# |HOAGG-;count;MANni;5|)
              (LETT #4# (QUOTE T) |HOAGG-;count;MANni;5|)))))))))
      (LETT #1# (CDR #1#) |HOAGG-;count;MANni;5|)
      (GO G190)
      G191
      (EXIT NIL))
     (COND (#4# #3#) ((QUOTE T) 0)))))))

(DEFUN |HOAGG-;members;AL;6| (|x| |$|) (SPADCALL |x| (QREFELT |$| 14)))

(DEFUN |HOAGG-;count;SANni;7| (|s| |x| |$|)
 (SPADCALL
  (CONS (FUNCTION |HOAGG-;count;SANni;7!0|) (VECTOR |$| |s|))
  |x|
  (QREFELT |$| 24)))

(DEFUN |HOAGG-;count;SANni;7!0| (|#1| |$$|)
```

```
 (SPADCALL (QREFELT |$$| 1) |#1| (QREFELT (QREFELT |$$| 0) 23)))

(DEFUN |HOAGG-;member?;SAB;8| (|e| |c| |$|)
 (SPADCALL
  (CONS (FUNCTION |HOAGG-;member?;SAB;8!0|) (VECTOR |$| |e|))
  |c|
  (QREFELT |$| 26)))

(DEFUN |HOAGG-;member?;SAB;8!0| (|#1| |$$|)
 (SPADCALL (QREFELT |$$| 1) |#1| (QREFELT (QREFELT |$$| 0) 23)))

(DEFUN |HOAGG-;=;2AB;9| (|x| |y| |$|)
 (PROG (|b| #1=#:G82416 |a| #2=#:G82415 #3=#:G82412 #4=#:G82410 #5=#:G82411)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |x| (SPADCALL |y| (QREFELT |$| 28)) (QREFELT |$| 29))
      (PROGN
       (LETT #5# NIL |HOAGG-;=;2AB;9|)
       (SEQ
        (LETT |b| NIL |HOAGG-;=;2AB;9|)
        (LETT #1# (SPADCALL |y| (QREFELT |$| 14)) |HOAGG-;=;2AB;9|)
        (LETT |a| NIL |HOAGG-;=;2AB;9|)
        (LETT #2# (SPADCALL |x| (QREFELT |$| 14)) |HOAGG-;=;2AB;9|)
        G190
        (COND
         ((OR
           (ATOM #2#)
           (PROGN (LETT |a| (CAR #2#) |HOAGG-;=;2AB;9|) NIL)
           (ATOM #1#)
           (PROGN (LETT |b| (CAR #1#) |HOAGG-;=;2AB;9|) NIL))
          (GO G191)))
        (SEQ
         (EXIT
          (PROGN
           (LETT #3# (SPADCALL |a| |b| (QREFELT |$| 23)) |HOAGG-;=;2AB;9|)
           (COND
             (#5#
              (LETT #4#
               (COND (#4# #3#) ((QUOTE T) (QUOTE NIL)))
               |HOAGG-;=;2AB;9|))
             ((QUOTE T)
              (PROGN
               (LETT #4# #3# |HOAGG-;=;2AB;9|)
               (LETT #5# (QUOTE T) |HOAGG-;=;2AB;9|)))))))
        (LETT #2#
         (PROG1
          (CDR #2#)
          (LETT #1# (CDR #1#) |HOAGG-;=;2AB;9|))
         |HOAGG-;=;2AB;9|)
```

```
        (GO G190)
        G191
        (EXIT NIL))
       (COND (#5# #4#) ((QUOTE T) (QUOTE T)))))
     ((QUOTE T) (QUOTE NIL)))))))

(DEFUN |HOAGG-;coerce;AOf;10| (|x| |$|)
 (PROG (#1=#:G82420 |a| #2=#:G82421)
  (RETURN
   (SEQ
    (SPADCALL
     (SPADCALL
      (PROGN
       (LETT #1# NIL |HOAGG-;coerce;AOf;10|)
       (SEQ
        (LETT |a| NIL |HOAGG-;coerce;AOf;10|)
        (LETT #2# (SPADCALL |x| (QREFELT |$| 14)) |HOAGG-;coerce;AOf;10|)
        G190
        (COND
         ((OR
           (ATOM #2#)
           (PROGN (LETT |a| (CAR #2#) |HOAGG-;coerce;AOf;10|) NIL))
          (GO G191)))
        (SEQ
         (EXIT
          (LETT #1#
           (CONS (SPADCALL |a| (QREFELT |$| 32)) #1#)
           |HOAGG-;coerce;AOf;10|)))
        (LETT #2# (CDR #2#) |HOAGG-;coerce;AOf;10|)
        (GO G190)
        G191
        (EXIT (NREVERSE0 #1#))))
      (QREFELT |$| 34))
     (QREFELT |$| 35))))))

(DEFUN |HomogeneousAggregate&| (|#1| |#2|)
 (PROG (|DV$1| |DV$2| |dv$| |$| |pv$|)
  (RETURN
   (PROGN
    (LETT |DV$1| (|devaluate| |#1|) . #1=(|HomogeneousAggregate&|))
    (LETT |DV$2| (|devaluate| |#2|) . #1#)
    (LETT |dv$| (LIST (QUOTE |HomogeneousAggregate&|) |DV$1| |DV$2|) . #1#)
    (LETT |$| (make-array 38) . #1#)
    (QSETREFV |$| 0 |dv$|)
    (QSETREFV |$| 3
     (LETT |pv$|
      (|buildPredVector| 0 0
       (LIST
        (|HasAttribute| |#1| (QUOTE |finiteAggregate|))
        (|HasAttribute| |#1| (QUOTE |shallowlyMutable|))
```

```
        (|HasCategory| |#2| (LIST (QUOTE |Evalable|) (|devaluate| |#2|)))
        (|HasCategory| |#2| (QUOTE (|SetCategory|)))))
      . #1#))
    (|stuffDomainSlots| |$|)
    (QSETREFV |$| 6 |#1|)
    (QSETREFV |$| 7 |#2|)
    (COND
     ((|testBitVector| |pv$| 3)
       (QSETREFV |$| 12 (CONS (|dispatchFunction| |HOAGG-;eval;ALA;1|) |$|))))
    (COND
     ((|testBitVector| |pv$| 1)
      (PROGN
       (QSETREFV |$| 16 (CONS (|dispatchFunction| |HOAGG-;#;ANni;2|) |$|))
       (QSETREFV |$| 19 (CONS (|dispatchFunction| |HOAGG-;any?;MAB;3|) |$|))
       (QSETREFV |$| 20 (CONS (|dispatchFunction| |HOAGG-;every?;MAB;4|) |$|))
       (QSETREFV |$| 21 (CONS (|dispatchFunction| |HOAGG-;count;MANni;5|) |$|))
       (QSETREFV |$| 22 (CONS (|dispatchFunction| |HOAGG-;members;AL;6|) |$|))
       (COND
        ((|testBitVector| |pv$| 4)
         (PROGN
          (QSETREFV |$| 25
           (CONS (|dispatchFunction| |HOAGG-;count;SANni;7|) |$|))
          (QSETREFV |$| 27
           (CONS (|dispatchFunction| |HOAGG-;member?;SAB;8|) |$|))
          (QSETREFV |$| 30
           (CONS (|dispatchFunction| |HOAGG-;=;2AB;9|) |$|))
          (QSETREFV |$| 36
           (CONS (|dispatchFunction| |HOAGG-;coerce;AOf;10|) |$|)))))))))
    |$|))))

(setf (get
 (QUOTE |HomogeneousAggregate&|)
 (QUOTE |infovec|))
 (LIST
  (QUOTE #(NIL NIL NIL NIL NIL NIL (|local| |#1|) (|local| |#2|) (|List| 37)
   (0 . |eval|) (|Mapping| 7 7) (6 . |map|) (12 . |eval|) (|List| 7)
   (18 . |parts|) (|NonNegativeInteger|) (23 . |#|) (|Boolean|)
   (|Mapping| 17 7) (28 . |any?|) (34 . |every?|) (40 . |count|)
   (46 . |members|) (51 . |=|) (57 . |count|) (63 . |count|) (69 . |any?|)
   (75 . |member?|) (81 . |#|) (86 . |size?|) (92 . |=|) (|OutputForm|)
   (98 . |coerce|) (|List| |$|) (103 . |commaSeparate|) (108 . |bracket|)
   (113 . |coerce|) (|Equation| 7)))
  (QUOTE #(|members| 118 |member?| 123 |every?| 129 |eval| 135 |count| 141
   |coerce| 153 |any?| 158 |=| 164 |#| 170))
  (QUOTE NIL)
  (CONS
   (|makeByteWordVec2| 1 (QUOTE NIL))
   (CONS
    (QUOTE #())
    (CONS
```

```
    (QUOTE #())
    (|makeByteWordVec2| 36
     (QUOTE (2 7 0 0 8 9 2 6 0 10 0 11 2 0 0 0 8 12 1 6 13 0 14 1 0 15 0
       16 2 0 17 18 0 19 2 0 17 18 0 20 2 0 15 18 0 21 1 0 13 0 22 2 7 17
       0 0 23 2 6 15 18 0 24 2 0 15 7 0 25 2 6 17 18 0 26 2 0 17 7 0 27 1
       6 15 0 28 2 6 17 0 15 29 2 0 17 0 0 30 1 7 31 0 32 1 31 0 33 34 1
       31 0 0 35 1 0 31 0 36 1 0 13 0 22 2 0 17 7 0 27 2 0 17 18 0 20 2 0
       0 0 8 12 2 0 15 7 0 25 2 0 15 18 0 21 1 0 31 0 36 2 0 17 18 0 19 2
       0 17 0 0 30 1 0 15 0 16))))))
  (QUOTE |lookupComplete|)))
```

———-

21.29 INS.lsp BOOTSTRAP

INS depends on itself. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **INS** category which we can write into the **MID** directory. We compile the lisp code and copy the **INS.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

— **INS.lsp BOOTSTRAP** —

```
(/VERSIONCHECK 2)

(SETQ |IntegerNumberSystem;AL| (QUOTE NIL))

(DEFUN |IntegerNumberSystem| NIL
  (LET (#:G1068)
    (COND
      (|IntegerNumberSystem;AL|)
      (T (SETQ |IntegerNumberSystem;AL| (|IntegerNumberSystem;|))))))

(DEFUN |IntegerNumberSystem;| NIL (PROG (#0=#:G1066)
  (RETURN
    (PROG1
      (LETT #0#
        (|sublisV|
          (PAIR
            (QUOTE (#1=#:G1060 #2=#:G1061 #3=#:G1062
                    #4=#:G1063 #5=#:G1064 #6=#:G1065))
            (LIST
              (QUOTE (|Integer|))
              (QUOTE (|Integer|))
              (QUOTE (|Integer|))
              (QUOTE (|InputForm|))
              (QUOTE (|Pattern| (|Integer|)))
              (QUOTE (|Integer|))))
          (|Join|
```

```
            (|UniqueFactorizationDomain|)
            (|EuclideanDomain|)
            (|OrderedIntegralDomain|)
            (|DifferentialRing|)
            (|ConvertibleTo| (QUOTE #1#))
            (|RetractableTo| (QUOTE #2#))
            (|LinearlyExplicitRingOver| (QUOTE #3#))
            (|ConvertibleTo| (QUOTE #4#))
            (|ConvertibleTo| (QUOTE #5#))
            (|PatternMatchable| (QUOTE #6#))
            (|CombinatorialFunctionCategory|)
            (|RealConstant|)
            (|CharacteristicZero|)
            (|StepThrough|)
            (|mkCategory|
              (QUOTE |domain|)
              (QUOTE (
                ((|odd?| ((|Boolean|) $)) T)
                ((|even?| ((|Boolean|) $)) T)
                ((|base| ($)) T)
                ((|length| ($ $)) T)
                ((|shift| ($ $ $)) T)
                ((|bit?| ((|Boolean|) $ $)) T)
                ((|positiveRemainder| ($ $ $)) T)
                ((|symmetricRemainder| ($ $ $)) T)
                ((|rational?| ((|Boolean|) $)) T)
                ((|rational| ((|Fraction| (|Integer|)) $)) T)
                ((|rationalIfCan|
                  ((|Union| (|Fraction| (|Integer|)) "failed") $)) T)
                ((|random| ($)) T)
                ((|random| ($ $)) T)
                ((|hash| ($ $)) T)
                ((|copy| ($ $)) T)
                ((|inc| ($ $)) T)
                ((|dec| ($ $)) T)
                ((|mask| ($ $)) T)
                ((|addmod| ($ $ $ $)) T)
                ((|submod| ($ $ $ $)) T)
                ((|mulmod| ($ $ $ $)) T)
                ((|powmod| ($ $ $ $)) T)
                ((|invmod| ($ $ $)) T)))
              (QUOTE ((|multiplicativeValuation| T) (|canonicalUnitNormal| T)))
              (QUOTE ((|Fraction| (|Integer|)) (|Boolean|))) NIL)))
          |IntegerNumberSystem|)
        (SETELT #0# 0 (QUOTE (|IntegerNumberSystem|)))))))

(setf (get (QUOTE |IntegerNumberSystem|) (QUOTE NILADIC)) T)
```

21.30 INS-.lsp BOOTSTRAP

INS- depends on **INS**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **INS-** category which we can write into the **MID** directory. We compile the lisp code and copy the **INS-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

— **INS-.lsp BOOTSTRAP** —

```
(/VERSIONCHECK 2)

(PUT
  (QUOTE |INS-;characteristic;Nni;1|)
  (QUOTE |SPADreplace|)
  (QUOTE (XLAM NIL 0)))

(DEFUN |INS-;characteristic;Nni;1| ($) 0)

(DEFUN |INS-;differentiate;2S;2| (|x| $)
  (|spadConstant| $ 9))

(DEFUN |INS-;even?;SB;3| (|x| $)
 (COND
   ((SPADCALL |x| (QREFELT $ 12)) (QUOTE NIL))
   ((QUOTE T) (QUOTE T))))

(DEFUN |INS-;positive?;SB;4| (|x| $)
  (SPADCALL (|spadConstant| $ 9) |x| (QREFELT $ 14)))

(PUT
  (QUOTE |INS-;copy;2S;5|)
  (QUOTE |SPADreplace|)
  (QUOTE (XLAM (|x|) |x|)))

(DEFUN |INS-;copy;2S;5| (|x| $) |x|)

(DEFUN |INS-;bit?;2SB;6| (|x| |i| $)
  (SPADCALL
    (SPADCALL |x|
      (SPADCALL |i| (QREFELT $ 17))
      (QREFELT $ 18))
    (QREFELT $ 12)))

(DEFUN |INS-;mask;2S;7| (|n| $)
  (SPADCALL
    (SPADCALL (|spadConstant| $ 20) |n| (QREFELT $ 18))
    (QREFELT $ 21)))

(PUT
```

```
  (QUOTE |INS-;rational?;SB;8|)
  (QUOTE |SPADreplace|)
  (QUOTE (XLAM (|x|) (QUOTE T))))

(DEFUN |INS-;rational?;SB;8| (|x| $)
  (QUOTE T))

(DEFUN |INS-;euclideanSize;SNni;9| (|x| $)
  (PROG (#0=#:G1078 #1=#:G1079)
    (RETURN
      (COND
        ((SPADCALL |x| (|spadConstant| $ 9) (QREFELT $ 24))
          (|error| "euclideanSize called on zero"))
        ((SPADCALL |x| (|spadConstant| $ 9) (QREFELT $ 14))
          (PROG1
            (LETT #0#
              (- (SPADCALL |x| (QREFELT $ 26)))
              |INS-;euclideanSize;SNni;9|)
            (|check-subtype|
              (>= #0# 0)
              (QUOTE (|NonNegativeInteger|))
              #0#)))
        ((QUOTE T)
          (PROG1
            (LETT #1#
              (SPADCALL |x| (QREFELT $ 26))
              |INS-;euclideanSize;SNni;9|)
            (|check-subtype|
              (>= #1# 0)
              (QUOTE (|NonNegativeInteger|))
              #1#)))))))

(DEFUN |INS-;convert;SF;10| (|x| $)
  (SPADCALL (SPADCALL |x| (QREFELT $ 26)) (QREFELT $ 29)))

(DEFUN |INS-;convert;SDf;11| (|x| $)
  (FLOAT (SPADCALL |x| (QREFELT $ 26)) MOST-POSITIVE-LONG-FLOAT))

(DEFUN |INS-;convert;SIf;12| (|x| $)
  (SPADCALL (SPADCALL |x| (QREFELT $ 26)) (QREFELT $ 34)))

(DEFUN |INS-;retract;SI;13| (|x| $)
  (SPADCALL |x| (QREFELT $ 26)))

(DEFUN |INS-;convert;SP;14| (|x| $)
  (SPADCALL (SPADCALL |x| (QREFELT $ 26)) (QREFELT $ 38)))

(DEFUN |INS-;factor;SF;15| (|x| $)
  (SPADCALL |x| (QREFELT $ 42)))
```

```
(DEFUN |INS-;squareFree;SF;16| (|x| $)
  (SPADCALL |x| (QREFELT $ 45)))

(DEFUN |INS-;prime?;SB;17| (|x| $)
  (SPADCALL |x| (QREFELT $ 48)))

(DEFUN |INS-;factorial;2S;18| (|x| $)
  (SPADCALL |x| (QREFELT $ 51)))

(DEFUN |INS-;binomial;3S;19| (|n| |m| $)
  (SPADCALL |n| |m| (QREFELT $ 53)))

(DEFUN |INS-;permutation;3S;20| (|n| |m| $)
  (SPADCALL |n| |m| (QREFELT $ 55)))

(DEFUN |INS-;retractIfCan;SU;21| (|x| $)
  (CONS 0 (SPADCALL |x| (QREFELT $ 26))))

(DEFUN |INS-;init;S;22| ($)
  (|spadConstant| $ 9))

(DEFUN |INS-;nextItem;SU;23| (|n| $)
  (COND
    ((SPADCALL |n| (QREFELT $ 60))
       (CONS 0 (|spadConstant| $ 20)))
    ((SPADCALL (|spadConstant| $ 9) |n| (QREFELT $ 14))
       (CONS 0 (SPADCALL |n| (QREFELT $ 17))))
    ((QUOTE T)
       (CONS 0 (SPADCALL (|spadConstant| $ 20) |n| (QREFELT $ 61))))))

(DEFUN |INS-;patternMatch;SP2Pmr;24| (|x| |p| |l| $)
  (SPADCALL |x| |p| |l| (QREFELT $ 66)))

(DEFUN |INS-;rational;SF;25| (|x| $)
  (SPADCALL (SPADCALL |x| (QREFELT $ 26)) (QREFELT $ 70)))

(DEFUN |INS-;rationalIfCan;SU;26| (|x| $)
  (CONS 0 (SPADCALL (SPADCALL |x| (QREFELT $ 26)) (QREFELT $ 70))))

(DEFUN |INS-;symmetricRemainder;3S;27| (|x| |n| $)
  (PROG (|r|)
    (RETURN
      (SEQ
        (LETT |r|
          (SPADCALL |x| |n| (QREFELT $ 74))
          |INS-;symmetricRemainder;3S;27|)
        (EXIT
          (COND
            ((SPADCALL |r| (|spadConstant| $ 9) (QREFELT $ 24)) |r|)
            ((QUOTE T)
```

```
              (SEQ
                (COND
                  ((SPADCALL |n| (|spadConstant| $ 9) (QREFELT $ 14))
                    (LETT |n|
                      (SPADCALL |n| (QREFELT $ 17))
                      |INS-;symmetricRemainder;3S;27|)))
                (EXIT
                  (COND
                    ((SPADCALL (|spadConstant| $ 9) |r| (QREFELT $ 14))
                      (COND
                        ((SPADCALL |n|
                            (SPADCALL 2 |r| (QREFELT $ 76))
                            (QREFELT $ 14))
                          (SPADCALL |r| |n| (QREFELT $ 61)))
                        ((QUOTE T) |r|)))
                    ((NULL
                      (SPADCALL
                         (|spadConstant| $ 9)
                         (SPADCALL
                           (SPADCALL 2 |r| (QREFELT $ 76))
                           |n|
                           (QREFELT $ 77))
                         (QREFELT $ 14)))
                       (SPADCALL |r| |n| (QREFELT $ 77)))
                     ((QUOTE T) |r|))))))))))))

(DEFUN |INS-;invmod;3S;28| (|a| |b| $)
  (PROG (|q| |r| |r1| |c| |c1| |d| |d1|)
    (RETURN
      (SEQ
        (COND
          ((SPADCALL |a| (QREFELT $ 79))
            (LETT |a| (SPADCALL |a| |b| (QREFELT $ 80)) |INS-;invmod;3S;28|)))
        (LETT |c| |a| |INS-;invmod;3S;28|)
        (LETT |c1| (|spadConstant| $ 20) |INS-;invmod;3S;28|)
        (LETT |d| |b| |INS-;invmod;3S;28|)
        (LETT |d1| (|spadConstant| $ 9) |INS-;invmod;3S;28|)
        (SEQ G190
          (COND
            ((NULL
              (COND
                ((SPADCALL |d| (QREFELT $ 60)) (QUOTE NIL))
                ((QUOTE T) (QUOTE T))))
              (GO G191)))
          (SEQ
            (LETT |q| (SPADCALL |c| |d| (QREFELT $ 81)) |INS-;invmod;3S;28|)
            (LETT |r|
              (SPADCALL |c| (SPADCALL |q| |d| (QREFELT $ 82)) (QREFELT $ 61))
              |INS-;invmod;3S;28|)
            (LETT |r1|
```

```
              (SPADCALL |c1| (SPADCALL |q| |d1| (QREFELT $ 82)) (QREFELT $ 61))
              |INS-;invmod;3S;28|)
            (LETT |c| |d| |INS-;invmod;3S;28|)
            (LETT |c1| |d1| |INS-;invmod;3S;28|)
            (LETT |d| |r| |INS-;invmod;3S;28|)
            (EXIT (LETT |d1| |r1| |INS-;invmod;3S;28|)))
          NIL
          (GO G190)
          G191
          (EXIT NIL))
        (COND
          ((NULL (SPADCALL |c| (QREFELT $ 83)))
            (EXIT (|error| "inverse does not exist"))))
        (EXIT
          (COND
            ((SPADCALL |c1| (QREFELT $ 79)) (SPADCALL |c1| |b| (QREFELT $ 77)))
            ((QUOTE T) |c1|)))))))

(DEFUN |INS-;powmod;4S;29| (|x| |n| |p| $)
  (PROG (|y| #0=#:G1137 |z|)
    (RETURN
      (SEQ
        (EXIT
          (SEQ
            (COND
              ((SPADCALL |x| (QREFELT $ 79))
                (LETT |x|
                  (SPADCALL |x| |p| (QREFELT $ 80))
                  |INS-;powmod;4S;29|)))
            (EXIT
              (COND
                ((SPADCALL |x| (QREFELT $ 60)) (|spadConstant| $ 9))
                ((SPADCALL |n| (QREFELT $ 60)) (|spadConstant| $ 20))
                ((QUOTE T)
                  (SEQ
                    (LETT |y| (|spadConstant| $ 20) |INS-;powmod;4S;29|)
                    (LETT |z| |x| |INS-;powmod;4S;29|)
                    (EXIT
                      (SEQ G190
                        NIL
                        (SEQ
                          (COND
                            ((SPADCALL |n| (QREFELT $ 12))
                              (LETT |y|
                                (SPADCALL |y| |z| |p| (QREFELT $ 85))
                                |INS-;powmod;4S;29|)))
                          (EXIT
                            (COND
                              ((SPADCALL
                                (LETT |n|
```

```
                                (SPADCALL |n|
                                  (SPADCALL
                                    (|spadConstant| $ 20)
                                    (QREFELT $ 17))
                                  (QREFELT $ 18))
                                 |INS-;powmod;4S;29|)
                               (QREFELT $ 60))
                              (PROGN
                                (LETT #0# |y| |INS-;powmod;4S;29|)
                                (GO #0#)))
                             ((QUOTE T)
                               (LETT |z|
                                 (SPADCALL |z| |z| |p| (QREFELT $ 85))
                                 |INS-;powmod;4S;29|)))))
                      NIL
                      (GO G190)
                      G191
                      (EXIT NIL)))))))))
       #0#
       (EXIT #0#)))))

(DEFUN |IntegerNumberSystem&| (|#1|)
  (PROG (DV$1 |dv$| $ |pv$|)
    (RETURN
      (PROGN
        (LETT DV$1 (|devaluate| |#1|) . #0=(|IntegerNumberSystem&|))
        (LETT |dv$| (LIST (QUOTE |IntegerNumberSystem&|) DV$1) . #0#)
        (LETT $ (make-array 87) . #0#)
        (QSETREFV $ 0 |dv$|)
        (QSETREFV $ 3 (LETT |pv$| (|buildPredVector| 0 0 NIL) . #0#))
        (|stuffDomainSlots| $)
        (QSETREFV $ 6 |#1|)
        $))))

(setf (get
  (QUOTE |IntegerNumberSystem&|)
  (QUOTE |infovec|))
  (LIST
    (QUOTE
      #(NIL NIL NIL NIL NIL NIL
        (|local| |#1|)
        (|NonNegativeInteger|)
        |INS-;characteristic;Nni;1|
        (0 . |Zero|)
        |INS-;differentiate;2S;2|
        (|Boolean|)
        (4 . |odd?|)
        |INS-;even?;SB;3|
        (9 . <)
        |INS-;positive?;SB;4|
```

```
|INS-;copy;2S;5|
(15 . -)
(20 . |shift|)
|INS-;bit?;2SB;6|
(26 . |One|)
(30 . |dec|)
|INS-;mask;2S;7|
|INS-;rational?;SB;8|
(35 . =)
(|Integer|)
(41 . |convert|)
|INS-;euclideanSize;SNni;9|
(|Float|)
(46 . |coerce|)
|INS-;convert;SF;10|
(|DoubleFloat|)
|INS-;convert;SDf;11|
(|InputForm|)
(51 . |convert|)
|INS-;convert;SIf;12|
|INS-;retract;SI;13|
(|Pattern| 25)
(56 . |coerce|)
|INS-;convert;SP;14|
(|Factored| 6)
(|IntegerFactorizationPackage| 6)
(61 . |factor|)
(|Factored| $)
|INS-;factor;SF;15|
(66 . |squareFree|)
|INS-;squareFree;SF;16|
(|IntegerPrimesPackage| 6)
(71 . |prime?|)
|INS-;prime?;SB;17|
(|IntegerCombinatoricFunctions| 6)
(76 . |factorial|)
|INS-;factorial;2S;18|
(81 . |binomial|)
|INS-;binomial;3S;19|
(87 . |permutation|)
|INS-;permutation;3S;20|
(|Union| 25 (QUOTE "failed"))
|INS-;retractIfCan;SU;21|
|INS-;init;S;22|
(93 . |zero?|)
(98 . -)
(|Union| $ (QUOTE "failed"))
|INS-;nextItem;SU;23|
(|PatternMatchResult| 25 6)
(|PatternMatchIntegerNumberSystem| 6)
```

```
   (104 . |patternMatch|)
   (|PatternMatchResult| 25 $)
   |INS-;patternMatch;SP2Pmr;24|
   (|Fraction| 25)
   (111 . |coerce|)
   |INS-;rational;SF;25|
   (|Union| 69 (QUOTE "failed"))
   |INS-;rationalIfCan;SU;26|
   (116 . |rem|)
   (|PositiveInteger|)
   (122 . *)
   (128 . +)
   |INS-;symmetricRemainder;3S;27|
   (134 . |negative?|)
   (139 . |positiveRemainder|)
   (145 . |quo|)
   (151 . *)
   (157 . |one?|)
   |INS-;invmod;3S;28|
   (162 . |mulmod|)
   |INS-;powmod;4S;29|))
(QUOTE
  #(|symmetricRemainder| 169 |squareFree| 175 |retractIfCan| 180
    |retract| 185 |rationalIfCan| 190 |rational?| 195 |rational| 200
    |prime?| 205 |powmod| 210 |positive?| 217 |permutation| 222
    |patternMatch| 228 |nextItem| 235 |mask| 240 |invmod| 245 |init| 251
    |factorial| 255 |factor| 260 |even?| 265 |euclideanSize| 270
    |differentiate| 275 |copy| 280 |convert| 285 |characteristic| 305
    |bit?| 309 |binomial| 315))
(QUOTE NIL)
(CONS
  (|makeByteWordVec2| 1 (QUOTE NIL))
  (CONS
    (QUOTE #())
    (CONS
      (QUOTE #())
      (|makeByteWordVec2| 86
        (QUOTE
          (0 6 0 9 1 6 11 0 12 2 6 11 0 0 14 1 6 0 0 17 2 6 0 0 0 18 0 6
           0 20 1 6 0 0 21 2 6 11 0 0 24 1 6 25 0 26 1 28 0 25 29 1 33 0
           25 34 1 37 0 25 38 1 41 40 6 42 1 41 40 6 45 1 47 11 6 48 1 50
           6 6 51 2 50 6 6 6 53 2 50 6 6 6 55 1 6 11 0 60 2 6 0 0 0 61 3
           65 64 6 37 64 66 1 69 0 25 70 2 6 0 0 0 74 2 6 0 75 0 76 2 6 0
           0 0 77 1 6 11 0 79 2 6 0 0 0 80 2 6 0 0 0 81 2 6 0 0 0 82 1 6
           11 0 83 3 6 0 0 0 0 85 2 0 0 0 0 78 1 0 43 0 46 1 0 57 0 58 1
           0 25 0 36 1 0 72 0 73 1 0 11 0 23 1 0 69 0 71 1 0 11 0 49 3 0
           0 0 0 0 86 1 0 11 0 15 2 0 0 0 0 56 3 0 67 0 37 67 68 1 0 62
           0 63 1 0 0 0 22 2 0 0 0 0 84 0 0 0 59 1 0 0 0 52 1 0 43 0 44
           1 0 11 0 13 1 0 7 0 27 1 0 0 0 10 1 0 0 0 16 1 0 31 0 32 1 0
           28 0 30 1 0 37 0 39 1 0 33 0 35 0 0 7 8 2 0 11 0 0 19 2 0 0
```

```
              0 0 54))))))
    (QUOTE |lookupComplete|)))
```

———-

21.31 INTDOM.lsp BOOTSTRAP

INTDOM depends on itself. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **INTDOM** category which we can write into the **MID** directory. We compile the lisp code and copy the **INTDOM.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— INTDOM.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |IntegralDomain;AL| (QUOTE NIL))

(DEFUN |IntegralDomain| NIL
  (LET (#:G83060)
    (COND
      (|IntegralDomain;AL|)
      (T (SETQ |IntegralDomain;AL| (|IntegralDomain;|))))))

(DEFUN |IntegralDomain;| NIL
  (PROG (#1=#:G83058)
    (RETURN
      (PROG1
        (LETT #1#
          (|Join|
            (|CommutativeRing|)
            (|Algebra| (QUOTE |$|))
            (|EntireRing|)
            (|mkCategory|
              (QUOTE |domain|)
              (QUOTE (
                ((|exquo| ((|Union| |$| "failed") |$| |$|)) T)
                ((|unitNormal|
                  ((|Record|
                     (|:| |unit| |$|)
                     (|:| |canonical| |$|)
                     (|:| |associate| |$|)) |$|)) T)
                ((|unitCanonical| (|$| |$|)) T)
                ((|associates?| ((|Boolean|) |$| |$|)) T)
```

```
               ((|unit?| ((|Boolean|) |$|)) T)))
              NIL
              (QUOTE ((|Boolean|)))
              NIL))
           |IntegralDomain|)
        (SETELT #1# 0 (QUOTE (|IntegralDomain|)))))))

(setf (get (QUOTE |IntegralDomain|) (QUOTE NILADIC)) T)
```

———-

21.32 INTDOM-.lsp BOOTSTRAP

INTDOM- depends on **INTDOM**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **INTDOM-** category which we can write into the **MID** directory. We compile the lisp code and copy the **INTDOM-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— INTDOM-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |INTDOM-;unitNormal;SR;1| (|x| |$|)
  (VECTOR (|spadConstant| |$| 7) |x| (|spadConstant| |$| 7)))

(DEFUN |INTDOM-;unitCanonical;2S;2| (|x| |$|)
  (QVELT (SPADCALL |x| (QREFELT |$| 10)) 1))

(DEFUN |INTDOM-;recip;SU;3| (|x| |$|)
  (COND
    ((SPADCALL |x| (QREFELT |$| 13)) (CONS 1 "failed"))
    ((QUOTE T) (SPADCALL (|spadConstant| |$| 7) |x| (QREFELT |$| 15)))))

(DEFUN |INTDOM-;unit?;SB;4| (|x| |$|)
  (COND
    ((QEQCAR (SPADCALL |x| (QREFELT |$| 17)) 1) (QUOTE NIL))
    ((QUOTE T) (QUOTE T))))

(DEFUN |INTDOM-;associates?;2SB;5| (|x| |y| |$|)
  (SPADCALL
    (QVELT (SPADCALL |x| (QREFELT |$| 10)) 1)
    (QVELT (SPADCALL |y| (QREFELT |$| 10)) 1)
    (QREFELT |$| 19)))
```

```
(DEFUN |INTDOM-;associates?;2SB;6| (|x| |y| |$|)
  (COND
    ((SPADCALL |x| (QREFELT |$| 13)) (SPADCALL |y| (QREFELT |$| 13)))
    ((OR
        (SPADCALL |y| (QREFELT |$| 13))
        (OR
          (QEQCAR (SPADCALL |x| |y| (QREFELT |$| 15)) 1)
          (QEQCAR (SPADCALL |y| |x| (QREFELT |$| 15)) 1)))
      (QUOTE NIL))
    ((QUOTE T) (QUOTE T))))

(DEFUN |IntegralDomain&| (|#1|)
  (PROG (|DV$1| |dv$| |$| |pv$|)
    (RETURN
      (PROGN
        (LETT |DV$1| (|devaluate| |#1|) . #1=(|IntegralDomain&|))
        (LETT |dv$| (LIST (QUOTE |IntegralDomain&|) |DV$1|) . #1#)
        (LETT |$| (make-array 21) . #1#)
        (QSETREFV |$| 0 |dv$|)
        (QSETREFV |$| 3 (LETT |pv$| (|buildPredVector| 0 0 NIL) . #1#))
        (|stuffDomainSlots| |$|)
        (QSETREFV |$| 6 |#1|)
        (COND
          ((|HasCategory| |#1| (QUOTE (|Field|))))
          ((QUOTE T)
            (QSETREFV |$| 9
              (CONS (|dispatchFunction| |INTDOM-;unitNormal;SR;1|) |$|))))
        (COND
          ((|HasAttribute| |#1| (QUOTE |canonicalUnitNormal|))
            (QSETREFV |$| 20
              (CONS (|dispatchFunction| |INTDOM-;associates?;2SB;5|) |$|)))
          ((QUOTE T)
            (QSETREFV |$| 20
              (CONS (|dispatchFunction| |INTDOM-;associates?;2SB;6|) |$|))))
        |$|))))

(setf (get
  (QUOTE |IntegralDomain&|)
  (QUOTE |infovec|))
  (LIST
    (QUOTE
      #(NIL NIL NIL NIL NIL NIL
        (|local| |#1|)
        (0 . |One|)
        (|Record|
          (|:| |unit| |$|)
          (|:| |canonical| |$|)
          (|:| |associate| |$|))
        (4 . |unitNormal|)
        (9 . |unitNormal|)
```

```
      |INTDOM-;unitCanonical;2S;2|
      (|Boolean|)
      (14 . |zero?|)
      (|Union| |$| (QUOTE "failed"))
      (19 . |exquo|)
      |INTDOM-;recip;SU;3|
      (25 . |recip|)
      |INTDOM-;unit?;SB;4|
      (30 . |=|)
      (36 . |associates?|)))
   (QUOTE
     #(|unitNormal| 42 |unitCanonical| 47 |unit?| 52 |recip| 57
       |associates?| 62))
   (QUOTE NIL)
   (CONS
     (|makeByteWordVec2| 1 (QUOTE NIL))
     (CONS
       (QUOTE #())
       (CONS
         (QUOTE #())
         (|makeByteWordVec2| 20
           (QUOTE
             (0 6 0 7 1 0 8 0 9 1 6 8 0 10 1 6 12 0 13 2 6 14 0 0 15 1 6 14
              0 17 2 6 12 0 0 19 2 0 12 0 0 20 1 0 8 0 9 1 0 0 0 11 1 0 12 0
              18 1 0 14 0 16 2 0 12 0 0 20))))))
  (QUOTE |lookupComplete|)))
```

———-

21.33 LNAGG.lsp BOOTSTRAP

LNAGG depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **LNAGG** category which we can write into the **MID** directory. We compile the lisp code and copy the **LNAGG.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— LNAGG.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |LinearAggregate;CAT| (QUOTE NIL))

(SETQ |LinearAggregate;AL| (QUOTE NIL))
```

```
(DEFUN |LinearAggregate| (#1=#:G85818)
 (LET (#2=#:G85819)
  (COND
   ((SETQ #2# (|assoc| (|devaluate| #1#) |LinearAggregate;AL|)) (CDR #2#))
   (T
    (SETQ |LinearAggregate;AL|
     (|cons5|
      (CONS (|devaluate| #1#) (SETQ #2# (|LinearAggregate;| #1#)))
      |LinearAggregate;AL|))
    #2#))))

(DEFUN |LinearAggregate;| (|t#1|)
 (PROG (#1=#:G85817)
  (RETURN
   (PROG1
    (LETT #1#
     (|sublisV|
      (PAIR (QUOTE (|t#1|)) (LIST (|devaluate| |t#1|)))
      (|sublisV|
       (PAIR (QUOTE (#2=#:G85816)) (LIST (QUOTE (|Integer|))))
       (COND
        (|LinearAggregate;CAT|)
        ((QUOTE T)
         (LETT |LinearAggregate;CAT|
          (|Join|
           (|IndexedAggregate| (QUOTE #2#) (QUOTE |t#1|))
           (|Collection| (QUOTE |t#1|))
           (|mkCategory|
            (QUOTE |domain|)
            (QUOTE (
             ((|new| (|$| (|NonNegativeInteger|) |t#1|)) T)
             ((|concat| (|$| |$| |t#1|)) T)
             ((|concat| (|$| |t#1| |$|)) T)
             ((|concat| (|$| |$| |$|)) T)
             ((|concat| (|$| (|List| |$|))) T)
             ((|map| (|$| (|Mapping| |t#1| |t#1| |t#1|) |$| |$|)) T)
             ((|elt| (|$| |$| (|UniversalSegment| (|Integer|)))) T)
             ((|delete| (|$| |$| (|Integer|))) T)
             ((|delete| (|$| |$| (|UniversalSegment| (|Integer|)))) T)
             ((|insert| (|$| |t#1| |$| (|Integer|))) T)
             ((|insert| (|$| |$| |$| (|Integer|))) T)
             ((|setelt| (|t#1| |$| (|UniversalSegment| (|Integer|)) |t#1|))
              (|has| |$| (ATTRIBUTE |shallowlyMutable|)))))
            NIL
            (QUOTE (
              (|UniversalSegment| (|Integer|))
              (|Integer|)
              (|List| |$|)
              (|NonNegativeInteger|)))
             NIL))
```

```
          . #3=(|LinearAggregate|))))))
       . #3#)
     (SETELT #1# 0 (LIST (QUOTE |LinearAggregate|) (|devaluate| |t#1|)))))))
```

———-

21.34 LNAGG-.lsp BOOTSTRAP

LNAGG- depends on **LNAGG**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **LNAGG-** category which we can write into the **MID** directory. We compile the lisp code and copy the **LNAGG-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— LNAGG-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |LNAGG-;indices;AL;1| (|a| |$|)
 (PROG (#1=#:G85833 |i| #2=#:G85834)
  (RETURN
   (SEQ
    (PROGN
     (LETT #1# NIL |LNAGG-;indices;AL;1|)
     (SEQ
      (LETT |i| (SPADCALL |a| (QREFELT |$| 9)) |LNAGG-;indices;AL;1|)
      (LETT #2# (SPADCALL |a| (QREFELT |$| 10)) |LNAGG-;indices;AL;1|)
      G190
      (COND ((|>| |i| #2#) (GO G191)))
      (SEQ (EXIT (LETT #1# (CONS |i| #1#) |LNAGG-;indices;AL;1|)))
      (LETT |i| (|+| |i| 1) |LNAGG-;indices;AL;1|)
      (GO G190)
      G191
      (EXIT (NREVERSE0 #1#)))))))))

(DEFUN |LNAGG-;index?;IAB;2| (|i| |a| |$|)
 (COND
  ((OR
     (|<| |i| (SPADCALL |a| (QREFELT |$| 9)))
     (|<| (SPADCALL |a| (QREFELT |$| 10)) |i|))
    (QUOTE NIL))
  ((QUOTE T) (QUOTE T))))

(DEFUN |LNAGG-;concat;ASA;3| (|a| |x| |$|)
 (SPADCALL |a| (SPADCALL 1 |x| (QREFELT |$| 16)) (QREFELT |$| 17)))
```

```
(DEFUN |LNAGG-;concat;S2A;4| (|x| |y| |$|)
 (SPADCALL (SPADCALL 1 |x| (QREFELT |$| 16)) |y| (QREFELT |$| 17)))

(DEFUN |LNAGG-;insert;SAIA;5| (|x| |a| |i| |$|)
 (SPADCALL (SPADCALL 1 |x| (QREFELT |$| 16)) |a| |i| (QREFELT |$| 20)))

(DEFUN |LNAGG-;maxIndex;AI;6| (|l| |$|)
 (|+| (|-| (SPADCALL |l| (QREFELT |$| 22)) 1) (SPADCALL |l| (QREFELT |$| 9))))

(DEFUN |LinearAggregate&| (|#1| |#2|)
 (PROG (|DV$1| |DV$2| |dv$| |$| |pv$|)
  (RETURN
   (PROGN
    (LETT |DV$1| (|devaluate| |#1|) . #1=(|LinearAggregate&|))
    (LETT |DV$2| (|devaluate| |#2|) . #1#)
    (LETT |dv$| (LIST (QUOTE |LinearAggregate&|) |DV$1| |DV$2|) . #1#)
    (LETT |$| (make-array 25) . #1#)
    (QSETREFV |$| 0 |dv$|)
    (QSETREFV |$| 3
     (LETT |pv$|
      (|buildPredVector| 0 0
       (LIST (|HasAttribute| |#1| (QUOTE |shallowlyMutable|))))
      . #1#))
    (|stuffDomainSlots| |$|)
    (QSETREFV |$| 6 |#1|)
    (QSETREFV |$| 7 |#2|)
    (COND
     ((|HasAttribute| |#1| (QUOTE |finiteAggregate|))
      (QSETREFV |$| 23
       (CONS (|dispatchFunction| |LNAGG-;maxIndex;AI;6|) |$|))))
    |$|))))

(setf (get
 (QUOTE |LinearAggregate&|)
 (QUOTE |infovec|))
 (LIST
  (QUOTE #(NIL NIL NIL NIL NIL NIL (|local| |#1|) (|local| |#2|) (|Integer|)
    (0 . |minIndex|) (5 . |maxIndex|) (|List| 8) |LNAGG-;indices;AL;1|
    (|Boolean|) |LNAGG-;index?;IAB;2| (|NonNegativeInteger|) (10 . |new|)
    (16 . |concat|) |LNAGG-;concat;ASA;3| |LNAGG-;concat;S2A;4|
    (22 . |insert|) |LNAGG-;insert;SAIA;5| (29 . |#|) (34 . |maxIndex|)
    (|List| |$|)))
  (QUOTE #(|maxIndex| 39 |insert| 44 |indices| 51 |index?| 56 |concat| 62))
  (QUOTE NIL)
  (CONS
   (|makeByteWordVec2| 1 (QUOTE NIL))
   (CONS
    (QUOTE #())
    (CONS
     (QUOTE #())
```

```
    (|makeByteWordVec2| 23 (QUOTE (1 6 8 0 9 1 6 8 0 10 2 6 0 15 7
     16 2 6 0 0 0 17 3 6 0 0 0 8 20 1 6 15 0 22 1 0 8 0 23 1 0 8 0 23 3 0
     0 7 0 8 21 1 0 11 0 12 2 0 13 8 0 14 2 0 0 0 7 18 2 0 0 7 0 19))))))
  (QUOTE |lookupComplete|)))
```

———-

21.35 LSAGG.lsp BOOTSTRAP

LSAGG depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **LSAGG** category which we can write into the **MID** directory. We compile the lisp code and copy the **LSAGG.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— LSAGG.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |ListAggregate;CAT| (QUOTE NIL))

(SETQ |ListAggregate;AL| (QUOTE NIL))

(DEFUN |ListAggregate| (#1=#:G87500)
 (LET (#2=#:G87501)
  (COND
   ((SETQ #2# (|assoc| (|devaluate| #1#) |ListAggregate;AL|)) (CDR #2#))
   (T
    (SETQ |ListAggregate;AL|
     (|cons5|
      (CONS (|devaluate| #1#) (SETQ #2# (|ListAggregate;| #1#)))
      |ListAggregate;AL|))
    #2#))))

(DEFUN |ListAggregate;| (|t#1|)
 (PROG (#1=#:G87499)
  (RETURN
   (PROG1
    (LETT #1#
     (|sublisV|
      (PAIR (QUOTE (|t#1|)) (LIST (|devaluate| |t#1|)))
      (COND
       (|ListAggregate;CAT|)
       ((QUOTE T)
        (LETT |ListAggregate;CAT|
         (|Join|
```

```
          (|StreamAggregate| (QUOTE |t#1|))
          (|FiniteLinearAggregate| (QUOTE |t#1|))
          (|ExtensibleLinearAggregate| (QUOTE |t#1|))
          (|mkCategory|
           (QUOTE |domain|)
           (QUOTE ((((|list| (|$| |t#1|)) T)))
           NIL
           (QUOTE NIL)
            NIL))
          . #2=(|ListAggregate|)))))
     . #2#)
    (SETELT #1# 0 (LIST (QUOTE |ListAggregate|) (|devaluate| |t#1|)))))))
```

———-

21.36 LSAGG-.lsp BOOTSTRAP

LSAGG- depends on **LSAGG**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **LSAGG-** category which we can write into the **MID** directory. We compile the lisp code and copy the **LSAGG-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— LSAGG-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |LSAGG-;sort!;M2A;1| (|f| |l| |$|)
 (|LSAGG-;mergeSort| |f| |l| (SPADCALL |l| (QREFELT |$| 9)) |$|))

(DEFUN |LSAGG-;list;SA;2| (|x| |$|)
 (SPADCALL |x| (SPADCALL (QREFELT |$| 12)) (QREFELT |$| 13)))

(DEFUN |LSAGG-;reduce;MAS;3| (|f| |x| |$|)
 (COND
  ((SPADCALL |x| (QREFELT |$| 16))
    (|error| "reducing over an empty list needs the 3 argument form"))
  ((QUOTE T)
    (SPADCALL |f|
     (SPADCALL |x| (QREFELT |$| 17))
     (SPADCALL |x| (QREFELT |$| 18))
     (QREFELT |$| 20)))))

(DEFUN |LSAGG-;merge;M3A;4| (|f| |p| |q| |$|)
 (SPADCALL |f|
  (SPADCALL |p| (QREFELT |$| 22))
```

```
  (SPADCALL |q| (QREFELT |$| 22))
  (QREFELT |$| 23)))

(DEFUN |LSAGG-;select!;M2A;5| (|f| |x| |$|)
 (PROG (|y| |z|)
  (RETURN
   (SEQ
    (SEQ G190
     (COND
      ((NULL
        (COND
         ((OR
           (SPADCALL |x| (QREFELT |$| 16))
           (SPADCALL (SPADCALL |x| (QREFELT |$| 18)) |f|))
          (QUOTE NIL))
         ((QUOTE T) (QUOTE T))))
       (GO G191)))
     (SEQ
      (EXIT
       (LETT |x| (SPADCALL |x| (QREFELT |$| 17)) |LSAGG-;select!;M2A;5|)))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT
     (COND
      ((SPADCALL |x| (QREFELT |$| 16)) |x|)
      ((QUOTE T)
       (SEQ
        (LETT |y| |x| |LSAGG-;select!;M2A;5|)
        (LETT |z| (SPADCALL |y| (QREFELT |$| 17)) |LSAGG-;select!;M2A;5|)
        (SEQ
         G190
         (COND
          ((NULL
            (COND
             ((SPADCALL |z| (QREFELT |$| 16)) (QUOTE NIL))
             ((QUOTE T) (QUOTE T))))
           (GO G191)))
          (SEQ
           (EXIT
            (COND
             ((SPADCALL (SPADCALL |z| (QREFELT |$| 18)) |f|)
               (SEQ
                (LETT |y| |z| |LSAGG-;select!;M2A;5|)
                (EXIT
                 (LETT |z|
                  (SPADCALL |z| (QREFELT |$| 17))
                  |LSAGG-;select!;M2A;5|))))
             ((QUOTE T)
```

```
            (SEQ
             (LETT |z| (SPADCALL |z| (QREFELT |$| 17)) |LSAGG-;select!;M2A;5|)
             (EXIT (SPADCALL |y| |z| (QREFELT |$| 25))))))))
         NIL
         (GO G190)
         G191
         (EXIT NIL))
        (EXIT |x|)))))))))

(DEFUN |LSAGG-;merge!;M3A;6| (|f| |p| |q| |$|)
 (PROG (|r| |t|)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |p| (QREFELT |$| 16)) |q|)
     ((SPADCALL |q| (QREFELT |$| 16)) |p|)
     ((SPADCALL |p| |q| (QREFELT |$| 28))
       (|error| "cannot merge a list into itself"))
     ((QUOTE T)
      (SEQ
       (COND
        ((SPADCALL
          (SPADCALL |p| (QREFELT |$| 18))
          (SPADCALL |q| (QREFELT |$| 18))
          |f|)
         (SEQ
          (LETT |r| (LETT |t| |p| |LSAGG-;merge!;M3A;6|) |LSAGG-;merge!;M3A;6|)
          (EXIT
           (LETT |p| (SPADCALL |p| (QREFELT |$| 17)) |LSAGG-;merge!;M3A;6|))))
        ((QUOTE T)
         (SEQ
          (LETT |r| (LETT |t| |q| |LSAGG-;merge!;M3A;6|) |LSAGG-;merge!;M3A;6|)
          (EXIT
           (LETT |q| (SPADCALL |q| (QREFELT |$| 17)) |LSAGG-;merge!;M3A;6|)))))
       (SEQ
        G190
        (COND
         ((NULL
            (COND
             ((OR
               (SPADCALL |p| (QREFELT |$| 16))
               (SPADCALL |q| (QREFELT |$| 16)))
              (QUOTE NIL))
             ((QUOTE T) (QUOTE T))))
          (GO G191)))
        (SEQ
         (EXIT
          (COND
           ((SPADCALL
              (SPADCALL |p| (QREFELT |$| 18))
```

```
              (SPADCALL |q| (QREFELT |$| 18))
              |f|)
             (SEQ
              (SPADCALL |t| |p| (QREFELT |$| 25))
              (LETT |t| |p| |LSAGG-;merge!;M3A;6|)
              (EXIT
               (LETT |p|
                (SPADCALL |p| (QREFELT |$| 17))
                |LSAGG-;merge!;M3A;6|))))
            ((QUOTE T)
             (SEQ
              (SPADCALL |t| |q| (QREFELT |$| 25))
              (LETT |t| |q| |LSAGG-;merge!;M3A;6|)
              (EXIT
               (LETT |q|
                (SPADCALL |q| (QREFELT |$| 17))
                |LSAGG-;merge!;M3A;6|)))))))
        NIL
        (GO G190)
        G191
        (EXIT NIL))
       (SPADCALL |t|
        (COND
         ((SPADCALL |p| (QREFELT |$| 16)) |q|)
         ((QUOTE T) |p|))
        (QREFELT |$| 25))
       (EXIT |r|)))))))))

(DEFUN |LSAGG-;insert!;SAIA;7| (|s| |x| |i| |$|)
 (PROG (|m| #1=#:G87547 |y| |z|)
  (RETURN
   (SEQ
    (LETT |m| (SPADCALL |x| (QREFELT |$| 31)) |LSAGG-;insert!;SAIA;7|)
    (EXIT
     (COND
      ((|<| |i| |m|) (|error| "index out of range"))
      ((EQL |i| |m|) (SPADCALL |s| |x| (QREFELT |$| 13)))
      ((QUOTE T)
       (SEQ
        (LETT |y|
         (SPADCALL |x|
          (PROG1
           (LETT #1# (|-| (|-| |i| 1) |m|) |LSAGG-;insert!;SAIA;7|)
           (|check-subtype| (|>=| #1# 0) (QUOTE (|NonNegativeInteger|)) #1#))
          (QREFELT |$| 32))
         |LSAGG-;insert!;SAIA;7|)
        (LETT |z| (SPADCALL |y| (QREFELT |$| 17)) |LSAGG-;insert!;SAIA;7|)
        (SPADCALL |y| (SPADCALL |s| |z| (QREFELT |$| 13)) (QREFELT |$| 25))
        (EXIT |x|)))))))))
```

```
(DEFUN |LSAGG-;insert!;2AIA;8| (|w| |x| |i| |$|)
 (PROG (|m| #1=#:G87551 |y| |z|)
  (RETURN
   (SEQ
    (LETT |m| (SPADCALL |x| (QREFELT |$| 31)) |LSAGG-;insert!;2AIA;8|)
    (EXIT
     (COND
      ((|<| |i| |m|) (|error| "index out of range"))
      ((EQL |i| |m|) (SPADCALL |w| |x| (QREFELT |$| 34)))
      ((QUOTE T)
       (SEQ
        (LETT |y|
         (SPADCALL |x|
          (PROG1
           (LETT #1# (|-| (|-| |i| 1) |m|) |LSAGG-;insert!;2AIA;8|)
           (|check-subtype| (|>=| #1# 0) (QUOTE (|NonNegativeInteger|)) #1#))
          (QREFELT |$| 32))
         |LSAGG-;insert!;2AIA;8|)
        (LETT |z| (SPADCALL |y| (QREFELT |$| 17)) |LSAGG-;insert!;2AIA;8|)
        (SPADCALL |y| |w| (QREFELT |$| 25))
        (SPADCALL |y| |z| (QREFELT |$| 34))
        (EXIT |x|)))))))))

(DEFUN |LSAGG-;remove!;M2A;9| (|f| |x| |$|)
 (PROG (|p| |q|)
  (RETURN
   (SEQ
    (SEQ
     G190
     (COND
      ((NULL
        (COND
         ((SPADCALL |x| (QREFELT |$| 16)) (QUOTE NIL))
         ((QUOTE T) (SPADCALL (SPADCALL |x| (QREFELT |$| 18)) |f|))))
       (GO G191)))
     (SEQ
      (EXIT
       (LETT |x| (SPADCALL |x| (QREFELT |$| 17)) |LSAGG-;remove!;M2A;9|)))
      NIL
      (GO G190)
      G191
      (EXIT NIL))
    (EXIT
     (COND
      ((SPADCALL |x| (QREFELT |$| 16)) |x|)
      ((QUOTE T)
       (SEQ
        (LETT |p| |x| |LSAGG-;remove!;M2A;9|)
        (LETT |q| (SPADCALL |x| (QREFELT |$| 17)) |LSAGG-;remove!;M2A;9|)
        (SEQ
```

```
         G190
         (COND
          ((NULL
             (COND
              ((SPADCALL |q| (QREFELT |$| 16)) (QUOTE NIL))
              ((QUOTE T) (QUOTE T))))
            (GO G191)))
         (SEQ
          (EXIT
           (COND
            ((SPADCALL (SPADCALL |q| (QREFELT |$| 18)) |f|)
              (LETT |q|
               (SPADCALL |p| (SPADCALL |q| (QREFELT |$| 17)) (QREFELT |$| 25))
               |LSAGG-;remove!;M2A;9|))
            ((QUOTE T)
              (SEQ
               (LETT |p| |q| |LSAGG-;remove!;M2A;9|)
               (EXIT
                (LETT |q|
                  (SPADCALL |q| (QREFELT |$| 17))
                  |LSAGG-;remove!;M2A;9|)))))))
         NIL
         (GO G190)
         G191
         (EXIT NIL))
        (EXIT |x|)))))))))

(DEFUN |LSAGG-;delete!;AIA;10| (|x| |i| |$|)
 (PROG (|m| #1=#:G87564 |y|)
  (RETURN
   (SEQ
    (LETT |m| (SPADCALL |x| (QREFELT |$| 31)) |LSAGG-;delete!;AIA;10|)
    (EXIT
     (COND
      ((|<| |i| |m|) (|error| "index out of range"))
      ((EQL |i| |m|) (SPADCALL |x| (QREFELT |$| 17)))
      ((QUOTE T)
       (SEQ
        (LETT |y|
         (SPADCALL |x|
          (PROG1
           (LETT #1# (|-| (|-| |i| 1) |m|) |LSAGG-;delete!;AIA;10|)
           (|check-subtype| (|>=| #1# 0) (QUOTE (|NonNegativeInteger|)) #1#))
          (QREFELT |$| 32))
         |LSAGG-;delete!;AIA;10|)
        (SPADCALL |y| (SPADCALL |y| 2 (QREFELT |$| 32)) (QREFELT |$| 25))
        (EXIT |x|)))))))))

(DEFUN |LSAGG-;delete!;AUsA;11| (|x| |i| |$|)
 (PROG (|l| |m| |h| #1=#:G87569 #2=#:G87570 |t| #3=#:G87571)
```

```
  (RETURN
   (SEQ
    (LETT |l| (SPADCALL |i| (QREFELT |$| 39)) |LSAGG-;delete!;AUsA;11|)
    (LETT |m| (SPADCALL |x| (QREFELT |$| 31)) |LSAGG-;delete!;AUsA;11|)
    (EXIT
     (COND
      ((|<| |l| |m|) (|error| "index out of range"))
      ((QUOTE T)
       (SEQ
        (LETT |h|
         (COND
          ((SPADCALL |i| (QREFELT |$| 40)) (SPADCALL |i| (QREFELT |$| 41)))
          ((QUOTE T) (SPADCALL |x| (QREFELT |$| 42))))
         |LSAGG-;delete!;AUsA;11|)
        (EXIT
         (COND
          ((|<| |h| |l|) |x|)
          ((EQL |l| |m|)
           (SPADCALL |x|
            (PROG1
             (LETT #1# (|-| (|+| |h| 1) |m|) |LSAGG-;delete!;AUsA;11|)
             (|check-subtype| (|>=| #1# 0) (QUOTE (|NonNegativeInteger|)) #1#))
            (QREFELT |$| 32)))
          ((QUOTE T)
           (SEQ
            (LETT |t|
             (SPADCALL |x|
              (PROG1
               (LETT #2# (|-| (|-| |l| 1) |m|) |LSAGG-;delete!;AUsA;11|)
               (|check-subtype| (|>=| #2# 0)
                 (QUOTE (|NonNegativeInteger|)) #2#))
              (QREFELT |$| 32))
             |LSAGG-;delete!;AUsA;11|)
            (SPADCALL |t|
             (SPADCALL |t|
              (PROG1
               (LETT #3# (|+| (|-| |h| |l|) 2) |LSAGG-;delete!;AUsA;11|)
               (|check-subtype| (|>=| #3# 0)
                (QUOTE (|NonNegativeInteger|)) #3#))
              (QREFELT |$| 32))
             (QREFELT |$| 25))
            (EXIT |x|)))))))))))))

(DEFUN |LSAGG-;find;MAU;12| (|f| |x| |$|)
 (SEQ
  (SEQ
   G190
   (COND
    ((NULL
     (COND
```

```
      ((OR
         (SPADCALL |x| (QREFELT |$| 16))
         (SPADCALL (SPADCALL |x| (QREFELT |$| 18)) |f|))
        (QUOTE NIL))
      ((QUOTE T) (QUOTE T))))
     (GO G191)))
   (SEQ (EXIT (LETT |x| (SPADCALL |x| (QREFELT |$| 17)) |LSAGG-;find;MAU;12|)))
   NIL
   (GO G190)
   G191
   (EXIT NIL))
  (EXIT
   (COND
    ((SPADCALL |x| (QREFELT |$| 16)) (CONS 1 "failed"))
    ((QUOTE T) (CONS 0 (SPADCALL |x| (QREFELT |$| 18))))))))

(DEFUN |LSAGG-;position;MAI;13| (|f| |x| |$|)
 (PROG (|k|)
  (RETURN
   (SEQ
    (SEQ
     (LETT |k| (SPADCALL |x| (QREFELT |$| 31)) |LSAGG-;position;MAI;13|)
     G190
     (COND
      ((NULL
        (COND
         ((OR
           (SPADCALL |x| (QREFELT |$| 16))
           (SPADCALL (SPADCALL |x| (QREFELT |$| 18)) |f|))
          (QUOTE NIL))
         ((QUOTE T) (QUOTE T))))
       (GO G191)))
     (SEQ
      (EXIT
       (LETT |x| (SPADCALL |x| (QREFELT |$| 17)) |LSAGG-;position;MAI;13|)))
     (LETT |k| (|+| |k| 1) |LSAGG-;position;MAI;13|)
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT
     (COND
      ((SPADCALL |x| (QREFELT |$| 16)) (|-| (SPADCALL |x| (QREFELT |$| 31)) 1))
      ((QUOTE T) |k|)))))))

(DEFUN |LSAGG-;mergeSort| (|f| |p| |n| |$|)
 (PROG (#1=#:G87593 |l| |q|)
  (RETURN
   (SEQ
    (COND
     ((EQL |n| 2)
```

```
      (COND
       ((SPADCALL
         (SPADCALL (SPADCALL |p| (QREFELT |$| 17)) (QREFELT |$| 18))
         (SPADCALL |p| (QREFELT |$| 18)) |f|)
        (LETT |p| (SPADCALL |p| (QREFELT |$| 47)) |LSAGG-;mergeSort|))))
     (EXIT
      (COND
       ((|<| |n| 3) |p|)
       ((QUOTE T)
        (SEQ
         (LETT |l|
          (PROG1
           (LETT #1# (QUOTIENT2 |n| 2) |LSAGG-;mergeSort|)
           (|check-subtype| (|>=| #1# 0) (QUOTE (|NonNegativeInteger|)) #1#))
          |LSAGG-;mergeSort|)
         (LETT |q| (SPADCALL |p| |l| (QREFELT |$| 48)) |LSAGG-;mergeSort|)
         (LETT |p| (|LSAGG-;mergeSort| |f| |p| |l| |$|) |LSAGG-;mergeSort|)
         (LETT |q|
           (|LSAGG-;mergeSort| |f| |q| (|-| |n| |l|) |$|)
           |LSAGG-;mergeSort|)
         (EXIT (SPADCALL |f| |p| |q| (QREFELT |$| 23)))))))))

(DEFUN |LSAGG-;sorted?;MAB;15| (|f| |l| |$|)
 (PROG (#1=#:G87603 |p|)
  (RETURN
   (SEQ
    (EXIT
     (COND
      ((SPADCALL |l| (QREFELT |$| 16)) (QUOTE T))
      ((QUOTE T)
       (SEQ
        (LETT |p| (SPADCALL |l| (QREFELT |$| 17)) |LSAGG-;sorted?;MAB;15|)
        (SEQ
         G190
         (COND
          ((NULL
            (COND
             ((SPADCALL |p| (QREFELT |$| 16)) (QUOTE NIL))
             ((QUOTE T) (QUOTE T))))
           (GO G191)))
         (SEQ
          (EXIT
           (COND
            ((NULL
              (SPADCALL
               (SPADCALL |l| (QREFELT |$| 18))
               (SPADCALL |p| (QREFELT |$| 18))
               |f|))
             (PROGN (LETT #1# (QUOTE NIL) |LSAGG-;sorted?;MAB;15|) (GO #1#)))
            ((QUOTE T)
```

```
            (LETT |p|
             (SPADCALL
              (LETT |l| |p| |LSAGG-;sorted?;MAB;15|)
              (QREFELT |$| 17))
             |LSAGG-;sorted?;MAB;15|)))))
          NIL
          (GO G190)
          G191
          (EXIT NIL))
         (EXIT (QUOTE T))))))
    #1#
    (EXIT #1#)))))

(DEFUN |LSAGG-;reduce;MA2S;16| (|f| |x| |i| |$|)
 (PROG (|r|)
  (RETURN
   (SEQ
    (LETT |r| |i| |LSAGG-;reduce;MA2S;16|)
    (SEQ
     G190
     (COND
      ((NULL
        (COND
         ((SPADCALL |x| (QREFELT |$| 16)) (QUOTE NIL))
         ((QUOTE T) (QUOTE T))))
       (GO G191)))
     (SEQ
      (LETT |r|
       (SPADCALL |r| (SPADCALL |x| (QREFELT |$| 18)) |f|)
       |LSAGG-;reduce;MA2S;16|)
      (EXIT
       (LETT |x| (SPADCALL |x| (QREFELT |$| 17)) |LSAGG-;reduce;MA2S;16|)))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT |r|)))))

(DEFUN |LSAGG-;reduce;MA3S;17| (|f| |x| |i| |a| |$|)
 (PROG (|r|)
  (RETURN
   (SEQ
    (LETT |r| |i| |LSAGG-;reduce;MA3S;17|)
    (SEQ
     G190
     (COND
      ((NULL
        (COND
         ((OR
            (SPADCALL |x| (QREFELT |$| 16))
```

```
            (SPADCALL |r| |a| (QREFELT |$| 51)))
           (QUOTE NIL))
         ((QUOTE T) (QUOTE T)))) (GO G191)))
     (SEQ
      (LETT |r|
       (SPADCALL |r| (SPADCALL |x| (QREFELT |$| 18)) |f|)
       |LSAGG-;reduce;MA3S;17|)
      (EXIT
       (LETT |x| (SPADCALL |x| (QREFELT |$| 17)) |LSAGG-;reduce;MA3S;17|)))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT |r|)))))

(DEFUN |LSAGG-;new;NniSA;18| (|n| |s| |$|)
 (PROG (|k| |l|)
  (RETURN
   (SEQ
    (LETT |l| (SPADCALL (QREFELT |$| 12)) |LSAGG-;new;NniSA;18|)
    (SEQ
     (LETT |k| 1 |LSAGG-;new;NniSA;18|)
     G190
     (COND ((QSGREATERP |k| |n|) (GO G191)))
     (SEQ
      (EXIT
       (LETT |l| (SPADCALL |s| |l| (QREFELT |$| 13)) |LSAGG-;new;NniSA;18|)))
     (LETT |k| (QSADD1 |k|) |LSAGG-;new;NniSA;18|)
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT |l|)))))

(DEFUN |LSAGG-;map;M3A;19| (|f| |x| |y| |$|)
 (PROG (|z|)
  (RETURN
   (SEQ
    (LETT |z| (SPADCALL (QREFELT |$| 12)) |LSAGG-;map;M3A;19|)
    (SEQ
     G190
     (COND
      ((NULL
        (COND
         ((OR (SPADCALL |x| (QREFELT |$| 16)) (SPADCALL |y| (QREFELT |$| 16)))
          (QUOTE NIL))
         ((QUOTE T) (QUOTE T))))
       (GO G191)))
     (SEQ
      (LETT |z|
       (SPADCALL
```

```
        (SPADCALL
          (SPADCALL |x| (QREFELT |$| 18))
          (SPADCALL |y| (QREFELT |$| 18))
          |f|)
         |z|
         (QREFELT |$| 13))
       |LSAGG-;map;M3A;19|)
      (LETT |x| (SPADCALL |x| (QREFELT |$| 17)) |LSAGG-;map;M3A;19|)
      (EXIT (LETT |y| (SPADCALL |y| (QREFELT |$| 17)) |LSAGG-;map;M3A;19|)))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT (SPADCALL |z| (QREFELT |$| 47)))))))

(DEFUN |LSAGG-;reverse!;2A;20| (|x| |$|)
 (PROG (|z| |y|)
  (RETURN
   (SEQ
    (COND
     ((OR
       (SPADCALL |x| (QREFELT |$| 16))
       (SPADCALL
        (LETT |y| (SPADCALL |x| (QREFELT |$| 17)) |LSAGG-;reverse!;2A;20|)
        (QREFELT |$| 16)))
      |x|)
     ((QUOTE T)
      (SEQ
       (SPADCALL |x| (SPADCALL (QREFELT |$| 12)) (QREFELT |$| 25))
       (SEQ
        G190
        (COND
         ((NULL
           (COND
            ((SPADCALL |y| (QREFELT |$| 16)) (QUOTE NIL))
            ((QUOTE T) (QUOTE T))))
          (GO G191)))
        (SEQ
         (LETT |z| (SPADCALL |y| (QREFELT |$| 17)) |LSAGG-;reverse!;2A;20|)
         (SPADCALL |y| |x| (QREFELT |$| 25))
         (LETT |x| |y| |LSAGG-;reverse!;2A;20|)
         (EXIT (LETT |y| |z| |LSAGG-;reverse!;2A;20|)))
         NIL
         (GO G190)
         G191
         (EXIT NIL))
       (EXIT |x|)))))))))

(DEFUN |LSAGG-;copy;2A;21| (|x| |$|)
 (PROG (|k| |y|)
```

```
  (RETURN
   (SEQ
    (LETT |y| (SPADCALL (QREFELT |$| 12)) |LSAGG-;copy;2A;21|)
    (SEQ
     (LETT |k| 0 |LSAGG-;copy;2A;21|)
     G190
     (COND
      ((NULL
        (COND
         ((SPADCALL |x| (QREFELT |$| 16)) (QUOTE NIL))
         ((QUOTE T) (QUOTE T))))
       (GO G191)))
     (SEQ
      (COND
       ((EQL |k| 1000)
         (COND
           ((SPADCALL |x| (QREFELT |$| 56)) (EXIT (|error| "cyclic list"))))))
      (LETT |y|
       (SPADCALL (SPADCALL |x| (QREFELT |$| 18)) |y| (QREFELT |$| 13))
       |LSAGG-;copy;2A;21|)
      (EXIT (LETT |x| (SPADCALL |x| (QREFELT |$| 17)) |LSAGG-;copy;2A;21|)))
     (LETT |k| (QSADD1 |k|) |LSAGG-;copy;2A;21|)
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT (SPADCALL |y| (QREFELT |$| 47)))))))

(DEFUN |LSAGG-;copyInto!;2AIA;22| (|y| |x| |s| |$|)
 (PROG (|m| #1=#:G87636 |z|)
  (RETURN
   (SEQ
    (LETT |m| (SPADCALL |y| (QREFELT |$| 31)) |LSAGG-;copyInto!;2AIA;22|)
    (EXIT
     (COND
      ((|<| |s| |m|) (|error| "index out of range"))
      ((QUOTE T)
       (SEQ
        (LETT |z|
         (SPADCALL |y|
          (PROG1
           (LETT #1# (|-| |s| |m|) |LSAGG-;copyInto!;2AIA;22|)
           (|check-subtype| (|>=| #1# 0) (QUOTE (|NonNegativeInteger|)) #1#))
          (QREFELT |$| 32))
         |LSAGG-;copyInto!;2AIA;22|)
        (SEQ
         G190
         (COND
          ((NULL
            (COND
             ((OR
```

```
              (SPADCALL |z| (QREFELT |$| 16))
              (SPADCALL |x| (QREFELT |$| 16)))
             (QUOTE NIL))
            ((QUOTE T) (QUOTE T))))
           (GO G191)))
         (SEQ
          (SPADCALL |z| (SPADCALL |x| (QREFELT |$| 18)) (QREFELT |$| 58))
          (LETT |x| (SPADCALL |x| (QREFELT |$| 17)) |LSAGG-;copyInto!;2AIA;22|)
          (EXIT
           (LETT |z|
            (SPADCALL |z| (QREFELT |$| 17))
            |LSAGG-;copyInto!;2AIA;22|)))
         NIL
         (GO G190)
         G191
         (EXIT NIL))
        (EXIT |y|)))))))))

(DEFUN |LSAGG-;position;SA2I;23| (|w| |x| |s| |$|)
 (PROG (|m| #1=#:G87644 |k|)
  (RETURN
   (SEQ
    (LETT |m| (SPADCALL |x| (QREFELT |$| 31)) |LSAGG-;position;SA2I;23|)
    (EXIT
     (COND
      ((|<| |s| |m|) (|error| "index out of range"))
      ((QUOTE T)
       (SEQ
        (LETT |x|
         (SPADCALL |x|
          (PROG1
           (LETT #1# (|-| |s| |m|) |LSAGG-;position;SA2I;23|)
           (|check-subtype| (|>=| #1# 0) (QUOTE (|NonNegativeInteger|)) #1#))
          (QREFELT |$| 32))
         |LSAGG-;position;SA2I;23|)
        (SEQ
         (LETT |k| |s| |LSAGG-;position;SA2I;23|)
         G190
         (COND
          ((NULL
            (COND
             ((OR
               (SPADCALL |x| (QREFELT |$| 16))
               (SPADCALL |w| (SPADCALL |x| (QREFELT |$| 18)) (QREFELT |$| 51)))
              (QUOTE NIL))
             ((QUOTE T) (QUOTE T))))
           (GO G191)))
          (SEQ
           (EXIT
            (LETT |x|
```

```
            (SPADCALL |x| (QREFELT |$| 17)) |LSAGG-;position;SA2I;23|)))
          (LETT |k| (|+| |k| 1) |LSAGG-;position;SA2I;23|)
          (GO G190)
          G191
          (EXIT NIL))
         (EXIT
          (COND
           ((SPADCALL |x| (QREFELT |$| 16))
             (|-| (SPADCALL |x| (QREFELT |$| 31)) 1))
           ((QUOTE T) |k|))))))))))))

(DEFUN |LSAGG-;removeDuplicates!;2A;24| (|l| |$|)
 (PROG (|p|)
  (RETURN
   (SEQ
    (LETT |p| |l| |LSAGG-;removeDuplicates!;2A;24|)
    (SEQ
     G190
     (COND
      ((NULL
        (COND
         ((SPADCALL |p| (QREFELT |$| 16)) (QUOTE NIL))
         ((QUOTE T) (QUOTE T))))
       (GO G191)))
     (SEQ
      (EXIT
       (LETT |p|
        (SPADCALL |p|
         (SPADCALL
          (CONS
           (FUNCTION |LSAGG-;removeDuplicates!;2A;24!0|)
           (VECTOR |$| |p|))
          (SPADCALL |p| (QREFELT |$| 17))
          (QREFELT |$| 61))
         (QREFELT |$| 25))
        |LSAGG-;removeDuplicates!;2A;24|)))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT |l|)))))

(DEFUN |LSAGG-;removeDuplicates!;2A;24!0| (|#1| |$$|)
 (PROG (|$|)
  (LETT |$| (QREFELT |$$| 0) |LSAGG-;removeDuplicates!;2A;24|)
  (RETURN
   (PROGN
    (SPADCALL |#1|
     (SPADCALL (QREFELT |$$| 1) (QREFELT |$| 18))
     (QREFELT |$| 51))))))
```

```
(DEFUN |LSAGG-;<;2AB;25| (|x| |y| |$|)
 (PROG (#1=#:G87662)
  (RETURN
   (SEQ
    (EXIT
     (SEQ
      (SEQ
       G190
       (COND
        ((NULL
          (COND
           ((OR
              (SPADCALL |x| (QREFELT |$| 16))
              (SPADCALL |y| (QREFELT |$| 16)))
            (QUOTE NIL))
           ((QUOTE T) (QUOTE T))))
         (GO G191)))
       (SEQ
        (EXIT
         (COND
          ((NULL
            (SPADCALL
             (SPADCALL |x| (QREFELT |$| 18))
             (SPADCALL |y| (QREFELT |$| 18))
             (QREFELT |$| 51)))
           (PROGN
            (LETT #1#
              (SPADCALL
               (SPADCALL |x| (QREFELT |$| 18))
               (SPADCALL |y| (QREFELT |$| 18))
              (QREFELT |$| 63))
             |LSAGG-;<;2AB;25|)
            (GO #1#)))
          ((QUOTE T)
           (SEQ
            (LETT |x| (SPADCALL |x| (QREFELT |$| 17)) |LSAGG-;<;2AB;25|)
            (EXIT
             (LETT |y| (SPADCALL |y| (QREFELT |$| 17)) |LSAGG-;<;2AB;25|)))))))
       NIL
       (GO G190)
       G191
       (EXIT NIL))
      (EXIT
       (COND
        ((SPADCALL |x| (QREFELT |$| 16))
          (COND
           ((SPADCALL |y| (QREFELT |$| 16)) (QUOTE NIL))
           ((QUOTE T) (QUOTE T))))
        ((QUOTE T) (QUOTE NIL))))))
```

```
    #1#
    (EXIT #1#)))))

(DEFUN |ListAggregate&| (|#1| |#2|)
 (PROG (|DV$1| |DV$2| |dv$| |$| |pv$|)
  (RETURN
   (PROGN
    (LETT |DV$1| (|devaluate| |#1|) . #1=(|ListAggregate&|))
    (LETT |DV$2| (|devaluate| |#2|) . #1#)
    (LETT |dv$| (LIST (QUOTE |ListAggregate&|) |DV$1| |DV$2|) . #1#)
    (LETT |$| (make-array 66) . #1#)
    (QSETREFV |$| 0 |dv$|)
    (QSETREFV |$| 3 (LETT |pv$| (|buildPredVector| 0 0 NIL) . #1#))
    (|stuffDomainSlots| |$|)
    (QSETREFV |$| 6 |#1|)
    (QSETREFV |$| 7 |#2|)
    (COND
     ((|HasCategory| |#2| (QUOTE (|SetCategory|)))
       (QSETREFV |$| 52
         (CONS (|dispatchFunction| |LSAGG-;reduce;MA3S;17|) |$|))))
    (COND
     ((|HasCategory| |#2| (QUOTE (|SetCategory|)))
       (PROGN
        (QSETREFV |$| 60
         (CONS (|dispatchFunction| |LSAGG-;position;SA2I;23|) |$|))
        (QSETREFV |$| 62
         (CONS (|dispatchFunction| |LSAGG-;removeDuplicates!;2A;24|) |$|))))))
    (COND
     ((|HasCategory| |#2| (QUOTE (|OrderedSet|)))
       (QSETREFV |$| 64 (CONS (|dispatchFunction| |LSAGG-;<;2AB;25|) |$|))))
    |$|))))

(setf (get
 (QUOTE |ListAggregate&|)
 (QUOTE |infovec|))
 (LIST
  (QUOTE #(NIL NIL NIL NIL NIL NIL (|local| |#1|) (|local| |#2|)
   (|NonNegativeInteger|) (0 . |#|) (|Mapping| 15 7 7) |LSAGG-;sort!;M2A;1|
   (5 . |empty|) (9 . |concat|) |LSAGG-;list;SA;2| (|Boolean|)
   (15 . |empty?|) (20 . |rest|) (25 . |first|) (|Mapping| 7 7 7)
   (30 . |reduce|) |LSAGG-;reduce;MAS;3| (37 . |copy|) (42 . |merge!|)
   |LSAGG-;merge;M3A;4| (49 . |setrest!|) (|Mapping| 15 7)
   |LSAGG-;select!;M2A;5| (55 . |eq?|) |LSAGG-;merge!;M3A;6|
   (|Integer|) (61 . |minIndex|) (66 . |rest|) |LSAGG-;insert!;SAIA;7|
   (72 . |concat!|) |LSAGG-;insert!;2AIA;8| |LSAGG-;remove!;M2A;9|
   |LSAGG-;delete!;AIA;10| (|UniversalSegment| 30) (78 . |lo|)
   (83 . |hasHi|) (88 . |hi|) (93 . |maxIndex|) |LSAGG-;delete!;AUsA;11|
   (|Union| 7 (QUOTE "failed")) |LSAGG-;find;MAU;12|
   |LSAGG-;position;MAI;13| (98 . |reverse!|) (103 . |split!|)
   |LSAGG-;sorted?;MAB;15| |LSAGG-;reduce;MA2S;16| (109 . |=|)
```

```
  (115 . |reduce|) |LSAGG-;new;NniSA;18| |LSAGG-;map;M3A;19|
  |LSAGG-;reverse!;2A;20| (123 . |cyclic?|) |LSAGG-;copy;2A;21|
  (128 . |setfirst!|) |LSAGG-;copyInto!;2AIA;22| (134 . |position|)
  (141 . |remove!|) (147 . |removeDuplicates!|) (152 . |<|) (158 . |<|)
  (|Mapping| 7 7)))
 (QUOTE #(|sorted?| 164 |sort!| 170 |select!| 176 |reverse!| 182
   |removeDuplicates!| 187 |remove!| 192 |reduce| 198 |position| 219
   |new| 232 |merge!| 238 |merge| 245 |map| 252 |list| 259 |insert!|
   264 |find| 278 |delete!| 284 |copyInto!| 296 |copy| 303 |<| 308))
  (QUOTE NIL)
  (CONS
   (|makeByteWordVec2| 1 (QUOTE NIL))
   (CONS
    (QUOTE #())
    (CONS
     (QUOTE #())
     (|makeByteWordVec2| 64 (QUOTE (1 6 8 0 9 0 6 0 12 2 6 0 7 0 13 1 6
      15 0 16 1 6 0 0 17 1 6 7 0 18 3 6 7 19 0 7 20 1 6 0 0 22 3 6 0 10
      0 0 23 2 6 0 0 0 25 2 6 15 0 0 28 1 6 30 0 31 2 6 0 0 8 32 2 6 0 0
      0 34 1 38 30 0 39 1 38 15 0 40 1 38 30 0 41 1 6 30 0 42 1 6 0 0 47
      2 6 0 0 30 48 2 7 15 0 0 51 4 0 7 19 0 7 7 52 1 6 15 0 56 2 6 7 0
      7 58 3 0 30 7 0 30 60 2 6 0 26 0 61 1 0 0 0 62 2 7 15 0 0 63 2 0 15
      0 0 64 2 0 15 10 0 49 2 0 0 10 0 11 2 0 0 26 0 27 1 0 0 0 55 1 0 0
      0 62 2 0 0 26 0 36 3 0 7 19 0 7 50 4 0 7 19 0 7 7 52 2 0 7 19 0 21
      2 0 30 26 0 46 3 0 30 7 0 30 60 2 0 0 8 7 53 3 0 0 10 0 0 29 3 0 0
      10 0 0 24 3 0 0 19 0 0 54 1 0 0 7 14 3 0 0 7 0 30 33 3 0 0 0 0 30
      35 2 0 44 26 0 45 2 0 0 0 38 43 2 0 0 0 30 37 3 0 0 0 0 30 59 1 0
      0 0 57 2 0 15 0 0 64))))))
  (QUOTE |lookupComplete|)))
```

———-

21.37 MONOID.lsp BOOTSTRAP

MONOID depends on itself. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **MONOID** category which we can write into the **MID** directory. We compile the lisp code and copy the **MONOID.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— MONOID.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |Monoid;AL| (QUOTE NIL))
```

```
(DEFUN |Monoid| NIL
  (LET (#:G82434)
    (COND
      (|Monoid;AL|)
      (T (SETQ |Monoid;AL| (|Monoid;|))))))

(DEFUN |Monoid;| NIL
  (PROG (#1=#:G82432)
    (RETURN
      (PROG1
        (LETT #1#
          (|Join|
            (|SemiGroup|)
            (|mkCategory|
              (QUOTE |domain|)
              (QUOTE (
                ((|One| (|$|) |constant|) T)
                ((|sample| (|$|) |constant|) T)
                ((|one?| ((|Boolean|) |$|)) T)
                ((|**| (|$| |$| (|NonNegativeInteger|))) T)
                ((|^| (|$| |$| (|NonNegativeInteger|))) T)
                ((|recip| ((|Union| |$| "failed") |$|)) T)))
              NIL
              (QUOTE ((|NonNegativeInteger|) (|Boolean|)))
              NIL))
            |Monoid|)
         (SETELT #1# 0 (QUOTE (|Monoid|)))))))

(setf (get (QUOTE |Monoid|) (QUOTE NILADIC)) T)
```

———-

21.38 MONOID-.lsp BOOTSTRAP

MONOID- depends on **MONOID**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **MONOID-** category which we can write into the **MID** directory. We compile the lisp code and copy the **MONOID-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— MONOID-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |MONOID-;^;SNniS;1| (|x| |n| |$|)
```

```
  (SPADCALL |x| |n| (QREFELT |$| 8)))

(DEFUN |MONOID-;one?;SB;2| (|x| |$|)
  (SPADCALL |x| (|spadConstant| |$| 10) (QREFELT |$| 12)))

(DEFUN |MONOID-;sample;S;3| (|$|)
  (|spadConstant| |$| 10))

(DEFUN |MONOID-;recip;SU;4| (|x| |$|)
  (COND
    ((SPADCALL |x| (QREFELT |$| 15)) (CONS 0 |x|))
    ((QUOTE T) (CONS 1 "failed"))))

(DEFUN |MONOID-;**;SNniS;5| (|x| |n| |$|)
  (COND
    ((ZEROP |n|) (|spadConstant| |$| 10))
    ((QUOTE T) (SPADCALL |x| |n| (QREFELT |$| 20)))))

(DEFUN |Monoid&| (|#1|)
  (PROG (|DV$1| |dv$| |$| |pv$|)
    (RETURN
      (PROGN
        (LETT |DV$1| (|devaluate| |#1|) . #1=(|Monoid&|))
        (LETT |dv$| (LIST (QUOTE |Monoid&|) |DV$1|) . #1#)
        (LETT |$| (make-array 22) . #1#)
        (QSETREFV |$| 0 |dv$|)
        (QSETREFV |$| 3 (LETT |pv$| (|buildPredVector| 0 0 NIL) . #1#))
        (|stuffDomainSlots| |$|)
        (QSETREFV |$| 6 |#1|)
        |$|))))

(setf (get
  (QUOTE |Monoid&|)
  (QUOTE |infovec|))
  (LIST
    (QUOTE
      #(NIL NIL NIL NIL NIL NIL
        (|local| |#1|)
        (|NonNegativeInteger|)
        (0 . |**|)
        |MONOID-;^;SNniS;1|
        (6 . |One|)
        (|Boolean|)
        (10 . |=|)
        |MONOID-;one?;SB;2|
        |MONOID-;sample;S;3|
        (16 . |one?|)
        (|Union| |$| (QUOTE "failed"))
        |MONOID-;recip;SU;4|
        (|PositiveInteger|)
```

```
        (|RepeatedSquaring| 6)
        (21 . |expt|)
        |MONOID-;**;SNniS;5|))
    (QUOTE #(|sample| 27 |recip| 31 |one?| 36 |^| 41 |**| 47))
    (QUOTE NIL)
    (CONS
      (|makeByteWordVec2| 1 (QUOTE NIL))
      (CONS
        (QUOTE #())
        (CONS
          (QUOTE #())
          (|makeByteWordVec2| 21
            (QUOTE
              (2 6 0 0 7 8 0 6 0 10 2 6 11 0 0 12 1 6 11 0 15 2 19 6 6 18 20
              0 0 0 14 1 0 16 0 17 1 0 11 0 13 2 0 0 0 7 9 2 0 0 0 7 21))))))
  (QUOTE |lookupComplete|)))
```

———-

21.39 MTSCAT.lsp BOOTSTRAP

MTSCAT depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **MTSCAT** category which we can write into the **MID** directory. We compile the lisp code and copy the **MTSCAT.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— **MTSCAT.lsp BOOTSTRAP** —

```
(|/VERSIONCHECK| 2)

(SETQ |MultivariateTaylorSeriesCategory;CAT| (QUOTE NIL))

(SETQ |MultivariateTaylorSeriesCategory;AL| (QUOTE NIL))

(DEFUN |MultivariateTaylorSeriesCategory|
 (|&REST| #1=#:G83334 |&AUX| #2=#:G83332)
   (DSETQ #2# #1#)
   (LET (#3=#:G83333)
    (COND
     ((SETQ #3#
       (|assoc| (|devaluateList| #2#) |MultivariateTaylorSeriesCategory;AL|))
       (CDR #3#))
     (T
      (SETQ |MultivariateTaylorSeriesCategory;AL|
```

```
       (|cons5|
        (CONS
         (|devaluateList| #2#)
         (SETQ #3# (APPLY (FUNCTION |MultivariateTaylorSeriesCategory;|) #2#)))
        |MultivariateTaylorSeriesCategory;AL|))
       #3#))))

(DEFUN |MultivariateTaylorSeriesCategory;| (|t#1| |t#2|)
 (PROG (#1=#:G83331)
  (RETURN
   (PROG1
    (LETT #1#
     (|sublisV|
      (PAIR (QUOTE (|t#1| |t#2|)) (LIST (|devaluate| |t#1|) (|devaluate| |t#2|)))
      (|sublisV|
       (PAIR (QUOTE (#2=#:G83330)) (LIST (QUOTE (|IndexedExponents| |t#2|))))
       (COND
        (|MultivariateTaylorSeriesCategory;CAT|)
        ((QUOTE T)
         (LETT |MultivariateTaylorSeriesCategory;CAT|
          (|Join|
           (|PartialDifferentialRing| (QUOTE |t#2|))
           (|PowerSeriesCategory| (QUOTE |t#1|) (QUOTE #2#) (QUOTE |t#2|))
           (|InnerEvalable| (QUOTE |t#2|) (QUOTE |$|))
           (|Evalable| (QUOTE |$|))
           (|mkCategory|
            (QUOTE |domain|)
            (QUOTE (
             ((|coefficient| (|$| |$| |t#2| (|NonNegativeInteger|))) T)
             ((|coefficient|
                 (|$| |$| (|List| |t#2|) (|List| (|NonNegativeInteger|))))
                 T)
             ((|extend| (|$| |$| (|NonNegativeInteger|))) T)
             ((|monomial| (|$| |$| |t#2| (|NonNegativeInteger|))) T)
             ((|monomial|
                (|$| |$| (|List| |t#2|) (|List| (|NonNegativeInteger|))))
                 T)
             ((|order| ((|NonNegativeInteger|) |$| |t#2|)) T)
             ((|order|
                ((|NonNegativeInteger|) |$| |t#2| (|NonNegativeInteger|)))
                 T)
             ((|polynomial|
               ((|Polynomial| |t#1|) |$| (|NonNegativeInteger|)))
                T)
             ((|polynomial|
               ((|Polynomial| |t#1|)
                |$|
                (|NonNegativeInteger|)
                (|NonNegativeInteger|)))
                T)
```

```
            ((|integrate| (|$| |$| |t#2|))
               (|has| |t#1| (|Algebra| (|Fraction| (|Integer|)))))
            (QUOTE (
             ((|RadicalCategory|)
               (|has| |t#1| (|Algebra| (|Fraction| (|Integer|)))))
             ((|TranscendentalFunctionCategory|)
               (|has| |t#1| (|Algebra| (|Fraction| (|Integer|))))))
            (QUOTE
             ((|Polynomial| |t#1|)
              (|NonNegativeInteger|)
              (|List| |t#2|)
              (|List| (|NonNegativeInteger|))))
              NIL))
          . #3=(|MultivariateTaylorSeriesCategory|))))))
         . #3#)
    (SETELT #1# 0
     (LIST
      (QUOTE |MultivariateTaylorSeriesCategory|)
      (|devaluate| |t#1|)
      (|devaluate| |t#2|)))))))
```

———-

21.40 OINTDOM.lsp BOOTSTRAP

OINTDOM depends on itself. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **OINTDOM** category which we can write into the **MID** directory. We compile the lisp code and copy the **OINTDOM.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— OINTDOM.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |OrderedIntegralDomain;AL| (QUOTE NIL))

(DEFUN |OrderedIntegralDomain| NIL
  (LET (#:G84531)
    (COND
      (|OrderedIntegralDomain;AL|)
      (T (SETQ |OrderedIntegralDomain;AL| (|OrderedIntegralDomain;|))))))

(DEFUN |OrderedIntegralDomain;| NIL
  (PROG (#1=#:G84529)
    (RETURN
```

```
      (PROG1
        (LETT #1#
          (|Join| (|IntegralDomain|) (|OrderedRing|)) |OrderedIntegralDomain|)
        (SETELT #1# 0 (QUOTE (|OrderedIntegralDomain|)))))))

(setf (get (QUOTE |OrderedIntegralDomain|) (QUOTE NILADIC)) T)
```

———-

21.41 ORDRING.lsp BOOTSTRAP

ORDRING depends on **INT**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **ORDRING** category which we can write into the **MID** directory. We compile the lisp code and copy the **ORDRING.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Technically I can't justify this bootstrap stanza based on the lattice since **INT** is already bootstrapped. However using **INT** naked generates a "value stack overflow" error suggesting an infinite recursive loop. This code is here to experiment with breaking that loop.

Note that this code is not included in the generated catdef.spad file.

— ORDRING.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |OrderedRing;AL| (QUOTE NIL))

(DEFUN |OrderedRing| NIL
  (LET (#:G84457)
    (COND
      (|OrderedRing;AL|)
      (T (SETQ |OrderedRing;AL| (|OrderedRing;|))))))

(DEFUN |OrderedRing;| NIL
  (PROG (#1=#:G84455)
    (RETURN
      (PROG1
        (LETT #1#
          (|Join|
            (|OrderedAbelianGroup|)
            (|Ring|)
            (|Monoid|)
            (|mkCategory|
              (QUOTE |domain|)
              (QUOTE (
```

```
              ((|positive?| ((|Boolean|) |$|)) T)
              ((|negative?| ((|Boolean|) |$|)) T)
              ((|sign| ((|Integer|) |$|)) T)
              ((|abs| (|$| |$|)) T)))
            NIL
            (QUOTE ((|Integer|) (|Boolean|)))
            NIL))
          |OrderedRing|)
       (SETELT #1# 0 (QUOTE (|OrderedRing|)))))))

(setf (get (QUOTE |OrderedRing|) (QUOTE NILADIC)) T)
```

———-

21.42 ORDRING-.lsp BOOTSTRAP

ORDRING- depends on **ORDRING**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **ORDRING-** category which we can write into the **MID** directory. We compile the lisp code and copy the **ORDRING-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— ORDRING-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |ORDRING-;positive?;SB;1| (|x| |$|)
  (SPADCALL (|spadConstant| |$| 7) |x| (QREFELT |$| 9)))

(DEFUN |ORDRING-;negative?;SB;2| (|x| |$|)
  (SPADCALL |x| (|spadConstant| |$| 7) (QREFELT |$| 9)))

(DEFUN |ORDRING-;sign;SI;3| (|x| |$|)
  (COND
    ((SPADCALL |x| (QREFELT |$| 12)) 1)
    ((SPADCALL |x| (QREFELT |$| 13)) -1)
    ((SPADCALL |x| (QREFELT |$| 15)) 0)
    ((QUOTE T)
      (|error| "x satisfies neither positive?, negative? or zero?"))))

(DEFUN |ORDRING-;abs;2S;4| (|x| |$|)
  (COND
    ((SPADCALL |x| (QREFELT |$| 12)) |x|)
    ((SPADCALL |x| (QREFELT |$| 13)) (SPADCALL |x| (QREFELT |$| 18)))
    ((SPADCALL |x| (QREFELT |$| 15)) (|spadConstant| |$| 7))
```

```
    ((QUOTE T)
      (|error| "x satisfies neither positive?, negative? or zero?"))))

(DEFUN |OrderedRing&| (|#1|)
  (PROG (|DV$1| |dv$| |$| |pv$|)
    (RETURN
      (PROGN
        (LETT |DV$1| (|devaluate| |#1|) . #1=(|OrderedRing&|))
        (LETT |dv$| (LIST (QUOTE |OrderedRing&|) |DV$1|) . #1#)
        (LETT |$| (make-array 20) . #1#)
        (QSETREFV |$| 0 |dv$|)
        (QSETREFV |$| 3 (LETT |pv$| (|buildPredVector| 0 0 NIL) . #1#))
        (|stuffDomainSlots| |$|)
        (QSETREFV |$| 6 |#1|)
        |$|))))

(setf (get
  (QUOTE |OrderedRing&|)
  (QUOTE |infovec|))
  (LIST
    (QUOTE
      #(NIL NIL NIL NIL NIL NIL
        (|local| |#1|)
        (0 . |Zero|)
        (|Boolean|)
        (4 . |<|)
        |ORDRING-;positive?;SB;1|
        |ORDRING-;negative?;SB;2|
        (10 . |positive?|)
        (15 . |negative?|)
        (20 . |One|)
        (24 . |zero?|)
        (|Integer|)
        |ORDRING-;sign;SI;3|
        (29 . |-|)
        |ORDRING-;abs;2S;4|))
    (QUOTE #(|sign| 34 |positive?| 39 |negative?| 44 |abs| 49))
    (QUOTE NIL)
    (CONS
      (|makeByteWordVec2| 1 (QUOTE NIL))
      (CONS
        (QUOTE #())
        (CONS
          (QUOTE #())
          (|makeByteWordVec2| 19
            (QUOTE
              (0 6 0 7 2 6 8 0 0 9 1 6 8 0 12 1 6 8 0 13 0 6 0 14 1 6 8 0 15
               1 6 0 0 18 1 0 16 0 17 1 0 8 0 10 1 0 8 0 11 1 0 0 0 19))))))
   (QUOTE |lookupComplete|)))
```

———-

21.43 POLYCAT.lsp BOOTSTRAP

POLYCAT depends on itself. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **POLYCAT** category which we can write into the **MID** directory. We compile the lisp code and copy the **POLYCAT.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— POLYCAT.lsp BOOTSTRAP —

```
(/VERSIONCHECK 2)

(SETQ |PolynomialCategory;CAT| (QUOTE NIL))

(SETQ |PolynomialCategory;AL| (QUOTE NIL))

(DEFUN |PolynomialCategory| (&REST #0=#:G1430 &AUX #1=#:G1428)
 (DSETQ #1# #0#)
 (LET (#2=#:G1429)
  (COND
   ((SETQ #2# (|assoc| (|devaluateList| #1#) |PolynomialCategory;AL|))
     (CDR #2#))
   (T
     (SETQ |PolynomialCategory;AL|
      (|cons5|
       (CONS (|devaluateList| #1#)
        (SETQ #2# (APPLY (FUNCTION |PolynomialCategory;|) #1#)))
       |PolynomialCategory;AL|))
     #2#))))

(DEFUN |PolynomialCategory;| (|t#1| |t#2| |t#3|)
 (PROG (#0=#:G1427)
  (RETURN
   (PROG1
    (LETT #0#
     (|sublisV|
      (PAIR (QUOTE (|t#1| |t#2| |t#3|)) (LIST (|devaluate| |t#1|) (|devaluate| |t#2|) (|devaluate| |t#3|)))
      (COND
       (|PolynomialCategory;CAT|)
       ((QUOTE T)
        (LETT |PolynomialCategory;CAT|
         (|Join|
          (|PartialDifferentialRing| (QUOTE |t#3|))
          (|FiniteAbelianMonoidRing| (QUOTE |t#1|) (QUOTE |t#2|))
```

```
(|Evalable| (QUOTE $))
(|InnerEvalable| (QUOTE |t#3|) (QUOTE |t#1|))
(|InnerEvalable| (QUOTE |t#3|) (QUOTE $))
(|RetractableTo| (QUOTE |t#3|))
(|FullyLinearlyExplicitRingOver| (QUOTE |t#1|))
(|mkCategory| (QUOTE |domain|)
 (QUOTE
  ((((|degree| ((|NonNegativeInteger|) $ |t#3|)) T)
   ((|degree| ((|List| (|NonNegativeInteger|)) $ (|List| |t#3|))) T)
   ((|coefficient| ($ $ |t#3| (|NonNegativeInteger|))) T)
   ((|coefficient| ($ $ (|List| |t#3|)
     (|List| (|NonNegativeInteger|)))) T)
   ((|monomials| ((|List| $) $)) T)
   ((|univariate| ((|SparseUnivariatePolynomial| $) $ |t#3|)) T)
   ((|univariate| ((|SparseUnivariatePolynomial| |t#1|) $)) T)
   ((|mainVariable| ((|Union| |t#3| "failed") $)) T)
   ((|minimumDegree| ((|NonNegativeInteger|) $ |t#3|)) T)
   ((|minimumDegree| ((|List| (|NonNegativeInteger|)) $
     (|List| |t#3|))) T)
   ((|monicDivide|
     ((|Record| (|:| |quotient| $) (|:| |remainder| $)) $ $ |t#3|))
     T)
   ((|monomial| ($ $ |t#3| (|NonNegativeInteger|))) T)
   ((|monomial| ($ $ (|List| |t#3|) (|List| (|NonNegativeInteger|))))
     T)
   ((|multivariate| ($ (|SparseUnivariatePolynomial| |t#1|) |t#3|))
     T)
   ((|multivariate| ($ (|SparseUnivariatePolynomial| $) |t#3|)) T)
   ((|isPlus| ((|Union| (|List| $) "failed") $)) T)
   ((|isTimes| ((|Union| (|List| $) "failed") $)) T)
   ((|isExpt|
     ((|Union|
      (|Record| (|:| |var| |t#3|)
                (|:| |exponent| (|NonNegativeInteger|)))
      "failed") $))
     T)
   ((|totalDegree| ((|NonNegativeInteger|) $)) T)
   ((|totalDegree| ((|NonNegativeInteger|) $ (|List| |t#3|))) T)
   ((|variables| ((|List| |t#3|) $)) T)
   ((|primitiveMonomials| ((|List| $) $)) T)
   ((|resultant| ($ $ $ |t#3|)) (|has| |t#1| (|CommutativeRing|)))
   ((|discriminant| ($ $ |t#3|)) (|has| |t#1| (|CommutativeRing|)))
   ((|content| ($ $ |t#3|)) (|has| |t#1| (|GcdDomain|)))
   ((|primitivePart| ($ $)) (|has| |t#1| (|GcdDomain|)))
   ((|primitivePart| ($ $ |t#3|)) (|has| |t#1| (|GcdDomain|)))
   ((|squareFree| ((|Factored| $) $)) (|has| |t#1| (|GcdDomain|)))
   ((|squareFreePart| ($ $)) (|has| |t#1| (|GcdDomain|)))))
 (QUOTE
  (((|OrderedSet|) (|has| |t#1| (|OrderedSet|)))
   ((|ConvertibleTo| (|InputForm|))
```

```
            (AND (|has| |t#3| (|ConvertibleTo| (|InputForm|)))
                 (|has| |t#1| (|ConvertibleTo| (|InputForm|)))))
           ((|ConvertibleTo| (|Pattern| (|Integer|)))
            (AND (|has| |t#3| (|ConvertibleTo| (|Pattern| (|Integer|))))
                 (|has| |t#1| (|ConvertibleTo| (|Pattern| (|Integer|))))))
           ((|ConvertibleTo| (|Pattern| (|Float|)))
            (AND (|has| |t#3| (|ConvertibleTo| (|Pattern| (|Float|))))
                 (|has| |t#1| (|ConvertibleTo| (|Pattern| (|Float|))))))
           ((|PatternMatchable| (|Integer|))
            (AND
             (|has| |t#3| (|PatternMatchable| (|Integer|)))
             (|has| |t#1| (|PatternMatchable| (|Integer|)))))
           ((|PatternMatchable| (|Float|))
            (AND
             (|has| |t#3| (|PatternMatchable| (|Float|)))
             (|has| |t#1| (|PatternMatchable| (|Float|)))))
           ((|GcdDomain|) (|has| |t#1| (|GcdDomain|)))
           (|canonicalUnitNormal|
             (|has| |t#1| (ATTRIBUTE |canonicalUnitNormal|)))
           ((|PolynomialFactorizationExplicit|)
             (|has| |t#1| (|PolynomialFactorizationExplicit|)))))
         (QUOTE
          ((|Factored| $)
           (|List| $)
           (|List| |t#3|)
           (|NonNegativeInteger|)
           (|SparseUnivariatePolynomial| $)
           (|SparseUnivariatePolynomial| |t#1|)
           (|List| (|NonNegativeInteger|))))
          NIL))
        . #1=(|PolynomialCategory|)))))
   . #1#)
 (SETELT #0# 0
  (LIST (QUOTE |PolynomialCategory|)
   (|devaluate| |t#1|) (|devaluate| |t#2|) (|devaluate| |t#3|)))))))
```

———-

21.44 POLYCAT-.lsp BOOTSTRAP

POLYCAT- depends on **POLYCAT**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **POLYCAT-** category which we can write into the **MID** directory. We compile the lisp code and copy the **POLYCAT-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— POLYCAT-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)


(/VERSIONCHECK 2)

(DEFUN |POLYCAT-;eval;SLS;1| (|p| |l| $)
 (PROG (#0=#:G1444 #1=#:G1438 #2=#:G1445 #3=#:G1446 |lvar| #4=#:G1447
        |e| #5=#:G1448)
  (RETURN
   (SEQ
    (COND
     ((NULL |l|) |p|)
     ((QUOTE T)
      (SEQ
       (SEQ
        (EXIT
         (SEQ
          (LETT |e| NIL |POLYCAT-;eval;SLS;1|)
          (LETT #0# |l| |POLYCAT-;eval;SLS;1|)
          G190
          (COND
           ((OR (ATOM #0#)
                (PROGN (LETT |e| (CAR #0#) |POLYCAT-;eval;SLS;1|) NIL))
            (GO G191)))
          (SEQ
           (EXIT
            (COND
             ((QEQCAR
               (SPADCALL (SPADCALL |e| (QREFELT $ 11)) (QREFELT $ 13)) 1)
              (PROGN
               (LETT #1#
                (|error| "cannot find a variable to evaluate")
                |POLYCAT-;eval;SLS;1|)
               (GO #1#))))))
          (LETT #0# (CDR #0#) |POLYCAT-;eval;SLS;1|)
          (GO G190)
          G191
          (EXIT NIL)))
        #1# (EXIT #1#))
       (LETT |lvar|
        (PROGN
         (LETT #2# NIL |POLYCAT-;eval;SLS;1|)
         (SEQ
          (LETT |e| NIL |POLYCAT-;eval;SLS;1|)
          (LETT #3# |l| |POLYCAT-;eval;SLS;1|)
          G190
          (COND
```

```
           ((OR (ATOM #3#)
                (PROGN (LETT |e| (CAR #3#) |POLYCAT-;eval;SLS;1|) NIL))
            (GO G191)))
          (SEQ
           (EXIT
            (LETT #2#
             (CONS (SPADCALL (SPADCALL |e| (QREFELT $ 11)) (QREFELT $ 14))
                   #2#)
             |POLYCAT-;eval;SLS;1|)))
          (LETT #3# (CDR #3#) |POLYCAT-;eval;SLS;1|)
          (GO G190)
          G191
          (EXIT (NREVERSE0 #2#))))
         |POLYCAT-;eval;SLS;1|)
        (EXIT
         (SPADCALL |p| |lvar|
          (PROGN
           (LETT #4# NIL |POLYCAT-;eval;SLS;1|)
           (SEQ
            (LETT |e| NIL |POLYCAT-;eval;SLS;1|)
            (LETT #5# |l| |POLYCAT-;eval;SLS;1|)
            G190
            (COND
             ((OR (ATOM #5#)
                  (PROGN (LETT |e| (CAR #5#) |POLYCAT-;eval;SLS;1|) NIL))
              (GO G191)))
            (SEQ
             (EXIT
              (LETT #4# (CONS (SPADCALL |e| (QREFELT $ 15)) #4#)
               |POLYCAT-;eval;SLS;1|)))
            (LETT #5# (CDR #5#) |POLYCAT-;eval;SLS;1|)
            (GO G190)
            G191
            (EXIT (NREVERSE0 #4#))))
          (QREFELT $ 18))))))))))

(DEFUN |POLYCAT-;monomials;SL;2| (|p| $)
 (PROG (|ml|)
  (RETURN
   (SEQ
    (LETT |ml| NIL |POLYCAT-;monomials;SL;2|)
    (SEQ G190
     (COND
      ((NULL
        (COND
         ((SPADCALL |p| (|spadConstant| $ 21) (QREFELT $ 24)) (QUOTE NIL))
         ((QUOTE T) (QUOTE T))))
       (GO G191)))
     (SEQ
      (LETT |ml|
```

```
       (CONS (SPADCALL |p| (QREFELT $ 25)) |ml|)
       |POLYCAT-;monomials;SL;2|)
      (EXIT
       (LETT |p| (SPADCALL |p| (QREFELT $ 26)) |POLYCAT-;monomials;SL;2|)))
      NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT (REVERSE |ml|))))))

(DEFUN |POLYCAT-;isPlus;SU;3| (|p| $)
 (PROG (|l|)
  (RETURN
   (COND
    ((NULL
       (CDR (LETT |l| (SPADCALL |p| (QREFELT $ 28)) |POLYCAT-;isPlus;SU;3|)))
      (CONS 1 "failed"))
    ((QUOTE T) (CONS 0 |l|))))))

(DEFUN |POLYCAT-;isTimes;SU;4| (|p| $)
 (PROG (|lv| #0=#:G1470 |v| #1=#:G1471 |l| |r|)
  (RETURN
   (SEQ
    (COND
     ((OR (NULL
           (LETT |lv| (SPADCALL |p| (QREFELT $ 31)) |POLYCAT-;isTimes;SU;4|))
          (NULL (SPADCALL |p| (QREFELT $ 32))))
      (CONS 1 "failed"))
     ((QUOTE T)
      (SEQ
       (LETT |l|
        (PROGN
         (LETT #0# NIL |POLYCAT-;isTimes;SU;4|)
         (SEQ
          (LETT |v| NIL |POLYCAT-;isTimes;SU;4|)
          (LETT #1# |lv| |POLYCAT-;isTimes;SU;4|)
          G190
          (COND
           ((OR (ATOM #1#)
                (PROGN (LETT |v| (CAR #1#) |POLYCAT-;isTimes;SU;4|) NIL))
            (GO G191)))
          (SEQ
           (EXIT
            (LETT #0#
             (CONS
              (SPADCALL (|spadConstant| $ 33) |v|
               (SPADCALL |p| |v| (QREFELT $ 36)) (QREFELT $ 37))
              #0#)
              |POLYCAT-;isTimes;SU;4|)))
          (LETT #1# (CDR #1#) |POLYCAT-;isTimes;SU;4|)
```

```
          (GO G190)
          G191
         (EXIT (NREVERSE0 #0#))))
        |POLYCAT-;isTimes;SU;4|)
       (LETT |r| (SPADCALL |p| (QREFELT $ 38)) |POLYCAT-;isTimes;SU;4|)
       (EXIT
        (COND
         ((SPADCALL |r| (|spadConstant| $ 34) (QREFELT $ 39))
           (COND
            ((NULL (CDR |lv|)) (CONS 1 "failed"))
            ((QUOTE T) (CONS 0 |l|))))
         ((QUOTE T)
           (CONS 0 (CONS (SPADCALL |r| (QREFELT $ 40)) |l|)))))))))))))

(DEFUN |POLYCAT-;isExpt;SU;5| (|p| $)
 (PROG (|u| |d|)
  (RETURN
   (SEQ
    (LETT |u| (SPADCALL |p| (QREFELT $ 42)) |POLYCAT-;isExpt;SU;5|)
    (EXIT
     (COND
      ((OR (QEQCAR |u| 1)
           (NULL
            (SPADCALL |p|
             (SPADCALL (|spadConstant| $ 33)
              (QCDR |u|)
              (LETT |d| (SPADCALL |p| (QCDR |u|) (QREFELT $ 36))
               |POLYCAT-;isExpt;SU;5|)
              (QREFELT $ 37))
             (QREFELT $ 24))))
       (CONS 1 "failed"))
      ((QUOTE T) (CONS 0 (CONS (QCDR |u|) |d|)))))))))

(DEFUN |POLYCAT-;coefficient;SVarSetNniS;6| (|p| |v| |n| $)
 (SPADCALL (SPADCALL |p| |v| (QREFELT $ 47)) |n| (QREFELT $ 49)))

(DEFUN |POLYCAT-;coefficient;SLLS;7| (|p| |lv| |ln| $)
 (COND
  ((NULL |lv|)
   (COND
    ((NULL |ln|) |p|)
    ((QUOTE T) (|error| "mismatched lists in coefficient"))))
  ((NULL |ln|) (|error| "mismatched lists in coefficient"))
  ((QUOTE T)
   (SPADCALL
    (SPADCALL
     (SPADCALL |p| (|SPADfirst| |lv|) (QREFELT $ 47))
     (|SPADfirst| |ln|)
     (QREFELT $ 49))
    (CDR |lv|)
```

```
    (CDR |ln|)
    (QREFELT $ 52)))))

(DEFUN |POLYCAT-;monomial;SLLS;8| (|p| |lv| |ln| $)
 (COND
  ((NULL |lv|)
   (COND
    ((NULL |ln|) |p|)
    ((QUOTE T) (|error| "mismatched lists in monomial"))))
  ((NULL |ln|) (|error| "mismatched lists in monomial"))
  ((QUOTE T)
   (SPADCALL
    (SPADCALL |p| (|SPADfirst| |lv|) (|SPADfirst| |ln|) (QREFELT $ 37))
    (CDR |lv|)
    (CDR |ln|)
    (QREFELT $ 54)))))

(DEFUN |POLYCAT-;retract;SVarSet;9| (|p| $)
 (PROG (#0=#:G1496 |q|)
  (RETURN
   (SEQ
    (LETT |q|
     (PROG2
      (LETT #0# (SPADCALL |p| (QREFELT $ 42)) |POLYCAT-;retract;SVarSet;9|)
      (QCDR #0#)
      (|check-union| (QEQCAR #0# 0) (QREFELT $ 9) #0#))
     |POLYCAT-;retract;SVarSet;9|)
    (EXIT
     (COND
      ((SPADCALL (SPADCALL |q| (QREFELT $ 56)) |p| (QREFELT $ 24)) |q|)
      ((QUOTE T) (|error| "Polynomial is not a single variable")))))))))

(DEFUN |POLYCAT-;retractIfCan;SU;10| (|p| $)
 (PROG (|q| #0=#:G1504)
  (RETURN
   (SEQ
    (EXIT
     (SEQ
      (SEQ
       (LETT |q| (SPADCALL |p| (QREFELT $ 42)) |POLYCAT-;retractIfCan;SU;10|)
       (EXIT
        (COND
         ((QEQCAR |q| 0)
          (COND
           ((SPADCALL (SPADCALL (QCDR |q|) (QREFELT $ 56)) |p| (QREFELT $ 24))
             (PROGN
              (LETT #0# |q| |POLYCAT-;retractIfCan;SU;10|)
              (GO #0#)))))))))
      (EXIT (CONS 1 "failed"))))
     #0#
```

```
     (EXIT #0#)))))

(DEFUN |POLYCAT-;mkPrim| (|p| $)
 (SPADCALL
  (|spadConstant| $ 34)
  (SPADCALL |p| (QREFELT $ 59))
  (QREFELT $ 60)))

(DEFUN |POLYCAT-;primitiveMonomials;SL;12| (|p| $)
 (PROG (#0=#:G1509 |q| #1=#:G1510)
  (RETURN
   (SEQ
    (PROGN
     (LETT #0# NIL |POLYCAT-;primitiveMonomials;SL;12|)
     (SEQ
      (LETT |q| NIL |POLYCAT-;primitiveMonomials;SL;12|)
      (LETT #1# (SPADCALL |p| (QREFELT $ 28)) |POLYCAT-;primitiveMonomials;SL;12|)
      G190
      (COND
       ((OR (ATOM #1#)
            (PROGN
             (LETT |q| (CAR #1#) |POLYCAT-;primitiveMonomials;SL;12|)
             NIL))
        (GO G191)))
      (SEQ
       (EXIT
        (LETT #0# (CONS (|POLYCAT-;mkPrim| |q| $) #0#)
         |POLYCAT-;primitiveMonomials;SL;12|)))
      (LETT #1# (CDR #1#) |POLYCAT-;primitiveMonomials;SL;12|)
      (GO G190)
      G191
      (EXIT (NREVERSE0 #0#)))))))))

(DEFUN |POLYCAT-;totalDegree;SNni;13| (|p| $)
 (PROG (#0=#:G1512 |d| |u|)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |p| (QREFELT $ 62)) 0)
     ((QUOTE T)
      (SEQ
       (LETT |u|
        (SPADCALL |p|
         (PROG2
          (LETT #0#
           (SPADCALL |p| (QREFELT $ 42))
           |POLYCAT-;totalDegree;SNni;13|)
          (QCDR #0#)
          (|check-union| (QEQCAR #0# 0) (QREFELT $ 9) #0#))
         (QREFELT $ 47))
```

```
        |POLYCAT-;totalDegree;SNni;13|)
       (LETT |d| 0 |POLYCAT-;totalDegree;SNni;13|)
       (SEQ G190
        (COND
         ((NULL
          (COND
           ((SPADCALL |u| (|spadConstant| $ 63) (QREFELT $ 64)) (QUOTE NIL))
           ((QUOTE T) (QUOTE T)))) (GO G191)))
        (SEQ
         (LETT |d|
          (MAX |d|
           (+
            (SPADCALL |u| (QREFELT $ 65))
            (SPADCALL (SPADCALL |u| (QREFELT $ 66)) (QREFELT $ 67))))
          |POLYCAT-;totalDegree;SNni;13|)
         (EXIT
          (LETT |u|
           (SPADCALL |u| (QREFELT $ 68)) |POLYCAT-;totalDegree;SNni;13|)))
        NIL
        (GO G190)
        G191
        (EXIT NIL))
       (EXIT |d|)))))))

(DEFUN |POLYCAT-;totalDegree;SLNni;14| (|p| |lv| $)
 (PROG (#0=#:G1520 |v| |w| |d| |u|)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |p| (QREFELT $ 62)) 0)
     ((QUOTE T)
      (SEQ
       (LETT |u|
        (SPADCALL |p|
         (LETT |v|
          (PROG2
           (LETT #0#
            (SPADCALL |p| (QREFELT $ 42))
            |POLYCAT-;totalDegree;SLNni;14|)
           (QCDR #0#)
           (|check-union| (QEQCAR #0# 0) (QREFELT $ 9) #0#))
          |POLYCAT-;totalDegree;SLNni;14|)
         (QREFELT $ 47))
        |POLYCAT-;totalDegree;SLNni;14|)
       (LETT |d| 0 |POLYCAT-;totalDegree;SLNni;14|)
       (LETT |w| 0 |POLYCAT-;totalDegree;SLNni;14|)
       (COND
        ((SPADCALL |v| |lv| (QREFELT $ 70))
          (LETT |w| 1 |POLYCAT-;totalDegree;SLNni;14|)))
       (SEQ G190
```

```
        (COND
         ((NULL
          (COND
           ((SPADCALL |u| (|spadConstant| $ 63) (QREFELT $ 64)) (QUOTE NIL))
           ((QUOTE T) (QUOTE T)))) (GO G191)))
        (SEQ
         (LETT |d|
          (MAX |d|
           (+
            (* |w| (SPADCALL |u| (QREFELT $ 65)))
            (SPADCALL (SPADCALL |u| (QREFELT $ 66)) |lv| (QREFELT $ 71))))
          |POLYCAT-;totalDegree;SLNni;14|)
         (EXIT
          (LETT |u|
           (SPADCALL |u| (QREFELT $ 68))
           |POLYCAT-;totalDegree;SLNni;14|)))
         NIL
         (GO G190)
         G191
         (EXIT NIL))
       (EXIT |d|))))))))

(DEFUN |POLYCAT-;resultant;2SVarSetS;15| (|p1| |p2| |mvar| $)
 (SPADCALL
  (SPADCALL |p1| |mvar| (QREFELT $ 47))
  (SPADCALL |p2| |mvar| (QREFELT $ 47))
  (QREFELT $ 73)))

(DEFUN |POLYCAT-;discriminant;SVarSetS;16| (|p| |var| $)
 (SPADCALL (SPADCALL |p| |var| (QREFELT $ 47)) (QREFELT $ 75)))

(DEFUN |POLYCAT-;allMonoms| (|l| $)
 (PROG (#0=#:G1532 |p| #1=#:G1533)
  (RETURN
   (SEQ
    (SPADCALL
     (SPADCALL
      (PROGN
       (LETT #0# NIL |POLYCAT-;allMonoms|)
       (SEQ
        (LETT |p| NIL |POLYCAT-;allMonoms|)
        (LETT #1# |l| |POLYCAT-;allMonoms|)
        G190
        (COND
         ((OR (ATOM #1#) (PROGN (LETT |p| (CAR #1#) |POLYCAT-;allMonoms|) NIL))
          (GO G191)))
        (SEQ
         (EXIT
          (LETT #0#
           (CONS (SPADCALL |p| (QREFELT $ 77)) #0#)
```

```
          |POLYCAT-;allMonoms|)))
        (LETT #1# (CDR #1#) |POLYCAT-;allMonoms|)
        (GO G190)
        G191
        (EXIT (NREVERSE0 #0#))))
      (QREFELT $ 79))
     (QREFELT $ 80))))))

(DEFUN |POLYCAT-;P2R| (|p| |b| |n| $)
 (PROG (|w| |bj| #0=#:G1538 |i| #1=#:G1537)
  (RETURN
   (SEQ
    (LETT |w|
     (SPADCALL |n| (|spadConstant| $ 22) (QREFELT $ 82))
     |POLYCAT-;P2R|)
    (SEQ
     (LETT |bj| NIL |POLYCAT-;P2R|)
     (LETT #0# |b| |POLYCAT-;P2R|)
     (LETT |i| (SPADCALL |w| (QREFELT $ 84)) |POLYCAT-;P2R|)
     (LETT #1# (QVSIZE |w|) |POLYCAT-;P2R|)
     G190
     (COND
      ((OR (> |i| #1#)
           (ATOM #0#)
           (PROGN (LETT |bj| (CAR #0#) |POLYCAT-;P2R|) NIL))
       (GO G191)))
     (SEQ
      (EXIT
       (SPADCALL |w| |i| (SPADCALL |p| |bj| (QREFELT $ 85)) (QREFELT $ 86))))
     (LETT |i|
      (PROG1 (+ |i| 1) (LETT #0# (CDR #0#) |POLYCAT-;P2R|)) |POLYCAT-;P2R|)
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT |w|)))))

(DEFUN |POLYCAT-;eq2R| (|l| |b| $)
 (PROG (#0=#:G1542 |bj| #1=#:G1543 #2=#:G1544 |p| #3=#:G1545)
  (RETURN
   (SEQ
    (SPADCALL
     (PROGN
      (LETT #0# NIL |POLYCAT-;eq2R|)
      (SEQ
       (LETT |bj| NIL |POLYCAT-;eq2R|)
       (LETT #1# |b| |POLYCAT-;eq2R|)
       G190
       (COND
        ((OR (ATOM #1#)
             (PROGN (LETT |bj| (CAR #1#) |POLYCAT-;eq2R|) NIL)) (GO G191)))
```

```
       (SEQ
        (EXIT
         (LETT #0#
          (CONS
           (PROGN
            (LETT #2# NIL |POLYCAT-;eq2R|)
            (SEQ
             (LETT |p| NIL |POLYCAT-;eq2R|)
             (LETT #3# |l| |POLYCAT-;eq2R|)
             G190
             (COND
              ((OR (ATOM #3#) (PROGN (LETT |p| (CAR #3#) |POLYCAT-;eq2R|) NIL))
               (GO G191)))
             (SEQ
              (EXIT
               (LETT #2#
                (CONS (SPADCALL |p| |bj| (QREFELT $ 85)) #2#)
                |POLYCAT-;eq2R|)))
             (LETT #3# (CDR #3#) |POLYCAT-;eq2R|)
             (GO G190)
             G191
             (EXIT (NREVERSE0 #2#))))
            #0#)
           |POLYCAT-;eq2R|)))
        (LETT #1# (CDR #1#) |POLYCAT-;eq2R|)
        (GO G190)
        G191
        (EXIT (NREVERSE0 #0#))))
     (QREFELT $ 89))))))

(DEFUN |POLYCAT-;reducedSystem;MM;20| (|m| $)
 (PROG (#0=#:G1555 |r| #1=#:G1556 |b| #2=#:G1557 |bj| #3=#:G1558 |d| |mm| |l|)
  (RETURN
   (SEQ
    (LETT |l| (SPADCALL |m| (QREFELT $ 92)) |POLYCAT-;reducedSystem;MM;20|)
    (LETT |b|
     (SPADCALL
      (SPADCALL
       (PROGN
        (LETT #0# NIL |POLYCAT-;reducedSystem;MM;20|)
        (SEQ
         (LETT |r| NIL |POLYCAT-;reducedSystem;MM;20|)
         (LETT #1# |l| |POLYCAT-;reducedSystem;MM;20|)
         G190
         (COND
          ((OR (ATOM #1#)
               (PROGN (LETT |r| (CAR #1#) |POLYCAT-;reducedSystem;MM;20|) NIL))
           (GO G191)))
         (SEQ
          (EXIT
```

```
          (LETT #0#
           (CONS (|POLYCAT-;allMonoms| |r| $) #0#)
           |POLYCAT-;reducedSystem;MM;20|)))
         (LETT #1# (CDR #1#) |POLYCAT-;reducedSystem;MM;20|)
         (GO G190)
         G191
         (EXIT (NREVERSE0 #0#))))
        (QREFELT $ 79))
       (QREFELT $ 80))
      |POLYCAT-;reducedSystem;MM;20|)
     (LETT |d|
      (PROGN
       (LETT #2# NIL |POLYCAT-;reducedSystem;MM;20|)
       (SEQ
        (LETT |bj| NIL |POLYCAT-;reducedSystem;MM;20|)
        (LETT #3# |b| |POLYCAT-;reducedSystem;MM;20|)
        G190
        (COND
         ((OR (ATOM #3#)
              (PROGN (LETT |bj| (CAR #3#) |POLYCAT-;reducedSystem;MM;20|) NIL))
          (GO G191)))
        (SEQ
         (EXIT
          (LETT #2#
           (CONS (SPADCALL |bj| (QREFELT $ 59)) #2#)
           |POLYCAT-;reducedSystem;MM;20|)))
        (LETT #3# (CDR #3#) |POLYCAT-;reducedSystem;MM;20|)
        (GO G190)
        G191
        (EXIT (NREVERSE0 #2#))))
       |POLYCAT-;reducedSystem;MM;20|)
     (LETT |mm|
      (|POLYCAT-;eq2R| (|SPADfirst| |l|) |d| $) |POLYCAT-;reducedSystem;MM;20|)
     (LETT |l| (CDR |l|) |POLYCAT-;reducedSystem;MM;20|)
     (SEQ G190
      (COND
       ((NULL (COND ((NULL |l|) (QUOTE NIL)) ((QUOTE T) (QUOTE T)))) (GO G191)))
      (SEQ
       (LETT |mm|
        (SPADCALL |mm| (|POLYCAT-;eq2R| (|SPADfirst| |l|) |d| $) (QREFELT $ 93))
        |POLYCAT-;reducedSystem;MM;20|)
       (EXIT (LETT |l| (CDR |l|) |POLYCAT-;reducedSystem;MM;20|)))
      NIL
      (GO G190)
      G191
      (EXIT NIL))
     (EXIT |mm|)))))

(DEFUN |POLYCAT-;reducedSystem;MVR;21| (|m| |v| $)
 (PROG (#0=#:G1570 |s| #1=#:G1571 |b| #2=#:G1572 |bj| #3=#:G1573 |d| |n|
```

```
       |mm| |w| |l| |r|)
 (RETURN
  (SEQ
   (LETT |l| (SPADCALL |m| (QREFELT $ 92)) |POLYCAT-;reducedSystem;MVR;21|)
   (LETT |r| (SPADCALL |v| (QREFELT $ 97)) |POLYCAT-;reducedSystem;MVR;21|)
   (LETT |b|
    (SPADCALL
     (SPADCALL
      (|POLYCAT-;allMonoms| |r| $)
      (SPADCALL
       (PROGN
        (LETT #0# NIL |POLYCAT-;reducedSystem;MVR;21|)
        (SEQ
         (LETT |s| NIL |POLYCAT-;reducedSystem;MVR;21|)
         (LETT #1# |l| |POLYCAT-;reducedSystem;MVR;21|)
         G190
         (COND
          ((OR (ATOM #1#)
               (PROGN
                (LETT |s| (CAR #1#) |POLYCAT-;reducedSystem;MVR;21|)
                NIL))
           (GO G191)))
         (SEQ
          (EXIT
           (LETT #0#
            (CONS (|POLYCAT-;allMonoms| |s| $) #0#)
            |POLYCAT-;reducedSystem;MVR;21|)))
         (LETT #1# (CDR #1#) |POLYCAT-;reducedSystem;MVR;21|)
         (GO G190)
         G191
         (EXIT (NREVERSE0 #0#))))
       (QREFELT $ 79))
      (QREFELT $ 98))
     (QREFELT $ 80))
    |POLYCAT-;reducedSystem;MVR;21|)
   (LETT |d|
    (PROGN
     (LETT #2# NIL |POLYCAT-;reducedSystem;MVR;21|)
     (SEQ
      (LETT |bj| NIL |POLYCAT-;reducedSystem;MVR;21|)
      (LETT #3# |b| |POLYCAT-;reducedSystem;MVR;21|)
      G190
      (COND
       ((OR (ATOM #3#)
            (PROGN (LETT |bj| (CAR #3#) |POLYCAT-;reducedSystem;MVR;21|) NIL))
        (GO G191)))
      (SEQ
       (EXIT
        (LETT #2#
         (CONS (SPADCALL |bj| (QREFELT $ 59)) #2#)
```

```
          |POLYCAT-;reducedSystem;MVR;21|)))
       (LETT #3# (CDR #3#) |POLYCAT-;reducedSystem;MVR;21|)
       (GO G190)
       G191
       (EXIT (NREVERSE0 #2#))))
      |POLYCAT-;reducedSystem;MVR;21|)
    (LETT |n| (LENGTH |d|) |POLYCAT-;reducedSystem;MVR;21|)
    (LETT |mm|
     (|POLYCAT-;eq2R| (|SPADfirst| |l|) |d| $)
     |POLYCAT-;reducedSystem;MVR;21|)
    (LETT |w|
     (|POLYCAT-;P2R| (|SPADfirst| |r|) |d| |n| $)
     |POLYCAT-;reducedSystem;MVR;21|)
    (LETT |l| (CDR |l|) |POLYCAT-;reducedSystem;MVR;21|)
    (LETT |r| (CDR |r|) |POLYCAT-;reducedSystem;MVR;21|)
    (SEQ G190
     (COND
      ((NULL (COND ((NULL |l|) (QUOTE NIL)) ((QUOTE T) (QUOTE T))))
       (GO G191)))
     (SEQ
      (LETT |mm|
       (SPADCALL |mm| (|POLYCAT-;eq2R| (|SPADfirst| |l|) |d| $) (QREFELT $ 93))
       |POLYCAT-;reducedSystem;MVR;21|)
      (LETT |w|
       (SPADCALL |w|
        (|POLYCAT-;P2R| (|SPADfirst| |r|) |d| |n| $)
        (QREFELT $ 99))
       |POLYCAT-;reducedSystem;MVR;21|)
      (LETT |l| (CDR |l|) |POLYCAT-;reducedSystem;MVR;21|)
      (EXIT (LETT |r| (CDR |r|) |POLYCAT-;reducedSystem;MVR;21|)))
      NIL
      (GO G190)
      G191
      (EXIT NIL))
    (EXIT (CONS |mm| |w|))))))

(DEFUN |POLYCAT-;gcdPolynomial;3Sup;22| (|pp| |qq| $)
 (SPADCALL |pp| |qq| (QREFELT $ 104)))

(DEFUN |POLYCAT-;solveLinearPolynomialEquation;LSupU;23| (|lpp| |pp| $)
 (SPADCALL |lpp| |pp| (QREFELT $ 109)))

(DEFUN |POLYCAT-;factorPolynomial;SupF;24| (|pp| $)
 (SPADCALL |pp| (QREFELT $ 114)))

(DEFUN |POLYCAT-;factorSquareFreePolynomial;SupF;25| (|pp| $)
 (SPADCALL |pp| (QREFELT $ 117)))

(DEFUN |POLYCAT-;factor;SF;26| (|p| $)
 (PROG (|v| |ansR| #0=#:G1615 |w| #1=#:G1616 |up| |ansSUP| #2=#:G1617
```

```
      |ww| #3=#:G1618)
 (RETURN
  (SEQ
   (LETT |v| (SPADCALL |p| (QREFELT $ 42)) |POLYCAT-;factor;SF;26|)
   (EXIT
    (COND
     ((QEQCAR |v| 1)
      (SEQ
       (LETT |ansR|
        (SPADCALL (SPADCALL |p| (QREFELT $ 38)) (QREFELT $ 120))
        |POLYCAT-;factor;SF;26|)
       (EXIT
        (SPADCALL
         (SPADCALL (SPADCALL |ansR| (QREFELT $ 122)) (QREFELT $ 40))
         (PROGN
          (LETT #0# NIL |POLYCAT-;factor;SF;26|)
          (SEQ
           (LETT |w| NIL |POLYCAT-;factor;SF;26|)
           (LETT #1#
            (SPADCALL |ansR| (QREFELT $ 126))
            |POLYCAT-;factor;SF;26|)
           G190
           (COND
            ((OR (ATOM #1#)
                 (PROGN (LETT |w| (CAR #1#) |POLYCAT-;factor;SF;26|) NIL))
             (GO G191)))
           (SEQ
            (EXIT
             (LETT #0#
              (CONS
               (VECTOR (QVELT |w| 0)
                 (SPADCALL (QVELT |w| 1) (QREFELT $ 40)) (QVELT |w| 2))
               #0#)
              |POLYCAT-;factor;SF;26|)))
           (LETT #1# (CDR #1#) |POLYCAT-;factor;SF;26|)
           (GO G190)
           G191
           (EXIT (NREVERSE0 #0#))))
         (QREFELT $ 130)))))
     ((QUOTE T)
      (SEQ
       (LETT |up|
        (SPADCALL |p| (QCDR |v|) (QREFELT $ 47)) |POLYCAT-;factor;SF;26|)
       (LETT |ansSUP| (SPADCALL |up| (QREFELT $ 114)) |POLYCAT-;factor;SF;26|)
       (EXIT
        (SPADCALL
         (SPADCALL
          (SPADCALL |ansSUP| (QREFELT $ 131)) (QCDR |v|) (QREFELT $ 132))
         (PROGN
          (LETT #2# NIL |POLYCAT-;factor;SF;26|)
```

```
          (SEQ
           (LETT |ww| NIL |POLYCAT-;factor;SF;26|)
           (LETT #3#
            (SPADCALL |ansSUP| (QREFELT $ 135))
            |POLYCAT-;factor;SF;26|)
           G190
           (COND
            ((OR (ATOM #3#)
                 (PROGN (LETT |ww| (CAR #3#) |POLYCAT-;factor;SF;26|) NIL))
             (GO G191)))
           (SEQ
            (EXIT
             (LETT #2#
              (CONS
               (VECTOR (QVELT |ww| 0) (SPADCALL (QVELT |ww| 1) (QCDR |v|)
                       (QREFELT $ 132)) (QVELT |ww| 2))
               #2#)
              |POLYCAT-;factor;SF;26|)))
           (LETT #3# (CDR #3#) |POLYCAT-;factor;SF;26|)
           (GO G190)
           G191
           (EXIT (NREVERSE0 #2#))))
         (QREFELT $ 130))))))))))))

(DEFUN |POLYCAT-;conditionP;MU;27| (|mat| $)
 (PROG (|ll| #0=#:G1653 |z| #1=#:G1654 |ch| |l| #2=#:G1655 #3=#:G1656
        #4=#:G1625 #5=#:G1623 #6=#:G1624 #7=#:G1657 |vars| |degs|
        #8=#:G1658 |d| #9=#:G1659 |nd| #10=#:G1652 #11=#:G1632 |deg1|
        |redmons| #12=#:G1660 |v| #13=#:G1662 |u| #14=#:G1661 |llR|
        |monslist| |ans| #15=#:G1663 #16=#:G1664 |mons| #17=#:G1665 |m|
        #18=#:G1666 |i| #19=#:G1648 #20=#:G1646 #21=#:G1647)
  (RETURN
   (SEQ
    (EXIT
     (SEQ
      (LETT |ll|
       (SPADCALL (SPADCALL |mat| (QREFELT $ 137)) (QREFELT $ 92))
       |POLYCAT-;conditionP;MU;27|)
      (LETT |llR|
       (PROGN
        (LETT #0# NIL |POLYCAT-;conditionP;MU;27|)
        (SEQ
         (LETT |z| NIL |POLYCAT-;conditionP;MU;27|)
         (LETT #1# (|SPADfirst| |ll|) |POLYCAT-;conditionP;MU;27|)
         G190
         (COND
          ((OR (ATOM #1#)
               (PROGN (LETT |z| (CAR #1#) |POLYCAT-;conditionP;MU;27|) NIL))
           (GO G191)))
         (SEQ (EXIT (LETT #0# (CONS NIL #0#) |POLYCAT-;conditionP;MU;27|)))
```

```
   (LETT #1# (CDR #1#) |POLYCAT-;conditionP;MU;27|)
   (GO G190)
   G191
   (EXIT (NREVERSE0 #0#))))
 |POLYCAT-;conditionP;MU;27|)
(LETT |monslist| NIL |POLYCAT-;conditionP;MU;27|)
(LETT |ch| (SPADCALL (QREFELT $ 138)) |POLYCAT-;conditionP;MU;27|)
(SEQ
 (LETT |l| NIL |POLYCAT-;conditionP;MU;27|)
 (LETT #2# |ll| |POLYCAT-;conditionP;MU;27|)
 G190
 (COND
  ((OR (ATOM #2#)
       (PROGN (LETT |l| (CAR #2#) |POLYCAT-;conditionP;MU;27|) NIL))
   (GO G191)))
 (SEQ
  (LETT |mons|
   (PROGN
    (LETT #6# NIL |POLYCAT-;conditionP;MU;27|)
    (SEQ
     (LETT |u| NIL |POLYCAT-;conditionP;MU;27|)
     (LETT #3# |l| |POLYCAT-;conditionP;MU;27|)
     G190
     (COND
      ((OR (ATOM #3#)
           (PROGN (LETT |u| (CAR #3#) |POLYCAT-;conditionP;MU;27|) NIL))
       (GO G191)))
     (SEQ
      (EXIT
       (PROGN
        (LETT #4#
         (SPADCALL |u| (QREFELT $ 77))
         |POLYCAT-;conditionP;MU;27|)
        (COND
         (#6#
          (LETT #5#
           (SPADCALL #5# #4# (QREFELT $ 139))
           |POLYCAT-;conditionP;MU;27|))
         ((QUOTE T)
          (PROGN
           (LETT #5# #4# |POLYCAT-;conditionP;MU;27|)
           (LETT #6# (QUOTE T) |POLYCAT-;conditionP;MU;27|)))))))
     (LETT #3# (CDR #3#) |POLYCAT-;conditionP;MU;27|)
     (GO G190)
     G191
     (EXIT NIL))
    (COND (#6# #5#) ((QUOTE T) (|IdentityError| (QUOTE |setUnion|)))))
   |POLYCAT-;conditionP;MU;27|)
  (LETT |redmons| NIL |POLYCAT-;conditionP;MU;27|)
  (SEQ
```

```
(LETT |m| NIL |POLYCAT-;conditionP;MU;27|)
(LETT #7# |mons| |POLYCAT-;conditionP;MU;27|)
G190
(COND
 ((OR (ATOM #7#)
      (PROGN (LETT |m| (CAR #7#) |POLYCAT-;conditionP;MU;27|) NIL))
   (GO G191)))
(SEQ
 (LETT |vars|
  (SPADCALL |m| (QREFELT $ 31))
  |POLYCAT-;conditionP;MU;27|)
 (LETT |degs|
  (SPADCALL |m| |vars| (QREFELT $ 140))
  |POLYCAT-;conditionP;MU;27|)
 (LETT |deg1|
  (PROGN
   (LETT #8# NIL |POLYCAT-;conditionP;MU;27|)
   (SEQ
    (LETT |d| NIL |POLYCAT-;conditionP;MU;27|)
    (LETT #9# |degs| |POLYCAT-;conditionP;MU;27|)
    G190
    (COND
     ((OR (ATOM #9#)
          (PROGN
           (LETT |d| (CAR #9#) |POLYCAT-;conditionP;MU;27|)
           NIL))
      (GO G191)))
    (SEQ
     (EXIT
      (LETT #8#
       (CONS
        (SEQ
         (LETT |nd|
          (SPADCALL |d| |ch| (QREFELT $ 142))
          |POLYCAT-;conditionP;MU;27|)
         (EXIT
          (COND
           ((QEQCAR |nd| 1)
             (PROGN
               (LETT #10#
                (CONS 1 "failed") |POLYCAT-;conditionP;MU;27|)
               (GO #10#)))
           ((QUOTE T)
             (PROG1
              (LETT #11# (QCDR |nd|) |POLYCAT-;conditionP;MU;27|)
              (|check-subtype|
               (>= #11# 0) (QUOTE (|NonNegativeInteger|)) #11#))))))
        #8#)
       |POLYCAT-;conditionP;MU;27|)))))
   (LETT #9# (CDR #9#) |POLYCAT-;conditionP;MU;27|)
```

```
    (GO G190)
    G191
    (EXIT (NREVERSE0 #8#))))
   |POLYCAT-;conditionP;MU;27|)
  (LETT |redmons|
   (CONS
    (SPADCALL (|spadConstant| $ 33) |vars| |deg1| (QREFELT $ 54))
    |redmons|)
   |POLYCAT-;conditionP;MU;27|)
  (EXIT
   (LETT |llR|
    (PROGN
     (LETT #12# NIL |POLYCAT-;conditionP;MU;27|)
     (SEQ
      (LETT |v| NIL |POLYCAT-;conditionP;MU;27|)
      (LETT #13# |llR| |POLYCAT-;conditionP;MU;27|)
      (LETT |u| NIL |POLYCAT-;conditionP;MU;27|)
      (LETT #14# |l| |POLYCAT-;conditionP;MU;27|)
      G190
      (COND
       ((OR (ATOM #14#)
            (PROGN
             (LETT |u| (CAR #14#) |POLYCAT-;conditionP;MU;27|)
             NIL)
            (ATOM #13#)
            (PROGN (LETT |v| (CAR #13#) |POLYCAT-;conditionP;MU;27|) NIL))
        (GO G191)))
      (SEQ
       (EXIT
        (LETT #12#
         (CONS
          (CONS
           (SPADCALL
            (SPADCALL |u| |vars| |degs| (QREFELT $ 52))
            (QREFELT $ 143))
           |v|)
          #12#)
         |POLYCAT-;conditionP;MU;27|)))
       (LETT #14#
        (PROG1
         (CDR #14#)
         (LETT #13# (CDR #13#) |POLYCAT-;conditionP;MU;27|))
        |POLYCAT-;conditionP;MU;27|)
       (GO G190)
       G191
       (EXIT (NREVERSE0 #12#))))
    |POLYCAT-;conditionP;MU;27|)))
(LETT #7# (CDR #7#) |POLYCAT-;conditionP;MU;27|)
(GO G190)
G191
```

```
   (EXIT NIL))
  (EXIT
   (LETT |monslist|
    (CONS |redmons| |monslist|)
    |POLYCAT-;conditionP;MU;27|)))
 (LETT #2# (CDR #2#) |POLYCAT-;conditionP;MU;27|)
 (GO G190)
 G191
 (EXIT NIL))
(LETT |ans|
 (SPADCALL (SPADCALL (SPADCALL |llR| (QREFELT $ 89)) (QREFELT $ 144))
           (QREFELT $ 146))
 |POLYCAT-;conditionP;MU;27|)
(EXIT
 (COND
  ((QEQCAR |ans| 1) (CONS 1 "failed"))
  ((QUOTE T)
   (SEQ
    (LETT |i| 0 |POLYCAT-;conditionP;MU;27|)
    (EXIT
     (CONS 0
      (PRIMVEC2ARR
       (PROGN
        (LETT #15# (make-array (SIZE |monslist|))
         |POLYCAT-;conditionP;MU;27|)
        (SEQ
         (LETT #16# 0 |POLYCAT-;conditionP;MU;27|)
         (LETT |mons| NIL |POLYCAT-;conditionP;MU;27|)
         (LETT #17# |monslist| |POLYCAT-;conditionP;MU;27|)
         G190
         (COND
          ((OR (ATOM #17#)
               (PROGN
                (LETT |mons| (CAR #17#) |POLYCAT-;conditionP;MU;27|)
                NIL))
           (GO G191)))
         (SEQ
          (EXIT
           (SETELT #15# #16#
            (PROGN
             (LETT #21# NIL |POLYCAT-;conditionP;MU;27|)
             (SEQ
              (LETT |m| NIL |POLYCAT-;conditionP;MU;27|)
              (LETT #18# |mons| |POLYCAT-;conditionP;MU;27|)
              G190
              (COND
               ((OR (ATOM #18#)
                    (PROGN
                     (LETT |m| (CAR #18#) |POLYCAT-;conditionP;MU;27|)
                     NIL))
```

```
                    (GO G191)))
                  (SEQ
                   (EXIT
                    (PROGN
                     (LETT #19#
                      (SPADCALL |m|
                       (SPADCALL
                        (SPADCALL
                         (QCDR |ans|)
                         (LETT |i| (+ |i| 1) |POLYCAT-;conditionP;MU;27|)
                         (QREFELT $ 147))
                        (QREFELT $ 40))
                       (QREFELT $ 148))
                      |POLYCAT-;conditionP;MU;27|)
                     (COND
                      (#21#
                       (LETT #20#
                        (SPADCALL #20# #19# (QREFELT $ 149))
                        |POLYCAT-;conditionP;MU;27|))
                      ((QUOTE T)
                        (PROGN
                         (LETT #20# #19# |POLYCAT-;conditionP;MU;27|)
                         (LETT #21#
                          (QUOTE T)
                          |POLYCAT-;conditionP;MU;27|)))))))
                   (LETT #18# (CDR #18#) |POLYCAT-;conditionP;MU;27|)
                   (GO G190)
                   G191
                   (EXIT NIL))
                  (COND (#21# #20#) ((QUOTE T) (|spadConstant| $ 21)))))))
                (LETT #17#
                 (PROG1
                  (CDR #17#)
                  (LETT #16# (QSADD1 #16#) |POLYCAT-;conditionP;MU;27|))
                 |POLYCAT-;conditionP;MU;27|)
                (GO G190)
                G191
                (EXIT NIL))
               #15#)))))))))
   #10#
   (EXIT #10#)))))

(DEFUN |POLYCAT-;charthRoot;SU;28| (|p| $)
 (PROG (|vars| |ans| |ch|)
  (RETURN
   (SEQ
    (LETT |vars| (SPADCALL |p| (QREFELT $ 31)) |POLYCAT-;charthRoot;SU;28|)
    (EXIT
     (COND
      ((NULL |vars|)
```

```
        (SEQ
         (LETT |ans|
          (SPADCALL (SPADCALL |p| (QREFELT $ 143)) (QREFELT $ 151))
          |POLYCAT-;charthRoot;SU;28|)
         (EXIT
          (COND
           ((QEQCAR |ans| 1) (CONS 1 "failed"))
           ((QUOTE T) (CONS 0 (SPADCALL (QCDR |ans|) (QREFELT $ 40))))))))
      ((QUOTE T)
       (SEQ
        (LETT |ch| (SPADCALL (QREFELT $ 138)) |POLYCAT-;charthRoot;SU;28|)
        (EXIT (|POLYCAT-;charthRootlv| |p| |vars| |ch| $))))))))))

(DEFUN |POLYCAT-;charthRootlv| (|p| |vars| |ch| $)
 (PROG (|v| |dd| |cp| |d| #0=#:G1687 |ans| |ansx| #1=#:G1694)
  (RETURN
   (SEQ
    (EXIT
     (COND
      ((NULL |vars|)
       (SEQ
        (LETT |ans|
         (SPADCALL (SPADCALL |p| (QREFELT $ 143)) (QREFELT $ 151))
         |POLYCAT-;charthRootlv|)
        (EXIT
         (COND
          ((QEQCAR |ans| 1) (CONS 1 "failed"))
          ((QUOTE T) (CONS 0 (SPADCALL (QCDR |ans|) (QREFELT $ 40))))))))
      ((QUOTE T)
       (SEQ
        (LETT |v| (|SPADfirst| |vars|) |POLYCAT-;charthRootlv|)
        (LETT |vars| (CDR |vars|) |POLYCAT-;charthRootlv|)
        (LETT |d| (SPADCALL |p| |v| (QREFELT $ 36)) |POLYCAT-;charthRootlv|)
        (LETT |ans| (|spadConstant| $ 21) |POLYCAT-;charthRootlv|)
        (SEQ G190
         (COND ((NULL (< 0 |d|)) (GO G191)))
         (SEQ
          (LETT |dd|
           (SPADCALL |d| |ch| (QREFELT $ 142))
           |POLYCAT-;charthRootlv|)
          (EXIT
           (COND
            ((QEQCAR |dd| 1)
              (PROGN
               (LETT #1# (CONS 1 "failed") |POLYCAT-;charthRootlv|)
               (GO #1#)))
            ((QUOTE T)
             (SEQ
              (LETT |cp|
               (SPADCALL |p| |v| |d| (QREFELT $ 154))
```

```
        |POLYCAT-;charthRootlv|)
       (LETT |p|
        (SPADCALL |p|
         (SPADCALL |cp| |v| |d| (QREFELT $ 37))
         (QREFELT $ 155))
        |POLYCAT-;charthRootlv|)
       (LETT |ansx|
        (|POLYCAT-;charthRootlv| |cp| |vars| |ch| $)
        |POLYCAT-;charthRootlv|)
       (EXIT
        (COND
         ((QEQCAR |ansx| 1)
           (PROGN
            (LETT #1# (CONS 1 "failed") |POLYCAT-;charthRootlv|)
            (GO #1#)))
         ((QUOTE T)
          (SEQ
           (LETT |d|
            (SPADCALL |p| |v| (QREFELT $ 36))
            |POLYCAT-;charthRootlv|)
           (EXIT
            (LETT |ans|
             (SPADCALL |ans|
              (SPADCALL (QCDR |ansx|) |v|
               (PROG1
                (LETT #0# (QCDR |dd|) |POLYCAT-;charthRootlv|)
                (|check-subtype| (>= #0# 0)
                 (QUOTE (|NonNegativeInteger|)) #0#))
               (QREFELT $ 37))
              (QREFELT $ 149))
             |POLYCAT-;charthRootlv|)))))))))))
 NIL
 (GO G190)
 G191
 (EXIT NIL))
(LETT |ansx|
 (|POLYCAT-;charthRootlv| |p| |vars| |ch| $)
 |POLYCAT-;charthRootlv|)
(EXIT
 (COND
  ((QEQCAR |ansx| 1)
   (PROGN
    (LETT #1# (CONS 1 "failed") |POLYCAT-;charthRootlv|)
    (GO #1#)))
  ((QUOTE T)
   (PROGN
    (LETT #1#
     (CONS 0 (SPADCALL |ans| (QCDR |ansx|) (QREFELT $ 149)))
     |POLYCAT-;charthRootlv|)
    (GO #1#)))))))))
```

```
    #1#
    (EXIT #1#)))))

(DEFUN |POLYCAT-;monicDivide;2SVarSetR;30| (|p1| |p2| |mvar| $)
 (PROG (|result|)
  (RETURN
   (SEQ
    (LETT |result|
     (SPADCALL
      (SPADCALL |p1| |mvar| (QREFELT $ 47))
      (SPADCALL |p2| |mvar| (QREFELT $ 47))
      (QREFELT $ 157))
     |POLYCAT-;monicDivide;2SVarSetR;30|)
    (EXIT
     (CONS
      (SPADCALL (QCAR |result|) |mvar| (QREFELT $ 132))
      (SPADCALL (QCDR |result|) |mvar| (QREFELT $ 132))))))))

(DEFUN |POLYCAT-;squareFree;SF;31| (|p| $)
 (SPADCALL |p| (QREFELT $ 160)))

(DEFUN |POLYCAT-;squareFree;SF;32| (|p| $)
 (SPADCALL |p| (QREFELT $ 163)))

(DEFUN |POLYCAT-;squareFree;SF;33| (|p| $)
 (SPADCALL |p| (QREFELT $ 163)))

(DEFUN |POLYCAT-;squareFreePart;2S;34| (|p| $)
 (PROG (|s| |f| #0=#:G1710 #1=#:G1708 #2=#:G1706 #3=#:G1707)
  (RETURN
   (SEQ
    (SPADCALL
     (SPADCALL
      (LETT |s| (SPADCALL |p| (QREFELT $ 164)) |POLYCAT-;squareFreePart;2S;34|)
      (QREFELT $ 165))
     (PROGN
      (LETT #3# NIL |POLYCAT-;squareFreePart;2S;34|)
      (SEQ
       (LETT |f| NIL |POLYCAT-;squareFreePart;2S;34|)
       (LETT #0# (SPADCALL |s| (QREFELT $ 168)) |POLYCAT-;squareFreePart;2S;34|)
       G190
       (COND
        ((OR (ATOM #0#)
             (PROGN (LETT |f| (CAR #0#) |POLYCAT-;squareFreePart;2S;34|) NIL))
         (GO G191)))
       (SEQ
        (EXIT
         (PROGN
          (LETT #1# (QCAR |f|) |POLYCAT-;squareFreePart;2S;34|)
          (COND
```

```
           (#3#
            (LETT #2#
             (SPADCALL #2# #1# (QREFELT $ 148))
             |POLYCAT-;squareFreePart;2S;34|))
           ((QUOTE T)
            (PROGN
             (LETT #2# #1# |POLYCAT-;squareFreePart;2S;34|)
             (LETT #3# (QUOTE T) |POLYCAT-;squareFreePart;2S;34|)))))))
       (LETT #0# (CDR #0#) |POLYCAT-;squareFreePart;2S;34|)
       (GO G190)
       G191
       (EXIT NIL))
      (COND (#3# #2#) ((QUOTE T) (|spadConstant| $ 33))))
     (QREFELT $ 148))))))

(DEFUN |POLYCAT-;content;SVarSetS;35| (|p| |v| $)
 (SPADCALL (SPADCALL |p| |v| (QREFELT $ 47)) (QREFELT $ 170)))

(DEFUN |POLYCAT-;primitivePart;2S;36| (|p| $)
 (PROG (#0=#:G1713)
  (RETURN
   (COND
    ((SPADCALL |p| (QREFELT $ 172)) |p|)
    ((QUOTE T)
     (QVELT
      (SPADCALL
       (PROG2
        (LETT #0#
         (SPADCALL |p| (SPADCALL |p| (QREFELT $ 173)) (QREFELT $ 174))
         |POLYCAT-;primitivePart;2S;36|)
        (QCDR #0#)
        (|check-union| (QEQCAR #0# 0) (QREFELT $ 6) #0#))
       (QREFELT $ 176))
      1))))))

(DEFUN |POLYCAT-;primitivePart;SVarSetS;37| (|p| |v| $)
 (PROG (#0=#:G1720)
  (RETURN
   (COND
    ((SPADCALL |p| (QREFELT $ 172)) |p|)
    ((QUOTE T)
     (QVELT
      (SPADCALL
       (PROG2
        (LETT #0#
         (SPADCALL |p| (SPADCALL |p| |v| (QREFELT $ 178)) (QREFELT $ 179))
         |POLYCAT-;primitivePart;SVarSetS;37|)
        (QCDR #0#)
        (|check-union| (QEQCAR #0# 0) (QREFELT $ 6) #0#))
       (QREFELT $ 176))
```

```
      1))))))

(DEFUN |POLYCAT-;<;2SB;38| (|p| |q| $)
 (PROG (|dp| |dq|)
  (RETURN
   (SEQ
    (LETT |dp| (SPADCALL |p| (QREFELT $ 59)) |POLYCAT-;<;2SB;38|)
    (LETT |dq| (SPADCALL |q| (QREFELT $ 59)) |POLYCAT-;<;2SB;38|)
    (EXIT
     (COND
      ((SPADCALL |dp| |dq| (QREFELT $ 181))
        (SPADCALL
         (|spadConstant| $ 22)
         (SPADCALL |q| (QREFELT $ 38))
         (QREFELT $ 182)))
      ((SPADCALL |dq| |dp| (QREFELT $ 181))
        (SPADCALL
         (SPADCALL |p| (QREFELT $ 38))
         (|spadConstant| $ 22)
         (QREFELT $ 182)))
      ((QUOTE T)
       (SPADCALL
        (SPADCALL (SPADCALL |p| |q| (QREFELT $ 155)) (QREFELT $ 38))
        (|spadConstant| $ 22)
        (QREFELT $ 182)))))))))

(DEFUN |POLYCAT-;patternMatch;SP2Pmr;39| (|p| |pat| |l| $)
 (SPADCALL |p| |pat| |l| (QREFELT $ 187)))

(DEFUN |POLYCAT-;patternMatch;SP2Pmr;40| (|p| |pat| |l| $)
 (SPADCALL |p| |pat| |l| (QREFELT $ 193)))

(DEFUN |POLYCAT-;convert;SP;41| (|x| $)
 (SPADCALL (ELT $ 196) (ELT $ 197) |x| (QREFELT $ 201)))

(DEFUN |POLYCAT-;convert;SP;42| (|x| $)
 (SPADCALL (ELT $ 203) (ELT $ 204) |x| (QREFELT $ 208)))

(DEFUN |POLYCAT-;convert;SIf;43| (|p| $)
 (SPADCALL (ELT $ 211) (ELT $ 212) |p| (QREFELT $ 216)))

(DEFUN |PolynomialCategory&| (|#1| |#2| |#3| |#4|)
 (PROG (DV$1 DV$2 DV$3 DV$4 |dv$| $ |pv$|)
  (RETURN
   (PROGN
    (LETT DV$1 (|devaluate| |#1|) . #0=(|PolynomialCategory&|))
    (LETT DV$2 (|devaluate| |#2|) . #0#)
    (LETT DV$3 (|devaluate| |#3|) . #0#)
    (LETT DV$4 (|devaluate| |#4|) . #0#)
    (LETT |dv$| (LIST (QUOTE |PolynomialCategory&|) DV$1 DV$2 DV$3 DV$4) . #0#)
```

```
(LETT $ (make-array 226) . #0#)
(QSETREFV $ 0 |dv$|)
(QSETREFV $ 3
 (LETT |pv$|
  (|buildPredVector| 0 0
   (LIST
    (|HasCategory| |#2| (QUOTE (|PolynomialFactorizationExplicit|)))
    (|HasAttribute| |#2| (QUOTE |canonicalUnitNormal|))
    (|HasCategory| |#2| (QUOTE (|GcdDomain|)))
    (|HasCategory| |#2| (QUOTE (|CommutativeRing|)))
    (|HasCategory| |#4| (QUOTE (|PatternMatchable| (|Float|))))
    (|HasCategory| |#2| (QUOTE (|PatternMatchable| (|Float|))))
    (|HasCategory| |#4| (QUOTE (|PatternMatchable| (|Integer|))))
    (|HasCategory| |#2| (QUOTE (|PatternMatchable| (|Integer|))))
    (|HasCategory| |#4| (QUOTE (|ConvertibleTo| (|Pattern| (|Float|)))))
    (|HasCategory| |#2| (QUOTE (|ConvertibleTo| (|Pattern| (|Float|)))))
    (|HasCategory| |#4| (QUOTE (|ConvertibleTo| (|Pattern| (|Integer|)))))
    (|HasCategory| |#2| (QUOTE (|ConvertibleTo| (|Pattern| (|Integer|)))))
    (|HasCategory| |#4| (QUOTE (|ConvertibleTo| (|InputForm|))))
    (|HasCategory| |#2| (QUOTE (|ConvertibleTo| (|InputForm|))))
    (|HasCategory| |#2| (QUOTE (|OrderedSet|)))))
 . #0#))
(|stuffDomainSlots| $)
(QSETREFV $ 6 |#1|)
(QSETREFV $ 7 |#2|)
(QSETREFV $ 8 |#3|)
(QSETREFV $ 9 |#4|)
(COND
 ((|testBitVector| |pv$| 4)
  (PROGN
   (QSETREFV $ 74
    (CONS (|dispatchFunction| |POLYCAT-;resultant;2SVarSetS;15|) $))
   (QSETREFV $ 76
    (CONS (|dispatchFunction| |POLYCAT-;discriminant;SVarSetS;16|) $)))))
(COND
 ((|HasCategory| |#2| (QUOTE (|IntegralDomain|)))
  (PROGN
   (QSETREFV $ 95
    (CONS (|dispatchFunction| |POLYCAT-;reducedSystem;MM;20|) $))
   (QSETREFV $ 102
    (CONS (|dispatchFunction| |POLYCAT-;reducedSystem;MVR;21|) $)))))
(COND
 ((|testBitVector| |pv$| 1)
  (PROGN
   (QSETREFV $ 105
    (CONS (|dispatchFunction| |POLYCAT-;gcdPolynomial;3Sup;22|) $))
   (QSETREFV $ 112
    (CONS
     (|dispatchFunction|
       |POLYCAT-;solveLinearPolynomialEquation;LSupU;23|)
```

```
      $))
    (QSETREFV $ 116
     (CONS (|dispatchFunction| |POLYCAT-;factorPolynomial;SupF;24|) $))
    (QSETREFV $ 118
     (CONS
      (|dispatchFunction| |POLYCAT-;factorSquareFreePolynomial;SupF;25|)
      $))
    (QSETREFV $ 136 (CONS (|dispatchFunction| |POLYCAT-;factor;SF;26|) $))
    (COND
     ((|HasCategory| |#2| (QUOTE (|CharacteristicNonZero|)))
      (PROGN
       (QSETREFV $ 150
        (CONS (|dispatchFunction| |POLYCAT-;conditionP;MU;27|) $))))))))
(COND
 ((|HasCategory| |#2| (QUOTE (|CharacteristicNonZero|)))
  (PROGN
   (QSETREFV $ 152
    (CONS (|dispatchFunction| |POLYCAT-;charthRoot;SU;28|) $)))))
(COND
 ((|testBitVector| |pv$| 3)
  (PROGN
   (COND
    ((|HasCategory| |#2| (QUOTE (|EuclideanDomain|)))
     (COND
      ((|HasCategory| |#2| (QUOTE (|CharacteristicZero|)))
       (QSETREFV $ 161
        (CONS (|dispatchFunction| |POLYCAT-;squareFree;SF;31|) $)))
      ((QUOTE T)
       (QSETREFV $ 161
        (CONS (|dispatchFunction| |POLYCAT-;squareFree;SF;32|) $)))))
    ((QUOTE T)
     (QSETREFV $ 161
      (CONS (|dispatchFunction| |POLYCAT-;squareFree;SF;33|) $))))
   (QSETREFV $ 169
    (CONS (|dispatchFunction| |POLYCAT-;squareFreePart;2S;34|) $))
   (QSETREFV $ 171
    (CONS (|dispatchFunction| |POLYCAT-;content;SVarSetS;35|) $))
   (QSETREFV $ 177
    (CONS (|dispatchFunction| |POLYCAT-;primitivePart;2S;36|) $))
   (QSETREFV $ 180
    (CONS (|dispatchFunction| |POLYCAT-;primitivePart;SVarSetS;37|) $)))))
(COND
 ((|testBitVector| |pv$| 15)
  (PROGN
   (QSETREFV $ 183 (CONS (|dispatchFunction| |POLYCAT-;<;2SB;38|) $))
   (COND
    ((|testBitVector| |pv$| 8)
     (COND
      ((|testBitVector| |pv$| 7)
       (QSETREFV $ 189
```

```
            (CONS
             (|dispatchFunction| |POLYCAT-;patternMatch;SP2Pmr;39|)
             $))))))
       (COND
        ((|testBitVector| |pv$| 6)
         (COND
         ((|testBitVector| |pv$| 5)
          (QSETREFV $ 195
           (CONS
            (|dispatchFunction| |POLYCAT-;patternMatch;SP2Pmr;40|)
            $)))))))))
    (COND
     ((|testBitVector| |pv$| 12)
      (COND
       ((|testBitVector| |pv$| 11)
        (QSETREFV $ 202
         (CONS (|dispatchFunction| |POLYCAT-;convert;SP;41|) $))))))
    (COND
     ((|testBitVector| |pv$| 10)
      (COND
       ((|testBitVector| |pv$| 9)
        (QSETREFV $ 209
         (CONS (|dispatchFunction| |POLYCAT-;convert;SP;42|) $))))))
    (COND
     ((|testBitVector| |pv$| 14)
      (COND
       ((|testBitVector| |pv$| 13)
        (QSETREFV $ 217
         (CONS (|dispatchFunction| |POLYCAT-;convert;SIf;43|) $))))))
    $))))

(setf (get
 (QUOTE |PolynomialCategory&|)
 (QUOTE |infovec|))
 (LIST (QUOTE
  #(NIL NIL NIL NIL NIL NIL (|local| |#1|) (|local| |#2|) (|local| |#3|)
    (|local| |#4|) (|Equation| 6) (0 . |lhs|) (|Union| 9 (QUOTE "failed"))
    (5 . |retractIfCan|) (10 . |retract|) (15 . |rhs|) (|List| 9) (|List| $)
    (20 . |eval|) (|List| 221) |POLYCAT-;eval;SLS;1| (27 . |Zero|)
    (31 . |Zero|) (|Boolean|) (35 . =) (41 . |leadingMonomial|)
    (46 . |reductum|) |POLYCAT-;monomials;SL;2| (51 . |monomials|)
    (|Union| 17 (QUOTE "failed")) |POLYCAT-;isPlus;SU;3| (56 . |variables|)
    (61 . |monomial?|) (66 . |One|) (70 . |One|) (|NonNegativeInteger|)
    (74 . |degree|) (80 . |monomial|) (87 . |leadingCoefficient|) (92 . =)
    (98 . |coerce|) |POLYCAT-;isTimes;SU;4| (103 . |mainVariable|)
    (|Record| (|:| |var| 9) (|:| |exponent| 35))
    (|Union| 43 (QUOTE "failed")) |POLYCAT-;isExpt;SU;5|
    (|SparseUnivariatePolynomial| $) (108 . |univariate|)
    (|SparseUnivariatePolynomial| 6) (114 . |coefficient|)
    |POLYCAT-;coefficient;SVarSetNniS;6| (|List| 35) (120 . |coefficient|)
```

```
|POLYCAT-;coefficient;SLLS;7| (127 . |monomial|)
|POLYCAT-;monomial;SLLS;8| (134 . |coerce|)
|POLYCAT-;retract;SVarSet;9| |POLYCAT-;retractIfCan;SU;10|
(139 . |degree|) (144 . |monomial|) |POLYCAT-;primitiveMonomials;SL;12|
(150 . |ground?|) (155 . |Zero|) (159 . =) (165 . |degree|)
(170 . |leadingCoefficient|) (175 . |totalDegree|) (180 . |reductum|)
|POLYCAT-;totalDegree;SNni;13| (185 . |member?|) (191 . |totalDegree|)
|POLYCAT-;totalDegree;SLNni;14| (197 . |resultant|) (203 . |resultant|)
(210 . |discriminant|) (215 . |discriminant|) (221 . |primitiveMonomials|)
(|List| 6) (226 . |concat|) (231 . |removeDuplicates!|) (|Vector| 7)
(236 . |new|) (|Integer|) (242 . |minIndex|) (247 . |coefficient|)
(253 . |qsetelt!|) (|List| 220) (|Matrix| 7) (260 . |matrix|)
(|List| 78) (|Matrix| 6) (265 . |listOfLists|) (270 . |vertConcat|)
(|Matrix| $) (276 . |reducedSystem|) (|Vector| 6) (281 . |entries|)
(286 . |concat|) (292 . |concat|)
(|Record| (|:| |mat| 88) (|:| |vec| 81)) (|Vector| $)
(298 . |reducedSystem|) (|GeneralPolynomialGcdPackage| 8 9 7 6)
(304 . |gcdPolynomial|) (310 . |gcdPolynomial|)
(|Union| 107 (QUOTE "failed")) (|List| 48)
(|PolynomialFactorizationByRecursion| 7 8 9 6)
(316 . |solveLinearPolynomialEquationByRecursion|)
(|Union| 111 (QUOTE "failed")) (|List| 46)
(322 . |solveLinearPolynomialEquation|) (|Factored| 48)
(328 . |factorByRecursion|) (|Factored| 46) (333 . |factorPolynomial|)
(338 . |factorSquareFreeByRecursion|)
(343 . |factorSquareFreePolynomial|) (|Factored| $) (348 . |factor|)
(|Factored| 7) (353 . |unit|)
(|Union| (QUOTE "nil") (QUOTE "sqfr") (QUOTE "irred") (QUOTE "prime"))
(|Record| (|:| |flg| 123) (|:| |fctr| 7) (|:| |xpnt| 83))
(|List| 124) (358 . |factorList|)
(|Record| (|:| |flg| 123) (|:| |fctr| 6) (|:| |xpnt| 83))
(|List| 127) (|Factored| 6) (363 . |makeFR|) (369 . |unit|)
(374 . |multivariate|)
(|Record| (|:| |flg| 123) (|:| |fctr| 48) (|:| |xpnt| 83))
(|List| 133) (380 . |factorList|) (385 . |factor|) (390 . |transpose|)
(395 . |characteristic|) (399 . |setUnion|) (405 . |degree|)
(|Union| $ (QUOTE "failed")) (411 . |exquo|) (417 . |ground|)
(422 . |transpose|) (|Union| 101 (QUOTE "failed")) (427 . |conditionP|)
(432 . |elt|) (438 . *) (444 . +) (450 . |conditionP|)
(455 . |charthRoot|) (460 . |charthRoot|) (465 . |Zero|)
(469 . |coefficient|) (476 . -)
(|Record| (|:| |quotient| $) (|:| |remainder| $))
(482 . |monicDivide|) |POLYCAT-;monicDivide;2SVarSetR;30|
(|MultivariateSquareFree| 8 9 7 6) (488 . |squareFree|)
(493 . |squareFree|) (|PolynomialSquareFree| 9 8 7 6)
(498 . |squareFree|) (503 . |squareFree|) (508 . |unit|)
(|Record| (|:| |factor| 6) (|:| |exponent| 83)) (|List| 166)
(513 . |factors|) (518 . |squareFreePart|) (523 . |content|)
(528 . |content|) (534 . |zero?|) (539 . |content|) (544 . |exquo|)
(|Record| (|:| |unit| $) (|:| |canonical| $) (|:| |associate| $))
```

```
   (550 . |unitNormal|) (555 . |primitivePart|) (560 . |content|)
   (566 . |exquo|) (572 . |primitivePart|) (578 . <) (584 . <) (590 . <)
   (|PatternMatchResult| 83 6) (|Pattern| 83)
   (|PatternMatchPolynomialCategory| 83 8 9 7 6) (596 . |patternMatch|)
   (|PatternMatchResult| 83 $) (603 . |patternMatch|)
   (|PatternMatchResult| (|Float|) 6) (|Pattern| (|Float|))
   (|PatternMatchPolynomialCategory| (|Float|) 8 9 7 6)
   (610 . |patternMatch|) (|PatternMatchResult| (|Float|) $)
   (617 . |patternMatch|) (624 . |convert|) (629 . |convert|)
   (|Mapping| 185 9) (|Mapping| 185 7)
   (|PolynomialCategoryLifting| 8 9 7 6 185) (634 . |map|)
   (641 . |convert|) (646 . |convert|) (651 . |convert|) (|Mapping| 191 9)
   (|Mapping| 191 7) (|PolynomialCategoryLifting| 8 9 7 6 191)
   (656 . |map|) (663 . |convert|) (|InputForm|) (668 . |convert|)
   (673 . |convert|) (|Mapping| 210 9) (|Mapping| 210 7)
   (|PolynomialCategoryLifting| 8 9 7 6 210) (678 . |map|)
   (685 . |convert|) (|Record| (|:| |mat| 219) (|:| |vec| (|Vector| 83)))
   (|Matrix| 83) (|List| 7) (|Equation| $) (|Union| 83 (QUOTE "failed"))
   (|Union| 224 (QUOTE "failed")) (|Fraction| 83)
   (|Union| 7 (QUOTE "failed"))))
 (QUOTE #(|totalDegree| 690 |squareFreePart| 701 |squareFree| 706
  |solveLinearPolynomialEquation| 711 |retractIfCan| 717 |retract| 722
  |resultant| 727 |reducedSystem| 734 |primitivePart| 745
  |primitiveMonomials| 756 |patternMatch| 761 |monomials| 775
  |monomial| 780 |monicDivide| 787 |isTimes| 794 |isPlus| 799
  |isExpt| 804 |gcdPolynomial| 809 |factorSquareFreePolynomial| 815
  |factorPolynomial| 820 |factor| 825 |eval| 830 |discriminant| 836
  |convert| 842 |content| 857 |conditionP| 863 |coefficient| 868
  |charthRoot| 882 < 887))
 (QUOTE NIL)
 (CONS (|makeByteWordVec2| 1 (QUOTE NIL))
  (CONS (QUOTE #())
   (CONS (QUOTE #())
    (|makeByteWordVec2| 217 (QUOTE
     (1 10 6 0 11 1 6 12 0 13 1 6 9 0 14 1 10 6 0 15 3 6 0 0 16 17 18 0 6 0
      21 0 7 0 22 2 6 23 0 0 24 1 6 0 0 25 1 6 0 0 26 1 6 17 0 28 1 6 16 0
      31 1 6 23 0 32 0 6 0 33 0 7 0 34 2 6 35 0 9 36 3 6 0 0 9 35 37 1 6 7
      0 38 2 7 23 0 0 39 1 6 0 7 40 1 6 12 0 42 2 6 46 0 9 47 2 48 6 0 35
      49 3 6 0 0 16 51 52 3 6 0 0 16 51 54 1 6 0 9 56 1 6 8 0 59 2 6 0 7 8
      60 1 6 23 0 62 0 48 0 63 2 48 23 0 0 64 1 48 35 0 65 1 48 6 0 66 1 6
      35 0 67 1 48 0 0 68 2 16 23 9 0 70 2 6 35 0 16 71 2 48 6 0 0 73 3 0
      0 0 0 9 74 1 48 6 0 75 2 0 0 0 9 76 1 6 17 0 77 1 78 0 17 79 1 78 0
      0 80 2 81 0 35 7 82 1 81 83 0 84 2 6 7 0 8 85 3 81 7 0 83 7 86 1 88
      0 87 89 1 91 90 0 92 2 88 0 0 0 93 1 0 88 94 95 1 96 78 0 97 2 78 0
      0 0 98 2 81 0 0 0 99 2 0 100 94 101 102 2 103 48 48 48 104 2 0 46 46
      46 105 2 108 106 107 48 109 2 0 110 111 46 112 1 108 113 48 114 1 0
      115 46 116 1 108 113 48 117 1 0 115 46 118 1 7 119 0 120 1 121 7 0
      122 1 121 125 0 126 2 129 0 6 128 130 1 113 48 0 131 2 6 0 46 9 132
      1 113 134 0 135 1 0 119 0 136 1 91 0 0 137 0 6 35 138 2 78 0 0 0 139
      2 6 51 0 16 140 2 83 141 0 0 142 1 6 7 0 143 1 88 0 0 144 1 7 145 94
```

```
      146 2 81 7 0 83 147 2 6 0 0 0 148 2 6 0 0 0 149 1 0 145 94 150 1 7
      141 0 151 1 0 141 0 152 0 8 0 153 3 6 0 0 9 35 154 2 6 0 0 0 155 2
      48 156 0 0 157 1 159 129 6 160 1 0 119 0 161 1 162 129 6 163 1 6 119
      0 164 1 129 6 0 165 1 129 167 0 168 1 0 0 0 169 1 48 6 0 170 2 0 0 0
      9 171 1 6 23 0 172 1 6 7 0 173 2 6 141 0 7 174 1 6 175 0 176 1 0 0 0
      177 2 6 0 0 9 178 2 6 141 0 0 179 2 0 0 0 9 180 2 8 23 0 0 181 2 7 23
      0 0 182 2 0 23 0 0 183 3 186 184 6 185 184 187 3 0 188 0 185 188 189
      3 192 190 6 191 190 193 3 0 194 0 191 194 195 1 9 185 0 196 1 7 185
      0 197 3 200 185 198 199 6 201 1 0 185 0 202 1 9 191 0 203 1 7 191 0
      204 3 207 191 205 206 6 208 1 0 191 0 209 1 9 210 0 211 1 7 210 0
      212 3 215 210 213 214 6 216 1 0 210 0 217 2 0 35 0 16 72 1 0 35 0 69
      1 0 0 0 169 1 0 119 0 161 2 0 110 111 46 112 1 0 12 0 58 1 0 9 0 57
      3 0 0 0 0 9 74 1 0 88 94 95 2 0 100 94 101 102 2 0 0 0 9 180 1 0 0 0
      177 1 0 17 0 61 3 0 188 0 185 188 189 3 0 194 0 191 194 195 1 0 17 0
      27 3 0 0 0 16 51 55 3 0 156 0 0 9 158 1 0 29 0 41 1 0 29 0 30 1 0 44
      0 45 2 0 46 46 46 105 1 0 115 46 118 1 0 115 46 116 1 0 119 0 136 2
      0 0 0 19 20 2 0 0 0 9 76 1 0 210 0 217 1 0 185 0 202 1 0 191 0 209 2
      0 0 0 9 171 1 0 145 94 150 3 0 0 0 16 51 53 3 0 0 0 9 35 50 1 0 141
      0 152 2 0 23 0 0 183))))))
  (QUOTE |lookupComplete|)))
```

———-

21.45 PSETCAT.lsp BOOTSTRAP

PSETCAT depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **PSETCAT** category which we can write into the **MID** directory. We compile the lisp code and copy the **PSETCAT.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— PSETCAT.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |PolynomialSetCategory;CAT| (QUOTE NIL))

(SETQ |PolynomialSetCategory;AL| (QUOTE NIL))

(DEFUN |PolynomialSetCategory| (|&REST| #1=#:G82375 |&AUX| #2=#:G82373)
 (DSETQ #2# #1#)
 (LET (#3=#:G82374)
  (COND
   ((SETQ #3# (|assoc| (|devaluateList| #2#) |PolynomialSetCategory;AL|))
     (CDR #3#))
```

```
   (T
    (SETQ |PolynomialSetCategory;AL|
     (|cons5|
      (CONS
       (|devaluateList| #2#)
       (SETQ #3# (APPLY (FUNCTION |PolynomialSetCategory;|) #2#)))
      |PolynomialSetCategory;AL|))
    #3#))))

(DEFUN |PolynomialSetCategory;| (|t#1| |t#2| |t#3| |t#4|)
 (PROG (#1=#:G82372)
  (RETURN
   (PROG1
    (LETT #1#
     (|sublisV|
      (PAIR
       (QUOTE (|t#1| |t#2| |t#3| |t#4|))
       (LIST
        (|devaluate| |t#1|)
        (|devaluate| |t#2|)
        (|devaluate| |t#3|)
        (|devaluate| |t#4|)))
      (|sublisV|
       (PAIR (QUOTE (#2=#:G82371)) (LIST (QUOTE (|List| |t#4|))))
       (COND
        (|PolynomialSetCategory;CAT|)
        ((QUOTE T)
         (LETT |PolynomialSetCategory;CAT|
          (|Join|
           (|SetCategory|)
           (|Collection| (QUOTE |t#4|))
           (|CoercibleTo| (QUOTE #2#))
           (|mkCategory|
            (QUOTE |domain|)
            (QUOTE (
             ((|retractIfCan| ((|Union| |$| "failed") (|List| |t#4|))) T)
             ((|retract| (|$| (|List| |t#4|))) T)
             ((|mvar| (|t#3| |$|)) T)
             ((|variables| ((|List| |t#3|) |$|)) T)
             ((|mainVariables| ((|List| |t#3|) |$|)) T)
             ((|mainVariable?| ((|Boolean|) |t#3| |$|)) T)
             ((|collectUnder| (|$| |$| |t#3|)) T)
             ((|collect| (|$| |$| |t#3|)) T)
             ((|collectUpper| (|$| |$| |t#3|)) T)
             ((|sort|
              ((|Record|
                (|:| |under| |$|)
                (|:| |floor| |$|)
                (|:| |upper| |$|)) |$| |t#3|))
               T)
```

```
           ((|trivialIdeal?| ((|Boolean|) |$|)) T)
           ((|roughBase?| ((|Boolean|) |$|))
             (|has| |t#1| (|IntegralDomain|)))
           ((|roughSubIdeal?| ((|Boolean|) |$| |$|))
             (|has| |t#1| (|IntegralDomain|)))
           ((|roughEqualIdeals?| ((|Boolean|) |$| |$|))
             (|has| |t#1| (|IntegralDomain|)))
           ((|roughUnitIdeal?| ((|Boolean|) |$|))
             (|has| |t#1| (|IntegralDomain|)))
           ((|headRemainder|
             ((|Record|
               (|:| |num| |t#4|)
               (|:| |den| |t#1|)) |t#4| |$|))
              (|has| |t#1| (|IntegralDomain|)))
           ((|remainder|
             ((|Record|
               (|:| |rnum| |t#1|)
               (|:| |polnum| |t#4|)
               (|:| |den| |t#1|)) |t#4| |$|))
              (|has| |t#1| (|IntegralDomain|)))
           ((|rewriteIdealWithHeadRemainder|
            ((|List| |t#4|) (|List| |t#4|) |$|))
              (|has| |t#1| (|IntegralDomain|)))
           ((|rewriteIdealWithRemainder| ((|List| |t#4|) (|List| |t#4|) |$|))
             (|has| |t#1| (|IntegralDomain|)))
           ((|triangular?| ((|Boolean|) |$|))
             (|has| |t#1| (|IntegralDomain|)))))
          (QUOTE ((|finiteAggregate| T)))
          (QUOTE ((|Boolean|) (|List| |t#4|) (|List| |t#3|)))
          NIL))
   . #3=(|PolynomialSetCategory|))))))
 . #3#)
(SETELT #1# 0
 (LIST
  (QUOTE |PolynomialSetCategory|)
  (|devaluate| |t#1|)
  (|devaluate| |t#2|)
  (|devaluate| |t#3|)
  (|devaluate| |t#4|)))))))
```

———-

21.46 PSETCAT-.lsp BOOTSTRAP

PSETCAT- depends on **PSETCAT**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **PSETCAT-** category which we can write into the **MID** directory. We compile the lisp code and copy the **PSETCAT-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— PSETCAT-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |PSETCAT-;elements| (|ps| |$|)
 (PROG (|lp|)
  (RETURN
   (LETT |lp| (SPADCALL |ps| (QREFELT |$| 12)) |PSETCAT-;elements|))))

(DEFUN |PSETCAT-;variables1| (|lp| |$|)
 (PROG (#1=#:G82392 |p| #2=#:G82393 |lvars|)
  (RETURN
   (SEQ
    (LETT |lvars|
     (PROGN
      (LETT #1# NIL |PSETCAT-;variables1|)
      (SEQ
       (LETT |p| NIL |PSETCAT-;variables1|)
       (LETT #2# |lp| |PSETCAT-;variables1|)
       G190
       (COND
        ((OR (ATOM #2#) (PROGN (LETT |p| (CAR #2#) |PSETCAT-;variables1|) NIL))
          (GO G191)))
       (SEQ
        (EXIT
         (LETT #1#
          (CONS (SPADCALL |p| (QREFELT |$| 14)) #1#) |PSETCAT-;variables1|)))
       (LETT #2# (CDR #2#) |PSETCAT-;variables1|)
       (GO G190)
       G191
       (EXIT (NREVERSE0 #1#))))
     |PSETCAT-;variables1|)
    (EXIT
     (SPADCALL
      (CONS (FUNCTION |PSETCAT-;variables1!0|) |$|)
      (SPADCALL (SPADCALL |lvars| (QREFELT |$| 18)) (QREFELT |$| 19))
      (QREFELT |$| 21)))))))

(DEFUN |PSETCAT-;variables1!0| (|#1| |#2| |$|)
 (SPADCALL |#2| |#1| (QREFELT |$| 16)))

(DEFUN |PSETCAT-;variables2| (|lp| |$|)
 (PROG (#1=#:G82397 |p| #2=#:G82398 |lvars|)
  (RETURN
   (SEQ
    (LETT |lvars|
```

```
     (PROGN
      (LETT #1# NIL |PSETCAT-;variables2|)
      (SEQ
       (LETT |p| NIL |PSETCAT-;variables2|)
       (LETT #2# |lp| |PSETCAT-;variables2|)
       G190
       (COND
        ((OR (ATOM #2#) (PROGN (LETT |p| (CAR #2#) |PSETCAT-;variables2|) NIL))
          (GO G191)))
       (SEQ
        (EXIT
         (LETT #1#
          (CONS (SPADCALL |p| (QREFELT |$| 22)) #1#) |PSETCAT-;variables2|)))
       (LETT #2# (CDR #2#) |PSETCAT-;variables2|)
       (GO G190)
       G191
       (EXIT (NREVERSE0 #1#))))
     |PSETCAT-;variables2|)
    (EXIT
     (SPADCALL
      (CONS (FUNCTION |PSETCAT-;variables2!0|) |$|)
      (SPADCALL |lvars| (QREFELT |$| 19))
      (QREFELT |$| 21)))))))

(DEFUN |PSETCAT-;variables2!0| (|#1| |#2| |$|)
 (SPADCALL |#2| |#1| (QREFELT |$| 16)))

(DEFUN |PSETCAT-;variables;SL;4| (|ps| |$|)
 (|PSETCAT-;variables1| (|PSETCAT-;elements| |ps| |$|) |$|))

(DEFUN |PSETCAT-;mainVariables;SL;5| (|ps| |$|)
 (|PSETCAT-;variables2|
  (SPADCALL (ELT |$| 24) (|PSETCAT-;elements| |ps| |$|) (QREFELT |$| 26))
  |$|))

(DEFUN |PSETCAT-;mainVariable?;VarSetSB;6| (|v| |ps| |$|)
 (PROG (|lp|)
  (RETURN
   (SEQ
    (LETT |lp|
     (SPADCALL (ELT |$| 24) (|PSETCAT-;elements| |ps| |$|) (QREFELT |$| 26))
     |PSETCAT-;mainVariable?;VarSetSB;6|)
    (SEQ
     G190
     (COND
      ((NULL
        (COND
         ((OR
           (NULL |lp|)
           (SPADCALL
```

```
            (SPADCALL (|SPADfirst| |lp|) (QREFELT |$| 22))
            |v|
            (QREFELT |$| 28)))
          (QUOTE NIL))
         ((QUOTE T) (QUOTE T))))
       (GO G191)))
     (SEQ (EXIT (LETT |lp| (CDR |lp|) |PSETCAT-;mainVariable?;VarSetSB;6|)))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT (COND ((NULL |lp|) (QUOTE NIL)) ((QUOTE T) (QUOTE T))))))))

(DEFUN |PSETCAT-;collectUnder;SVarSetS;7| (|ps| |v| |$|)
 (PROG (|p| |lp| |lq|)
  (RETURN
   (SEQ
    (LETT |lp|
     (|PSETCAT-;elements| |ps| |$|)
     |PSETCAT-;collectUnder;SVarSetS;7|)
    (LETT |lq| NIL |PSETCAT-;collectUnder;SVarSetS;7|)
    (SEQ
     G190
     (COND
      ((NULL (COND ((NULL |lp|) (QUOTE NIL)) ((QUOTE T) (QUOTE T))))
        (GO G191)))
     (SEQ
      (LETT |p| (|SPADfirst| |lp|) |PSETCAT-;collectUnder;SVarSetS;7|)
      (LETT |lp| (CDR |lp|) |PSETCAT-;collectUnder;SVarSetS;7|)
      (EXIT
       (COND
        ((OR
          (SPADCALL |p| (QREFELT |$| 24))
          (SPADCALL (SPADCALL |p| (QREFELT |$| 22)) |v| (QREFELT |$| 16)))
         (LETT |lq| (CONS |p| |lq|) |PSETCAT-;collectUnder;SVarSetS;7|)))))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT (SPADCALL |lq| (QREFELT |$| 30)))))))

(DEFUN |PSETCAT-;collectUpper;SVarSetS;8| (|ps| |v| |$|)
 (PROG (|p| |lp| |lq|)
  (RETURN
   (SEQ
    (LETT |lp|
     (|PSETCAT-;elements| |ps| |$|)
     |PSETCAT-;collectUpper;SVarSetS;8|)
    (LETT |lq| NIL |PSETCAT-;collectUpper;SVarSetS;8|)
    (SEQ
```

```
     G190
     (COND
      ((NULL (COND ((NULL |lp|) (QUOTE NIL)) ((QUOTE T) (QUOTE T))))
        (GO G191)))
     (SEQ
      (LETT |p| (|SPADfirst| |lp|) |PSETCAT-;collectUpper;SVarSetS;8|)
      (LETT |lp| (CDR |lp|) |PSETCAT-;collectUpper;SVarSetS;8|)
      (EXIT
       (COND
        ((NULL (SPADCALL |p| (QREFELT |$| 24)))
         (COND
          ((SPADCALL |v| (SPADCALL |p| (QREFELT |$| 22)) (QREFELT |$| 16))
           (LETT |lq| (CONS |p| |lq|) |PSETCAT-;collectUpper;SVarSetS;8|)))))))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT (SPADCALL |lq| (QREFELT |$| 30)))))))

(DEFUN |PSETCAT-;collect;SVarSetS;9| (|ps| |v| |$|)
 (PROG (|p| |lp| |lq|)
  (RETURN
   (SEQ
    (LETT |lp| (|PSETCAT-;elements| |ps| |$|) |PSETCAT-;collect;SVarSetS;9|)
    (LETT |lq| NIL |PSETCAT-;collect;SVarSetS;9|)
    (SEQ
     G190
     (COND
      ((NULL (COND ((NULL |lp|) (QUOTE NIL)) ((QUOTE T) (QUOTE T))))
        (GO G191)))
     (SEQ
      (LETT |p| (|SPADfirst| |lp|) |PSETCAT-;collect;SVarSetS;9|)
      (LETT |lp| (CDR |lp|) |PSETCAT-;collect;SVarSetS;9|)
      (EXIT
       (COND
        ((NULL (SPADCALL |p| (QREFELT |$| 24)))
         (COND
          ((SPADCALL (SPADCALL |p| (QREFELT |$| 22)) |v| (QREFELT |$| 28))
            (LETT |lq| (CONS |p| |lq|) |PSETCAT-;collect;SVarSetS;9|)))))))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT (SPADCALL |lq| (QREFELT |$| 30)))))))

(DEFUN |PSETCAT-;sort;SVarSetR;10| (|ps| |v| |$|)
 (PROG (|p| |lp| |us| |vs| |ws|)
  (RETURN
   (SEQ
    (LETT |lp| (|PSETCAT-;elements| |ps| |$|) |PSETCAT-;sort;SVarSetR;10|)
```

```
    (LETT |us| NIL |PSETCAT-;sort;SVarSetR;10|)
    (LETT |vs| NIL |PSETCAT-;sort;SVarSetR;10|)
    (LETT |ws| NIL |PSETCAT-;sort;SVarSetR;10|)
    (SEQ
     G190
     (COND
      ((NULL (COND ((NULL |lp|) (QUOTE NIL)) ((QUOTE T) (QUOTE T))))
       (GO G191)))
     (SEQ
      (LETT |p| (|SPADfirst| |lp|) |PSETCAT-;sort;SVarSetR;10|)
      (LETT |lp| (CDR |lp|) |PSETCAT-;sort;SVarSetR;10|)
      (EXIT
       (COND
        ((OR
           (SPADCALL |p| (QREFELT |$| 24))
           (SPADCALL (SPADCALL |p| (QREFELT |$| 22)) |v| (QREFELT |$| 16)))
          (LETT |us| (CONS |p| |us|) |PSETCAT-;sort;SVarSetR;10|))
        ((SPADCALL (SPADCALL |p| (QREFELT |$| 22)) |v| (QREFELT |$| 28))
          (LETT |vs| (CONS |p| |vs|) |PSETCAT-;sort;SVarSetR;10|))
        ((QUOTE T) (LETT |ws| (CONS |p| |ws|) |PSETCAT-;sort;SVarSetR;10|)))))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT
     (VECTOR
      (SPADCALL |us| (QREFELT |$| 30))
      (SPADCALL |vs| (QREFELT |$| 30))
      (SPADCALL |ws| (QREFELT |$| 30))))))))

(DEFUN |PSETCAT-;=;2SB;11| (|ps1| |ps2| |$|)
 (PROG (#1=#:G82439 #2=#:G82440 #3=#:G82437 |p| #4=#:G82438)
  (RETURN
   (SEQ
    (SPADCALL
     (SPADCALL
      (PROGN
       (LETT #1# NIL |PSETCAT-;=;2SB;11|)
       (SEQ
        (LETT |p| NIL |PSETCAT-;=;2SB;11|)
        (LETT #2# (|PSETCAT-;elements| |ps1| |$|) |PSETCAT-;=;2SB;11|)
        G190
        (COND
         ((OR (ATOM #2#) (PROGN (LETT |p| (CAR #2#) |PSETCAT-;=;2SB;11|) NIL))
           (GO G191)))
        (SEQ (EXIT (LETT #1# (CONS |p| #1#) |PSETCAT-;=;2SB;11|)))
        (LETT #2# (CDR #2#) |PSETCAT-;=;2SB;11|)
        (GO G190)
        G191
        (EXIT (NREVERSE0 #1#))))
```

```
      (QREFELT |$| 37))
     (SPADCALL
      (PROGN
       (LETT #3# NIL |PSETCAT-;=;2SB;11|)
       (SEQ
        (LETT |p| NIL |PSETCAT-;=;2SB;11|)
        (LETT #4# (|PSETCAT-;elements| |ps2| |$|) |PSETCAT-;=;2SB;11|)
        G190
        (COND
         ((OR (ATOM #4#) (PROGN (LETT |p| (CAR #4#) |PSETCAT-;=;2SB;11|) NIL))
           (GO G191)))
        (SEQ (EXIT (LETT #3# (CONS |p| #3#) |PSETCAT-;=;2SB;11|)))
        (LETT #4# (CDR #4#) |PSETCAT-;=;2SB;11|)
        (GO G190)
        G191
        (EXIT (NREVERSE0 #3#))))
      (QREFELT |$| 37))
     (QREFELT |$| 38))))))

(DEFUN |PSETCAT-;localInf?| (|p| |q| |$|)
 (SPADCALL
  (SPADCALL |p| (QREFELT |$| 40))
  (SPADCALL |q| (QREFELT |$| 40))
  (QREFELT |$| 41)))

(DEFUN |PSETCAT-;localTriangular?| (|lp| |$|)
 (PROG (|q| |p|)
  (RETURN
   (SEQ
    (LETT |lp|
     (SPADCALL (ELT |$| 42) |lp| (QREFELT |$| 26)) |PSETCAT-;localTriangular?|)
    (EXIT
     (COND
      ((NULL |lp|) (QUOTE T))
      ((SPADCALL (ELT |$| 24) |lp| (QREFELT |$| 43)) (QUOTE NIL))
      ((QUOTE T)
       (SEQ
        (LETT |lp|
         (SPADCALL
          (CONS (FUNCTION |PSETCAT-;localTriangular?!0|) |$|)
          |lp|
          (QREFELT |$| 45))
         |PSETCAT-;localTriangular?|)
        (LETT |p| (|SPADfirst| |lp|) |PSETCAT-;localTriangular?|)
        (LETT |lp| (CDR |lp|) |PSETCAT-;localTriangular?|)
        (SEQ
         G190
         (COND
          ((NULL
            (COND
```

```
             ((NULL |lp|) (QUOTE NIL))
             ((QUOTE T)
               (SPADCALL
                (SPADCALL
                 (LETT |q|
                  (|SPADfirst| |lp|)
                  |PSETCAT-;localTriangular?|)
                 (QREFELT |$| 22))
                (SPADCALL |p| (QREFELT |$| 22)) (QREFELT |$| 16)))))
           (GO G191)))
         (SEQ
          (LETT |p| |q| |PSETCAT-;localTriangular?|)
          (EXIT (LETT |lp| (CDR |lp|) |PSETCAT-;localTriangular?|)))
         NIL
         (GO G190)
         G191
         (EXIT NIL))
        (EXIT (NULL |lp|))))))))))

(DEFUN |PSETCAT-;localTriangular?!0| (|#1| |#2| |$|)
 (SPADCALL
  (SPADCALL |#2| (QREFELT |$| 22))
  (SPADCALL |#1| (QREFELT |$| 22))
  (QREFELT |$| 16)))

(DEFUN |PSETCAT-;triangular?;SB;14| (|ps| |$|)
 (|PSETCAT-;localTriangular?| (|PSETCAT-;elements| |ps| |$|) |$|))

(DEFUN |PSETCAT-;trivialIdeal?;SB;15| (|ps| |$|)
 (NULL
  (SPADCALL (ELT |$| 42) (|PSETCAT-;elements| |ps| |$|) (QREFELT |$| 26))))

(DEFUN |PSETCAT-;roughUnitIdeal?;SB;16| (|ps| |$|)
 (SPADCALL
  (ELT |$| 24)
  (SPADCALL (ELT |$| 42) (|PSETCAT-;elements| |ps| |$|) (QREFELT |$| 26))
  (QREFELT |$| 43)))

(DEFUN |PSETCAT-;relativelyPrimeLeadingMonomials?| (|p| |q| |$|)
 (PROG (|dp| |dq|)
  (RETURN
   (SEQ
    (LETT |dp|
     (SPADCALL |p| (QREFELT |$| 40))
     |PSETCAT-;relativelyPrimeLeadingMonomials?|)
    (LETT |dq|
     (SPADCALL |q| (QREFELT |$| 40))
     |PSETCAT-;relativelyPrimeLeadingMonomials?|)
    (EXIT
     (SPADCALL
```

```
      (SPADCALL |dp| |dq| (QREFELT |$| 49))
      (SPADCALL |dp| |dq| (QREFELT |$| 50))
      (QREFELT |$| 51)))))))

(DEFUN |PSETCAT-;roughBase?;SB;18| (|ps| |$|)
 (PROG (|p| |lp| |rB?| |copylp|)
  (RETURN
   (SEQ
    (LETT |lp|
     (SPADCALL (ELT |$| 42) (|PSETCAT-;elements| |ps| |$|) (QREFELT |$| 26))
     |PSETCAT-;roughBase?;SB;18|)
    (EXIT
     (COND
      ((NULL |lp|) (QUOTE T))
      ((QUOTE T)
       (SEQ
        (LETT |rB?| (QUOTE T) |PSETCAT-;roughBase?;SB;18|)
        (SEQ
         G190
         (COND
          ((NULL (COND ((NULL |lp|) (QUOTE NIL)) ((QUOTE T) |rB?|)))
           (GO G191)))
         (SEQ
          (LETT |p| (|SPADfirst| |lp|) |PSETCAT-;roughBase?;SB;18|)
          (LETT |lp| (CDR |lp|) |PSETCAT-;roughBase?;SB;18|)
          (LETT |copylp| |lp| |PSETCAT-;roughBase?;SB;18|)
          (EXIT
           (SEQ
            G190
            (COND
             ((NULL (COND ((NULL |copylp|) (QUOTE NIL)) ((QUOTE T) |rB?|)))
              (GO G191)))
             (SEQ
              (LETT |rB?|
               (|PSETCAT-;relativelyPrimeLeadingMonomials?| |p|
                (|SPADfirst| |copylp|) |$|)
               |PSETCAT-;roughBase?;SB;18|)
              (EXIT (LETT |copylp| (CDR |copylp|) |PSETCAT-;roughBase?;SB;18|)))
             NIL
             (GO G190)
             G191
             (EXIT NIL))))
          NIL
          (GO G190)
          G191
          (EXIT NIL))
         (EXIT |rB?|)))))))))

(DEFUN |PSETCAT-;roughSubIdeal?;2SB;19| (|ps1| |ps2| |$|)
 (PROG (|lp|)
```

```
  (RETURN
   (SEQ
    (LETT |lp|
     (SPADCALL (|PSETCAT-;elements| |ps1| |$|) |ps2| (QREFELT |$| 53))
     |PSETCAT-;roughSubIdeal?;2SB;19|)
    (EXIT (NULL (SPADCALL (ELT |$| 42) |lp| (QREFELT |$| 26))))))))

(DEFUN |PSETCAT-;roughEqualIdeals?;2SB;20| (|ps1| |ps2| |$|)
 (COND
  ((SPADCALL |ps1| |ps2| (QREFELT |$| 55)) (QUOTE T))
  ((SPADCALL |ps1| |ps2| (QREFELT |$| 56))
    (SPADCALL |ps2| |ps1| (QREFELT |$| 56)))
  ((QUOTE T) (QUOTE NIL))))

(DEFUN |PSETCAT-;exactQuo| (|r| |s| |$|)
 (SPADCALL |r| |s| (QREFELT |$| 58)))

(DEFUN |PSETCAT-;exactQuo| (|r| |s| |$|)
 (PROG (#1=#:G82473)
  (RETURN
   (PROG2
    (LETT #1# (SPADCALL |r| |s| (QREFELT |$| 60)) |PSETCAT-;exactQuo|)
    (QCDR #1#)
    (|check-union| (QEQCAR #1# 0) (QREFELT |$| 7) #1#)))))

(DEFUN |PSETCAT-;headRemainder;PSR;23| (|a| |ps| |$|)
 (PROG (|lp1| |p| |e| |g| |#G47| |#G48| |lca| |lcp| |r| |lp2|)
  (RETURN
   (SEQ
    (LETT |lp1|
     (SPADCALL (ELT |$| 42) (|PSETCAT-;elements| |ps| |$|) (QREFELT |$| 26))
     |PSETCAT-;headRemainder;PSR;23|)
    (EXIT
     (COND
      ((NULL |lp1|) (CONS |a| (|spadConstant| |$| 61)))
      ((SPADCALL (ELT |$| 24) |lp1| (QREFELT |$| 43))
        (CONS (SPADCALL |a| (QREFELT |$| 62)) (|spadConstant| |$| 61)))
      ((QUOTE T)
       (SEQ
        (LETT |r| (|spadConstant| |$| 61) |PSETCAT-;headRemainder;PSR;23|)
        (LETT |lp1|
         (SPADCALL
          (CONS (|function| |PSETCAT-;localInf?|) |$|)
          (REVERSE (|PSETCAT-;elements| |ps| |$|))
          (QREFELT |$| 45))
         |PSETCAT-;headRemainder;PSR;23|)
        (LETT |lp2| |lp1| |PSETCAT-;headRemainder;PSR;23|)
        (SEQ
         G190
         (COND
```

```
((NULL
  (COND
   ((OR (SPADCALL |a| (QREFELT |$| 42)) (NULL |lp2|)) (QUOTE NIL))
   ((QUOTE T) (QUOTE T))))
 (GO G191)))
(SEQ
 (LETT |p| (|SPADfirst| |lp2|) |PSETCAT-;headRemainder;PSR;23|)
 (LETT |e|
  (SPADCALL
   (SPADCALL |a| (QREFELT |$| 40))
   (SPADCALL |p| (QREFELT |$| 40))
   (QREFELT |$| 63))
  |PSETCAT-;headRemainder;PSR;23|)
 (EXIT
  (COND
   ((QEQCAR |e| 0)
    (SEQ
     (LETT |g|
      (SPADCALL
       (LETT |lca|
        (SPADCALL |a| (QREFELT |$| 64))
        |PSETCAT-;headRemainder;PSR;23|)
       (LETT |lcp|
        (SPADCALL |p| (QREFELT |$| 64))
        |PSETCAT-;headRemainder;PSR;23|)
       (QREFELT |$| 65))
      |PSETCAT-;headRemainder;PSR;23|)
     (PROGN
      (LETT |#G47|
       (|PSETCAT-;exactQuo| |lca| |g| |$|)
       |PSETCAT-;headRemainder;PSR;23|)
      (LETT |#G48|
       (|PSETCAT-;exactQuo| |lcp| |g| |$|)
       |PSETCAT-;headRemainder;PSR;23|)
      (LETT |lca| |#G47| |PSETCAT-;headRemainder;PSR;23|)
      (LETT |lcp| |#G48| |PSETCAT-;headRemainder;PSR;23|))
     (LETT |a|
      (SPADCALL
       (SPADCALL |lcp|
        (SPADCALL |a| (QREFELT |$| 62))
        (QREFELT |$| 66))
       (SPADCALL
        (SPADCALL |lca| (QCDR |e|) (QREFELT |$| 67))
        (SPADCALL |p| (QREFELT |$| 62)) (QREFELT |$| 68))
       (QREFELT |$| 69))
      |PSETCAT-;headRemainder;PSR;23|)
     (LETT |r|
      (SPADCALL |r| |lcp| (QREFELT |$| 70))
      |PSETCAT-;headRemainder;PSR;23|)
     (EXIT (LETT |lp2| |lp1| |PSETCAT-;headRemainder;PSR;23|))))
```

```
            ((QUOTE T)
             (LETT |lp2| (CDR |lp2|) |PSETCAT-;headRemainder;PSR;23|)))))
         NIL
         (GO G190)
         G191
         (EXIT NIL))
        (EXIT (CONS |a| |r|))))))))))

(DEFUN |PSETCAT-;makeIrreducible!| (|frac| |$|)
 (PROG (|g|)
  (RETURN
   (SEQ
    (LETT |g|
     (SPADCALL (QCDR |frac|) (QCAR |frac|) (QREFELT |$| 73))
     |PSETCAT-;makeIrreducible!|)
    (EXIT
     (COND
      ((SPADCALL |g| (QREFELT |$| 74)) |frac|)
      ((QUOTE T)
       (SEQ
        (PROGN
         (RPLACA |frac| (SPADCALL (QCAR |frac|) |g| (QREFELT |$| 75)))
         (QCAR |frac|))
        (PROGN
         (RPLACD |frac| (|PSETCAT-;exactQuo| (QCDR |frac|) |g| |$|))
         (QCDR |frac|))
        (EXIT |frac|)))))))))

(DEFUN |PSETCAT-;remainder;PSR;25| (|a| |ps| |$|)
 (PROG (|hRa| |r| |lca| |g| |b| |c|)
  (RETURN
   (SEQ
    (LETT |hRa|
     (|PSETCAT-;makeIrreducible!| (SPADCALL |a| |ps| (QREFELT |$| 76)) |$|)
     |PSETCAT-;remainder;PSR;25|)
    (LETT |a| (QCAR |hRa|) |PSETCAT-;remainder;PSR;25|)
    (LETT |r| (QCDR |hRa|) |PSETCAT-;remainder;PSR;25|)
    (EXIT
     (COND
      ((SPADCALL |a| (QREFELT |$| 42))
        (VECTOR (|spadConstant| |$| 61) |a| |r|))
      ((QUOTE T)
       (SEQ
        (LETT |b|
         (SPADCALL
          (|spadConstant| |$| 61)
          (SPADCALL |a| (QREFELT |$| 40))
          (QREFELT |$| 67))
         |PSETCAT-;remainder;PSR;25|)
        (LETT |c| (SPADCALL |a| (QREFELT |$| 64)) |PSETCAT-;remainder;PSR;25|)
```

```
(SEQ
 G190
 (COND
  ((NULL
    (COND
     ((SPADCALL
        (LETT |a|
         (SPADCALL |a| (QREFELT |$| 62))
         |PSETCAT-;remainder;PSR;25|)
        (QREFELT |$| 42))
       (QUOTE NIL))
     ((QUOTE T) (QUOTE T))))
   (GO G191)))
 (SEQ
  (LETT |hRa|
   (|PSETCAT-;makeIrreducible!|
    (SPADCALL |a| |ps| (QREFELT |$| 76))
    |$|)
   |PSETCAT-;remainder;PSR;25|)
  (LETT |a| (QCAR |hRa|) |PSETCAT-;remainder;PSR;25|)
  (LETT |r|
   (SPADCALL |r| (QCDR |hRa|) (QREFELT |$| 70))
   |PSETCAT-;remainder;PSR;25|)
  (LETT |g|
   (SPADCALL |c|
    (LETT |lca|
     (SPADCALL |a| (QREFELT |$| 64))
     |PSETCAT-;remainder;PSR;25|)
    (QREFELT |$| 65))
   |PSETCAT-;remainder;PSR;25|)
  (LETT |b|
   (SPADCALL
    (SPADCALL
     (SPADCALL
      (QCDR |hRa|)
      (|PSETCAT-;exactQuo| |c| |g| |$|)
      (QREFELT |$| 70))
     |b|
     (QREFELT |$| 66))
    (SPADCALL
     (|PSETCAT-;exactQuo| |lca| |g| |$|)
     (SPADCALL |a| (QREFELT |$| 40))
     (QREFELT |$| 67))
    (QREFELT |$| 77))
   |PSETCAT-;remainder;PSR;25|)
  (EXIT (LETT |c| |g| |PSETCAT-;remainder;PSR;25|)))
 NIL
 (GO G190)
 G191
 (EXIT NIL))
```

```
        (EXIT (VECTOR |c| |b| |r|))))))))))

(DEFUN |PSETCAT-;rewriteIdealWithHeadRemainder;LSL;26| (|ps| |cs| |$|)
 (PROG (|p| |rs|)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |cs| (QREFELT |$| 80)) |ps|)
     ((SPADCALL |cs| (QREFELT |$| 81)) (LIST (|spadConstant| |$| 82)))
     ((QUOTE T)
      (SEQ
       (LETT |ps|
        (SPADCALL (ELT |$| 42) |ps| (QREFELT |$| 26))
        |PSETCAT-;rewriteIdealWithHeadRemainder;LSL;26|)
       (EXIT
        (COND
         ((NULL |ps|) |ps|)
         ((SPADCALL (ELT |$| 24) |ps| (QREFELT |$| 43))
           (LIST (|spadConstant| |$| 83)))
         ((QUOTE T)
           (SEQ
            (LETT |rs| NIL |PSETCAT-;rewriteIdealWithHeadRemainder;LSL;26|)
            (SEQ
             G190
             (COND
              ((NULL (COND ((NULL |ps|) (QUOTE NIL)) ((QUOTE T) (QUOTE T))))
                (GO G191)))
             (SEQ
              (LETT |p|
               (|SPADfirst| |ps|)
               |PSETCAT-;rewriteIdealWithHeadRemainder;LSL;26|)
              (LETT |ps|
               (CDR |ps|)
               |PSETCAT-;rewriteIdealWithHeadRemainder;LSL;26|)
              (LETT |p|
               (QCAR (SPADCALL |p| |cs| (QREFELT |$| 76)))
               |PSETCAT-;rewriteIdealWithHeadRemainder;LSL;26|)
              (EXIT
               (COND
                ((NULL (SPADCALL |p| (QREFELT |$| 42)))
                 (COND
                  ((SPADCALL |p| (QREFELT |$| 24))
                   (SEQ
                    (LETT |ps| NIL
                     |PSETCAT-;rewriteIdealWithHeadRemainder;LSL;26|)
                    (EXIT
                     (LETT |rs|
                      (LIST (|spadConstant| |$| 83))
                      |PSETCAT-;rewriteIdealWithHeadRemainder;LSL;26|))))
                  ((QUOTE T)
```

```
                (SEQ
                 (SPADCALL |p| (QREFELT |$| 84))
                 (EXIT
                  (LETT |rs|
                   (CONS |p| |rs|)
                   |PSETCAT-;rewriteIdealWithHeadRemainder;LSL;26|)))))))))
            NIL
            (GO G190)
            G191
            (EXIT NIL))
           (EXIT (SPADCALL |rs| (QREFELT |$| 85)))))))))))))))

(DEFUN |PSETCAT-;rewriteIdealWithRemainder;LSL;27| (|ps| |cs| |$|)
 (PROG (|p| |rs|)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |cs| (QREFELT |$| 80)) |ps|)
     ((SPADCALL |cs| (QREFELT |$| 81)) (LIST (|spadConstant| |$| 82)))
     ((QUOTE T)
      (SEQ
       (LETT |ps|
        (SPADCALL (ELT |$| 42) |ps| (QREFELT |$| 26))
        |PSETCAT-;rewriteIdealWithRemainder;LSL;27|)
       (EXIT
        (COND
         ((NULL |ps|) |ps|)
         ((SPADCALL (ELT |$| 24) |ps| (QREFELT |$| 43))
           (LIST (|spadConstant| |$| 83)))
         ((QUOTE T)
          (SEQ
           (LETT |rs| NIL |PSETCAT-;rewriteIdealWithRemainder;LSL;27|)
           (SEQ
            G190
            (COND
             ((NULL (COND ((NULL |ps|) (QUOTE NIL)) ((QUOTE T) (QUOTE T))))
               (GO G191)))
            (SEQ
             (LETT |p|
              (|SPADfirst| |ps|)
              |PSETCAT-;rewriteIdealWithRemainder;LSL;27|)
             (LETT |ps| (CDR |ps|) |PSETCAT-;rewriteIdealWithRemainder;LSL;27|)
             (LETT |p|
              (QVELT (SPADCALL |p| |cs| (QREFELT |$| 87)) 1)
              |PSETCAT-;rewriteIdealWithRemainder;LSL;27|)
             (EXIT
              (COND
               ((NULL (SPADCALL |p| (QREFELT |$| 42)))
                (COND
                 ((SPADCALL |p| (QREFELT |$| 24))
```

```
                (SEQ
                 (LETT |ps| NIL |PSETCAT-;rewriteIdealWithRemainder;LSL;27|)
                 (EXIT
                  (LETT |rs|
                   (LIST (|spadConstant| |$| 83))
                   |PSETCAT-;rewriteIdealWithRemainder;LSL;27|))))
               ((QUOTE T)
                (LETT |rs|
                 (CONS (SPADCALL |p| (QREFELT |$| 88)) |rs|)
                 |PSETCAT-;rewriteIdealWithRemainder;LSL;27|)))))))
           NIL
           (GO G190)
           G191
           (EXIT NIL))
          (EXIT (SPADCALL |rs| (QREFELT |$| 85)))))))))))))

(DEFUN |PolynomialSetCategory&| (|#1| |#2| |#3| |#4| |#5|)
 (PROG (|DV$1| |DV$2| |DV$3| |DV$4| |DV$5| |dv$| |$| |pv$|)
  (RETURN
   (PROGN
    (LETT |DV$1| (|devaluate| |#1|) . #1=(|PolynomialSetCategory&|))
    (LETT |DV$2| (|devaluate| |#2|) . #1#)
    (LETT |DV$3| (|devaluate| |#3|) . #1#)
    (LETT |DV$4| (|devaluate| |#4|) . #1#)
    (LETT |DV$5| (|devaluate| |#5|) . #1#)
    (LETT |dv$|
     (LIST
      (QUOTE |PolynomialSetCategory&|)
      |DV$1| |DV$2| |DV$3| |DV$4| |DV$5|) . #1#)
    (LETT |$| (make-array 90) . #1#)
    (QSETREFV |$| 0 |dv$|)
    (QSETREFV |$| 3
     (LETT |pv$|
      (|buildPredVector| 0 0
       (LIST (|HasCategory| |#2| (QUOTE (|IntegralDomain|))))) . #1#))
    (|stuffDomainSlots| |$|)
    (QSETREFV |$| 6 |#1|)
    (QSETREFV |$| 7 |#2|)
    (QSETREFV |$| 8 |#3|)
    (QSETREFV |$| 9 |#4|)
    (QSETREFV |$| 10 |#5|)
    (COND
     ((|testBitVector| |pv$| 1)
      (PROGN
       (QSETREFV |$| 48
        (CONS (|dispatchFunction| |PSETCAT-;roughUnitIdeal?;SB;16|) |$|))
       (QSETREFV |$| 52
        (CONS (|dispatchFunction| |PSETCAT-;roughBase?;SB;18|) |$|))
       (QSETREFV |$| 54
        (CONS (|dispatchFunction| |PSETCAT-;roughSubIdeal?;2SB;19|) |$|))
```

```
       (QSETREFV |$| 57
        (CONS (|dispatchFunction| |PSETCAT-;roughEqualIdeals?;2SB;20|) |$|))))))
    (COND
     ((|HasCategory| |#2| (QUOTE (|GcdDomain|)))
     (COND
      ((|HasCategory| |#4| (QUOTE (|ConvertibleTo| (|Symbol|))))
       (PROGN
        (QSETREFV |$| 72
         (CONS (|dispatchFunction| |PSETCAT-;headRemainder;PSR;23|) |$|))
        (QSETREFV |$| 79
         (CONS (|dispatchFunction| |PSETCAT-;remainder;PSR;25|) |$|))
        (QSETREFV |$| 86
         (CONS
          (|dispatchFunction| |PSETCAT-;rewriteIdealWithHeadRemainder;LSL;26|)
          |$|))
        (QSETREFV |$| 89
         (CONS
          (|dispatchFunction| |PSETCAT-;rewriteIdealWithRemainder;LSL;27|)
          |$|)))))))
    |$|))))

(setf (get
 (QUOTE |PolynomialSetCategory&|)
 (QUOTE |infovec|))
 (LIST
  (QUOTE
   #(NIL NIL NIL NIL NIL NIL (|local| |#1|) (|local| |#2|) (|local| |#3|)
     (|local| |#4|) (|local| |#5|) (|List| 10) (0 . |members|) (|List| 9)
     (5 . |variables|) (|Boolean|) (10 . |<|) (|List| |$|) (16 . |concat|)
     (21 . |removeDuplicates|) (|Mapping| 15 9 9) (26 . |sort|)
     (32 . |mvar|) |PSETCAT-;variables;SL;4| (37 . |ground?|)
     (|Mapping| 15 10) (42 . |remove|) |PSETCAT-;mainVariables;SL;5|
     (48 . |=|) |PSETCAT-;mainVariable?;VarSetSB;6| (54 . |construct|)
     |PSETCAT-;collectUnder;SVarSetS;7| |PSETCAT-;collectUpper;SVarSetS;8|
     |PSETCAT-;collect;SVarSetS;9| (|Record| (|:| |under| |$|)
     (|:| |floor| |$|) (|:| |upper| |$|)) |PSETCAT-;sort;SVarSetR;10|
     (|Set| 10) (59 . |brace|) (64 . |=|) |PSETCAT-;=;2SB;11|
     (70 . |degree|) (75 . |<|) (81 . |zero?|) (86 . |any?|)
     (|Mapping| 15 10 10) (92 . |sort|) |PSETCAT-;triangular?;SB;14|
     |PSETCAT-;trivialIdeal?;SB;15| (98 . |roughUnitIdeal?|)
     (103 . |sup|) (109 . |+|) (115 . |=|) (121 . |roughBase?|)
     (126 . |rewriteIdealWithRemainder|) (132 . |roughSubIdeal?|)
     (138 . |=|) (144 . |roughSubIdeal?|) (150 . |roughEqualIdeals?|)
     (156 . |quo|) (|Union| |$| (QUOTE "failed")) (162 . |exquo|)
     (168 . |One|) (172 . |reductum|) (177 . |subtractIfCan|)
     (183 . |leadingCoefficient|) (188 . |gcd|) (194 . |*|)
     (200 . |monomial|) (206 . |*|) (212 . |-|) (218 . |*|)
     (|Record| (|:| |num| 10) (|:| |den| 7)) (224 . |headRemainder|)
     (230 . |gcd|) (236 . |one?|) (241 . |exactQuotient!|)
     (247 . |headRemainder|) (253 . |+|) (|Record| (|:| |rnum| 7)
```

```
    (|:| |polnum| 10) (|:| |den| 7)) (259 . |remainder|)
    (265 . |trivialIdeal?|) (270 . |roughUnitIdeal?|)
    (275 . |Zero|) (279 . |One|) (283 . |primitivePart!|)
    (288 . |removeDuplicates|) (293 . |rewriteIdealWithHeadRemainder|)
    (299 . |remainder|) (305 . |unitCanonical|)
    (310 . |rewriteIdealWithRemainder|)))
 (QUOTE #(|variables| 316 |trivialIdeal?| 321 |triangular?| 326
    |sort| 331 |roughUnitIdeal?| 337 |roughSubIdeal?| 342
    |roughEqualIdeals?| 348 |roughBase?| 354 |rewriteIdealWithRemainder|
    359 |rewriteIdealWithHeadRemainder| 365 |remainder| 371 |mainVariables|
    377 |mainVariable?| 382 |headRemainder| 388 |collectUpper| 394
    |collectUnder| 400 |collect| 406 |=| 412))
 (QUOTE NIL)
 (CONS
  (|makeByteWordVec2| 1 (QUOTE NIL))
  (CONS
   (QUOTE #())
   (CONS
    (QUOTE #())
    (|makeByteWordVec2| 89 (QUOTE (1 6 11 0 12 1 10 13 0 14 2 9 15 0
     0 16 1 13 0 17 18 1 13 0 0 19 2 13 0 20 0 21 1 10 9 0 22 1 10 15 0 24
     2 11 0 25 0 26 2 9 15 0 0 28 1 6 0 11 30 1 36 0 11 37 2 36 15 0 0 38 1
     10 8 0 40 2 8 15 0 0 41 1 10 15 0 42 2 11 15 25 0 43 2 11 0 44 0 45 1
     0 15 0 48 2 8 0 0 0 49 2 8 0 0 0 50 2 8 15 0 0 51 1 0 15 0 52 2 6 11 11
     0 53 2 0 15 0 0 54 2 6 15 0 0 55 2 6 15 0 0 56 2 0 15 0 0 57 2 7 0 0 0
     58 2 7 59 0 0 60 0 7 0 61 1 10 0 0 62 2 8 59 0 0 63 1 10 7 0 64 2 7 0
     0 0 65 2 10 0 7 0 66 2 10 0 7 8 67 2 10 0 0 0 68 2 10 0 0 0 69 2 7 0
     0 0 70 2 0 71 10 0 72 2 10 7 7 0 73 1 7 15 0 74 2 10 0 0 7 75 2 6 71
     10 0 76 2 10 0 0 0 77 2 0 78 10 0 79 1 6 15 0 80 1 6 15 0 81 0 10 0
     82 0 10 0 83 1 10 0 0 84 1 11 0 0 85 2 0 11 11 0 86 2 6 78 10 0 87 1
     10 0 0 88 2 0 11 11 0 89 1 0 13 0 23 1 0 15 0 47 1 0 15 0 46 2 0 34
     0 9 35 1 0 15 0 48 2 0 15 0 0 54 2 0 15 0 0 57 1 0 15 0 52 2 0 11 11
     0 89 2 0 11 11 0 86 2 0 78 10 0 79 1 0 13 0 27 2 0 15 9 0 29 2 0 71
     10 0 72 2 0 0 0 9 32 2 0 0 0 9 31 2 0 0 0 9 33 2 0 15 0 0 39))))))
 (QUOTE |lookupComplete|)))
```

———-

21.47 QFCAT.lsp BOOTSTRAP

QFCAT depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **QFCAT** category which we can write into the **MID** directory. We compile the lisp code and copy the **QFCAT.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— QFCAT.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |QuotientFieldCategory;CAT| (QUOTE NIL))

(SETQ |QuotientFieldCategory;AL| (QUOTE NIL))

(DEFUN |QuotientFieldCategory| (#1=#:G103631)
 (LET (#2=#:G103632)
  (COND
   ((SETQ #2# (|assoc| (|devaluate| #1#) |QuotientFieldCategory;AL|))
     (CDR #2#))
   (T
    (SETQ |QuotientFieldCategory;AL|
     (|cons5|
      (CONS (|devaluate| #1#) (SETQ #2# (|QuotientFieldCategory;| #1#)))
      |QuotientFieldCategory;AL|))
     #2#))))

(DEFUN |QuotientFieldCategory;| (|t#1|)
 (PROG (#1=#:G103630)
  (RETURN
   (PROG1
    (LETT #1#
     (|sublisV|
      (PAIR (QUOTE (|t#1|)) (LIST (|devaluate| |t#1|)))
      (COND
       (|QuotientFieldCategory;CAT|)
       ((QUOTE T)
        (LETT |QuotientFieldCategory;CAT|
         (|Join|
          (|Field|)
          (|Algebra| (QUOTE |t#1|))
          (|RetractableTo| (QUOTE |t#1|))
          (|FullyEvalableOver| (QUOTE |t#1|))
          (|DifferentialExtension| (QUOTE |t#1|))
          (|FullyLinearlyExplicitRingOver| (QUOTE |t#1|))
          (|Patternable| (QUOTE |t#1|))
          (|FullyPatternMatchable| (QUOTE |t#1|))
          (|mkCategory|
           (QUOTE |domain|)
           (QUOTE (
            ((|/| (|$| |t#1| |t#1|)) T)
            ((|numer| (|t#1| |$|)) T)
            ((|denom| (|t#1| |$|)) T)
            ((|numerator| (|$| |$|)) T)
            ((|denominator| (|$| |$|)) T)
            ((|wholePart| (|t#1| |$|)) (|has| |t#1| (|EuclideanDomain|)))
```

```
          ((|fractionPart| (|$| |$|)) (|has| |t#1| (|EuclideanDomain|)))
          ((|random| (|$|)) (|has| |t#1| (|IntegerNumberSystem|)))
          ((|ceiling| (|t#1| |$|)) (|has| |t#1| (|IntegerNumberSystem|)))
          ((|floor| (|t#1| |$|)) (|has| |t#1| (|IntegerNumberSystem|)))))
        (QUOTE (
          ((|StepThrough|) (|has| |t#1| (|StepThrough|)))
          ((|RetractableTo| (|Integer|))
             (|has| |t#1| (|RetractableTo| (|Integer|))))
          ((|RetractableTo| (|Fraction| (|Integer|)))
             (|has| |t#1| (|RetractableTo| (|Integer|))))
          ((|OrderedSet|) (|has| |t#1| (|OrderedSet|)))
          ((|OrderedIntegralDomain|) (|has| |t#1| (|OrderedIntegralDomain|)))
          ((|RealConstant|) (|has| |t#1| (|RealConstant|)))
          ((|ConvertibleTo| (|InputForm|))
             (|has| |t#1| (|ConvertibleTo| (|InputForm|))))
          ((|CharacteristicZero|) (|has| |t#1| (|CharacteristicZero|)))
          ((|CharacteristicNonZero|) (|has| |t#1| (|CharacteristicNonZero|)))
          ((|RetractableTo| (|Symbol|))
             (|has| |t#1| (|RetractableTo| (|Symbol|))))
          ((|PolynomialFactorizationExplicit|)
             (|has| |t#1| (|PolynomialFactorizationExplicit|)))))
        (QUOTE NIL) NIL)) . #2=(|QuotientFieldCategory|))))) . #2#)
  (SETELT #1# 0
    (LIST (QUOTE |QuotientFieldCategory|) (|devaluate| |t#1|)))))))
```

———-

21.48 QFCAT-.lsp BOOTSTRAP

QFCAT- depends on **QFCAT**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **QFCAT-** category which we can write into the **MID** directory. We compile the lisp code and copy the **QFCAT-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— QFCAT-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |QFCAT-;numerator;2A;1| (|x| |$|)
 (SPADCALL (SPADCALL |x| (QREFELT |$| 8)) (QREFELT |$| 9)))

(DEFUN |QFCAT-;denominator;2A;2| (|x| |$|)
 (SPADCALL (SPADCALL |x| (QREFELT |$| 11)) (QREFELT |$| 9)))

(DEFUN |QFCAT-;init;A;3| (|$|)
```

```
 (SPADCALL (|spadConstant| |$| 13) (|spadConstant| |$| 14) (QREFELT |$| 15)))

(DEFUN |QFCAT-;nextItem;AU;4| (|n| |$|)
 (PROG (|m|)
  (RETURN
   (SEQ
    (LETT |m|
     (SPADCALL
      (SPADCALL |n| (QREFELT |$| 8))
      (QREFELT |$| 18))
     |QFCAT-;nextItem;AU;4|)
    (EXIT
     (COND
      ((QEQCAR |m| 1)
       (|error| "We seem to have a Fraction of a finite object"))
      ((QUOTE T)
       (CONS 0
        (SPADCALL (QCDR |m|) (|spadConstant| |$| 14) (QREFELT |$| 15))))))))))

(DEFUN |QFCAT-;map;M2A;5| (|fn| |x| |$|)
 (SPADCALL
  (SPADCALL (SPADCALL |x| (QREFELT |$| 8)) |fn|)
  (SPADCALL (SPADCALL |x| (QREFELT |$| 11)) |fn|)
  (QREFELT |$| 15)))

(DEFUN |QFCAT-;reducedSystem;MM;6| (|m| |$|)
 (SPADCALL |m| (QREFELT |$| 26)))

(DEFUN |QFCAT-;characteristic;Nni;7| (|$|)
 (SPADCALL (QREFELT |$| 30)))

(DEFUN |QFCAT-;differentiate;AMA;8| (|x| |deriv| |$|)
 (PROG (|n| |d|)
  (RETURN
   (SEQ
    (LETT |n| (SPADCALL |x| (QREFELT |$| 8)) |QFCAT-;differentiate;AMA;8|)
    (LETT |d| (SPADCALL |x| (QREFELT |$| 11)) |QFCAT-;differentiate;AMA;8|)
    (EXIT
     (SPADCALL
      (SPADCALL
       (SPADCALL (SPADCALL |n| |deriv|) |d| (QREFELT |$| 32))
       (SPADCALL |n| (SPADCALL |d| |deriv|) (QREFELT |$| 32))
       (QREFELT |$| 33))
      (SPADCALL |d| 2 (QREFELT |$| 35)) (QREFELT |$| 15)))))))

(DEFUN |QFCAT-;convert;AIf;9| (|x| |$|)
 (SPADCALL
  (SPADCALL (SPADCALL |x| (QREFELT |$| 8)) (QREFELT |$| 38))
  (SPADCALL (SPADCALL |x| (QREFELT |$| 11)) (QREFELT |$| 38))
  (QREFELT |$| 39)))
```

```
(DEFUN |QFCAT-;convert;AF;10| (|x| |$|)
 (SPADCALL
  (SPADCALL (SPADCALL |x| (QREFELT |$| 8)) (QREFELT |$| 42))
  (SPADCALL (SPADCALL |x| (QREFELT |$| 11)) (QREFELT |$| 42))
  (QREFELT |$| 43)))

(DEFUN |QFCAT-;convert;ADf;11| (|x| |$|)
 (|/|
  (SPADCALL (SPADCALL |x| (QREFELT |$| 8)) (QREFELT |$| 46))
  (SPADCALL (SPADCALL |x| (QREFELT |$| 11)) (QREFELT |$| 46))))

(DEFUN |QFCAT-;<;2AB;12| (|x| |y| |$|)
 (SPADCALL
  (SPADCALL
   (SPADCALL |x| (QREFELT |$| 8))
   (SPADCALL |y| (QREFELT |$| 11))
   (QREFELT |$| 32))
  (SPADCALL
   (SPADCALL |y| (QREFELT |$| 8))
   (SPADCALL |x| (QREFELT |$| 11))
   (QREFELT |$| 32))
  (QREFELT |$| 49)))

(DEFUN |QFCAT-;<;2AB;13| (|x| |y| |$|)
 (PROG (|#G19| |#G20| |#G21| |#G22|)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL
       (SPADCALL |x| (QREFELT |$| 11))
       (|spadConstant| |$| 51)
       (QREFELT |$| 49))
      (PROGN
       (LETT |#G19| |y| |QFCAT-;<;2AB;13|)
       (LETT |#G20| |x| |QFCAT-;<;2AB;13|)
       (LETT |x| |#G19| |QFCAT-;<;2AB;13|)
       (LETT |y| |#G20| |QFCAT-;<;2AB;13|))))
    (COND
     ((SPADCALL
        (SPADCALL |y| (QREFELT |$| 11))
        (|spadConstant| |$| 51)
        (QREFELT |$| 49))
       (PROGN
        (LETT |#G21| |y| |QFCAT-;<;2AB;13|)
        (LETT |#G22| |x| |QFCAT-;<;2AB;13|)
        (LETT |x| |#G21| |QFCAT-;<;2AB;13|)
        (LETT |y| |#G22| |QFCAT-;<;2AB;13|))))
    (EXIT
     (SPADCALL
```

```
      (SPADCALL
       (SPADCALL |x| (QREFELT |$| 8))
       (SPADCALL |y| (QREFELT |$| 11))
       (QREFELT |$| 32))
      (SPADCALL
       (SPADCALL |y| (QREFELT |$| 8))
       (SPADCALL |x| (QREFELT |$| 11))
       (QREFELT |$| 32))
      (QREFELT |$| 49)))))))

(DEFUN |QFCAT-;<;2AB;14| (|x| |y| |$|)
 (SPADCALL
  (SPADCALL
   (SPADCALL |x| (QREFELT |$| 8))
   (SPADCALL |y| (QREFELT |$| 11))
   (QREFELT |$| 32))
  (SPADCALL
   (SPADCALL |y| (QREFELT |$| 8))
   (SPADCALL |x| (QREFELT |$| 11))
   (QREFELT |$| 32))
  (QREFELT |$| 49)))

(DEFUN |QFCAT-;fractionPart;2A;15| (|x| |$|)
 (SPADCALL |x|
  (SPADCALL (SPADCALL |x| (QREFELT |$| 52)) (QREFELT |$| 9))
  (QREFELT |$| 53)))

(DEFUN |QFCAT-;coerce;SA;16| (|s| |$|)
 (SPADCALL (SPADCALL |s| (QREFELT |$| 56)) (QREFELT |$| 9)))

(DEFUN |QFCAT-;retract;AS;17| (|x| |$|)
 (SPADCALL (SPADCALL |x| (QREFELT |$| 58)) (QREFELT |$| 59)))

(DEFUN |QFCAT-;retractIfCan;AU;18| (|x| |$|)
 (PROG (|r|)
  (RETURN
   (SEQ
    (LETT |r| (SPADCALL |x| (QREFELT |$| 62)) |QFCAT-;retractIfCan;AU;18|)
    (EXIT
     (COND
      ((QEQCAR |r| 1) (CONS 1 "failed"))
      ((QUOTE T) (SPADCALL (QCDR |r|) (QREFELT |$| 64)))))))))

(DEFUN |QFCAT-;convert;AP;19| (|x| |$|)
 (SPADCALL
  (SPADCALL (SPADCALL |x| (QREFELT |$| 8)) (QREFELT |$| 67))
  (SPADCALL (SPADCALL |x| (QREFELT |$| 11)) (QREFELT |$| 67))
  (QREFELT |$| 68)))

(DEFUN |QFCAT-;patternMatch;AP2Pmr;20| (|x| |p| |l| |$|)
```

```
 (SPADCALL |x| |p| |l| (QREFELT |$| 72)))

(DEFUN |QFCAT-;convert;AP;21| (|x| |$|)
 (SPADCALL
  (SPADCALL (SPADCALL |x| (QREFELT |$| 8)) (QREFELT |$| 76))
  (SPADCALL (SPADCALL |x| (QREFELT |$| 11)) (QREFELT |$| 76))
  (QREFELT |$| 77)))

(DEFUN |QFCAT-;patternMatch;AP2Pmr;22| (|x| |p| |l| |$|)
 (SPADCALL |x| |p| |l| (QREFELT |$| 81)))

(DEFUN |QFCAT-;coerce;FA;23| (|x| |$|)
 (SPADCALL
  (SPADCALL (SPADCALL |x| (QREFELT |$| 86)) (QREFELT |$| 87))
  (SPADCALL (SPADCALL |x| (QREFELT |$| 88)) (QREFELT |$| 87))
  (QREFELT |$| 89)))

(DEFUN |QFCAT-;retract;AI;24| (|x| |$|)
 (SPADCALL (SPADCALL |x| (QREFELT |$| 58)) (QREFELT |$| 91)))

(DEFUN |QFCAT-;retractIfCan;AU;25| (|x| |$|)
 (PROG (|u|)
  (RETURN
   (SEQ
    (LETT |u| (SPADCALL |x| (QREFELT |$| 62)) |QFCAT-;retractIfCan;AU;25|)
    (EXIT
     (COND
      ((QEQCAR |u| 1) (CONS 1 "failed"))
      ((QUOTE T) (SPADCALL (QCDR |u|) (QREFELT |$| 94)))))))))

(DEFUN |QFCAT-;random;A;26| (|$|)
 (PROG (|d|)
  (RETURN
   (SEQ
    (SEQ
     G190
     (COND
       ((NULL
         (SPADCALL
          (LETT |d| (SPADCALL (QREFELT |$| 96)) |QFCAT-;random;A;26|)
          (QREFELT |$| 97)))
        (GO G191)))
     (SEQ (EXIT |d|))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT (SPADCALL (SPADCALL (QREFELT |$| 96)) |d| (QREFELT |$| 15)))))))

(DEFUN |QFCAT-;reducedSystem;MVR;27| (|m| |v| |$|)
```

```
 (PROG (|n|)
  (RETURN
   (SEQ
    (LETT |n|
     (SPADCALL
      (SPADCALL (SPADCALL |v| (QREFELT |$| 100)) |m| (QREFELT |$| 101))
      (QREFELT |$| 102))
     |QFCAT-;reducedSystem;MVR;27|)
    (EXIT
     (CONS
      (SPADCALL |n|
       (SPADCALL |n| (QREFELT |$| 103))
       (SPADCALL |n| (QREFELT |$| 104))
       (|+| 1 (SPADCALL |n| (QREFELT |$| 105)))
       (SPADCALL |n| (QREFELT |$| 106))
       (QREFELT |$| 107))
      (SPADCALL |n| (SPADCALL |n| (QREFELT |$| 105)) (QREFELT |$| 109))))))))

(DEFUN |QuotientFieldCategory&| (|#1| |#2|)
 (PROG (|DV$1| |DV$2| |dv$| |$| |pv$|)
  (RETURN
   (PROGN
    (LETT |DV$1| (|devaluate| |#1|) . #1=(|QuotientFieldCategory&|))
    (LETT |DV$2| (|devaluate| |#2|) . #1#)
    (LETT |dv$| (LIST (QUOTE |QuotientFieldCategory&|) |DV$1| |DV$2|) . #1#)
    (LETT |$| (make-array 119) . #1#)
    (QSETREFV |$| 0 |dv$|)
    (QSETREFV |$| 3
     (LETT |pv$|
      (|buildPredVector| 0 0 (LIST
       (|HasCategory| |#2| (QUOTE (|PolynomialFactorizationExplicit|)))
       (|HasCategory| |#2| (QUOTE (|IntegerNumberSystem|)))
       (|HasCategory| |#2| (QUOTE (|EuclideanDomain|)))
       (|HasCategory| |#2| (QUOTE (|RetractableTo| (|Symbol|))))
       (|HasCategory| |#2| (QUOTE (|CharacteristicNonZero|)))
       (|HasCategory| |#2| (QUOTE (|CharacteristicZero|)))
       (|HasCategory| |#2| (QUOTE (|ConvertibleTo| (|InputForm|))))
       (|HasCategory| |#2| (QUOTE (|RealConstant|)))
       (|HasCategory| |#2| (QUOTE (|OrderedIntegralDomain|)))
       (|HasCategory| |#2| (QUOTE (|OrderedSet|)))
       (|HasCategory| |#2| (QUOTE (|RetractableTo| (|Integer|))))
       (|HasCategory| |#2| (QUOTE (|StepThrough|)))))) . #1#))
    (|stuffDomainSlots| |$|)
    (QSETREFV |$| 6 |#1|)
    (QSETREFV |$| 7 |#2|)
    (COND
     ((|testBitVector| |pv$| 12)
       (PROGN
        (QSETREFV |$| 16 (CONS (|dispatchFunction| |QFCAT-;init;A;3|) |$|))
        (QSETREFV |$| 20
```

```
     (CONS (|dispatchFunction| |QFCAT-;nextItem;AU;4|) |$|)))))
(COND
 ((|testBitVector| |pv$| 7)
   (QSETREFV |$| 40
    (CONS (|dispatchFunction| |QFCAT-;convert;AIf;9|) |$|))))
(COND
 ((|testBitVector| |pv$| 8)
   (PROGN
    (QSETREFV |$| 44
     (CONS (|dispatchFunction| |QFCAT-;convert;AF;10|) |$|))
    (QSETREFV |$| 47
     (CONS (|dispatchFunction| |QFCAT-;convert;ADf;11|) |$|)))))
(COND
 ((|testBitVector| |pv$| 9)
  (COND
   ((|HasAttribute| |#2| (QUOTE |canonicalUnitNormal|))
     (QSETREFV |$| 50 (CONS (|dispatchFunction| |QFCAT-;<;2AB;12|) |$|)))
   ((QUOTE T)
     (QSETREFV |$| 50 (CONS (|dispatchFunction| |QFCAT-;<;2AB;13|) |$|)))))
 ((|testBitVector| |pv$| 10)
  (QSETREFV |$| 50 (CONS (|dispatchFunction| |QFCAT-;<;2AB;14|) |$|))))
(COND
 ((|testBitVector| |pv$| 3)
   (QSETREFV |$| 54
    (CONS (|dispatchFunction| |QFCAT-;fractionPart;2A;15|) |$|))))
(COND
 ((|testBitVector| |pv$| 4)
  (PROGN
   (QSETREFV |$| 57 (CONS (|dispatchFunction| |QFCAT-;coerce;SA;16|) |$|))
   (QSETREFV |$| 60 (CONS (|dispatchFunction| |QFCAT-;retract;AS;17|) |$|))
   (QSETREFV |$| 65
    (CONS (|dispatchFunction| |QFCAT-;retractIfCan;AU;18|) |$|)))))
(COND
 ((|HasCategory| |#2| (QUOTE (|ConvertibleTo| (|Pattern| (|Integer|)))))
  (PROGN
   (QSETREFV |$| 69 (CONS (|dispatchFunction| |QFCAT-;convert;AP;19|) |$|))
   (COND
    ((|HasCategory| |#2| (QUOTE (|PatternMatchable| (|Integer|))))
      (QSETREFV |$| 74
       (CONS
        (|dispatchFunction| |QFCAT-;patternMatch;AP2Pmr;20|) |$|)))))))
(COND
 ((|HasCategory| |#2| (QUOTE (|ConvertibleTo| (|Pattern| (|Float|)))))
  (PROGN
   (QSETREFV |$| 78 (CONS (|dispatchFunction| |QFCAT-;convert;AP;21|) |$|))
   (COND
    ((|HasCategory| |#2| (QUOTE (|PatternMatchable| (|Float|))))
     (QSETREFV |$| 83
      (CONS (|dispatchFunction| |QFCAT-;patternMatch;AP2Pmr;22|) |$|)))))))
(COND
```

```
     ((|testBitVector| |pv$| 11)
      (PROGN
       (QSETREFV |$| 90 (CONS (|dispatchFunction| |QFCAT-;coerce;FA;23|) |$|))
       (COND
        ((|domainEqual| |#2| (|Integer|)))
        ((QUOTE T)
         (PROGN
          (QSETREFV |$| 92
           (CONS (|dispatchFunction| |QFCAT-;retract;AI;24|) |$|))
          (QSETREFV |$| 95
           (CONS (|dispatchFunction| |QFCAT-;retractIfCan;AU;25|) |$|)))))))
    (COND
     ((|testBitVector| |pv$| 2)
       (QSETREFV |$| 98 (CONS (|dispatchFunction| |QFCAT-;random;A;26|) |$|))))
    |$|))))

(setf (get
 (QUOTE |QuotientFieldCategory&|)
 (QUOTE |infovec|))
 (LIST
  (QUOTE #(NIL NIL NIL NIL NIL NIL (|local| |#1|) (|local| |#2|)
   (0 . |numer|) (5 . |coerce|) |QFCAT-;numerator;2A;1| (10 . |denom|)
   |QFCAT-;denominator;2A;2| (15 . |init|) (19 . |One|) (23 . |/|)
   (29 . |init|) (|Union| |$| (QUOTE "failed")) (33 . |nextItem|)
   (38 . |One|) (42 . |nextItem|) (|Mapping| 7 7) |QFCAT-;map;M2A;5|
   (|Matrix| 7) (|Matrix| 6) (|MatrixCommonDenominator| 7 6)
   (47 . |clearDenominator|) (|Matrix| |$|) |QFCAT-;reducedSystem;MM;6|
   (|NonNegativeInteger|) (52 . |characteristic|)
   |QFCAT-;characteristic;Nni;7| (56 . |*|) (62 . |-|)
   (|PositiveInteger|) (68 . |**|) |QFCAT-;differentiate;AMA;8|
   (|InputForm|) (74 . |convert|) (79 . |/|) (85 . |convert|)
   (|Float|) (90 . |convert|) (95 . |/|) (101 . |convert|)
   (|DoubleFloat|) (106 . |convert|) (111 . |convert|) (|Boolean|)
   (116 . |<|) (122 . |<|) (128 . |Zero|) (132 . |wholePart|)
   (137 . |-|) (143 . |fractionPart|) (|Symbol|) (148 . |coerce|)
   (153 . |coerce|) (158 . |retract|) (163 . |retract|)
   (168 . |retract|) (|Union| 7 (QUOTE "failed")) (173 . |retractIfCan|)
   (|Union| 55 (QUOTE "failed")) (178 . |retractIfCan|)
   (183 . |retractIfCan|) (|Pattern| 84) (188 . |convert|)
   (193 . |/|) (199 . |convert|) (|PatternMatchResult| 84 6)
   (|PatternMatchQuotientFieldCategory| 84 7 6) (204 . |patternMatch|)
   (|PatternMatchResult| 84 |$|) (211 . |patternMatch|) (|Pattern| 41)
   (218 . |convert|) (223 . |/|) (229 . |convert|)
   (|PatternMatchResult| 41 6)
   (|PatternMatchQuotientFieldCategory| 41 7 6) (234 . |patternMatch|)
   (|PatternMatchResult| 41 |$|) (241 . |patternMatch|) (|Integer|)
   (|Fraction| 84) (248 . |numer|) (253 . |coerce|) (258 . |denom|)
   (263 . |/|) (269 . |coerce|) (274 . |retract|) (279 . |retract|)
   (|Union| 84 (QUOTE "failed")) (284 . |retractIfCan|)
   (289 . |retractIfCan|) (294 . |random|) (298 . |zero?|)
```

```
 (303 . |random|) (|Vector| 6) (307 . |coerce|) (312 . |horizConcat|)
 (318 . |reducedSystem|) (323 . |minRowIndex|) (328 . |maxRowIndex|)
 (333 . |minColIndex|) (338 . |maxColIndex|) (343 . |subMatrix|)
 (|Vector| 7) (352 . |column|) (|Record| (|:| |mat| 23)
 (|:| |vec| 108)) (|Vector| |$|) |QFCAT-;reducedSystem;MVR;27|
 (|Union| 85 (QUOTE "failed")) (|Record| (|:| |mat| 115)
 (|:| |vec| (|Vector| 84))) (|Matrix| 84) (|List| 55) (|List| 29)
 (|OutputForm|)))
(QUOTE #(|retractIfCan| 358 |retract| 368 |reducedSystem| 378
 |random| 389 |patternMatch| 393 |numerator| 407 |nextItem| 412
 |map| 417 |init| 423 |fractionPart| 427 |differentiate| 432
 |denominator| 438 |convert| 443 |coerce| 468 |characteristic| 478
 |<| 482))
(QUOTE NIL)
(CONS
 (|makeByteWordVec2| 1 (QUOTE NIL))
 (CONS
  (QUOTE #())
  (CONS (QUOTE #())
   (|makeByteWordVec2| 112 (QUOTE
    (1 6 7 0 8 1 6 0 7 9 1 6 7 0 11 0 7 0 13 0 7 0 14 2 6 0 7 7 15 0
     0 0 16 1 7 17 0 18 0 6 0 19 1 0 17 0 20 1 25 23 24 26 0 7 29 30 2
     7 0 0 0 32 2 7 0 0 0 33 2 7 0 0 34 35 1 7 37 0 38 2 37 0 0 0 39 1
     0 37 0 40 1 7 41 0 42 2 41 0 0 0 43 1 0 41 0 44 1 7 45 0 46 1 0 45
     0 47 2 7 48 0 0 49 2 0 48 0 0 50 0 7 0 51 1 6 7 0 52 2 6 0 0 0 53 1
     0 0 0 54 1 7 0 55 56 1 0 0 55 57 1 6 7 0 58 1 7 55 0 59 1 0 55 0 60
     1 6 61 0 62 1 7 63 0 64 1 0 63 0 65 1 7 66 0 67 2 66 0 0 0 68 1 0
     66 0 69 3 71 70 6 66 70 72 3 0 73 0 66 73 74 1 7 75 0 76 2 75 0 0
     0 77 1 0 75 0 78 3 80 79 6 75 79 81 3 0 82 0 75 82 83 1 85 84 0 86
     1 6 0 84 87 1 85 84 0 88 2 6 0 0 0 89 1 0 0 85 90 1 7 84 0 91 1 0
     84 0 92 1 7 93 0 94 1 0 93 0 95 0 7 0 96 1 7 48 0 97 0 0 0 98 1 24
     0 99 100 2 24 0 0 0 101 1 6 23 27 102 1 23 84 0 103 1 23 84 0 104
     1 23 84 0 105 1 23 84 0 106 5 23 0 0 84 84 84 84 107 2 23 108 0 84
     109 1 0 93 0 95 1 0 63 0 65 1 0 84 0 92 1 0 55 0 60 2 0 110 27 111
     112 1 0 23 27 28 0 0 0 98 3 0 82 0 75 82 83 3 0 73 0 66 73 74 1 0
     0 0 10 1 0 17 0 20 2 0 0 21 0 22 0 0 0 16 1 0 0 0 54 2 0 0 0 21 36
     1 0 0 0 12 1 0 45 0 47 1 0 37 0 40 1 0 41 0 44 1 0 66 0 69 1 0 75 0
     78 1 0 0 55 57 1 0 0 85 90 0 0 29 31 2 0 48 0 0 50))))))
(QUOTE |lookupComplete|)))
```

———-

21.49 RCAGG.lsp BOOTSTRAP

RCAGG depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **RCAGG** category which we can write into the **MID** directory. We compile the lisp code and copy the **RCAGG.o** file to the **OUT** directory.

This is eventually forcibly replaced by a recompiled version.
Note that this code is not included in the generated catdef.spad file.

— RCAGG.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |RecursiveAggregate;CAT| (QUOTE NIL))

(SETQ |RecursiveAggregate;AL| (QUOTE NIL))

(DEFUN |RecursiveAggregate| (#1=#:G84501)
 (LET (#2=#:G84502)
  (COND
   ((SETQ #2# (|assoc| (|devaluate| #1#) |RecursiveAggregate;AL|)) (CDR #2#))
   (T
    (SETQ |RecursiveAggregate;AL|
     (|cons5|
      (CONS (|devaluate| #1#) (SETQ #2# (|RecursiveAggregate;| #1#)))
      |RecursiveAggregate;AL|))
    #2#))))

(DEFUN |RecursiveAggregate;| (|t#1|)
 (PROG (#1=#:G84500)
  (RETURN
   (PROG1
    (LETT #1#
     (|sublisV|
      (PAIR (QUOTE (|t#1|)) (LIST (|devaluate| |t#1|)))
      (COND
       (|RecursiveAggregate;CAT|)
       ((QUOTE T)
        (LETT |RecursiveAggregate;CAT|
         (|Join|
          (|HomogeneousAggregate| (QUOTE |t#1|))
          (|mkCategory|
           (QUOTE |domain|)
           (QUOTE (
            ((|children| ((|List| |$|) |$|)) T)
            ((|nodes| ((|List| |$|) |$|)) T)
            ((|leaf?| ((|Boolean|) |$|)) T)
            ((|value| (|t#1| |$|)) T)
            ((|elt| (|t#1| |$| "value")) T)
            ((|cyclic?| ((|Boolean|) |$|)) T)
            ((|leaves| ((|List| |t#1|) |$|)) T)
            ((|distance| ((|Integer|) |$| |$|)) T)
            ((|child?| ((|Boolean|) |$| |$|)) (|has| |t#1| (|SetCategory|)))
            ((|node?| ((|Boolean|) |$| |$|)) (|has| |t#1| (|SetCategory|)))
```

```
           ((|setchildren!| (|$| |$| (|List| |$|)))
             (|has| |$| (ATTRIBUTE |shallowlyMutable|)))
           ((|setelt| (|t#1| |$| "value" |t#1|))
             (|has| |$| (ATTRIBUTE |shallowlyMutable|)))
           ((|setvalue!| (|t#1| |$| |t#1|))
             (|has| |$| (ATTRIBUTE |shallowlyMutable|)))))
          NIL
          (QUOTE ((|List| |$|) (|Boolean|) (|Integer|) (|List| |t#1|)))
          NIL))
        . #2=(|RecursiveAggregate|)))))
    . #2#)
   (SETELT #1# 0 (LIST (QUOTE |RecursiveAggregate|) (|devaluate| |t#1|)))))))
```

———-

21.50 RCAGG-.lsp BOOTSTRAP

RCAGG- depends on **RCAGG**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **RCAGG-** category which we can write into the **MID** directory. We compile the lisp code and copy the **RCAGG-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— **RCAGG-.lsp BOOTSTRAP** —

```
(|/VERSIONCHECK| 2)

(DEFUN |RCAGG-;elt;AvalueS;1| (|x| G84515 |$|)
 (SPADCALL |x| (QREFELT |$| 8)))

(DEFUN |RCAGG-;setelt;Avalue2S;2| (|x| G84517 |y| |$|)
 (SPADCALL |x| |y| (QREFELT |$| 11)))

(DEFUN |RCAGG-;child?;2AB;3| (|x| |l| |$|)
 (SPADCALL |x| (SPADCALL |l| (QREFELT |$| 14)) (QREFELT |$| 17)))

(DEFUN |RecursiveAggregate&| (|#1| |#2|)
 (PROG (|DV$1| |DV$2| |dv$| |$| |pv$|)
  (RETURN
   (PROGN
    (LETT |DV$1| (|devaluate| |#1|) . #1=(|RecursiveAggregate&|))
    (LETT |DV$2| (|devaluate| |#2|) . #1#)
    (LETT |dv$| (LIST (QUOTE |RecursiveAggregate&|) |DV$1| |DV$2|) . #1#)
    (LETT |$| (make-array 19) . #1#)
    (QSETREFV |$| 0 |dv$|)
    (QSETREFV |$| 3
```

```
     (LETT |pv$|
      (|buildPredVector| 0 0
       (LIST
        (|HasAttribute| |#1| (QUOTE |shallowlyMutable|))
        (|HasCategory| |#2| (QUOTE (|SetCategory|)))))
      . #1#))
    (|stuffDomainSlots| |$|)
    (QSETREFV |$| 6 |#1|)
    (QSETREFV |$| 7 |#2|)
    (COND
     ((|testBitVector| |pv$| 1)
      (QSETREFV |$| 12
       (CONS (|dispatchFunction| |RCAGG-;setelt;Avalue2S;2|) |$|))))
    (COND
     ((|testBitVector| |pv$| 2)
      (QSETREFV |$| 18 (CONS (|dispatchFunction| |RCAGG-;child?;2AB;3|) |$|))))
    |$|))))

(setf (get
 (QUOTE |RecursiveAggregate&|)
 (QUOTE |infovec|))
 (LIST
  (QUOTE #(NIL NIL NIL NIL NIL NIL (|local| |#1|) (|local| |#2|)
   (0 . |value|) (QUOTE "value") |RCAGG-;elt;AvalueS;1| (5 . |setvalue!|)
   (11 . |setelt|) (|List| |$|) (18 . |children|) (|Boolean|) (|List| 6)
   (23 . |member?|) (29 . |child?|)))
  (QUOTE #(|setelt| 35 |elt| 42 |child?| 48))
  (QUOTE NIL)
  (CONS (|makeByteWordVec2| 1 (QUOTE NIL))
 (CONS
  (QUOTE #())
  (CONS
   (QUOTE #())
   (|makeByteWordVec2| 18 (QUOTE (1 6 7 0 8 2 6 7 0 7 11 3 0 7 0 9 7 12
     1 6 13 0 14 2 16 15 6 0 17 2 0 15 0 0 18 3 0 7 0 9 7 12 2 0 7 0 9
     10 2 0 15 0 0 18))))))
  (QUOTE |lookupComplete|)))
```

21.51 RING.lsp BOOTSTRAP

RING depends on itself. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **RING** category which we can write into the **MID** directory. We compile the lisp code and copy the **RING.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— **RING.lsp BOOTSTRAP** —

```
(|/VERSIONCHECK| 2)

(SETQ |Ring;AL| (QUOTE NIL))

(DEFUN |Ring| NIL
  (LET (#:G82789)
    (COND
      (|Ring;AL|)
      (T (SETQ |Ring;AL| (|Ring;|))))))

(DEFUN |Ring;| NIL
  (PROG (#1=#:G82787)
    (RETURN
      (PROG1
        (LETT #1#
          (|Join|
            (|Rng|)
            (|Monoid|)
            (|LeftModule| (QUOTE |$|))
            (|mkCategory|
              (QUOTE |domain|)
              (QUOTE (
                ((|characteristic| ((|NonNegativeInteger|))) T)
                ((|coerce| (|$| (|Integer|))) T)))
              (QUOTE ((|unitsKnown| T)))
              (QUOTE ((|Integer|) (|NonNegativeInteger|)))
              NIL))
           |Ring|)
        (SETELT #1# 0 (QUOTE (|Ring|)))))))

(setf (get (QUOTE |Ring|) (QUOTE NILADIC)) T)
```

———-

21.52 RING-.lsp BOOTSTRAP

RING- depends on **RING**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **RING-** category which we can write into the **MID** directory. We compile the lisp code and copy the **RING-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— RING-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |RING-;coerce;IS;1| (|n| |$|)
  (SPADCALL |n| (|spadConstant| |$| 7) (QREFELT |$| 9)))

(DEFUN |Ring&| (|#1|)
  (PROG (|DV$1| |dv$| |$| |pv$|)
    (RETURN
      (PROGN
        (LETT |DV$1| (|devaluate| |#1|) . #1=(|Ring&|))
        (LETT |dv$| (LIST (QUOTE |Ring&|) |DV$1|) . #1#)
        (LETT |$| (make-array 12) . #1#)
        (QSETREFV |$| 0 |dv$|)
        (QSETREFV |$| 3 (LETT |pv$| (|buildPredVector| 0 0 NIL) . #1#))
        (|stuffDomainSlots| |$|)
        (QSETREFV |$| 6 |#1|)
        |$|))))

(setf (get
  (QUOTE |Ring&|)
  (QUOTE |infovec|))
  (LIST
    (QUOTE
      #(NIL NIL NIL NIL NIL NIL
        (|local| |#1|)
        (0 . |One|)
        (|Integer|)
        (4 . |*|)
        |RING-;coerce;IS;1|
        (|OutputForm|)))
  (QUOTE #(|coerce| 10))
  (QUOTE NIL)
  (CONS
    (|makeByteWordVec2| 1 (QUOTE NIL))
    (CONS
      (QUOTE #())
      (CONS
        (QUOTE #())
        (|makeByteWordVec2| 10 (QUOTE (0 6 0 7 2 6 0 8 0 9 1 0 0 8 10))))))
   (QUOTE |lookupComplete|)))
```

———-

21.53 RNG.lsp BOOTSTRAP

RNG depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **RNG** category which we can write into the **MID** directory. We compile the lisp code and copy the **RNG.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— **RNG.lsp BOOTSTRAP** —

```
(|/VERSIONCHECK| 2)

(SETQ |Rng;AL| (QUOTE NIL))

(DEFUN |Rng| NIL
  (LET (#:G82722)
    (COND
      (|Rng;AL|)
      (T (SETQ |Rng;AL| (|Rng;|))))))

(DEFUN |Rng;| NIL
  (PROG (#1=#:G82720)
    (RETURN
      (PROG1
        (LETT #1# (|Join| (|AbelianGroup|) (|SemiGroup|)) |Rng|)
        (SETELT #1# 0 (QUOTE (|Rng|)))))))

(setf (get (QUOTE |Rng|) (QUOTE NILADIC)) T)
```

———-

21.54 RNS.lsp BOOTSTRAP

RNS depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **RNS** category which we can write into the **MID** directory. We compile the lisp code and copy the **RNS.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— **RNS.lsp BOOTSTRAP** —

```
(|/VERSIONCHECK| 2)
```

```
(SETQ |RealNumberSystem;AL| (QUOTE NIL))

(DEFUN |RealNumberSystem| NIL
  (LET (#:G105478)
    (COND
      (|RealNumberSystem;AL|)
      (T (SETQ |RealNumberSystem;AL| (|RealNumberSystem;|))))))

(DEFUN |RealNumberSystem;| NIL
  (PROG (#1=#:G105476)
    (RETURN
      (PROG1
        (LETT #1#
          (|sublisV|
            (PAIR
              (QUOTE (#2=#:G105472 #3=#:G105473 #4=#:G105474 #5=#:G105475))
              (LIST
                (QUOTE (|Integer|))
                (QUOTE (|Fraction| (|Integer|)))
                (QUOTE (|Pattern| (|Float|)))
                (QUOTE (|Float|))))
            (|Join|
              (|Field|)
              (|OrderedRing|)
              (|RealConstant|)
              (|RetractableTo| (QUOTE #2#))
              (|RetractableTo| (QUOTE #3#))
              (|RadicalCategory|)
              (|ConvertibleTo| (QUOTE #4#))
              (|PatternMatchable| (QUOTE #5#))
              (|CharacteristicZero|)
              (|mkCategory|
                (QUOTE |domain|)
                (QUOTE (
                  ((|norm| (|$| |$|)) T)
                  ((|ceiling| (|$| |$|)) T)
                  ((|floor| (|$| |$|)) T)
                  ((|wholePart| ((|Integer|) |$|)) T)
                  ((|fractionPart| (|$| |$|)) T)
                  ((|truncate| (|$| |$|)) T)
                  ((|round| (|$| |$|)) T)
                  ((|abs| (|$| |$|)) T)))
                NIL
                (QUOTE ((|Integer|)))
                NIL)))
          |RealNumberSystem|)
        (SETELT #1# 0 (QUOTE (|RealNumberSystem|)))))))

(setf (get (QUOTE |RealNumberSystem|) (QUOTE NILADIC)) T)
```

———-

21.55 RNS-.lsp BOOTSTRAP

RNS- depends **RNS**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **RNS-** category which we can write into the **MID** directory. We compile the lisp code and copy the **RNS.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— RNS-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(PUT
  (QUOTE |RNS-;characteristic;Nni;1|)
  (QUOTE |SPADreplace|)
  (QUOTE (XLAM NIL 0)))

(DEFUN |RNS-;characteristic;Nni;1| (|$|) 0)

(DEFUN |RNS-;fractionPart;2S;2| (|x| |$|)
  (SPADCALL |x| (SPADCALL |x| (QREFELT |$| 9)) (QREFELT |$| 10)))

(DEFUN |RNS-;truncate;2S;3| (|x| |$|)
  (COND
    ((SPADCALL |x| (QREFELT |$| 13))
      (SPADCALL
        (SPADCALL
          (SPADCALL |x| (QREFELT |$| 14))
          (QREFELT |$| 15))
        (QREFELT |$| 14)))
    ((QUOTE T) (SPADCALL |x| (QREFELT |$| 15)))))

(DEFUN |RNS-;round;2S;4| (|x| |$|)
  (COND
    ((SPADCALL |x| (QREFELT |$| 13))
      (SPADCALL
        (SPADCALL |x|
          (SPADCALL
            (|spadConstant| |$| 17)
            (SPADCALL 2 (QREFELT |$| 19))
            (QREFELT |$| 20))
          (QREFELT |$| 10))
```

```
        (QREFELT |$| 9)))
    ((QUOTE T)
      (SPADCALL
        (SPADCALL |x|
          (SPADCALL
            (|spadConstant| |$| 17)
            (SPADCALL 2 (QREFELT |$| 19))
            (QREFELT |$| 20))
          (QREFELT |$| 21))
        (QREFELT |$| 9)))))

(DEFUN |RNS-;norm;2S;5| (|x| |$|)
  (SPADCALL |x| (QREFELT |$| 23)))

(DEFUN |RNS-;coerce;FS;6| (|x| |$|)
  (SPADCALL
    (SPADCALL
      (SPADCALL |x| (QREFELT |$| 26))
      (QREFELT |$| 19))
    (SPADCALL
      (SPADCALL |x| (QREFELT |$| 27))
      (QREFELT |$| 19))
    (QREFELT |$| 20)))

(DEFUN |RNS-;convert;SP;7| (|x| |$|)
  (SPADCALL (SPADCALL |x| (QREFELT |$| 30)) (QREFELT |$| 32)))

(DEFUN |RNS-;floor;2S;8| (|x| |$|)
  (PROG (|x1|)
    (RETURN
      (SEQ
        (LETT |x1|
          (SPADCALL (SPADCALL |x| (QREFELT |$| 34)) (QREFELT |$| 19))
          |RNS-;floor;2S;8|)
        (EXIT
          (COND
            ((SPADCALL |x| |x1| (QREFELT |$| 35)) |x|)
            ((SPADCALL |x| (|spadConstant| |$| 36) (QREFELT |$| 37))
              (SPADCALL |x1| (|spadConstant| |$| 17) (QREFELT |$| 10)))
            ((QUOTE T) |x1|)))))))

(DEFUN |RNS-;ceiling;2S;9| (|x| |$|)
  (PROG (|x1|)
    (RETURN
      (SEQ
        (LETT |x1|
          (SPADCALL (SPADCALL |x| (QREFELT |$| 34)) (QREFELT |$| 19))
          |RNS-;ceiling;2S;9|)
        (EXIT
          (COND
```

```
            ((SPADCALL |x| |x1| (QREFELT |$| 35)) |x|)
            ((SPADCALL |x| (|spadConstant| |$| 36) (QREFELT |$| 37)) |x1|)
            ((QUOTE T)
              (SPADCALL |x1| (|spadConstant| |$| 17) (QREFELT |$| 21)))))))))

(DEFUN |RNS-;patternMatch;SP2Pmr;10| (|x| |p| |l| |$|)
  (PROG (|r|)
    (RETURN
      (SEQ
        (COND
          ((SPADCALL |p| (QREFELT |$| 40))
            (SPADCALL |p| |x| |l| (QREFELT |$| 42)))
          ((SPADCALL |p| (QREFELT |$| 43))
            (SEQ
              (LETT |r|
                (SPADCALL |p| (QREFELT |$| 45))
                |RNS-;patternMatch;SP2Pmr;10|)
              (EXIT
                (COND
                  ((QEQCAR |r| 0)
                    (COND
                      ((SPADCALL
                         (SPADCALL |x| (QREFELT |$| 30))
                         (QCDR |r|)
                         (QREFELT |$| 46))
                        |l|)
                       ((QUOTE T) (SPADCALL (QREFELT |$| 47)))))
                   ((QUOTE T) (SPADCALL (QREFELT |$| 47)))))))
          ((QUOTE T) (SPADCALL (QREFELT |$| 47))))))))

(DEFUN |RealNumberSystem&| (|#1|)
  (PROG (|DV$1| |dv$| |$| |pv$|)
    (RETURN
      (PROGN
        (LETT |DV$1| (|devaluate| |#1|) . #1=(|RealNumberSystem&|))
        (LETT |dv$| (LIST (QUOTE |RealNumberSystem&|) |DV$1|) . #1#)
        (LETT |$| (make-array 52) . #1#)
        (QSETREFV |$| 0 |dv$|)
        (QSETREFV |$| 3 (LETT |pv$| (|buildPredVector| 0 0 NIL) . #1#))
        (|stuffDomainSlots| |$|)
        (QSETREFV |$| 6 |#1|)
        |$|))))

(setf (get
  (QUOTE |RealNumberSystem&|)
  (QUOTE |infovec|))
  (LIST
    (QUOTE
      #(NIL NIL NIL NIL NIL NIL
        (|local| |#1|)
```

```
    (|NonNegativeInteger|)
    |RNS-;characteristic;Nni;1|
    (0 . |truncate|)
    (5 . |-|)
    |RNS-;fractionPart;2S;2|
    (|Boolean|)
    (11 . |negative?|)
    (16 . |-|)
    (21 . |floor|)
    |RNS-;truncate;2S;3|
    (26 . |One|)
    (|Integer|)
    (30 . |coerce|)
    (35 . |/|)
    (41 . |+|)
    |RNS-;round;2S;4|
    (47 . |abs|)
    |RNS-;norm;2S;5|
    (|Fraction| 18)
    (52 . |numer|)
    (57 . |denom|)
    |RNS-;coerce;FS;6|
    (|Float|)
    (62 . |convert|)
    (|Pattern| 29)
    (67 . |coerce|)
    |RNS-;convert;SP;7|
    (72 . |wholePart|)
    (77 . |=|)
    (83 . |Zero|)
    (87 . |<|)
    |RNS-;floor;2S;8|
    |RNS-;ceiling;2S;9|
    (93 . |generic?|)
    (|PatternMatchResult| 29 6)
    (98 . |addMatch|)
    (105 . |constant?|)
    (|Union| 29 (QUOTE "failed"))
    (110 . |retractIfCan|)
    (115 . |=|)
    (121 . |failed|)
    (|PatternMatchResult| 29 |$|)
    |RNS-;patternMatch;SP2Pmr;10|
    (|DoubleFloat|)
    (|OutputForm|)))
(QUOTE
  #(|truncate| 125 |round| 130 |patternMatch| 135 |norm| 142
    |fractionPart| 147 |floor| 152 |convert| 157 |coerce| 162
    |characteristic| 172 |ceiling| 176))
(QUOTE NIL)
```

```
    (CONS
      (|makeByteWordVec2| 1 (QUOTE NIL))
      (CONS
        (QUOTE #())
        (CONS
          (QUOTE #())
          (|makeByteWordVec2| 49
            (QUOTE
              (1 6 0 0 9 2 6 0 0 0 10 1 6 12 0 13 1 6 0 0 14 1 6 0 0 15 0 6 0
               17 1 6 0 18 19 2 6 0 0 0 20 2 6 0 0 0 21 1 6 0 0 23 1 25 18 0
               26 1 25 18 0 27 1 6 29 0 30 1 31 0 29 32 1 6 18 0 34 2 6 12 0
               0 35 0 6 0 36 2 6 12 0 0 37 1 31 12 0 40 3 41 0 31 6 0 42 1 31
               12 0 43 1 31 44 0 45 2 29 12 0 0 46 0 41 0 47 1 0 0 0 16 1 0 0
               0 22 3 0 48 0 31 48 49 1 0 0 0 24 1 0 0 0 11 1 0 0 0 38 1 0 31
               0 33 1 0 0 25 28 1 0 0 25 28 0 0 7 8 1 0 0 0 39))))))
    (QUOTE |lookupComplete|)))
```

———-

21.56 SETAGG.lsp BOOTSTRAP

SETAGG depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **SETAGG** category which we can write into the **MID** directory. We compile the lisp code and copy the **SETAGG.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— SETAGG.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |SetAggregate;CAT| (QUOTE NIL))

(SETQ |SetAggregate;AL| (QUOTE NIL))

(DEFUN |SetAggregate| (#1=#:G83200)
 (LET (#2=#:G83201)
  (COND
   ((SETQ #2# (|assoc| (|devaluate| #1#) |SetAggregate;AL|)) (CDR #2#))
   (T
    (SETQ |SetAggregate;AL|
     (|cons5|
      (CONS (|devaluate| #1#) (SETQ #2# (|SetAggregate;| #1#)))
      |SetAggregate;AL|))
    #2#))))
```

```
(DEFUN |SetAggregate;| (|t#1|)
 (PROG (#1=#:G83199)
  (RETURN
   (PROG1
    (LETT #1#
     (|sublisV|
      (PAIR (QUOTE (|t#1|)) (LIST (|devaluate| |t#1|)))
      (COND
       (|SetAggregate;CAT|)
       ((QUOTE T)
        (LETT |SetAggregate;CAT|
         (|Join|
          (|SetCategory|)
          (|Collection| (QUOTE |t#1|))
          (|mkCategory|
           (QUOTE |domain|)
           (QUOTE (
            ((|<| ((|Boolean|) |$| |$|)) T)
            ((|brace| (|$|)) T)
            ((|brace| (|$| (|List| |t#1|))) T)
            ((|set| (|$|)) T)
            ((|set| (|$| (|List| |t#1|))) T)
            ((|intersect| (|$| |$| |$|)) T)
            ((|difference| (|$| |$| |$|)) T)
            ((|difference| (|$| |$| |t#1|)) T)
            ((|symmetricDifference| (|$| |$| |$|)) T)
            ((|subset?| ((|Boolean|) |$| |$|)) T)
            ((|union| (|$| |$| |$|)) T)
            ((|union| (|$| |$| |t#1|)) T)
            ((|union| (|$| |t#1| |$|)) T)))
           (QUOTE ((|partiallyOrderedSet| T)))
           (QUOTE ((|Boolean|) (|List| |t#1|)))
           NIL))
          . #2=(|SetAggregate|)))))
      . #2#)
    (SETELT #1# 0 (LIST (QUOTE |SetAggregate|) (|devaluate| |t#1|)))))))
```

21.57 SETAGG-.lsp BOOTSTRAP

SETAGG- depends on **SETAGG**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **SETAGG-** category which we can write into the **MID** directory. We compile the lisp code and copy the **SETAGG-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— SETAGG-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |SETAGG-;symmetricDifference;3A;1| (|x| |y| |$|)
 (SPADCALL
  (SPADCALL |x| |y| (QREFELT |$| 8))
  (SPADCALL |y| |x| (QREFELT |$| 8))
  (QREFELT |$| 9)))

(DEFUN |SETAGG-;union;ASA;2| (|s| |x| |$|)
 (SPADCALL |s| (SPADCALL (LIST |x|) (QREFELT |$| 12)) (QREFELT |$| 9)))

(DEFUN |SETAGG-;union;S2A;3| (|x| |s| |$|)
 (SPADCALL |s| (SPADCALL (LIST |x|) (QREFELT |$| 12)) (QREFELT |$| 9)))

(DEFUN |SETAGG-;difference;ASA;4| (|s| |x| |$|)
 (SPADCALL |s| (SPADCALL (LIST |x|) (QREFELT |$| 12)) (QREFELT |$| 8)))

(DEFUN |SetAggregate&| (|#1| |#2|)
 (PROG (|DV$1| |DV$2| |dv$| |$| |pv$|)
  (RETURN
   (PROGN
    (LETT |DV$1| (|devaluate| |#1|) . #1=(|SetAggregate&|))
    (LETT |DV$2| (|devaluate| |#2|) . #1#)
    (LETT |dv$| (LIST (QUOTE |SetAggregate&|) |DV$1| |DV$2|) . #1#)
    (LETT |$| (make-array 16) . #1#)
    (QSETREFV |$| 0 |dv$|)
    (QSETREFV |$| 3 (LETT |pv$| (|buildPredVector| 0 0 NIL) . #1#))
    (|stuffDomainSlots| |$|)
    (QSETREFV |$| 6 |#1|)
    (QSETREFV |$| 7 |#2|)
    |$|))))

(setf (get
 (QUOTE |SetAggregate&|)
 (QUOTE |infovec|))
 (LIST
  (QUOTE #(NIL NIL NIL NIL NIL NIL (|local| |#1|) (|local| |#2|)
    (0 . |difference|) (6 . |union|) |SETAGG-;symmetricDifference;3A;1|
    (|List| 7) (12 . |brace|) |SETAGG-;union;ASA;2| |SETAGG-;union;S2A;3|
    |SETAGG-;difference;ASA;4|))
  (QUOTE #(|union| 17 |symmetricDifference| 29 |difference| 35))
  (QUOTE NIL)
  (CONS
   (|makeByteWordVec2| 1 (QUOTE NIL))
   (CONS
    (QUOTE #())
```

```
    (CONS
     (QUOTE #())
     (|makeByteWordVec2| 15 (QUOTE (2 6 0 0 0 8 2 6 0 0 0 9 1 6 0 11 12 2
       0 0 7 0 14 2 0 0 0 7 13 2 0 0 0 0 10 2 0 0 0 7 15))))))
  (QUOTE |lookupComplete|)))
```

———-

21.58 SETCAT.lsp BOOTSTRAP

SETCAT needs **SINT** which needs **UFD** which needs **GCDDOM** which needs **COMRING** which needs **RING** which needs **RNG** which needs **ABELGRP** which needs **CABMON** which needs **ABELMON** which needs **ABELSG** which needs **SETCAT**. We break this chain with **SETCAT.lsp** which we cache here. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **SETCAT** category which we can write into the **MID** directory. We compile the lisp code and copy the **SETCAT.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— SETCAT.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |SetCategory;AL| (QUOTE NIL))

(DEFUN |SetCategory| NIL
  (LET (#:G82359)
    (COND
      (|SetCategory;AL|)
      (T (SETQ |SetCategory;AL| (|SetCategory;|))))))

(DEFUN |SetCategory;| NIL
  (PROG (#1=#:G82357)
    (RETURN
      (PROG1
        (LETT #1#
          (|sublisV|
            (PAIR
              (QUOTE (#2=#:G82356))
              (LIST (QUOTE (|OutputForm|))))
            (|Join|
              (|BasicType|)
              (|CoercibleTo| (QUOTE #2#))
              (|mkCategory|
                (QUOTE |domain|)
```

```
                (QUOTE (
                  ((|hash| ((|SingleInteger|) |$|)) T)
                  ((|latex| ((|String|) |$|)) T)))
                NIL
                (QUOTE ((|String|) (|SingleInteger|)))
                NIL)))
          |SetCategory|)
        (SETELT #1# 0 (QUOTE (|SetCategory|)))))))

(setf (get (QUOTE |SetCategory|) (QUOTE NILADIC)) T)
```

———-

21.59 SETCAT-.lsp BOOTSTRAP

SETCAT- is the implementation of the operations exported by **SETCAT**. It comes into existance whenever **SETCAT** gets compiled by Axiom. However this will not happen at the lisp level so we also cache this information here. See the explanation under the **SETCAT.lsp** section for more details.

Note that this code is not included in the generated catdef.spad file.

— **SETCAT-.lsp BOOTSTRAP** —

```
(|/VERSIONCHECK| 2)

(PUT
  (QUOTE |SETCAT-;hash;SSi;1|)
  (QUOTE |SPADreplace|)
  (QUOTE (XLAM (|s|) 0)))

(DEFUN |SETCAT-;hash;SSi;1| (|s| |$|) 0)

(PUT
  (QUOTE |SETCAT-;latex;SS;2|)
  (QUOTE |SPADreplace|)
  (QUOTE (XLAM (|s|) "\\mbox{\\bf Unimplemented}")))

(DEFUN |SETCAT-;latex;SS;2| (|s| |$|)
  "\\mbox{\\bf Unimplemented}")

(DEFUN |SetCategory&| (|#1|)
 (PROG (|DV$1| |dv$| |$| |pv$|)
   (RETURN
     (PROGN
       (LETT |DV$1| (|devaluate| |#1|) . #1=(|SetCategory&|))
```

```
       (LETT |dv$| (LIST (QUOTE |SetCategory&|) |DV$1|) . #1#)
       (LETT |$| (make-array 11) . #1#)
       (QSETREFV |$| 0 |dv$|)
       (QSETREFV |$| 3 (LETT |pv$| (|buildPredVector| 0 0 NIL) . #1#))
       (|stuffDomainSlots| |$|)
       (QSETREFV |$| 6 |#1|)
       |$|))))

(setf (get
  (QUOTE |SetCategory&|)
  (QUOTE |infovec|))
  (LIST
    (QUOTE
      #(NIL NIL NIL NIL NIL NIL
        (|local| |#1|)
        (|SingleInteger|)
        |SETCAT-;hash;SSi;1|
        (|String|)
        |SETCAT-;latex;SS;2|))
    (QUOTE
      #(|latex| 0 |hash| 5))
    (QUOTE NIL)
    (CONS
      (|makeByteWordVec2| 1 (QUOTE NIL))
      (CONS
        (QUOTE #())
        (CONS
          (QUOTE #())
          (|makeByteWordVec2|
            10
            (QUOTE (1 0 9 0 10 1 0 7 0 8))))))
    (QUOTE |lookupComplete|)))
```

———-

21.60 STAGG.lsp BOOTSTRAP

STAGG depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **STAGG** category which we can write into the **MID** directory. We compile the lisp code and copy the **STAGG.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— STAGG.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |StreamAggregate;CAT| (QUOTE NIL))

(SETQ |StreamAggregate;AL| (QUOTE NIL))

(DEFUN |StreamAggregate| (#1=#:G87035)
 (LET (#2=#:G87036)
  (COND
   ((SETQ #2# (|assoc| (|devaluate| #1#) |StreamAggregate;AL|)) (CDR #2#))
   (T
    (SETQ |StreamAggregate;AL|
     (|cons5|
      (CONS (|devaluate| #1#) (SETQ #2# (|StreamAggregate;| #1#)))
      |StreamAggregate;AL|))
    #2#))))
v
(DEFUN |StreamAggregate;| (|t#1|)
 (PROG (#1=#:G87034)
  (RETURN
   (PROG1
    (LETT #1#
     (|sublisV|
      (PAIR (QUOTE (|t#1|)) (LIST (|devaluate| |t#1|)))
      (COND
       (|StreamAggregate;CAT|)
       ((QUOTE T)
        (LETT |StreamAggregate;CAT|
         (|Join|
          (|UnaryRecursiveAggregate| (QUOTE |t#1|))
          (|LinearAggregate| (QUOTE |t#1|))
          (|mkCategory|
           (QUOTE |domain|)
           (QUOTE (
            ((|explicitlyFinite?| ((|Boolean|) |$|)) T)
            ((|possiblyInfinite?| ((|Boolean|) |$|)) T)))
           NIL
           (QUOTE ((|Boolean|)))
           NIL))
         . #2=(|StreamAggregate|)))))
      . #2#)
    (SETELT #1# 0 (LIST (QUOTE |StreamAggregate|) (|devaluate| |t#1|)))))))
```

21.61 STAGG-.lsp BOOTSTRAP

STAGG- depends on **STAGG**. We need to break this cycle to build the algebra. So we

keep a cached copy of the translated **STAGG-** category which we can write into the **MID** directory. We compile the lisp code and copy the **STAGG-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— STAGG-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |STAGG-;explicitlyFinite?;AB;1| (|x| |$|)
 (COND ((SPADCALL |x| (QREFELT |$| 9)) (QUOTE NIL)) ((QUOTE T) (QUOTE T))))

(DEFUN |STAGG-;possiblyInfinite?;AB;2| (|x| |$|)
 (SPADCALL |x| (QREFELT |$| 9)))

(DEFUN |STAGG-;first;ANniA;3| (|x| |n| |$|)
 (PROG (#1=#:G87053 |i|)
  (RETURN
   (SEQ
    (SPADCALL
     (PROGN
      (LETT #1# NIL |STAGG-;first;ANniA;3|)
      (SEQ
       (LETT |i| 1 |STAGG-;first;ANniA;3|)
       G190
       (COND ((QSGREATERP |i| |n|) (GO G191)))
       (SEQ
        (EXIT
         (LETT #1#
          (CONS
           (|STAGG-;c2| |x|
            (LETT |x| (SPADCALL |x| (QREFELT |$| 12)) |STAGG-;first;ANniA;3|)
            |$|)
           #1#)
          |STAGG-;first;ANniA;3|)))
        (LETT |i| (QSADD1 |i|) |STAGG-;first;ANniA;3|)
        (GO G190)
       G191
        (EXIT (NREVERSE0 #1#))))
     (QREFELT |$| 14))))))

(DEFUN |STAGG-;c2| (|x| |r| |$|)
 (COND
   ((SPADCALL |x| (QREFELT |$| 17)) (|error| "Index out of range"))
   ((QUOTE T) (SPADCALL |x| (QREFELT |$| 18)))))

(DEFUN |STAGG-;elt;AIS;5| (|x| |i| |$|)
 (PROG (#1=#:G87056)
```

```
  (RETURN
   (SEQ
    (LETT |i| (|-| |i| (SPADCALL |x| (QREFELT |$| 20))) |STAGG-;elt;AIS;5|)
    (COND
     ((OR
       (|<| |i| 0)
       (SPADCALL
        (LETT |x|
         (SPADCALL |x|
          (PROG1
           (LETT #1# |i| |STAGG-;elt;AIS;5|)
           (|check-subtype| (|>=| #1# 0) (QUOTE (|NonNegativeInteger|)) #1#))
          (QREFELT |$| 21))
         |STAGG-;elt;AIS;5|)
        (QREFELT |$| 17)))
      (EXIT (|error| "index out of range"))))
    (EXIT (SPADCALL |x| (QREFELT |$| 18)))))))

(DEFUN |STAGG-;elt;AUsA;6| (|x| |i| |$|)
 (PROG (|l| #1=#:G87060 |h| #2=#:G87062 #3=#:G87063)
  (RETURN
   (SEQ
    (LETT |l|
     (|-| (SPADCALL |i| (QREFELT |$| 24)) (SPADCALL |x| (QREFELT |$| 20)))
     |STAGG-;elt;AUsA;6|)
    (EXIT
     (COND
      ((|<| |l| 0) (|error| "index out of range"))
      ((NULL (SPADCALL |i| (QREFELT |$| 25)))
        (SPADCALL
         (SPADCALL |x|
          (PROG1
           (LETT #1# |l| |STAGG-;elt;AUsA;6|)
           (|check-subtype| (|>=| #1# 0) (QUOTE (|NonNegativeInteger|)) #1#))
          (QREFELT |$| 21))
         (QREFELT |$| 26)))
      ((QUOTE T)
       (SEQ
        (LETT |h|
         (|-| (SPADCALL |i| (QREFELT |$| 27)) (SPADCALL |x| (QREFELT |$| 20)))
         |STAGG-;elt;AUsA;6|)
        (EXIT
         (COND
          ((|<| |h| |l|) (SPADCALL (QREFELT |$| 28)))
          ((QUOTE T)
           (SPADCALL
            (SPADCALL |x|
             (PROG1
              (LETT #2# |l| |STAGG-;elt;AUsA;6|)
              (|check-subtype|
```

```
              (|>=| #2# 0) (QUOTE (|NonNegativeInteger|)) #2#))
             (QREFELT |$| 21))
            (PROG1
             (LETT #3# (|+| (|-| |h| |l|) 1) |STAGG-;elt;AUsA;6|)
             (|check-subtype| (|>=| #3# 0) (QUOTE (|NonNegativeInteger|)) #3#))
            (QREFELT |$| 29))))))))))))))

(DEFUN |STAGG-;concat;3A;7| (|x| |y| |$|)
 (SPADCALL (SPADCALL |x| (QREFELT |$| 26)) |y| (QREFELT |$| 31)))

(DEFUN |STAGG-;concat;LA;8| (|l| |$|)
 (COND
  ((NULL |l|) (SPADCALL (QREFELT |$| 28)))
  ((QUOTE T)
   (SPADCALL
    (SPADCALL (|SPADfirst| |l|) (QREFELT |$| 26))
    (SPADCALL (CDR |l|) (QREFELT |$| 34))
    (QREFELT |$| 31)))))

(DEFUN |STAGG-;map!;M2A;9| (|f| |l| |$|)
 (PROG (|y|)
  (RETURN
   (SEQ
    (LETT |y| |l| |STAGG-;map!;M2A;9|)
    (SEQ
     G190
     (COND
      ((NULL
        (COND
         ((SPADCALL |l| (QREFELT |$| 17)) (QUOTE NIL))
         ((QUOTE T) (QUOTE T))))
       (GO G191)))
     (SEQ
      (SPADCALL |l|
       (SPADCALL (SPADCALL |l| (QREFELT |$| 18)) |f|) (QREFELT |$| 36))
      (EXIT (LETT |l| (SPADCALL |l| (QREFELT |$| 12)) |STAGG-;map!;M2A;9|)))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT |y|)))))

(DEFUN |STAGG-;fill!;ASA;10| (|x| |s| |$|)
 (PROG (|y|)
  (RETURN
   (SEQ
    (LETT |y| |x| |STAGG-;fill!;ASA;10|)
    (SEQ
     G190
     (COND
```

```
      ((NULL
        (COND
         ((SPADCALL |y| (QREFELT |$| 17)) (QUOTE NIL)) ((QUOTE T) (QUOTE T))))
       (GO G191)))
     (SEQ
      (SPADCALL |y| |s| (QREFELT |$| 36))
      (EXIT (LETT |y| (SPADCALL |y| (QREFELT |$| 12)) |STAGG-;fill!;ASA;10|)))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT |x|)))))

(DEFUN |STAGG-;setelt;AI2S;11| (|x| |i| |s| |$|)
 (PROG (#1=#:G87081)
  (RETURN
   (SEQ
    (LETT |i|
     (|-| |i| (SPADCALL |x| (QREFELT |$| 20))) |STAGG-;setelt;AI2S;11|)
    (COND
     ((OR
       (|<| |i| 0)
       (SPADCALL
        (LETT |x|
         (SPADCALL |x|
          (PROG1
           (LETT #1# |i| |STAGG-;setelt;AI2S;11|)
           (|check-subtype| (|>=| #1# 0) (QUOTE (|NonNegativeInteger|)) #1#))
          (QREFELT |$| 21))
         |STAGG-;setelt;AI2S;11|)
        (QREFELT |$| 17)))
      (EXIT (|error| "index out of range"))))
    (EXIT (SPADCALL |x| |s| (QREFELT |$| 36)))))))

(DEFUN |STAGG-;setelt;AUs2S;12| (|x| |i| |s| |$|)
 (PROG (|l| |h| #1=#:G87086 #2=#:G87087 |z| |y|)
  (RETURN
   (SEQ
    (LETT |l|
     (|-| (SPADCALL |i| (QREFELT |$| 24)) (SPADCALL |x| (QREFELT |$| 20)))
      |STAGG-;setelt;AUs2S;12|)
    (EXIT
     (COND
      ((|<| |l| 0) (|error| "index out of range"))
      ((QUOTE T)
       (SEQ
        (LETT |h|
         (COND
          ((SPADCALL |i| (QREFELT |$| 25))
           (|-|
```

```
            (SPADCALL |i| (QREFELT |$| 27))
            (SPADCALL |x| (QREFELT |$| 20))))
          ((QUOTE T) (SPADCALL |x| (QREFELT |$| 41))))
         |STAGG-;setelt;AUs2S;12|)
        (EXIT
         (COND
          ((|<| |h| |l|) |s|)
          ((QUOTE T)
            (SEQ
             (LETT |y|
              (SPADCALL |x|
               (PROG1
                (LETT #1# |l| |STAGG-;setelt;AUs2S;12|)
                (|check-subtype|
                 (|>=| #1# 0) (QUOTE (|NonNegativeInteger|)) #1#))
               (QREFELT |$| 21))
              |STAGG-;setelt;AUs2S;12|)
             (LETT |z|
              (SPADCALL |y|
               (PROG1
                (LETT #2# (|+| (|-| |h| |l|) 1) |STAGG-;setelt;AUs2S;12|)
                (|check-subtype|
                  (|>=| #2# 0) (QUOTE (|NonNegativeInteger|)) #2#))
               (QREFELT |$| 21))
              |STAGG-;setelt;AUs2S;12|)
             (SEQ
               G190
               (COND
                ((NULL
                  (COND
                   ((SPADCALL |y| |z| (QREFELT |$| 42)) (QUOTE NIL))
                   ((QUOTE T) (QUOTE T))))
                 (GO G191)))
               (SEQ
                (SPADCALL |y| |s| (QREFELT |$| 36))
                (EXIT
                 (LETT |y|
                  (SPADCALL |y| (QREFELT |$| 12))
                  |STAGG-;setelt;AUs2S;12|)))
               NIL
               (GO G190)
               G191
               (EXIT NIL))
             (EXIT |s|))))))))))))

(DEFUN |STAGG-;concat!;3A;13| (|x| |y| |$|)
 (SEQ
  (COND
   ((SPADCALL |x| (QREFELT |$| 17)) |y|)
   ((QUOTE T)
```

```
     (SEQ
      (SPADCALL (SPADCALL |x| (QREFELT |$| 44)) |y| (QREFELT |$| 45))
      (EXIT |x|)))))))

(DEFUN |StreamAggregate&| (|#1| |#2|)
 (PROG (|DV$1| |DV$2| |dv$| |$| |pv$|)
  (RETURN
   (PROGN
    (LETT |DV$1| (|devaluate| |#1|) . #1=(|StreamAggregate&|))
    (LETT |DV$2| (|devaluate| |#2|) . #1#)
    (LETT |dv$| (LIST (QUOTE |StreamAggregate&|) |DV$1| |DV$2|) . #1#)
    (LETT |$| (make-array 51) . #1#)
    (QSETREFV |$| 0 |dv$|)
    (QSETREFV |$| 3 (LETT |pv$| (|buildPredVector| 0 0 NIL) . #1#))
    (|stuffDomainSlots| |$|)
    (QSETREFV |$| 6 |#1|)
    (QSETREFV |$| 7 |#2|)
    (COND
     ((|HasAttribute| |#1| (QUOTE |shallowlyMutable|))
      (PROGN
       (QSETREFV |$| 32 (CONS (|dispatchFunction| |STAGG-;concat;3A;7|) |$|))
       (QSETREFV |$| 35 (CONS (|dispatchFunction| |STAGG-;concat;LA;8|) |$|))
       (QSETREFV |$| 38 (CONS (|dispatchFunction| |STAGG-;map!;M2A;9|) |$|))
       (QSETREFV |$| 39 (CONS (|dispatchFunction| |STAGG-;fill!;ASA;10|) |$|))
       (QSETREFV |$| 40
         (CONS (|dispatchFunction| |STAGG-;setelt;AI2S;11|) |$|))
       (QSETREFV |$| 43
         (CONS (|dispatchFunction| |STAGG-;setelt;AUs2S;12|) |$|))
       (QSETREFV |$| 46
         (CONS (|dispatchFunction| |STAGG-;concat!;3A;13|) |$|)))))
    |$|))))

(setf (get
 (QUOTE |StreamAggregate&|)
 (QUOTE |infovec|))
 (LIST
  (QUOTE #(NIL NIL NIL NIL NIL NIL (|local| |#1|) (|local| |#2|)
   (|Boolean|) (0 . |cyclic?|) |STAGG-;explicitlyFinite?;AB;1|
   |STAGG-;possiblyInfinite?;AB;2| (5 . |rest|) (|List| 7)
   (10 . |construct|) (|NonNegativeInteger|) |STAGG-;first;ANniA;3|
   (15 . |empty?|) (20 . |first|) (|Integer|) (25 . |minIndex|)
   (30 . |rest|) |STAGG-;elt;AIS;5| (|UniversalSegment| 19) (36 . |lo|)
   (41 . |hasHi|) (46 . |copy|) (51 . |hi|) (56 . |empty|) (60 . |first|)
   |STAGG-;elt;AUsA;6| (66 . |concat!|) (72 . |concat|) (|List| |$|)
   (78 . |concat|) (83 . |concat|) (88 . |setfirst!|) (|Mapping| 7 7)
   (94 . |map!|) (100 . |fill!|) (106 . |setelt|) (113 . |maxIndex|)
   (118 . |eq?|) (124 . |setelt|) (131 . |tail|) (136 . |setrest!|)
   (142 . |concat!|) (QUOTE "rest") (QUOTE "last") (QUOTE "first")
   (QUOTE "value")))
  (QUOTE #(|setelt| 148 |possiblyInfinite?| 162 |map!| 167 |first| 173
```

```
   |fill!| 179 |explicitlyFinite?| 185 |elt| 190 |concat!| 202 |concat| 208))
 (QUOTE NIL)
 (CONS
  (|makeByteWordVec2| 1 (QUOTE NIL))
  (CONS
   (QUOTE #())
   (CONS
    (QUOTE #())
    (|makeByteWordVec2| 46 (QUOTE (1 6 8 0 9 1 6 0 0 12 1 6 0 13 14 1 6
      8 0 17 1 6 7 0 18 1 6 19 0 20 2 6 0 0 15 21 1 23 19 0 24 1 23 8
      0 25 1 6 0 0 26 1 23 19 0 27 0 6 0 28 2 6 0 0 15 29 2 6 0 0 0 31
      2 0 0 0 0 32 1 6 0 33 34 1 0 0 33 35 2 6 7 0 7 36 2 0 0 37 0 38 2
      0 0 0 7 39 3 0 7 0 19 7 40 1 6 19 0 41 2 6 8 0 0 42 3 0 7 0 23 7 43
      1 6 0 0 44 2 6 0 0 0 45 2 0 0 0 0 46 3 0 7 0 19 7 40 3 0 7 0 23 7 43
      1 0 8 0 11 2 0 0 37 0 38 2 0 0 0 15 16 2 0 0 0 7 39 1 0 8 0 10 2 0 7
      0 19 22 2 0 0 0 23 30 2 0 0 0 0 46 1 0 0 33 35 2 0 0 0 0 32))))))
 (QUOTE |lookupComplete|)))
```

———-

21.62 TSETCAT.lsp BOOTSTRAP

TSETCAT depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **TSETCAT** category which we can write into the **MID** directory. We compile the lisp code and copy the **TSETCAT.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— TSETCAT.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |TriangularSetCategory;CAT| (QUOTE NIL))

(SETQ |TriangularSetCategory;AL| (QUOTE NIL))

(DEFUN |TriangularSetCategory| (|&REST| #1=#:G82394 |&AUX| #2=#:G82392)
 (DSETQ #2# #1#)
 (LET (#3=#:G82393)
  (COND
   ((SETQ #3# (|assoc| (|devaluateList| #2#) |TriangularSetCategory;AL|))
     (CDR #3#))
   (T
    (SETQ |TriangularSetCategory;AL|
     (|cons5|
      (CONS
```

```
       (|devaluateList| #2#)
       (SETQ #3# (APPLY (FUNCTION |TriangularSetCategory;|) #2#)))
      |TriangularSetCategory;AL|))
     #3#))))

(DEFUN |TriangularSetCategory;| (|t#1| |t#2| |t#3| |t#4|)
 (PROG (#1=#:G82391)
  (RETURN
   (PROG1
    (LETT #1#
     (|sublisV|
      (PAIR
       (QUOTE (|t#1| |t#2| |t#3| |t#4|))
       (LIST
        (|devaluate| |t#1|)
        (|devaluate| |t#2|)
        (|devaluate| |t#3|)
        (|devaluate| |t#4|)))
      (COND
       (|TriangularSetCategory;CAT|)
       ((QUOTE T)
        (LETT |TriangularSetCategory;CAT|
         (|Join|
          (|PolynomialSetCategory|
           (QUOTE |t#1|)
           (QUOTE |t#2|)
           (QUOTE |t#3|)
           (QUOTE |t#4|))
          (|mkCategory|
           (QUOTE |domain|)
           (QUOTE (
            ((|infRittWu?| ((|Boolean|) |$| |$|)) T)
            ((|basicSet| (
               (|Union|
                (|Record| (|:| |bas| |$|) (|:| |top| (|List| |t#4|))) "failed")
               (|List| |t#4|)
               (|Mapping| (|Boolean|) |t#4| |t#4|)))
              T)
            ((|basicSet| (
               (|Union|
                (|Record| (|:| |bas| |$|) (|:| |top| (|List| |t#4|))) "failed")
               (|List| |t#4|)
               (|Mapping| (|Boolean|) |t#4|)
               (|Mapping| (|Boolean|) |t#4| |t#4|)))
              T)
            ((|initials| ((|List| |t#4|) |$|)) T)
            ((|degree| ((|NonNegativeInteger|) |$|)) T)
            ((|quasiComponent| (
               (|Record|
                (|:| |close| (|List| |t#4|))
```

```
    (|:| |open| (|List| |t#4|)))
   |$|))
  T)
((|normalized?| ((|Boolean|) |t#4| |$|)) T)
((|normalized?| ((|Boolean|) |$|)) T)
((|reduced?| (
   (|Boolean|)
   |t#4|
   |$|
   (|Mapping| (|Boolean|) |t#4| |t#4|)))
  T)
((|stronglyReduced?| ((|Boolean|) |t#4| |$|)) T)
((|headReduced?| ((|Boolean|) |t#4| |$|)) T)
((|initiallyReduced?| ((|Boolean|) |t#4| |$|)) T)
((|autoReduced?| (
  (|Boolean|)
  |$|
  (|Mapping| (|Boolean|) |t#4| (|List| |t#4|))))
  T)
((|stronglyReduced?| ((|Boolean|) |$|)) T)
((|headReduced?| ((|Boolean|) |$|)) T)
((|initiallyReduced?| ((|Boolean|) |$|)) T)
((|reduce| (
   |t#4|
   |t#4|
   |$|
   (|Mapping| |t#4| |t#4| |t#4|)
   (|Mapping| (|Boolean|) |t#4| |t#4|)))
  T)
((|rewriteSetWithReduction| (
  (|List| |t#4|)
  (|List| |t#4|)
  |$|
  (|Mapping| |t#4| |t#4| |t#4|)
  (|Mapping| (|Boolean|) |t#4| |t#4|)))
  T)
((|stronglyReduce| (|t#4| |t#4| |$|)) T)
((|headReduce| (|t#4| |t#4| |$|)) T)
((|initiallyReduce| (|t#4| |t#4| |$|)) T)
((|removeZero| (|t#4| |t#4| |$|)) T)
((|collectQuasiMonic| (|$| |$|)) T)
((|reduceByQuasiMonic| (|t#4| |t#4| |$|)) T)
((|zeroSetSplit| ((|List| |$|) (|List| |t#4|))) T)
((|zeroSetSplitIntoTriangularSystems|
  ((|List|
    (|Record| (|:| |close| |$|) (|:| |open| (|List| |t#4|))))
   (|List| |t#4|)))
  T)
((|first| ((|Union| |t#4| "failed") |$|)) T)
((|last| ((|Union| |t#4| "failed") |$|)) T)
```

```
           ((|rest| ((|Union| |$| "failed") |$|)) T)
           ((|algebraicVariables| ((|List| |t#3|) |$|)) T)
           ((|algebraic?| ((|Boolean|) |t#3| |$|)) T)
           ((|select| ((|Union| |t#4| "failed") |$| |t#3|)) T)
           ((|extendIfCan| ((|Union| |$| "failed") |$| |t#4|)) T)
           ((|extend| (|$| |$| |t#4|)) T)
           ((|coHeight| ((|NonNegativeInteger|) |$|))
             (|has| |t#3| (|Finite|)))))
          (QUOTE (
           (|finiteAggregate| T)
           (|shallowlyMutable| T)))
          (QUOTE ((|NonNegativeInteger|)
           (|Boolean|)
           (|List| |t#3|)
           (|List| (|Record| (|:| |close| |$|) (|:| |open| (|List| |t#4|))))
           (|List| |t#4|)
           (|List| |$|)))
          NIL))
    . #2=(|TriangularSetCategory|)))))
   . #2#)
  (SETELT #1# 0
   (LIST
    (QUOTE |TriangularSetCategory|)
    (|devaluate| |t#1|)
    (|devaluate| |t#2|)
    (|devaluate| |t#3|)
    (|devaluate| |t#4|)))))))
```

———-

21.63 TSETCAT-.lsp BOOTSTRAP

TSETCAT depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **TSETCAT** category which we can write into the **MID** directory. We compile the lisp code and copy the **TSETCAT.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— TSETCAT-.lsp BOOTSTRAP —

```
(/VERSIONCHECK 2)

(DEFUN |TSETCAT-;=;2SB;1| (|ts| |us| $)
 (PROG (#0=#:G1475 #1=#:G1481)
  (RETURN
```

```
   (COND
    ((SPADCALL |ts| (QREFELT $ 12)) (SPADCALL |us| (QREFELT $ 12)))
    ((OR
      (SPADCALL |us| (QREFELT $ 12))
      (NULL
       (SPADCALL
        (PROG2
         (LETT #0# (SPADCALL |ts| (QREFELT $ 14)) |TSETCAT-;=;2SB;1|)
         (QCDR #0#)
         (|check-union| (QEQCAR #0# 0) (QREFELT $ 10) #0#))
        (PROG2
         (LETT #0# (SPADCALL |us| (QREFELT $ 14)) |TSETCAT-;=;2SB;1|)
         (QCDR #0#)
         (|check-union| (QEQCAR #0# 0) (QREFELT $ 10) #0#))
        (QREFELT $ 15))))
      (QUOTE NIL))
     ((QUOTE T)
      (SPADCALL
       (PROG2
        (LETT #1# (SPADCALL |ts| (QREFELT $ 17)) |TSETCAT-;=;2SB;1|)
        (QCDR #1#)
        (|check-union| (QEQCAR #1# 0) (QREFELT $ 6) #1#))
       (PROG2
        (LETT #1# (SPADCALL |us| (QREFELT $ 17)) |TSETCAT-;=;2SB;1|)
        (QCDR #1#)
        (|check-union| (QEQCAR #1# 0) (QREFELT $ 6) #1#))
       (QREFELT $ 18)))))))

(DEFUN |TSETCAT-;infRittWu?;2SB;2| (|ts| |us| $)
 (PROG (|p| #0=#:G1489 |q| |v|)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |us| (QREFELT $ 12))
       (COND
        ((SPADCALL |ts| (QREFELT $ 12)) (QUOTE NIL))
        ((QUOTE T) (QUOTE T))))
     ((SPADCALL |ts| (QREFELT $ 12)) (QUOTE NIL))
     ((QUOTE T)
      (SEQ
       (LETT |p|
        (PROG2
         (LETT #0# (SPADCALL |ts| (QREFELT $ 20)) |TSETCAT-;infRittWu?;2SB;2|)
         (QCDR #0#)
         (|check-union| (QEQCAR #0# 0) (QREFELT $ 10) #0#))
        |TSETCAT-;infRittWu?;2SB;2|)
       (LETT |q|
        (PROG2
         (LETT #0# (SPADCALL |us| (QREFELT $ 20)) |TSETCAT-;infRittWu?;2SB;2|)
         (QCDR #0#)
```

```
         (|check-union| (QEQCAR #0# 0) (QREFELT $ 10) #0#))
        |TSETCAT-;infRittWu?;2SB;2|)
       (EXIT
        (COND
         ((SPADCALL |p| |q| (QREFELT $ 21)) (QUOTE T))
         ((SPADCALL |p| |q| (QREFELT $ 22)) (QUOTE NIL))
         ((QUOTE T)
          (SEQ
           (LETT |v| (SPADCALL |p| (QREFELT $ 23)) |TSETCAT-;infRittWu?;2SB;2|)
           (EXIT
            (SPADCALL
             (SPADCALL |ts| |v| (QREFELT $ 24))
             (SPADCALL |us| |v| (QREFELT $ 24)) (QREFELT $ 25)))))))))))))

(DEFUN |TSETCAT-;reduced?;PSMB;3| (|p| |ts| |redOp?| $)
 (PROG (|lp|)
  (RETURN
   (SEQ
    (LETT |lp| (SPADCALL |ts| (QREFELT $ 28)) |TSETCAT-;reduced?;PSMB;3|)
    (SEQ
     G190
     (COND
      ((NULL
        (COND
         ((NULL |lp|) (QUOTE NIL))
         ((QUOTE T) (SPADCALL |p| (|SPADfirst| |lp|) |redOp?|))))
       (GO G191)))
     (SEQ (EXIT (LETT |lp| (CDR |lp|) |TSETCAT-;reduced?;PSMB;3|)))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT (NULL |lp|))))))

(DEFUN |TSETCAT-;basicSet;LMU;4| (|ps| |redOp?| $)
 (PROG (|b| |bs| |p| |ts|)
  (RETURN
   (SEQ
    (LETT |ps|
     (SPADCALL (ELT $ 31) |ps| (QREFELT $ 33))
     |TSETCAT-;basicSet;LMU;4|)
    (EXIT
     (COND
      ((SPADCALL (ELT $ 34) |ps| (QREFELT $ 35)) (CONS 1 "failed"))
      ((QUOTE T)
       (SEQ
        (LETT |ps|
         (SPADCALL (ELT $ 21) |ps| (QREFELT $ 36))
         |TSETCAT-;basicSet;LMU;4|)
        (LETT |bs| (SPADCALL (QREFELT $ 37)) |TSETCAT-;basicSet;LMU;4|)
```

```
        (LETT |ts| NIL |TSETCAT-;basicSet;LMU;4|)
        (SEQ
         G190
         (COND
          ((NULL (COND ((NULL |ps|) (QUOTE NIL)) ((QUOTE T) (QUOTE T))))
            (GO G191)))
         (SEQ
          (LETT |b| (|SPADfirst| |ps|) |TSETCAT-;basicSet;LMU;4|)
          (LETT |bs|
           (SPADCALL |bs| |b| (QREFELT $ 38))
           |TSETCAT-;basicSet;LMU;4|)
          (LETT |ps| (CDR |ps|) |TSETCAT-;basicSet;LMU;4|)
          (EXIT
           (SEQ
            G190
            (COND
             ((NULL
              (COND
               ((OR
                 (NULL |ps|)
                 (SPADCALL
                  (LETT |p| (|SPADfirst| |ps|) |TSETCAT-;basicSet;LMU;4|)
                  |bs| |redOp?| (QREFELT $ 39)))
                 (QUOTE NIL))
                ((QUOTE T) (QUOTE T))))
              (GO G191)))
             (SEQ
              (LETT |ts| (CONS |p| |ts|) |TSETCAT-;basicSet;LMU;4|)
              (EXIT (LETT |ps| (CDR |ps|) |TSETCAT-;basicSet;LMU;4|)))
             NIL
             (GO G190)
             G191
             (EXIT NIL))))
           NIL
           (GO G190)
           G191
           (EXIT NIL))
        (EXIT (CONS 0 (CONS |bs| |ts|))))))))))))

(DEFUN |TSETCAT-;basicSet;LMMU;5| (|ps| |pred?| |redOp?| $)
 (PROG (|bps| |b| |bs| |p| |gps| |ts|)
  (RETURN
   (SEQ
    (LETT |ps|
     (SPADCALL (ELT $ 31) |ps| (QREFELT $ 33))
     |TSETCAT-;basicSet;LMMU;5|)
    (EXIT
     (COND
      ((SPADCALL (ELT $ 34) |ps| (QREFELT $ 35)) (CONS 1 "failed"))
      ((QUOTE T)
```

```
(SEQ
 (LETT |gps| NIL |TSETCAT-;basicSet;LMMU;5|)
 (LETT |bps| NIL |TSETCAT-;basicSet;LMMU;5|)
 (SEQ
  G190
  (COND
   ((NULL (COND ((NULL |ps|) (QUOTE NIL)) ((QUOTE T) (QUOTE T))))
     (GO G191)))
  (SEQ
   (LETT |p| (|SPADfirst| |ps|) |TSETCAT-;basicSet;LMMU;5|)
   (LETT |ps| (CDR |ps|) |TSETCAT-;basicSet;LMMU;5|)
   (EXIT
    (COND
     ((SPADCALL |p| |pred?|)
      (LETT |gps| (CONS |p| |gps|) |TSETCAT-;basicSet;LMMU;5|))
     ((QUOTE T)
      (LETT |bps| (CONS |p| |bps|) |TSETCAT-;basicSet;LMMU;5|))))))
   NIL
   (GO G190)
   G191
   (EXIT NIL))
 (LETT |gps|
  (SPADCALL (ELT $ 21) |gps| (QREFELT $ 36)) |TSETCAT-;basicSet;LMMU;5|)
 (LETT |bs| (SPADCALL (QREFELT $ 37)) |TSETCAT-;basicSet;LMMU;5|)
 (LETT |ts| NIL |TSETCAT-;basicSet;LMMU;5|)
 (SEQ
  G190
  (COND
   ((NULL (COND ((NULL |gps|) (QUOTE NIL)) ((QUOTE T) (QUOTE T))))
    (GO G191)))
  (SEQ
   (LETT |b| (|SPADfirst| |gps|) |TSETCAT-;basicSet;LMMU;5|)
   (LETT |bs|
    (SPADCALL |bs| |b| (QREFELT $ 38)) |TSETCAT-;basicSet;LMMU;5|)
   (LETT |gps| (CDR |gps|) |TSETCAT-;basicSet;LMMU;5|)
   (EXIT
    (SEQ
     G190
     (COND
      ((NULL
       (COND
        ((OR
          (NULL |gps|)
          (SPADCALL
           (LETT |p| (|SPADfirst| |gps|) |TSETCAT-;basicSet;LMMU;5|)
           |bs| |redOp?| (QREFELT $ 39)))
         (QUOTE NIL))
        ((QUOTE T) (QUOTE T))))
       (GO G191)))
     (SEQ
```

```
            (LETT |ts| (CONS |p| |ts|) |TSETCAT-;basicSet;LMMU;5|)
            (EXIT (LETT |gps| (CDR |gps|) |TSETCAT-;basicSet;LMMU;5|)))
           NIL
           (GO G190)
           G191
           (EXIT NIL))))
         NIL
         (GO G190)
         G191
         (EXIT NIL))
        (LETT |ts|
         (SPADCALL
          (ELT $ 21)
          (SPADCALL |ts| |bps| (QREFELT $ 43))
          (QREFELT $ 36))
         |TSETCAT-;basicSet;LMMU;5|)
        (EXIT (CONS 0 (CONS |bs| |ts|))))))))))))

(DEFUN |TSETCAT-;initials;SL;6| (|ts| $)
 (PROG (|p| |ip| |lip| |lp|)
  (RETURN
   (SEQ
    (LETT |lip| NIL |TSETCAT-;initials;SL;6|)
    (EXIT
     (COND
      ((SPADCALL |ts| (QREFELT $ 12)) |lip|)
      ((QUOTE T)
       (SEQ
        (LETT |lp| (SPADCALL |ts| (QREFELT $ 28)) |TSETCAT-;initials;SL;6|)
        (SEQ
         G190
         (COND
          ((NULL (COND ((NULL |lp|) (QUOTE NIL)) ((QUOTE T) (QUOTE T))))
            (GO G191)))
         (SEQ
          (LETT |p| (|SPADfirst| |lp|) |TSETCAT-;initials;SL;6|)
          (COND
           ((NULL
             (SPADCALL
              (LETT |ip|
               (SPADCALL |p| (QREFELT $ 45))
               |TSETCAT-;initials;SL;6|)
              (QREFELT $ 34)))
            (LETT |lip|
             (CONS (SPADCALL |ip| (QREFELT $ 46)) |lip|)
             |TSETCAT-;initials;SL;6|)))
           (EXIT (LETT |lp| (CDR |lp|) |TSETCAT-;initials;SL;6|)))
          NIL
          (GO G190)
          G191
```

```
         (EXIT NIL))
        (EXIT (SPADCALL |lip| (QREFELT $ 47))))))))))))

(DEFUN |TSETCAT-;degree;SNni;7| (|ts| $)
 (PROG (|lp| |d|)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |ts| (QREFELT $ 12)) 0)
     ((QUOTE T)
      (SEQ
       (LETT |lp| (SPADCALL |ts| (QREFELT $ 28)) |TSETCAT-;degree;SNni;7|)
       (LETT |d|
        (SPADCALL (|SPADfirst| |lp|) (QREFELT $ 50))
        |TSETCAT-;degree;SNni;7|)
       (SEQ
        G190
        (COND
         ((NULL
           (COND
            ((NULL
              (LETT |lp| (CDR |lp|) |TSETCAT-;degree;SNni;7|)) (QUOTE NIL))
             ((QUOTE T) (QUOTE T))))
          (GO G191)))
        (SEQ
         (EXIT
          (LETT |d|
           (* |d| (SPADCALL (|SPADfirst| |lp|) (QREFELT $ 50)))
           |TSETCAT-;degree;SNni;7|)))
        NIL
        (GO G190)
        G191
        (EXIT NIL))
       (EXIT |d|)))))))))

(DEFUN |TSETCAT-;quasiComponent;SR;8| (|ts| $)
 (CONS (SPADCALL |ts| (QREFELT $ 28)) (SPADCALL |ts| (QREFELT $ 52))))

(DEFUN |TSETCAT-;normalized?;PSB;9| (|p| |ts| $)
 (SPADCALL |p| (SPADCALL |ts| (QREFELT $ 28)) (QREFELT $ 56)))

(DEFUN |TSETCAT-;stronglyReduced?;PSB;10| (|p| |ts| $)
 (SPADCALL |p| (SPADCALL |ts| (QREFELT $ 28)) (QREFELT $ 58)))

(DEFUN |TSETCAT-;headReduced?;PSB;11| (|p| |ts| $)
 (SPADCALL (SPADCALL |p| (QREFELT $ 60)) |ts| (QREFELT $ 61)))

(DEFUN |TSETCAT-;initiallyReduced?;PSB;12| (|p| |ts| $)
 (PROG (|lp| |red|)
  (RETURN
```

```
(SEQ
 (LETT |lp|
  (SPADCALL |ts| (QREFELT $ 28)) |TSETCAT-;initiallyReduced?;PSB;12|)
 (LETT |red| (QUOTE T) |TSETCAT-;initiallyReduced?;PSB;12|)
 (SEQ
  G190
  (COND
   ((NULL
     (COND
      ((OR (NULL |lp|) (SPADCALL |p| (QREFELT $ 34))) (QUOTE NIL))
      ((QUOTE T) |red|)))
    (GO G191)))
   (SEQ
    (SEQ
     G190
     (COND
      ((NULL
       (COND
        ((NULL |lp|) (QUOTE NIL))
        ((QUOTE T)
         (SPADCALL
          (SPADCALL |p| (QREFELT $ 23))
          (SPADCALL (|SPADfirst| |lp|) (QREFELT $ 23))
          (QREFELT $ 63)))))
       (GO G191)))
     (SEQ (EXIT (LETT |lp| (CDR |lp|) |TSETCAT-;initiallyReduced?;PSB;12|)))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT
     (COND
      ((NULL (NULL |lp|))
       (COND
        ((SPADCALL
          (SPADCALL (|SPADfirst| |lp|) (QREFELT $ 23))
          (SPADCALL |p| (QREFELT $ 23))
          (QREFELT $ 64))
         (COND
          ((SPADCALL |p| (|SPADfirst| |lp|) (QREFELT $ 65))
           (SEQ
            (LETT |lp| (CDR |lp|) |TSETCAT-;initiallyReduced?;PSB;12|)
            (EXIT
             (LETT |p|
              (SPADCALL |p| (QREFELT $ 45))
              |TSETCAT-;initiallyReduced?;PSB;12|))))
          ((QUOTE T)
           (LETT |red| (QUOTE NIL) |TSETCAT-;initiallyReduced?;PSB;12|))))
        ((QUOTE T)
         (LETT |p|
```

```
            (SPADCALL |p| (QREFELT $ 45))
            |TSETCAT-;initiallyReduced?;PSB;12|)))))))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT |red|)))))

(DEFUN |TSETCAT-;reduce;PSMMP;13| (|p| |ts| |redOp| |redOp?| $)
 (PROG (|ts0| #0=#:G1572 |reductor| #1=#:G1575)
  (RETURN
   (SEQ
    (COND
     ((OR (SPADCALL |ts| (QREFELT $ 12)) (SPADCALL |p| (QREFELT $ 34))) |p|)
     ((QUOTE T)
      (SEQ
       (LETT |ts0| |ts| |TSETCAT-;reduce;PSMMP;13|)
       (SEQ
        G190
        (COND
         ((NULL
          (COND
           ((OR
             (SPADCALL |ts| (QREFELT $ 12))
             (SPADCALL |p| (QREFELT $ 34)))
            (QUOTE NIL))
           ((QUOTE T) (QUOTE T))))
          (GO G191)))
        (SEQ
         (LETT |reductor|
          (PROG2
           (LETT #0# (SPADCALL |ts| (QREFELT $ 14)) |TSETCAT-;reduce;PSMMP;13|)
           (QCDR #0#)
           (|check-union| (QEQCAR #0# 0) (QREFELT $ 10) #0#))
          |TSETCAT-;reduce;PSMMP;13|)
         (LETT |ts|
          (PROG2
           (LETT #1# (SPADCALL |ts| (QREFELT $ 17)) |TSETCAT-;reduce;PSMMP;13|)
           (QCDR #1#)
           (|check-union| (QEQCAR #1# 0) (QREFELT $ 6) #1#))
          |TSETCAT-;reduce;PSMMP;13|)
         (EXIT
          (COND
           ((NULL (SPADCALL |p| |reductor| |redOp?|))
            (SEQ
             (LETT |p|
              (SPADCALL |p| |reductor| |redOp|) |TSETCAT-;reduce;PSMMP;13|)
             (EXIT (LETT |ts| |ts0| |TSETCAT-;reduce;PSMMP;13|)))))))
        NIL
        (GO G190)
```

```
        G191
        (EXIT NIL))
       (EXIT |p|)))))))))

(DEFUN |TSETCAT-;rewriteSetWithReduction;LSMML;14|
    (|lp| |ts| |redOp| |redOp?| $)
 (PROG (|p| |rs|)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |ts| (QREFELT $ 69)) |lp|)
     ((QUOTE T)
      (SEQ
       (LETT |lp|
        (SPADCALL (ELT $ 31) |lp| (QREFELT $ 33))
        |TSETCAT-;rewriteSetWithReduction;LSMML;14|)
       (EXIT
        (COND
         ((NULL |lp|) |lp|)
         ((SPADCALL (ELT $ 34) |lp| (QREFELT $ 35))
           (LIST (|spadConstant| $ 70)))
         ((QUOTE T)
          (SEQ
           (LETT |rs| NIL |TSETCAT-;rewriteSetWithReduction;LSMML;14|)
           (SEQ
            G190
            (COND
             ((NULL (COND ((NULL |lp|) (QUOTE NIL)) ((QUOTE T) (QUOTE T))))
              (GO G191)))
            (SEQ
             (LETT |p|
              (|SPADfirst| |lp|)
              |TSETCAT-;rewriteSetWithReduction;LSMML;14|)
             (LETT |lp| (CDR |lp|) |TSETCAT-;rewriteSetWithReduction;LSMML;14|)
             (LETT |p|
              (SPADCALL
               (SPADCALL |p| |ts| |redOp| |redOp?| (QREFELT $ 71))
               (QREFELT $ 46))
              |TSETCAT-;rewriteSetWithReduction;LSMML;14|)
             (EXIT
              (COND
               ((NULL (SPADCALL |p| (QREFELT $ 31)))
                 (COND
                  ((SPADCALL |p| (QREFELT $ 34))
                   (SEQ
                    (LETT |lp| NIL |TSETCAT-;rewriteSetWithReduction;LSMML;14|)
                    (EXIT
                     (LETT |rs|
                      (LIST (|spadConstant| $ 70))
                      |TSETCAT-;rewriteSetWithReduction;LSMML;14|))))
```

```
                ((QUOTE T)
                 (LETT |rs|
                  (CONS |p| |rs|)
                  |TSETCAT-;rewriteSetWithReduction;LSMML;14|))))))))
           NIL
           (GO G190)
           G191
           (EXIT NIL))
          (EXIT (SPADCALL |rs| (QREFELT $ 47)))))))))))))))

(DEFUN |TSETCAT-;stronglyReduce;PSP;15| (|p| |ts| $)
 (SPADCALL |p| |ts| (ELT $ 73) (ELT $ 65) (QREFELT $ 71)))

(DEFUN |TSETCAT-;headReduce;PSP;16| (|p| |ts| $)
 (SPADCALL |p| |ts| (ELT $ 75) (ELT $ 76) (QREFELT $ 71)))

(DEFUN |TSETCAT-;initiallyReduce;PSP;17| (|p| |ts| $)
 (SPADCALL |p| |ts| (ELT $ 78) (ELT $ 79) (QREFELT $ 71)))

(DEFUN |TSETCAT-;removeZero;PSP;18| (|p| |ts| $)
 (PROG (|v| |tsv-| #0=#:G1599 #1=#:G1608 |q|)
  (RETURN
   (SEQ
    (EXIT
     (COND
      ((OR (SPADCALL |p| (QREFELT $ 34)) (SPADCALL |ts| (QREFELT $ 12))) |p|)
      ((QUOTE T)
       (SEQ
        (LETT |v| (SPADCALL |p| (QREFELT $ 23)) |TSETCAT-;removeZero;PSP;18|)
        (LETT |tsv-|
         (SPADCALL |ts| |v| (QREFELT $ 81))
         |TSETCAT-;removeZero;PSP;18|)
        (COND
         ((SPADCALL |v| |ts| (QREFELT $ 82))
          (SEQ
           (LETT |q|
            (SPADCALL |p|
             (PROG2
              (LETT #0#
               (SPADCALL |ts| |v| (QREFELT $ 83))
               |TSETCAT-;removeZero;PSP;18|)
              (QCDR #0#)
              (|check-union| (QEQCAR #0# 0) (QREFELT $ 10) #0#))
             (QREFELT $ 73))
            |TSETCAT-;removeZero;PSP;18|)
           (EXIT
            (COND
             ((SPADCALL |q| (QREFELT $ 31))
               (PROGN (LETT #1# |q| |TSETCAT-;removeZero;PSP;18|) (GO #1#)))
              ((SPADCALL (SPADCALL |q| |tsv-| (QREFELT $ 84)) (QREFELT $ 31))
```

```
                (PROGN
                 (LETT #1#
                  (|spadConstant| $ 85)
                  |TSETCAT-;removeZero;PSP;18|) (GO #1#))))))))
        (EXIT
         (COND
          ((SPADCALL |tsv-| (QREFELT $ 12)) |p|)
          ((QUOTE T)
           (SEQ
            (LETT |q| (|spadConstant| $ 85) |TSETCAT-;removeZero;PSP;18|)
            (SEQ
             G190
             (COND
              ((NULL
                (SPADCALL (SPADCALL |p| |v| (QREFELT $ 86)) (QREFELT $ 88)))
               (GO G191)))
             (SEQ
              (LETT |q|
               (SPADCALL
                (SPADCALL
                 (SPADCALL
                  (SPADCALL |p| (QREFELT $ 45))
                   |tsv-| (QREFELT $ 84))
                  (SPADCALL |p| (QREFELT $ 89))
                 (QREFELT $ 90))
                |q|
                (QREFELT $ 91))
               |TSETCAT-;removeZero;PSP;18|)
              (EXIT
               (LETT |p|
                (SPADCALL |p| (QREFELT $ 92))
                |TSETCAT-;removeZero;PSP;18|)))
             NIL
             (GO G190)
             G191
             (EXIT NIL))
            (EXIT
             (SPADCALL |q|
              (SPADCALL |p| |tsv-| (QREFELT $ 84))
              (QREFELT $ 91)))))))))
    #1#
    (EXIT #1#)))))

(DEFUN |TSETCAT-;reduceByQuasiMonic;PSP;19| (|p| |ts| $)
 (COND
  ((OR (SPADCALL |p| (QREFELT $ 34)) (SPADCALL |ts| (QREFELT $ 12))) |p|)
  ((QUOTE T)
    (QVELT (SPADCALL |p| (SPADCALL |ts| (QREFELT $ 94)) (QREFELT $ 96)) 1))))

(DEFUN |TSETCAT-;autoReduced?;SMB;20| (|ts| |redOp?| $)
```

```
 (PROG (|p| |lp|)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |ts| (QREFELT $ 12)) (QUOTE T))
     ((QUOTE T)
      (SEQ
       (LETT |lp|
        (SPADCALL |ts| (QREFELT $ 28)) |TSETCAT-;autoReduced?;SMB;20|)
       (LETT |p| (|SPADfirst| |lp|) |TSETCAT-;autoReduced?;SMB;20|)
       (LETT |lp| (CDR |lp|) |TSETCAT-;autoReduced?;SMB;20|)
       (SEQ
        G190
        (COND
         ((NULL
           (COND
            ((NULL |lp|) (QUOTE NIL))
            ((QUOTE T) (SPADCALL |p| |lp| |redOp?|))))
          (GO G191)))
        (SEQ
         (LETT |p| (|SPADfirst| |lp|) |TSETCAT-;autoReduced?;SMB;20|)
         (EXIT (LETT |lp| (CDR |lp|) |TSETCAT-;autoReduced?;SMB;20|)))
        NIL
        (GO G190)
        G191
        (EXIT NIL))
       (EXIT (NULL |lp|)))))))))

(DEFUN |TSETCAT-;stronglyReduced?;SB;21| (|ts| $)
 (SPADCALL |ts| (ELT $ 58) (QREFELT $ 100)))

(DEFUN |TSETCAT-;normalized?;SB;22| (|ts| $)
 (SPADCALL |ts| (ELT $ 56) (QREFELT $ 100)))

(DEFUN |TSETCAT-;headReduced?;SB;23| (|ts| $)
 (SPADCALL |ts| (ELT $ 103) (QREFELT $ 100)))

(DEFUN |TSETCAT-;initiallyReduced?;SB;24| (|ts| $)
 (SPADCALL |ts| (ELT $ 105) (QREFELT $ 100)))

(DEFUN |TSETCAT-;mvar;SV;25| (|ts| $)
 (PROG (#0=#:G1627)
  (RETURN
   (COND
    ((SPADCALL |ts| (QREFELT $ 12))
      (|error| "Error from TSETCAT in mvar : #1 is empty"))
    ((QUOTE T)
      (SPADCALL
       (PROG2
        (LETT #0# (SPADCALL |ts| (QREFELT $ 14)) |TSETCAT-;mvar;SV;25|)
```

```
        (QCDR #0#)
        (|check-union| (QEQCAR #0# 0) (QREFELT $ 10) #0#))
       (QREFELT $ 23)))))))

(DEFUN |TSETCAT-;first;SU;26| (|ts| $)
 (PROG (|lp|)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |ts| (QREFELT $ 12)) (CONS 1 "failed"))
     ((QUOTE T)
      (SEQ
       (LETT |lp|
        (SPADCALL (ELT $ 22) (SPADCALL |ts| (QREFELT $ 28)) (QREFELT $ 36))
        |TSETCAT-;first;SU;26|)
       (EXIT (CONS 0 (|SPADfirst| |lp|)))))))))))

(DEFUN |TSETCAT-;last;SU;27| (|ts| $)
 (PROG (|lp|)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |ts| (QREFELT $ 12)) (CONS 1 "failed"))
     ((QUOTE T)
      (SEQ
       (LETT |lp|
        (SPADCALL (ELT $ 21) (SPADCALL |ts| (QREFELT $ 28)) (QREFELT $ 36))
        |TSETCAT-;last;SU;27|)
       (EXIT (CONS 0 (|SPADfirst| |lp|)))))))))))

(DEFUN |TSETCAT-;rest;SU;28| (|ts| $)
 (PROG (|lp|)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |ts| (QREFELT $ 12)) (CONS 1 "failed"))
     ((QUOTE T)
      (SEQ
       (LETT |lp|
        (SPADCALL (ELT $ 22) (SPADCALL |ts| (QREFELT $ 28)) (QREFELT $ 36))
        |TSETCAT-;rest;SU;28|)
       (EXIT (CONS 0 (SPADCALL (CDR |lp|) (QREFELT $ 110)))))))))))

(DEFUN |TSETCAT-;coerce;SL;29| (|ts| $)
 (SPADCALL (ELT $ 22) (SPADCALL |ts| (QREFELT $ 28)) (QREFELT $ 36)))

(DEFUN |TSETCAT-;algebraicVariables;SL;30| (|ts| $)
 (PROG (#0=#:G1652 |p| #1=#:G1653)
  (RETURN
   (SEQ
```

```
    (PROGN
     (LETT #0# NIL |TSETCAT-;algebraicVariables;SL;30|)
     (SEQ
      (LETT |p| NIL |TSETCAT-;algebraicVariables;SL;30|)
      (LETT #1#
       (SPADCALL |ts| (QREFELT $ 28)) |TSETCAT-;algebraicVariables;SL;30|)
      G190
      (COND
       ((OR
         (ATOM #1#)
         (PROGN (LETT |p| (CAR #1#) |TSETCAT-;algebraicVariables;SL;30|) NIL))
        (GO G191)))
       (SEQ
        (EXIT
         (LETT #0#
          (CONS (SPADCALL |p| (QREFELT $ 23)) #0#)
          |TSETCAT-;algebraicVariables;SL;30|)))
       (LETT #1# (CDR #1#) |TSETCAT-;algebraicVariables;SL;30|)
       (GO G190)
       G191
       (EXIT (NREVERSE0 #0#))))))))

(DEFUN |TSETCAT-;algebraic?;VSB;31| (|v| |ts| $)
 (SPADCALL |v| (SPADCALL |ts| (QREFELT $ 115)) (QREFELT $ 116)))

(DEFUN |TSETCAT-;select;SVU;32| (|ts| |v| $)
 (PROG (|lp|)
  (RETURN
   (SEQ
    (LETT |lp|
     (SPADCALL (ELT $ 22) (SPADCALL |ts| (QREFELT $ 28)) (QREFELT $ 36))
     |TSETCAT-;select;SVU;32|)
    (SEQ
     G190
     (COND
      ((NULL
        (COND
         ((OR
           (NULL |lp|)
           (SPADCALL |v|
            (SPADCALL (|SPADfirst| |lp|) (QREFELT $ 23))
            (QREFELT $ 64)))
          (QUOTE NIL))
         ((QUOTE T) (QUOTE T))))
       (GO G191)))
     (SEQ (EXIT (LETT |lp| (CDR |lp|) |TSETCAT-;select;SVU;32|)))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
```

```
    (EXIT
     (COND
      ((NULL |lp|) (CONS 1 "failed"))
      ((QUOTE T) (CONS 0 (|SPADfirst| |lp|)))))))))

(DEFUN |TSETCAT-;collectQuasiMonic;2S;33| (|ts| $)
 (PROG (|newlp| |lp|)
  (RETURN
   (SEQ
    (LETT |lp|
     (SPADCALL |ts| (QREFELT $ 28))
     |TSETCAT-;collectQuasiMonic;2S;33|)
    (LETT |newlp| NIL |TSETCAT-;collectQuasiMonic;2S;33|)
    (SEQ
     G190
     (COND
      ((NULL (COND ((NULL |lp|) (QUOTE NIL)) ((QUOTE T) (QUOTE T))))
        (GO G191)))
     (SEQ
      (COND
       ((SPADCALL (SPADCALL (|SPADfirst| |lp|) (QREFELT $ 45)) (QREFELT $ 34))
        (LETT |newlp|
         (CONS (|SPADfirst| |lp|) |newlp|)
         |TSETCAT-;collectQuasiMonic;2S;33|)))
      (EXIT (LETT |lp| (CDR |lp|) |TSETCAT-;collectQuasiMonic;2S;33|)))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT (SPADCALL |newlp| (QREFELT $ 110)))))))

(DEFUN |TSETCAT-;collectUnder;SVS;34| (|ts| |v| $)
 (PROG (|lp|)
  (RETURN
   (SEQ
    (LETT |lp|
     (SPADCALL (ELT $ 22) (SPADCALL |ts| (QREFELT $ 28)) (QREFELT $ 36))
     |TSETCAT-;collectUnder;SVS;34|)
    (SEQ
     G190
     (COND
      ((NULL
        (COND
         ((OR
           (NULL |lp|)
           (SPADCALL
            (SPADCALL (|SPADfirst| |lp|) (QREFELT $ 23))
            |v|
            (QREFELT $ 63)))
          (QUOTE NIL))
```

```
         ((QUOTE T) (QUOTE T))))
       (GO G191)))
     (SEQ (EXIT (LETT |lp| (CDR |lp|) |TSETCAT-;collectUnder;SVS;34|)))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT (SPADCALL |lp| (QREFELT $ 110)))))))

(DEFUN |TSETCAT-;collectUpper;SVS;35| (|ts| |v| $)
 (PROG (|lp2| |lp1|)
  (RETURN
   (SEQ
    (LETT |lp1|
     (SPADCALL (ELT $ 22) (SPADCALL |ts| (QREFELT $ 28)) (QREFELT $ 36))
     |TSETCAT-;collectUpper;SVS;35|)
    (LETT |lp2| NIL |TSETCAT-;collectUpper;SVS;35|)
    (SEQ
     G190
     (COND
      ((NULL
        (COND
         ((NULL |lp1|) (QUOTE NIL))
         ((QUOTE T)
          (SPADCALL |v|
           (SPADCALL (|SPADfirst| |lp1|) (QREFELT $ 23))
           (QREFELT $ 63)))))
       (GO G191)))
     (SEQ
      (LETT |lp2|
       (CONS (|SPADfirst| |lp1|) |lp2|)
       |TSETCAT-;collectUpper;SVS;35|)
      (EXIT (LETT |lp1| (CDR |lp1|) |TSETCAT-;collectUpper;SVS;35|)))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT (SPADCALL (REVERSE |lp2|) (QREFELT $ 110)))))))

(DEFUN |TSETCAT-;construct;LS;36| (|lp| $)
 (PROG (|rif|)
  (RETURN
   (SEQ
    (LETT |rif| (SPADCALL |lp| (QREFELT $ 122)) |TSETCAT-;construct;LS;36|)
    (EXIT
     (COND
      ((QEQCAR |rif| 0) (QCDR |rif|))
      ((QUOTE T)
       (|error| "in construct : LP -> $ from TSETCAT : bad arg")))))))))
```

```
(DEFUN |TSETCAT-;retractIfCan;LU;37| (|lp| $)
 (PROG (|rif|)
  (RETURN
   (SEQ
    (COND
     ((NULL |lp|) (CONS 0 (SPADCALL (QREFELT $ 37))))
     ((QUOTE T)
      (SEQ
       (LETT |lp|
        (SPADCALL (ELT $ 22) |lp| (QREFELT $ 36))
        |TSETCAT-;retractIfCan;LU;37|)
       (LETT |rif|
        (SPADCALL (CDR |lp|) (QREFELT $ 122))
        |TSETCAT-;retractIfCan;LU;37|)
       (EXIT
        (COND
         ((QEQCAR |rif| 0)
           (SPADCALL (QCDR |rif|) (|SPADfirst| |lp|) (QREFELT $ 124)))
          ((QUOTE T)
            (|error| "in retractIfCan : LP -> ... from TSETCAT : bad arg")
  ))))))))))

(DEFUN |TSETCAT-;extend;SPS;38| (|ts| |p| $)
 (PROG (|eif|)
  (RETURN
   (SEQ
    (LETT |eif| (SPADCALL |ts| |p| (QREFELT $ 124)) |TSETCAT-;extend;SPS;38|)
    (EXIT
     (COND
      ((QEQCAR |eif| 0) (QCDR |eif|))
      ((QUOTE T)
       (|error| "in extend : ($,P) -> $ from TSETCAT : bad ars")))))))))

(DEFUN |TSETCAT-;coHeight;SNni;39| (|ts| $)
 (PROG (|n| |m| #0=#:G1696)
  (RETURN
   (SEQ
    (LETT |n| (SPADCALL (QREFELT $ 127)) |TSETCAT-;coHeight;SNni;39|)
    (LETT |m|
     (LENGTH (SPADCALL |ts| (QREFELT $ 28)))
     |TSETCAT-;coHeight;SNni;39|)
    (EXIT
     (PROG2
      (LETT #0# (SPADCALL |n| |m| (QREFELT $ 128)) |TSETCAT-;coHeight;SNni;39|)
      (QCDR #0#)
      (|check-union| (QEQCAR #0# 0) (|NonNegativeInteger|) #0#)))))))

(DEFUN |TriangularSetCategory&| (|#1| |#2| |#3| |#4| |#5|)
 (PROG (DV$1 DV$2 DV$3 DV$4 DV$5 |dv$| $ |pv$|)
  (RETURN
```

```
   (PROGN
    (LETT DV$1 (|devaluate| |#1|) . #0=(|TriangularSetCategory&|))
    (LETT DV$2 (|devaluate| |#2|) . #0#)
    (LETT DV$3 (|devaluate| |#3|) . #0#)
    (LETT DV$4 (|devaluate| |#4|) . #0#)
    (LETT DV$5 (|devaluate| |#5|) . #0#)
    (LETT |dv$|
     (LIST (QUOTE |TriangularSetCategory&|) DV$1 DV$2 DV$3 DV$4 DV$5) . #0#)
    (LETT $ (make-array 131) . #0#)
    (QSETREFV $ 0 |dv$|)
    (QSETREFV $ 3
     (LETT |pv$|
      (|buildPredVector| 0 0 (LIST (|HasCategory| |#4| (QUOTE (|Finite|)))))
      . #0#))
    (|stuffDomainSlots| $)
    (QSETREFV $ 6 |#1|)
    (QSETREFV $ 7 |#2|)
    (QSETREFV $ 8 |#3|)
    (QSETREFV $ 9 |#4|)
    (QSETREFV $ 10 |#5|)
    (COND
     ((|testBitVector| |pv$| 1)
       (QSETREFV $ 129
        (CONS (|dispatchFunction| |TSETCAT-;coHeight;SNni;39|) $))))
     $))))

(setf (get
 (QUOTE |TriangularSetCategory&|)
 (QUOTE |infovec|))
 (LIST (QUOTE
  #(NIL NIL NIL NIL NIL NIL (|local| |#1|) (|local| |#2|) (|local| |#3|)
    (|local| |#4|) (|local| |#5|) (|Boolean|) (0 . |empty?|)
    (|Union| 10 (QUOTE "failed")) (5 . |first|) (10 . =)
    (|Union| $ (QUOTE "failed")) (16 . |rest|) (21 . =) |TSETCAT-;=;2SB;1|
    (27 . |last|) (32 . |infRittWu?|) (38 . |supRittWu?|) (44 . |mvar|)
    (49 . |collectUpper|) (55 . |infRittWu?|) |TSETCAT-;infRittWu?;2SB;2|
    (|List| 10) (61 . |members|) (|Mapping| 11 10 10)
    |TSETCAT-;reduced?;PSMB;3| (66 . |zero?|) (|Mapping| 11 10)
    (71 . |remove|) (77 . |ground?|) (82 . |any?|) (88 . |sort|)
    (94 . |empty|) (98 . |extend|) (104 . |reduced?|)
    (|Record| (|:| |bas| $) (|:| |top| 27))
    (|Union| 40 (QUOTE "failed")) |TSETCAT-;basicSet;LMU;4|
    (111 . |concat|) |TSETCAT-;basicSet;LMMU;5| (117 . |init|)
    (122 . |primPartElseUnitCanonical|) (127 . |removeDuplicates|)
    |TSETCAT-;initials;SL;6| (|NonNegativeInteger|) (132 . |mdeg|)
    |TSETCAT-;degree;SNni;7| (137 . |initials|)
    (|Record| (|:| |close| 27) (|:| |open| 27))
    |TSETCAT-;quasiComponent;SR;8| (|List| $) (142 . |normalized?|)
    |TSETCAT-;normalized?;PSB;9| (148 . |reduced?|)
    |TSETCAT-;stronglyReduced?;PSB;10| (154 . |head|)
```

```
   (159 . |stronglyReduced?|) |TSETCAT-;headReduced?;PSB;11|
   (165 . <) (171 . =) (177 . |reduced?|)
   |TSETCAT-;initiallyReduced?;PSB;12| (|Mapping| 10 10 10)
   |TSETCAT-;reduce;PSMMP;13| (183 . |trivialIdeal?|) (188 . |One|)
   (192 . |reduce|) |TSETCAT-;rewriteSetWithReduction;LSMML;14|
   (200 . |lazyPrem|) |TSETCAT-;stronglyReduce;PSP;15|
   (206 . |headReduce|) (212 . |headReduced?|)
   |TSETCAT-;headReduce;PSP;16| (218 . |initiallyReduce|)
   (224 . |initiallyReduced?|) |TSETCAT-;initiallyReduce;PSP;17|
   (230 . |collectUnder|) (236 . |algebraic?|) (242 . |select|)
   (248 . |removeZero|) (254 . |Zero|) (258 . |degree|) (|Integer|)
   (264 . |positive?|) (269 . |mainMonomial|) (274 . *) (280 . +)
   (286 . |tail|) |TSETCAT-;removeZero;PSP;18| (291 . |collectQuasiMonic|)
   (|Record| (|:| |rnum| 7) (|:| |polnum| 10) (|:| |den| 7))
   (296 . |remainder|) |TSETCAT-;reduceByQuasiMonic;PSP;19|
   (|Mapping| 11 10 27) |TSETCAT-;autoReduced?;SMB;20|
   (302 . |autoReduced?|) |TSETCAT-;stronglyReduced?;SB;21|
   |TSETCAT-;normalized?;SB;22| (308 . |headReduced?|)
   |TSETCAT-;headReduced?;SB;23| (314 . |initiallyReduced?|)
   |TSETCAT-;initiallyReduced?;SB;24| |TSETCAT-;mvar;SV;25|
   |TSETCAT-;first;SU;26| |TSETCAT-;last;SU;27| (320 . |construct|)
   |TSETCAT-;rest;SU;28| |TSETCAT-;coerce;SL;29| (|List| 9)
   |TSETCAT-;algebraicVariables;SL;30| (325 . |algebraicVariables|)
   (330 . |member?|) |TSETCAT-;algebraic?;VSB;31|
   |TSETCAT-;select;SVU;32| |TSETCAT-;collectQuasiMonic;2S;33|
   |TSETCAT-;collectUnder;SVS;34| |TSETCAT-;collectUpper;SVS;35|
   (336 . |retractIfCan|) |TSETCAT-;construct;LS;36|
   (341 . |extendIfCan|) |TSETCAT-;retractIfCan;LU;37|
   |TSETCAT-;extend;SPS;38| (347 . |size|) (351 . |subtractIfCan|)
   (357 . |coHeight|) (|OutputForm|)))
(QUOTE #(|stronglyReduced?| 362 |stronglyReduce| 373 |select| 379
   |rewriteSetWithReduction| 385 |retractIfCan| 393 |rest| 398 |removeZero|
   403 |reduced?| 409 |reduceByQuasiMonic| 416 |reduce| 422
   |quasiComponent| 430 |normalized?| 435 |mvar| 446 |last| 451
   |initials| 456 |initiallyReduced?| 461 |initiallyReduce| 472
   |infRittWu?| 478 |headReduced?| 484 |headReduce| 495 |first| 501
   |extend| 506 |degree| 512 |construct| 517 |collectUpper| 522
   |collectUnder| 528 |collectQuasiMonic| 534 |coerce| 539 |coHeight|
   544 |basicSet| 549 |autoReduced?| 562 |algebraicVariables| 568
   |algebraic?| 573 = 579))
(QUOTE NIL)
(CONS
 (|makeByteWordVec2| 1 (QUOTE NIL))
 (CONS (QUOTE #()) (CONS (QUOTE #())
  (|makeByteWordVec2| 129 (QUOTE (1 6 11 0 12 1 6 13 0 14 2 10 11 0 0 15
    1 6 16 0 17 2 6 11 0 0 18 1 6 13 0 20 2 10 11 0 0 21 2 10 11 0 0 22
    1 10 9 0 23 2 6 0 0 9 24 2 6 11 0 0 25 1 6 27 0 28 1 10 11 0 31 2 27
    0 32 0 33 1 10 11 0 34 2 27 11 32 0 35 2 27 0 29 0 36 0 6 0 37 2 6 0
    0 10 38 3 6 11 10 0 29 39 2 27 0 0 0 43 1 10 0 0 45 1 10 0 0 46 1 27
    0 0 47 1 10 49 0 50 1 6 27 0 52 2 10 11 0 55 56 2 10 11 0 55 58 1 10
```

```
    0 0 60 2 6 11 10 0 61 2 9 11 0 0 63 2 9 11 0 0 64 2 10 11 0 0 65 1 6
    11 0 69 0 10 0 70 4 6 10 10 0 67 29 71 2 10 0 0 0 73 2 10 0 0 0 75 2
    10 11 0 0 76 2 10 0 0 0 78 2 10 11 0 0 79 2 6 0 0 9 81 2 6 11 9 0 82
    2 6 13 0 9 83 2 6 10 10 0 84 0 10 0 85 2 10 49 0 9 86 1 87 11 0 88 1
    10 0 0 89 2 10 0 0 0 90 2 10 0 0 0 91 1 10 0 0 92 1 6 0 0 94 2 6 95
    10 0 96 2 6 11 0 98 100 2 10 11 0 55 103 2 10 11 0 55 105 1 6 0 27
    110 1 6 113 0 115 2 113 11 9 0 116 1 6 16 27 122 2 6 16 0 10 124 0
    9 49 127 2 49 16 0 0 128 1 0 49 0 129 1 0 11 0 101 2 0 11 10 0 59 2
    0 10 10 0 74 2 0 13 0 9 118 4 0 27 27 0 67 29 72 1 0 16 27 125 1 0
    16 0 111 2 0 10 10 0 93 3 0 11 10 0 29 30 2 0 10 10 0 97 4 0 10 10 0
    67 29 68 1 0 53 0 54 1 0 11 0 102 2 0 11 10 0 57 1 0 9 0 107 1 0 13
    0 109 1 0 27 0 48 1 0 11 0 106 2 0 11 10 0 66 2 0 10 10 0 80 2 0 11
    0 0 26 1 0 11 0 104 2 0 11 10 0 62 2 0 10 10 0 77 1 0 13 0 108 2 0 0
    0 10 126 1 0 49 0 51 1 0 0 27 123 2 0 0 0 9 121 2 0 0 0 9 120 1 0 0
    0 119 1 0 27 0 112 1 0 49 0 129 3 0 41 27 32 29 44 2 0 41 27 29 42 2
    0 11 0 98 99 1 0 113 0 114 2 0 11 9 0 117 2 0 11 0 0 19))))))
  (QUOTE |lookupComplete|)))
```

———-

21.64 UFD.lsp BOOTSTRAP

UFD needs **GCDDOM** which needs **COMRING** which needs **RING** which needs **RNG** which needs **ABELGRP** which needs **CABMON** which needs **ABELMON** which needs **ABELSG** which needs **SETCAT** which needs **SINT** which needs **UFD**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **UFD** category which we can write into the **MID** directory. We compile the lisp code and copy the **UFD.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— **UFD.lsp BOOTSTRAP** —

```
(|/VERSIONCHECK| 2)

(SETQ |UniqueFactorizationDomain;AL| (QUOTE NIL))

(DEFUN |UniqueFactorizationDomain| NIL
  (LET (#:G83334)
    (COND
      (|UniqueFactorizationDomain;AL|)
      (T
        (SETQ
          |UniqueFactorizationDomain;AL|
          (|UniqueFactorizationDomain;|))))))
```

```
(DEFUN |UniqueFactorizationDomain;| NIL
  (PROG (#1=#:G83332)
    (RETURN
      (PROG1
        (LETT #1#
          (|Join|
            (|GcdDomain|)
            (|mkCategory|
              (QUOTE |domain|)
              (QUOTE (
                ((|prime?| ((|Boolean|) |$|)) T)
                ((|squareFree| ((|Factored| |$|) |$|)) T)
                ((|squareFreePart| (|$| |$|)) T)
                ((|factor| ((|Factored| |$|) |$|)) T)))
              NIL
              (QUOTE ((|Factored| |$|) (|Boolean|)))
              NIL))
          |UniqueFactorizationDomain|)
        (SETELT #1# 0 (QUOTE (|UniqueFactorizationDomain|)))))))

(setf (get (QUOTE |UniqueFactorizationDomain|) (QUOTE NILADIC)) T)
```

———-

21.65 UFD-.lsp BOOTSTRAP

UFD- needs **UFD**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **UFD-** category which we can write into the **MID** directory. We compile the lisp code and copy the **UFD-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— UFD-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |UFD-;squareFreePart;2S;1| (|x| |$|)
  (PROG (|s| |f| #1=#:G83349 #2=#:G83347 #3=#:G83345 #4=#:G83346)
    (RETURN
      (SEQ
        (SPADCALL
          (SPADCALL
            (LETT |s|
              (SPADCALL |x| (QREFELT |$| 8))
```

```
            |UFD-;squareFreePart;2S;1|)
          (QREFELT |$| 10))
        (PROGN
          (LETT #4# NIL |UFD-;squareFreePart;2S;1|)
          (SEQ
            (LETT |f| NIL |UFD-;squareFreePart;2S;1|)
            (LETT #1#
              (SPADCALL |s| (QREFELT |$| 13))
              |UFD-;squareFreePart;2S;1|)
            G190
            (COND
              ((OR
                 (ATOM #1#)
                 (PROGN
                   (LETT |f| (CAR #1#) |UFD-;squareFreePart;2S;1|)
                   NIL))
               (GO G191)))
            (SEQ
              (EXIT
                (PROGN
                  (LETT #2# (QCAR |f|) |UFD-;squareFreePart;2S;1|)
                  (COND
                    (#4#
                      (LETT #3#
                        (SPADCALL #3# #2# (QREFELT |$| 14))
                        |UFD-;squareFreePart;2S;1|))
                    ((QUOTE T)
                      (PROGN
                        (LETT #3# #2# |UFD-;squareFreePart;2S;1|)
                        (LETT #4# (QUOTE T)
                          |UFD-;squareFreePart;2S;1|)))))))
            (LETT #1# (CDR #1#) |UFD-;squareFreePart;2S;1|)
            (GO G190)
            G191
            (EXIT NIL))
          (COND
            (#4# #3#)
            ((QUOTE T) (|spadConstant| |$| 15))))
      (QREFELT |$| 14))))))

(DEFUN |UFD-;prime?;SB;2| (|x| |$|)
  (EQL
    (LENGTH (SPADCALL (SPADCALL |x| (QREFELT |$| 17)) (QREFELT |$| 21))) 1))

(DEFUN |UniqueFactorizationDomain&| (|#1|)
  (PROG (|DV$1| |dv$| |$| |pv$|)
    (RETURN
      (PROGN
        (LETT |DV$1| (|devaluate| |#1|) . #1=(|UniqueFactorizationDomain&|))
        (LETT |dv$| (LIST (QUOTE |UniqueFactorizationDomain&|) |DV$1|) . #1#)
```

```
        (LETT |$| (make-array 24) . #1#)
        (QSETREFV |$| 0 |dv$|)
        (QSETREFV |$| 3 (LETT |pv$| (|buildPredVector| 0 0 NIL) . #1#))
        (|stuffDomainSlots| |$|)
        (QSETREFV |$| 6 |#1|)
        |$|))))

(setf (get
  (QUOTE |UniqueFactorizationDomain&|)
  (QUOTE |infovec|))
  (LIST
    (QUOTE
      #(NIL NIL NIL NIL NIL NIL
        (|local| |#1|)
        (|Factored| |$|)
        (0 . |squareFree|)
        (|Factored| 6)
        (5 . |unit|)
        (|Record| (|:| |factor| 6) (|:| |exponent| (|Integer|)))
        (|List| 11)
        (10 . |factors|)
        (15 . |*|)
        (21 . |One|)
        |UFD-;squareFreePart;2S;1|
        (25 . |factor|)
        (|Union|
          (QUOTE "nil") (QUOTE "sqfr") (QUOTE "irred") (QUOTE "prime"))
        (|Record| (|:| |flg| 18) (|:| |fctr| 6) (|:| |xpnt| (|Integer|)))
        (|List| 19)
        (30 . |factorList|)
        (|Boolean|)
        |UFD-;prime?;SB;2|))
    (QUOTE #(|squareFreePart| 35 |prime?| 40))
    (QUOTE NIL)
    (CONS
      (|makeByteWordVec2| 1 (QUOTE NIL))
      (CONS
        (QUOTE #())
        (CONS
          (QUOTE #())
          (|makeByteWordVec2| 23
            (QUOTE
              (1 6 7 0 8 1 9 6 0 10 1 9 12 0 13 2 6 0 0 0 14 0 6 0 15 1 6 7
               0 17 1 9 20 0 21 1 0 0 0 16 1 0 22 0 23))))))
    (QUOTE |lookupComplete|)))
```

21.66 ULSCAT.lsp BOOTSTRAP

ULSCAT depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **ULSCAT** category which we can write into the **MID** directory. We compile the lisp code and copy the **ULSCAT.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— ULSCAT.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |UnivariateLaurentSeriesCategory;CAT| (QUOTE NIL))

(SETQ |UnivariateLaurentSeriesCategory;AL| (QUOTE NIL))

(DEFUN |UnivariateLaurentSeriesCategory| (#1=#:G83278)
 (LET (#2=#:G83279)
  (COND
   ((SETQ #2# (|assoc| (|devaluate| #1#) |UnivariateLaurentSeriesCategory;AL|))
     (CDR #2#))
   (T
    (SETQ |UnivariateLaurentSeriesCategory;AL|
     (|cons5|
      (CONS
       (|devaluate| #1#)
       (SETQ #2# (|UnivariateLaurentSeriesCategory;| #1#)))
      |UnivariateLaurentSeriesCategory;AL|))
     #2#))))

(DEFUN |UnivariateLaurentSeriesCategory;| (|t#1|)
 (PROG (#1=#:G83277)
  (RETURN
   (PROG1
    (LETT #1#
     (|sublisV|
      (PAIR (QUOTE (|t#1|)) (LIST (|devaluate| |t#1|)))
      (|sublisV|
       (PAIR (QUOTE (#2=#:G83276)) (LIST (QUOTE (|Integer|))))
       (COND
        (|UnivariateLaurentSeriesCategory;CAT|)
        ((QUOTE T)
         (LETT |UnivariateLaurentSeriesCategory;CAT|
          (|Join|
           (|UnivariatePowerSeriesCategory| (QUOTE |t#1|) (QUOTE #2#))
           (|mkCategory| (QUOTE |domain|)
            (QUOTE (
```

```
          ((|series|
            (|$|
             (|Stream| (|Record| (|:| |k| (|Integer|)) (|:| |c| |t#1|)))))
             T)
          ((|multiplyCoefficients| (|$| (|Mapping| |t#1| (|Integer|)) |$|))
             T)
          ((|rationalFunction|
            ((|Fraction| (|Polynomial| |t#1|)) |$| (|Integer|)))
             (|has| |t#1| (|IntegralDomain|)))
          ((|rationalFunction|
            ((|Fraction| (|Polynomial| |t#1|)) |$| (|Integer|) (|Integer|)))
             (|has| |t#1| (|IntegralDomain|)))
          ((|integrate| (|$| |$|))
            (|has| |t#1| (|Algebra| (|Fraction| (|Integer|)))))
          ((|integrate| (|$| |$| (|Symbol|)))
            (AND
             (|has| |t#1|
               (SIGNATURE |variables| ((|List| (|Symbol|)) |t#1|)))
             (|has| |t#1| (SIGNATURE |integrate| (|t#1| |t#1| (|Symbol|))))
             (|has| |t#1| (|Algebra| (|Fraction| (|Integer|))))))
          ((|integrate| (|$| |$| (|Symbol|)))
            (AND
             (|has| |t#1| (|AlgebraicallyClosedFunctionSpace| (|Integer|)))
             (|has| |t#1| (|PrimitiveFunctionCategory|))
             (|has| |t#1| (|TranscendentalFunctionCategory|))
             (|has| |t#1| (|Algebra| (|Fraction| (|Integer|))))))))
        (QUOTE (
         ((|RadicalCategory|)
           (|has| |t#1| (|Algebra| (|Fraction| (|Integer|)))))
         ((|TranscendentalFunctionCategory|)
           (|has| |t#1| (|Algebra| (|Fraction| (|Integer|)))))
         ((|Field|) (|has| |t#1| (|Field|)))))
        (QUOTE (
         (|Symbol|)
         (|Fraction| (|Polynomial| |t#1|))
         (|Integer|)
         (|Stream| (|Record| (|:| |k| (|Integer|)) (|:| |c| |t#1|)))))
         NIL))
        . #3=(|UnivariateLaurentSeriesCategory|))))))
       . #3#)
(SETELT #1# 0
 (LIST (QUOTE |UnivariateLaurentSeriesCategory|) (|devaluate| |t#1|)))))))
```

———-

21.67 UPOLYC.lsp BOOTSTRAP

UPOLYC depends on itself. We need to break this cycle to build the algebra. So we

keep a cached copy of the translated **UPOLYC** category which we can write into the **MID** directory. We compile the lisp code and copy the **UPOLYC.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— UPOLYC.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |UnivariatePolynomialCategory;CAT| (QUOTE NIL))

(SETQ |UnivariatePolynomialCategory;AL| (QUOTE NIL))

(DEFUN |UnivariatePolynomialCategory| (#1=#:G103214)
  (LET (#2=#:G103215)
    (COND
      ((SETQ #2# (|assoc| (|devaluate| #1#) |UnivariatePolynomialCategory;AL|))
         (CDR #2#))
      (T
        (SETQ |UnivariatePolynomialCategory;AL|
          (|cons5|
            (CONS
              (|devaluate| #1#)
              (SETQ #2# (|UnivariatePolynomialCategory;| #1#)))
            |UnivariatePolynomialCategory;AL|))
         #2#))))

(DEFUN |UnivariatePolynomialCategory;| (|t#1|)
  (PROG (#1=#:G103213)
    (RETURN
      (PROG1
        (LETT #1#
          (|sublisV|
            (PAIR (QUOTE (|t#1|)) (LIST (|devaluate| |t#1|)))
            (|sublisV|
              (PAIR
                (QUOTE (#2=#:G103211 #3=#:G103212))
                (LIST
                  (QUOTE (|NonNegativeInteger|))
                  (QUOTE (|SingletonAsOrderedSet|))))
              (COND
                (|UnivariatePolynomialCategory;CAT|)
                ((QUOTE T)
                  (LETT |UnivariatePolynomialCategory;CAT|
                    (|Join|
                      (|PolynomialCategory|
                        (QUOTE |t#1|) (QUOTE #2#) (QUOTE #3#))
                      (|Eltable| (QUOTE |t#1|) (QUOTE |t#1|))
```

```
(|Eltable| (QUOTE |$|) (QUOTE |$|))
(|DifferentialRing|)
(|DifferentialExtension| (QUOTE |t#1|))
(|mkCategory|
  (QUOTE |domain|)
  (QUOTE (
    ((|vectorise|
      ((|Vector| |t#1|) |$| (|NonNegativeInteger|))) T)
    ((|makeSUP|
      ((|SparseUnivariatePolynomial| |t#1|) |$|)) T)
    ((|unmakeSUP|
      (|$| (|SparseUnivariatePolynomial| |t#1|))) T)
    ((|multiplyExponents|
      (|$| |$| (|NonNegativeInteger|))) T)
    ((|divideExponents|
      ((|Union| |$| "failed")
       |$|
       (|NonNegativeInteger|))) T)
    ((|monicDivide|
      ((|Record|
         (|:| |quotient| |$|)
         (|:| |remainder| |$|))
       |$|
       |$|)) T)
    ((|karatsubaDivide|
      ((|Record|
         (|:| |quotient| |$|)
         (|:| |remainder| |$|))
       |$|
       (|NonNegativeInteger|))) T)
    ((|shiftRight| (|$| |$| (|NonNegativeInteger|))) T)
    ((|shiftLeft| (|$| |$| (|NonNegativeInteger|))) T)
    ((|pseudoRemainder| (|$| |$| |$|)) T)
    ((|differentiate|
      (|$| |$| (|Mapping| |t#1| |t#1|) |$|)) T)
    ((|discriminant| (|t#1| |$|))
      (|has| |t#1| (|CommutativeRing|)))
    ((|resultant| (|t#1| |$| |$|))
      (|has| |t#1| (|CommutativeRing|)))
    ((|elt|
       ((|Fraction| |$|)
        (|Fraction| |$|)
        (|Fraction| |$|)))
      (|has| |t#1| (|IntegralDomain|)))
    ((|order| ((|NonNegativeInteger|) |$| |$|))
      (|has| |t#1| (|IntegralDomain|)))
    ((|subResultantGcd| (|$| |$| |$|))
      (|has| |t#1| (|IntegralDomain|)))
    ((|composite| ((|Union| |$| "failed") |$| |$|))
      (|has| |t#1| (|IntegralDomain|)))
```

```
                  ((|composite|
                     ((|Union| (|Fraction| |$|) "failed")
                      (|Fraction| |$|)
                      |$|))
                    (|has| |t#1| (|IntegralDomain|)))
                  ((|pseudoQuotient| (|$| |$| |$|))
                    (|has| |t#1| (|IntegralDomain|)))
                  ((|pseudoDivide|
                     ((|Record|
                       (|:| |coef| |t#1|)
                       (|:| |quotient| |$|)
                       (|:| |remainder| |$|))
                      |$|
                      |$|))
                    (|has| |t#1| (|IntegralDomain|)))
                  ((|separate|
                     ((|Record|
                       (|:| |primePart| |$|)
                       (|:| |commonPart| |$|))
                      |$|
                      |$|))
                    (|has| |t#1| (|GcdDomain|)))
                  ((|elt| (|t#1| (|Fraction| |$|) |t#1|))
                    (|has| |t#1| (|Field|)))
                  ((|integrate| (|$| |$|))
                    (|has| |t#1|
                      (|Algebra| (|Fraction| (|Integer|)))))))
                 (QUOTE (
                   ((|StepThrough|) (|has| |t#1| (|StepThrough|)))
                   ((|Eltable|
                      (|Fraction| |$|)
                      (|Fraction| |$|))
                     (|has| |t#1| (|IntegralDomain|)))
                   ((|EuclideanDomain|) (|has| |t#1| (|Field|)))
                   (|additiveValuation| (|has| |t#1| (|Field|)))))
                 (QUOTE (
                   (|Fraction| |$|)
                   (|NonNegativeInteger|)
                   (|SparseUnivariatePolynomial| |t#1|)
                   (|Vector| |t#1|)))
                 NIL))
          . #4=(|UnivariatePolynomialCategory|))))))
  . #4#)
(SETELT #1# 0
  (LIST
    (QUOTE |UnivariatePolynomialCategory|)
    (|devaluate| |t#1|))))))
```

———-

21.68 UPOLYC-.lsp BOOTSTRAP

UPOLYC- depends on **UPOLYC**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **UPOLYC-** category which we can write into the **MID** directory. We compile the lisp code and copy the **UPOLYC-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— UPOLYC-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |UPOLYC-;variables;SL;1| (|p| |$|)
  (COND
    ((OR
      (SPADCALL |p| (QREFELT |$| 9))
      (ZEROP (SPADCALL |p| (QREFELT |$| 11))))
     NIL)
    ((QUOTE T) (LIST (SPADCALL (QREFELT |$| 13))))))

(DEFUN |UPOLYC-;degree;SSaosNni;2| (|p| |v| |$|)
  (SPADCALL |p| (QREFELT |$| 11)))

(DEFUN |UPOLYC-;totalDegree;SLNni;3| (|p| |lv| |$|)
  (COND ((NULL |lv|) 0) ((QUOTE T) (SPADCALL |p| (QREFELT |$| 17)))))

(DEFUN |UPOLYC-;degree;SLL;4| (|p| |lv| |$|)
  (COND ((NULL |lv|) NIL) ((QUOTE T) (LIST (SPADCALL |p| (QREFELT |$| 11))))))

(DEFUN |UPOLYC-;eval;SLLS;5| (|p| |lv| |lq| |$|)
  (COND
    ((NULL |lv|) |p|)
    ((NULL (NULL (CDR |lv|)))
      (|error| "can only eval a univariate polynomial once"))
    ((QUOTE T)
      (SPADCALL |p| (|SPADfirst| |lv|) (|SPADfirst| |lq|) (QREFELT |$| 21)))))

(DEFUN |UPOLYC-;eval;SSaos2S;6| (|p| |v| |q| |$|)
  (SPADCALL |p| |q| (QREFELT |$| 24)))

(DEFUN |UPOLYC-;eval;SLLS;7| (|p| |lv| |lr| |$|)
  (COND
    ((NULL |lv|) |p|)
    ((NULL (NULL (CDR |lv|)))
      (|error| "can only eval a univariate polynomial once"))
    ((QUOTE T)
      (SPADCALL |p| (|SPADfirst| |lv|) (|SPADfirst| |lr|) (QREFELT |$| 26)))))
```

```
(DEFUN |UPOLYC-;eval;SSaosRS;8| (|p| |v| |r| |$|)
  (SPADCALL (SPADCALL |p| |r| (QREFELT |$| 29)) (QREFELT |$| 30)))

(DEFUN |UPOLYC-;eval;SLS;9| (|p| |le| |$|)
  (COND
    ((NULL |le|) |p|)
    ((NULL (NULL (CDR |le|)))
      (|error| "can only eval a univariate polynomial once"))
    ((QUOTE T)
      (COND
        ((QEQCAR
           (SPADCALL
             (SPADCALL (|SPADfirst| |le|) (QREFELT |$| 33))
             (QREFELT |$| 35))
            1)
          |p|)
        ((QUOTE T)
          (SPADCALL |p|
            (SPADCALL (|SPADfirst| |le|) (QREFELT |$| 36))
            (QREFELT |$| 24)))))))

(DEFUN |UPOLYC-;mainVariable;SU;10| (|p| |$|)
  (COND
    ((ZEROP (SPADCALL |p| (QREFELT |$| 11))) (CONS 1 "failed"))
    ((QUOTE T) (CONS 0 (SPADCALL (QREFELT |$| 13))))))

(DEFUN |UPOLYC-;minimumDegree;SSaosNni;11| (|p| |v| |$|)
  (SPADCALL |p| (QREFELT |$| 40)))

(DEFUN |UPOLYC-;minimumDegree;SLL;12| (|p| |lv| |$|)
  (COND ((NULL |lv|) NIL) ((QUOTE T) (LIST (SPADCALL |p| (QREFELT |$| 40))))))

(DEFUN |UPOLYC-;monomial;SSaosNniS;13| (|p| |v| |n| |$|)
  (SPADCALL
    (CONS (FUNCTION |UPOLYC-;monomial;SSaosNniS;13!0|) (VECTOR |$| |n|))
    |p|
    (QREFELT |$| 45)))

(DEFUN |UPOLYC-;monomial;SSaosNniS;13!0| (|#1| |$$|)
  (SPADCALL |#1| (QREFELT |$$| 1) (QREFELT (QREFELT |$$| 0) 43)))

(DEFUN |UPOLYC-;coerce;SaosS;14| (|v| |$|)
  (SPADCALL (|spadConstant| |$| 48) 1 (QREFELT |$| 49)))

(DEFUN |UPOLYC-;makeSUP;SSup;15| (|p| |$|)
  (COND
    ((SPADCALL |p| (QREFELT |$| 9)) (|spadConstant| |$| 52))
    ((QUOTE T)
      (SPADCALL
```

```
        (SPADCALL
          (SPADCALL |p| (QREFELT |$| 53))
          (SPADCALL |p| (QREFELT |$| 11))
          (QREFELT |$| 54))
        (SPADCALL
          (SPADCALL |p| (QREFELT |$| 55))
          (QREFELT |$| 56))
        (QREFELT |$| 57)))))

(DEFUN |UPOLYC-;unmakeSUP;SupS;16| (|sp| |$|)
  (COND
    ((SPADCALL |sp| (QREFELT |$| 59)) (|spadConstant| |$| 60))
    ((QUOTE T)
      (SPADCALL
        (SPADCALL
          (SPADCALL |sp| (QREFELT |$| 61))
          (SPADCALL |sp| (QREFELT |$| 62))
          (QREFELT |$| 49))
        (SPADCALL (SPADCALL |sp| (QREFELT |$| 63)) (QREFELT |$| 64))
      (QREFELT |$| 65)))))

(DEFUN |UPOLYC-;karatsubaDivide;SNniR;17| (|p| |n| |$|)
  (SPADCALL |p|
    (SPADCALL (|spadConstant| |$| 48) |n| (QREFELT |$| 49))
    (QREFELT |$| 68)))

(DEFUN |UPOLYC-;shiftRight;SNniS;18| (|p| |n| |$|)
  (QCAR
    (SPADCALL |p|
      (SPADCALL (|spadConstant| |$| 48) |n| (QREFELT |$| 49))
      (QREFELT |$| 68))))

(DEFUN |UPOLYC-;shiftLeft;SNniS;19| (|p| |n| |$|)
  (SPADCALL |p|
    (SPADCALL (|spadConstant| |$| 48) |n| (QREFELT |$| 49)) (QREFELT |$| 71)))

(DEFUN |UPOLYC-;solveLinearPolynomialEquation;LSupU;20| (|lpp| |pp| |$|)
  (SPADCALL |lpp| |pp| (QREFELT |$| 77)))

(DEFUN |UPOLYC-;factorPolynomial;SupF;21| (|pp| |$|)
  (SPADCALL |pp| (QREFELT |$| 83)))

(DEFUN |UPOLYC-;factorSquareFreePolynomial;SupF;22| (|pp| |$|)
  (SPADCALL |pp| (QREFELT |$| 86)))

(DEFUN |UPOLYC-;factor;SF;23| (|p| |$|)
  (PROG (|ansR| #1=#:G103310 |w| #2=#:G103311)
    (RETURN
      (SEQ
        (COND
```

```
          ((ZEROP (SPADCALL |p| (QREFELT |$| 11)))
            (SEQ
              (LETT |ansR|
                (SPADCALL
                  (SPADCALL |p| (QREFELT |$| 53))
                  (QREFELT |$| 89))
                |UPOLYC-;factor;SF;23|)
              (EXIT
                (SPADCALL
                  (SPADCALL
                    (SPADCALL |ansR| (QREFELT |$| 91))
                    (QREFELT |$| 30))
                  (PROGN
                    (LETT #1# NIL |UPOLYC-;factor;SF;23|)
                    (SEQ
                      (LETT |w| NIL |UPOLYC-;factor;SF;23|)
                      (LETT #2#
                        (SPADCALL |ansR| (QREFELT |$| 95))
                        |UPOLYC-;factor;SF;23|)
                      G190
                      (COND
                        ((OR
                          (ATOM #2#)
                          (PROGN
                            (LETT |w| (CAR #2#) |UPOLYC-;factor;SF;23|)
                            NIL))
                         (GO G191)))
                      (SEQ
                        (EXIT
                          (LETT #1#
                            (CONS
                              (VECTOR
                                (QVELT |w| 0)
                                (SPADCALL (QVELT |w| 1) (QREFELT |$| 30))
                                (QVELT |w| 2))
                               #1#)
                              |UPOLYC-;factor;SF;23|)))
                      (LETT #2# (CDR #2#) |UPOLYC-;factor;SF;23|)
                      (GO G190)
                      G191
                      (EXIT (NREVERSE0 #1#))))
                  (QREFELT |$| 99)))))
          ((QUOTE T)
            (SPADCALL
              (ELT |$| 64)
              (SPADCALL (SPADCALL |p| (QREFELT |$| 56)) (QREFELT |$| 100))
              (QREFELT |$| 104)))))))

(DEFUN |UPOLYC-;vectorise;SNniV;24| (|p| |n| |$|)
  (PROG (|v| |m| |i| #1=#:G103316 #2=#:G103312)
```

```
    (RETURN
      (SEQ
        (LETT |m|
          (SPADCALL
            (LETT |v|
              (SPADCALL |n| (|spadConstant| |$| 106) (QREFELT |$| 108))
              |UPOLYC-;vectorise;SNniV;24|)
            (QREFELT |$| 110))
          |UPOLYC-;vectorise;SNniV;24|)
        (SEQ
          (LETT |i|
            (SPADCALL |v| (QREFELT |$| 110))
            |UPOLYC-;vectorise;SNniV;24|)
          (LETT #1# (QVSIZE |v|) |UPOLYC-;vectorise;SNniV;24|)
          G190
          (COND ((|>| |i| #1#) (GO G191)))
          (SEQ
            (EXIT
              (SPADCALL |v| |i|
                (SPADCALL |p|
                  (PROG1
                    (LETT #2# (|-| |i| |m|) |UPOLYC-;vectorise;SNniV;24|)
                    (|check-subtype|
                      (|>=| #2# 0)
                      (QUOTE (|NonNegativeInteger|))
                      #2#))
                  (QREFELT |$| 111))
                (QREFELT |$| 112))))
          (LETT |i| (|+| |i| 1) |UPOLYC-;vectorise;SNniV;24|)
          (GO G190)
          G191
          (EXIT NIL))
        (EXIT |v|)))))

(DEFUN |UPOLYC-;retract;SR;25| (|p| |$|)
  (COND
    ((SPADCALL |p| (QREFELT |$| 9)) (|spadConstant| |$| 106))
    ((ZEROP (SPADCALL |p| (QREFELT |$| 11))) (SPADCALL |p| (QREFELT |$| 53)))
    ((QUOTE T) (|error| "Polynomial is not of degree 0"))))

(DEFUN |UPOLYC-;retractIfCan;SU;26| (|p| |$|)
  (COND
    ((SPADCALL |p| (QREFELT |$| 9)) (CONS 0 (|spadConstant| |$| 106)))
    ((ZEROP (SPADCALL |p| (QREFELT |$| 11)))
       (CONS 0 (SPADCALL |p| (QREFELT |$| 53))))
    ((QUOTE T) (CONS 1 "failed"))))

(DEFUN |UPOLYC-;init;S;27| (|$|)
  (SPADCALL (|spadConstant| |$| 117) (QREFELT |$| 30)))
```

```
(DEFUN |UPOLYC-;nextItemInner| (|n| |$|)
  (PROG (|nn| |n1| |n2| #1=#:G103337 |n3|)
    (RETURN
      (SEQ
        (COND
          ((SPADCALL |n| (QREFELT |$| 9))
            (CONS
              0
              (SPADCALL
                (PROG2
                  (LETT #1#
                    (SPADCALL (|spadConstant| |$| 106) (QREFELT |$| 120))
                    |UPOLYC-;nextItemInner|)
                  (QCDR #1#)
                  (|check-union| (QEQCAR #1# 0) (QREFELT |$| 7) #1#))
                (QREFELT |$| 30))))
          ((ZEROP (SPADCALL |n| (QREFELT |$| 11)))
            (SEQ
              (LETT |nn|
                (SPADCALL (SPADCALL |n| (QREFELT |$| 53)) (QREFELT |$| 120))
                |UPOLYC-;nextItemInner|)
              (EXIT
                (COND
                  ((QEQCAR |nn| 1) (CONS 1 "failed"))
                  ((QUOTE T)
                    (CONS 0 (SPADCALL (QCDR |nn|) (QREFELT |$| 30))))))))
          ((QUOTE T)
            (SEQ
              (LETT |n1|
                (SPADCALL |n| (QREFELT |$| 55))
                |UPOLYC-;nextItemInner|)
              (LETT |n2|
                (|UPOLYC-;nextItemInner| |n1| |$|)
                |UPOLYC-;nextItemInner|)
              (EXIT
                (COND
                  ((QEQCAR |n2| 0)
                    (CONS
                      0
                      (SPADCALL
                        (SPADCALL
                          (SPADCALL |n| (QREFELT |$| 53))
                          (SPADCALL |n| (QREFELT |$| 11))
                          (QREFELT |$| 49))
                        (QCDR |n2|)
                        (QREFELT |$| 65))))
                  ((|<|
                     (|+| 1 (SPADCALL |n1| (QREFELT |$| 11)))
                     (SPADCALL |n| (QREFELT |$| 11)))
                    (CONS
```

```
                    0
                    (SPADCALL
                      (SPADCALL
                        (SPADCALL |n| (QREFELT |$| 53))
                        (SPADCALL |n| (QREFELT |$| 11))
                        (QREFELT |$| 49))
                      (SPADCALL
                        (PROG2
                          (LETT #1#
                            (SPADCALL
                              (|spadConstant| |$| 117)
                              (QREFELT |$| 120))
                            |UPOLYC-;nextItemInner|)
                          (QCDR #1#)
                          (|check-union| (QEQCAR #1# 0) (QREFELT |$| 7) #1#))
                        (|+| 1 (SPADCALL |n1| (QREFELT |$| 11)))
                        (QREFELT |$| 49))
                      (QREFELT |$| 65))))
                  ((QUOTE T)
                    (SEQ
                      (LETT |n3|
                        (SPADCALL
                          (SPADCALL |n| (QREFELT |$| 53))
                          (QREFELT |$| 120))
                        |UPOLYC-;nextItemInner|)
                      (EXIT
                        (COND
                          ((QEQCAR |n3| 1) (CONS 1 "failed"))
                          ((QUOTE T)
                            (CONS
                              0
                              (SPADCALL
                                (QCDR |n3|)
                                (SPADCALL |n| (QREFELT |$| 11))
                                (QREFELT |$| 49)))))))))))

(DEFUN |UPOLYC-;nextItem;SU;29| (|n| |$|)
  (PROG (|n1| #1=#:G103350)
    (RETURN
      (SEQ
        (LETT |n1| (|UPOLYC-;nextItemInner| |n| |$|) |UPOLYC-;nextItem;SU;29|)
        (EXIT
          (COND
            ((QEQCAR |n1| 1)
              (CONS
                0
                (SPADCALL
                  (PROG2
                    (LETT #1#
                      (SPADCALL (|spadConstant| |$| 117) (QREFELT |$| 120))
```

```
                    |UPOLYC-;nextItem;SU;29|)
                   (QCDR #1#)
                   (|check-union| (QEQCAR #1# 0) (QREFELT |$| 7) #1#))
                  (|+| 1 (SPADCALL |n| (QREFELT |$| 11)))
                 (QREFELT |$| 49))))
            ((QUOTE T) |n1|)))))))

(DEFUN |UPOLYC-;content;SSaosS;30| (|p| |v| |$|)
  (SPADCALL (SPADCALL |p| (QREFELT |$| 123)) (QREFELT |$| 30)))

(DEFUN |UPOLYC-;primeFactor| (|p| |q| |$|)
  (PROG (#1=#:G103356 |p1|)
    (RETURN
      (SEQ
        (LETT |p1|
           (PROG2
             (LETT #1#
               (SPADCALL |p|
                 (SPADCALL |p| |q| (QREFELT |$| 125))
                 (QREFELT |$| 126))
               |UPOLYC-;primeFactor|)
             (QCDR #1#)
             (|check-union| (QEQCAR #1# 0) (QREFELT |$| 6) #1#))
           |UPOLYC-;primeFactor|)
        (EXIT
          (COND
            ((SPADCALL |p1| |p| (QREFELT |$| 127)) |p|)
            ((QUOTE T) (|UPOLYC-;primeFactor| |p1| |q| |$|)))))))))

(DEFUN |UPOLYC-;separate;2SR;32| (|p| |q| |$|)
  (PROG (|a| #1=#:G103362)
    (RETURN
      (SEQ
        (LETT |a|
          (|UPOLYC-;primeFactor| |p| |q| |$|)
          |UPOLYC-;separate;2SR;32|)
        (EXIT
          (CONS
            |a|
            (PROG2
              (LETT #1#
                (SPADCALL |p| |a| (QREFELT |$| 126))
                |UPOLYC-;separate;2SR;32|)
              (QCDR #1#)
              (|check-union| (QEQCAR #1# 0) (QREFELT |$| 6) #1#)))))))))

(DEFUN |UPOLYC-;differentiate;SM2S;33| (|x| |deriv| |x'| |$|)
  (PROG (|dg| |lc| #1=#:G103367 |d|)
    (RETURN
      (SEQ
```

```
(LETT |d| (|spadConstant| |$| 60) |UPOLYC-;differentiate;SM2S;33|)
(SEQ G190
  (COND
    ((NULL
       (|<| 0
         (LETT |dg|
           (SPADCALL |x| (QREFELT |$| 11))
           |UPOLYC-;differentiate;SM2S;33|)))
      (GO G191)))
  (SEQ
    (LETT |lc|
      (SPADCALL |x| (QREFELT |$| 53))
      |UPOLYC-;differentiate;SM2S;33|)
    (LETT |d|
      (SPADCALL
        (SPADCALL |d|
          (SPADCALL |x'|
            (SPADCALL
              (SPADCALL |dg| |lc| (QREFELT |$| 131))
              (PROG1
                (LETT #1# (|-| |dg| 1) |UPOLYC-;differentiate;SM2S;33|)
                (|check-subtype|
                  (|>=| #1# 0)
                  (QUOTE (|NonNegativeInteger|))
                  #1#))
              (QREFELT |$| 49))
            (QREFELT |$| 71))
          (QREFELT |$| 65))
          (SPADCALL
            (SPADCALL |lc| |deriv|)
            |dg|
            (QREFELT |$| 49))
        (QREFELT |$| 65))
      |UPOLYC-;differentiate;SM2S;33|)
    (EXIT
      (LETT |x|
        (SPADCALL |x| (QREFELT |$| 55))
        |UPOLYC-;differentiate;SM2S;33|)))
   NIL
   (GO G190)
   G191
   (EXIT NIL))
(EXIT
  (SPADCALL |d|
    (SPADCALL
      (SPADCALL
        (SPADCALL |x| (QREFELT |$| 53))
        |deriv|)
      (QREFELT |$| 30))
    (QREFELT |$| 65)))))))
```

```
(DEFUN |UPOLYC-;ncdiff| (|n| |x'| |$|)
  (PROG (#1=#:G103385 |n1|)
    (RETURN
      (COND
        ((ZEROP |n|) (|spadConstant| |$| 60))
        ((ZEROP
          (LETT |n1|
            (PROG1
              (LETT #1# (|-| |n| 1) |UPOLYC-;ncdiff|)
              (|check-subtype|
                (|>=| #1# 0)
                (QUOTE (|NonNegativeInteger|))
                #1#))
            |UPOLYC-;ncdiff|))
          |x'|)
        ((QUOTE T)
          (SPADCALL
            (SPADCALL |x'|
              (SPADCALL (|spadConstant| |$| 48) |n1| (QREFELT |$| 49))
              (QREFELT |$| 71))
            (SPADCALL
              (SPADCALL (|spadConstant| |$| 48) 1 (QREFELT |$| 49))
              (|UPOLYC-;ncdiff| |n1| |x'| |$|)
              (QREFELT |$| 71))
            (QREFELT |$| 65)))))))

(DEFUN |UPOLYC-;differentiate;SM2S;35| (|x| |deriv| |x'| |$|)
  (PROG (|dg| |lc| |d|)
    (RETURN
      (SEQ
        (LETT |d| (|spadConstant| |$| 60) |UPOLYC-;differentiate;SM2S;35|)
        (SEQ G190
          (COND
            ((NULL
               (|<| 0
                 (LETT |dg|
                   (SPADCALL |x| (QREFELT |$| 11))
                   |UPOLYC-;differentiate;SM2S;35|)))
              (GO G191)))
          (SEQ
            (LETT |lc|
              (SPADCALL |x| (QREFELT |$| 53))
              |UPOLYC-;differentiate;SM2S;35|)
            (LETT |d|
              (SPADCALL
                (SPADCALL |d|
                  (SPADCALL
                    (SPADCALL |lc| |deriv|)
                    |dg|
```

```
                  (QREFELT |$| 49))
                (QREFELT |$| 65))
              (SPADCALL |lc|
                (|UPOLYC-;ncdiff| |dg| |x'| |$|)
                (QREFELT |$| 134))
            (QREFELT |$| 65))
            |UPOLYC-;differentiate;SM2S;35|)
           (EXIT
             (LETT |x|
               (SPADCALL |x| (QREFELT |$| 55))
               |UPOLYC-;differentiate;SM2S;35|)))
          NIL
          (GO G190)
          G191
          (EXIT NIL))
        (EXIT
          (SPADCALL |d|
            (SPADCALL
              (SPADCALL (SPADCALL |x| (QREFELT |$| 53)) |deriv|)
              (QREFELT |$| 30))
            (QREFELT |$| 65)))))))

(DEFUN |UPOLYC-;differentiate;SMS;36| (|x| |deriv| |$|)
  (SPADCALL |x| |deriv| (|spadConstant| |$| 47) (QREFELT |$| 135)))

(DEFUN |UPOLYC-;differentiate;2S;37| (|x| |$|)
  (PROG (|dg| #1=#:G103394 |d|)
    (RETURN
      (SEQ
        (LETT |d| (|spadConstant| |$| 60) |UPOLYC-;differentiate;2S;37|)
        (SEQ G190
          (COND
            ((NULL
              (|<| 0
                (LETT |dg|
                  (SPADCALL |x| (QREFELT |$| 11))
                  |UPOLYC-;differentiate;2S;37|)))
              (GO G191)))
            (SEQ
              (LETT |d|
                (SPADCALL |d|
                  (SPADCALL
                    (SPADCALL |dg|
                      (SPADCALL |x| (QREFELT |$| 53)) (QREFELT |$| 131))
                    (PROG1
                      (LETT #1# (|-| |dg| 1) |UPOLYC-;differentiate;2S;37|)
                      (|check-subtype|
                        (|>=| #1# 0)
                        (QUOTE (|NonNegativeInteger|))
                        #1#))
```

```
                (QREFELT |$| 49))
               (QREFELT |$| 65))
              |UPOLYC-;differentiate;2S;37|)
             (EXIT
               (LETT |x|
                 (SPADCALL |x| (QREFELT |$| 55))
                 |UPOLYC-;differentiate;2S;37|)))
            NIL
            (GO G190)
            G191
            (EXIT NIL))
          (EXIT |d|)))))

(DEFUN |UPOLYC-;differentiate;SSaosS;38| (|x| |v| |$|)
  (SPADCALL |x| (QREFELT |$| 138)))

(DEFUN |UPOLYC-;elt;3F;39| (|g| |f| |$|)
  (SPADCALL
    (SPADCALL
      (SPADCALL |g| (QREFELT |$| 141))
      |f|
      (QREFELT |$| 143))
    (SPADCALL (SPADCALL |g| (QREFELT |$| 144)) |f| (QREFELT |$| 143))
    (QREFELT |$| 145)))

(DEFUN |UPOLYC-;pseudoQuotient;3S;40| (|p| |q| |$|)
  (PROG (|n| #1=#:G103440 #2=#:G103442)
    (RETURN
      (SEQ
        (LETT |n|
          (|+|
            (|-|
              (SPADCALL |p| (QREFELT |$| 11))
              (SPADCALL |q| (QREFELT |$| 11))) 1)
           |UPOLYC-;pseudoQuotient;3S;40|)
        (EXIT
          (COND
            ((|<| |n| 1) (|spadConstant| |$| 60))
            ((QUOTE T)
              (PROG2
                (LETT #2#
                  (SPADCALL
                    (SPADCALL
                      (SPADCALL
                        (SPADCALL
                          (SPADCALL |q| (QREFELT |$| 53))
                          (PROG1
                            (LETT #1# |n| |UPOLYC-;pseudoQuotient;3S;40|)
                            (|check-subtype|
                              (|>=| #1# 0)
```

```
                              (QUOTE (|NonNegativeInteger|))
                              #1#))
                            (QREFELT |$| 147))
                          |p|
                          (QREFELT |$| 134))
                        (SPADCALL |p| |q| (QREFELT |$| 148))
                        (QREFELT |$| 149))
                      |q|
                      (QREFELT |$| 126))
                    |UPOLYC-;pseudoQuotient;3S;40|)
                  (QCDR #2#)
                  (|check-union| (QEQCAR #2# 0) (QREFELT |$| 6) #2#)))))))))

(DEFUN |UPOLYC-;pseudoDivide;2SR;41| (|p| |q| |$|)
  (PROG (|n| |prem| #1=#:G103448 |lc| #2=#:G103450)
    (RETURN
      (SEQ
        (LETT |n|
          (|+|
            (|-|
              (SPADCALL |p| (QREFELT |$| 11))
              (SPADCALL |q| (QREFELT |$| 11))) 1)
          |UPOLYC-;pseudoDivide;2SR;41|)
        (EXIT
          (COND
            ((|<| |n| 1)
              (VECTOR (|spadConstant| |$| 48) (|spadConstant| |$| 60) |p|))
            ((QUOTE T)
              (SEQ
                (LETT |prem|
                  (SPADCALL |p| |q| (QREFELT |$| 148))
                  |UPOLYC-;pseudoDivide;2SR;41|)
                (LETT |lc|
                  (SPADCALL
                    (SPADCALL |q| (QREFELT |$| 53))
                    (PROG1
                      (LETT #1# |n| |UPOLYC-;pseudoDivide;2SR;41|)
                      (|check-subtype|
                        (|>=| #1# 0)
                        (QUOTE (|NonNegativeInteger|)) #1#))
                    (QREFELT |$| 147))
                  |UPOLYC-;pseudoDivide;2SR;41|)
                (EXIT
                  (VECTOR |lc|
                    (PROG2
                      (LETT #2#
                        (SPADCALL
                          (SPADCALL
                            (SPADCALL |lc| |p| (QREFELT |$| 134))
                            |prem|
```

```
                          (QREFELT |$| 149))
                         |q|
                         (QREFELT |$| 126))
                        |UPOLYC-;pseudoDivide;2SR;41|)
                       (QCDR #2#)
                       (|check-union| (QEQCAR #2# 0) (QREFELT |$| 6) #2#))
                      |prem|))))))))))

(DEFUN |UPOLYC-;composite;FSU;42| (|f| |q| |$|)
  (PROG (|n| |d|)
    (RETURN
      (SEQ
        (LETT |n|
          (SPADCALL (SPADCALL |f| (QREFELT |$| 141)) |q| (QREFELT |$| 153))
          |UPOLYC-;composite;FSU;42|)
        (EXIT
          (COND
            ((QEQCAR |n| 1) (CONS 1 "failed"))
            ((QUOTE T)
              (SEQ
                (LETT |d|
                  (SPADCALL
                    (SPADCALL |f| (QREFELT |$| 144)) |q| (QREFELT |$| 153))
                  |UPOLYC-;composite;FSU;42|)
                (EXIT
                  (COND
                    ((QEQCAR |d| 1) (CONS 1 "failed"))
                    ((QUOTE T)
                      (CONS
                        0
                        (SPADCALL
                          (QCDR |n|)
                          (QCDR |d|)
                          (QREFELT |$| 154))))))))))))))

(DEFUN |UPOLYC-;composite;2SU;43| (|p| |q| |$|)
  (PROG (|cqr| |v| |u| |w| #1=#:G103476)
    (RETURN
      (SEQ
        (COND
          ((SPADCALL |p| (QREFELT |$| 157)) (CONS 0 |p|))
          ((QUOTE T)
            (SEQ
              (EXIT
                (SEQ
                  (LETT |cqr|
                    (SPADCALL |p| |q| (QREFELT |$| 158))
                    |UPOLYC-;composite;2SU;43|)
                  (COND
                    ((SPADCALL (QVELT |cqr| 2) (QREFELT |$| 157))
```

```
                (SEQ
                  (LETT |v|
                    (SPADCALL
                      (QVELT |cqr| 2)
                      (QVELT |cqr| 0)
                      (QREFELT |$| 159))
                    |UPOLYC-;composite;2SU;43|)
                  (EXIT
                    (COND
                      ((QEQCAR |v| 0)
                        (SEQ
                          (LETT |u|
                            (SPADCALL
                              (QVELT |cqr| 1)
                              |q|
                              (QREFELT |$| 153))
                            |UPOLYC-;composite;2SU;43|)
                          (EXIT
                            (COND
                              ((QEQCAR |u| 0)
                                (SEQ
                                  (LETT |w|
                                    (SPADCALL
                                      (QCDR |u|)
                                      (QVELT |cqr| 0)
                                      (QREFELT |$| 159))
                                    |UPOLYC-;composite;2SU;43|)
                                  (EXIT
                                    (COND
                                      ((QEQCAR |w| 0)
                                        (PROGN
                                          (LETT #1#
                                            (CONS
                                              0
                                              (SPADCALL
                                                (QCDR |v|)
                                                (SPADCALL
                                                  (SPADCALL
                                                   (|spadConstant| |$| 48)
                                                   1
                                                   (QREFELT |$| 49))
                                                  (QCDR |w|)
                                                  (QREFELT |$| 71))
                                                (QREFELT |$| 65)))
                                            |UPOLYC-;composite;2SU;43|)
                                          (GO #1#)))))))))))))
                    (EXIT (CONS 1 "failed"))))
                #1#
                (EXIT #1#))))))))
```

```
(DEFUN |UPOLYC-;elt;S2F;44| (|p| |f| |$|)
  (PROG (|n| #1=#:G103483 |ans|)
    (RETURN
      (SEQ
        (COND
          ((SPADCALL |p| (QREFELT |$| 9)) (|spadConstant| |$| 161))
          ((QUOTE T)
            (SEQ
              (LETT |ans|
                (SPADCALL
                  (SPADCALL (SPADCALL |p| (QREFELT |$| 53)) (QREFELT |$| 30))
                  (QREFELT |$| 162))
                |UPOLYC-;elt;S2F;44|)
              (LETT |n| (SPADCALL |p| (QREFELT |$| 11)) |UPOLYC-;elt;S2F;44|)
              (SEQ G190
                (COND
                  ((NULL
                    (COND
                      ((SPADCALL
                        (LETT |p|
                          (SPADCALL |p| (QREFELT |$| 55))
                          |UPOLYC-;elt;S2F;44|)
                        (QREFELT |$| 9))
                       (QUOTE NIL))
                      ((QUOTE T) (QUOTE T))))
                   (GO G191)))
                (SEQ
                  (EXIT
                    (LETT |ans|
                      (SPADCALL
                        (SPADCALL |ans|
                          (SPADCALL |f|
                            (PROG1
                              (LETT #1#
                                (|-| |n|
                                  (LETT |n|
                                    (SPADCALL |p| (QREFELT |$| 11))
                                    |UPOLYC-;elt;S2F;44|))
                                |UPOLYC-;elt;S2F;44|)
                              (|check-subtype|
                                (|>=| #1# 0)
                                (QUOTE (|NonNegativeInteger|))
                                #1#))
                            (QREFELT |$| 163))
                          (QREFELT |$| 164))
                        (SPADCALL
                          (SPADCALL
                            (SPADCALL |p| (QREFELT |$| 53))
                            (QREFELT |$| 30))
                          (QREFELT |$| 162))
```

```
                    (QREFELT |$| 165))
                  |UPOLYC-;elt;S2F;44|)))
              NIL
              (GO G190)
              G191
              (EXIT NIL))
             (EXIT
              (COND
                ((ZEROP |n|) |ans|)
                ((QUOTE T)
                  (SPADCALL |ans|
                    (SPADCALL |f| |n| (QREFELT |$| 166))
                    (QREFELT |$| 164)))))))))))

(DEFUN |UPOLYC-;order;2SNni;45| (|p| |q| |$|)
  (PROG (|u| #1=#:G103497 |ans|)
    (RETURN
      (SEQ
        (EXIT
          (COND
            ((SPADCALL |p| (QREFELT |$| 9))
              (|error| "order: arguments must be nonzero"))
            ((|<| (SPADCALL |q| (QREFELT |$| 11)) 1)
              (|error| "order: place must be non-trivial"))
            ((QUOTE T)
              (SEQ
                (LETT |ans| 0 |UPOLYC-;order;2SNni;45|)
                (EXIT
                  (SEQ G190
                    NIL
                    (SEQ
                      (LETT |u|
                        (SPADCALL |p| |q| (QREFELT |$| 126))
                        |UPOLYC-;order;2SNni;45|)
                      (EXIT
                        (COND
                          ((QEQCAR |u| 1)
                            (PROGN
                              (LETT #1# |ans| |UPOLYC-;order;2SNni;45|)
                              (GO #1#)))
                          ((QUOTE T)
                            (SEQ
                              (LETT |p| (QCDR |u|) |UPOLYC-;order;2SNni;45|)
                              (EXIT
                                (LETT
                                  |ans|
                                  (|+| |ans| 1)
                                  |UPOLYC-;order;2SNni;45|)))))))
                    NIL
                    (GO G190)
```

```
                  G191
                  (EXIT NIL)))))))
         #1#
         (EXIT #1#)))))

(DEFUN |UPOLYC-;squareFree;SF;46| (|p| |$|)
  (SPADCALL |p| (QREFELT |$| 170)))

(DEFUN |UPOLYC-;squareFreePart;2S;47| (|p| |$|)
  (SPADCALL |p| (QREFELT |$| 172)))

(DEFUN |UPOLYC-;gcdPolynomial;3Sup;48| (|pp| |qq| |$|)
  (COND
    ((SPADCALL |pp| (QREFELT |$| 174)) (SPADCALL |qq| (QREFELT |$| 175)))
    ((SPADCALL |qq| (QREFELT |$| 174)) (SPADCALL |pp| (QREFELT |$| 175)))
    ((QUOTE T)
      (SPADCALL
        (SPADCALL
          (SPADCALL
            (SPADCALL |pp| (QREFELT |$| 176))
            (SPADCALL |qq| (QREFELT |$| 176)) (QREFELT |$| 125))
          (SPADCALL
            (SPADCALL
              (SPADCALL |pp| (QREFELT |$| 177))
              (SPADCALL |qq| (QREFELT |$| 177)) (QREFELT |$| 178))
            (QREFELT |$| 177))
          (QREFELT |$| 179))
        (QREFELT |$| 175)))))

(DEFUN |UPOLYC-;squareFreePolynomial;SupF;49| (|pp| |$|)
  (SPADCALL |pp| (QREFELT |$| 182)))

(DEFUN |UPOLYC-;elt;F2R;50| (|f| |r| |$|)
  (SPADCALL
    (SPADCALL
      (SPADCALL |f| (QREFELT |$| 141))
      |r|
      (QREFELT |$| 29))
    (SPADCALL (SPADCALL |f| (QREFELT |$| 144)) |r| (QREFELT |$| 29))
    (QREFELT |$| 184)))

(DEFUN |UPOLYC-;euclideanSize;SNni;51| (|x| |$|)
  (COND
    ((SPADCALL |x| (QREFELT |$| 9))
      (|error| "euclideanSize called on 0 in Univariate Polynomial"))
    ((QUOTE T) (SPADCALL |x| (QREFELT |$| 11)))))

(DEFUN |UPOLYC-;divide;2SR;52| (|x| |y| |$|)
  (PROG (|lc| |f| #1=#:G103510 |n| |quot|)
    (RETURN
```

```
(SEQ
  (COND
    ((SPADCALL |y| (QREFELT |$| 9))
      (|error| "division by 0 in Univariate Polynomials"))
    ((QUOTE T)
      (SEQ
        (LETT |quot| (|spadConstant| |$| 60) |UPOLYC-;divide;2SR;52|)
        (LETT |lc|
          (SPADCALL (SPADCALL |y| (QREFELT |$| 53)) (QREFELT |$| 187))
          |UPOLYC-;divide;2SR;52|)
        (SEQ G190
          (COND
            ((NULL
               (COND
                 ((OR
                    (SPADCALL |x| (QREFELT |$| 9))
                    (|<|
                      (SPADCALL |x| (QREFELT |$| 11))
                      (SPADCALL |y| (QREFELT |$| 11))))
                  (QUOTE NIL)) ((QUOTE T) (QUOTE T))))
              (GO G191)))
            (SEQ
              (LETT |f|
                (SPADCALL |lc|
                  (SPADCALL |x| (QREFELT |$| 53))
                  (QREFELT |$| 188))
                |UPOLYC-;divide;2SR;52|)
              (LETT |n|
                (PROG1
                  (LETT #1#
                    (|-|
                      (SPADCALL |x| (QREFELT |$| 11))
                      (SPADCALL |y| (QREFELT |$| 11)))
                     |UPOLYC-;divide;2SR;52|)
                  (|check-subtype|
                    (|>=| #1# 0)
                    (QUOTE (|NonNegativeInteger|))
                    #1#))
                |UPOLYC-;divide;2SR;52|)
              (LETT |quot|
                (SPADCALL |quot|
                  (SPADCALL |f| |n| (QREFELT |$| 49))
                  (QREFELT |$| 65))
                |UPOLYC-;divide;2SR;52|)
              (EXIT
                (LETT |x|
                  (SPADCALL |x|
                    (SPADCALL
                      (SPADCALL |f| |n| (QREFELT |$| 49))
                      |y|
```

```
                        (QREFELT |$| 71))
                      (QREFELT |$| 149))
                    |UPOLYC-;divide;2SR;52|)))
                NIL
                (GO G190)
                G191
                (EXIT NIL))
              (EXIT (CONS |quot| |x|)))))))))

(DEFUN |UPOLYC-;integrate;2S;53| (|p| |$|)
  (PROG (|l| |d| |ans|)
    (RETURN
      (SEQ
        (LETT |ans| (|spadConstant| |$| 60) |UPOLYC-;integrate;2S;53|)
        (SEQ G190
          (COND
            ((NULL
              (COND
                ((SPADCALL |p| (|spadConstant| |$| 60) (QREFELT |$| 127))
                  (QUOTE NIL))
                ((QUOTE T) (QUOTE T)))) (GO G191)))
          (SEQ
            (LETT |l|
              (SPADCALL |p| (QREFELT |$| 53))
              |UPOLYC-;integrate;2S;53|)
            (LETT |d|
              (|+| 1 (SPADCALL |p| (QREFELT |$| 11)))
              |UPOLYC-;integrate;2S;53|)
            (LETT |ans|
              (SPADCALL |ans|
                (SPADCALL
                  (SPADCALL (SPADCALL |d| (QREFELT |$| 191)) (QREFELT |$| 192))
                  (SPADCALL |l| |d| (QREFELT |$| 49)) (QREFELT |$| 193))
                (QREFELT |$| 65))
              |UPOLYC-;integrate;2S;53|)
            (EXIT
              (LETT |p|
                (SPADCALL |p| (QREFELT |$| 55))
                |UPOLYC-;integrate;2S;53|)))
          NIL
          (GO G190)
          G191
          (EXIT NIL))
        (EXIT |ans|)))))

(DEFUN |UnivariatePolynomialCategory&| (|#1| |#2|)
  (PROG (|DV$1| |DV$2| |dv$| |$| |pv$|)
    (RETURN
      (PROGN
        (LETT |DV$1| (|devaluate| |#1|) . #1=(|UnivariatePolynomialCategory&|))
```

```
(LETT |DV$2| (|devaluate| |#2|) . #1#)
(LETT |dv$|
  (LIST (QUOTE |UnivariatePolynomialCategory&|) |DV$1| |DV$2|) . #1#)
(LETT |$| (make-array 201) . #1#)
(QSETREFV |$| 0 |dv$|)
(QSETREFV |$| 3
  (LETT |pv$|
    (|buildPredVector| 0 0
      (LIST
        (|HasCategory| |#2|
          (QUOTE (|Algebra| (|Fraction| (|Integer|)))))
        (|HasCategory| |#2| (QUOTE (|Field|)))
        (|HasCategory| |#2| (QUOTE (|GcdDomain|)))
        (|HasCategory| |#2| (QUOTE (|IntegralDomain|)))
        (|HasCategory| |#2| (QUOTE (|CommutativeRing|)))
        (|HasCategory| |#2| (QUOTE (|StepThrough|)))) . #1#))
(|stuffDomainSlots| |$|)
(QSETREFV |$| 6 |#1|)
(QSETREFV |$| 7 |#2|)
(COND
  ((|HasCategory| |#2| (QUOTE (|PolynomialFactorizationExplicit|)))
    (PROGN
      (QSETREFV |$| 81
        (CONS
          (|dispatchFunction|
             |UPOLYC-;solveLinearPolynomialEquation;LSupU;20|)
          |$|))
      (QSETREFV |$| 85
        (CONS
          (|dispatchFunction| |UPOLYC-;factorPolynomial;SupF;21|)
          |$|))
      (QSETREFV |$| 87
         (CONS
            (|dispatchFunction|
               |UPOLYC-;factorSquareFreePolynomial;SupF;22|)
            |$|))
      (QSETREFV |$| 105
        (CONS (|dispatchFunction| |UPOLYC-;factor;SF;23|) |$|)))))
(COND
  ((|testBitVector| |pv$| 6)
    (PROGN
      (QSETREFV |$| 118
        (CONS (|dispatchFunction| |UPOLYC-;init;S;27|) |$|))
      NIL
      (QSETREFV |$| 122
        (CONS (|dispatchFunction| |UPOLYC-;nextItem;SU;29|) |$|)))))
(COND
  ((|testBitVector| |pv$| 3)
    (PROGN
      (QSETREFV |$| 124
```

```
        (CONS (|dispatchFunction| |UPOLYC-;content;SSaosS;30|) |$|))
      NIL
      (QSETREFV |$| 129
        (CONS (|dispatchFunction| |UPOLYC-;separate;2SR;32|) |$|))))))
(COND
  ((|testBitVector| |pv$| 5)
    (QSETREFV |$| 133
      (CONS
        (|dispatchFunction| |UPOLYC-;differentiate;SM2S;33|)
        |$|)))
  ((QUOTE T)
    (PROGN
      (QSETREFV |$| 133
        (CONS
          (|dispatchFunction| |UPOLYC-;differentiate;SM2S;35|)
          |$|))))))
(COND
  ((|testBitVector| |pv$| 4)
    (PROGN
      (QSETREFV |$| 146
        (CONS (|dispatchFunction| |UPOLYC-;elt;3F;39|) |$|))
      (QSETREFV |$| 150
        (CONS (|dispatchFunction| |UPOLYC-;pseudoQuotient;3S;40|) |$|))
      (QSETREFV |$| 152
        (CONS (|dispatchFunction| |UPOLYC-;pseudoDivide;2SR;41|) |$|))
      (QSETREFV |$| 156
        (CONS (|dispatchFunction| |UPOLYC-;composite;FSU;42|) |$|))
      (QSETREFV |$| 160
        (CONS (|dispatchFunction| |UPOLYC-;composite;2SU;43|) |$|))
      (QSETREFV |$| 167
        (CONS (|dispatchFunction| |UPOLYC-;elt;S2F;44|) |$|))
      (QSETREFV |$| 168
        (CONS (|dispatchFunction| |UPOLYC-;order;2SNni;45|) |$|)))))
(COND
  ((|testBitVector| |pv$| 3)
    (PROGN
      (QSETREFV |$| 171
        (CONS (|dispatchFunction| |UPOLYC-;squareFree;SF;46|) |$|))
      (QSETREFV |$| 173
        (CONS
          (|dispatchFunction| |UPOLYC-;squareFreePart;2S;47|)
          |$|)))))
(COND
  ((|HasCategory| |#2| (QUOTE (|PolynomialFactorizationExplicit|)))
    (PROGN
      (QSETREFV |$| 180
        (CONS
          (|dispatchFunction| |UPOLYC-;gcdPolynomial;3Sup;48|)
          |$|))
      (QSETREFV |$| 183
```

```
              (CONS
                (|dispatchFunction| |UPOLYC-;squareFreePolynomial;SupF;49|)
                |$|)))))
        (COND
          ((|testBitVector| |pv$| 2)
            (PROGN
              (QSETREFV |$| 185
                (CONS (|dispatchFunction| |UPOLYC-;elt;F2R;50|) |$|))
              (QSETREFV |$| 186
                (CONS
                  (|dispatchFunction| |UPOLYC-;euclideanSize;SNni;51|)
                  |$|))
              (QSETREFV |$| 189
                (CONS (|dispatchFunction| |UPOLYC-;divide;2SR;52|) |$|)))))
        (COND
          ((|testBitVector| |pv$| 1)
             (QSETREFV |$| 194
               (CONS
                 (|dispatchFunction| |UPOLYC-;integrate;2S;53|)
                 |$|))))) |$|))))

(setf (get
  (QUOTE |UnivariatePolynomialCategory&|)
  (QUOTE |infovec|))
  (LIST
    (QUOTE
      #(NIL NIL NIL NIL NIL NIL
        (|local| |#1|)
        (|local| |#2|)
        (|Boolean|)
        (0 . |zero?|)
        (|NonNegativeInteger|)
        (5 . |degree|)
        (|SingletonAsOrderedSet|)
        (10 . |create|)
        (|List| 12)
        |UPOLYC-;variables;SL;1|
        |UPOLYC-;degree;SSaosNni;2|
        (14 . |totalDegree|)
        |UPOLYC-;totalDegree;SLNni;3|
        (|List| 10)
        |UPOLYC-;degree;SLL;4|
        (19 . |eval|)
        (|List| |$|)
        |UPOLYC-;eval;SLLS;5|
        (26 . |elt|)
        |UPOLYC-;eval;SSaos2S;6|
        (32 . |eval|)
        (|List| 7)
        |UPOLYC-;eval;SLLS;7|
```

```
(39 . |elt|)
(45 . |coerce|)
|UPOLYC-;eval;SSaosRS;8|
(|Equation| 6)
(50 . |lhs|)
(|Union| 12 (QUOTE "failed"))
(55 . |mainVariable|)
(60 . |rhs|)
(|List| 197)
|UPOLYC-;eval;SLS;9|
|UPOLYC-;mainVariable;SU;10|
(65 . |minimumDegree|)
|UPOLYC-;minimumDegree;SSaosNni;11|
|UPOLYC-;minimumDegree;SLL;12|
(70 . |+|)
(|Mapping| 10 10)
(76 . |mapExponents|)
|UPOLYC-;monomial;SSaosNniS;13|
(82 . |One|)
(86 . |One|)
(90 . |monomial|)
|UPOLYC-;coerce;SaosS;14|
(|SparseUnivariatePolynomial| 7)
(96 . |Zero|)
(100 . |leadingCoefficient|)
(105 . |monomial|)
(111 . |reductum|)
(116 . |makeSUP|)
(121 . |+|)
|UPOLYC-;makeSUP;SSup;15|
(127 . |zero?|)
(132 . |Zero|)
(136 . |leadingCoefficient|)
(141 . |degree|)
(146 . |reductum|)
(151 . |unmakeSUP|)
(156 . |+|)
|UPOLYC-;unmakeSUP;SupS;16|
(|Record| (|:| |quotient| |$|) (|:| |remainder| |$|))
(162 . |monicDivide|)
|UPOLYC-;karatsubaDivide;SNniR;17|
|UPOLYC-;shiftRight;SNniS;18|
(168 . |*|)
|UPOLYC-;shiftLeft;SNniS;19|
(|Union| 74 (QUOTE "failed"))
(|List| 75)
(|SparseUnivariatePolynomial| 6)
(|PolynomialFactorizationByRecursionUnivariate| 7 6)
(174 . |solveLinearPolynomialEquationByRecursion|)
(|Union| 79 (QUOTE "failed"))
```

```
(|List| 80)
(|SparseUnivariatePolynomial| |$|)
(180 . |solveLinearPolynomialEquation|)
(|Factored| 75)
(186 . |factorByRecursion|)
(|Factored| 80)
(191 . |factorPolynomial|)
(196 . |factorSquareFreeByRecursion|)
(201 . |factorSquareFreePolynomial|)
(|Factored| |$|)
(206 . |factor|)
(|Factored| 7)
(211 . |unit|)
(|Union| (QUOTE "nil") (QUOTE "sqfr") (QUOTE "irred") (QUOTE "prime"))
(|Record| (|:| |flg| 92) (|:| |fctr| 7) (|:| |xpnt| 109))
(|List| 93)
(216 . |factorList|)
(|Record| (|:| |flg| 92) (|:| |fctr| 6) (|:| |xpnt| 109))
(|List| 96)
(|Factored| 6)
(221 . |makeFR|)
(227 . |factorPolynomial|)
(|Mapping| 6 51)
(|Factored| 51)
(|FactoredFunctions2| 51 6)
(232 . |map|)
(238 . |factor|)
(243 . |Zero|)
(|Vector| 7)
(247 . |new|)
(|Integer|)
(253 . |minIndex|)
(258 . |coefficient|)
(264 . |qsetelt!|)
|UPOLYC-;vectorise;SNniV;24|
|UPOLYC-;retract;SR;25|
(|Union| 7 (QUOTE "failed"))
|UPOLYC-;retractIfCan;SU;26|
(271 . |init|)
(275 . |init|)
(|Union| |$| (QUOTE "failed"))
(279 . |nextItem|)
(284 . |One|)
(288 . |nextItem|)
(293 . |content|)
(298 . |content|)
(304 . |gcd|)
(310 . |exquo|)
(316 . |=|)
(|Record| (|:| |primePart| |$|) (|:| |commonPart| |$|))
```

```
(322 . |separate|)
(328 . |Zero|)
(332 . |*|)
(|Mapping| 7 7)
(338 . |differentiate|)
(345 . |*|)
(351 . |differentiate|)
|UPOLYC-;differentiate;SMS;36|
|UPOLYC-;differentiate;2S;37|
(358 . |differentiate|)
|UPOLYC-;differentiate;SSaosS;38|
(|Fraction| 6)
(363 . |numer|)
(|Fraction| |$|)
(368 . |elt|)
(374 . |denom|)
(379 . |/|)
(385 . |elt|)
(391 . |**|)
(397 . |pseudoRemainder|)
(403 . |-|)
(409 . |pseudoQuotient|)
(|Record| (|:| |coef| 7) (|:| |quotient| |$|) (|:| |remainder| |$|))
(415 . |pseudoDivide|)
(421 . |composite|)
(427 . |/|)
(|Union| 142 (QUOTE "failed"))
(433 . |composite|)
(439 . |ground?|)
(444 . |pseudoDivide|)
(450 . |exquo|)
(456 . |composite|)
(462 . |Zero|)
(466 . |coerce|)
(471 . |**|)
(477 . |*|)
(483 . |+|)
(489 . |**|)
(495 . |elt|)
(501 . |order|)
(|UnivariatePolynomialSquareFree| 7 6)
(507 . |squareFree|)
(512 . |squareFree|)
(517 . |squareFreePart|)
(522 . |squareFreePart|)
(527 . |zero?|)
(532 . |unitCanonical|)
(537 . |content|)
(542 . |primitivePart|)
(547 . |subResultantGcd|)
```

```
    (553 . |*|)
    (559 . |gcdPolynomial|)
    (|UnivariatePolynomialSquareFree| 6 75)
    (565 . |squareFree|)
    (570 . |squareFreePolynomial|)
    (575 . |/|)
    (581 . |elt|)
    (587 . |euclideanSize|)
    (592 . |inv|)
    (597 . |*|)
    (603 . |divide|)
    (|Fraction| 109)
    (609 . |coerce|)
    (614 . |inv|)
    (619 . |*|)
    (625 . |integrate|)
    (|Symbol|)
    (|List| 195)
    (|Equation| |$|)
    (|Union| 109 (QUOTE "failed"))
    (|Union| 190 (QUOTE "failed"))
    (|OutputForm|)))
(QUOTE
  #(|vectorise| 630 |variables| 636 |unmakeSUP| 641 |totalDegree| 646
    |squareFreePolynomial| 652 |squareFreePart| 657 |squareFree| 662
    |solveLinearPolynomialEquation| 667 |shiftRight| 673 |shiftLeft| 679
    |separate| 685 |retractIfCan| 691 |retract| 696 |pseudoQuotient| 701
    |pseudoDivide| 707 |order| 713 |nextItem| 719 |monomial| 724
    |minimumDegree| 731 |makeSUP| 743 |mainVariable| 748
    |karatsubaDivide| 753 |integrate| 759 |init| 764 |gcdPolynomial| 768
    |factorSquareFreePolynomial| 774 |factorPolynomial| 779 |factor| 784
    |eval| 789 |euclideanSize| 823 |elt| 828 |divide| 846
    |differentiate| 852 |degree| 876 |content| 888 |composite| 894
    |coerce| 906))
(QUOTE NIL)
(CONS
  (|makeByteWordVec2| 1 (QUOTE NIL))
  (CONS
    (QUOTE #())
    (CONS
      (QUOTE #())
      (|makeByteWordVec2| 194
        (QUOTE
          (1 6 8 0 9 1 6 10 0 11 0 12 0 13 1 6 10 0 17 3 6 0 0 12 0 21 2
           6 0 0 0 24 3 6 0 0 12 7 26 2 6 7 0 7 29 1 6 0 7 30 1 32 6 0 33
           1 6 34 0 35 1 32 6 0 36 1 6 10 0 40 2 10 0 0 0 43 2 6 0 44 0
           45 0 6 0 47 0 7 0 48 2 6 0 7 10 49 0 51 0 52 1 6 7 0 53 2 51
           0 7 10 54 1 6 0 0 55 1 6 51 0 56 2 51 0 0 0 57 1 51 8 0 59 0
           6 0 60 1 51 7 0 61 1 51 10 0 62 1 51 0 0 63 1 6 0 51 64 2 6 0
           0 0 65 2 6 67 0 0 68 2 6 0 0 0 71 2 76 73 74 75 77 2 0 78 79
```

```
        80 81 1 76 82 75 83 1 0 84 80 85 1 76 82 75 86 1 0 84 80 87 1
        7 88 0 89 1 90 7 0 91 1 90 94 0 95 2 98 0 6 97 99 1 7 84 80
        100 2 103 98 101 102 104 1 0 88 0 105 0 7 0 106 2 107 0 10 7
        108 1 107 109 0 110 2 6 7 0 10 111 3 107 7 0 109 7 112 0 7 0
        117 0 0 0 118 1 7 119 0 120 0 75 0 121 1 0 119 0 122 1 6 7 0
        123 2 0 0 0 12 124 2 6 0 0 0 125 2 6 119 0 0 126 2 6 8 0 0 127
        2 0 128 0 0 129 0 75 0 130 2 7 0 10 0 131 3 0 0 0 132 0 133 2
        6 0 7 0 134 3 6 0 0 132 0 135 1 6 0 0 138 1 140 6 0 141 2 6
        142 0 142 143 1 140 6 0 144 2 140 0 0 0 145 2 0 142 142 142
        146 2 7 0 0 10 147 2 6 0 0 0 148 2 6 0 0 0 149 2 0 0 0 0 150
        2 0 151 0 0 152 2 6 119 0 0 153 2 140 0 6 6 154 2 0 155 142 0
        156 1 6 8 0 157 2 6 151 0 0 158 2 6 119 0 7 159 2 0 119 0 0
        160 0 140 0 161 1 140 0 6 162 2 140 0 0 109 163 2 140 0 0 0
        164 2 140 0 0 0 165 2 140 0 0 10 166 2 0 142 0 142 167 2 0 10
        0 0 168 1 169 98 6 170 1 0 88 0 171 1 169 6 6 172 1 0 0 0 173
        1 75 8 0 174 1 75 0 0 175 1 75 6 0 176 1 75 0 0 177 2 75 0 0
        0 178 2 75 0 6 0 179 2 0 80 80 80 180 1 181 82 75 182 1 0 84
        80 183 2 7 0 0 0 184 2 0 7 142 7 185 1 0 10 0 186 1 7 0 0 187
        2 7 0 0 0 188 2 0 67 0 0 189 1 190 0 109 191 1 190 0 0 192 2
        6 0 190 0 193 1 0 0 0 194 2 0 107 0 10 113 1 0 14 0 15 1 0 0
        51 66 2 0 10 0 14 18 1 0 84 80 183 1 0 0 0 173 1 0 88 0 171 2
        0 78 79 80 81 2 0 0 0 10 70 2 0 0 0 10 72 2 0 128 0 0 129 1 0
        115 0 116 1 0 7 0 114 2 0 0 0 0 150 2 0 151 0 0 152 2 0 10 0
        0 168 1 0 119 0 122 3 0 0 0 12 10 46 2 0 19 0 14 42 2 0 10 0
        12 41 1 0 51 0 58 1 0 34 0 39 2 0 67 0 10 69 1 0 0 0 194 0 0
        0 118 2 0 80 80 80 180 1 0 84 80 87 1 0 84 80 85 1 0 88 0 105
        3 0 0 0 12 0 25 3 0 0 0 14 22 23 3 0 0 0 14 27 28 3 0 0 0 12
        7 31 2 0 0 0 37 38 1 0 10 0 186 2 0 142 0 142 167 2 0 7 142 7
        185 2 0 142 142 142 146 2 0 67 0 0 189 3 0 0 0 132 0 133 2 0
        0 0 132 136 1 0 0 0 137 2 0 0 0 12 139 2 0 10 0 12 16 2 0 19
        0 14 20 2 0 0 0 12 124 2 0 119 0 0 160 2 0 155 142 0 156 1 0
        0 12 50))))))
(QUOTE |lookupComplete|)))
```

———-

21.69 URAGG.lsp BOOTSTRAP

URAGG depends on a chain of files. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **URAGG** category which we can write into the **MID** directory. We compile the lisp code and copy the **URAGG.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— URAGG.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(SETQ |UnaryRecursiveAggregate;CAT| (QUOTE NIL))

(SETQ |UnaryRecursiveAggregate;AL| (QUOTE NIL))

(DEFUN |UnaryRecursiveAggregate| (#1=#:G84596)
 (LET (#2=#:G84597)
  (COND
   ((SETQ #2# (|assoc| (|devaluate| #1#) |UnaryRecursiveAggregate;AL|))
     (CDR #2#))
   (T
    (SETQ |UnaryRecursiveAggregate;AL|
     (|cons5|
      (CONS (|devaluate| #1#) (SETQ #2# (|UnaryRecursiveAggregate;| #1#)))
      |UnaryRecursiveAggregate;AL|))
    #2#))))

(DEFUN |UnaryRecursiveAggregate;| (|t#1|)
 (PROG (#1=#:G84595)
  (RETURN
   (PROG1
    (LETT #1#
     (|sublisV|
      (PAIR (QUOTE (|t#1|)) (LIST (|devaluate| |t#1|)))
      (COND
       (|UnaryRecursiveAggregate;CAT|)
       ((QUOTE T)
        (LETT |UnaryRecursiveAggregate;CAT|
         (|Join|
          (|RecursiveAggregate| (QUOTE |t#1|))
          (|mkCategory|
           (QUOTE |domain|)
           (QUOTE (
            ((|concat| (|$| |$| |$|)) T)
            ((|concat| (|$| |t#1| |$|)) T)
            ((|first| (|t#1| |$|)) T)
            ((|elt| (|t#1| |$| "first")) T)
            ((|first| (|$| |$| (|NonNegativeInteger|))) T)
            ((|rest| (|$| |$|)) T)
            ((|elt| (|$| |$| "rest")) T)
            ((|rest| (|$| |$| (|NonNegativeInteger|))) T)
            ((|last| (|t#1| |$|)) T)
            ((|elt| (|t#1| |$| "last")) T)
            ((|last| (|$| |$| (|NonNegativeInteger|))) T)
            ((|tail| (|$| |$|)) T)
            ((|second| (|t#1| |$|)) T)
            ((|third| (|t#1| |$|)) T)
            ((|cycleEntry| (|$| |$|)) T)
```

```
            ((|cycleLength| ((|NonNegativeInteger|) |$|)) T)
            ((|cycleTail| (|$| |$|)) T)
            ((|concat!| (|$| |$| |$|))
              (|has| |$| (ATTRIBUTE |shallowlyMutable|)))
            ((|concat!| (|$| |$| |t#1|))
              (|has| |$| (ATTRIBUTE |shallowlyMutable|)))
            ((|cycleSplit!| (|$| |$|))
              (|has| |$| (ATTRIBUTE |shallowlyMutable|)))
            ((|setfirst!| (|t#1| |$| |t#1|))
              (|has| |$| (ATTRIBUTE |shallowlyMutable|)))
            ((|setelt| (|t#1| |$| "first" |t#1|))
              (|has| |$| (ATTRIBUTE |shallowlyMutable|)))
            ((|setrest!| (|$| |$| |$|))
              (|has| |$| (ATTRIBUTE |shallowlyMutable|)))
            ((|setelt| (|$| |$| "rest" |$|))
              (|has| |$| (ATTRIBUTE |shallowlyMutable|)))
            ((|setlast!| (|t#1| |$| |t#1|))
              (|has| |$| (ATTRIBUTE |shallowlyMutable|)))
            ((|setelt| (|t#1| |$| "last" |t#1|))
              (|has| |$| (ATTRIBUTE |shallowlyMutable|)))
            ((|split!| (|$| |$| (|Integer|)))
              (|has| |$| (ATTRIBUTE |shallowlyMutable|)))))
          NIL
          (QUOTE ((|Integer|) (|NonNegativeInteger|)))
          NIL))
        . #2=(|UnaryRecursiveAggregate|)))))
   . #2#)
  (SETELT #1# 0
   (LIST (QUOTE |UnaryRecursiveAggregate|) (|devaluate| |t#1|)))))))
```

———-

21.70 URAGG-.lsp BOOTSTRAP

URAGG- depends on **URAGG**. We need to break this cycle to build the algebra. So we keep a cached copy of the translated **URAGG-** category which we can write into the **MID** directory. We compile the lisp code and copy the **URAGG-.o** file to the **OUT** directory. This is eventually forcibly replaced by a recompiled version.

Note that this code is not included in the generated catdef.spad file.

— URAGG-.lsp BOOTSTRAP —

```
(|/VERSIONCHECK| 2)

(DEFUN |URAGG-;elt;AfirstS;1| (|x| G84610 |$|)
 (SPADCALL |x| (QREFELT |$| 8)))
```

```
(DEFUN |URAGG-;elt;AlastS;2| (|x| G84612 |$|)
 (SPADCALL |x| (QREFELT |$| 11)))

(DEFUN |URAGG-;elt;ArestA;3| (|x| G84614 |$|)
 (SPADCALL |x| (QREFELT |$| 14)))

(DEFUN |URAGG-;second;AS;4| (|x| |$|)
 (SPADCALL (SPADCALL |x| (QREFELT |$| 14)) (QREFELT |$| 8)))

(DEFUN |URAGG-;third;AS;5| (|x| |$|)
 (SPADCALL
  (SPADCALL (SPADCALL |x| (QREFELT |$| 14)) (QREFELT |$| 14))
  (QREFELT |$| 8)))

(DEFUN |URAGG-;cyclic?;AB;6| (|x| |$|)
 (COND
  ((OR
    (SPADCALL |x| (QREFELT |$| 20))
    (SPADCALL (|URAGG-;findCycle| |x| |$|) (QREFELT |$| 20)))
     (QUOTE NIL))
   ((QUOTE T) (QUOTE T))))

(DEFUN |URAGG-;last;AS;7| (|x| |$|)
 (SPADCALL (SPADCALL |x| (QREFELT |$| 22)) (QREFELT |$| 8)))

(DEFUN |URAGG-;nodes;AL;8| (|x| |$|)
 (PROG (|l|)
  (RETURN
   (SEQ
    (LETT |l| NIL |URAGG-;nodes;AL;8|)
    (SEQ
     G190
     (COND
      ((NULL
        (COND
          ((SPADCALL |x| (QREFELT |$| 20)) (QUOTE NIL))
          ((QUOTE T) (QUOTE T))))
        (GO G191)))
     (SEQ
      (LETT |l| (CONS |x| |l|) |URAGG-;nodes;AL;8|)
      (EXIT (LETT |x| (SPADCALL |x| (QREFELT |$| 14)) |URAGG-;nodes;AL;8|)))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT (NREVERSE |l|))))))

(DEFUN |URAGG-;children;AL;9| (|x| |$|)
 (PROG (|l|)
```

```
  (RETURN
   (SEQ
    (LETT |l| NIL |URAGG-;children;AL;9|)
    (EXIT
     (COND
      ((SPADCALL |x| (QREFELT |$| 20)) |l|)
      ((QUOTE T) (CONS (SPADCALL |x| (QREFELT |$| 14)) |l|)))))))

(DEFUN |URAGG-;leaf?;AB;10| (|x| |$|)
 (SPADCALL |x| (QREFELT |$| 20)))

(DEFUN |URAGG-;value;AS;11| (|x| |$|)
 (COND
  ((SPADCALL |x| (QREFELT |$| 20)) (|error| "value of empty object"))
  ((QUOTE T) (SPADCALL |x| (QREFELT |$| 8)))))

(DEFUN |URAGG-;less?;ANniB;12| (|l| |n| |$|)
 (PROG (|i|)
  (RETURN
   (SEQ
    (LETT |i| |n| |URAGG-;less?;ANniB;12|)
    (SEQ
     G190
     (COND
      ((NULL
        (COND
         ((|<| 0 |i|)
          (COND
           ((SPADCALL |l| (QREFELT |$| 20)) (QUOTE NIL))
           ((QUOTE T) (QUOTE T))))
         ((QUOTE T) (QUOTE NIL))))
       (GO G191)))
     (SEQ
      (LETT |l| (SPADCALL |l| (QREFELT |$| 14)) |URAGG-;less?;ANniB;12|)
      (EXIT (LETT |i| (|-| |i| 1) |URAGG-;less?;ANniB;12|)))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT (|<| 0 |i|)))))))

(DEFUN |URAGG-;more?;ANniB;13| (|l| |n| |$|)
 (PROG (|i|)
  (RETURN
   (SEQ
    (LETT |i| |n| |URAGG-;more?;ANniB;13|)
    (SEQ
     G190
     (COND
      ((NULL
```

```
       (COND
        ((|<| 0 |i|)
         (COND
          ((SPADCALL |l| (QREFELT |$| 20)) (QUOTE NIL))
          ((QUOTE T) (QUOTE T))))
        ((QUOTE T) (QUOTE NIL))))
       (GO G191)))
     (SEQ
      (LETT |l| (SPADCALL |l| (QREFELT |$| 14)) |URAGG-;more?;ANniB;13|)
      (EXIT (LETT |i| (|-| |i| 1) |URAGG-;more?;ANniB;13|)))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT
     (COND
      ((ZEROP |i|)
       (COND
        ((SPADCALL |l| (QREFELT |$| 20)) (QUOTE NIL))
        ((QUOTE T) (QUOTE T))))
      ((QUOTE T) (QUOTE NIL))))))))

(DEFUN |URAGG-;size?;ANniB;14| (|l| |n| |$|)
 (PROG (|i|)
  (RETURN
   (SEQ
    (LETT |i| |n| |URAGG-;size?;ANniB;14|)
    (SEQ
     G190
     (COND
      ((NULL
       (COND
        ((SPADCALL |l| (QREFELT |$| 20)) (QUOTE NIL))
        ((QUOTE T) (|<| 0 |i|))))
       (GO G191)))
     (SEQ
      (LETT |l| (SPADCALL |l| (QREFELT |$| 14)) |URAGG-;size?;ANniB;14|)
      (EXIT (LETT |i| (|-| |i| 1) |URAGG-;size?;ANniB;14|)))
     NIL
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT
     (COND
      ((SPADCALL |l| (QREFELT |$| 20)) (ZEROP |i|))
      ((QUOTE T) (QUOTE NIL))))))))

(DEFUN |URAGG-;#;ANni;15| (|x| |$|)
 (PROG (|k|)
  (RETURN
```

```
   (SEQ
    (SEQ
     (LETT |k| 0 |URAGG-;#;ANni;15|)
     G190
     (COND
      ((NULL
        (COND
         ((SPADCALL |x| (QREFELT |$| 20)) (QUOTE NIL))
         ((QUOTE T) (QUOTE T))))
       (GO G191)))
     (SEQ
      (COND
       ((EQL |k| 1000)
         (COND
          ((SPADCALL |x| (QREFELT |$| 33)) (EXIT (|error| "cyclic list"))))))
       (EXIT (LETT |x| (SPADCALL |x| (QREFELT |$| 14)) |URAGG-;#;ANni;15|)))
     (LETT |k| (QSADD1 |k|) |URAGG-;#;ANni;15|)
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT |k|)))))

(DEFUN |URAGG-;tail;2A;16| (|x| |$|)
 (PROG (|k| |y|)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |x| (QREFELT |$| 20)) (|error| "empty list"))
     ((QUOTE T)
      (SEQ
       (LETT |y| (SPADCALL |x| (QREFELT |$| 14)) |URAGG-;tail;2A;16|)
       (SEQ
        (LETT |k| 0 |URAGG-;tail;2A;16|)
        G190
        (COND
         ((NULL
           (COND
            ((SPADCALL |y| (QREFELT |$| 20)) (QUOTE NIL))
            ((QUOTE T) (QUOTE T))))
          (GO G191)))
        (SEQ
         (COND
          ((EQL |k| 1000)
            (COND
             ((SPADCALL |x| (QREFELT |$| 33))
               (EXIT (|error| "cyclic list"))))))
         (EXIT
          (LETT |y|
           (SPADCALL (LETT |x| |y| |URAGG-;tail;2A;16|) (QREFELT |$| 14))
           |URAGG-;tail;2A;16|)))
```

```
        (LETT |k| (QSADD1 |k|) |URAGG-;tail;2A;16|)
        (GO G190)
        G191
        (EXIT NIL))
       (EXIT |x|))))))))

(DEFUN |URAGG-;findCycle| (|x| |$|)
 (PROG (#1=#:G84667 |y|)
  (RETURN
   (SEQ
    (EXIT
     (SEQ
      (LETT |y| (SPADCALL |x| (QREFELT |$| 14)) |URAGG-;findCycle|)
      (SEQ
       G190
       (COND
        ((NULL
          (COND
           ((SPADCALL |y| (QREFELT |$| 20)) (QUOTE NIL))
           ((QUOTE T) (QUOTE T))))
         (GO G191)))
       (SEQ
        (COND
         ((SPADCALL |x| |y| (QREFELT |$| 36))
          (PROGN (LETT #1# |x| |URAGG-;findCycle|) (GO #1#))))
        (LETT |x| (SPADCALL |x| (QREFELT |$| 14)) |URAGG-;findCycle|)
        (LETT |y| (SPADCALL |y| (QREFELT |$| 14)) |URAGG-;findCycle|)
        (COND
         ((SPADCALL |y| (QREFELT |$| 20))
          (PROGN (LETT #1# |y| |URAGG-;findCycle|) (GO #1#))))
        (COND
         ((SPADCALL |x| |y| (QREFELT |$| 36))
          (PROGN (LETT #1# |y| |URAGG-;findCycle|) (GO #1#))))
        (EXIT (LETT |y| (SPADCALL |y| (QREFELT |$| 14)) |URAGG-;findCycle|)))
       NIL
       (GO G190)
       G191
       (EXIT NIL))
      (EXIT |y|)))
    #1#
    (EXIT #1#)))))

(DEFUN |URAGG-;cycleTail;2A;18| (|x| |$|)
 (PROG (|y| |z|)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL
       (LETT |y|
        (LETT |x| (SPADCALL |x| (QREFELT |$| 37)) |URAGG-;cycleTail;2A;18|)
```

```
        |URAGG-;cycleTail;2A;18|)
       (QREFELT |$| 20))
      |x|)
     ((QUOTE T)
       (SEQ
        (LETT |z| (SPADCALL |x| (QREFELT |$| 14)) |URAGG-;cycleTail;2A;18|)
        (SEQ
         G190
         (COND
          ((NULL
            (COND
             ((SPADCALL |x| |z| (QREFELT |$| 36)) (QUOTE NIL))
             ((QUOTE T) (QUOTE T))))
           (GO G191)))
         (SEQ
          (LETT |y| |z| |URAGG-;cycleTail;2A;18|)
          (EXIT
           (LETT |z|
            (SPADCALL |z| (QREFELT |$| 14)) |URAGG-;cycleTail;2A;18|)))
         NIL
         (GO G190)
         G191
         (EXIT NIL))
        (EXIT |y|)))))))

(DEFUN |URAGG-;cycleEntry;2A;19| (|x| |$|)
 (PROG (|l| |z| |k| |y|)
  (RETURN
   (SEQ
    (COND
     ((SPADCALL |x| (QREFELT |$| 20)) |x|)
     ((SPADCALL
       (LETT |y| (|URAGG-;findCycle| |x| |$|) |URAGG-;cycleEntry;2A;19|)
       (QREFELT |$| 20))
      |y|)
     ((QUOTE T)
      (SEQ
       (LETT |z| (SPADCALL |y| (QREFELT |$| 14)) |URAGG-;cycleEntry;2A;19|)
       (SEQ
        (LETT |l| 1 |URAGG-;cycleEntry;2A;19|)
        G190
        (COND
         ((NULL
           (COND
            ((SPADCALL |y| |z| (QREFELT |$| 36)) (QUOTE NIL))
            ((QUOTE T) (QUOTE T))))
          (GO G191)))
        (SEQ
         (EXIT
          (LETT |z|
```

```
          (SPADCALL |z| (QREFELT |$| 14)) |URAGG-;cycleEntry;2A;19|)))
        (LETT |l|
         (QSADD1 |l|) |URAGG-;cycleEntry;2A;19|) (GO G190) G191 (EXIT NIL))
       (LETT |y| |x| |URAGG-;cycleEntry;2A;19|)
       (SEQ
        (LETT |k| 1 |URAGG-;cycleEntry;2A;19|)
        G190
        (COND ((QSGREATERP |k| |l|) (GO G191)))
        (SEQ
         (EXIT
          (LETT |y|
           (SPADCALL |y| (QREFELT |$| 14)) |URAGG-;cycleEntry;2A;19|)))
        (LETT |k| (QSADD1 |k|) |URAGG-;cycleEntry;2A;19|)
        (GO G190)
        G191
        (EXIT NIL))
       (SEQ
        G190
        (COND
          ((NULL
            (COND
             ((SPADCALL |x| |y| (QREFELT |$| 36)) (QUOTE NIL))
             ((QUOTE T) (QUOTE T))))
           (GO G191)))
        (SEQ
         (LETT |x| (SPADCALL |x| (QREFELT |$| 14)) |URAGG-;cycleEntry;2A;19|)
         (EXIT
          (LETT |y|
           (SPADCALL |y| (QREFELT |$| 14)) |URAGG-;cycleEntry;2A;19|)))
        NIL
        (GO G190)
        G191
        (EXIT NIL))
       (EXIT |x|)))))))

(DEFUN |URAGG-;cycleLength;ANni;20| (|x| |$|)
 (PROG (|k| |y|)
  (RETURN
   (SEQ
    (COND
     ((OR
       (SPADCALL |x| (QREFELT |$| 20))
       (SPADCALL
        (LETT |x| (|URAGG-;findCycle| |x| |$|) |URAGG-;cycleLength;ANni;20|)
        (QREFELT |$| 20)))
      0)
     ((QUOTE T)
      (SEQ
       (LETT |y| (SPADCALL |x| (QREFELT |$| 14)) |URAGG-;cycleLength;ANni;20|)
       (SEQ
```

```
        (LETT |k| 1 |URAGG-;cycleLength;ANni;20|)
        G190
        (COND
         ((NULL
           (COND
            ((SPADCALL |x| |y| (QREFELT |$| 36)) (QUOTE NIL))
            ((QUOTE T) (QUOTE T))))
          (GO G191)))
        (SEQ
         (EXIT
          (LETT |y|
           (SPADCALL |y| (QREFELT |$| 14)) |URAGG-;cycleLength;ANni;20|)))
        (LETT |k| (QSADD1 |k|) |URAGG-;cycleLength;ANni;20|)
        (GO G190)
         G191
         (EXIT NIL))
       (EXIT |k|)))))))))

(DEFUN |URAGG-;rest;ANniA;21| (|x| |n| |$|)
 (PROG (|i|)
  (RETURN
   (SEQ
    (SEQ
     (LETT |i| 1 |URAGG-;rest;ANniA;21|)
     G190
     (COND ((QSGREATERP |i| |n|) (GO G191)))
     (SEQ
      (EXIT
       (COND
        ((SPADCALL |x| (QREFELT |$| 20)) (|error| "Index out of range"))
        ((QUOTE T)
          (LETT |x| (SPADCALL |x| (QREFELT |$| 14)) |URAGG-;rest;ANniA;21|)))))
     (LETT |i| (QSADD1 |i|) |URAGG-;rest;ANniA;21|)
     (GO G190)
     G191
     (EXIT NIL))
    (EXIT |x|)))))

(DEFUN |URAGG-;last;ANniA;22| (|x| |n| |$|)
 (PROG (|m| #1=#:G84694)
  (RETURN
   (SEQ
    (LETT |m| (SPADCALL |x| (QREFELT |$| 42)) |URAGG-;last;ANniA;22|)
    (EXIT
     (COND
      ((|<| |m| |n|) (|error| "index out of range"))
      ((QUOTE T)
       (SPADCALL
        (SPADCALL |x|
         (PROG1
```

```
          (LETT #1# (|-| |m| |n|) |URAGG-;last;ANniA;22|)
          (|check-subtype| (|>=| #1# 0) (QUOTE (|NonNegativeInteger|)) #1#))
         (QREFELT |$| 43))
        (QREFELT |$| 44)))))))))

(DEFUN |URAGG-;=;2AB;23| (|x| |y| |$|)
 (PROG (|k| #1=#:G84705)
  (RETURN
   (SEQ
    (EXIT
     (COND
      ((SPADCALL |x| |y| (QREFELT |$| 36)) (QUOTE T))
      ((QUOTE T)
       (SEQ
        (SEQ
         (LETT |k| 0 |URAGG-;=;2AB;23|)
         G190
         (COND
          ((NULL
            (COND
             ((OR
               (SPADCALL |x| (QREFELT |$| 20))
               (SPADCALL |y| (QREFELT |$| 20)))
              (QUOTE NIL))
             ((QUOTE T) (QUOTE T))))
           (GO G191)))
         (SEQ
          (COND
           ((EQL |k| 1000)
            (COND
             ((SPADCALL |x| (QREFELT |$| 33))
              (EXIT (|error| "cyclic list"))))))
          (COND
           ((NULL
             (SPADCALL
              (SPADCALL |x| (QREFELT |$| 8))
              (SPADCALL |y| (QREFELT |$| 8))
              (QREFELT |$| 46)))
            (EXIT (PROGN (LETT #1# (QUOTE NIL) |URAGG-;=;2AB;23|) (GO #1#)))))
          (LETT |x| (SPADCALL |x| (QREFELT |$| 14)) |URAGG-;=;2AB;23|)
          (EXIT (LETT |y| (SPADCALL |y| (QREFELT |$| 14)) |URAGG-;=;2AB;23|)))
         (LETT |k| (QSADD1 |k|) |URAGG-;=;2AB;23|)
         (GO G190)
         G191
         (EXIT NIL))
        (EXIT
         (COND
          ((SPADCALL |x| (QREFELT |$| 20)) (SPADCALL |y| (QREFELT |$| 20)))
          ((QUOTE T) (QUOTE NIL))))))))
    #1#
```

```
    (EXIT #1#)))))

(DEFUN |URAGG-;node?;2AB;24| (|u| |v| |$|)
 (PROG (|k| #1=#:G84711)
  (RETURN
   (SEQ
    (EXIT
     (SEQ
      (SEQ
       (LETT |k| 0 |URAGG-;node?;2AB;24|)
       G190
       (COND
         ((NULL
          (COND
           ((SPADCALL |v| (QREFELT |$| 20)) (QUOTE NIL))
           ((QUOTE T) (QUOTE T))))
         (GO G191)))
       (SEQ
        (EXIT
         (COND
          ((SPADCALL |u| |v| (QREFELT |$| 48))
            (PROGN (LETT #1# (QUOTE T) |URAGG-;node?;2AB;24|) (GO #1#)))
          ((QUOTE T)
           (SEQ
            (COND
             ((EQL |k| 1000)
              (COND
               ((SPADCALL |v| (QREFELT |$| 33))
                  (EXIT (|error| "cyclic list"))))))
            (EXIT
             (LETT |v|
              (SPADCALL |v| (QREFELT |$| 14))
              |URAGG-;node?;2AB;24|)))))))
       (LETT |k| (QSADD1 |k|) |URAGG-;node?;2AB;24|) (GO G190) G191 (EXIT NIL))
      (EXIT (SPADCALL |u| |v| (QREFELT |$| 48)))))
    #1# (EXIT #1#)))))

(DEFUN |URAGG-;setelt;Afirst2S;25| (|x| G84713 |a| |$|)
 (SPADCALL |x| |a| (QREFELT |$| 50)))

(DEFUN |URAGG-;setelt;Alast2S;26| (|x| G84715 |a| |$|)
 (SPADCALL |x| |a| (QREFELT |$| 52)))

(DEFUN |URAGG-;setelt;Arest2A;27| (|x| G84717 |a| |$|)
 (SPADCALL |x| |a| (QREFELT |$| 54)))

(DEFUN |URAGG-;concat;3A;28| (|x| |y| |$|)
 (SPADCALL (SPADCALL |x| (QREFELT |$| 44)) |y| (QREFELT |$| 56)))

(DEFUN |URAGG-;setlast!;A2S;29| (|x| |s| |$|)
```

```
 (SEQ
  (COND
   ((SPADCALL |x| (QREFELT |$| 20)) (|error| "setlast: empty list"))
   ((QUOTE T)
    (SEQ
     (SPADCALL (SPADCALL |x| (QREFELT |$| 22)) |s| (QREFELT |$| 50))
     (EXIT |s|))))))

(DEFUN |URAGG-;setchildren!;ALA;30| (|u| |lv| |$|)
 (COND
  ((EQL (LENGTH |lv|) 1) (SPADCALL |u| (|SPADfirst| |lv|) (QREFELT |$| 54)))
  ((QUOTE T) (|error| "wrong number of children specified"))))

(DEFUN |URAGG-;setvalue!;A2S;31| (|u| |s| |$|)
 (SPADCALL |u| |s| (QREFELT |$| 50)))

(DEFUN |URAGG-;split!;AIA;32| (|p| |n| |$|)
 (PROG (#1=#:G84725 |q|)
  (RETURN
   (SEQ
    (COND
     ((|<| |n| 1) (|error| "index out of range"))
     ((QUOTE T)
      (SEQ
       (LETT |p|
        (SPADCALL |p|
         (PROG1
          (LETT #1# (|-| |n| 1) |URAGG-;split!;AIA;32|)
          (|check-subtype| (|>=| #1# 0) (QUOTE (|NonNegativeInteger|)) #1#))
         (QREFELT |$| 43))
        |URAGG-;split!;AIA;32|)
       (LETT |q| (SPADCALL |p| (QREFELT |$| 14)) |URAGG-;split!;AIA;32|)
       (SPADCALL |p| (SPADCALL (QREFELT |$| 61)) (QREFELT |$| 54))
       (EXIT |q|))))))))

(DEFUN |URAGG-;cycleSplit!;2A;33| (|x| |$|)
 (PROG (|y| |z|)
  (RETURN
   (SEQ
    (COND
     ((OR
       (SPADCALL
        (LETT |y| (SPADCALL |x| (QREFELT |$| 37)) |URAGG-;cycleSplit!;2A;33|)
        (QREFELT |$| 20))
       (SPADCALL |x| |y| (QREFELT |$| 36))) |y|)
     ((QUOTE T)
      (SEQ
       (LETT |z| (SPADCALL |x| (QREFELT |$| 14)) |URAGG-;cycleSplit!;2A;33|)
       (SEQ G190
        (COND
```

```
         ((NULL
           (COND
            ((SPADCALL |z| |y| (QREFELT |$| 36)) (QUOTE NIL))
            ((QUOTE T) (QUOTE T))))
          (GO G191)))
        (SEQ
         (LETT |x| |z| |URAGG-;cycleSplit!;2A;33|)
         (EXIT
          (LETT |z|
           (SPADCALL |z| (QREFELT |$| 14)) |URAGG-;cycleSplit!;2A;33|)))
        NIL
        (GO G190)
        G191
        (EXIT NIL))
       (SPADCALL |x|
        (SPADCALL (QREFELT |$| 61)) (QREFELT |$| 54)) (EXIT |y|)))))))))

(DEFUN |UnaryRecursiveAggregate&| (|#1| |#2|)
 (PROG (|DV$1| |DV$2| |dv$| |$| |pv$|)
  (RETURN
   (PROGN
    (LETT |DV$1| (|devaluate| |#1|) . #1=(|UnaryRecursiveAggregate&|))
    (LETT |DV$2| (|devaluate| |#2|) . #1#)
    (LETT |dv$| (LIST (QUOTE |UnaryRecursiveAggregate&|) |DV$1| |DV$2|) . #1#)
    (LETT |$| (make-array 66) . #1#)
    (QSETREFV |$| 0 |dv$|)
    (QSETREFV |$| 3
     (LETT |pv$|
      (|buildPredVector| 0 0
       (LIST (|HasAttribute| |#1| (QUOTE |shallowlyMutable|))))
      . #1#))
    (|stuffDomainSlots| |$|)
    (QSETREFV |$| 6 |#1|)
    (QSETREFV |$| 7 |#2|)
    (COND
      ((|HasAttribute| |#1| (QUOTE |finiteAggregate|))
        (QSETREFV |$| 45
          (CONS (|dispatchFunction| |URAGG-;last;ANniA;22|) |$|))))
    (COND
     ((|HasCategory| |#2| (QUOTE (|SetCategory|)))
      (PROGN
       (QSETREFV |$| 47 (CONS (|dispatchFunction| |URAGG-;=;2AB;23|) |$|))
       (QSETREFV |$| 49
        (CONS (|dispatchFunction| |URAGG-;node?;2AB;24|) |$|)))))
    (COND
     ((|testBitVector| |pv$| 1)
      (PROGN
       (QSETREFV |$| 51
        (CONS (|dispatchFunction| |URAGG-;setelt;Afirst2S;25|) |$|))
       (QSETREFV |$| 53
```

```
        (CONS (|dispatchFunction| |URAGG-;setelt;Alast2S;26|) |$|))
       (QSETREFV |$| 55
        (CONS (|dispatchFunction| |URAGG-;setelt;Arest2A;27|) |$|))
       (QSETREFV |$| 57
        (CONS (|dispatchFunction| |URAGG-;concat;3A;28|) |$|))
       (QSETREFV |$| 58
        (CONS (|dispatchFunction| |URAGG-;setlast!;A2S;29|) |$|))
       (QSETREFV |$| 59
        (CONS (|dispatchFunction| |URAGG-;setchildren!;ALA;30|) |$|))
       (QSETREFV |$| 60
        (CONS (|dispatchFunction| |URAGG-;setvalue!;A2S;31|) |$|))
       (QSETREFV |$| 63
        (CONS (|dispatchFunction| |URAGG-;split!;AIA;32|) |$|))
       (QSETREFV |$| 64
        (CONS (|dispatchFunction| |URAGG-;cycleSplit!;2A;33|) |$|)))))
    |$|))))

(setf (get
 (QUOTE |UnaryRecursiveAggregate&|)
 (QUOTE |infovec|))
 (LIST
  (QUOTE #(NIL NIL NIL NIL NIL NIL (|local| |#1|) (|local| |#2|)
   (0 . |first|) (QUOTE "first") |URAGG-;elt;AfirstS;1| (5 . |last|)
   (QUOTE "last") |URAGG-;elt;AlastS;2| (10 . |rest|) (QUOTE "rest")
   |URAGG-;elt;ArestA;3| |URAGG-;second;AS;4| |URAGG-;third;AS;5|
   (|Boolean|) (15 . |empty?|) |URAGG-;cyclic?;AB;6| (20 . |tail|)
   |URAGG-;last;AS;7| (|List| |$|) |URAGG-;nodes;AL;8|
   |URAGG-;children;AL;9| |URAGG-;leaf?;AB;10| |URAGG-;value;AS;11|
   (|NonNegativeInteger|) |URAGG-;less?;ANniB;12| |URAGG-;more?;ANniB;13|
   |URAGG-;size?;ANniB;14| (25 . |cyclic?|) |URAGG-;#;ANni;15|
   |URAGG-;tail;2A;16| (30 . |eq?|) (36 . |cycleEntry|)
   |URAGG-;cycleTail;2A;18| |URAGG-;cycleEntry;2A;19|
   |URAGG-;cycleLength;ANni;20| |URAGG-;rest;ANniA;21| (41 . |#|)
   (46 . |rest|) (52 . |copy|) (57 . |last|) (63 . |=|) (69 . |=|)
   (75 . |=|) (81 . |node?|) (87 . |setfirst!|) (93 . |setelt|)
   (100 . |setlast!|) (106 . |setelt|) (113 . |setrest!|)
   (119 . |setelt|) (126 . |concat!|) (132 . |concat|) (138 . |setlast!|)
   (144 . |setchildren!|) (150 . |setvalue!|) (156 . |empty|) (|Integer|)
   (160 . |split!|) (166 . |cycleSplit!|) (QUOTE "value")))
  (QUOTE #(|value| 171 |third| 176 |tail| 181 |split!| 186 |size?| 192
   |setvalue!| 198 |setlast!| 204 |setelt| 210 |setchildren!| 231 |second|
   237 |rest| 242 |nodes| 248 |node?| 253 |more?| 259 |less?| 265 |leaf?|
   271 |last| 276 |elt| 287 |cyclic?| 305 |cycleTail| 310 |cycleSplit!|
   315 |cycleLength| 320 |cycleEntry| 325 |concat| 330 |children| 336 |=|
   341 |#| 347))
  (QUOTE NIL)
  (CONS
   (|makeByteWordVec2| 1 (QUOTE NIL))
   (CONS
    (QUOTE #())
```

```
   (CONS
    (QUOTE #())
    (|makeByteWordVec2| 64 (QUOTE (1 6 7 0 8 1 6 7 0 11 1 6 0 0 14 1 6
      19 0 20 1 6 0 0 22 1 6 19 0 33 2 6 19 0 0 36 1 6 0 0 37 1 6 29 0
      42 2 6 0 0 29 43 1 6 0 0 44 2 0 0 0 29 45 2 7 19 0 0 46 2 0 19 0
      0 47 2 6 19 0 0 48 2 0 19 0 0 49 2 6 7 0 7 50 3 0 7 0 9 7 51 2 6
      7 0 7 52 3 0 7 0 12 7 53 2 6 0 0 0 54 3 0 0 0 15 0 55 2 6 0 0 0 56
      2 0 0 0 0 57 2 0 7 0 7 58 2 0 0 0 24 59 2 0 7 0 7 60 0 6 0 61 2 0
      0 0 62 63 1 0 0 0 64 1 0 7 0 28 1 0 7 0 18 1 0 0 0 35 2 0 0 0 62 63
      2 0 19 0 29 32 2 0 7 0 7 60 2 0 7 0 7 58 3 0 7 0 12 7 53 3 0 0 0 15
      0 55 3 0 7 0 9 7 51 2 0 0 0 24 59 1 0 7 0 17 2 0 0 0 29 41 1 0 24 0
      25 2 0 19 0 0 49 2 0 19 0 29 31 2 0 19 0 29 30 1 0 19 0 27 2 0 0 0
      29 45 1 0 7 0 23 2 0 7 0 12 13 2 0 0 0 15 16 2 0 7 0 9 10 1 0 19 0
      21 1 0 0 0 38 1 0 0 0 64 1 0 29 0 40 1 0 0 0 39 2 0 0 0 0 57 1 0 24
      0 26 2 0 19 0 0 47 1 0 29 0 34))))))
  (QUOTE |lookupComplete|)))
```

———————-

Chapter 22

Chunk collections

— algebra —

```
\getchunk{category ABELGRP AbelianGroup}
\getchunk{category ABELMON AbelianMonoid}
\getchunk{category AMR AbelianMonoidRing}
\getchunk{category ABELSG AbelianSemiGroup}
\getchunk{category AGG Aggregate}
\getchunk{category ALGEBRA Algebra}
\getchunk{category ACF AlgebraicallyClosedField}
\getchunk{category ACFS AlgebraicallyClosedFunctionSpace}
\getchunk{category AHYP ArcHyperbolicFunctionCategory}
\getchunk{category ATRIG ArcTrigonometricFunctionCategory}
\getchunk{category ALAGG AssociationListAggregate}
\getchunk{category ATTREG AttributeRegistry}

\getchunk{category BGAGG BagAggregate}
\getchunk{category BASTYPE BasicType}
\getchunk{category BLMETCT BlowUpMethodCategory}
\getchunk{category BMODULE BiModule}
\getchunk{category BRAGG BinaryRecursiveAggregate}
\getchunk{category BTCAT BinaryTreeCategory}
\getchunk{category BTAGG BitAggregate}

\getchunk{category CACHSET CachableSet}
\getchunk{category CABMON CancellationAbelianMonoid}
\getchunk{category CHARNZ CharacteristicNonZero}
\getchunk{category CHARZ CharacteristicZero}
\getchunk{category KOERCE CoercibleTo}
\getchunk{category CLAGG Collection}
\getchunk{category CFCAT CombinatorialFunctionCategory}
\getchunk{category COMBOPC CombinatorialOpsCategory}
```

```
\getchunk{category COMRING CommutativeRing}
\getchunk{category COMPCAT ComplexCategory}
\getchunk{category KONVERT ConvertibleTo}

\getchunk{category DQAGG DequeueAggregate}
\getchunk{category DSTRCAT DesingTreeCategory}
\getchunk{category DIAGG Dictionary}
\getchunk{category DIOPS DictionaryOperations}
\getchunk{category DIFEXT DifferentialExtension}
\getchunk{category DPOLCAT DifferentialPolynomialCategory}
\getchunk{category DIFRING DifferentialRing}
\getchunk{category DIVCAT DivisorCategory}
\getchunk{category DVARCAT DifferentialVariableCategory}
\getchunk{category DIRPCAT DirectProductCategory}
\getchunk{category DIVRING DivisionRing}
\getchunk{category DLAGG DoublyLinkedAggregate}

\getchunk{category ELEMFUN ElementaryFunctionCategory}
\getchunk{category ELTAB Eltable}
\getchunk{category ELTAGG EltableAggregate}
\getchunk{category ENTIRER EntireRing}
\getchunk{category EUCDOM EuclideanDomain}
\getchunk{category EVALAB Evalable}
\getchunk{category ES ExpressionSpace}
\getchunk{category ELAGG ExtensibleLinearAggregate}
\getchunk{category XF ExtensionField}

\getchunk{category FIELD Field}
\getchunk{category FPC FieldOfPrimeCharacteristic}
\getchunk{category FILECAT FileCategory}
\getchunk{category FNCAT FileNameCategory}
\getchunk{category FINITE Finite}
\getchunk{category FAMR FiniteAbelianMonoidRing}
\getchunk{category FAXF FiniteAlgebraicExtensionField}
\getchunk{category FDIVCAT FiniteDivisorCategory}
\getchunk{category FFIELDC FiniteFieldCategory}
\getchunk{category FLAGG FiniteLinearAggregate}
\getchunk{category FINRALG FiniteRankAlgebra}
\getchunk{category FINAALG FiniteRankNonAssociativeAlgebra}
\getchunk{category FSAGG FiniteSetAggregate}
\getchunk{category FPS FloatingPointSystem}
\getchunk{category FORTFN FortranFunctionCategory}
\getchunk{category FMTC FortranMachineTypeCategory}
\getchunk{category FMC FortranMatrixCategory}
\getchunk{category FMFUN FortranMatrixFunctionCategory}
\getchunk{category FORTCAT FortranProgramCategory}
\getchunk{category FVC FortranVectorCategory}
\getchunk{category FVFUN FortranVectorFunctionCategory}
\getchunk{category FRAMALG FramedAlgebra}
\getchunk{category FRNAALG FramedNonAssociativeAlgebra}
```

```
\getchunk{category FAMONC FreeAbelianMonoidCategory}
\getchunk{category FLALG FreeLieAlgebra}
\getchunk{category FMCAT FreeModuleCat}
\getchunk{category FEVALAB FullyEvalableOver}
\getchunk{category FLINEXP FullyLinearlyExplicitRingOver}
\getchunk{category FPATMAB FullyPatternMatchable}
\getchunk{category FRETRCT FullyRetractableTo}
\getchunk{category FFCAT FunctionFieldCategory}
\getchunk{category FS FunctionSpace}

\getchunk{category GCDDOM GcdDomain}
\getchunk{category GRALG GradedAlgebra}
\getchunk{category GRMOD GradedModule}
\getchunk{category GROUP Group}

\getchunk{category HOAGG HomogeneousAggregate}
\getchunk{category HYPCAT HyperbolicFunctionCategory}

\getchunk{category IXAGG IndexedAggregate}
\getchunk{category IDPC IndexedDirectProductCategory}
\getchunk{category INFCLCT InfinitlyClosePointCategory}
\getchunk{category IEVALAB InnerEvalable}
\getchunk{category INS IntegerNumberSystem}
\getchunk{category INTDOM IntegralDomain}
\getchunk{category INTCAT IntervalCategory}

\getchunk{category KDAGG KeyedDictionary}

\getchunk{category LZSTAGG LazyStreamAggregate}
\getchunk{category LALG LeftAlgebra}
\getchunk{category LMODULE LeftModule}
\getchunk{category LIECAT LieAlgebra}
\getchunk{category LNAGG LinearAggregate}
\getchunk{category LINEXP LinearlyExplicitRingOver}
\getchunk{category LODOCAT LinearOrdinaryDifferentialOperatorCategory}
\getchunk{category LFCAT LiouvillianFunctionCategory}
\getchunk{category LSAGG ListAggregate}
\getchunk{category LOCPOWC LocalPowerSeriesCategory}
\getchunk{category LOGIC Logic}

\getchunk{category MATCAT MatrixCategory}
\getchunk{category MODULE Module}
\getchunk{category MONAD Monad}
\getchunk{category MONADWU MonadWithUnit}
\getchunk{category MONOGEN MonogenicAlgebra}
\getchunk{category MLO MonogenicLinearOperator}
\getchunk{category MONOID Monoid}
\getchunk{category MDAGG MultiDictionary}
\getchunk{category MSETAGG MultisetAggregate}
\getchunk{category MTSCAT MultivariateTaylorSeriesCategory}
```

```
\getchunk{category NAALG NonAssociativeAlgebra}
\getchunk{category NASRING NonAssociativeRing}
\getchunk{category NARNG NonAssociativeRng}
\getchunk{category NTSCAT NormalizedTriangularSetCategory}
\getchunk{category NUMINT NumericalIntegrationCategory}
\getchunk{category OPTCAT NumericalOptimizationCategory}

\getchunk{category OC OctonionCategory}
\getchunk{category A1AGG OneDimensionalArrayAggregate}
\getchunk{category OM OpenMath}
\getchunk{category OAGROUP OrderedAbelianGroup}
\getchunk{category OAMON OrderedAbelianMonoid}
\getchunk{category OAMONS OrderedAbelianMonoidSup}
\getchunk{category OASGP OrderedAbelianSemiGroup}
\getchunk{category OCAMON OrderedCancellationAbelianMonoid}
\getchunk{category ORDFIN OrderedFinite}
\getchunk{category OINTDOM OrderedIntegralDomain}
\getchunk{category ORDMON OrderedMonoid}
\getchunk{category OMSAGG OrderedMultisetAggregate}
\getchunk{category ORDRING OrderedRing}
\getchunk{category ORDSET OrderedSet}
\getchunk{category ODECAT OrdinaryDifferentialEquationsSolverCategory}

\getchunk{category PADICCT PAdicIntegerCategory}
\getchunk{category PDECAT PartialDifferentialEquationsSolverCategory}
\getchunk{category PDRING PartialDifferentialRing}
\getchunk{category PTRANFN PartialTranscendentalFunctions}
\getchunk{category PATAB Patternable}
\getchunk{category PATMAB PatternMatchable}
\getchunk{category PERMCAT PermutationCategory}
\getchunk{category PLACESC PlacesCategory}
\getchunk{category PPCURVE PlottablePlaneCurveCategory}
\getchunk{category PSCURVE PlottableSpaceCurveCategory}
\getchunk{category PTCAT PointCategory}
\getchunk{category POLYCAT PolynomialCategory}
\getchunk{category PFECAT PolynomialFactorizationExplicit}
\getchunk{category PSETCAT PolynomialSetCategory}
\getchunk{category PSCAT PowerSeriesCategory}
\getchunk{category PRIMCAT PrimitiveFunctionCategory}
\getchunk{category PID PrincipalIdealDomain}
\getchunk{category PRQAGG PriorityQueueAggregate}
\getchunk{category PRSPCAT ProjectiveSpaceCategory}
\getchunk{category PACEXTC PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory}
\getchunk{category PACFFC PseudoAlgebraicClosureOfFiniteFieldCategory}
\getchunk{category PACPERC PseudoAlgebraicClosureOfPerfectFieldCategory}
\getchunk{category PACRATC PseudoAlgebraicClosureOfRationalNumberCategory}

\getchunk{category QUATCAT QuaternionCategory}
\getchunk{category QUAGG QueueAggregate}
```

```
\getchunk{category QFCAT QuotientFieldCategory}

\getchunk{category RADCAT RadicalCategory}
\getchunk{category RCFIELD RealClosedField}
\getchunk{category REAL RealConstant}
\getchunk{category RNS RealNumberSystem}
\getchunk{category RRCC RealRootCharacterizationCategory}
\getchunk{category RMATCAT RectangularMatrixCategory}
\getchunk{category RCAGG RecursiveAggregate}
\getchunk{category RPOLCAT RecursivePolynomialCategory}
\getchunk{category RSETCAT RegularTriangularSetCategory}
\getchunk{category RETRACT RetractableTo}
\getchunk{category RMODULE RightModule}
\getchunk{category RING Ring}
\getchunk{category RNG Rng}

\getchunk{category SEGCAT SegmentCategory}
\getchunk{category SEGXCAT SegmentExpansionCategory}
\getchunk{category SGROUP SemiGroup}
\getchunk{category SETAGG SetAggregate}
\getchunk{category SETCAT SetCategory}
\getchunk{category SETCATD SetCategoryWithDegree}
\getchunk{category SEXCAT SExpressionCategory}
\getchunk{category SPFCAT SpecialFunctionCategory}
\getchunk{category SNTSCAT SquareFreeNormalizedTriangularSetCategory}
\getchunk{category SFRTCAT SquareFreeRegularTriangularSetCategory}
\getchunk{category SMATCAT SquareMatrixCategory}
\getchunk{category SKAGG StackAggregate}
\getchunk{category STEP StepThrough}
\getchunk{category STAGG StreamAggregate}
\getchunk{category SRAGG StringAggregate}
\getchunk{category STRICAT StringCategory}

\getchunk{category TBAGG TableAggregate}
\getchunk{category SPACEC ThreeSpaceCategory}
\getchunk{category TSETCAT TriangularSetCategory}
\getchunk{category TRIGCAT TrigonometricFunctionCategory}
\getchunk{category TRANFUN TranscendentalFunctionCategory}
\getchunk{category ARR2CAT TwoDimensionalArrayCategory}
\getchunk{category TYPE Type}

\getchunk{category URAGG UnaryRecursiveAggregate}
\getchunk{category UFD UniqueFactorizationDomain}
\getchunk{category ULSCAT UnivariateLaurentSeriesCategory}
\getchunk{category ULSCCAT UnivariateLaurentSeriesConstructorCategory}
\getchunk{category UPOLYC UnivariatePolynomialCategory}
\getchunk{category UPSCAT UnivariatePowerSeriesCategory}
\getchunk{category UPXSCAT UnivariatePuiseuxSeriesCategory}
\getchunk{category UPXSCCA UnivariatePuiseuxSeriesConstructorCategory}
\getchunk{category OREPCAT UnivariateSkewPolynomialCategory}
```

```
\getchunk{category UTSCAT UnivariateTaylorSeriesCategory}

\getchunk{category VECTCAT VectorCategory}
\getchunk{category VSPACE VectorSpace}

\getchunk{category XALG XAlgebra}
\getchunk{category XFALG XFreeAlgebra}
\getchunk{category XPOLYC XPolynomialsCat}
```

———-

— dotabb —

```
digraph dotabb {
 ranksep=1.25;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

\getchunk{CATEGORY.dotabb}
\getchunk{A1AGG.dotabb}
\getchunk{ABELGRP.dotabb}
\getchunk{ABELMON.dotabb}
\getchunk{ABELSG.dotabb}
\getchunk{ACF.dotabb}
\getchunk{ACFS.dotabb}
\getchunk{AGG.dotabb}
\getchunk{AHYP.dotabb}
\getchunk{ALAGG.dotabb}
\getchunk{ALGEBRA.dotabb}
\getchunk{AMR.dotabb}
\getchunk{ARR2CAT.dotabb}
\getchunk{ATRIG.dotabb}
\getchunk{ATTREG.dotabb}
\getchunk{BASTYPE.dotabb}
\getchunk{BGAGG.dotabb}
\getchunk{BLMETCT.dotabb}
\getchunk{BMODULE.dotabb}
\getchunk{BRAGG.dotabb}
\getchunk{BTAGG.dotabb}
\getchunk{BTCAT.dotabb}
\getchunk{CABMON.dotabb}
\getchunk{CACHSET.dotabb}
\getchunk{CFCAT.dotabb}
\getchunk{CHARNZ.dotabb}
\getchunk{CHARZ.dotabb}
\getchunk{CLAGG.dotabb}
\getchunk{COMBOPC.dotabb}
\getchunk{COMPCAT.dotabb}
\getchunk{COMRING.dotabb}
```

```
\getchunk{DIAGG.dotabb}
\getchunk{DIFEXT.dotabb}
\getchunk{DIFRING.dotabb}
\getchunk{DIOPS.dotabb}
\getchunk{DIRPCAT.dotabb}
\getchunk{DIVCAT.dotabb}
\getchunk{DIVRING.dotabb}
\getchunk{DLAGG.dotabb}
\getchunk{DPOLCAT.dotabb}
\getchunk{DQAGG.dotabb}
\getchunk{DSTRCAT.dotabb}
\getchunk{DVARCAT.dotabb}
\getchunk{ELAGG.dotabb}
\getchunk{ELEMFUN.dotabb}
\getchunk{ELTAB.dotabb}
\getchunk{ELTAGG.dotabb}
\getchunk{ENTIRER.dotabb}
\getchunk{ES.dotabb}
\getchunk{EUCDOM.dotabb}
\getchunk{EVALAB.dotabb}
\getchunk{FAMONC.dotabb}
\getchunk{FAMR.dotabb}
\getchunk{FAXF.dotabb}
\getchunk{FDIVCAT.dotabb}
\getchunk{FEVALAB.dotabb}
\getchunk{FFCAT.dotabb}
\getchunk{FFIELDC.dotabb}
\getchunk{FIELD.dotabb}
\getchunk{FILECAT.dotabb}
\getchunk{FINAALG.dotabb}
\getchunk{FINITE.dotabb}
\getchunk{FINRALG.dotabb}
\getchunk{FLAGG.dotabb}
\getchunk{FLALG.dotabb}
\getchunk{FLINEXP.dotabb}
\getchunk{FMC.dotabb}
\getchunk{FMCAT.dotabb}
\getchunk{FMFUN.dotabb}
\getchunk{FMTC.dotabb}
\getchunk{FNCAT.dotabb}
\getchunk{FORTCAT.dotabb}
\getchunk{FORTFN.dotabb}
\getchunk{FPATMAB.dotabb}
\getchunk{FPC.dotabb}
\getchunk{FPS.dotabb}
\getchunk{FRAMALG.dotabb}
\getchunk{FRETRCT.dotabb}
\getchunk{FRNAALG.dotabb}
\getchunk{FS.dotabb}
\getchunk{FSAGG.dotabb}
```

```
\getchunk{FVC.dotabb}
\getchunk{FVFUN.dotabb}
\getchunk{GCDDOM.dotabb}
\getchunk{GRALG.dotabb}
\getchunk{GRMOD.dotabb}
\getchunk{GROUP.dotabb}
\getchunk{HOAGG.dotabb}
\getchunk{HYPCAT.dotabb}
\getchunk{IDPC.dotabb}
\getchunk{INFCLCT.dotabb}
\getchunk{IEVALAB.dotabb}
\getchunk{INS.dotabb}
\getchunk{INTCAT.dotabb}
\getchunk{INTDOM.dotabb}
\getchunk{IXAGG.dotabb}
\getchunk{KDAGG.dotabb}
\getchunk{KOERCE.dotabb}
\getchunk{KONVERT.dotabb}
\getchunk{LALG.dotabb}
\getchunk{LFCAT.dotabb}
\getchunk{LIECAT.dotabb}
\getchunk{LINEXP.dotabb}
\getchunk{LMODULE.dotabb}
\getchunk{LNAGG.dotabb}
\getchunk{LODOCAT.dotabb}
\getchunk{LOGIC.dotabb}
\getchunk{LSAGG.dotabb}
\getchunk{LZSTAGG.dotabb}
\getchunk{MATCAT.dotabb}
\getchunk{MDAGG.dotabb}
\getchunk{MLO.dotabb}
\getchunk{MODULE.dotabb}
\getchunk{MONAD.dotabb}
\getchunk{MONADWU.dotabb}
\getchunk{MONOGEN.dotabb}
\getchunk{MONOID.dotabb}
\getchunk{MSETAGG.dotabb}
\getchunk{MTSCAT.dotabb}
\getchunk{NAALG.dotabb}
\getchunk{NARNG.dotabb}
\getchunk{NASRING.dotabb}
\getchunk{NTSCAT.dotabb}
\getchunk{NUMINT.dotabb}
\getchunk{OAGROUP.dotabb}
\getchunk{OAMON.dotabb}
\getchunk{OAMONS.dotabb}
\getchunk{OASGP.dotabb}
\getchunk{OC.dotabb}
\getchunk{OCAMON.dotabb}
\getchunk{ODECAT.dotabb}
```

```
\getchunk{OINTDOM.dotabb}
\getchunk{OM.dotabb}
\getchunk{OMSAGG.dotabb}
\getchunk{OPTCAT.dotabb}
\getchunk{ORDFIN.dotabb}
\getchunk{ORDMON.dotabb}
\getchunk{ORDRING.dotabb}
\getchunk{ORDSET.dotabb}
\getchunk{OREPCAT.dotabb}
\getchunk{PACFFC.dotabb}
\getchunk{PACPERC.dotabb}
\getchunk{PACRATC.dotabb}
\getchunk{PADICCT.dotabb}
\getchunk{PATAB.dotabb}
\getchunk{PATMAB.dotabb}
\getchunk{PDECAT.dotabb}
\getchunk{PDRING.dotabb}
\getchunk{PERMCAT.dotabb}
\getchunk{PFECAT.dotabb}
\getchunk{PID.dotabb}
\getchunk{PLACESC.dotabb}
\getchunk{POLYCAT.dotabb}
\getchunk{PPCURVE.dotabb}
\getchunk{PRIMCAT.dotabb}
\getchunk{PRQAGG.dotabb}
\getchunk{PRSPCAT.dotabb}
\getchunk{PSCAT.dotabb}
\getchunk{PSCURVE.dotabb}
\getchunk{PSETCAT.dotabb}
\getchunk{PTCAT.dotabb}
\getchunk{PTRANFN.dotabb}
\getchunk{QFCAT.dotabb}
\getchunk{QUAGG.dotabb}
\getchunk{QUATCAT.dotabb}
\getchunk{RADCAT.dotabb}
\getchunk{RCAGG.dotabb}
\getchunk{RCFIELD.dotabb}
\getchunk{REAL.dotabb}
\getchunk{RETRACT.dotabb}
\getchunk{RING.dotabb}
\getchunk{RMATCAT.dotabb}
\getchunk{RMODULE.dotabb}
\getchunk{RNG.dotabb}
\getchunk{RNS.dotabb}
\getchunk{RPOLCAT.dotabb}
\getchunk{RRCC.dotabb}
\getchunk{RSETCAT.dotabb}
\getchunk{SEGCAT.dotabb}
\getchunk{SEGXCAT.dotabb}
\getchunk{SETAGG.dotabb}
```

```
\getchunk{SETCAT.dotabb}
\getchunk{SETCATD.dotabb}
\getchunk{SEXCAT.dotabb}
\getchunk{SFRTCAT.dotabb}
\getchunk{SGROUP.dotabb}
\getchunk{SKAGG.dotabb}
\getchunk{SMATCAT.dotabb}
\getchunk{SNTSCAT.dotabb}
\getchunk{SPACEC.dotabb}
\getchunk{SPFCAT.dotabb}
\getchunk{SRAGG.dotabb}
\getchunk{STAGG.dotabb}
\getchunk{STEP.dotabb}
\getchunk{STRICAT.dotabb}
\getchunk{TBAGG.dotabb}
\getchunk{TRANFUN.dotabb}
\getchunk{TRIGCAT.dotabb}
\getchunk{TSETCAT.dotabb}
\getchunk{TYPE.dotabb}
\getchunk{UFD.dotabb}
\getchunk{ULSCAT.dotabb}
\getchunk{ULSCCAT.dotabb}
\getchunk{UPOLYC.dotabb}
\getchunk{UPSCAT.dotabb}
\getchunk{UPXSCAT.dotabb}
\getchunk{UPXSCCA.dotabb}
\getchunk{URAGG.dotabb}
\getchunk{UTSCAT.dotabb}
\getchunk{VECTCAT.dotabb}
\getchunk{VSPACE.dotabb}
\getchunk{XALG.dotabb}
\getchunk{XF.dotabb}
\getchunk{XFALG.dotabb}
\getchunk{XPOLYC.dotabb}
}
```

———-

— dotfull —

```
digraph dotfull {
 ranksep=1.25;
 nodesep=1.5;
 fontsize=10;
 bgcolor="#ECEA81";
 node [shape=box, color=white, style=filled];

\getchunk{CATEGORY.dotfull}
\getchunk{A1AGG.dotfull}
```

```
\getchunk{ABELGRP.dotfull}
\getchunk{ABELMON.dotfull}
\getchunk{ABELSG.dotfull}
\getchunk{ACF.dotfull}
\getchunk{ACFS.dotfull}
\getchunk{AGG.dotfull}
\getchunk{AHYP.dotfull}
\getchunk{ALAGG.dotfull}
\getchunk{ALGEBRA.dotfull}
\getchunk{AMR.dotfull}
\getchunk{ARR2CAT.dotfull}
\getchunk{ATRIG.dotfull}
\getchunk{ATTREG.dotfull}
\getchunk{BASTYPE.dotfull}
\getchunk{BGAGG.dotfull}
\getchunk{BLMETCT.dotfull}
\getchunk{BMODULE.dotfull}
\getchunk{BRAGG.dotfull}
\getchunk{BTAGG.dotfull}
\getchunk{BTCAT.dotfull}
\getchunk{CABMON.dotfull}
\getchunk{CACHSET.dotfull}
\getchunk{CFCAT.dotfull}
\getchunk{CHARNZ.dotfull}
\getchunk{CHARZ.dotfull}
\getchunk{CLAGG.dotfull}
\getchunk{COMBOPC.dotfull}
\getchunk{COMPCAT.dotfull}
\getchunk{COMRING.dotfull}
\getchunk{DIAGG.dotfull}
\getchunk{DIFEXT.dotfull}
\getchunk{DIFRING.dotfull}
\getchunk{DIOPS.dotfull}
\getchunk{DIRPCAT.dotfull}
\getchunk{DIVCAT.dotfull}
\getchunk{DIVRING.dotfull}
\getchunk{DLAGG.dotfull}
\getchunk{DPOLCAT.dotfull}
\getchunk{DQAGG.dotfull}
\getchunk{DSTRCAT.dotfull}
\getchunk{DVARCAT.dotfull}
\getchunk{ELAGG.dotfull}
\getchunk{ELEMFUN.dotfull}
\getchunk{ELTAB.dotfull}
\getchunk{ELTAGG.dotfull}
\getchunk{ENTIRER.dotfull}
\getchunk{ES.dotfull}
\getchunk{EUCDOM.dotfull}
\getchunk{EVALAB.dotfull}
\getchunk{FAMONC.dotfull}
```

```
\getchunk{FAMR.dotfull}
\getchunk{FAXF.dotfull}
\getchunk{FDIVCAT.dotfull}
\getchunk{FEVALAB.dotfull}
\getchunk{FFCAT.dotfull}
\getchunk{FFIELDC.dotfull}
\getchunk{FIELD.dotfull}
\getchunk{FILECAT.dotfull}
\getchunk{FINAALG.dotfull}
\getchunk{FINITE.dotfull}
\getchunk{FINRALG.dotfull}
\getchunk{FLAGG.dotfull}
\getchunk{FLALG.dotfull}
\getchunk{FLINEXP.dotfull}
\getchunk{FMC.dotfull}
\getchunk{FMCAT.dotfull}
\getchunk{FMFUN.dotfull}
\getchunk{FMTC.dotfull}
\getchunk{FNCAT.dotfull}
\getchunk{FORTCAT.dotfull}
\getchunk{FORTFN.dotfull}
\getchunk{FPATMAB.dotfull}
\getchunk{FPC.dotfull}
\getchunk{FPS.dotfull}
\getchunk{FRAMALG.dotfull}
\getchunk{FRETRCT.dotfull}
\getchunk{FRNAALG.dotfull}
\getchunk{FS.dotfull}
\getchunk{FSAGG.dotfull}
\getchunk{FVC.dotfull}
\getchunk{FVFUN.dotfull}
\getchunk{GCDDOM.dotfull}
\getchunk{GRALG.dotfull}
\getchunk{GRMOD.dotfull}
\getchunk{GROUP.dotfull}
\getchunk{HOAGG.dotfull}
\getchunk{HYPCAT.dotfull}
\getchunk{IDPC.dotfull}
\getchunk{INFCLCT.dotfull}
\getchunk{IEVALAB.dotfull}
\getchunk{INS.dotfull}
\getchunk{INTCAT.dotfull}
\getchunk{INTDOM.dotfull}
\getchunk{IXAGG.dotfull}
\getchunk{KDAGG.dotfull}
\getchunk{KOERCE.dotfull}
\getchunk{KONVERT.dotfull}
\getchunk{LALG.dotfull}
\getchunk{LFCAT.dotfull}
\getchunk{LIECAT.dotfull}
```

```
\getchunk{LINEXP.dotfull}
\getchunk{LMODULE.dotfull}
\getchunk{LNAGG.dotfull}
\getchunk{LODOCAT.dotfull}
\getchunk{LOGIC.dotfull}
\getchunk{LSAGG.dotfull}
\getchunk{LZSTAGG.dotfull}
\getchunk{MATCAT.dotfull}
\getchunk{MDAGG.dotfull}
\getchunk{MLO.dotfull}
\getchunk{MODULE.dotfull}
\getchunk{MONAD.dotfull}
\getchunk{MONADWU.dotfull}
\getchunk{MONOGEN.dotfull}
\getchunk{MONOID.dotfull}
\getchunk{MSETAGG.dotfull}
\getchunk{MTSCAT.dotfull}
\getchunk{NAALG.dotfull}
\getchunk{NARNG.dotfull}
\getchunk{NASRING.dotfull}
\getchunk{NTSCAT.dotfull}
\getchunk{NUMINT.dotfull}
\getchunk{OAGROUP.dotfull}
\getchunk{OAMON.dotfull}
\getchunk{OAMONS.dotfull}
\getchunk{OASGP.dotfull}
\getchunk{OC.dotfull}
\getchunk{OCAMON.dotfull}
\getchunk{ODECAT.dotfull}
\getchunk{OINTDOM.dotfull}
\getchunk{OM.dotfull}
\getchunk{OMSAGG.dotfull}
\getchunk{OPTCAT.dotfull}
\getchunk{ORDFIN.dotfull}
\getchunk{ORDMON.dotfull}
\getchunk{ORDRING.dotfull}
\getchunk{ORDSET.dotfull}
\getchunk{OREPCAT.dotfull}
\getchunk{PACFFC.dotfull}
\getchunk{PACPERC.dotfull}
\getchunk{PACRATC.dotfull}
\getchunk{PADICCT.dotfull}
\getchunk{PATAB.dotfull}
\getchunk{PATMAB.dotfull}
\getchunk{PDECAT.dotfull}
\getchunk{PDRING.dotfull}
\getchunk{PERMCAT.dotfull}
\getchunk{PFECAT.dotfull}
\getchunk{PID.dotfull}
\getchunk{PLACESC.dotfull}
```

```
\getchunk{POLYCAT.dotfull}
\getchunk{PPCURVE.dotfull}
\getchunk{PRIMCAT.dotfull}
\getchunk{PRQAGG.dotfull}
\getchunk{PRSPCAT.dotfull}
\getchunk{PSCAT.dotfull}
\getchunk{PSCURVE.dotfull}
\getchunk{PSETCAT.dotfull}
\getchunk{PTCAT.dotfull}
\getchunk{PTRANFN.dotfull}
\getchunk{QFCAT.dotfull}
\getchunk{QUAGG.dotfull}
\getchunk{QUATCAT.dotfull}
\getchunk{RADCAT.dotfull}
\getchunk{RCAGG.dotfull}
\getchunk{RCFIELD.dotfull}
\getchunk{REAL.dotfull}
\getchunk{RETRACT.dotfull}
\getchunk{RING.dotfull}
\getchunk{RMATCAT.dotfull}
\getchunk{RMODULE.dotfull}
\getchunk{RNG.dotfull}
\getchunk{RNS.dotfull}
\getchunk{RPOLCAT.dotfull}
\getchunk{RRCC.dotfull}
\getchunk{RSETCAT.dotfull}
\getchunk{SEGCAT.dotfull}
\getchunk{SEGXCAT.dotfull}
\getchunk{SETAGG.dotfull}
\getchunk{SETCAT.dotfull}
\getchunk{SETCATD.dotfull}
\getchunk{SEXCAT.dotfull}
\getchunk{SFRTCAT.dotfull}
\getchunk{SGROUP.dotfull}
\getchunk{SKAGG.dotfull}
\getchunk{SMATCAT.dotfull}
\getchunk{SNTSCAT.dotfull}
\getchunk{SPACEC.dotfull}
\getchunk{SPFCAT.dotfull}
\getchunk{SRAGG.dotfull}
\getchunk{STAGG.dotfull}
\getchunk{STEP.dotfull}
\getchunk{STRICAT.dotfull}
\getchunk{TBAGG.dotfull}
\getchunk{TRANFUN.dotfull}
\getchunk{TRIGCAT.dotfull}
\getchunk{TSETCAT.dotfull}
\getchunk{TYPE.dotfull}
\getchunk{UFD.dotfull}
\getchunk{ULSCAT.dotfull}
```

```
\getchunk{ULSCCAT.dotfull}
\getchunk{UPOLYC.dotfull}
\getchunk{UPSCAT.dotfull}
\getchunk{UPXSCAT.dotfull}
\getchunk{UPXSCCA.dotfull}
\getchunk{URAGG.dotfull}
\getchunk{UTSCAT.dotfull}
\getchunk{VECTCAT.dotfull}
\getchunk{VSPACE.dotfull}
\getchunk{XALG.dotfull}
\getchunk{XF.dotfull}
\getchunk{XFALG.dotfull}
\getchunk{XPOLYC.dotfull}
}
```

———-

Bibliography

[1] N. Jacobson: Structure and Representations of Jordan Algebras AMS, Providence, 1968

[2] MacLane and Birkhoff, Algebra 2d Edition, MacMillan 1979

[3] Encyclopedic Dictionary of Mathematics, MIT Press, 1977

[4] R.D. Schafer: An Introduction to Nonassociative Algebras Academic Press, New York, 1966

[5] R. Wisbauer: Bimodule Structure of Algebra Lecture Notes Univ. Duesseldorf 1991

[6] J. Grabmeier, A. Scheerhorn: Finite Fields in AXIOM. AXIOM Technical Report Series, ATR/5 NP2522.

[7] R. Rioboo, *Real Algebraic Closure of an ordered Field : Implementation in Axiom.*, In proceedings of the ISSAC'92 Conference, Berkeley 1992 pp. 206-215.

[8] Z. Ligatsikas, R. Rioboo, M. F. Roy *Generic computation of the real closure of an ordered field.*, In Mathematics and Computers in Simulation Volume 42, Issue 4-6, November 1996.

[9] D. LAZARD "A new method for solving algebraic systems of positive dimension" Discr. App. Math. 33:147-160,1991

[10] P. AUBRY, D. LAZARD and M. MORENO MAZA "On the Theories of Triangular Sets" Journal of Symbol. Comp. (to appear)

[11] M. MORENO MAZA and R. RIOBOO "Computations of gcd over algebraic towers of simple extensions" In proceedings of AAECC11 Paris, 1995.

[12] M. MORENO MAZA "Calculs de pgcd au-dessus des tours d'extensions simples et resolution des systemes d'equations algebriques" These, Universite P.etM. Curie, Paris, 1997.

Index

-?
ABELGRP, 522
ACF, 1298
ACFS, 1500
ALGEBRA, 949
AMR, 1103
BMODULE, 728
CHARNZ, 835
CHARZ, 840
COMPCAT, 1625
COMRING, 845
DIFEXT, 955
DIFRING, 850
DIRPCAT, 1004
DIVCAT, 1181
DIVRING, 1014
DPOLCAT, 1310
ENTIRER, 855
EUCDOM, 1187
FAMR, 1143
FAXF, 1587
FDIVCAT, 618
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FINAALG, 1022
FINRALG, 1331
FLALG, 1043
FLINEXP, 962
FMCAT, 859
FMTC, 1110
FPC, 1325
FPS, 1532
FRAMALG, 1541
FRNAALG, 1118
FS, 1341
GCDDOM, 1131
GRALG, 314
GRMOD, 177
INS, 1231
INTCAT, 1153
INTDOM, 1049
LALG, 865
LIECAT, 968
LINEXP, 871
LMODULE, 657
LOCPOWC, 1243
LODOCAT, 973
MATCAT, 437
MLO, 1055
MODULE, 876
MONOGEN, 1602
MTSCAT, 1196
NAALG, 982
NARNG, 681
NASRING, 742
OAGROUP, 756
OC, 1062
OINTDOM, 1137
ORDRING, 881
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PDRING, 887
PFECAT, 1204
PID, 1168
PLACESC, 341
POLYCAT, 1260
PSCAT, 1161
PTCAT, 896
QFCAT, 1380

QUATCAT, 1073
RCFIELD, 1392
RING, 772
RMATCAT, 903
RMODULE, 716
RNG, 720
RNS, 1403
RPOLCAT, 1414
SMATCAT, 1085
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
VECTCAT, 811
VSPACE, 989
XALG, 942
XF, 1513
XFALG, 994
XPOLYC, 1096
?/Γ*E*30*F*?
BTAGG, 734
LOGIC, 96
?<?
A1AGG, 687
ACFS, 1500
ALAGG, 824
BTAGG, 734
CACHESET, 284
COMPCAT, 1625
DIRPCAT, 1004
DPOLCAT, 1310
DVARCAT, 296
ES, 302
FLAGG, 558
FMTC, 1110
FPS, 1532
FS, 1341
FSAGG, 624
INS, 1231
INTCAT, 1153
LSAGG, 663
MSETAGG, 675
OAGROUP, 756
OAMON, 577
OAMONS, 760
OASGP, 458
OC, 1062
OCAMON, 697
OINTDOM, 1137
OMSAGG, 765
ORDFIN, 337
ORDMON, 462
ORDRING, 881
ORDSET, 218
PERMCAT, 582
POLYCAT, 1260
PTCAT, 896
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1392
RNS, 1403
RPOLCAT, 1414
SETAGG, 494
SRAGG, 789
STRICAT, 923
ULSCCAT, 1559
UPOLYC, 1477
VECTCAT, 811
?<=?
A1AGG, 687
ACFS, 1500
ALAGG, 824
BTAGG, 734
CACHESET, 284
COMPCAT, 1625
DIRPCAT, 1004
DIVCAT, 1181
DPOLCAT, 1310
DVARCAT, 296
ES, 302
FLAGG, 558
FMTC, 1110
FPS, 1532
FS, 1341
INS, 1231
INTCAT, 1153
LSAGG, 663
OAGROUP, 756

OAMON, 577
OAMONS, 760
OASGP, 458
OC, 1062
OCAMON, 697
OINTDOM, 1137
ORDFIN, 337
ORDMON, 462
ORDRING, 881
ORDSET, 218
PERMCAT, 582
POLYCAT, 1260
PTCAT, 896
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1392
RNS, 1403
RPOLCAT, 1414
SRAGG, 789
STRICAT, 923
ULSCCAT, 1559
UPOLYC, 1477
VECTCAT, 811
?>?
A1AGG, 687
ACFS, 1500
ALAGG, 824
BTAGG, 734
CACHESET, 284
COMPCAT, 1625
DIRPCAT, 1004
DPOLCAT, 1310
DVARCAT, 296
ES, 302
FLAGG, 558
FMTC, 1110
FPS, 1532
FS, 1341
INS, 1231
INTCAT, 1153
LSAGG, 663
OAGROUP, 756
OAMON, 577
OAMONS, 760
OASGP, 458
OC, 1062
OCAMON, 697
OINTDOM, 1137
ORDFIN, 337
ORDMON, 462
ORDRING, 881
ORDSET, 218
PERMCAT, 582
POLYCAT, 1260
PTCAT, 896
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1392
RNS, 1403
RPOLCAT, 1414
SRAGG, 789
STRICAT, 923
ULSCCAT, 1559
UPOLYC, 1477
VECTCAT, 811
?>=?
A1AGG, 687
ACFS, 1500
ALAGG, 824
BTAGG, 734
CACHESET, 284
COMPCAT, 1625
DIRPCAT, 1004
DPOLCAT, 1310
DVARCAT, 296
ES, 302
FLAGG, 558
FMTC, 1110
FPS, 1532
FS, 1341
INS, 1231
INTCAT, 1153
LSAGG, 663
OAGROUP, 756
OAMON, 577
OAMONS, 760
OASGP, 458
OC, 1062
OCAMON, 697
OINTDOM, 1137
ORDFIN, 337
ORDMON, 462

ORDRING, 881
ORDSET, 218
PERMCAT, 582
POLYCAT, 1260
PTCAT, 896
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1392
RNS, 1403
RPOLCAT, 1414
SRAGG, 789
STRICAT, 923
ULSCCAT, 1559
UPOLYC, 1477
VECTCAT, 811
?T*E*30*F*/?
BTAGG, 734
LOGIC, 96
?**?
ACF, 1298
ACFS, 1500
ALGEBRA, 949
AMR, 1103
CHARNZ, 835
CHARZ, 840
COMPCAT, 1625
COMRING, 845
DIFEXT, 955
DIFRING, 850
DIRPCAT, 1004
DIVRING, 1014
DPOLCAT, 1310
ELEMFUN, 28
ENTIRER, 855
EUCDOM, 1187
FAMR, 1143
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FINAALG, 1022
FINRALG, 1331
FLINEXP, 962
FMTC, 1110
FPC, 1325
FPS, 1532
FRAMALG, 1541
FRNAALG, 1118
FS, 1341
GCDDOM, 1131
GROUP, 400
INS, 1231
INTCAT, 1153
INTDOM, 1049
LALG, 865
LFCAT, 193
LINEXP, 871
LOCPOWC, 1243
LODOCAT, 973
MATCAT, 437
MLO, 1055
MONAD, 198
MONADWU, 327
MONOGEN, 1602
MONOID, 332
MTSCAT, 1196
NAALG, 982
NARNG, 681
NASRING, 742
OC, 1062
OINTDOM, 1137
ORDMON, 462
ORDRING, 881
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PDRING, 887
PERMCAT, 582
PFECAT, 1204
PID, 1168
POLYCAT, 1260
PSCAT, 1161
QFCAT, 1380
QUATCAT, 1073
RADCAT, 53
RCFIELD, 1392
RING, 772
RNG, 720
RNS, 1403

RPOLCAT, 1414
SGROUP, 240
SMATCAT, 1085
TRANFUN, 118
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
XALG, 942
XF, 1513
XFALG, 994
XPOLYC, 1096
?*?
ABELGRP, 522
ABELMON, 269
ABELSG, 124
ACF, 1298
ACFS, 1500
ALGEBRA, 949
AMR, 1103
BMODULE, 728
CABMON, 381
CHARNZ, 835
CHARZ, 840
COMPCAT, 1625
COMRING, 845
DIFEXT, 955
DIFRING, 850
DIRPCAT, 1004
DIVCAT, 1181
DIVRING, 1014
DPOLCAT, 1310
ENTIRER, 855
EUCDOM, 1187
FAMONC, 565
FAMR, 1143
FAXF, 1587
FDIVCAT, 618
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FINAALG, 1022
FINRALG, 1331
FLALG, 1043
FLINEXP, 962
FMCAT, 859
FMTC, 1110
FPC, 1325
FPS, 1532
FRAMALG, 1541
FRNAALG, 1118
FS, 1341
GCDDOM, 1131
GRALG, 314
GRMOD, 177
GROUP, 400
INS, 1231
INTCAT, 1153
INTDOM, 1049
LALG, 865
LIECAT, 968
LINEXP, 871
LMODULE, 657
LODOCAT, 973
MATCAT, 437
MLO, 1055
MODULE, 876
MONAD, 198
MONADWU, 327
MONOGEN, 1602
MONOID, 332
MTSCAT, 1196
NAALG, 982
NARNG, 681
NASRING, 742
OAGROUP, 756
OAMON, 577
OAMONS, 760
OASGP, 458
OC, 1062
OCAMON, 697
OINTDOM, 1137
ORDMON, 462
ORDRING, 881
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371

PACRATC, 1613
PADICCT, 1251
PDRING, 887
PERMCAT, 582
PFECAT, 1204
PID, 1168
PLACESC, 341
POLYCAT, 1260
PSCAT, 1161
PTCAT, 896
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1392
RING, 772
RMATCAT, 903
RMODULE, 716
RNG, 720
RNS, 1403
RPOLCAT, 1414
SGROUP, 240
SMATCAT, 1085
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
VECTCAT, 811
VSPACE, 989
XALG, 942
XF, 1513
XFALG, 994
XPOLYC, 1096
?+?
ABELGRP, 522
ABELMON, 269
ABELSG, 124
ACF, 1298
ACFS, 1500
ALGEBRA, 949
AMR, 1103
BMODULE, 728
CABMON, 381
CHARNZ, 835
CHARZ, 840
COMPCAT, 1625
COMRING, 845
DIFEXT, 955
DIFRING, 850
DIRPCAT, 1004
DIVCAT, 1181
DIVRING, 1014
DPOLCAT, 1310
ENTIRER, 855
EUCDOM, 1187
FAMONC, 565
FAMR, 1143
FAXF, 1587
FDIVCAT, 618
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FINAALG, 1022
FINRALG, 1331
FLALG, 1043
FLINEXP, 962
FMCAT, 859
FMTC, 1110
FPC, 1325
FPS, 1532
FRAMALG, 1541
FRNAALG, 1118
FS, 1341
GCDDOM, 1131
GRALG, 314
GRMOD, 177
INS, 1231
INTCAT, 1153
INTDOM, 1049
LALG, 865
LIECAT, 968
LINEXP, 871
LMODULE, 657
LOCPOWC, 1243
LODOCAT, 973
MATCAT, 437
MLO, 1055
MODULE, 876
MONOGEN, 1602
MTSCAT, 1196

NAALG, 982
NARNG, 681
NASRING, 742
OAGROUP, 756
OAMON, 577
OAMONS, 760
OASGP, 458
OC, 1062
OCAMON, 697
OINTDOM, 1137
ORDRING, 881
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PDRING, 887
PFECAT, 1204
PID, 1168
PLACESC, 341
POLYCAT, 1260
PSCAT, 1161
PTCAT, 896
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1392
RING, 772
RMATCAT, 903
RMODULE, 716
RNG, 720
RNS, 1403
RPOLCAT, 1414
SMATCAT, 1085
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
VECTCAT, 811
VSPACE, 989
XALG, 942
XF, 1513
XFALG, 994
XPOLYC, 1096
?-?
ABELGRP, 522
ACF, 1298
ACFS, 1500
ALGEBRA, 949
AMR, 1103
BMODULE, 728
CHARNZ, 835
CHARZ, 840
COMPCAT, 1625
COMRING, 845
DIFEXT, 955
DIFRING, 850
DIRPCAT, 1004
DIVCAT, 1181
DIVRING, 1014
DPOLCAT, 1310
ENTIRER, 855
EUCDOM, 1187
FAMR, 1143
FAXF, 1587
FDIVCAT, 618
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FINAALG, 1022
FINRALG, 1331
FLALG, 1043
FLINEXP, 962
FMCAT, 859
FMTC, 1110
FPC, 1325
FPS, 1532
FRAMALG, 1541
FRNAALG, 1118
FS, 1341
GCDDOM, 1131
GRALG, 314
GRMOD, 177
INS, 1231
INTCAT, 1153
INTDOM, 1049
LALG, 865
LIECAT, 968

LINEXP, 871
LMODULE, 657
LOCPOWC, 1243
LODOCAT, 973
MATCAT, 437
MLO, 1055
MODULE, 876
MONOGEN, 1602
MTSCAT, 1196
NAALG, 982
NARNG, 681
NASRING, 742
OAGROUP, 756
OC, 1062
OINTDOM, 1137
ORDRING, 881
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PDRING, 887
PFECAT, 1204
PID, 1168
PLACESC, 341
POLYCAT, 1260
PSCAT, 1161
PTCAT, 896
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1392
RING, 772
RMATCAT, 903
RMODULE, 716
RNG, 720
RNS, 1403
RPOLCAT, 1414
SMATCAT, 1085
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
VECTCAT, 811
VSPACE, 989
XALG, 942
XF, 1513
XFALG, 994
XPOLYC, 1096

?..?

SEGCAT, 110
SEGXCAT, 236

?.?

A1AGG, 687
AFSPCAT, 273
ALAGG, 824
BTAGG, 734
COMPCAT, 1625
DIRPCAT, 1004
ELAGG, 550
ELTAB, 30
ELTAGG, 77
FEVALAB, 160
FLAGG, 558
FRNAALG, 1118
IXAGG, 320
LNAGG, 407
LOCPOWC, 1243
LODOCAT, 973
LSAGG, 663
LZSTAGG, 640
OC, 1062
PERMCAT, 582
PLACESC, 341
PRSPCAT, 346
PTCAT, 896
QFCAT, 1380
QUATCAT, 1073
SEXCAT, 248
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464

UPXSCCA, 1574
UTSCAT, 1281
VECTCAT, 811
?.first
ALAGG, 824
LSAGG, 663
LZSTAGG, 640
STAGG, 588
URAGG, 509
?.last
ALAGG, 824
LSAGG, 663
LZSTAGG, 640
STAGG, 588
URAGG, 509
?.left
BRAGG, 374
BTCAT, 528
?.rest
ALAGG, 824
LSAGG, 663
LZSTAGG, 640
STAGG, 588
URAGG, 509
?.right
BRAGG, 374
BTCAT, 528
?.value
ALAGG, 824
BRAGG, 374
BTCAT, 528
DLAGG, 394
DSTRCAT, 133
LSAGG, 663
LZSTAGG, 640
RCAGG, 353
STAGG, 588
URAGG, 509
?/?
ACF, 1298
ACFS, 1500
AMR, 1103
COMPCAT, 1625
DIRPCAT, 1004
DPOLCAT, 1310
FAMR, 1143
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FLALG, 1043
FPC, 1325
FPS, 1532
FS, 1341
GROUP, 400
LIECAT, 968
LOCPOWC, 1243
MATCAT, 437
MONOGEN, 1602
MTSCAT, 1196
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PERMCAT, 582
POLYCAT, 1260
PSCAT, 1161
QFCAT, 1380
RCFIELD, 1392
RMATCAT, 903
RNS, 1403
RPOLCAT, 1414
SMATCAT, 1085
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
VSPACE, 989
XF, 1513
?=?
A1AGG, 687
ABELGRP, 522
ABELMON, 269
ABELSG, 124
ACF, 1298
ACFS, 1500
AFSPCAT, 273
ALAGG, 824
ALGEBRA, 949

AMR, 1103
ARR2CAT, 359
BASTYPE, 15
BGAGG, 279
BLMETCT, 128
BMODULE, 728
BRAGG, 374
BTAGG, 734
BTCAT, 528
CABMON, 381
CACHESET, 284
CHARNZ, 835
CHARZ, 840
CLAGG, 289
COMPCAT, 1625
COMRING, 845
DIAGG, 535
DIFEXT, 955
DIFRING, 850
DIOPS, 387
DIRPCAT, 1004
DIVCAT, 1181
DIVRING, 1014
DLAGG, 394
DPOLCAT, 1310
DQAGG, 542
DSTRCAT, 133
DVARCAT, 296
ELAGG, 550
ENTIRER, 855
ES, 302
EUCDOM, 1187
FAMONC, 565
FAMR, 1143
FAXF, 1587
FDIVCAT, 618
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FILECAT, 165
FINAALG, 1022
FINITE, 169
FINRALG, 1331
FLAGG, 558
FLALG, 1043
FLINEXP, 962
FMCAT, 859
FMTC, 1110
FNCAT, 173
FPATMAB, 93
FPC, 1325
FPS, 1532
FRAMALG, 1541
FRNAALG, 1118
FS, 1341
FSAGG, 624
GCDDOM, 1131
GRALG, 314
GRMOD, 177
GROUP, 400
HOAGG, 183
IDPC, 189
INFCLCT, 1366
INS, 1231
INTCAT, 1153
INTDOM, 1049
IXAGG, 320
KDAGG, 632
LALG, 865
LIECAT, 968
LINEXP, 871
LMODULE, 657
LNAGG, 407
LOCPOWC, 1243
LODOCAT, 973
LOGIC, 96
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
MDAGG, 572
MLO, 1055
MODULE, 876
MONAD, 198
MONADWU, 327
MONOGEN, 1602
MONOID, 332
MSETAGG, 675
MTSCAT, 1196
NAALG, 982
NARNG, 681
NASRING, 742
NTSCAT, 749

NUMINT, 203
OAGROUP, 756
OAMON, 577
OAMONS, 760
OASGP, 458
OC, 1062
OCAMON, 697
ODECAT, 213
OINTDOM, 1137
OMSAGG, 765
OPTCAT, 208
ORDFIN, 337
ORDMON, 462
ORDRING, 881
ORDSET, 218
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PATMAB, 227
PDECAT, 222
PDRING, 887
PERMCAT, 582
PFECAT, 1204
PID, 1168
PLACESC, 341
POLYCAT, 1260
PRQAGG, 482
PRSPCAT, 346
PSCAT, 1161
PSETCAT, 468
PTCAT, 896
QFCAT, 1380
QUAGG, 488
QUATCAT, 1073
RCAGG, 353
RCFIELD, 1392
RING, 772
RMATCAT, 903
RMODULE, 716
RNG, 720
RNS, 1403
RPOLCAT, 1414
RRCC, 231
RSETCAT, 703
SETAGG, 494
SETCAT, 115
SETCATD, 244
SEXCAT, 248
SFRTCAT, 779
SGROUP, 240
SKAGG, 502
SMATCAT, 1085
SNTSCAT, 914
SPACEC, 258
SRAGG, 789
STAGG, 588
STEP, 253
STRICAT, 923
TBAGG, 800
TSETCAT, 598
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
URAGG, 509
UTSCAT, 1281
VECTCAT, 811
VSPACE, 989
XALG, 942
XF, 1513
XFALG, 994
XPOLYC, 1096
?^?
ACF, 1298
ACFS, 1500
ALGEBRA, 949
AMR, 1103
CHARNZ, 835
CHARZ, 840
COMPCAT, 1625
COMRING, 845
DIFEXT, 955
DIFRING, 850
DIRPCAT, 1004
DIVRING, 1014
DPOLCAT, 1310

ENTIRER, 855
EUCDOM, 1187
FAMR, 1143
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FINRALG, 1331
FLINEXP, 962
FMTC, 1110
FPC, 1325
FPS, 1532
FRAMALG, 1541
FS, 1341
GCDDOM, 1131
GROUP, 400
INS, 1231
INTCAT, 1153
INTDOM, 1049
LALG, 865
LINEXP, 871
LOCPOWC, 1243
LODOCAT, 973
MLO, 1055
MONOGEN, 1602
MONOID, 332
MTSCAT, 1196
OC, 1062
OINTDOM, 1137
ORDMON, 462
ORDRING, 881
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PDRING, 887
PERMCAT, 582
PFECAT, 1204
PID, 1168
POLYCAT, 1260
PSCAT, 1161
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1392
RING, 772
RNG, 720
RNS, 1403
RPOLCAT, 1414
SGROUP, 240
SMATCAT, 1085
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
XALG, 942
XF, 1513
XFALG, 994
XPOLYC, 1096
? =?
PRSPCAT, 346
RRCC, 231
?~=?
A1AGG, 687
ABELGRP, 522
ABELMON, 269
ABELSG, 124
ACF, 1298
ACFS, 1500
AFSPCAT, 273
ALAGG, 824
ALGEBRA, 949
AMR, 1103
ARR2CAT, 359
BASTYPE, 15
BGAGG, 279
BLMETCT, 128
BMODULE, 728
BRAGG, 374
BTAGG, 734
BTCAT, 528
CABMON, 381
CACHESET, 284
CHARNZ, 835
CHARZ, 840
CLAGG, 289
COMPCAT, 1625

COMRING, 845
DIAGG, 535
DIFEXT, 955
DIFRING, 850
DIOPS, 387
DIRPCAT, 1004
DIVCAT, 1181
DIVRING, 1014
DLAGG, 394
DPOLCAT, 1310
DQAGG, 542
DSTRCAT, 133
DVARCAT, 296
ELAGG, 550
ENTIRER, 855
ES, 302
EUCDOM, 1187
FAMONC, 565
FAMR, 1143
FAXF, 1587
FDIVCAT, 618
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FILECAT, 165
FINAALG, 1022
FINITE, 169
FINRALG, 1331
FLAGG, 558
FLALG, 1043
FLINEXP, 962
FMCAT, 859
FMTC, 1110
FNCAT, 173
FPATMAB, 93
FPC, 1325
FPS, 1532
FRAMALG, 1541
FRNAALG, 1118
FS, 1341
FSAGG, 624
GCDDOM, 1131
GRALG, 314
GRMOD, 177
GROUP, 400
HOAGG, 183
IDPC, 189
INFCLCT, 1366
INS, 1231
INTCAT, 1153
INTDOM, 1049
IXAGG, 320
KDAGG, 632
LALG, 865
LIECAT, 968
LINEXP, 871
LMODULE, 657
LNAGG, 407
LOCPOWC, 1243
LODOCAT, 973
LOGIC, 96
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
MDAGG, 572
MLO, 1055
MODULE, 876
MONAD, 198
MONADWU, 327
MONOGEN, 1602
MONOID, 332
MSETAGG, 675
MTSCAT, 1196
NAALG, 982
NARNG, 681
NASRING, 742
NTSCAT, 749
NUMINT, 203
OAGROUP, 756
OAMON, 577
OAMONS, 760
OASGP, 458
OC, 1062
OCAMON, 697
ODECAT, 213
OINTDOM, 1137
OPTCAT, 208
ORDFIN, 337
ORDMON, 462
ORDRING, 881
ORDSET, 218
OREPCAT, 931

PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PATMAB, 227
PDECAT, 222
PDRING, 887
PERMCAT, 582
PFECAT, 1204
PID, 1168
PLACESC, 341
POLYCAT, 1260
PRQAGG, 482
PSCAT, 1161
PSETCAT, 468
PTCAT, 896
QFCAT, 1380
QUAGG, 488
QUATCAT, 1073
RCAGG, 353
RCFIELD, 1392
RING, 772
RMATCAT, 903
RMODULE, 716
RNG, 720
RNS, 1403
RPOLCAT, 1414
RSETCAT, 703
SETAGG, 494
SETCAT, 115
SETCATD, 244
SEXCAT, 248
SFRTCAT, 779
SGROUP, 240
SKAGG, 502
SMATCAT, 1085
SNTSCAT, 914
SPACEC, 258
SRAGG, 789
STAGG, 588
STEP, 253
STRICAT, 923
TBAGG, 800
TSETCAT, 598
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
URAGG, 509
UTSCAT, 1281
VECTCAT, 811
VSPACE, 989
XALG, 942
XF, 1513
XFALG, 994
XPOLYC, 1096
?~?
OMSAGG, 765
?and?
BTAGG, 734
?or?
BTAGG, 734
?quo?
ACF, 1298
ACFS, 1500
COMPCAT, 1625
EUCDOM, 1187
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FPC, 1325
FPS, 1532
FS, 1341
INS, 1231
LOCPOWC, 1243
MONOGEN, 1602
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477

UPXSCAT, 1464
UPXSCCA, 1574
XF, 1513
?rem?
ACF, 1298
ACFS, 1500
COMPCAT, 1625
EUCDOM, 1187
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FPC, 1325
FPS, 1532
FS, 1341
INS, 1231
LOCPOWC, 1243
MONOGEN, 1602
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPXSCAT, 1464
UPXSCCA, 1574
XF, 1513
#?
A1AGG, 687
ALAGG, 824
ARR2CAT, 359
BGAGG, 279
BRAGG, 374
BTAGG, 734
BTCAT, 528
CLAGG, 289
DIAGG, 535
DIOPS, 387
DIRPCAT, 1004
DLAGG, 394
DQAGG, 542
ELAGG, 550
FLAGG, 558
FSAGG, 624
HOAGG, 183
IXAGG, 320
KDAGG, 632
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
MDAGG, 572
MSETAGG, 675
NTSCAT, 749
OMSAGG, 765
PRQAGG, 482
PSETCAT, 468
PTCAT, 896
QUAGG, 488
RCAGG, 353
RMATCAT, 903
RSETCAT, 703
SETAGG, 494
SEXCAT, 248
SFRTCAT, 779
SKAGG, 502
SMATCAT, 1085
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
URAGG, 509
VECTCAT, 811
#?
DSTRCAT, 133
^?
BTAGG, 734
~?
BTAGG, 734
LOGIC, 96

0
ABELGRP, 522
ABELMON, 269
ACF, 1298

ACFS, 1500
ALGEBRA, 949
AMR, 1103
BMODULE, 728
CABMON, 381
CHARNZ, 835
CHARZ, 840
COMPCAT, 1625
COMRING, 845
DIFEXT, 955
DIFRING, 850
DIRPCAT, 1004
DIVCAT, 1181
DIVRING, 1014
DPOLCAT, 1310
ENTIRER, 855
EUCDOM, 1187
FAMONC, 565
FAMR, 1143
FAXF, 1587
FDIVCAT, 618
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FINAALG, 1022
FINRALG, 1331
FLALG, 1043
FLINEXP, 962
FMCAT, 859
FMTC, 1110
FPC, 1325
FPS, 1532
FRAMALG, 1541
FRNAALG, 1118
FS, 1341
GCDDOM, 1131
GRALG, 314
GRMOD, 177
INS, 1231
INTCAT, 1153
INTDOM, 1049
LALG, 865
LIECAT, 968
LINEXP, 871
LMODULE, 657
LOCPOWC, 1243
LODOCAT, 973
MLO, 1055
MODULE, 876
MONOGEN, 1602
MTSCAT, 1196
NAALG, 982
NARNG, 681
NASRING, 742
OAGROUP, 756
OAMON, 577
OAMONS, 760
OASGP, 458
OC, 1062
OCAMON, 697
OINTDOM, 1137
ORDRING, 881
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PDRING, 887
PFECAT, 1204
PID, 1168
POLYCAT, 1260
PSCAT, 1161
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1392
RING, 772
RMATCAT, 903
RMODULE, 716
RNG, 720
RNS, 1403
RPOLCAT, 1414
SMATCAT, 1085
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
VSPACE, 989

XALG, 942
XF, 1513
XFALG, 994
XPOLYC, 1096

1

ACF, 1298
ACFS, 1500
ALGEBRA, 949
AMR, 1103
CHARNZ, 835
CHARZ, 840
COMPCAT, 1625
COMRING, 845
DIFEXT, 955
DIFRING, 850
DIRPCAT, 1004
DIVRING, 1014
DPOLCAT, 1310
ENTIRER, 855
EUCDOM, 1187
FAMR, 1143
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FINRALG, 1331
FLINEXP, 962
FMTC, 1110
FPC, 1325
FPS, 1532
FRAMALG, 1541
FS, 1341
GCDDOM, 1131
GRALG, 314
GROUP, 400
INS, 1231
INTCAT, 1153
INTDOM, 1049
LALG, 865
LINEXP, 871
LOCPOWC, 1243
LODOCAT, 973
MLO, 1055
MONADWU, 327
MONOGEN, 1602
MONOID, 332
MTSCAT, 1196
NASRING, 742
OC, 1062
OINTDOM, 1137
ORDMON, 462
ORDRING, 881
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PDRING, 887
PERMCAT, 582
PFECAT, 1204
PID, 1168
POLYCAT, 1260
PSCAT, 1161
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1392
RING, 772
RNS, 1403
RPOLCAT, 1414
SMATCAT, 1085
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
XALG, 942
XF, 1513
XFALG, 994
XPOLYC, 1096

A1AGG, 684
?<?, 687
?<=?, 687
?>?, 687
?>=?, 687
?.?, 687
?=?, 687
?~=?, 687

#?, 687
any?, 687
coerce, 687
concat, 687
construct, 687
convert, 687
copy, 687
copyInto, 687
count, 687
delete, 687
elt, 687
empty, 687
empty?, 687
entries, 687
entry?, 687
eq?, 687
eval, 687
every?, 687
fill, 687
find, 687
finiteAggregate, 687
first, 687
hash, 687
index?, 687
indices, 687
insert, 687
latex, 687
less?, 687
map, 687
max, 687
maxIndex, 687
member?, 687
members, 687
merge, 687
min, 687
minIndex, 687
more?, 687
new, 687
parts, 687
position, 687
qelt, 687
qsetelt, 687
reduce, 687
remove, 687
removeDuplicates, 687
reverse, 687
sample, 687
select, 687
setelt, 687
shallowlyMutable, 687
size?, 687
sort, 687
sorted?, 687
swap, 687
ABELGRP, 520
-?, 522
?*?, 522
?+?, 522
?-?, 522
?=?, 522
?~=?, 522
0, 522
coerce, 522
hash, 522
latex, 522
sample, 522
subtractIfCan, 522
zero?, 522
AbelianGroup, 520
AbelianMonoid, 267
AbelianMonoidRing, 1101
AbelianSemiGroup, 123
ABELMON, 267
?*?, 269
?+?, 269
?=?, 269
?~=?, 269
0, 269
coerce, 269
hash, 269
latex, 269
sample, 269
zero?, 269
ABELSG, 123
?*?, 124
?+?, 124
?=?, 124
?~=?, 124
coerce, 124
hash, 124
latex, 124
abs

COMPCAT, 1625
DIRPCAT, 1004
FPS, 1532
INS, 1231
OC, 1062
OINTDOM, 1137
ORDRING, 881
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1392
RNS, 1403
SPFCAT, 60
ULSCCAT, 1559
absolutelyIrreducible?
FFCAT, 1647
ACF, 1293
-?, 1298
?**?, 1298
?*?, 1298
?+?, 1298
?-?, 1298
?/?, 1298
?=?, 1298
?^?, 1298
?~=?, 1298
?quo?, 1298
?rem?, 1298
0, 1298
1, 1298
associates?, 1298
canonicalsClosed, 1299
canonicalUnitNormal, 1298
characteristic, 1298
coerce, 1298
commutative("*"), 1299
divide, 1298
euclideanSize, 1298
expressIdealMember, 1298
exquo, 1298
extendedEuclidean, 1298
factor, 1298
gcd, 1298
gcdPolynomial, 1298
hash, 1298
inv, 1298
latex, 1298
lcm, 1298
leftUnitary, 1299
multiEuclidean, 1298
noZeroDivisors, 1299
nthRoot, 1298
one?, 1298
prime?, 1298
principalIdeal, 1298
recip, 1298
rightUnitary, 1299
rootOf, 1298
rootsOf, 1298
sample, 1298
sizeLess?, 1298
sqrt, 1298
squareFree, 1298
squareFreePart, 1298
subtractIfCan, 1298
unit?, 1298
unitCanonical, 1298
unitNormal, 1298
unitsKnown, 1299
zero?, 1298
zeroOf, 1298
zerosOf, 1298
ACFS, 1495
-?, 1500
?<?, 1500
?<=?, 1500
?>?, 1500
?>=?, 1500
?**?, 1500
?*?, 1500
?+?, 1500
?-?, 1500
?/?, 1500
?=?, 1500
?^?, 1500
?~=?, 1500
?quo?, 1500
?rem?, 1500
0, 1500
1, 1500
applyQuote, 1500
associates?, 1500
belong?, 1500

box, 1500
canonicalsClosed, 1500
canonicalUnitNormal, 1500
characteristic, 1500
charthRoot, 1500
coerce, 1500
commutative("*"), 1500
commutator, 1500
conjugate, 1500
convert, 1500
D, 1500
definingPolynomial, 1500
denom, 1500
denominator, 1500
differentiate, 1500
distribute, 1500
divide, 1500
elt, 1500
euclideanSize, 1500
eval, 1500
even?, 1500
expressIdealMember, 1500
exquo, 1500
extendedEuclidean, 1500
factor, 1500
freeOf?, 1500
gcd, 1500
gcdPolynomial, 1500
ground, 1500
ground?, 1500
hash, 1500
height, 1500
inv, 1500
is?, 1500
isExpt, 1500
isMult, 1500
isPlus, 1500
isPower, 1500
isTimes, 1500
kernel, 1500
kernels, 1500
latex, 1500
lcm, 1500
leftUnitary, 1500
mainKernel, 1500
map, 1500
max, 1500
min, 1500
minPoly, 1500
multiEuclidean, 1500
noZeroDivisors, 1500
nthRoot, 1500
numer, 1500
numerator, 1500
odd?, 1500
one?, 1500
operator, 1500
operators, 1500
paren, 1500
patternMatch, 1500
prime?, 1500
principalIdeal, 1500
recip, 1500
reducedSystem, 1500
retract, 1500
retractIfCan, 1500
rightUnitary, 1500
rootOf, 1500
rootsOf, 1500
sample, 1500
sizeLess?, 1500
sqrt, 1500
squareFree, 1500
squareFreePart, 1500
subst, 1500
subtractIfCan, 1500
tower, 1500
unit?, 1500
unitCanonical, 1500
unitNormal, 1500
unitsKnown, 1500
univariate, 1500
variables, 1500
zero?, 1500
zeroOf, 1500
zerosOf, 1500

acos
ATRIG, 8
COMPCAT, 1625
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196

TRANFUN, 118
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
acosh
AHYP, 6
COMPCAT, 1625
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
acoshIfCan
PTRANFN, 43
acosIfCan
PTRANFN, 43
acot
ATRIG, 8
COMPCAT, 1625
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
acoth
AHYP, 6
COMPCAT, 1625
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
acothIfCan
PTRANFN, 43
acotIfCan
PTRANFN, 43
acsc
ATRIG, 8
COMPCAT, 1625
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
acsch
AHYP, 6
COMPCAT, 1625
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
acschIfCan
PTRANFN, 43
acscIfCan
PTRANFN, 43
actualExtensionV
INFCLCT, 1366
additiveValuation
ATTREG, 11
COMPCAT, 1626
UPOLYC, 1477
addmod
INS, 1231
adjoint
LODOCAT, 973
affinePoint
AFSPCAT, 273
AffineSpaceCategory, 272
AFSPCAT, 272

?.?, 273
?=?, 273
?~=?, 273
affinePoint, 273
coerce, 273
conjugate, 273
definingField, 273
degree, 273
hash, 273
latex, 273
list, 273
orbit, 273
origin, 273
pointValue, 273
rational?, 273
removeConjugate, 273
setelt, 273
AGG, 69
empty?, 70
eq?, 70
finiteAggregate, 70
less?, 70
more?, 70
sample, 70
size?, 70
Aggregate, 69
AHYP, 5
acosh, 6
acoth, 6
acsch, 6
asech, 6
asinh, 6
atanh, 6
airyAi
SPFCAT, 60
airyBi
SPFCAT, 60
ALAGG, 819
?<?, 824
?<=?, 824
?>?, 824
?>=?, 824
?.?, 824
?.first, 824
?.last, 824
?.rest, 824
?.value, 824
?=?, 824
?~=?, 824
#?, 824
any?, 824
assoc, 824
bag, 824
child?, 824
children, 824
coerce, 824
concat, 824
construct, 824
convert, 824
copy, 824
copyInto, 824
count, 824
cycleEntry, 824
cycleLength, 824
cycleSplit, 824
cycleTail, 824
cyclic?, 824
delete, 824
dictionary, 824
distance, 824
elt, 824
empty, 824
empty?, 824
entries, 824
entry?, 824
eq?, 824
eval, 824
every?, 824
explicitlyFinite?, 824
extract, 824
fill, 824
find, 824
finiteAggregate, 824
first, 824
hash, 824
index?, 824
indices, 824
insert, 824
inspect, 824
key?, 824
keys, 824
last, 824

latex, 824
leaf?, 824
leaves, 824
less?, 824
list, 824
map, 824
max, 824
maxIndex, 824
member?, 824
members, 824
merge, 824
min, 824
minIndex, 824
more?, 824
new, 824
node?, 824
nodes, 824
parts, 824
position, 824
possiblyInfinite?, 824
qelt, 824
qsetelt, 824
reduce, 824
remove, 824
removeDuplicates, 824
rest, 824
reverse, 824
sample, 824
search, 824
second, 824
select, 824
setchildren, 824
setelt, 824
setfirst, 824
setlast, 824
setrest, 824
setvalue, 824
shallowlyMutable, 824
size?, 824
sort, 824
sorted?, 824
split, 824
swap, 824
table, 824
tail, 824
third, 824
value, 824
ALGEBRA, 947
-?, 949
?**?, 949
?*?, 949
?+?, 949
?-?, 949
?=?, 949
?^?, 949
?~=?, 949
0, 949
1, 949
characteristic, 949
coerce, 949
hash, 949
latex, 949
leftUnitary, 949
one?, 949
recip, 949
rightUnitary, 949
sample, 949
subtractIfCan, 949
unitsKnown, 949
zero?, 949
Algebra, 947
algebraic?
FAXF, 1587
NTSCAT, 749
PACEXTC, 1668
PACRATC, 1613
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
XF, 1513
AlgebraicallyClosedField, 1293
AlgebraicallyClosedFunctionSpace, 1495
algebraicCoefficients?
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
algebraicVariables
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914

TSETCAT, 598
algSplitSimple
FFCAT, 1647
allRootsOf
RCFIELD, 1392
RRCC, 231
alternative?
FINAALG, 1022
FRNAALG, 1118
AMR, 1101
-?, 1103
?**?, 1103
?*?, 1103
?+?, 1103
?-?, 1103
?/?, 1103
?=?, 1103
?^?, 1103
?~=?, 1103
0, 1103
1, 1103
associates?, 1103
characteristic, 1103
charthRoot, 1103
coefficient, 1103
coerce, 1103
commutative("*"), 1103
degree, 1103
exquo, 1103
hash, 1103
latex, 1103
leadingCoefficient, 1103
leadingMonomial, 1103
leftUnitary, 1103
map, 1103
monomial, 1103
monomial?, 1103
noZeroDivisors, 1103
one?, 1103
recip, 1103
reductum, 1103
rightUnitary, 1103
sample, 1103
subtractIfCan, 1103
unit?, 1103
unitCanonical, 1103
unitNormal, 1103
unitsKnown, 1103
zero?, 1103
antiAssociative?
FINAALG, 1022
FRNAALG, 1118
antiCommutative?
FINAALG, 1022
FRNAALG, 1118
antiCommutator
FINAALG, 1022
FRNAALG, 1118
NAALG, 982
NARNG, 681
NASRING, 742
antisymmetric?
MATCAT, 437
RMATCAT, 903
SMATCAT, 1085
any?
A1AGG, 687
ALAGG, 824
ARR2CAT, 359
BGAGG, 279
BRAGG, 374
BTAGG, 734
BTCAT, 528
CLAGG, 289
DIAGG, 535
DIOPS, 387
DIRPCAT, 1004
DLAGG, 394
DQAGG, 542
DSTRCAT, 133
ELAGG, 550
FLAGG, 558
FSAGG, 624
HOAGG, 183
IXAGG, 320
KDAGG, 632
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
MDAGG, 572
MSETAGG, 675

NTSCAT, 749
OMSAGG, 765
PRQAGG, 482
PSETCAT, 468
PTCAT, 896
QUAGG, 488
RCAGG, 353
RMATCAT, 903
RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SKAGG, 502
SMATCAT, 1085
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
URAGG, 509
VECTCAT, 811
apply
FRNAALG, 1118
LODOCAT, 973
OREPCAT, 931
applyQuote
ACFS, 1500
FS, 1341
approximate
ATTREG, 12
FPS, 1532
INTCAT, 1153
LOCPOWC, 1243
PADICCT, 1251
RCFIELD, 1392
RRCC, 231
ULSCAT, 1452
ULSCCAT, 1559
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
arbitraryPrecision
ATTREG, 11
ArcHyperbolicFunctionCategory, 5
ArcTrigonometricFunctionCategory, 7
argument
COMPCAT, 1625
ARR2CAT, 357
?=?, 359
?~=?, 359
#?, 359
any?, 359
coerce, 359
column, 359
copy, 359
count, 359
elt, 359
empty, 359
empty?, 359
eq?, 359
eval, 359
every?, 359
fill, 359
finiteAggregate, 359
hash, 359
latex, 359
less?, 359
map, 359
maxColIndex, 359
maxRowIndex, 359
member?, 359
members, 359
minColIndex, 359
minRowIndex, 359
more?, 359
ncols, 359
new, 359
nrows, 359
parts, 359
qelt, 359
qsetelt, 359
row, 359
sample, 359
setColumn, 359
setelt, 359
setRow, 359
shallowlyMutable, 359
size?, 359
asec
ATRIG, 8
COMPCAT, 1625

INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
asech
AHYP, 6
COMPCAT, 1625
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
asechIfCan
PTRANFN, 43
asecIfCan
PTRANFN, 43
asin
ATRIG, 8
COMPCAT, 1625
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
asinh
AHYP, 6
COMPCAT, 1625
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
asinhIfCan
PTRANFN, 43
asinIfCan
PTRANFN, 43
assoc
ALAGG, 824
associates?
ACF, 1298
ACFS, 1500
AMR, 1103
COMPCAT, 1625
DPOLCAT, 1310
EUCDOM, 1187
FAMR, 1143
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FMTC, 1110
FPC, 1325
FPS, 1532
FS, 1341
GCDDOM, 1131
INS, 1231
INTCAT, 1153
INTDOM, 1049
LOCPOWC, 1243
MONOGEN, 1602
MTSCAT, 1196
OINTDOM, 1137
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PFECAT, 1204
PID, 1168
POLYCAT, 1260
PSCAT, 1161
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
RPOLCAT, 1414

UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
XF, 1513
AssociationListAggregate, 819
associative?
FINAALG, 1022
FRNAALG, 1118
associator
FINAALG, 1022
FRNAALG, 1118
NAALG, 982
NARNG, 681
NASRING, 742
associatorDependence
FINAALG, 1022
FRNAALG, 1118
atan
ATRIG, 8
COMPCAT, 1625
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
atanh
AHYP, 6
COMPCAT, 1625
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
atanhIfCan
PTRANFN, 43
atanIfCan
PTRANFN, 43
atom?
SEXCAT, 248
ATRIG, 7
acos, 8
acot, 8
acsc, 8
asec, 8
asin, 8
atan, 8
ATTREG, 10
additiveValuation, 11
approximate, 12
arbitraryPrecision, 11
canonical, 11
canonicalsClosed, 11, 1668
canonicalUnitNormal, 11
central, 11
commutative("*"), 11
complex, 12
finiteAggregate, 11, 133
JacobiIdentity, 11
leftUnitary, 11
multiplicativeValuation, 11
noetherian, 11
noZeroDivisors, 11
NullSquare, 11
partiallyOrderedSet, 11
rightUnitary, 11
shallowlyMutable, 11
unitsKnown, 11, 582
AttributeRegistry, 10
augment
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
autoReduced?
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598

back
    DQAGG, 542
    QUAGG, 488
bag
    ALAGG, 824
    BGAGG, 279
    DIAGG, 535
    DIOPS, 387
    DQAGG, 542
    FSAGG, 624
    KDAGG, 632
    MDAGG, 572
    MSETAGG, 675
    OMSAGG, 765
    PRQAGG, 482
    QUAGG, 488
    SKAGG, 502
    TBAGG, 800
BagAggregate, 277
base
    FPS, 1532
    INS, 1231
basicSet
    NTSCAT, 749
    RSETCAT, 703
    SFRTCAT, 779
    SNTSCAT, 914
    TSETCAT, 598
BasicType, 14
basis
    COMPCAT, 1625
    FAXF, 1587
    FFCAT, 1647
    FRAMALG, 1541
    FRNAALG, 1118
    MONOGEN, 1602
BASTYPE, 14
    ?=?, 15
    ?~=?, 15
belong?
    ACFS, 1500
    ES, 302
    FS, 1341
besselI
    SPFCAT, 60
besselJ
    SPFCAT, 60
besselK
    SPFCAT, 60
besselY
    SPFCAT, 60
Beta
    SPFCAT, 60
BGAGG, 277
    ?=?, 279
    ?~=?, 279
    #?, 279
    any?, 279
    bag, 279
    coerce, 279
    copy, 279
    count, 279
    empty, 279
    empty?, 279
    eq?, 279
    eval, 279
    every?, 279
    extract, 279
    hash, 279
    insert, 279
    inspect, 279
    latex, 279
    less?, 279
    map, 279
    member?, 279
    members, 279
    more?, 279
    parts, 279
    sample, 279
    shallowlyMutable, 279
    size?, 279
BiModule, 726
BinaryRecursiveAggregate, 372
BinaryTreeCategory, 526
binomial
    CFCAT, 21
    COMBOPC, 74
    INS, 1231
binomThmExpt
    DPOLCAT, 1310
    FAMR, 1143
    POLYCAT, 1260

RPOLCAT, 1414
UPOLYC, 1477
bit?
INS, 1231
BitAggregate, 731
bits
FPS, 1532
BLMETCT, 127
?=?, 128
?~=?, 128
chartCoord, 128
coerce, 128
createHN, 128
excepCoord, 128
hash, 128
infClsPt?, 128
latex, 128
quotValuation, 128
ramifMult, 128
transCoord, 128
type, 128
BlowUpMethodCategory, 127
BMODULE, 726
-?, 728
?*?, 728
?+?, 728
?-?, 728
?=?, 728
?~=?, 728
0, 728
coerce, 728
hash, 728
latex, 728
leftUnitary, 728
rightUnitary, 728
sample, 728
subtractIfCan, 728
zero?, 728
box
ACFS, 1500
ES, 302
FS, 1341
brace
FSAGG, 624
MSETAGG, 675
OMSAGG, 765
SETAGG, 494
BRAGG, 372
?.left, 374
?.right, 374
?.value, 374
?=?, 374
?~=?, 374
#?, 374
any?, 374
child?, 374
children, 374
coerce, 374
copy, 374
count, 374
cyclic?, 374
distance, 374
empty, 374
empty?, 374
eq?, 374
eval, 374
every?, 374
finiteAggregate, 375
hash, 374
latex, 374
leaf?, 374
leaves, 374
left, 374
less?, 374
map, 374
member?, 374
members, 374
more?, 374
node?, 374
nodes, 374
parts, 374
right, 374
sample, 374
setchildren, 374
setelt, 374
setleft, 374
setright, 374
setvalue, 374
shallowlyMutable, 375
size?, 374
value, 374
branchPoint?

FFCAT, 1647
branchPointAtInfinity?
FFCAT, 1647
BTAGG, 731
?/Γ*E*30*F*?, 734
?<?, 734
?<=?, 734
?>?, 734
?>=?, 734
?Γ*E*30*F*/?, 734
?.?, 734
?=?, 734
?~=?, 734
?and?, 734
?or?, 734
#?, 734
^?, 734
~?, 734
any?, 734
coerce, 734
concat, 734
construct, 734
convert, 734
copy, 734
copyInto, 734
count, 734
delete, 734
elt, 734
empty, 734
empty?, 734
entries, 734
entry?, 734
eq?, 734
eval, 734
every?, 734
fill, 734
find, 734
finiteAggregate, 734
first, 734
hash, 734
index?, 734
indices, 734
insert, 734
latex, 734
less?, 734
map, 734
max, 734
maxIndex, 734
member?, 734
members, 734
merge, 734
min, 734
minIndex, 734
more?, 734
nand, 734
new, 734
nor, 734
not?, 734
parts, 734
position, 734
qelt, 734
qsetelt, 734
reduce, 734
remove, 734
removeDuplicates, 734
reverse, 734
sample, 734
select, 734
setelt, 734
shallowlyMutable, 734
size?, 734
sort, 734
sorted?, 734
swap, 734
xor, 734
BTCAT, 526
?.left, 528
?.right, 528
?.value, 528
?=?, 528
?~=?, 528
#?, 528
any?, 528
child?, 528
children, 528
coerce, 528
copy, 528
count, 528
cyclic?, 528
distance, 528
empty, 528
empty?, 528

eq?, 528
eval, 528
every?, 528
finiteAggregate, 528
hash, 528
latex, 528
leaf?, 528
leaves, 528
left, 528
less?, 528
map, 528
member?, 528
members, 528
more?, 528
node, 528
node?, 528
nodes, 528
parts, 528
right, 528
sample, 528
setchildren, 528
setelt, 528
setleft, 528
setright, 528
setvalue, 528
shallowlyMutable, 529
size?, 528
value, 528
BY
SEGCAT, 110
SEGXCAT, 236

CABMON, 380
?*?, 381
?+?, 381
?=?, 381
?~=?, 381
0, 381
coerce, 381
hash, 381
latex, 381
sample, 381
subtractIfCan, 381
zero?, 381
CachableSet, 283
CACHESET
?<?, 284
?<=?, 284
?>?, 284
?>=?, 284
?=?, 284
?~=?, 284
coerce, 284
hash, 284
latex, 284
max, 284
min, 284
position, 284
setPosition, 284
CACHSET, 283
CancellationAbelianMonoid, 380
canonical
ATTREG, 11
canonicalsClosed
ACF, 1299
ACFS, 1500
ATTREG, 11, 1668
COMPCAT, 1626
FAXF, 1587
FFCAT, 1648
FFIELDC, 1521
FIELD, 1224
FPC, 1325
FPS, 1533
FS, 1341
LOCPOWC, 1243
MONOGEN, 1602
PACFFC, 1549
PACPERC, 1371
PACRATC, 1613
QFCAT, 1380
RCFIELD, 1393
ULSCAT, 1452
ULSCCAT, 1560
UPXSCAT, 1464
UPXSCCA, 1575
XF, 1513
canonicalUnitNormal
ACF, 1298
ACFS, 1500
ATTREG, 11
COMPCAT, 1626

DPOLCAT, 1310
FAXF, 1587
FFCAT, 1648
FFIELDC, 1521
FIELD, 1224
FPC, 1325
FPS, 1533
FS, 1341
INS, 1231
INTDOM, 1049
LOCPOWC, 1243
MONOGEN, 1602
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
POLYCAT, 1260
QFCAT, 1380
RCFIELD, 1392
RPOLCAT, 1415
ULSCAT, 1452
ULSCCAT, 1560
UPOLYC, 1478
UPXSCAT, 1464
UPXSCCA, 1575
XF, 1513
car
SEXCAT, 248
cardinality
FSAGG, 624
CATEGORY, 3
Category, 3
cdr
SEXCAT, 248
ceiling
FPS, 1532
QFCAT, 1380
RNS, 1403
ULSCCAT, 1559
center
LOCPOWC, 1243
ULSCAT, 1452
ULSCCAT, 1559
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
central
ATTREG, 11
CFCAT, 20
binomial, 21
factorial, 21
permutation, 21
characteristic
ACF, 1298
ACFS, 1500
ALGEBRA, 949
AMR, 1103
CHARNZ, 835
CHARZ, 840
COMPCAT, 1625
COMRING, 845
DIFEXT, 955
DIFRING, 850
DIRPCAT, 1004
DIVRING, 1014
DPOLCAT, 1310
ENTIRER, 855
EUCDOM, 1187
FAMR, 1143
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FINRALG, 1331
FLINEXP, 962
FMTC, 1110
FPC, 1325
FPS, 1532
FRAMALG, 1541
FS, 1341
GCDDOM, 1131
INS, 1231
INTCAT, 1153
INTDOM, 1049
LALG, 865
LINEXP, 871
LOCPOWC, 1243
LODOCAT, 973
MLO, 1055
MONOGEN, 1602
MTSCAT, 1196

NASRING, 742
OC, 1062
OINTDOM, 1137
ORDRING, 881
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PDRING, 887
PFECAT, 1204
PID, 1168
POLYCAT, 1260
PSCAT, 1161
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1392
RING, 772
RNS, 1403
RPOLCAT, 1414
SMATCAT, 1085
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
XALG, 942
XF, 1513
XFALG, 994
XPOLYC, 1096
CharacteristicNonZero, 833
characteristicPolynomial
COMPCAT, 1625
FFCAT, 1647
FINRALG, 1331
FRAMALG, 1541
MONOGEN, 1602
CharacteristicZero, 838
CHARNZ, 833
-?, 835
?**?, 835
?*?, 835
?+?, 835
?-?, 835
?=?, 835
?^?, 835
?~=?, 835
0, 835
1, 835
characteristic, 835
charthRoot, 835
coerce, 835
hash, 835
latex, 835
one?, 835
recip, 835
sample, 835
subtractIfCan, 835
unitsKnown, 835
zero?, 835
chartCoord
BLMETCT, 128
charthRoot
ACFS, 1500
AMR, 1103
CHARNZ, 835
COMPCAT, 1625
DPOLCAT, 1310
FAMR, 1143
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FINRALG, 1331
FPC, 1325
FRAMALG, 1541
FS, 1341
LOCPOWC, 1243
MONOGEN, 1602
MTSCAT, 1196
OC, 1062
PACEXTC, 1668
PACFFC, 1548
PACRATC, 1613
PFECAT, 1204
POLYCAT, 1260
PSCAT, 1161
QFCAT, 1380
QUATCAT, 1073

RPOLCAT, 1414
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
XF, 1513
chartV
INFCLCT, 1366
CHARZ, 838
-?, 840
?**?, 840
?*?, 840
?+?, 840
?-?, 840
?=?, 840
?^?, 840
?~=?, 840
0, 840
1, 840
characteristic, 840
coerce, 840
hash, 840
latex, 840
one?, 840
recip, 840
sample, 840
subtractIfCan, 840
unitsKnown, 840
zero?, 840
check
SPACEC, 258
child?
ALAGG, 824
BRAGG, 374
BTCAT, 528
DLAGG, 394
DSTRCAT, 133
LSAGG, 663
LZSTAGG, 640
RCAGG, 353
STAGG, 588
URAGG, 509
children
ALAGG, 824
BRAGG, 374
BTCAT, 528
DLAGG, 394
DSTRCAT, 133
LSAGG, 663
LZSTAGG, 640
RCAGG, 353
STAGG, 588
URAGG, 509
Ci
LFCAT, 193
CLAGG, 287
?=?, 289
?~=?, 289
#?, 289
any?, 289
coerce, 289
construct, 289
convert, 289
copy, 289
count, 289
empty, 289
empty?, 289
eq?, 289
eval, 289
every?, 289
find, 289
finiteAggregate, 289
hash, 289
latex, 289
less?, 289
map, 289
member?, 289
members, 289
more?, 289
parts, 289
reduce, 289
remove, 289
removeDuplicates, 289
sample, 289
select, 289
size?, 289
closedCurve
SPACEC, 258
closedCurve?

SPACEC, 258
coef
FLALG, 1043
XFALG, 994
XPOLYC, 1096
coefficient
AMR, 1103
DIVCAT, 1181
DPOLCAT, 1310
FAMONC, 565
FAMR, 1143
FMCAT, 859
LOCPOWC, 1243
LODOCAT, 973
MLO, 1055
MTSCAT, 1196
OREPCAT, 931
POLYCAT, 1260
PSCAT, 1161
RPOLCAT, 1414
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
coefficients
DPOLCAT, 1310
FAMR, 1143
FMCAT, 859
LODOCAT, 973
OREPCAT, 931
POLYCAT, 1260
RPOLCAT, 1414
UPOLYC, 1477
UTSCAT, 1281
coefOfFirstNonZeroTerm
LOCPOWC, 1243
coerce
A1AGG, 687
ABELGRP, 522
ABELMON, 269
ABELSG, 124
ACF, 1298
ACFS, 1500
AFSPCAT, 273
ALAGG, 824
ALGEBRA, 949
AMR, 1103
ARR2CAT, 359
BGAGG, 279
BLMETCT, 128
BMODULE, 728
BRAGG, 374
BTAGG, 734
BTCAT, 528
CABMON, 381
CACHESET, 284
CHARNZ, 835
CHARZ, 840
CLAGG, 289
COMPCAT, 1625
COMRING, 845
DIAGG, 535
DIFEXT, 955
DIFRING, 850
DIOPS, 387
DIRPCAT, 1004
DIVCAT, 1181
DIVRING, 1014
DLAGG, 394
DPOLCAT, 1310
DQAGG, 542
DSTRCAT, 133
DVARCAT, 296
ELAGG, 550
ENTIRER, 855
ES, 302
EUCDOM, 1187
FAMONC, 565
FAMR, 1143
FAXF, 1587
FDIVCAT, 618
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FILECAT, 165
FINAALG, 1022
FINITE, 169
FINRALG, 1331
FLAGG, 558

FLALG, 1043
FLINEXP, 962
FMC, 142
FMCAT, 859
FMFUN, 146
FMTC, 1110
FNCAT, 173
FORTCAT, 85
FORTFN, 137
FPATMAB, 93
FPC, 1325
FPS, 1532
FRAMALG, 1541
FRETRCT, 89
FRNAALG, 1118
FS, 1341
FSAGG, 624
FVFUN, 155
GCDDOM, 1131
GRALG, 314
GRMOD, 177
GROUP, 400
HOAGG, 183
IDPC, 189
INFCLCT, 1366
INS, 1231
INTCAT, 1153
INTDOM, 1049
IXAGG, 320
KDAGG, 632
KOERCE, 18
LALG, 865
LIECAT, 968
LINEXP, 871
LMODULE, 657
LNAGG, 407
LOCPOWC, 1243
LODOCAT, 973
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
MDAGG, 572
MLO, 1055
MODULE, 876
MONAD, 198
MONADWU, 327
MONOGEN, 1602
MONOID, 332
MSETAGG, 675
MTSCAT, 1196
NAALG, 982
NARNG, 681
NASRING, 742
NTSCAT, 749
NUMINT, 203
OAGROUP, 756
OAMON, 577
OAMONS, 760
OASGP, 458
OC, 1062
OCAMON, 697
ODECAT, 213
OINTDOM, 1137
OMSAGG, 765
OPTCAT, 208
ORDFIN, 337
ORDMON, 462
ORDRING, 881
ORDSET, 218
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PATMAB, 227
PDECAT, 222
PDRING, 887
PERMCAT, 582
PFECAT, 1204
PID, 1168
PLACESC, 341
POLYCAT, 1260
PPCURVE, 99
PRQAGG, 482
PRSPCAT, 346
PSCAT, 1161
PSCURVE, 103
PSETCAT, 468
PTCAT, 896
QFCAT, 1380
QUAGG, 488

QUATCAT, 1073
RCAGG, 353
RCFIELD, 1392
RETRACT, 57
RING, 772
RMATCAT, 903
RMODULE, 716
RNG, 720
RNS, 1403
RPOLCAT, 1414
RRCC, 231
RSETCAT, 703
SETAGG, 494
SETCAT, 115
SETCATD, 244
SEXCAT, 248
SFRTCAT, 779
SGROUP, 240
SKAGG, 502
SMATCAT, 1085
SNTSCAT, 914
SPACEC, 258
SRAGG, 789
STAGG, 588
STEP, 253
STRICAT, 923
TBAGG, 800
TSETCAT, 598
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
URAGG, 509
UTSCAT, 1281
VECTCAT, 811
VSPACE, 989
XALG, 942
XF, 1513
XFALG, 994
XPOLYC, 1096
CoercibleTo, 17
coHeight
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
collect
DIVCAT, 1181
NTSCAT, 749
PSETCAT, 468
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
Collection, 287
collectQuasiMonic
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
collectUnder
NTSCAT, 749
PSETCAT, 468
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
collectUpper
NTSCAT, 749
PSETCAT, 468
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
column
ARR2CAT, 359
MATCAT, 437
RMATCAT, 903
SMATCAT, 1085
columnSpace
MATCAT, 437
CombinatorialFunctionCategory, 20
CombinatorialOpsCategory, 73
COMBOPC, 73
binomial, 74
factorial, 74
factorials, 74

permutation, 74
product, 74
summation, 74
commutative("*")
ACF, 1299
ACFS, 1500
AMR, 1103
ATTREG, 11
COMPCAT, 1626
COMRING, 845
DIRPCAT, 1005
DPOLCAT, 1311
EUCDOM, 1187
FAMR, 1143
FAXF, 1587
FFCAT, 1648
FFIELDC, 1521
FIELD, 1224
FMTC, 1110
FPC, 1326
FPS, 1533
FS, 1341
GCDDOM, 1131
INS, 1231
INTCAT, 1153
INTDOM, 1049
LOCPOWC, 1243
MONOGEN, 1603
MTSCAT, 1196
OINTDOM, 1137
PACEXTC, 1668
PACFFC, 1549
PACPERC, 1372
PACRATC, 1614
PADICCT, 1251
PFECAT, 1204
PID, 1168
POLYCAT, 1260
PSCAT, 1161
QFCAT, 1380
RCFIELD, 1393
RPOLCAT, 1415
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1560
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1575
UTSCAT, 1281
XF, 1513
XFALG, 994
commutative?
FINAALG, 1022
FRNAALG, 1118
CommutativeRing, 843
commutator
ACFS, 1500
FINAALG, 1022
FRNAALG, 1118
FS, 1341
GROUP, 400
NAALG, 982
NARNG, 681
NASRING, 742
PERMCAT, 582
COMPCAT, 1619
-?, 1625
?<?, 1625
?<=?, 1625
?>?, 1625
?>=?, 1625
?**?, 1625
?*?, 1625
?+?, 1625
?-?, 1625
?.?, 1625
?/?, 1625
?=?, 1625
?^?, 1625
?~=?, 1625
?quo?, 1625
?rem?, 1625
0, 1625
1, 1625
abs, 1625
acos, 1625
acosh, 1625
acot, 1625
acoth, 1625
acsc, 1625
acsch, 1625

additiveValuation, 1626
argument, 1625
asec, 1625
asech, 1625
asin, 1625
asinh, 1625
associates?, 1625
atan, 1625
atanh, 1625
basis, 1625
canonicalsClosed, 1626
canonicalUnitNormal, 1626
characteristic, 1625
characteristicPolynomial, 1625
charthRoot, 1625
coerce, 1625
commutative("*"), 1626
complex, 1625, 1626
conditionP, 1625
conjugate, 1625
convert, 1625
coordinates, 1625
cos, 1625
cosh, 1625
cot, 1625
coth, 1625
createPrimitiveElement, 1625
csc, 1625
csch, 1625
D, 1625
definingPolynomial, 1625
derivationCoordinates, 1625
differentiate, 1625
discreteLog, 1625
discriminant, 1625
divide, 1625
euclideanSize, 1625
eval, 1625
exp, 1625
expressIdealMember, 1625
exquo, 1625
extendedEuclidean, 1625
factor, 1625
factorPolynomial, 1625
factorsOfCyclicGroupSize, 1625
factorSquareFreePolynomial, 1625
gcd, 1625
gcdPolynomial, 1625
generator, 1625
hash, 1625
imag, 1625
imaginary, 1625
index, 1625
init, 1625
inv, 1625
latex, 1625
lcm, 1625
leftUnitary, 1626
lift, 1625
log, 1625
lookup, 1625
map, 1625
max, 1625
min, 1625
minimalPolynomial, 1625
multiEuclidean, 1625
multiplicativeValuation, 1626
nextItem, 1625
norm, 1625
noZeroDivisors, 1626
nthRoot, 1625
one?, 1625
order, 1625
patternMatch, 1625
pi, 1625
polarCoordinates, 1625
prime?, 1625
primeFrobenius, 1625
primitive?, 1625
primitiveElement, 1625
principalIdeal, 1625
random, 1625
rank, 1625
rational, 1625
rational?, 1625
rationalIfCan, 1625
real, 1625
recip, 1625
reduce, 1625
reducedSystem, 1625
regularRepresentation, 1625
representationType, 1625

represents, 1625
retract, 1625
retractIfCan, 1625
rightUnitary, 1626
sample, 1625
sec, 1625
sech, 1625
sin, 1625
sinh, 1625
size, 1625
sizeLess?, 1625
solveLinearPolynomialEquation, 1625
sqrt, 1625
squareFree, 1625
squareFreePart, 1625
squareFreePolynomial, 1625
subtractIfCan, 1625
tableForDiscreteLogarithm, 1625
tan, 1625
tanh, 1625
trace, 1625
traceMatrix, 1625
unit?, 1625
unitCanonical, 1625
unitNormal, 1625
unitsKnown, 1626
zero?, 1625
complement
FSAGG, 624
complementaryBasis
FFCAT, 1647
complete
LOCPOWC, 1243
LZSTAGG, 640
MTSCAT, 1196
PADICCT, 1251
PSCAT, 1161
ULSCAT, 1452
ULSCCAT, 1559
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
complex
ATTREG, 12
COMPCAT, 1625, 1626
ComplexCategory, 1619
components
SPACEC, 258
composite
SPACEC, 258
UPOLYC, 1477
composites
SPACEC, 258
COMRING, 843
-?, 845
?**?, 845
?*?, 845
?+?, 845
?-?, 845
?=?, 845
?^?, 845
?~=?, 845
0, 845
1, 845
characteristic, 845
coerce, 845
commutative("*"), 845
hash, 845
latex, 845
leftUnitary, 845
one?, 845
recip, 845
rightUnitary, 845
sample, 845
subtractIfCan, 845
unitsKnown, 845
zero?, 845
concat
A1AGG, 687
ALAGG, 824
BTAGG, 734
DIVCAT, 1181
ELAGG, 550
FLAGG, 558
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
PTCAT, 896
SRAGG, 789
STAGG, 588
STRICAT, 923

URAGG, 509
VECTCAT, 811
conditionP
COMPCAT, 1625
DPOLCAT, 1310
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
MONOGEN, 1602
PACFFC, 1548
PFECAT, 1204
POLYCAT, 1260
QFCAT, 1380
RPOLCAT, 1414
ULSCCAT, 1559
UPOLYC, 1477
conditionsForIdempotents
FINAALG, 1022
FRNAALG, 1118
conjugate
ACFS, 1500
AFSPCAT, 273
COMPCAT, 1625
FS, 1341
GROUP, 400
OC, 1062
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PERMCAT, 582
PRSPCAT, 346
QUATCAT, 1073
constant
XFALG, 994
XPOLYC, 1096
constant?
XFALG, 994
XPOLYC, 1096
construct
A1AGG, 687
ALAGG, 824
BTAGG, 734
CLAGG, 289
DIAGG, 535
DIOPS, 387
ELAGG, 550
FLAGG, 558
FLALG, 1043
FSAGG, 624
KDAGG, 632
LIECAT, 968
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MDAGG, 572
MSETAGG, 675
NTSCAT, 749
OMSAGG, 765
PSETCAT, 468
PTCAT, 896
RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
VECTCAT, 811
contains?
INTCAT, 1153
content
DPOLCAT, 1310
FAMR, 1143
LODOCAT, 973
OREPCAT, 931
POLYCAT, 1260
UPOLYC, 1477
convert
A1AGG, 687
ACFS, 1500
ALAGG, 824
BTAGG, 734
CLAGG, 289
COMPCAT, 1625
DIAGG, 535
DIOPS, 387
DPOLCAT, 1310
ELAGG, 550
FFCAT, 1647

FLAGG, 558
FPS, 1532
FRAMALG, 1541
FRNAALG, 1118
FS, 1341
FSAGG, 624
INS, 1231
KDAGG, 632
KONVERT, 24
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MDAGG, 572
MONOGEN, 1602
MSETAGG, 675
NTSCAT, 749
OC, 1062
OMSAGG, 765
PATAB, 48
POLYCAT, 1260
PSETCAT, 468
PTCAT, 896
QFCAT, 1380
QUATCAT, 1073
REAL, 107
RNS, 1403
RPOLCAT, 1414
RSETCAT, 703
SEGCAT, 110
SEGXCAT, 236
SETAGG, 494
SEXCAT, 248
SFRTCAT, 779
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
ULSCCAT, 1559
UPOLYC, 1477
VECTCAT, 811

ConvertibleTo, 23

coordinates
COMPCAT, 1625
FAXF, 1587
FFCAT, 1647
FINAALG, 1022
FINRALG, 1331
FRAMALG, 1541
FRNAALG, 1118
MONOGEN, 1602

copy
A1AGG, 687
ALAGG, 824
ARR2CAT, 359
BGAGG, 279
BRAGG, 374
BTAGG, 734
BTCAT, 528
CLAGG, 289
DIAGG, 535
DIOPS, 387
DIRPCAT, 1004
DLAGG, 394
DQAGG, 542
DSTRCAT, 133
ELAGG, 550
FLAGG, 558
FSAGG, 624
HOAGG, 183
INS, 1231
IXAGG, 320
KDAGG, 632
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
MDAGG, 572
MSETAGG, 675
NTSCAT, 749
OMSAGG, 765
PRQAGG, 482
PSETCAT, 468
PTCAT, 896
QUAGG, 488
RCAGG, 353
RMATCAT, 903
RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SKAGG, 502

SMATCAT, 1085
SNTSCAT, 914
SPACEC, 258
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
URAGG, 509
VECTCAT, 811
cos
COMPCAT, 1625
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
TRIGCAT, 64
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
cosh
COMPCAT, 1625
HYPCAT, 33
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
coshIfCan
PTRANFN, 43
cosIfCan
PTRANFN, 43
cot
COMPCAT, 1625
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
TRIGCAT, 64
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
coth
COMPCAT, 1625
HYPCAT, 33
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
cothIfCan
PTRANFN, 43
cotIfCan
PTRANFN, 43
count
A1AGG, 687
ALAGG, 824
ARR2CAT, 359
BGAGG, 279
BRAGG, 374
BTAGG, 734
BTCAT, 528
CLAGG, 289
DIAGG, 535
DIOPS, 387
DIRPCAT, 1004
DLAGG, 394
DQAGG, 542
DSTRCAT, 133
ELAGG, 550
FLAGG, 558
FSAGG, 624
HOAGG, 183
IXAGG, 320
KDAGG, 632
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
MDAGG, 572

MSETAGG, 675
NTSCAT, 749
OMSAGG, 765
PRQAGG, 482
PSETCAT, 468
PTCAT, 896
QUAGG, 488
RCAGG, 353
RMATCAT, 903
RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SKAGG, 502
SMATCAT, 1085
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
URAGG, 509
VECTCAT, 811
create
INFCLCT, 1366
PLACESC, 341
create3Space
SPACEC, 258
createHN
BLMETCT, 128
createNormalElement
FAXF, 1587
createPrimitiveElement
COMPCAT, 1625
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
MONOGEN, 1602
PACFFC, 1548
cross
PTCAT, 896
VECTCAT, 811
csc
COMPCAT, 1625
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
TRIGCAT, 64
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
csch
COMPCAT, 1625
HYPCAT, 33
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
cschIfCan
PTRANFN, 43
cscIfCan
PTRANFN, 43
curve
SPACEC, 258
curve?
SPACEC, 258
curveV
INFCLCT, 1366
cycle
PERMCAT, 582
cycleEntry
ALAGG, 824
LSAGG, 663
LZSTAGG, 640
STAGG, 588
URAGG, 509
cycleLength
ALAGG, 824
LSAGG, 663
LZSTAGG, 640
STAGG, 588
URAGG, 509
cycles
PERMCAT, 582
cycleTail

ALAGG, 824
LSAGG, 663
LZSTAGG, 640
STAGG, 588
URAGG, 509
cyclic?
ALAGG, 824
BRAGG, 374
BTCAT, 528
DLAGG, 394
DSTRCAT, 133
LSAGG, 663
LZSTAGG, 640
RCAGG, 353
STAGG, 588
URAGG, 509

D
ACFS, 1500
COMPCAT, 1625
DIFEXT, 955
DIFRING, 850
DIRPCAT, 1004
DPOLCAT, 1310
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FS, 1341
INS, 1231
LOCPOWC, 1243
LODOCAT, 973
MONOGEN, 1602
MTSCAT, 1196
PACFFC, 1548
PDRING, 887
POLYCAT, 1260
QFCAT, 1380
QUATCAT, 1073
RPOLCAT, 1414
SMATCAT, 1085
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
dec
INS, 1231
decompose
FDIVCAT, 618
decreasePrecision
FPS, 1532
deepestInitial
RPOLCAT, 1414
deepestTail
RPOLCAT, 1414
definingField
AFSPCAT, 273
PRSPCAT, 346
definingPolynomial
ACFS, 1500
COMPCAT, 1625
ES, 302
FAXF, 1587
FFCAT, 1647
FS, 1341
MONOGEN, 1602
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
RRCC, 231
degree
AFSPCAT, 273
AMR, 1103
DIVCAT, 1181
DPOLCAT, 1310
FAMR, 1143
FAXF, 1587
FLALG, 1043
GRALG, 314
GRMOD, 177
INFCLCT, 1366
LOCPOWC, 1243
LODOCAT, 973
MLO, 1055
MTSCAT, 1196
NTSCAT, 749
OREPCAT, 931
PACEXTC, 1668
PACRATC, 1613

PLACESC, 341
POLYCAT, 1260
PRSPCAT, 346
PSCAT, 1161
RPOLCAT, 1414
RSETCAT, 703
SETCATD, 244
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
XF, 1513
XPOLYC, 1096
delay
LOCPOWC, 1243
delete
A1AGG, 687
ALAGG, 824
BTAGG, 734
ELAGG, 550
FLAGG, 558
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
PTCAT, 896
SRAGG, 789
STAGG, 588
STRICAT, 923
VECTCAT, 811
denom
ACFS, 1500
FS, 1341
QFCAT, 1380
ULSCCAT, 1559
denominator
ACFS, 1500
FS, 1341
QFCAT, 1380
ULSCCAT, 1559
depth
DQAGG, 542
SKAGG, 502
dequeue
DQAGG, 542
DequeueAggregate, 540
derivationCoordinates
COMPCAT, 1625
FFCAT, 1647
MONOGEN, 1602
DesingTreeCategory, 131
destruct
SEXCAT, 248
determinant
MATCAT, 437
SMATCAT, 1085
DIAGG, 533
?=?, 535
?~=?, 535
#?, 535
any?, 535
bag, 535
coerce, 535
construct, 535
convert, 535
copy, 535
count, 535
dictionary, 535
empty, 535
empty?, 535
eq?, 535
eval, 535
every?, 535
extract, 535
find, 535
finiteAggregate, 535
hash, 535
insert, 535
inspect, 535
latex, 535
less?, 535
map, 535
member?, 535
members, 535
more?, 535
parts, 535
reduce, 535

remove, 535
removeDuplicates, 535
sample, 535
select, 535
shallowlyMutable, 535
size?, 535
diagonal
SMATCAT, 1085
diagonal?
MATCAT, 437
RMATCAT, 903
SMATCAT, 1085
diagonalMatrix
MATCAT, 437
SMATCAT, 1085
diagonalProduct
SMATCAT, 1085
Dictionary, 533
dictionary
ALAGG, 824
DIAGG, 535
DIOPS, 387
FSAGG, 624
KDAGG, 632
MDAGG, 572
MSETAGG, 675
OMSAGG, 765
TBAGG, 800
DictionaryOperations, 385
DIFEXT, 953
-?, 955
?**?, 955
?*?, 955
?+?, 955
?-?, 955
?=?, 955
?^?, 955
?~=?, 955
0, 955
1, 955
characteristic, 955
coerce, 955
D, 955
differentiate, 955
hash, 955
latex, 955
one?, 955
recip, 955
sample, 955
subtractIfCan, 955
unitsKnown, 955
zero?, 955
difference
FSAGG, 624
MSETAGG, 675
OMSAGG, 765
SETAGG, 494
DifferentialExtension, 953
DifferentialPolynomialCategory, 1306
DifferentialRing, 848
DifferentialVariableCategory, 294
differentialVariables
DPOLCAT, 1310
differentiate
ACFS, 1500
COMPCAT, 1625
DIFEXT, 955
DIFRING, 850
DIRPCAT, 1004
DPOLCAT, 1310
DVARCAT, 296
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FS, 1341
INS, 1231
LOCPOWC, 1243
MONOGEN, 1602
MTSCAT, 1196
PACFFC, 1548
PDRING, 887
POLYCAT, 1260
QFCAT, 1380
QUATCAT, 1073
RPOLCAT, 1414
SMATCAT, 1085
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574

UTSCAT, 1281
DIFRING, 848
-?, 850
?**?, 850
?*?, 850
?+?, 850
?-?, 850
?=?, 850
?^?, 850
?~=?, 850
0, 850
1, 850
characteristic, 850
coerce, 850
D, 850
differentiate, 850
hash, 850
latex, 850
one?, 850
recip, 850
sample, 850
subtractIfCan, 850
unitsKnown, 850
zero?, 850
digamma
SPFCAT, 60
digits
FPS, 1532
PADICCT, 1251
dilog
LFCAT, 193
dimension
DIRPCAT, 1004
FAXF, 1587
PACEXTC, 1668
PACRATC, 1613
PTCAT, 896
VSPACE, 989
XF, 1513
DIOPS, 385
?=?, 387
?~=?, 387
#?, 387
any?, 387
bag, 387
coerce, 387
construct, 387
convert, 387
copy, 387
count, 387
dictionary, 387
empty, 387
empty?, 387
eq?, 387
eval, 387
every?, 387
extract, 387
find, 387
finiteAggregate, 388
hash, 387
insert, 387
inspect, 387
latex, 387
less?, 387
map, 387
member?, 387
members, 387
more?, 387
parts, 387
reduce, 387
remove, 387
removeDuplicates, 387
sample, 387
select, 387
shallowlyMutable, 387
size?, 387
directory
FNCAT, 173
directProduct
DIRPCAT, 1004
DirectProductCategory, 1001
directSum
LODOCAT, 973
DIRPCAT, 1001
-?, 1004
?<?, 1004
?<=?, 1004
?>?, 1004
?>=?, 1004
?**?, 1004
?*?, 1004
?+?, 1004

?-?, 1004
?.?, 1004
?/?, 1004
?=?, 1004
?^?, 1004
?~=?, 1004
#?, 1004
0, 1004
1, 1004
abs, 1004
any?, 1004
characteristic, 1004
coerce, 1004
commutative("*"), 1005
copy, 1004
count, 1004
D, 1004
differentiate, 1004
dimension, 1004
directProduct, 1004
dot, 1004
elt, 1004
empty, 1004
empty?, 1004
entries, 1004
entry?, 1004
eq?, 1004
eval, 1004
every?, 1004
fill, 1004
finiteAggregate, 1005
first, 1004
hash, 1004
index, 1004
index?, 1004
indices, 1004
latex, 1004
leftUnitary, 1005
less?, 1004
lookup, 1004
map, 1004
max, 1004
maxIndex, 1004
member?, 1004
members, 1004
min, 1004
minIndex, 1004
more?, 1004
negative?, 1004
one?, 1004
parts, 1004
positive?, 1004
qelt, 1004
qsetelt, 1004
random, 1004
recip, 1004
reducedSystem, 1004
retract, 1004
retractIfCan, 1004
rightUnitary, 1005
sample, 1004
setelt, 1004
sign, 1004
size, 1004
size?, 1004
subtractIfCan, 1004
sup, 1004
swap, 1004
unitsKnown, 1005
unitVector, 1004
zero?, 1004
discreteLog
COMPCAT, 1625
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FPC, 1325
MONOGEN, 1602
PACEXTC, 1668
PACFFC, 1548
PACRATC, 1613
XF, 1513
discriminant
COMPCAT, 1625
DPOLCAT, 1310
FFCAT, 1647
FINRALG, 1331
FRAMALG, 1541
MONOGEN, 1602
POLYCAT, 1260
RPOLCAT, 1414
UPOLYC, 1477

distance
ALAGG, 824
BRAGG, 374
BTCAT, 528
DLAGG, 394
DSTRCAT, 133
LSAGG, 663
LZSTAGG, 640
RCAGG, 353
STAGG, 588
URAGG, 509
distinguishedRootsOf
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
distribute
ACFS, 1500
ES, 302
FS, 1341
DIVCAT, 1179
-?, 1181
?<=?, 1181
?*?, 1181
?+?, 1181
?-?, 1181
?=?, 1181
?~=?, 1181
0, 1181
coefficient, 1181
coerce, 1181
collect, 1181
concat, 1181
degree, 1181
divOfPole, 1181
divOfZero, 1181
effective?, 1181
hash, 1181
highCommonTerms, 1181
incr, 1181
latex, 1181
mapCoef, 1181
mapGen, 1181
nthCoef, 1181
nthFactor, 1181
retract, 1181
retractIfCan, 1181
sample, 1181
size, 1181
split, 1181
subtractIfCan, 1181
supp, 1181
suppOfPole, 1181
suppOfZero, 1181
terms, 1181
zero?, 1181
divide
ACF, 1298
ACFS, 1500
COMPCAT, 1625
EUCDOM, 1187
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FPC, 1325
FPS, 1532
FS, 1341
INS, 1231
LOCPOWC, 1243
MONOGEN, 1602
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPXSCAT, 1464
UPXSCCA, 1574
XF, 1513
divideExponents
UPOLYC, 1477
DivisionRing, 1013
divisor
FDIVCAT, 618
DivisorCategory, 1179
divOfPole

DIVCAT, 1181
divOfZero
DIVCAT, 1181
DIVRING, 1013
-?, 1014
?**?, 1014
?*?, 1014
?+?, 1014
?-?, 1014
?=?, 1014
?^?, 1014
?~=?, 1014
0, 1014
1, 1014
characteristic, 1014
coerce, 1014
hash, 1014
inv, 1014
latex, 1014
leftUnitary, 1015
noZeroDivisors, 1014
one?, 1014
recip, 1014
rightUnitary, 1015
sample, 1014
subtractIfCan, 1014
unitsKnown, 1015
zero?, 1014
DLAGG, 392
?.value, 394
?=?, 394
?~=?, 394
#?, 394
any?, 394
child?, 394
children, 394
coerce, 394
concat, 394
copy, 394
count, 394
cyclic?, 394
distance, 394
empty, 394
empty?, 394
eq?, 394
eval, 394
every?, 394
hash, 394
head, 394
last, 394
latex, 394
leaf?, 394
leaves, 394
less?, 394
map, 394
member?, 394
members, 394
more?, 394
next, 394
node?, 394
nodes, 394
parts, 394
previous, 394
sample, 394
setchildren, 394
setelt, 394
setnext, 394
setprevious, 394
setvalue, 394
shallowlyMutable, 394
size?, 394
tail, 394
value, 394
dot
DIRPCAT, 1004
PTCAT, 896
VECTCAT, 811
DoublyLinkedAggregate, 392
DPOLCAT, 1306
-?, 1310
?<?, 1310
?<=?, 1310
?>?, 1310
?>=?, 1310
?**?, 1310
?*?, 1310
?+?, 1310
?-?, 1310
?/?, 1310
?=?, 1310
?^?, 1310
?~=?, 1310

0, 1310
1, 1310
associates?, 1310
binomThmExpt, 1310
canonicalUnitNormal, 1310
characteristic, 1310
charthRoot, 1310
coefficient, 1310
coefficients, 1310
coerce, 1310
commutative("*"), 1311
conditionP, 1310
content, 1310
convert, 1310
D, 1310
degree, 1310
differentialVariables, 1310
differentiate, 1310
discriminant, 1310
eval, 1310
exquo, 1310
factor, 1310
factorPolynomial, 1310
factorSquareFreePolynomial, 1310
gcd, 1310
gcdPolynomial, 1310
ground, 1310
ground?, 1310
hash, 1310
initial, 1310
isExpt, 1310
isobaric?, 1310
isPlus, 1310
isTimes, 1310
latex, 1310
lcm, 1310
leader, 1310
leadingCoefficient, 1310
leadingMonomial, 1310
leftUnitary, 1311
makeVariable, 1310
map, 1310
mapExponents, 1310
max, 1310
min, 1310
minimumDegree, 1310
monicDivide, 1310
monomial, 1310
monomial?, 1310
monomials, 1310
multivariate, 1310
noZeroDivisors, 1311
numberOfMonomials, 1310
one?, 1310
order, 1310
patternMatch, 1310
pomopo, 1310
prime?, 1310
primitiveMonomials, 1310
primitivePart, 1310
recip, 1310
reducedSystem, 1310
reductum, 1310
resultant, 1310
retract, 1310
retractIfCan, 1310
rightUnitary, 1311
sample, 1310
separant, 1310
solveLinearPolynomialEquation, 1310
squareFree, 1310
squareFreePart, 1310
squareFreePolynomial, 1310
subtractIfCan, 1310
totalDegree, 1310
unit?, 1310
unitCanonical, 1310
unitNormal, 1310
unitsKnown, 1311
univariate, 1310
variables, 1310
weight, 1310
weights, 1310
zero?, 1310
DQAGG, 540
?=?, 542
?~=?, 542
#?, 542
any?, 542
back, 542
bag, 542
bottom, 542

coerce, 542
copy, 542
count, 542
depth, 542
dequeue, 542
empty, 542
empty?, 542
enqueue, 542
eq?, 542
eval, 542
every?, 542
extract, 542
extractBottom, 542
extractTop, 542
finiteAggregate, 542
front, 542
hash, 542
height, 542
insert, 542
insertBottom, 542
insertTop, 542
inspect, 542
latex, 542
length, 542
less?, 542
map, 542
member?, 542
members, 542
more?, 542
parts, 542
pop, 542
push, 542
reverse, 542
rotate, 542
sample, 542
shallowlyMutable, 542
size?, 542
top, 542
DSTRCAT, 131
?.value, 133
?=?, 133
?~=?, 133
#?, 133
any?, 133
child?, 133
children, 133
coerce, 133
copy, 133
count, 133
cyclic?, 133
distance, 133
empty, 133
empty?, 133
eq?, 133
eval, 133
every?, 133
hash, 133
latex, 133
leaf?, 133
leaves, 133
less?, 133
map, 133
member?, 133
members, 133
more?, 133
node?, 133
nodes, 133
parts, 133
sample, 133
setchildren, 133
setelt, 133
setvalue, 133
shallowlyMutable, 133
size?, 133
tree, 133
value, 133
duplicates
MDAGG, 572
MSETAGG, 675
OMSAGG, 765
DVARCAT, 294
?<?, 296
?<=?, 296
?>?, 296
?>=?, 296
?=?, 296
?~=?, 296
coerce, 296
differentiate, 296
hash, 296
latex, 296
makeVariable, 296

max, 296
min, 296
order, 296
retract, 296
retractIfCan, 296
variable, 296
weight, 296

effective?
DIVCAT, 1181
Ei
LFCAT, 193
ELAGG, 547
?.?, 550
?=?, 550
?~=?, 550
#?, 550
any?, 550
coerce, 550
concat, 550
construct, 550
convert, 550
copy, 550
count, 550
delete, 550
elt, 550
empty, 550
empty?, 550
entries, 550
entry?, 550
eq?, 550
eval, 550
every?, 550
fill, 550
find, 550
first, 550
hash, 550
index?, 550
indices, 550
insert, 550
latex, 550
less?, 550
map, 550
maxIndex, 550
member?, 550
members, 550
merge, 550
minIndex, 550
more?, 550
new, 550
parts, 550
qelt, 550
qsetelt, 550
reduce, 550
remove, 550
removeDuplicates, 550
sample, 550
select, 550
setelt, 550
shallowlyMutable, 550
size?, 550
swap, 550
ElementaryFunctionCategory, 27
ELEMFUN, 27
?**?, 28
exp, 28
log, 28
elliptic
FFCAT, 1647
elt
A1AGG, 687
ACFS, 1500
ALAGG, 824
ARR2CAT, 359
BTAGG, 734
DIRPCAT, 1004
ELAGG, 550
ELTAGG, 77
ES, 302
FFCAT, 1647
FLAGG, 558
FS, 1341
IXAGG, 320
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
PTCAT, 896
RMATCAT, 903
SMATCAT, 1085
SRAGG, 789
STAGG, 588

STRICAT, 923
TBAGG, 800
UPOLYC, 1477
VECTCAT, 811
ELTAB, 29
?.?, 30
Eltable, 29
EltableAggregate, 76
ELTAGG, 76
?.?, 77
elt, 77
qelt, 77
qsetelt, 77
setelt, 77
shallowlyMutable, 77
empty
A1AGG, 687
ALAGG, 824
ARR2CAT, 359
BGAGG, 279
BRAGG, 374
BTAGG, 734
BTCAT, 528
CLAGG, 289
DIAGG, 535
DIOPS, 387
DIRPCAT, 1004
DLAGG, 394
DQAGG, 542
DSTRCAT, 133
ELAGG, 550
FLAGG, 558
FSAGG, 624
HOAGG, 183
IXAGG, 320
KDAGG, 632
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
MDAGG, 572
MSETAGG, 675
NTSCAT, 749
OMSAGG, 765
PRQAGG, 482
PSETCAT, 468
PTCAT, 896
QUAGG, 488
RCAGG, 353
RMATCAT, 903
RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SKAGG, 502
SMATCAT, 1085
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
URAGG, 509
VECTCAT, 811
empty?
A1AGG, 687
AGG, 70
ALAGG, 824
ARR2CAT, 359
BGAGG, 279
BRAGG, 374
BTAGG, 734
BTCAT, 528
CLAGG, 289
DIAGG, 535
DIOPS, 387
DIRPCAT, 1004
DLAGG, 394
DQAGG, 542
DSTRCAT, 133
ELAGG, 550
FLAGG, 558
FSAGG, 624
HOAGG, 183
IXAGG, 320
KDAGG, 632
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
MDAGG, 572
MSETAGG, 675
NTSCAT, 749

OMSAGG, 765
PRQAGG, 482
PSETCAT, 468
PTCAT, 896
QUAGG, 488
RCAGG, 353
RMATCAT, 903
RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SKAGG, 502
SMATCAT, 1085
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
URAGG, 509
VECTCAT, 811
enterPointData
SPACEC, 258
ENTIRER, 853
-?, 855
?**?, 855
?*?, 855
?+?, 855
?-?, 855
?=?, 855
?^?, 855
?~=?, 855
0, 855
1, 855
characteristic, 855
coerce, 855
hash, 855
latex, 855
leftUnitary, 855
noZeroDivisors, 855
one?, 855
recip, 855
rightUnitary, 855
sample, 855
subtractIfCan, 855
unitsKnown, 855
zero?, 855
EntireRing, 853
entries
A1AGG, 687
ALAGG, 824
BTAGG, 734
DIRPCAT, 1004
ELAGG, 550
FLAGG, 558
IXAGG, 320
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
PTCAT, 896
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
VECTCAT, 811
entry?
A1AGG, 687
ALAGG, 824
BTAGG, 734
DIRPCAT, 1004
ELAGG, 550
FLAGG, 558
IXAGG, 320
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
PTCAT, 896
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
VECTCAT, 811
eq
SEXCAT, 248
eq?
A1AGG, 687
AGG, 70
ALAGG, 824
ARR2CAT, 359
BGAGG, 279
BRAGG, 374
BTAGG, 734
BTCAT, 528

CLAGG, 289
DIAGG, 535
DIOPS, 387
DIRPCAT, 1004
DLAGG, 394
DQAGG, 542
DSTRCAT, 133
ELAGG, 550
FLAGG, 558
FSAGG, 624
HOAGG, 183
IXAGG, 320
KDAGG, 632
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
MDAGG, 572
MSETAGG, 675
NTSCAT, 749
OMSAGG, 765
PRQAGG, 482
PSETCAT, 468
PTCAT, 896
QUAGG, 488
RCAGG, 353
RMATCAT, 903
RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SKAGG, 502
SMATCAT, 1085
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
URAGG, 509
VECTCAT, 811
erf
LFCAT, 193
ES, 300
?<?, 302
?<=?, 302
?>?, 302
?>=?, 302
?=?, 302
?~=?, 302
belong?, 302
box, 302
coerce, 302
definingPolynomial, 302
distribute, 302
elt, 302
eval, 302
even?, 302
freeOf?, 302
hash, 302
height, 302
is?, 302
kernel, 302
kernels, 302
latex, 302
mainKernel, 302
map, 302
max, 302
min, 302
minPoly, 302
odd?, 302
operator, 302
operators, 302
paren, 302
retract, 302
retractIfCan, 302
subst, 302
tower, 302
EUCDOM, 1185
-?, 1187
?**?, 1187
?*?, 1187
?+?, 1187
?-?, 1187
?=?, 1187
?^?, 1187
?~=?, 1187
?quo?, 1187
?rem?, 1187
0, 1187
1, 1187
associates?, 1187
characteristic, 1187

coerce, 1187
commutative("*"), 1187
divide, 1187
euclideanSize, 1187
expressIdealMember, 1187
exquo, 1187
extendedEuclidean, 1187
gcd, 1187
gcdPolynomial, 1187
hash, 1187
latex, 1187
lcm, 1187
leftUnitary, 1187
multiEuclidean, 1187
noZeroDivisors, 1187
one?, 1187
principalIdeal, 1187
recip, 1187
rightUnitary, 1187
sample, 1187
sizeLess?, 1187
subtractIfCan, 1187
unit?, 1187
unitCanonical, 1187
unitNormal, 1187
unitsKnown, 1187
zero?, 1187
EuclideanDomain, 1185
euclideanSize
ACF, 1298
ACFS, 1500
COMPCAT, 1625
EUCDOM, 1187
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FPC, 1325
FPS, 1532
FS, 1341
INS, 1231
LOCPOWC, 1243
MONOGEN, 1602
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPXSCAT, 1464
UPXSCCA, 1574
XF, 1513
eval
A1AGG, 687
ACFS, 1500
ALAGG, 824
ARR2CAT, 359
BGAGG, 279
BRAGG, 374
BTAGG, 734
BTCAT, 528
CLAGG, 289
COMPCAT, 1625
DIAGG, 535
DIOPS, 387
DIRPCAT, 1004
DLAGG, 394
DPOLCAT, 1310
DQAGG, 542
DSTRCAT, 133
ELAGG, 550
ES, 302
EVALAB, 81
FEVALAB, 160
FLAGG, 558
FLALG, 1043
FS, 1341
FSAGG, 624
HOAGG, 183
IEVALAB, 37
IXAGG, 320
KDAGG, 632
LNAGG, 407
LOCPOWC, 1243
LSAGG, 663
LZSTAGG, 640
MATCAT, 437

MDAGG, 572
MSETAGG, 675
MTSCAT, 1196
NTSCAT, 749
OC, 1062
OMSAGG, 765
PERMCAT, 582
POLYCAT, 1260
PRQAGG, 482
PSETCAT, 468
PTCAT, 896
QFCAT, 1380
QUAGG, 488
QUATCAT, 1073
RCAGG, 353
RMATCAT, 903
RPOLCAT, 1414
RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SKAGG, 502
SMATCAT, 1085
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
URAGG, 509
UTSCAT, 1281
VECTCAT, 811
EVALAB, 80
eval, 81
Evalable, 80
even?
ACFS, 1500
ES, 302
FS, 1341
INS, 1231
every?
A1AGG, 687
ALAGG, 824
ARR2CAT, 359
BGAGG, 279
BRAGG, 374
BTAGG, 734
BTCAT, 528
CLAGG, 289
DIAGG, 535
DIOPS, 387
DIRPCAT, 1004
DLAGG, 394
DQAGG, 542
DSTRCAT, 133
ELAGG, 550
FLAGG, 558
FSAGG, 624
HOAGG, 183
IXAGG, 320
KDAGG, 632
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
MDAGG, 572
MSETAGG, 675
NTSCAT, 749
OMSAGG, 765
PRQAGG, 482
PSETCAT, 468
PTCAT, 896
QUAGG, 488
RCAGG, 353
RMATCAT, 903
RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SKAGG, 502
SMATCAT, 1085
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
URAGG, 509

VECTCAT, 811
exactQuotient
RPOLCAT, 1414
excepCoord
BLMETCT, 128
excpDivV
INFCLCT, 1366
exists?
FNCAT, 173
exp
COMPCAT, 1625
ELEMFUN, 28
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
expand
SEGXCAT, 236
expIfCan
PTRANFN, 43
explicitEntries?
LZSTAGG, 640
explicitlyEmpty?
LZSTAGG, 640
explicitlyFinite?
ALAGG, 824
LSAGG, 663
LZSTAGG, 640
STAGG, 588
exponent
FPS, 1532
expr
SEXCAT, 248
expressIdealMember
ACF, 1298
ACFS, 1500
COMPCAT, 1625
EUCDOM, 1187
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FPC, 1325
FPS, 1532
FS, 1341
INS, 1231
LOCPOWC, 1243
MONOGEN, 1602
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PID, 1168
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPXSCAT, 1464
UPXSCCA, 1574
XF, 1513
ExpressionSpace, 300
exquo
ACF, 1298
ACFS, 1500
AMR, 1103
COMPCAT, 1625
DPOLCAT, 1310
EUCDOM, 1187
FAMR, 1143
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FMTC, 1110
FPC, 1325
FPS, 1532
FS, 1341
GCDDOM, 1131
INS, 1231
INTCAT, 1153
INTDOM, 1049
LOCPOWC, 1243
LODOCAT, 973
MATCAT, 437

MONOGEN, 1602
MTSCAT, 1196
OINTDOM, 1137
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PFECAT, 1204
PID, 1168
POLYCAT, 1260
PSCAT, 1161
QFCAT, 1380
RCFIELD, 1392
RMATCAT, 903
RNS, 1403
RPOLCAT, 1414
SMATCAT, 1085
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
XF, 1513
extDegree
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
extend
LOCPOWC, 1243
LZSTAGG, 640
MTSCAT, 1196
NTSCAT, 749
PADICCT, 1251
PTCAT, 896
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
ULSCAT, 1452
ULSCCAT, 1559
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
extendedEuclidean
ACF, 1298
ACFS, 1500
COMPCAT, 1625
EUCDOM, 1187
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FPC, 1325
FPS, 1532
FS, 1341
INS, 1231
LOCPOWC, 1243
MONOGEN, 1602
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPXSCAT, 1464
UPXSCCA, 1574
XF, 1513
extendedSubResultantGcd
RPOLCAT, 1414
extendIfCan
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
ExtensibleLinearAggregate, 547
extension
FNCAT, 173
extensionDegree
FAXF, 1587

PACEXTC, 1668
PACRATC, 1613
XF, 1513
ExtensionField, 1510

factor
ACF, 1298
ACFS, 1500
COMPCAT, 1625
DPOLCAT, 1310
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FPC, 1325
FPS, 1532
FS, 1341
INS, 1231
LOCPOWC, 1243
MONOGEN, 1602
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PFECAT, 1204
POLYCAT, 1260
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
RPOLCAT, 1414
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPXSCAT, 1464
UPXSCCA, 1574
XF, 1513
factorial
CFCAT, 21
COMBOPC, 74
INS, 1231
factorials
COMBOPC, 74
factorPolynomial
COMPCAT, 1625
DPOLCAT, 1310
PFECAT, 1204
POLYCAT, 1260
QFCAT, 1380
RPOLCAT, 1414
ULSCCAT, 1559
UPOLYC, 1477
factorsOfCyclicGroupSize
COMPCAT, 1625
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
MONOGEN, 1602
PACFFC, 1548
factorSquareFreePolynomial
COMPCAT, 1625
DPOLCAT, 1310
PFECAT, 1204
POLYCAT, 1260
QFCAT, 1380
RPOLCAT, 1414
ULSCCAT, 1559
UPOLYC, 1477
FAMONC, 564
?*?, 565
?+?, 565
?=?, 565
?~=?, 565
0, 565
coefficient, 565
coerce, 565
hash, 565
highCommonTerms, 565
latex, 565
mapCoef, 565
mapGen, 565
nthCoef, 565
nthFactor, 565
retract, 565
retractIfCan, 565
sample, 565
size, 565
subtractIfCan, 565
terms, 565
zero?, 565
FAMR, 1141
-?, 1143

?**?, 1143
?*?, 1143
?+?, 1143
?-?, 1143
?/?, 1143
?=?, 1143
?^?, 1143
?~=?, 1143
0, 1143
1, 1143
associates?, 1143
binomThmExpt, 1143
characteristic, 1143
charthRoot, 1143
coefficient, 1143
coefficients, 1143
coerce, 1143
commutative("*"), 1143
content, 1143
degree, 1143
exquo, 1143
ground, 1143
ground?, 1143
hash, 1143
latex, 1143
leadingCoefficient, 1143
leadingMonomial, 1143
leftUnitary, 1144
map, 1143
mapExponents, 1143
minimumDegree, 1143
monomial, 1143
monomial?, 1143
noZeroDivisors, 1143
numberOfMonomials, 1143
one?, 1143
pomopo, 1143
primitivePart, 1143
recip, 1143
reductum, 1143
retract, 1143
retractIfCan, 1143
rightUnitary, 1144
sample, 1143
subtractIfCan, 1143
unit?, 1143
unitCanonical, 1143
unitNormal, 1143
unitsKnown, 1143
zero?, 1143
FAXF, 1583
-?, 1587
?**?, 1587
?*?, 1587
?+?, 1587
?-?, 1587
?/?, 1587
?=?, 1587
?^?, 1587
?~=?, 1587
?quo?, 1587
?rem?, 1587
0, 1587
1, 1587
algebraic?, 1587
associates?, 1587
basis, 1587
canonicalsClosed, 1587
canonicalUnitNormal, 1587
characteristic, 1587
charthRoot, 1587
coerce, 1587
commutative("*"), 1587
conditionP, 1587
coordinates, 1587
createNormalElement, 1587
createPrimitiveElement, 1587
D, 1587
definingPolynomial, 1587
degree, 1587
differentiate, 1587
dimension, 1587
discreteLog, 1587
divide, 1587
euclideanSize, 1587
expressIdealMember, 1587
exquo, 1587
extendedEuclidean, 1587
extensionDegree, 1587
factor, 1587
factorsOfCyclicGroupSize, 1587
Frobenius, 1587

gcd, 1587
gcdPolynomial, 1587
generator, 1587
hash, 1587
index, 1587
inGroundField?, 1587
init, 1587
inv, 1587
latex, 1587
lcm, 1587
leftUnitary, 1587
linearAssociatedExp, 1587
linearAssociatedLog, 1587
linearAssociatedOrder, 1587
lookup, 1587
minimalPolynomial, 1587
multiEuclidean, 1587
nextItem, 1587
norm, 1587
normal?, 1587
normalElement, 1587
noZeroDivisors, 1587
one?, 1587
order, 1587
prime?, 1587
primeFrobenius, 1587
primitive?, 1587
primitiveElement, 1587
principalIdeal, 1587
random, 1587
recip, 1587
representationType, 1587
represents, 1587
retract, 1587
retractIfCan, 1587
rightUnitary, 1588
sample, 1587
size, 1587
sizeLess?, 1587
squareFree, 1587
squareFreePart, 1587
subtractIfCan, 1587
tableForDiscreteLogarithm, 1587
trace, 1587
transcendenceDegree, 1587
transcendent?, 1587
unit?, 1587
unitCanonical, 1587
unitNormal, 1587
unitsKnown, 1587
zero?, 1587
FDIVCAT, 616
-?, 618
?*?, 618
?+?, 618
?-?, 618
?=?, 618
?~=?, 618
0, 618
coerce, 618
decompose, 618
divisor, 618
generator, 618
hash, 618
ideal, 618
latex, 618
principal?, 618
reduce, 618
sample, 618
subtractIfCan, 618
zero?, 618
FEVALAB, 159
?.?, 160
eval, 160
map, 160
FFCAT, 1642
-?, 1647
?**?, 1647
?*?, 1647
?+?, 1647
?-?, 1647
?/?, 1647
?=?, 1647
?^?, 1647
?~=?, 1647
?quo?, 1647
?rem?, 1647
0, 1647
1, 1647
absolutelyIrreducible?, 1647
algSplitSimple, 1647
associates?, 1647

basis, 1647
branchPoint?, 1647
branchPointAtInfinity?, 1647
canonicalsClosed, 1648
canonicalUnitNormal, 1648
characteristic, 1647
characteristicPolynomial, 1647
charthRoot, 1647
coerce, 1647
commutative("*"), 1648
complementaryBasis, 1647
conditionP, 1647
convert, 1647
coordinates, 1647
createPrimitiveElement, 1647
D, 1647
definingPolynomial, 1647
derivationCoordinates, 1647
differentiate, 1647
discreteLog, 1647
discriminant, 1647
divide, 1647
elliptic, 1647
elt, 1647
euclideanSize, 1647
expressIdealMember, 1647
exquo, 1647
extendedEuclidean, 1647
factor, 1647
factorsOfCyclicGroupSize, 1647
gcd, 1647
gcdPolynomial, 1647
generator, 1647
genus, 1647
hash, 1647
hyperelliptic, 1647
index, 1647
init, 1647
integral?, 1647
integralAtInfinity?, 1647
integralBasis, 1647
integralBasisAtInfinity, 1647
integralCoordinates, 1647
integralDerivationMatrix, 1647
integralMatrix, 1647
integralMatrixAtInfinity, 1647
integralRepresents, 1647
inv, 1647
inverseIntegralMatrix, 1647
inverseIntegralMatrixAtInfinity, 1647
latex, 1647
lcm, 1647
leftUnitary, 1648
lift, 1647
lookup, 1647
minimalPolynomial, 1647
multiEuclidean, 1647
nextItem, 1647
nonSingularModel, 1647
norm, 1647
normalizeAtInfinity, 1647
noZeroDivisors, 1648
numberOfComponents, 1647
one?, 1647
order, 1647
prime?, 1647
primeFrobenius, 1647
primitive?, 1647
primitiveElement, 1647
primitivePart, 1647
principalIdeal, 1647
ramified?, 1647
ramifiedAtInfinity?, 1647
random, 1647
rank, 1647
rationalPoint?, 1647
rationalPoints, 1647
recip, 1647
reduce, 1647
reduceBasisAtInfinity, 1647
reducedSystem, 1647
regularRepresentation, 1647
representationType, 1647
represents, 1647
retract, 1647
retractIfCan, 1647
rightUnitary, 1648
sample, 1647
singular?, 1647
singularAtInfinity?, 1647
size, 1647
sizeLess?, 1647

squareFree, 1647
squareFreePart, 1647
subtractIfCan, 1647
tableForDiscreteLogarithm, 1647
trace, 1647
traceMatrix, 1647
unit?, 1647
unitCanonical, 1647
unitNormal, 1647
unitsKnown, 1648
yCoordinates, 1647
zero?, 1647
FFIELDC, 1518
-?, 1521
?**?, 1521
?*?, 1521
?+?, 1521
?-?, 1521
?/?, 1521
?=?, 1521
?^?, 1521
?~=?, 1521
?quo?, 1521
?rem?, 1521
0, 1521
1, 1521
associates?, 1521
canonicalsClosed, 1521
canonicalUnitNormal, 1521
characteristic, 1521
charthRoot, 1521
coerce, 1521
commutative("*"), 1521
conditionP, 1521
createPrimitiveElement, 1521
D, 1521
differentiate, 1521
discreteLog, 1521
divide, 1521
euclideanSize, 1521
expressIdealMember, 1521
exquo, 1521
extendedEuclidean, 1521
factor, 1521
factorsOfCyclicGroupSize, 1521
gcd, 1521
gcdPolynomial, 1521
hash, 1521
index, 1521
init, 1521
inv, 1521
latex, 1521
lcm, 1521
leftUnitary, 1521
lookup, 1521
multiEuclidean, 1521
nextItem, 1521
noZeroDivisors, 1521
one?, 1521
order, 1521
prime?, 1521
primeFrobenius, 1521
primitive?, 1521
primitiveElement, 1521
principalIdeal, 1521
random, 1521
recip, 1521
representationType, 1521
rightUnitary, 1521
sample, 1521
size, 1521
sizeLess?, 1521
squareFree, 1521
squareFreePart, 1521
subtractIfCan, 1521
tableForDiscreteLogarithm, 1521
unit?, 1521
unitCanonical, 1521
unitNormal, 1521
unitsKnown, 1521
zero?, 1521
FIELD, 1221
-?, 1224
?**?, 1224
?*?, 1224
?+?, 1224
?-?, 1224
?/?, 1224
?=?, 1224
?^?, 1224
?~=?, 1224
?quo?, 1224

?rem?, 1224
0, 1224
1, 1224
associates?, 1224
canonicalsClosed, 1224
canonicalUnitNormal, 1224
characteristic, 1224
coerce, 1224
commutative("*"), 1224
divide, 1224
euclideanSize, 1224
expressIdealMember, 1224
exquo, 1224
extendedEuclidean, 1224
factor, 1224
gcd, 1224
gcdPolynomial, 1224
hash, 1224
inv, 1224
latex, 1224
lcm, 1224
leftUnitary, 1224
multiEuclidean, 1224
noZeroDivisors, 1224
one?, 1224
prime?, 1224
principalIdeal, 1224
recip, 1224
rightUnitary, 1224
sample, 1224
sizeLess?, 1224
squareFree, 1224
squareFreePart, 1224
subtractIfCan, 1224
unit?, 1224
unitCanonical, 1224
unitNormal, 1224
unitsKnown, 1224
zero?, 1224
Field, 1221
FieldOfPrimeCharacteristic, 1323
FILECAT, 163
?=?, 165
?~=?, 165
close, 165
coerce, 165
hash, 165
iomode, 165
latex, 165
name, 165
open, 165
read, 165
reopen, 165
write, 165
FileCategory, 163
filename
FNCAT, 173
FileNameCategory, 172
filterUpTo
LOCPOWC, 1243
FINAALG, 1019
-?, 1022
?**?, 1022
?*?, 1022
?+?, 1022
?-?, 1022
?=?, 1022
?~=?, 1022
0, 1022
alternative?, 1022
antiAssociative?, 1022
antiCommutative?, 1022
antiCommutator, 1022
associative?, 1022
associator, 1022
associatorDependence, 1022
coerce, 1022
commutative?, 1022
commutator, 1022
conditionsForIdempotents, 1022
coordinates, 1022
flexible?, 1022
hash, 1022
jacobiIdentity?, 1022
jordanAdmissible?, 1022
jordanAlgebra?, 1022
latex, 1022
leftAlternative?, 1022
leftCharacteristicPolynomial, 1022
leftDiscriminant, 1022
leftMinimalPolynomial, 1022
leftNorm, 1022

leftPower, 1022
leftRecip, 1022
leftRegularRepresentation, 1022
leftTrace, 1022
leftTraceMatrix, 1022
leftUnit, 1022
leftUnitary, 1022
leftUnits, 1022
lieAdmissible?, 1022
lieAlgebra?, 1022
noncommutativeJordanAlgebra?, 1022
plenaryPower, 1022
powerAssociative?, 1022
rank, 1022
recip, 1022
represents, 1022
rightAlternative?, 1022
rightCharacteristicPolynomial, 1022
rightDiscriminant, 1022
rightMinimalPolynomial, 1022
rightNorm, 1022
rightPower, 1022
rightRecip, 1022
rightRegularRepresentation, 1022
rightTrace, 1022
rightTraceMatrix, 1022
rightUnit, 1022
rightUnitary, 1022
rightUnits, 1022
sample, 1022
someBasis, 1022
structuralConstants, 1022
subtractIfCan, 1022
unit, 1022
unitsKnown, 1022
zero?, 1022
find
A1AGG, 687
ALAGG, 824
BTAGG, 734
CLAGG, 289
DIAGG, 535
DIOPS, 387
ELAGG, 550
FLAGG, 558
FSAGG, 624
KDAGG, 632
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MDAGG, 572
MSETAGG, 675
NTSCAT, 749
OMSAGG, 765
PSETCAT, 468
PTCAT, 896
RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
VECTCAT, 811
findCoef
LOCPOWC, 1243
FINITE, 168
?=?, 169
?~=?, 169
coerce, 169
hash, 169
index, 169
latex, 169
lookup, 169
random, 169
size, 169
Finite, 168
FiniteAbelianMonoidRing, 1141
finiteAggregate
A1AGG, 687
AGG, 70
ALAGG, 824
ARR2CAT, 359
ATTREG, 11, 133
BRAGG, 375
BTAGG, 734
BTCAT, 528
CLAGG, 289
DIAGG, 535
DIOPS, 388

DIRPCAT, 1005
DQAGG, 542
FLAGG, 558
FSAGG, 624
HOAGG, 183
IXAGG, 320
KDAGG, 633
LNAGG, 407
LSAGG, 663
MATCAT, 437
NTSCAT, 750
OMSAGG, 766
PRQAGG, 482
PSETCAT, 468
PTCAT, 897
QUAGG, 488
RMATCAT, 903
RSETCAT, 703
SFRTCAT, 780
SKAGG, 502
SMATCAT, 1085
SNTSCAT, 915
SRAGG, 790
STRICAT, 923
TBAGG, 800
TSETCAT, 599
URAGG, 509
VECTCAT, 811
FiniteAlgebraicExtensionField, 1583
FiniteDivisorCategory, 616
FiniteFieldCategory, 1518
FiniteLinearAggregate, 555
FiniteRankAlgebra, 1329
FiniteRankNonAssociativeAlgebra, 1019
FiniteSetAggregate, 621
FINRALG, 1329
-?, 1331
?**?, 1331
?*?, 1331
?+?, 1331
?-?, 1331
?=?, 1331
?^?, 1331
?~=?, 1331
0, 1331
1, 1331
characteristic, 1331
characteristicPolynomial, 1331
charthRoot, 1331
coerce, 1331
coordinates, 1331
discriminant, 1331
hash, 1331
latex, 1331
leftUnitary, 1331
minimalPolynomial, 1331
norm, 1331
one?, 1331
rank, 1331
recip, 1331
regularRepresentation, 1331
represents, 1331
rightUnitary, 1331
sample, 1331
subtractIfCan, 1331
trace, 1331
traceMatrix, 1331
unitsKnown, 1331
zero?, 1331
first
A1AGG, 687
ALAGG, 824
BTAGG, 734
DIRPCAT, 1004
ELAGG, 550
FLAGG, 558
IXAGG, 320
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
NTSCAT, 749
PTCAT, 896
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
URAGG, 509
VECTCAT, 811

FLAGG, 555
?<?, 558
?<=?, 558
?>?, 558
?>=?, 558
?.?, 558
?=?, 558
?~=?, 558
#?, 558
any?, 558
coerce, 558
concat, 558
construct, 558
convert, 558
copy, 558
copyInto, 558
count, 558
delete, 558
elt, 558
empty, 558
empty?, 558
entries, 558
entry?, 558
eq?, 558
eval, 558
every?, 558
fill, 558
find, 558
finiteAggregate, 558
first, 558
hash, 558
index?, 558
indices, 558
insert, 558
latex, 558
less?, 558
map, 558
max, 558
maxIndex, 558
member?, 558
members, 558
merge, 558
min, 558
minIndex, 558
more?, 558
new, 558
parts, 558
position, 558
qelt, 558
qsetelt, 558
reduce, 558
remove, 558
removeDuplicates, 558
reverse, 558
sample, 558
select, 558
setelt, 558
shallowlyMutable, 558
size?, 558
sort, 558
sorted?, 558
swap, 558
FLALG, 1041
-?, 1043
?*?, 1043
?+?, 1043
?-?, 1043
?/?, 1043
?=?, 1043
?~=?, 1043
0, 1043
coef, 1043
coerce, 1043
construct, 1043
degree, 1043
eval, 1043
hash, 1043
JacobiIdentity, 1043
latex, 1043
leftUnitary, 1043
LiePoly, 1043
lquo, 1043
mirror, 1043
NullSquare, 1043
rightUnitary, 1043
rquo, 1043
sample, 1043
subtractIfCan, 1043
trunc, 1043
varList, 1043
zero?, 1043
flexible?

FINAALG, 1022
FRNAALG, 1118
FLINEXP, 960
-?, 962
?**?, 962
?*?, 962
?+?, 962
?-?, 962
?=?, 962
?^?, 962
?~=?, 962
0, 962
1, 962
characteristic, 962
coerce, 962
hash, 962
latex, 962
one?, 962
recip, 962
reducedSystem, 962
sample, 962
subtractIfCan, 962
unitsKnown, 962
zero?, 962
float
FPS, 1532
SEXCAT, 248
float?
SEXCAT, 248
FloatingPointSystem, 1530
floor
FPS, 1532
QFCAT, 1380
RNS, 1403
ULSCCAT, 1559
FMC, 141
coerce, 142
outputAsFortran, 142
FMCAT, 858
-?, 859
?*?, 859
?+?, 859
?-?, 859
?=?, 859
?~=?, 859
0, 859
coefficient, 859
coefficients, 859
coerce, 859
hash, 859
latex, 859
leadingCoefficient, 859
leadingMonomial, 859
leadingTerm, 859
leftUnitary, 860
listOfTerms, 859
map, 859
monom, 859
monomial?, 859
monomials, 859
numberOfMonomials, 859
reductum, 859
retract, 859
retractIfCan, 859
rightUnitary, 860
sample, 859
subtractIfCan, 859
zero?, 859
FMFUN, 145
coerce, 146
outputAsFortran, 146
retract, 146
retractIfCan, 146
FMTC, 1108
-?, 1110
?<?, 1110
?<=?, 1110
?>?, 1110
?>=?, 1110
?**?, 1110
?*?, 1110
?+?, 1110
?-?, 1110
?=?, 1110
?^?, 1110
?~=?, 1110
0, 1110
1, 1110
associates?, 1110
characteristic, 1110
coerce, 1110
commutative("*"), 1110

exquo, 1110
hash, 1110
latex, 1110
leftUnitary, 1110
max, 1110
min, 1110
noZeroDivisors, 1110
one?, 1110
recip, 1110
retract, 1110
retractIfCan, 1110
rightUnitary, 1110
sample, 1110
subtractIfCan, 1110
unit?, 1110
unitCanonical, 1110
unitNormal, 1110
unitsKnown, 1110
zero?, 1110
FNCAT, 172
?=?, 173
?~=?, 173
coerce, 173
directory, 173
exists?, 173
extension, 173
filename, 173
hash, 173
latex, 173
name, 173
new, 173
readable?, 173
writable?, 173
FORTCAT, 84
coerce, 85
outputAsFortran, 85
FORTFN, 136
coerce, 137
outputAsFortran, 137
retract, 137
retractIfCan, 137
FortranFunctionCategory, 136
FortranMachineTypeCategory, 1108
FortranMatrixCategory, 141
FortranMatrixFunctionCategory, 145
FortranProgramCategory, 84
FortranVectorCategory, 150
FortranVectorFunctionCategory, 154
foundPlaces
PLACESC, 341
FPATMAB, 91
?=?, 93
?~=?, 93
coerce, 93
hash, 93
latex, 93
patternMatch, 93
FPC, 1323
-?, 1325
?**?, 1325
?*?, 1325
?+?, 1325
?-?, 1325
?/?, 1325
?=?, 1325
?^?, 1325
?~=?, 1325
?quo?, 1325
?rem?, 1325
0, 1325
1, 1325
associates?, 1325
canonicalsClosed, 1325
canonicalUnitNormal, 1325
characteristic, 1325
charthRoot, 1325
coerce, 1325
commutative("*"), 1326
discreteLog, 1325
divide, 1325
euclideanSize, 1325
expressIdealMember, 1325
exquo, 1325
extendedEuclidean, 1325
factor, 1325
gcd, 1325
gcdPolynomial, 1325
hash, 1325
inv, 1325
latex, 1325
lcm, 1325
leftUnitary, 1326

multiEuclidean, 1325
noZeroDivisors, 1326
one?, 1325
order, 1325
prime?, 1325
primeFrobenius, 1325
principalIdeal, 1325
recip, 1325
rightUnitary, 1326
sample, 1325
sizeLess?, 1325
squareFree, 1325
squareFreePart, 1325
subtractIfCan, 1325
unit?, 1325
unitCanonical, 1325
unitNormal, 1325
unitsKnown, 1326
zero?, 1325
FPS, 1530
-?, 1532
?<?, 1532
?<=?, 1532
?>?, 1532
?>=?, 1532
?**?, 1532
?*?, 1532
?+?, 1532
?-?, 1532
?/?, 1532
?=?, 1532
?^?, 1532
?~=?, 1532
?quo?, 1532
?rem?, 1532
0, 1532
1, 1532
abs, 1532
approximate, 1532
associates?, 1532
base, 1532
bits, 1532
canonicalsClosed, 1533
canonicalUnitNormal, 1533
ceiling, 1532
characteristic, 1532
coerce, 1532
commutative("*"), 1533
convert, 1532
decreasePrecision, 1532
digits, 1532
divide, 1532
euclideanSize, 1532
exponent, 1532
expressIdealMember, 1532
exquo, 1532
extendedEuclidean, 1532
factor, 1532
float, 1532
floor, 1532
fractionPart, 1532
gcd, 1532
gcdPolynomial, 1532
hash, 1532
increasePrecision, 1532
inv, 1532
latex, 1532
lcm, 1532
leftUnitary, 1533
mantissa, 1532
max, 1532
min, 1532
multiEuclidean, 1532
negative?, 1532
norm, 1532
noZeroDivisors, 1533
nthRoot, 1532
one?, 1532
order, 1532
patternMatch, 1532
positive?, 1532
precision, 1532
prime?, 1532
principalIdeal, 1532
recip, 1532
retract, 1532
retractIfCan, 1532
rightUnitary, 1533
round, 1532
sample, 1532
sign, 1532
sizeLess?, 1532

sqrt, 1532
squareFree, 1532
squareFreePart, 1532
subtractIfCan, 1532
truncate, 1532
unit?, 1532
unitCanonical, 1532
unitNormal, 1532
unitsKnown, 1533
wholePart, 1532
zero?, 1532
fractionPart
FPS, 1532
QFCAT, 1380
RNS, 1403
ULSCCAT, 1559
FRAMALG, 1539
-?, 1541
?**?, 1541
?*?, 1541
?+?, 1541
?-?, 1541
?=?, 1541
?^?, 1541
?~=?, 1541
0, 1541
1, 1541
basis, 1541
characteristic, 1541
characteristicPolynomial, 1541
charthRoot, 1541
coerce, 1541
convert, 1541
coordinates, 1541
discriminant, 1541
hash, 1541
latex, 1541
leftUnitary, 1541
minimalPolynomial, 1541
norm, 1541
one?, 1541
rank, 1541
recip, 1541
regularRepresentation, 1541
represents, 1541
rightUnitary, 1541
sample, 1541
subtractIfCan, 1541
trace, 1541
traceMatrix, 1541
unitsKnown, 1541
zero?, 1541
FramedAlgebra, 1539
FramedNonAssociativeAlgebra, 1115
FreeAbelianMonoidCategory, 564
FreeLieAlgebra, 1041
FreeModuleCat, 858
freeOf?
ACFS, 1500
ES, 302
FS, 1341
FRETRCT, 87
coerce, 89
retract, 89
retractIfCan, 89
FRNAALG, 1115
-?, 1118
?**?, 1118
?*?, 1118
?+?, 1118
?-?, 1118
?.?, 1118
?=?, 1118
?~=?, 1118
0, 1118
alternative?, 1118
antiAssociative?, 1118
antiCommutative?, 1118
antiCommutator, 1118
apply, 1118
associative?, 1118
associator, 1118
associatorDependence, 1118
basis, 1118
coerce, 1118
commutative?, 1118
commutator, 1118
conditionsForIdempotents, 1118
convert, 1118
coordinates, 1118
flexible?, 1118
hash, 1118

jacobiIdentity?, 1118
jordanAdmissible?, 1118
jordanAlgebra?, 1118
latex, 1118
leftAlternative?, 1118
leftCharacteristicPolynomial, 1118
leftDiscriminant, 1118
leftMinimalPolynomial, 1118
leftNorm, 1118
leftPower, 1118
leftRankPolynomial, 1118
leftRecip, 1118
leftRegularRepresentation, 1118
leftTrace, 1118
leftTraceMatrix, 1118
leftUnit, 1118
leftUnitary, 1118
leftUnits, 1118
lieAdmissible?, 1118
lieAlgebra?, 1118
noncommutativeJordanAlgebra?, 1118
plenaryPower, 1118
powerAssociative?, 1118
rank, 1118
recip, 1118
represents, 1118
rightAlternative?, 1118
rightCharacteristicPolynomial, 1118
rightDiscriminant, 1118
rightMinimalPolynomial, 1118
rightNorm, 1118
rightPower, 1118
rightRankPolynomial, 1118
rightRecip, 1118
rightRegularRepresentation, 1118
rightTrace, 1118
rightTraceMatrix, 1118
rightUnit, 1118
rightUnitary, 1118
rightUnits, 1118
sample, 1118
someBasis, 1118
structuralConstants, 1118
subtractIfCan, 1118
unit, 1118
unitsKnown, 1118
zero?, 1118
Frobenius
FAXF, 1587
PACEXTC, 1668
PACRATC, 1613
XF, 1513
front
DQAGG, 542
QUAGG, 488
frsenelC
LFCAT, 193
frsenelS
LFCAT, 193
frst
LZSTAGG, 640
FS, 1336
-?, 1341
?<?, 1341
?<=?, 1341
?>?, 1341
?>=?, 1341
?**?, 1341
?*?, 1341
?+?, 1341
?-?, 1341
?/?, 1341
?=?, 1341
?^?, 1341
?~=?, 1341
?quo?, 1341
?rem?, 1341
0, 1341
1, 1341
applyQuote, 1341
associates?, 1341
belong?, 1341
box, 1341
canonicalsClosed, 1341
canonicalUnitNormal, 1341
characteristic, 1341
charthRoot, 1341
coerce, 1341
commutative("*"), 1341
commutator, 1341
conjugate, 1341
convert, 1341

D, 1341
definingPolynomial, 1341
denom, 1341
denominator, 1341
differentiate, 1341
distribute, 1341
divide, 1341
elt, 1341
euclideanSize, 1341
eval, 1341
even?, 1341
expressIdealMember, 1341
exquo, 1341
extendedEuclidean, 1341
factor, 1341
freeOf?, 1341
gcd, 1341
gcdPolynomial, 1341
ground, 1341
ground?, 1341
hash, 1341
height, 1341
inv, 1341
is?, 1341
isExpt, 1341
isMult, 1341
isPlus, 1341
isPower, 1341
isTimes, 1341
kernel, 1341
kernels, 1341
latex, 1341
lcm, 1341
leftUnitary, 1341
mainKernel, 1341
map, 1341
max, 1341
min, 1341
minPoly, 1341
multiEuclidean, 1341
noZeroDivisors, 1341
numer, 1341
numerator, 1341
odd?, 1341
one?, 1341
operator, 1341
operators, 1341
paren, 1341
patternMatch, 1341
prime?, 1341
principalIdeal, 1341
recip, 1341
reducedSystem, 1341
retract, 1341
retractIfCan, 1341
rightUnitary, 1342
sample, 1341
sizeLess?, 1341
squareFree, 1341
squareFreePart, 1341
subst, 1341
subtractIfCan, 1341
tower, 1341
unit?, 1341
unitCanonical, 1341
unitNormal, 1341
unitsKnown, 1341
univariate, 1341
variables, 1341
zero?, 1341
FSAGG, 621
?<?, 624
?=?, 624
?~=?, 624
#?, 624
any?, 624
bag, 624
brace, 624
cardinality, 624
coerce, 624
complement, 624
construct, 624
convert, 624
copy, 624
count, 624
dictionary, 624
difference, 624
empty, 624
empty?, 624
eq?, 624
eval, 624
every?, 624

extract, 624
find, 624
finiteAggregate, 624
hash, 624
index, 624
insert, 624
inspect, 624
intersect, 624
latex, 624
less?, 624
lookup, 624
map, 624
max, 624
member?, 624
members, 624
min, 624
more?, 624
partiallyOrderedSet, 624
parts, 624
random, 624
reduce, 624
remove, 624
removeDuplicates, 624
sample, 624
select, 624
set, 624
shallowlyMutable, 624
size, 624
size?, 624
subset?, 624
symmetricDifference, 624
union, 624
universe, 624
fullOutput
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
FullyEvalableOver, 159
FullyLinearlyExplicitRingOver, 960
FullyPatternMatchable, 91
FullyRetractableTo, 87
FunctionFieldCategory, 1642
FunctionSpace, 1336
FVC, 150
FVFUN, 154
coerce, 155
outputAsFortran, 155
retract, 155
retractIfCan, 155

Gamma
SPFCAT, 60
gcd
ACF, 1298
ACFS, 1500
COMPCAT, 1625
DPOLCAT, 1310
EUCDOM, 1187
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FPC, 1325
FPS, 1532
FS, 1341
GCDDOM, 1131
INS, 1231
INTCAT, 1153
LOCPOWC, 1243
MONOGEN, 1602
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PFECAT, 1204
PID, 1168
POLYCAT, 1260
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
RPOLCAT, 1414
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPXSCAT, 1464
UPXSCCA, 1574
XF, 1513
GCDDOM, 1129
-?, 1131

?**?, 1131
?*?, 1131
?+?, 1131
?-?, 1131
?=?, 1131
?^?, 1131
?~=?, 1131
0, 1131
1, 1131
associates?, 1131
characteristic, 1131
coerce, 1131
commutative("*"), 1131
exquo, 1131
gcd, 1131
gcdPolynomial, 1131
hash, 1131
latex, 1131
lcm, 1131
leftUnitary, 1131
noZeroDivisors, 1131
one?, 1131
recip, 1131
rightUnitary, 1131
sample, 1131
subtractIfCan, 1131
unit?, 1131
unitCanonical, 1131
unitNormal, 1131
unitsKnown, 1131
zero?, 1131
GcdDomain, 1129
gcdPolynomial
ACF, 1298
ACFS, 1500
COMPCAT, 1625
DPOLCAT, 1310
EUCDOM, 1187
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FPC, 1325
FPS, 1532
FS, 1341
GCDDOM, 1131
INS, 1231
INTCAT, 1153
LOCPOWC, 1243
MONOGEN, 1602
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PFECAT, 1204
PID, 1168
POLYCAT, 1260
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
RPOLCAT, 1414
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPXSCAT, 1464
UPXSCCA, 1574
XF, 1513
generator
COMPCAT, 1625
FAXF, 1587
FDIVCAT, 618
FFCAT, 1647
MONOGEN, 1602
genus
FFCAT, 1647
GradedAlgebra, 313
GradedModule, 176
GRALG, 313
-?, 314
?*?, 314
?+?, 314
?-?, 314
?=?, 314
?~=?, 314
0, 314
1, 314
coerce, 314
degree, 314
hash, 314
latex, 314

product, 314
retract, 314
retractIfCan, 314
GRMOD, 176
-?, 177
?*?, 177
?+?, 177
?-?, 177
?=?, 177
?~=?, 177
0, 177
coerce, 177
degree, 177
hash, 177
latex, 177
ground
ACFS, 1500
DPOLCAT, 1310
FAMR, 1143
FS, 1341
POLYCAT, 1260
RPOLCAT, 1414
UPOLYC, 1477
ground?
ACFS, 1500
DPOLCAT, 1310
FAMR, 1143
FS, 1341
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
POLYCAT, 1260
RPOLCAT, 1414
UPOLYC, 1477
GROUP, 398
?**?, 400
?*?, 400
?/?, 400
?=?, 400
?^?, 400
?~=?, 400
1, 400
coerce, 400
commutator, 400
conjugate, 400
hash, 400
inv, 400
latex, 400
one?, 400
recip, 400
sample, 400
unitsKnown, 400
Group, 398

halfExtendedSubResultantGcd1
RPOLCAT, 1414
halfExtendedSubResultantGcd2
RPOLCAT, 1414
hash
A1AGG, 687
ABELGRP, 522
ABELMON, 269
ABELSG, 124
ACF, 1298
ACFS, 1500
AFSPCAT, 273
ALAGG, 824
ALGEBRA, 949
AMR, 1103
ARR2CAT, 359
BGAGG, 279
BLMETCT, 128
BMODULE, 728
BRAGG, 374
BTAGG, 734
BTCAT, 528
CABMON, 381
CACHESET, 284
CHARNZ, 835
CHARZ, 840
CLAGG, 289
COMPCAT, 1625
COMRING, 845
DIAGG, 535
DIFEXT, 955
DIFRING, 850
DIOPS, 387
DIRPCAT, 1004
DIVCAT, 1181
DIVRING, 1014
DLAGG, 394

DPOLCAT, 1310
DQAGG, 542
DSTRCAT, 133
DVARCAT, 296
ELAGG, 550
ENTIRER, 855
ES, 302
EUCDOM, 1187
FAMONC, 565
FAMR, 1143
FAXF, 1587
FDIVCAT, 618
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FILECAT, 165
FINAALG, 1022
FINITE, 169
FINRALG, 1331
FLAGG, 558
FLALG, 1043
FLINEXP, 962
FMCAT, 859
FMTC, 1110
FNCAT, 173
FPATMAB, 93
FPC, 1325
FPS, 1532
FRAMALG, 1541
FRNAALG, 1118
FS, 1341
FSAGG, 624
GCDDOM, 1131
GRALG, 314
GRMOD, 177
GROUP, 400
HOAGG, 183
IDPC, 189
INFCLCT, 1366
INS, 1231
INTCAT, 1153
INTDOM, 1049
IXAGG, 320
KDAGG, 632
LALG, 865
LIECAT, 968
LINEXP, 871
LMODULE, 657
LNAGG, 407
LOCPOWC, 1243
LODOCAT, 973
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
MDAGG, 572
MLO, 1055
MODULE, 876
MONAD, 198
MONADWU, 327
MONOGEN, 1602
MONOID, 332
MSETAGG, 675
MTSCAT, 1196
NAALG, 982
NARNG, 681
NASRING, 742
NTSCAT, 749
NUMINT, 203
OAGROUP, 756
OAMON, 577
OAMONS, 760
OASGP, 458
OC, 1062
OCAMON, 697
ODECAT, 213
OINTDOM, 1137
OMSAGG, 765
OPTCAT, 208
ORDFIN, 337
ORDMON, 462
ORDRING, 881
ORDSET, 218
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PATMAB, 227
PDECAT, 222
PDRING, 887
PERMCAT, 582

PFECAT, 1204
PID, 1168
PLACESC, 341
POLYCAT, 1260
PRQAGG, 482
PRSPCAT, 346
PSCAT, 1161
PSETCAT, 468
PTCAT, 896
QFCAT, 1380
QUAGG, 488
QUATCAT, 1073
RCAGG, 353
RCFIELD, 1392
RING, 772
RMATCAT, 903
RMODULE, 716
RNG, 720
RNS, 1403
RPOLCAT, 1414
RRCC, 231
RSETCAT, 703
SETAGG, 494
SETCAT, 115
SETCATD, 244
SEXCAT, 248
SFRTCAT, 779
SGROUP, 240
SKAGG, 502
SMATCAT, 1085
SNTSCAT, 914
SPACEC, 258
SRAGG, 789
STAGG, 588
STEP, 253
STRICAT, 923
TBAGG, 800
TSETCAT, 598
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
URAGG, 509
UTSCAT, 1281
VECTCAT, 811
VSPACE, 989
XALG, 942
XF, 1513
XFALG, 994
XPOLYC, 1096
head
DLAGG, 394
RPOLCAT, 1414
headReduce
NTSCAT, 749
RPOLCAT, 1414
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
headReduced?
NTSCAT, 749
RPOLCAT, 1414
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
headRemainder
NTSCAT, 749
PSETCAT, 468
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
height
ACFS, 1500
DQAGG, 542
ES, 302
FS, 1341
hi
SEGCAT, 110
SEGXCAT, 236
high
SEGCAT, 110
SEGXCAT, 236
highCommonTerms
DIVCAT, 1181
FAMONC, 565
HOAGG, 181

?=?, 183
?~=?, 183
#?, 183
any?, 183
coerce, 183
copy, 183
count, 183
empty, 183
empty?, 183
eq?, 183
eval, 183
every?, 183
finiteAggregate, 183
hash, 183
latex, 183
less?, 183
map, 183
member?, 183
members, 183
more?, 183
parts, 183
sample, 183
shallowlyMutable, 183
size?, 183
HomogeneousAggregate, 181
homogenize
PRSPCAT, 346
horizConcat
MATCAT, 437
HYPCAT, 32
cosh, 33
coth, 33
csch, 33
sech, 33
sinh, 33
tanh, 33
HyperbolicFunctionCategory, 32
hyperelliptic
FFCAT, 1647

ideal
FDIVCAT, 618
IDPC, 188
?=?, 189
?~=?, 189
coerce, 189
hash, 189
latex, 189
leadingCoefficient, 189
leadingSupport, 189
map, 189
monomial, 189
reductum, 189
IEVALAB, 35
eval, 37
imag
COMPCAT, 1625
imagE
OC, 1062
images
A1AGG, 684
ABELGRP, 521
ABELMON, 267
ABELSG, 123
ACF, 1293
ACFS, 1495
AFSPCAT, 272
AGG, 69
AHYP, 5
ALAGG, 819
ALGEBRA, 947
AMR, 1101
ARR2CAT, 357
ATRIG, 7
ATTREG, 10
BASTYPE, 14
BGAGG, 277
BLMETCT, 127
BMODULE, 727
BRAGG, 373
BTAGG, 731
BTCAT, 527
CABMON, 380
CACHSET, 283
CATEGORY, 3
CFCAT, 20
CHARNZ, 833
CHARZ, 838
CLAGG, 287
COMBOPC, 73
COMPCAT, 1619
COMRING, 843

DIAGG, 533
DIFEXT, 953
DIFRING, 848
DIOPS, 385
DIRPCAT, 1001
DIVCAT, 1179
DIVRING, 1013
DLAGG, 392
DPOLCAT, 1306
DQAGG, 540
DSTRCAT, 131
DVARCAT, 294
ELAGG, 547
ELEMFUN, 27
ELTAB, 29
ELTAGG, 76
ENTIRER, 853
ES, 300
EUCDOM, 1185
EVALAB, 80
FAMONC, 564
FAMR, 1141
FAXF, 1583
FDIVCAT, 617
FEVALAB, 159
FFCAT, 1642
FFIELDC, 1518
FIELD, 1221
FILECAT, 163
FINAALG, 1019
FINITE, 168
FINRALG, 1329
FLAGG, 555
FLALG, 1042
FLINEXP, 960
FMC, 141
FMCAT, 858
FMFUN, 145
FMTC, 1108
FNCAT, 172
FORTCAT, 84
FORTFN, 136
FPATMAB, 92
FPC, 1323
FPS, 1530
FRAMALG, 1539
FRETRCT, 87
FRNAALG, 1115
FS, 1336
FSAGG, 621
FVC, 150
FVFUN, 154
GCDDOM, 1129
GRALG, 313
GRMOD, 176
GROUP, 398
HOAGG, 181
HYPCAT, 32
IDPC, 188
IEVALAB, 35
INFCLCT, 1364
INS, 1228
INTCAT, 1150
INTDOM, 1047
IXAGG, 318
KDAGG, 630
KOERCE, 17
KONVERT, 23
LALG, 864
LFCAT, 192
LIECAT, 966
LINEXP, 869
LMODULE, 655
LNAGG, 404
LOCPOWC, 1239
LODOCAT, 971
LOGIC, 95
LSAGG, 659
LZSTAGG, 637
MATCAT, 412
MDAGG, 570
MLO, 1053
MODULE, 874
MONAD, 197
MONADWU, 325
MONOGEN, 1598
MONOID, 331
MSETAGG, 673
MTSCAT, 1193
NAALG, 980
NARNG, 679
NASRING, 740

NTSCAT, 745
NUMINT, 202
OAGROUP, 755
OAMON, 576
OAMONS, 759
OASGP, 456
OC, 1059
OCAMON, 696
ODECAT, 212
OINTDOM, 1135
OM, 39
OMSAGG, 763
OPTCAT, 207
ORDFIN, 336
ORDMON, 461
ORDRING, 879
ORDSET, 216
OREPCAT, 929
PACEXTC, 1665
PACFFC, 1546
PACPERC, 1369
PACRATC, 1611
PADICCT, 1249
PATAB, 47
PATMAB, 226
PDECAT, 221
PDRING, 885
PERMCAT, 580
PFECAT, 1202
PID, 1166
PLACESC, 340
POLYCAT, 1255
PPCURVE, 99
PRIMCAT, 50
PRQAGG, 480
PRSPCAT, 345
PSCAT, 1159
PSCURVE, 102
PSETCAT, 465
PTCAT, 894
PTRANFN, 42
QFCAT, 1376
QUAGG, 486
QUATCAT, 1071
RADCAT, 52
RCAGG, 351
RCFIELD, 1390
REAL, 106
RETRACT, 55
RING, 770
RMATCAT, 901
RMODULE, 715
RNG, 719
RNS, 1400
RPOLCAT, 1408
RRCC, 230
RSETCAT, 700
SEGCAT, 109
SEGXCAT, 235
SETAGG, 492
SETCAT, 113
SETCATD, 243
SEXCAT, 246
SFRTCAT, 775
SGROUP, 239
SKAGG, 500
SMATCAT, 1082
SNTSCAT, 910
SPACEC, 256
SPFCAT, 59
SRAGG, 786
STAGG, 586
STEP, 252
STRICAT, 920
TBAGG, 797
TRANFUN, 117
TRIGCAT, 63
TSETCAT, 595
TYPE, 66
UFD, 1172
ULSCAT, 1448
ULSCCAT, 1553
UPOLYC, 1471
UPSCAT, 1210
UPXSCAT, 1460
UPXSCCA, 1570
URAGG, 506
UTSCAT, 1277
VECTCAT, 808
VSPACE, 988
XALG, 940
XF, 1510

XFALG, 992
XPOLYC, 1094
imagI
OC, 1062
QUATCAT, 1073
imagi
OC, 1062
imaginary
COMPCAT, 1625
imagJ
OC, 1062
QUATCAT, 1073
imagj
OC, 1062
imagK
OC, 1062
QUATCAT, 1073
imagk
OC, 1062
inc
INS, 1231
incr
DIVCAT, 1181
SEGCAT, 110
SEGXCAT, 236
increasePrecision
FPS, 1532
index
COMPCAT, 1625
DIRPCAT, 1004
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FINITE, 169
FSAGG, 624
MONOGEN, 1602
OC, 1062
ORDFIN, 337
PACFFC, 1548
index?
A1AGG, 687
ALAGG, 824
BTAGG, 734
DIRPCAT, 1004
ELAGG, 550
FLAGG, 558
IXAGG, 320
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
PTCAT, 896
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
VECTCAT, 811
IndexedAggregate, 318
IndexedDirectProductCategory, 188
indices
A1AGG, 687
ALAGG, 824
BTAGG, 734
DIRPCAT, 1004
ELAGG, 550
FLAGG, 558
IXAGG, 320
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
PTCAT, 896
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
VECTCAT, 811
inf
INTCAT, 1153
INFCLCT, 1364
?=?, 1366
?~=?, 1366
actualExtensionV, 1366
chartV, 1366
coerce, 1366
create, 1366
curveV, 1366
degree, 1366
excpDivV, 1366
hash, 1366
latex, 1366
localParamV, 1366
localPointV, 1366
multV, 1366

pointV, 1366
setchart, 1366
setcurve, 1366
setexcpDiv, 1366
setlocalParam, 1366
setlocalPoint, 1366
setmult, 1366
setpoint, 1366
setsubmult, 1366
setsymbName, 1366
subMultV, 1366
symbNameV, 1366
infClsPt?
BLMETCT, 128
InfinitlyClosePointCategory, 1364
infRittWu?
NTSCAT, 749
RPOLCAT, 1414
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
inGroundField?
FAXF, 1587
PACEXTC, 1668
PACRATC, 1613
XF, 1513
init
COMPCAT, 1625
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
INS, 1231
MONOGEN, 1602
PACFFC, 1548
QFCAT, 1380
RPOLCAT, 1414
STEP, 253
ULSCCAT, 1559
UPOLYC, 1477
initial
DPOLCAT, 1310
initiallyReduce
NTSCAT, 749
RPOLCAT, 1414
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
initiallyReduced?
NTSCAT, 749
RPOLCAT, 1414
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
initials
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
InnerEvalable, 35
INS, 1228
-?, 1231
?<?, 1231
?<=?, 1231
?>?, 1231
?>=?, 1231
?**?, 1231
?*?, 1231
?+?, 1231
?-?, 1231
?=?, 1231
?^?, 1231
?~=?, 1231
?quo?, 1231
?rem?, 1231
0, 1231
1, 1231
abs, 1231
addmod, 1231
associates?, 1231
base, 1231
binomial, 1231
bit?, 1231
canonicalUnitNormal, 1231
characteristic, 1231
coerce, 1231
commutative("*"), 1231
convert, 1231
copy, 1231

D, 1231
dec, 1231
differentiate, 1231
divide, 1231
euclideanSize, 1231
even?, 1231
expressIdealMember, 1231
exquo, 1231
extendedEuclidean, 1231
factor, 1231
factorial, 1231
gcd, 1231
gcdPolynomial, 1231
hash, 1231
inc, 1231
init, 1231
invmod, 1231
latex, 1231
lcm, 1231
leftUnitary, 1231
length, 1231
mask, 1231
max, 1231
min, 1231
mulmod, 1231
multiEuclidean, 1231
multiplicativeValuation, 1231
negative?, 1231
nextItem, 1231
noZeroDivisors, 1231
odd?, 1231
one?, 1231
patternMatch, 1231
permutation, 1231
positive?, 1231
positiveRemainder, 1231
powmod, 1231
prime?, 1231
principalIdeal, 1231
random, 1231
rational, 1231
rational?, 1231
rationalIfCan, 1231
recip, 1231
reducedSystem, 1231
retract, 1231
retractIfCan, 1231
rightUnitary, 1231
sample, 1231
shift, 1231
sign, 1231
sizeLess?, 1231
squareFree, 1231
squareFreePart, 1231
submod, 1231
subtractIfCan, 1231
symmetricRemainder, 1231
unit?, 1231
unitCanonical, 1231
unitNormal, 1231
unitsKnown, 1231
zero?, 1231
insert
A1AGG, 687
ALAGG, 824
BTAGG, 734
ELAGG, 550
FLAGG, 558
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
PTCAT, 896
SRAGG, 789
STAGG, 588
STRICAT, 923
VECTCAT, 811
inspect
ALAGG, 824
BGAGG, 279
DIAGG, 535
DIOPS, 387
DQAGG, 542
FSAGG, 624
KDAGG, 632
MDAGG, 572
MSETAGG, 675
OMSAGG, 765
PRQAGG, 482
QUAGG, 488
SKAGG, 502
TBAGG, 800
INTCAT, 1150

-?, 1153
?<?, 1153
?<=?, 1153
?>?, 1153
?>=?, 1153
?**?, 1153
?*?, 1153
?+?, 1153
?-?, 1153
?=?, 1153
?^?, 1153
?~=?, 1153
0, 1153
1, 1153
acos, 1153
acosh, 1153
acot, 1153
acoth, 1153
acsc, 1153
acsch, 1153
approximate, 1153
asec, 1153
asech, 1153
asin, 1153
asinh, 1153
associates?, 1153
atan, 1153
atanh, 1153
characteristic, 1153
coerce, 1153
commutative("*"), 1153
contains?, 1153
cos, 1153
cosh, 1153
cot, 1153
coth, 1153
csc, 1153
csch, 1153
exp, 1153
exquo, 1153
gcd, 1153
gcdPolynomial, 1153
hash, 1153
inf, 1153
interval, 1153
latex, 1153
lcm, 1153
leftUnitary, 1153
log, 1153
max, 1153
min, 1153
negative?, 1153
noZeroDivisors, 1153
nthRoot, 1153
one?, 1153
pi, 1153
positive?, 1153
qinterval, 1153
recip, 1153
retract, 1153
retractIfCan, 1153
rightUnitary, 1153
sample, 1153
sec, 1153
sech, 1153
sin, 1153
sinh, 1153
sqrt, 1153
subtractIfCan, 1153
sup, 1153
tan, 1153
tanh, 1153
unit?, 1153
unitCanonical, 1153
unitNormal, 1153
unitsKnown, 1153
width, 1153
zero?, 1153
INTDOM, 1047
-?, 1049
?**?, 1049
?*?, 1049
?+?, 1049
?-?, 1049
?=?, 1049
?^?, 1049
?~=?, 1049
0, 1049
1, 1049
associates?, 1049
canonicalUnitNormal, 1049
characteristic, 1049

coerce, 1049
commutative("*"), 1049
exquo, 1049
hash, 1049
latex, 1049
leftUnitary, 1049
noZeroDivisors, 1049
one?, 1049
recip, 1049
rightUnitary, 1049
sample, 1049
subtractIfCan, 1049
unit?, 1049
unitCanonical, 1049
unitNormal, 1049
unitsKnown, 1049
zero?, 1049
integer
SEXCAT, 248
integer?
SEXCAT, 248
IntegerNumberSystem, 1228
integral
LFCAT, 193
PRIMCAT, 51
integral?
FFCAT, 1647
integralAtInfinity?
FFCAT, 1647
integralBasis
FFCAT, 1647
integralBasisAtInfinity
FFCAT, 1647
integralCoordinates
FFCAT, 1647
integralDerivationMatrix
FFCAT, 1647
IntegralDomain, 1047
integralMatrix
FFCAT, 1647
integralMatrixAtInfinity
FFCAT, 1647
integralRepresents
FFCAT, 1647
integrate
MTSCAT, 1196
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
internalAugment
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
intersect
FSAGG, 624
MSETAGG, 675
NTSCAT, 749
OMSAGG, 765
RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SNTSCAT, 914
interval
INTCAT, 1153
IntervalCategory, 1150
inv
ACF, 1298
ACFS, 1500
COMPCAT, 1625
DIVRING, 1014
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FPC, 1325
FPS, 1532
FS, 1341
GROUP, 400
LOCPOWC, 1243
MONOGEN, 1602
OC, 1062
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PERMCAT, 582
QFCAT, 1380
QUATCAT, 1073

RCFIELD, 1392
RNS, 1403
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
XF, 1513
inverse
MATCAT, 437
SMATCAT, 1085
inverseIntegralMatrix
FFCAT, 1647
inverseIntegralMatrixAtInfinity
FFCAT, 1647
invertible?
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
invertibleElseSplit?
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
invertibleSet
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
invmod
INS, 1231
iomode
FILECAT, 165
is?
ACFS, 1500
ES, 302
FS, 1341
isExpt
ACFS, 1500
DPOLCAT, 1310
FS, 1341
POLYCAT, 1260
RPOLCAT, 1414
UPOLYC, 1477
isMult
ACFS, 1500
FS, 1341
isobaric?
DPOLCAT, 1310
isPlus
ACFS, 1500
DPOLCAT, 1310
FS, 1341
POLYCAT, 1260
RPOLCAT, 1414
UPOLYC, 1477
isPower
ACFS, 1500
FS, 1341
isTimes
ACFS, 1500
DPOLCAT, 1310
FS, 1341
POLYCAT, 1260
RPOLCAT, 1414
UPOLYC, 1477
iteratedInitials
RPOLCAT, 1414
IXAGG, 318
?.?, 320
?=?, 320
?~=?, 320
#?, 320
any?, 320
coerce, 320
copy, 320
count, 320
elt, 320
empty, 320
empty?, 320
entries, 320
entry?, 320
eq?, 320
eval, 320
every?, 320
fill, 320
finiteAggregate, 320
first, 320
hash, 320
index?, 320
indices, 320
latex, 320

less?, 320
map, 320
maxIndex, 320
member?, 320
members, 320
minIndex, 320
more?, 320
parts, 320
qelt, 320
qsetelt, 320
sample, 320
setelt, 320
shallowlyMutable, 320
size?, 320
swap, 320

JacobiIdentity
ATTREG, 11
FLALG, 1043
LIECAT, 968
jacobiIdentity?
FINAALG, 1022
FRNAALG, 1118
jordanAdmissible?
FINAALG, 1022
FRNAALG, 1118
jordanAlgebra?
FINAALG, 1022
FRNAALG, 1118

karatsubaDivide
UPOLYC, 1477
KDAGG, 630
?=?, 632
?~=?, 632
#?, 632
any?, 632
bag, 632
coerce, 632
construct, 632
convert, 632
copy, 632
count, 632
dictionary, 632
empty, 632
empty?, 632
eq?, 632
eval, 632
every?, 632
extract, 632
find, 632
finiteAggregate, 633
hash, 632
insert, 632
inspect, 632
key?, 632
keys, 632
latex, 632
less?, 632
map, 632
member?, 632
members, 632
more?, 632
parts, 632
reduce, 632
remove, 632
removeDuplicates, 632
sample, 632
search, 632
select, 632
shallowlyMutable, 632
size?, 632
kernel
ACFS, 1500
ES, 302
FS, 1341
kernels
ACFS, 1500
ES, 302
FS, 1341
key?
ALAGG, 824
KDAGG, 632
TBAGG, 800
KeyedDictionary, 630
keys
ALAGG, 824
KDAGG, 632
TBAGG, 800
KOERCE, 17
coerce, 18
KONVERT, 23

convert, 24

LALG, 864
-?, 865
?**?, 865
?*?, 865
?+?, 865
?-?, 865
?=?, 865
?^?, 865
?~=?, 865
0, 865
1, 865
characteristic, 865
coerce, 865
hash, 865
latex, 865
one?, 865
recip, 865
sample, 865
subtractIfCan, 865
unitsKnown, 866
zero?, 865

last
ALAGG, 824
DLAGG, 394
LSAGG, 663
LZSTAGG, 640
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
STAGG, 588
TSETCAT, 598
URAGG, 509

lastNonNul
PRSPCAT, 346

lastNonNull
PRSPCAT, 346

lastSubResultant
NTSCAT, 749
RPOLCAT, 1414
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914

lastSubResultantElseSplit
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914

latex
A1AGG, 687
ABELGRP, 522
ABELMON, 269
ABELSG, 124
ACF, 1298
ACFS, 1500
AFSPCAT, 273
ALAGG, 824
ALGEBRA, 949
AMR, 1103
ARR2CAT, 359
BGAGG, 279
BLMETCT, 128
BMODULE, 728
BRAGG, 374
BTAGG, 734
BTCAT, 528
CABMON, 381
CACHESET, 284
CHARNZ, 835
CHARZ, 840
CLAGG, 289
COMPCAT, 1625
COMRING, 845
DIAGG, 535
DIFEXT, 955
DIFRING, 850
DIOPS, 387
DIRPCAT, 1004
DIVCAT, 1181
DIVRING, 1014
DLAGG, 394
DPOLCAT, 1310
DQAGG, 542
DSTRCAT, 133
DVARCAT, 296
ELAGG, 550
ENTIRER, 855
ES, 302
EUCDOM, 1187
FAMONC, 565

FAMR, 1143
FAXF, 1587
FDIVCAT, 618
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FILECAT, 165
FINAALG, 1022
FINITE, 169
FINRALG, 1331
FLAGG, 558
FLALG, 1043
FLINEXP, 962
FMCAT, 859
FMTC, 1110
FNCAT, 173
FPATMAB, 93
FPC, 1325
FPS, 1532
FRAMALG, 1541
FRNAALG, 1118
FS, 1341
FSAGG, 624
GCDDOM, 1131
GRALG, 314
GRMOD, 177
GROUP, 400
HOAGG, 183
IDPC, 189
INFCLCT, 1366
INS, 1231
INTCAT, 1153
INTDOM, 1049
IXAGG, 320
KDAGG, 632
LALG, 865
LIECAT, 968
LINEXP, 871
LMODULE, 657
LNAGG, 407
LOCPOWC, 1243
LODOCAT, 973
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
MDAGG, 572
MLO, 1055
MODULE, 876
MONAD, 198
MONADWU, 327
MONOGEN, 1602
MONOID, 332
MSETAGG, 675
MTSCAT, 1196
NAALG, 982
NARNG, 681
NASRING, 742
NTSCAT, 749
NUMINT, 203
OAGROUP, 756
OAMON, 577
OAMONS, 760
OASGP, 458
OC, 1062
OCAMON, 697
ODECAT, 213
OINTDOM, 1137
OMSAGG, 765
OPTCAT, 208
ORDFIN, 337
ORDMON, 462
ORDRING, 881
ORDSET, 218
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PATMAB, 227
PDECAT, 222
PDRING, 887
PERMCAT, 582
PFECAT, 1204
PID, 1168
PLACESC, 341
POLYCAT, 1260
PRQAGG, 482
PRSPCAT, 346
PSCAT, 1161
PSETCAT, 468
PTCAT, 896

QFCAT, 1380
QUAGG, 488
QUATCAT, 1073
RCAGG, 353
RCFIELD, 1392
RING, 772
RMATCAT, 903
RMODULE, 716
RNG, 720
RNS, 1403
RPOLCAT, 1414
RRCC, 231
RSETCAT, 703
SETAGG, 494
SETCAT, 115
SETCATD, 244
SEXCAT, 248
SFRTCAT, 779
SGROUP, 240
SKAGG, 502
SMATCAT, 1085
SNTSCAT, 914
SPACEC, 258
SRAGG, 789
STAGG, 588
STEP, 253
STRICAT, 923
TBAGG, 800
TSETCAT, 598
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
URAGG, 509
UTSCAT, 1281
VECTCAT, 811
VSPACE, 989
XALG, 942
XF, 1513
XFALG, 994
XPOLYC, 1096
laurent
ULSCCAT, 1559
UPXSCCA, 1574
laurentIfCan
UPXSCCA, 1574
laurentRep
UPXSCCA, 1574
LazardQuotient
RPOLCAT, 1414
LazardQuotient2
RPOLCAT, 1414
lazy?
LZSTAGG, 640
lazyEvaluate
LZSTAGG, 640
lazyPquo
RPOLCAT, 1414
lazyPrem
RPOLCAT, 1414
lazyPremWithDefault
RPOLCAT, 1414
lazyPseudoDivide
RPOLCAT, 1414
lazyResidueClass
RPOLCAT, 1414
LazyStreamAggregate, 637
lcm
ACF, 1298
ACFS, 1500
COMPCAT, 1625
DPOLCAT, 1310
EUCDOM, 1187
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FPC, 1325
FPS, 1532
FS, 1341
GCDDOM, 1131
INS, 1231
INTCAT, 1153
LOCPOWC, 1243
MONOGEN, 1602
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613

PADICCT, 1251
PFECAT, 1204
PID, 1168
POLYCAT, 1260
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
RPOLCAT, 1414
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPXSCAT, 1464
UPXSCCA, 1574
XF, 1513
leader
DPOLCAT, 1310
leadingCoefficient
AMR, 1103
DPOLCAT, 1310
FAMR, 1143
FMCAT, 859
IDPC, 189
LOCPOWC, 1243
LODOCAT, 973
MLO, 1055
MTSCAT, 1196
OREPCAT, 931
POLYCAT, 1260
PSCAT, 1161
RPOLCAT, 1414
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
leadingMonomial
AMR, 1103
DPOLCAT, 1310
FAMR, 1143
FMCAT, 859
LOCPOWC, 1243
MTSCAT, 1196
POLYCAT, 1260
PSCAT, 1161
RPOLCAT, 1414
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
leadingSupport
IDPC, 189
leadingTerm
FMCAT, 859
leaf?
ALAGG, 824
BRAGG, 374
BTCAT, 528
DLAGG, 394
DSTRCAT, 133
LSAGG, 663
LZSTAGG, 640
PLACESC, 341
RCAGG, 353
STAGG, 588
URAGG, 509
leastMonomial
RPOLCAT, 1414
leaves
ALAGG, 824
BRAGG, 374
BTCAT, 528
DLAGG, 394
DSTRCAT, 133
LSAGG, 663
LZSTAGG, 640
RCAGG, 353
STAGG, 588
URAGG, 509
left
BRAGG, 374
BTCAT, 528
LeftAlgebra, 864
leftAlternative?
FINAALG, 1022
FRNAALG, 1118
leftCharacteristicPolynomial

FINAALG, 1022
FRNAALG, 1118
leftDiscriminant
FINAALG, 1022
FRNAALG, 1118
leftDivide
LODOCAT, 973
OREPCAT, 931
leftExactQuotient
LODOCAT, 973
OREPCAT, 931
leftExtendedGcd
LODOCAT, 973
OREPCAT, 931
leftGcd
LODOCAT, 973
OREPCAT, 931
leftLcm
LODOCAT, 973
OREPCAT, 931
leftMinimalPolynomial
FINAALG, 1022
FRNAALG, 1118
LeftModule, 655
leftNorm
FINAALG, 1022
FRNAALG, 1118
leftPower
FINAALG, 1022
FRNAALG, 1118
MONAD, 198
MONADWU, 327
NAALG, 982
NARNG, 681
NASRING, 742
leftQuotient
LODOCAT, 973
OREPCAT, 931
leftRankPolynomial
FRNAALG, 1118
leftRecip
FINAALG, 1022
FRNAALG, 1118
MONADWU, 327
NASRING, 742
leftRegularRepresentation
FINAALG, 1022
FRNAALG, 1118
leftRemainder
LODOCAT, 973
OREPCAT, 931
leftTrace
FINAALG, 1022
FRNAALG, 1118
leftTraceMatrix
FINAALG, 1022
FRNAALG, 1118
leftTrim
SRAGG, 789
STRICAT, 923
leftUnit
FINAALG, 1022
FRNAALG, 1118
leftUnitary
ACF, 1299
ACFS, 1500
ALGEBRA, 949
AMR, 1103
ATTREG, 11
BMODULE, 728
COMPCAT, 1626
COMRING, 845
DIRPCAT, 1005
DIVRING, 1015
DPOLCAT, 1311
ENTIRER, 855
EUCDOM, 1187
FAMR, 1144
FAXF, 1587
FFCAT, 1648
FFIELDC, 1521
FIELD, 1224
FINAALG, 1022
FINRALG, 1331
FLALG, 1043
FMCAT, 860
FMTC, 1110
FPC, 1326
FPS, 1533
FRAMALG, 1541
FRNAALG, 1118
FS, 1341

GCDDOM, 1131
INS, 1231
INTCAT, 1153
INTDOM, 1049
LIECAT, 968
LOCPOWC, 1243
LODOCAT, 974
MLO, 1055
MODULE, 876
MONOGEN, 1603
MTSCAT, 1196
NAALG, 982
OC, 1062
OINTDOM, 1137
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1549
PACPERC, 1372
PACRATC, 1614
PADICCT, 1251
PFECAT, 1204
PID, 1168
POLYCAT, 1260
PSCAT, 1161
QFCAT, 1380
QUATCAT, 1074
RCFIELD, 1393
RMATCAT, 903
RPOLCAT, 1415
SMATCAT, 1085
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1560
UPOLYC, 1478
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
VSPACE, 989
XALG, 942
XF, 1513
XFALG, 994
XPOLYC, 1096
leftUnits
FINAALG, 1022
FRNAALG, 1118
length
DQAGG, 542
INS, 1231
PTCAT, 896
QUAGG, 488
VECTCAT, 811
less?
A1AGG, 687
AGG, 70
ALAGG, 824
ARR2CAT, 359
BGAGG, 279
BRAGG, 374
BTAGG, 734
BTCAT, 528
CLAGG, 289
DIAGG, 535
DIOPS, 387
DIRPCAT, 1004
DLAGG, 394
DQAGG, 542
DSTRCAT, 133
ELAGG, 550
FLAGG, 558
FSAGG, 624
HOAGG, 183
IXAGG, 320
KDAGG, 632
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
MDAGG, 572
MSETAGG, 675
NTSCAT, 749
OMSAGG, 765
PRQAGG, 482
PSETCAT, 468
PTCAT, 896
QUAGG, 488
RCAGG, 353
RMATCAT, 903
RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SKAGG, 502

SMATCAT, 1085
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
URAGG, 509
VECTCAT, 811
LFCAT, 192
?**?, 193
acos, 193
acosh, 193
acot, 193
acoth, 193
acsc, 193
acsch, 193
asec, 193
asech, 193
asin, 193
asinh, 193
atan, 193
atanh, 193
Ci, 193
cos, 193
cosh, 193
cot, 193
coth, 193
csc, 193
csch, 193
dilog, 193
Ei, 193
erf, 193
exp, 193
frsenelC, 193
frsenelS, 193
integral, 193
li, 193
log, 193
pi, 193
sec, 193
sech, 193
Si, 193
sin, 193
sinh, 193
tan, 193
tanh, 193
li
LFCAT, 193
lieAdmissible?
FINAALG, 1022
FRNAALG, 1118
LieAlgebra, 966
lieAlgebra?
FINAALG, 1022
FRNAALG, 1118
LIECAT, 966
-?, 968
?*?, 968
?+?, 968
?-?, 968
?/?, 968
?=?, 968
?~=?, 968
0, 968
coerce, 968
construct, 968
hash, 968
JacobiIdentity, 968
latex, 968
leftUnitary, 968
NullSquare, 968
rightUnitary, 968
sample, 968
subtractIfCan, 968
zero?, 968
LiePoly
FLALG, 1043
lift
COMPCAT, 1625
FFCAT, 1647
MONOGEN, 1602
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
LinearAggregate, 404
linearAssociatedExp
FAXF, 1587
linearAssociatedLog
FAXF, 1587
linearAssociatedOrder

FAXF, 1587
LinearlyExplicitRingOver, 869
LinearOrdinaryDifferentialOperatorCategory, 971
LINEXP, 869
-?, 871
?**?, 871
?*?, 871
?+?, 871
?-?, 871
?=?, 871
?^?, 871
?~=?, 871
0, 871
1, 871
characteristic, 871
coerce, 871
hash, 871
latex, 871
one?, 871
recip, 871
reducedSystem, 871
sample, 871
subtractIfCan, 871
unitsKnown, 871
zero?, 871
LiouvillianFunctionCategory, 192
list
AFSPCAT, 273
ALAGG, 824
LSAGG, 663
PRSPCAT, 346
list?
SEXCAT, 248
ListAggregate, 659
listBranches
PPCURVE, 99
PSCURVE, 103
listOfLists
MATCAT, 437
RMATCAT, 903
SMATCAT, 1085
listOfTerms
FMCAT, 859
lllip
SPACEC, 258
lllp
SPACEC, 258
llprop
SPACEC, 258
LMODULE, 655
-?, 657
?*?, 657
?+?, 657
?-?, 657
?=?, 657
?~=?, 657
0, 657
coerce, 657
hash, 657
latex, 657
sample, 657
subtractIfCan, 657
zero?, 657
LNAGG, 404
?.?, 407
?=?, 407
?~=?, 407
#?, 407
any?, 407
coerce, 407
concat, 407
construct, 407
convert, 407
copy, 407
count, 407
delete, 407
elt, 407
empty, 407
empty?, 407
entries, 407
entry?, 407
eq?, 407
eval, 407
every?, 407
fill, 407
find, 407
finiteAggregate, 407
first, 407
hash, 407
index?, 407
indices, 407

insert, 407
latex, 407
less?, 407
map, 407
maxIndex, 407
member?, 407
members, 407
minIndex, 407
more?, 407
new, 407
parts, 407
qelt, 407
qsetelt, 407
reduce, 407
remove, 407
removeDuplicates, 407
sample, 407
setelt, 407
shallowlyMutable, 407
size?, 407
swap, 407
lo
SEGCAT, 110
SEGXCAT, 236
localParam
PLACESC, 341
localParamV
INFCLCT, 1366
localPointV
INFCLCT, 1366
LocalPowerSeriesCategory, 1239
LOCPOWC, 1239
-?, 1243
?**?, 1243
?+?, 1243
?-?, 1243
?.?, 1243
?/?, 1243
?=?, 1243
?^?, 1243
?~=?, 1243
?quo?, 1243
?rem?, 1243
0, 1243
1, 1243
approximate, 1243
associates?, 1243
canonicalsClosed, 1243
canonicalUnitNormal, 1243
center, 1243
characteristic, 1243
charthRoot, 1243
coefficient, 1243
coefOfFirstNonZeroTerm, 1243
coerce, 1243
commutative("*"), 1243
complete, 1243
D, 1243
degree, 1243
delay, 1243
differentiate, 1243
divide, 1243
euclideanSize, 1243
eval, 1243
expressIdealMember, 1243
exquo, 1243
extend, 1243
extendedEuclidean, 1243
factor, 1243
filterUpTo, 1243
findCoef, 1243
gcd, 1243
gcdPolynomial, 1243
hash, 1243
inv, 1243
latex, 1243
lcm, 1243
leadingCoefficient, 1243
leadingMonomial, 1243
leftUnitary, 1243
map, 1243
monomial, 1243
monomial2series, 1243
monomial?, 1243
multiEuclidean, 1243
multiplyExponents, 1243
noZeroDivisors, 1243
one?, 1243
order, 1243
orderIfNegative, 1243
pole?, 1243
posExpnPart, 1243

prime?, 1243
principalIdeal, 1243
printInfo, 1243
recip, 1243
reductum, 1243
removeFirstZeroes, 1243
removeZeroes, 1243
rightUnitary, 1243
sample, 1243
sbt, 1243
series, 1243
shift, 1243
sizeLess?, 1243
squareFree, 1243
squareFreePart, 1243
subtractIfCan, 1243
terms, 1243
truncate, 1243
unit?, 1243
unitCanonical, 1243
unitNormal, 1243
unitsKnown, 1243
variable, 1243
variables, 1243
zero?, 1243
LODOCAT, 971
-?, 973
?**?, 973
?*?, 973
?+?, 973
?-?, 973
?.?, 973
?=?, 973
?^?, 973
?~=?, 973
0, 973
1, 973
adjoint, 973
apply, 973
characteristic, 973
coefficient, 973
coefficients, 973
coerce, 973
content, 973
D, 973
degree, 973
directSum, 973
exquo, 973
hash, 973
latex, 973
leadingCoefficient, 973
leftDivide, 973
leftExactQuotient, 973
leftExtendedGcd, 973
leftGcd, 973
leftLcm, 973
leftQuotient, 973
leftRemainder, 973
leftUnitary, 974
minimumDegree, 973
monicLeftDivide, 973
monicRightDivide, 973
monomial, 973
one?, 973
primitivePart, 973
recip, 973
reductum, 973
retract, 973
retractIfCan, 973
rightDivide, 973
rightExactQuotient, 973
rightExtendedGcd, 973
rightGcd, 973
rightLcm, 973
rightQuotient, 973
rightRemainder, 973
rightUnitary, 974
sample, 973
subtractIfCan, 973
symmetricPower, 973
symmetricProduct, 973
symmetricSquare, 973
unitsKnown, 973
zero?, 973
log
COMPCAT, 1625
ELEMFUN, 28
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
ULSCAT, 1452

ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
LOGIC, 95
?/Γ*E*30*F*?, 96
?Γ*E*30*F*/?, 96
?=?, 96
?~=?, 96
~?, 96
Logic, 95
logIfCan
PTRANFN, 43
lookup
COMPCAT, 1625
DIRPCAT, 1004
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FINITE, 169
FSAGG, 624
MONOGEN, 1602
OC, 1062
ORDFIN, 337
PACFFC, 1548
low
SEGCAT, 110
SEGXCAT, 236
lowerCase
SRAGG, 789
STRICAT, 923
lp
SPACEC, 258
lprop
SPACEC, 258
lquo
FLALG, 1043
XFALG, 994
XPOLYC, 1096
LSAGG, 659
?<?, 663
?<=?, 663
?>?, 663
?>=?, 663
?.?, 663
?.first, 663
?.last, 663
?.rest, 663
?.value, 663
?=?, 663
?~=?, 663
#?, 663
any?, 663
child?, 663
children, 663
coerce, 663
concat, 663
construct, 663
convert, 663
copy, 663
copyInto, 663
count, 663
cycleEntry, 663
cycleLength, 663
cycleSplit, 663
cycleTail, 663
cyclic?, 663
delete, 663
distance, 663
elt, 663
empty, 663
empty?, 663
entries, 663
entry?, 663
eq?, 663
eval, 663
every?, 663
explicitlyFinite?, 663
fill, 663
find, 663
finiteAggregate, 663
first, 663
hash, 663
index?, 663
indices, 663
insert, 663
last, 663
latex, 663
leaf?, 663
leaves, 663
less?, 663
list, 663

map, 663
max, 663
maxIndex, 663
member?, 663
members, 663
merge, 663
min, 663
minIndex, 663
more?, 663
new, 663
node?, 663
nodes, 663
parts, 663
position, 663
possiblyInfinite?, 663
qelt, 663
qsetelt, 663
reduce, 663
remove, 663
removeDuplicates, 663
rest, 663
reverse, 663
sample, 663
second, 663
select, 663
setchildren, 663
setelt, 663
setfirst, 663
setlast, 663
setrest, 663
setvalue, 663
shallowlyMutable, 663
size?, 663
sort, 663
sorted?, 663
split, 663
swap, 663
tail, 663
third, 663
value, 663
LZSTAGG, 637
?.?, 640
?.first, 640
?.last, 640
?.rest, 640
?.value, 640
?=?, 640
?~=?, 640
#?, 640
any?, 640
child?, 640
children, 640
coerce, 640
complete, 640
concat, 640
construct, 640
convert, 640
copy, 640
count, 640
cycleEntry, 640
cycleLength, 640
cycleSplit, 640
cycleTail, 640
cyclic?, 640
delete, 640
distance, 640
elt, 640
empty, 640
empty?, 640
entries, 640
entry?, 640
eq?, 640
eval, 640
every?, 640
explicitEntries?, 640
explicitlyEmpty?, 640
explicitlyFinite?, 640
extend, 640
fill, 640
find, 640
first, 640
frst, 640
hash, 640
index?, 640
indices, 640
insert, 640
last, 640
latex, 640
lazy?, 640
lazyEvaluate, 640
leaf?, 640
leaves, 640

less?, 640
map, 640
maxIndex, 640
member?, 640
members, 640
minIndex, 640
more?, 640
new, 640
node?, 640
nodes, 640
numberOfComputedEntries, 640
parts, 640
possiblyInfinite?, 640
qelt, 640
qsetelt, 640
reduce, 640
remove, 640
removeDuplicates, 640
rest, 640
rst, 640
sample, 640
second, 640
select, 640
setchildren, 640
setelt, 640
setfirst, 640
setlast, 640
setrest, 640
setvalue, 640
size?, 640
split, 640
swap, 640
tail, 640
third, 640
value, 640

magnitude
PTCAT, 896
VECTCAT, 811
mainCoefficients
RPOLCAT, 1414
mainContent
RPOLCAT, 1414
mainDefiningPolynomial
RCFIELD, 1392
mainForm
RCFIELD, 1392
mainKernel
ACFS, 1500
ES, 302
FS, 1341
mainMonomial
RPOLCAT, 1414
mainPrimitivePart
RPOLCAT, 1414
mainSquareFreePart
RPOLCAT, 1414
mainValue
RCFIELD, 1392
mainVariable
POLYCAT, 1260
RPOLCAT, 1414
UPOLYC, 1477
mainVariable?
NTSCAT, 749
PSETCAT, 468
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
mainVariables
NTSCAT, 749
PSETCAT, 468
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
makeSUP
UPOLYC, 1477
makeVariable
DPOLCAT, 1310
DVARCAT, 296
mantissa
FPS, 1532
map
A1AGG, 687
ACFS, 1500
ALAGG, 824
AMR, 1103
ARR2CAT, 359
BGAGG, 279
BRAGG, 374

BTAGG, 734
BTCAT, 528
CLAGG, 289
COMPCAT, 1625
DIAGG, 535
DIOPS, 387
DIRPCAT, 1004
DLAGG, 394
DPOLCAT, 1310
DQAGG, 542
DSTRCAT, 133
ELAGG, 550
ES, 302
FAMR, 1143
FEVALAB, 160
FLAGG, 558
FMCAT, 859
FS, 1341
FSAGG, 624
HOAGG, 183
IDPC, 189
IXAGG, 320
KDAGG, 632
LNAGG, 407
LOCPOWC, 1243
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
MDAGG, 572
MSETAGG, 675
MTSCAT, 1196
NTSCAT, 749
OC, 1062
OMSAGG, 765
POLYCAT, 1260
PRQAGG, 482
PSCAT, 1161
PSETCAT, 468
PTCAT, 896
QFCAT, 1380
QUAGG, 488
QUATCAT, 1073
RCAGG, 353
RMATCAT, 903
RPOLCAT, 1414
RSETCAT, 703
SEGXCAT, 236
SETAGG, 494
SFRTCAT, 779
SKAGG, 502
SMATCAT, 1085
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
URAGG, 509
UTSCAT, 1281
VECTCAT, 811
XFALG, 994
XPOLYC, 1096
mapCoef
DIVCAT, 1181
FAMONC, 565
mapExponents
DPOLCAT, 1310
FAMR, 1143
POLYCAT, 1260
RPOLCAT, 1414
UPOLYC, 1477
mapGen
DIVCAT, 1181
FAMONC, 565
mask
INS, 1231
MATCAT, 412
-?, 437
?**?, 437
?*?, 437
?+?, 437
?-?, 437
?/?, 437
?=?, 437
?~=?, 437
#?, 437

antisymmetric?, 437
any?, 437
coerce, 437
column, 437
columnSpace, 437
copy, 437
count, 437
determinant, 437
diagonal?, 437
diagonalMatrix, 437
elt, 437
empty, 437
empty?, 437
eq?, 437
eval, 437
every?, 437
exquo, 437
fill, 437
finiteAggregate, 437
hash, 437
horizConcat, 437
inverse, 437
latex, 437
less?, 437
listOfLists, 437
map, 437
matrix, 437
maxColIndex, 437
maxRowIndex, 437
member?, 437
members, 437
minColIndex, 437
minordet, 437
minRowIndex, 437
more?, 437
ncols, 437
new, 437
nrows, 437
nullity, 437
nullSpace, 437
parts, 437
pfaffian, 437
qelt, 437
qsetelt, 437
rank, 437
row, 437
rowEchelon, 437
sample, 437
scalarMatrix, 437
setColumn, 437
setelt, 437
setRow, 437
setsubMatrix, 437
shallowlyMutable, 437
size?, 437
square?, 437
squareTop, 437
subMatrix, 437
swapColumns, 437
swapRows, 437
symmetric?, 437
transpose, 437
vertConcat, 437
zero, 437
match
SRAGG, 789
STRICAT, 923
match?
SRAGG, 789
STRICAT, 923
matrix
MATCAT, 437
RMATCAT, 903
SMATCAT, 1085
MatrixCategory, 412
max
A1AGG, 687
ACFS, 1500
ALAGG, 824
BTAGG, 734
CACHESET, 284
COMPCAT, 1625
DIRPCAT, 1004
DPOLCAT, 1310
DVARCAT, 296
ES, 302
FLAGG, 558
FMTC, 1110
FPS, 1532
FS, 1341
FSAGG, 624
INS, 1231

INTCAT, 1153
LSAGG, 663
OAGROUP, 756
OAMON, 577
OAMONS, 760
OASGP, 458
OC, 1062
OCAMON, 697
OINTDOM, 1137
OMSAGG, 765
ORDFIN, 337
ORDMON, 462
ORDRING, 881
ORDSET, 218
PERMCAT, 582
POLYCAT, 1260
PRQAGG, 482
PTCAT, 896
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1392
RNS, 1403
RPOLCAT, 1414
SRAGG, 789
STRICAT, 923
ULSCCAT, 1559
UPOLYC, 1477
VECTCAT, 811
maxColIndex
ARR2CAT, 359
MATCAT, 437
RMATCAT, 903
SMATCAT, 1085
maxdeg
XPOLYC, 1096
maxIndex
A1AGG, 687
ALAGG, 824
BTAGG, 734
DIRPCAT, 1004
ELAGG, 550
FLAGG, 558
IXAGG, 320
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
PTCAT, 896
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
VECTCAT, 811
maxRowIndex
ARR2CAT, 359
MATCAT, 437
RMATCAT, 903
SMATCAT, 1085
maxTower
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
MDAGG, 570
?=?, 572
?~=?, 572
#?, 572
any?, 572
bag, 572
coerce, 572
construct, 572
convert, 572
copy, 572
count, 572
dictionary, 572
duplicates, 572
empty, 572
empty?, 572
eq?, 572
eval, 572
every?, 572
extract, 572
find, 572
hash, 572
insert, 572
inspect, 572
latex, 572
less?, 572
map, 572
member?, 572
members, 572
more?, 572
parts, 572

reduce, 572
remove, 572
removeDuplicates, 572
sample, 572
select, 572
shallowlyMutable, 572
size?, 572
mdeg
RPOLCAT, 1414
measure
NUMINT, 203
ODECAT, 213
OPTCAT, 208
PDECAT, 222
member?
A1AGG, 687
ALAGG, 824
ARR2CAT, 359
BGAGG, 279
BRAGG, 374
BTAGG, 734
BTCAT, 528
CLAGG, 289
DIAGG, 535
DIOPS, 387
DIRPCAT, 1004
DLAGG, 394
DQAGG, 542
DSTRCAT, 133
ELAGG, 550
FLAGG, 558
FSAGG, 624
HOAGG, 183
IXAGG, 320
KDAGG, 632
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
MDAGG, 572
MSETAGG, 675
NTSCAT, 749
OMSAGG, 765
PRQAGG, 482
PSETCAT, 468
PTCAT, 896
QUAGG, 488
RCAGG, 353
RMATCAT, 903
RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SKAGG, 502
SMATCAT, 1085
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
URAGG, 509
VECTCAT, 811
members
A1AGG, 687
ALAGG, 824
ARR2CAT, 359
BGAGG, 279
BRAGG, 374
BTAGG, 734
BTCAT, 528
CLAGG, 289
DIAGG, 535
DIOPS, 387
DIRPCAT, 1004
DLAGG, 394
DQAGG, 542
DSTRCAT, 133
ELAGG, 550
FLAGG, 558
FSAGG, 624
HOAGG, 183
IXAGG, 320
KDAGG, 632
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
MDAGG, 572
MSETAGG, 675
NTSCAT, 749
OMSAGG, 765
PRQAGG, 482

PSETCAT, 468
PTCAT, 896
QUAGG, 488
RCAGG, 353
RMATCAT, 903
RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SKAGG, 502
SMATCAT, 1085
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
URAGG, 509
VECTCAT, 811
merge
A1AGG, 687
ALAGG, 824
BTAGG, 734
FLAGG, 558
LSAGG, 663
OMSAGG, 765
PRQAGG, 482
PTCAT, 896
SPACEC, 258
SRAGG, 789
STRICAT, 923
VECTCAT, 811
mesh
SPACEC, 258
mesh?
SPACEC, 258
min
A1AGG, 687
ACFS, 1500
ALAGG, 824
BTAGG, 734
CACHESET, 284
COMPCAT, 1625
DIRPCAT, 1004
DPOLCAT, 1310
DVARCAT, 296
ES, 302
FLAGG, 558
FMTC, 1110
FPS, 1532
FS, 1341
FSAGG, 624
INS, 1231
INTCAT, 1153
LSAGG, 663
OAGROUP, 756
OAMON, 577
OAMONS, 760
OASGP, 458
OC, 1062
OCAMON, 697
OINTDOM, 1137
OMSAGG, 765
ORDFIN, 337
ORDMON, 462
ORDRING, 881
ORDSET, 218
PERMCAT, 582
POLYCAT, 1260
PTCAT, 896
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1392
RNS, 1403
RPOLCAT, 1414
SRAGG, 789
STRICAT, 923
ULSCCAT, 1559
UPOLYC, 1477
VECTCAT, 811
minColIndex
ARR2CAT, 359
MATCAT, 437
RMATCAT, 903
SMATCAT, 1085
mindeg
XFALG, 994
XPOLYC, 1096
mindegTerm
XFALG, 994
XPOLYC, 1096
minimalPolynomial
COMPCAT, 1625

FAXF, 1587
FFCAT, 1647
FINRALG, 1331
FRAMALG, 1541
MONOGEN, 1602
minimumDegree
DPOLCAT, 1310
FAMR, 1143
LODOCAT, 973
MLO, 1055
OREPCAT, 931
POLYCAT, 1260
RPOLCAT, 1414
UPOLYC, 1477
minIndex
A1AGG, 687
ALAGG, 824
BTAGG, 734
DIRPCAT, 1004
ELAGG, 550
FLAGG, 558
IXAGG, 320
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
PTCAT, 896
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
VECTCAT, 811
minordet
MATCAT, 437
SMATCAT, 1085
minPoly
ACFS, 1500
ES, 302
FS, 1341
minRowIndex
ARR2CAT, 359
MATCAT, 437
RMATCAT, 903
SMATCAT, 1085
mirror
FLALG, 1043
XFALG, 994
XPOLYC, 1096
MLO, 1053
-?, 1055
?**?, 1055
?*?, 1055
?+?, 1055
?-?, 1055
?=?, 1055
?^?, 1055
?~=?, 1055
0, 1055
1, 1055
characteristic, 1055
coefficient, 1055
coerce, 1055
degree, 1055
hash, 1055
latex, 1055
leadingCoefficient, 1055
leftUnitary, 1055
minimumDegree, 1055
monomial, 1055
one?, 1055
recip, 1055
reductum, 1055
rightUnitary, 1055
sample, 1055
subtractIfCan, 1055
unitsKnown, 1055
zero?, 1055
modifyPointData
SPACEC, 258
MODULE, 874
-?, 876
?*?, 876
?+?, 876
?-?, 876
?=?, 876
?~=?, 876
0, 876
coerce, 876
hash, 876
latex, 876
leftUnitary, 876
rightUnitary, 876
sample, 876

subtractIfCan, 876
zero?, 876
Module, 874
moduloP
PADICCT, 1251
modulus
PADICCT, 1251
MONAD, 197
?**?, 198
?*?, 198
?=?, 198
?~=?, 198
coerce, 198
hash, 198
latex, 198
leftPower, 198
rightPower, 198
Monad, 197
MonadWithUnit, 325
MONADWU, 325
?**?, 327
?*?, 327
?=?, 327
?~=?, 327
1, 327
coerce, 327
hash, 327
latex, 327
leftPower, 327
leftRecip, 327
one?, 327
recip, 327
rightPower, 327
rightRecip, 327
monic?
RPOLCAT, 1414
monicDivide
DPOLCAT, 1310
POLYCAT, 1260
RPOLCAT, 1414
UPOLYC, 1477
monicLeftDivide
LODOCAT, 973
OREPCAT, 931
monicModulo
RPOLCAT, 1414
monicRightDivide
LODOCAT, 973
OREPCAT, 931
MONOGEN, 1598
-?, 1602
?**?, 1602
?*?, 1602
?+?, 1602
?-?, 1602
?/?, 1602
?=?, 1602
?^?, 1602
?~=?, 1602
?quo?, 1602
?rem?, 1602
0, 1602
1, 1602
associates?, 1602
basis, 1602
canonicalsClosed, 1602
canonicalUnitNormal, 1602
characteristic, 1602
characteristicPolynomial, 1602
charthRoot, 1602
coerce, 1602
commutative("*"), 1603
conditionP, 1602
convert, 1602
coordinates, 1602
createPrimitiveElement, 1602
D, 1602
definingPolynomial, 1602
derivationCoordinates, 1602
differentiate, 1602
discreteLog, 1602
discriminant, 1602
divide, 1602
euclideanSize, 1602
expressIdealMember, 1602
exquo, 1602
extendedEuclidean, 1602
factor, 1602
factorsOfCyclicGroupSize, 1602
gcd, 1602
gcdPolynomial, 1602
generator, 1602

hash, 1602
index, 1602
init, 1602
inv, 1602
latex, 1602
lcm, 1602
leftUnitary, 1603
lift, 1602
lookup, 1602
minimalPolynomial, 1602
multiEuclidean, 1602
nextItem, 1602
norm, 1602
noZeroDivisors, 1603
one?, 1602
order, 1602
prime?, 1602
primeFrobenius, 1602
primitive?, 1602
primitiveElement, 1602
principalIdeal, 1602
random, 1602
rank, 1602
recip, 1602
reduce, 1602
reducedSystem, 1602
regularRepresentation, 1602
representationType, 1602
represents, 1602
retract, 1602
retractIfCan, 1602
rightUnitary, 1603
sample, 1602
size, 1602
sizeLess?, 1602
squareFree, 1602
squareFreePart, 1602
subtractIfCan, 1602
tableForDiscreteLogarithm, 1602
trace, 1602
traceMatrix, 1602
unit?, 1602
unitCanonical, 1602
unitNormal, 1602
unitsKnown, 1603
zero?, 1602
MonogenicAlgebra, 1598
MonogenicLinearOperator, 1053
MONOID, 331
  ?**?, 332
  ?*?, 332
  ?=?, 332
  ?^?, 332
  ?~=?, 332
  1, 332
  coerce, 332
  hash, 332
  latex, 332
  one?, 332
  recip, 332
  sample, 332
Monoid, 331
monom
  FMCAT, 859
  XFALG, 994
  XPOLYC, 1096
monomial
  AMR, 1103
  DPOLCAT, 1310
  FAMR, 1143
  IDPC, 189
  LOCPOWC, 1243
  LODOCAT, 973
  MLO, 1055
  MTSCAT, 1196
  OREPCAT, 931
  POLYCAT, 1260
  PSCAT, 1161
  RPOLCAT, 1414
  ULSCAT, 1452
  ULSCCAT, 1559
  UPOLYC, 1477
  UPSCAT, 1213
  UPXSCAT, 1464
  UPXSCCA, 1574
  UTSCAT, 1281
monomial2series
  LOCPOWC, 1243
monomial?
  AMR, 1103
  DPOLCAT, 1310
  FAMR, 1143

FMCAT, 859
LOCPOWC, 1243
MTSCAT, 1196
POLYCAT, 1260
PSCAT, 1161
RPOLCAT, 1414
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
XFALG, 994
XPOLYC, 1096
monomials
DPOLCAT, 1310
FMCAT, 859
POLYCAT, 1260
RPOLCAT, 1414
UPOLYC, 1477
more?
A1AGG, 687
AGG, 70
ALAGG, 824
ARR2CAT, 359
BGAGG, 279
BRAGG, 374
BTAGG, 734
BTCAT, 528
CLAGG, 289
DIAGG, 535
DIOPS, 387
DIRPCAT, 1004
DLAGG, 394
DQAGG, 542
DSTRCAT, 133
ELAGG, 550
FLAGG, 558
FSAGG, 624
HOAGG, 183
IXAGG, 320
KDAGG, 632
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
MDAGG, 572
MSETAGG, 675
NTSCAT, 749
OMSAGG, 765
PRQAGG, 482
PSETCAT, 468
PTCAT, 896
QUAGG, 488
RCAGG, 353
RMATCAT, 903
RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SKAGG, 502
SMATCAT, 1085
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
URAGG, 509
VECTCAT, 811
MSETAGG, 673
?<?, 675
?=?, 675
?~=?, 675
#?, 675
any?, 675
bag, 675
brace, 675
coerce, 675
construct, 675
convert, 675
copy, 675
count, 675
dictionary, 675
difference, 675
duplicates, 675
empty, 675
empty?, 675
eq?, 675
eval, 675
every?, 675
extract, 675

find, 675
hash, 675
insert, 675
inspect, 675
intersect, 675
latex, 675
less?, 675
map, 675
member?, 675
members, 675
more?, 675
partiallyOrderedSet, 675
parts, 675
reduce, 675
remove, 675
removeDuplicates, 675
sample, 675
select, 675
set, 675
shallowlyMutable, 675
size?, 675
subset?, 675
symmetricDifference, 675
union, 675
MTSCAT, 1193
-?, 1196
?**?, 1196
?*?, 1196
?+?, 1196
?-?, 1196
?/?, 1196
?=?, 1196
?^?, 1196
?~=?, 1196
0, 1196
1, 1196
acos, 1196
acosh, 1196
acot, 1196
acoth, 1196
acsc, 1196
acsch, 1196
asec, 1196
asech, 1196
asin, 1196
asinh, 1196
associates?, 1196
atan, 1196
atanh, 1196
characteristic, 1196
charthRoot, 1196
coefficient, 1196
coerce, 1196
commutative("*"), 1196
complete, 1196
cos, 1196
cosh, 1196
cot, 1196
coth, 1196
csc, 1196
csch, 1196
D, 1196
degree, 1196
differentiate, 1196
eval, 1196
exp, 1196
exquo, 1196
extend, 1196
hash, 1196
integrate, 1196
latex, 1196
leadingCoefficient, 1196
leadingMonomial, 1196
leftUnitary, 1196
log, 1196
map, 1196
monomial, 1196
monomial?, 1196
noZeroDivisors, 1196
nthRoot, 1196
one?, 1196
order, 1196
pi, 1196
pole?, 1196
polynomial, 1196
recip, 1196
reductum, 1196
rightUnitary, 1196
sample, 1196
sec, 1196
sech, 1196
sin, 1196

sinh, 1196
sqrt, 1196
subtractIfCan, 1196
tan, 1196
tanh, 1196
unit?, 1196
unitCanonical, 1196
unitNormal, 1196
unitsKnown, 1196
variables, 1196
zero?, 1196
mulmod
INS, 1231
MultiDictionary, 570
multiEuclidean
ACF, 1298
ACFS, 1500
COMPCAT, 1625
EUCDOM, 1187
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FPC, 1325
FPS, 1532
FS, 1341
INS, 1231
LOCPOWC, 1243
MONOGEN, 1602
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPXSCAT, 1464
UPXSCCA, 1574
XF, 1513
multiplicativeValuation
ATTREG, 11
COMPCAT, 1626
INS, 1231
multiplyCoefficients
ULSCAT, 1452
ULSCCAT, 1559
UTSCAT, 1281
multiplyExponents
LOCPOWC, 1243
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
MultisetAggregate, 673
multivariate
DPOLCAT, 1310
POLYCAT, 1260
RPOLCAT, 1414
UPOLYC, 1477
MultivariateTaylorSeriesCategory, 1193
multV
INFCLCT, 1366
mvar
NTSCAT, 749
PSETCAT, 468
RPOLCAT, 1414
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598

NAALG, 980
-?, 982
?**?, 982
?*?, 982
?+?, 982
?-?, 982
?=?, 982
?~=?, 982
0, 982
antiCommutator, 982
associator, 982
coerce, 982
commutator, 982
hash, 982

latex, 982
leftPower, 982
leftUnitary, 982
plenaryPower, 982
rightPower, 982
rightUnitary, 982
sample, 982
subtractIfCan, 982
zero?, 982
name
FILECAT, 165
FNCAT, 173
nand
BTAGG, 734
NARNG, 679
-?, 681
?**?, 681
?*?, 681
?+?, 681
?-?, 681
?=?, 681
?~=?, 681
0, 681
antiCommutator, 681
associator, 681
coerce, 681
commutator, 681
hash, 681
latex, 681
leftPower, 681
rightPower, 681
sample, 681
subtractIfCan, 681
zero?, 681
NASRING, 740
-?, 742
?**?, 742
?*?, 742
?+?, 742
?-?, 742
?=?, 742
?~=?, 742
0, 742
1, 742
antiCommutator, 742
associator, 742
characteristic, 742
coerce, 742
commutator, 742
hash, 742
latex, 742
leftPower, 742
leftRecip, 742
one?, 742
recip, 742
rightPower, 742
rightRecip, 742
sample, 742
subtractIfCan, 742
zero?, 742
ncols
ARR2CAT, 359
MATCAT, 437
RMATCAT, 903
SMATCAT, 1085
negative?
DIRPCAT, 1004
FPS, 1532
INS, 1231
INTCAT, 1153
OINTDOM, 1137
ORDRING, 881
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
RRCC, 231
ULSCCAT, 1559
new
A1AGG, 687
ALAGG, 824
ARR2CAT, 359
BTAGG, 734
ELAGG, 550
FLAGG, 558
FNCAT, 173
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
PTCAT, 896
SRAGG, 789
STAGG, 588

STRICAT, 923
VECTCAT, 811
newElement
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
next
DLAGG, 394
nextItem
COMPCAT, 1625
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
INS, 1231
MONOGEN, 1602
PACFFC, 1548
QFCAT, 1380
STEP, 253
ULSCCAT, 1559
UPOLYC, 1477
nextsubResultant2
RPOLCAT, 1414
node
BTCAT, 528
node?
ALAGG, 824
BRAGG, 374
BTCAT, 528
DLAGG, 394
DSTRCAT, 133
LSAGG, 663
LZSTAGG, 640
RCAGG, 353
STAGG, 588
URAGG, 509
nodes
ALAGG, 824
BRAGG, 374
BTCAT, 528
DLAGG, 394
DSTRCAT, 133
LSAGG, 663
LZSTAGG, 640
RCAGG, 353
STAGG, 588
URAGG, 509
noetherian
ATTREG, 11
NonAssociativeAlgebra, 980
NonAssociativeRing, 740
NonAssociativeRng, 679
noncommutativeJordanAlgebra?
FINAALG, 1022
FRNAALG, 1118
nonSingularModel
FFCAT, 1647
nor
BTAGG, 734
norm
COMPCAT, 1625
FAXF, 1587
FFCAT, 1647
FINRALG, 1331
FPS, 1532
FRAMALG, 1541
MONOGEN, 1602
OC, 1062
QUATCAT, 1073
RNS, 1403
normal?
FAXF, 1587
normalElement
FAXF, 1587
normalizeAtInfinity
FFCAT, 1647
normalized?
NTSCAT, 749
RPOLCAT, 1414
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
NormalizedTriangularSetCategory, 745
not?
BTAGG, 734
noZeroDivisors
ACF, 1299
ACFS, 1500
AMR, 1103
ATTREG, 11
COMPCAT, 1626

DIVRING, 1014
DPOLCAT, 1311
ENTIRER, 855
EUCDOM, 1187
FAMR, 1143
FAXF, 1587
FFCAT, 1648
FFIELDC, 1521
FIELD, 1224
FMTC, 1110
FPC, 1326
FPS, 1533
FS, 1341
GCDDOM, 1131
INS, 1231
INTCAT, 1153
INTDOM, 1049
LOCPOWC, 1243
MONOGEN, 1603
MTSCAT, 1196
OINTDOM, 1137
PACEXTC, 1668
PACFFC, 1549
PACPERC, 1371
PACRATC, 1614
PADICCT, 1251
PFECAT, 1204
PID, 1168
POLYCAT, 1260
PSCAT, 1161
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1392
RPOLCAT, 1415
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1560
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1575
UTSCAT, 1281
XF, 1513
XFALG, 994
XPOLYC, 1096
nrows
ARR2CAT, 359
MATCAT, 437
RMATCAT, 903
SMATCAT, 1085
nthCoef
DIVCAT, 1181
FAMONC, 565
nthFactor
DIVCAT, 1181
FAMONC, 565
nthRoot
ACF, 1298
ACFS, 1500
COMPCAT, 1625
FPS, 1532
INTCAT, 1153
MTSCAT, 1196
RADCAT, 53
RCFIELD, 1392
RNS, 1403
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
nthRootIfCan
PTRANFN, 43
NTSCAT, 745
?=?, 749
?~=?, 749
#?, 749
algebraic?, 749
algebraicCoefficients?, 749
algebraicVariables, 749
any?, 749
augment, 749
autoReduced?, 749
basicSet, 749
coerce, 749
coHeight, 749
collect, 749
collectQuasiMonic, 749
collectUnder, 749
collectUpper, 749
construct, 749
convert, 749

copy, 749
count, 749
degree, 749
empty, 749
empty?, 749
eq?, 749
eval, 749
every?, 749
extend, 749
extendIfCan, 749
find, 749
finiteAggregate, 750
first, 749
hash, 749
headReduce, 749
headReduced?, 749
headRemainder, 749
infRittWu?, 749
initiallyReduce, 749
initiallyReduced?, 749
initials, 749
internalAugment, 749
intersect, 749
invertible?, 749
invertibleElseSplit?, 749
invertibleSet, 749
last, 749
lastSubResultant, 749
lastSubResultantElseSplit, 749
latex, 749
less?, 749
mainVariable?, 749
mainVariables, 749
map, 749
member?, 749
members, 749
more?, 749
mvar, 749
normalized?, 749
parts, 749
purelyAlgebraic?, 749
purelyAlgebraicLeadingMonomial?, 749
purelyTranscendental?, 749
quasiComponent, 749
reduce, 749
reduceByQuasiMonic, 749
reduced?, 749
remainder, 749
remove, 749
removeDuplicates, 749
removeZero, 749
rest, 749
retract, 749
retractIfCan, 749
rewriteIdealWithHeadRemainder, 749
rewriteIdealWithRemainder, 749
rewriteSetWithReduction, 749
roughBase?, 749
roughEqualIdeals?, 749
roughSubIdeal?, 749
roughUnitIdeal?, 749
sample, 749
select, 749
shallowlyMutable, 750
size?, 749
sort, 749
squareFreePart, 749
stronglyReduce, 749
stronglyReduced?, 749
triangular?, 749
trivialIdeal?, 749
variables, 749
zeroSetSplit, 749
zeroSetSplitIntoTriangularSystems, 749
null?
SEXCAT, 248
nullity
MATCAT, 437
RMATCAT, 903
SMATCAT, 1085
nullSpace
MATCAT, 437
RMATCAT, 903
SMATCAT, 1085
NullSquare
ATTREG, 11
FLALG, 1043
LIECAT, 968
numberOfComponents
FFCAT, 1647
SPACEC, 258
numberOfComposites

SPACEC, 258
numberOfComputedEntries
LZSTAGG, 640
numberOfMonomials
DPOLCAT, 1310
FAMR, 1143
FMCAT, 859
POLYCAT, 1260
RPOLCAT, 1414
UPOLYC, 1477
numer
ACFS, 1500
FS, 1341
QFCAT, 1380
ULSCCAT, 1559
numerator
ACFS, 1500
FS, 1341
QFCAT, 1380
ULSCCAT, 1559
numericalIntegration
NUMINT, 203
NumericalIntegrationCategory, 202
numericalOptimization
OPTCAT, 208
NumericalOptimizationCategory, 207
NUMINT, 202
?=?, 203
?~=?, 203
coerce, 203
hash, 203
latex, 203
measure, 203
numericalIntegration, 203

OAGROUP, 755
-?, 756
?<?, 756
?<=?, 756
?>?, 756
?>=?, 756
?*?, 756
?+?, 756
?-?, 756
?=?, 756
?~=?, 756
0, 756
coerce, 756
hash, 756
latex, 756
max, 756
min, 756
sample, 756
subtractIfCan, 756
zero?, 756
OAMON, 576
?<?, 577
?<=?, 577
?>?, 577
?>=?, 577
?*?, 577
?+?, 577
?=?, 577
?~=?, 577
0, 577
coerce, 577
hash, 577
latex, 577
max, 577
min, 577
sample, 577
zero?, 577
OAMONS, 759
?<?, 760
?<=?, 760
?>?, 760
?>=?, 760
?*?, 760
?+?, 760
?=?, 760
?~=?, 760
0, 760
coerce, 760
hash, 760
latex, 760
max, 760
min, 760
sample, 760
subtractIfCan, 760
sup, 760
zero?, 760
OASGP, 456

?<?, 458
?<=?, 458
?>?, 458
?>=?, 458
?*?, 458
?+?, 458
?=?, 458
?~=?, 458
0, 458
coerce, 458
hash, 458
latex, 458
max, 458
min, 458
sample, 458
zero?, 458
objects
SPACEC, 258
OC, 1059
-?, 1062
?<?, 1062
?<=?, 1062
?>?, 1062
?>=?, 1062
?**?, 1062
?*?, 1062
?+?, 1062
?-?, 1062
?.?, 1062
?=?, 1062
?^?, 1062
?~=?, 1062
0, 1062
1, 1062
abs, 1062
characteristic, 1062
charthRoot, 1062
coerce, 1062
conjugate, 1062
convert, 1062
eval, 1062
hash, 1062
imagE, 1062
imagI, 1062
imagi, 1062
imagJ, 1062
imagj, 1062
imagK, 1062
imagk, 1062
index, 1062
inv, 1062
latex, 1062
leftUnitary, 1062
lookup, 1062
map, 1062
max, 1062
min, 1062
norm, 1062
octon, 1062
one?, 1062
random, 1062
rational, 1062
rational?, 1062
rationalIfCan, 1062
real, 1062
recip, 1062
retract, 1062
retractIfCan, 1062
rightUnitary, 1062
sample, 1062
size, 1062
subtractIfCan, 1062
unitsKnown, 1062
zero?, 1062
OCAMON, 696
?<?, 697
?<=?, 697
?>?, 697
?>=?, 697
?*?, 697
?+?, 697
?=?, 697
?~=?, 697
0, 697
coerce, 697
hash, 697
latex, 697
max, 697
min, 697
sample, 697
subtractIfCan, 697
zero?, 697

octon
OC, 1062
OctonionCategory, 1059
odd?
ACFS, 1500
ES, 302
FS, 1341
INS, 1231
ODECAT, 212
?=?, 213
?~=?, 213
coerce, 213
hash, 213
latex, 213
measure, 213
ODESolve, 213
ODESolve
ODECAT, 213
OINTDOM, 1135
-?, 1137
?<?, 1137
?<=?, 1137
?>?, 1137
?>=?, 1137
?**?, 1137
?*?, 1137
?+?, 1137
?-?, 1137
?=?, 1137
?^?, 1137
?~=?, 1137
0, 1137
1, 1137
abs, 1137
associates?, 1137
characteristic, 1137
coerce, 1137
commutative("*"), 1137
exquo, 1137
hash, 1137
latex, 1137
leftUnitary, 1137
max, 1137
min, 1137
negative?, 1137
noZeroDivisors, 1137
one?, 1137
positive?, 1137
recip, 1137
rightUnitary, 1137
sample, 1137
sign, 1137
subtractIfCan, 1137
unit?, 1137
unitCanonical, 1137
unitNormal, 1137
unitsKnown, 1137
zero?, 1137
OM, 39
OMwrite, 40
OMSAGG, 763
?<?, 765
?=?, 765
?~?, 765
#?, 765
any?, 765
bag, 765
brace, 765
coerce, 765
construct, 765
convert, 765
copy, 765
count, 765
dictionary, 765
difference, 765
duplicates, 765
empty, 765
empty?, 765
eq?, 765
eval, 765
every?, 765
extract, 765
find, 765
finiteAggregate, 766
hash, 765
insert, 765
inspect, 765
intersect, 765
latex, 765
less?, 765
map, 765
max, 765

member?, 765
members, 765
merge, 765
min, 765
more?, 765
partiallyOrderedSet, 765
parts, 765
reduce, 765
remove, 765
removeDuplicates, 765
sample, 765
select, 765
set, 765
shallowlyMutable, 765
size?, 765
subset?, 765
symmetricDifference, 765
union, 765
OMwrite
OM, 40
STRICAT, 923
one?
ACF, 1298
ACFS, 1500
ALGEBRA, 949
AMR, 1103
CHARNZ, 835
CHARZ, 840
COMPCAT, 1625
COMRING, 845
DIFEXT, 955
DIFRING, 850
DIRPCAT, 1004
DIVRING, 1014
DPOLCAT, 1310
ENTIRER, 855
EUCDOM, 1187
FAMR, 1143
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FINRALG, 1331
FLINEXP, 962
FMTC, 1110
FPC, 1325
FPS, 1532
FRAMALG, 1541
FS, 1341
GCDDOM, 1131
GROUP, 400
INS, 1231
INTCAT, 1153
INTDOM, 1049
LALG, 865
LINEXP, 871
LOCPOWC, 1243
LODOCAT, 973
MLO, 1055
MONADWU, 327
MONOGEN, 1602
MONOID, 332
MTSCAT, 1196
NASRING, 742
OC, 1062
OINTDOM, 1137
ORDMON, 462
ORDRING, 881
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PDRING, 887
PERMCAT, 582
PFECAT, 1204
PID, 1168
POLYCAT, 1260
PSCAT, 1161
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1392
RING, 772
RNS, 1403
RPOLCAT, 1414
SMATCAT, 1085
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213

UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
XALG, 942
XF, 1513
XFALG, 994
XPOLYC, 1096
OneDimensionalArrayAggregate, 684
open
FILECAT, 165
OpenMath, 39
operator
ACFS, 1500
ES, 302
FS, 1341
operators
ACFS, 1500
ES, 302
FS, 1341
OPTCAT, 207
?=?, 208
?~=?, 208
coerce, 208
hash, 208
latex, 208
measure, 208
numericalOptimization, 208
orbit
AFSPCAT, 273
PERMCAT, 582
PRSPCAT, 346
order
COMPCAT, 1625
DPOLCAT, 1310
DVARCAT, 296
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FPC, 1325
FPS, 1532
LOCPOWC, 1243
MONOGEN, 1602
MTSCAT, 1196
PACEXTC, 1668
PACFFC, 1548
PACRATC, 1613
PADICCT, 1251
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
XF, 1513
OrderedAbelianGroup, 755
OrderedAbelianMonoid, 576
OrderedAbelianMonoidSup, 759
OrderedAbelianSemiGroup, 456
OrderedCancellationAbelianMonoid, 696
OrderedFinite, 336
OrderedIntegralDomain, 1135
OrderedMonoid, 461
OrderedMultisetAggregate, 763
OrderedRing, 879
OrderedSet, 216
orderIfNegative
LOCPOWC, 1243
ORDFIN, 336
?<?, 337
?<=?, 337
?>?, 337
?>=?, 337
?=?, 337
?~=?, 337
coerce, 337
hash, 337
index, 337
latex, 337
lookup, 337
max, 337
min, 337
random, 337
size, 337
OrdinaryDifferentialEquationsSolverCategory, 212
ORDMON, 461
?<?, 462
?<=?, 462
?>?, 462
?>=?, 462
?**?, 462

?*?, 462
?=?, 462
?^?, 462
?~=?, 462
1, 462
coerce, 462
hash, 462
latex, 462
max, 462
min, 462
one?, 462
recip, 462
sample, 462
ORDRING, 879
-?, 881
?<?, 881
?<=?, 881
?>?, 881
?>=?, 881
?**?, 881
?*?, 881
?+?, 881
?-?, 881
?=?, 881
?^?, 881
?~=?, 881
0, 881
1, 881
abs, 881
characteristic, 881
coerce, 881
hash, 881
latex, 881
max, 881
min, 881
negative?, 881
one?, 881
positive?, 881
recip, 881
sample, 881
sign, 881
subtractIfCan, 881
unitsKnown, 881
zero?, 881
ORDSET, 216
?<?, 218
?<=?, 218
?>?, 218
?>=?, 218
?=?, 218
?~=?, 218
coerce, 218
hash, 218
latex, 218
max, 218
min, 218
OREPCAT, 929
-?, 931
?**?, 931
?*?, 931
?+?, 931
?-?, 931
?=?, 931
?^?, 931
?~=?, 931
0, 931
1, 931
apply, 931
characteristic, 931
coefficient, 931
coefficients, 931
coerce, 931
content, 931
degree, 931
exquo, 931
hash, 931
latex, 931
leadingCoefficient, 931
leftDivide, 931
leftExactQuotient, 931
leftExtendedGcd, 931
leftGcd, 931
leftLcm, 931
leftQuotient, 931
leftRemainder, 931
leftUnitary, 931
minimumDegree, 931
monicLeftDivide, 931
monicRightDivide, 931
monomial, 931
one?, 931
primitivePart, 931

recip, 931
reductum, 931
retract, 931
retractIfCan, 931
rightDivide, 931
rightExactQuotient, 931
rightExtendedGcd, 931
rightGcd, 931
rightLcm, 931
rightQuotient, 931
rightRemainder, 931
rightUnitary, 931
sample, 931
subtractIfCan, 931
unitsKnown, 931
zero?, 931
origin
AFSPCAT, 273
outerProduct
PTCAT, 896
VECTCAT, 811
outputAsFortran
FMC, 142
FMFUN, 146
FORTCAT, 85
FORTFN, 137
FVFUN, 155

PACEXTC, 1665
-?, 1668
?**?, 1668
?*?, 1668
?+?, 1668
?-?, 1668
?/?, 1668
?=?, 1668
?^?, 1668
?~=?, 1668
?quo?, 1668
?rem?, 1668
0, 1668
1, 1668
algebraic?, 1668
associates?, 1668
canonicalUnitNormal, 1668
characteristic, 1668
charthRoot, 1668
coerce, 1668
commutative("*"), 1668
conjugate, 1668
definingPolynomial, 1668
degree, 1668
dimension, 1668
discreteLog, 1668
distinguishedRootsOf, 1668
divide, 1668
euclideanSize, 1668
expressIdealMember, 1668
exquo, 1668
extDegree, 1668
extendedEuclidean, 1668
extensionDegree, 1668
factor, 1668
Frobenius, 1668
fullOutput, 1668
gcd, 1668
gcdPolynomial, 1668
ground?, 1668
hash, 1668
inGroundField?, 1668
inv, 1668
latex, 1668
lcm, 1668
leftUnitary, 1668
lift, 1668
maxTower, 1668
multiEuclidean, 1668
newElement, 1668
noZeroDivisors, 1668
one?, 1668
order, 1668
previousTower, 1668
prime?, 1668
primeFrobenius, 1668
principalIdeal, 1668
recip, 1668
reduce, 1668
retract, 1668
retractIfCan, 1668
rightUnitary, 1668
sample, 1668
setTower, 1668

sizeLess?, 1668
squareFree, 1668
squareFreePart, 1668
subtractIfCan, 1668
transcendenceDegree, 1668
transcendent?, 1668
unit?, 1668
unitCanonical, 1668
unitNormal, 1668
unitsKnown, 1668
vectorise, 1668
zero?, 1668
PACFFC, 1546
-?, 1548
?**?, 1548
?*?, 1548
?+?, 1548
?-?, 1548
?/?, 1548
?=?, 1548
?^?, 1548
?~=?, 1548
?quo?, 1548
?rem?, 1548
0, 1548
1, 1548
associates?, 1548
canonicalsClosed, 1549
canonicalUnitNormal, 1548
characteristic, 1548
charthRoot, 1548
coerce, 1548
commutative("*"), 1549
conditionP, 1548
conjugate, 1548
createPrimitiveElement, 1548
D, 1548
definingPolynomial, 1548
differentiate, 1548
discreteLog, 1548
distinguishedRootsOf, 1548
divide, 1548
euclideanSize, 1548
expressIdealMember, 1548
exquo, 1548
extDegree, 1548
extendedEuclidean, 1548
factor, 1548
factorsOfCyclicGroupSize, 1548
fullOutput, 1548
gcd, 1548
gcdPolynomial, 1548
ground?, 1548
hash, 1548
index, 1548
init, 1548
inv, 1548
latex, 1548
lcm, 1548
leftUnitary, 1549
lift, 1548
lookup, 1548
maxTower, 1548
multiEuclidean, 1548
newElement, 1548
nextItem, 1548
noZeroDivisors, 1549
one?, 1548
order, 1548
previousTower, 1548
prime?, 1548
primeFrobenius, 1548
primitive?, 1548
primitiveElement, 1548
principalIdeal, 1548
random, 1548
recip, 1548
reduce, 1548
representationType, 1548
rightUnitary, 1549
sample, 1548
setTower, 1548
size, 1548
sizeLess?, 1548
squareFree, 1548
squareFreePart, 1548
subtractIfCan, 1548
tableForDiscreteLogarithm, 1548
unit?, 1548
unitCanonical, 1548
unitNormal, 1548
unitsKnown, 1549

vectorise, 1548
zero?, 1548
PACPERC, 1369
-?, 1371
?**?, 1371
?*?, 1371
?+?, 1371
?-?, 1371
?/?, 1371
?=?, 1371
?^?, 1371
?~=?, 1371
?quo?, 1371
?rem?, 1371
0, 1371
1, 1371
associates?, 1371
canonicalsClosed, 1371
canonicalUnitNormal, 1371
characteristic, 1371
coerce, 1371
commutative("*"), 1372
conjugate, 1371
definingPolynomial, 1371
distinguishedRootsOf, 1371
divide, 1371
euclideanSize, 1371
expressIdealMember, 1371
exquo, 1371
extDegree, 1371
extendedEuclidean, 1371
factor, 1371
fullOutput, 1371
gcd, 1371
gcdPolynomial, 1371
ground?, 1371
hash, 1371
inv, 1371
latex, 1371
lcm, 1371
leftUnitary, 1372
lift, 1371
maxTower, 1371
multiEuclidean, 1371
newElement, 1371
noZeroDivisors, 1371
one?, 1371
previousTower, 1371
prime?, 1371
principalIdeal, 1371
recip, 1371
reduce, 1371
rightUnitary, 1372
sample, 1371
setTower, 1371
sizeLess?, 1371
squareFree, 1371
squareFreePart, 1371
subtractIfCan, 1371
unit?, 1371
unitCanonical, 1371
unitNormal, 1371
unitsKnown, 1372
vectorise, 1371
zero?, 1371
PACRATC, 1610
-?, 1613
?**?, 1613
?*?, 1613
?+?, 1613
?-?, 1613
?/?, 1613
?=?, 1613
?^?, 1613
?~=?, 1613
?quo?, 1613
?rem?, 1613
0, 1613
1, 1613
algebraic?, 1613
associates?, 1613
canonicalsClosed, 1613
canonicalUnitNormal, 1613
characteristic, 1613
charthRoot, 1613
coerce, 1613
commutative("*"), 1614
conjugate, 1613
definingPolynomial, 1613
degree, 1613
dimension, 1613
discreteLog, 1613

distinguishedRootsOf, 1613
divide, 1613
euclideanSize, 1613
expressIdealMember, 1613
exquo, 1613
extDegree, 1613
extendedEuclidean, 1613
extensionDegree, 1613
factor, 1613
Frobenius, 1613
fullOutput, 1613
gcd, 1613
gcdPolynomial, 1613
ground?, 1613
hash, 1613
inGroundField?, 1613
inv, 1613
latex, 1613
lcm, 1613
leftUnitary, 1614
lift, 1613
maxTower, 1613
multiEuclidean, 1613
newElement, 1613
noZeroDivisors, 1614
one?, 1613
order, 1613
previousTower, 1613
prime?, 1613
primeFrobenius, 1613
principalIdeal, 1613
recip, 1613
reduce, 1613
retract, 1613
retractIfCan, 1613
rightUnitary, 1614
sample, 1613
setTower, 1613
sizeLess?, 1613
squareFree, 1613
squareFreePart, 1613
subtractIfCan, 1613
transcendenceDegree, 1613
transcendent?, 1613
unit?, 1613
unitCanonical, 1613
unitNormal, 1613
unitsKnown, 1614
vectorise, 1613
zero?, 1613
PADICCT, 1249
-?, 1251
?**?, 1251
?*?, 1251
?+?, 1251
?-?, 1251
?=?, 1251
?^?, 1251
?~=?, 1251
?quo?, 1251
?rem?, 1251
0, 1251
1, 1251
approximate, 1251
associates?, 1251
characteristic, 1251
coerce, 1251
commutative("*"), 1251
complete, 1251
digits, 1251
divide, 1251
euclideanSize, 1251
expressIdealMember, 1251
exquo, 1251
extend, 1251
extendedEuclidean, 1251
gcd, 1251
gcdPolynomial, 1251
hash, 1251
latex, 1251
lcm, 1251
leftUnitary, 1251
moduloP, 1251
modulus, 1251
multiEuclidean, 1251
noZeroDivisors, 1251
one?, 1251
order, 1251
principalIdeal, 1251
quotientByP, 1251
recip, 1251
rightUnitary, 1251

root, 1251
sample, 1251
sizeLess?, 1251
sqrt, 1251
subtractIfCan, 1251
unit?, 1251
unitCanonical, 1251
unitNormal, 1251
unitsKnown, 1251
zero?, 1251
PAdicIntegerCategory, 1249
pair?
SEXCAT, 248
paren
ACFS, 1500
ES, 302
FS, 1341
PartialDifferentialEquationsSolverCategory, 221
PartialDifferentialRing, 885
partiallyOrderedSet
ATTREG, 11
FSAGG, 624
MSETAGG, 675
OMSAGG, 765
SETAGG, 494
PartialTranscendentalFunctions, 42
parts
A1AGG, 687
ALAGG, 824
ARR2CAT, 359
BGAGG, 279
BRAGG, 374
BTAGG, 734
BTCAT, 528
CLAGG, 289
DIAGG, 535
DIOPS, 387
DIRPCAT, 1004
DLAGG, 394
DQAGG, 542
DSTRCAT, 133
ELAGG, 550
FLAGG, 558
FSAGG, 624
HOAGG, 183
IXAGG, 320
KDAGG, 632
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
MDAGG, 572
MSETAGG, 675
NTSCAT, 749
OMSAGG, 765
PRQAGG, 482
PSETCAT, 468
PTCAT, 896
QUAGG, 488
RCAGG, 353
RMATCAT, 903
RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SKAGG, 502
SMATCAT, 1085
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
URAGG, 509
VECTCAT, 811
PATAB, 47
convert, 48
PATMAB, 226
?=?, 227
?~=?, 227
coerce, 227
hash, 227
latex, 227
patternMatch, 227
Patternable, 47
patternMatch
ACFS, 1500
COMPCAT, 1625
DPOLCAT, 1310
FPATMAB, 93
FPS, 1532
FS, 1341
INS, 1231

PATMAB, 227
POLYCAT, 1260
QFCAT, 1380
RNS, 1403
RPOLCAT, 1414
ULSCCAT, 1559
UPOLYC, 1477
PatternMatchable, 226
PDECAT, 221
?=?, 222
?~=?, 222
coerce, 222
hash, 222
latex, 222
measure, 222
PDESolve, 222
PDESolve
PDECAT, 222
PDRING, 885
-?, 887
?**?, 887
?*?, 887
?+?, 887
?-?, 887
?=?, 887
?^?, 887
?~=?, 887
0, 887
1, 887
characteristic, 887
coerce, 887
D, 887
differentiate, 887
hash, 887
latex, 887
one?, 887
recip, 887
sample, 887
subtractIfCan, 887
unitsKnown, 887
zero?, 887
PERMCAT, 580
?<?, 582
?<=?, 582
?>?, 582
?>=?, 582
?**?, 582
?*?, 582
?.?, 582
?/?, 582
?=?, 582
?^?, 582
?~=?, 582
1, 582
coerce, 582
commutator, 582
conjugate, 582
cycle, 582
cycles, 582
eval, 582
hash, 582
inv, 582
latex, 582
max, 582
min, 582
one?, 582
orbit, 582
recip, 582
sample, 582
permutation
CFCAT, 21
COMBOPC, 74
INS, 1231
PermutationCategory, 580
pfaffian
MATCAT, 437
PFECAT, 1202
-?, 1204
?**?, 1204
?*?, 1204
?+?, 1204
?-?, 1204
?=?, 1204
?^?, 1204
?~=?, 1204
0, 1204
1, 1204
associates?, 1204
characteristic, 1204
charthRoot, 1204
coerce, 1204
commutative("*"), 1204

conditionP, 1204
exquo, 1204
factor, 1204
factorPolynomial, 1204
factorSquareFreePolynomial, 1204
gcd, 1204
gcdPolynomial, 1204
hash, 1204
latex, 1204
lcm, 1204
leftUnitary, 1204
noZeroDivisors, 1204
one?, 1204
prime?, 1204
recip, 1204
rightUnitary, 1204
sample, 1204
solveLinearPolynomialEquation, 1204
squareFree, 1204
squareFreePart, 1204
squareFreePolynomial, 1204
subtractIfCan, 1204
unit?, 1204
unitCanonical, 1204
unitNormal, 1204
unitsKnown, 1204
zero?, 1204
pi
COMPCAT, 1625
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
PID, 1166
-?, 1168
?**?, 1168
?*?, 1168
?+?, 1168
?-?, 1168
?=?, 1168
?^?, 1168
?~=?, 1168
0, 1168
1, 1168
associates?, 1168
characteristic, 1168
coerce, 1168
commutative("*"), 1168
expressIdealMember, 1168
exquo, 1168
gcd, 1168
gcdPolynomial, 1168
hash, 1168
latex, 1168
lcm, 1168
leftUnitary, 1168
noZeroDivisors, 1168
one?, 1168
principalIdeal, 1168
recip, 1168
rightUnitary, 1168
sample, 1168
subtractIfCan, 1168
unit?, 1168
unitCanonical, 1168
unitNormal, 1168
unitsKnown, 1168
zero?, 1168
PLACESC, 340
-?, 341
?*?, 341
?+?, 341
?-?, 341
?.?, 341
?=?, 341
?~=?, 341
coerce, 341
create, 341
degree, 341
foundPlaces, 341
hash, 341
itsALeaf, 341
latex, 341
leaf?, 341
localParam, 341
reduce, 341
setDegree, 341

setFoundPlacesToEmpty, 341
setParam, 341
PlacesCategory, 340
plenaryPower
FINAALG, 1022
FRNAALG, 1118
NAALG, 982
PlottablePlaneCurveCategory, 98
PlottableSpaceCurveCategory, 102
point
PTCAT, 896
SPACEC, 258
point?
SPACEC, 258
PointCategory, 893
pointV
INFCLCT, 1366
pointValue
AFSPCAT, 273
PRSPCAT, 346
polarCoordinates
COMPCAT, 1625
pole?
LOCPOWC, 1243
MTSCAT, 1196
PSCAT, 1161
ULSCAT, 1452
ULSCCAT, 1559
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
POLYCAT, 1255
-?, 1260
?<?, 1260
?<=?, 1260
?>?, 1260
?>=?, 1260
?**?, 1260
?*?, 1260
?+?, 1260
?-?, 1260
?/?, 1260
?=?, 1260
?^?, 1260
?~=?, 1260
0, 1260
1, 1260
associates?, 1260
binomThmExpt, 1260
canonicalUnitNormal, 1260
characteristic, 1260
charthRoot, 1260
coefficient, 1260
coefficients, 1260
coerce, 1260
commutative("*"), 1260
conditionP, 1260
content, 1260
convert, 1260
D, 1260
degree, 1260
differentiate, 1260
discriminant, 1260
eval, 1260
exquo, 1260
factor, 1260
factorPolynomial, 1260
factorSquareFreePolynomial, 1260
gcd, 1260
gcdPolynomial, 1260
ground, 1260
ground?, 1260
hash, 1260
isExpt, 1260
isPlus, 1260
isTimes, 1260
latex, 1260
lcm, 1260
leadingCoefficient, 1260
leadingMonomial, 1260
leftUnitary, 1260
mainVariable, 1260
map, 1260
mapExponents, 1260
max, 1260
min, 1260
minimumDegree, 1260
monicDivide, 1260
monomial, 1260
monomial?, 1260
monomials, 1260

multivariate, 1260
noZeroDivisors, 1260
numberOfMonomials, 1260
one?, 1260
patternMatch, 1260
pomopo, 1260
prime?, 1260
primitiveMonomials, 1260
primitivePart, 1260
recip, 1260
reducedSystem, 1260
reductum, 1260
resultant, 1260
retract, 1260
retractIfCan, 1260
rightUnitary, 1261
sample, 1260
solveLinearPolynomialEquation, 1260
squareFree, 1260
squareFreePart, 1260
squareFreePolynomial, 1260
subtractIfCan, 1260
totalDegree, 1260
unit?, 1260
unitCanonical, 1260
unitNormal, 1260
unitsKnown, 1260
univariate, 1260
variables, 1260
zero?, 1260
polygamma
SPFCAT, 60
polygon
SPACEC, 258
polygon?
SPACEC, 258
polynomial
MTSCAT, 1196
UTSCAT, 1281
PolynomialCategory, 1255
PolynomialFactorizationExplicit, 1202
PolynomialSetCategory, 465
posExpnPart
LOCPOWC, 1243
position
A1AGG, 687
ALAGG, 824
BTAGG, 734
CACHESET, 284
FLAGG, 558
LSAGG, 663
PTCAT, 896
SRAGG, 789
STRICAT, 923
VECTCAT, 811
positive?
DIRPCAT, 1004
FPS, 1532
INS, 1231
INTCAT, 1153
OINTDOM, 1137
ORDRING, 881
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
RRCC, 231
ULSCCAT, 1559
positiveRemainder
INS, 1231
possiblyInfinite?
ALAGG, 824
LSAGG, 663
LZSTAGG, 640
STAGG, 588
powerAssociative?
FINAALG, 1022
FRNAALG, 1118
PowerSeriesCategory, 1159
powmod
INS, 1231
PPCURVE, 98
coerce, 99
listBranches, 99
xRange, 99
yRange, 99
pquo
RPOLCAT, 1414
precision
FPS, 1532
prefix?
SRAGG, 789
STRICAT, 923

prem
RPOLCAT, 1414
previous
DLAGG, 394
previousTower
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PRIMCAT, 50
integral, 51
prime?
ACF, 1298
ACFS, 1500
COMPCAT, 1625
DPOLCAT, 1310
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FPC, 1325
FPS, 1532
FS, 1341
INS, 1231
LOCPOWC, 1243
MONOGEN, 1602
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PFECAT, 1204
POLYCAT, 1260
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
RPOLCAT, 1414
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPXSCAT, 1464
UPXSCCA, 1574
XF, 1513
primeFrbenius
XF, 1513
primeFrobenius
COMPCAT, 1625
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FPC, 1325
MONOGEN, 1602
PACEXTC, 1668
PACFFC, 1548
PACRATC, 1613
primitive?
COMPCAT, 1625
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
MONOGEN, 1602
PACFFC, 1548
primitiveElement
COMPCAT, 1625
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
MONOGEN, 1602
PACFFC, 1548
PrimitiveFunctionCategory, 50
primitiveMonomials
DPOLCAT, 1310
POLYCAT, 1260
RPOLCAT, 1414
UPOLYC, 1477
primitivePart
DPOLCAT, 1310
FAMR, 1143
FFCAT, 1647
LODOCAT, 973
OREPCAT, 931
POLYCAT, 1260
RPOLCAT, 1414
UPOLYC, 1477
primPartElseUnitCanonical
RPOLCAT, 1414
principal?
FDIVCAT, 618
principalIdeal
ACF, 1298
ACFS, 1500
COMPCAT, 1625

EUCDOM, 1187
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FPC, 1325
FPS, 1532
FS, 1341
INS, 1231
LOCPOWC, 1243
MONOGEN, 1602
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PID, 1168
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPXSCAT, 1464
UPXSCCA, 1574
XF, 1513
PrincipalIdealDomain, 1166
printInfo
LOCPOWC, 1243
PriorityQueueAggregate, 480
PROAGG
shallowlyMutable, 482
product
COMBOPC, 74
GRALG, 314
projectivePoint
PRSPCAT, 346
ProjectiveSpaceCategory, 345
PRQAGG, 480
?=?, 482
?~=?, 482
#?, 482
any?, 482
bag, 482
coerce, 482
copy, 482
count, 482
empty, 482
empty?, 482
eq?, 482
eval, 482
every?, 482
extract, 482
finiteAggregate, 482
hash, 482
insert, 482
inspect, 482
latex, 482
less?, 482
map, 482
max, 482
member?, 482
members, 482
merge, 482
more?, 482
parts, 482
sample, 482
size?, 482
PRSPCAT, 345
?.?, 346
?=?, 346
? =?, 346
coerce, 346
conjugate, 346
definingField, 346
degree, 346
hash, 346
homogenize, 346
lastNonNul, 346
lastNonNull, 346
latex, 346
list, 346
orbit, 346
pointValue, 346
projectivePoint, 346
rational?, 346
removeConjugate, 346
setelt, 346
PSCAT, 1159
-?, 1161
?**?, 1161
?*?, 1161

?+?, 1161
?-?, 1161
?/?, 1161
?=?, 1161
?^?, 1161
?~=?, 1161
0, 1161
1, 1161
associates?, 1161
characteristic, 1161
charthRoot, 1161
coefficient, 1161
coerce, 1161
commutative("*"), 1161
complete, 1161
degree, 1161
exquo, 1161
hash, 1161
latex, 1161
leadingCoefficient, 1161
leadingMonomial, 1161
leftUnitary, 1161
map, 1161
monomial, 1161
monomial?, 1161
noZeroDivisors, 1161
one?, 1161
pole?, 1161
recip, 1161
reductum, 1161
rightUnitary, 1161
sample, 1161
subtractIfCan, 1161
unit?, 1161
unitCanonical, 1161
unitNormal, 1161
unitsKnown, 1161
variables, 1161
zero?, 1161
PSCURVE, 102
coerce, 103
listBranches, 103
xRange, 103
yRange, 103
zRange, 103
PSETCAT, 465
?=?, 468
?~=?, 468
#?, 468
any?, 468
coerce, 468
collect, 468
collectUnder, 468
collectUpper, 468
construct, 468
convert, 468
copy, 468
count, 468
empty, 468
empty?, 468
eq?, 468
eval, 468
every?, 468
find, 468
finiteAggregate, 468
hash, 468
headRemainder, 468
latex, 468
less?, 468
mainVariable?, 468
mainVariables, 468
map, 468
member?, 468
members, 468
more?, 468
mvar, 468
parts, 468
reduce, 468
remainder, 468
remove, 468
removeDuplicates, 468
retract, 468
retractIfCan, 468
rewriteIdealWithHeadRemainder, 468
rewriteIdealWithRemainder, 468
roughBase?, 468
roughEqualIdeals?, 468
roughSubIdeal?, 468
roughUnitIdeal?, 468
sample, 468
select, 468
size?, 468

sort, 468
triangular?, 468
trivialIdeal?, 468
variables, 468
PseudoAlgebraicClosureOfAlgExtOfRationalNumberCategory, 1665
PseudoAlgebraicClosureOfFiniteFieldCategory, 1546
PseudoAlgebraicClosureOfPerfectFieldCategory, 1369
PseudoAlgebraicClosureOfRationalNumberCategory, 1610
pseudoDivide
RPOLCAT, 1414
UPOLYC, 1477
pseudoQuotient
UPOLYC, 1477
pseudoRemainder
UPOLYC, 1477
PTCAT, 893
-?, 896
?<?, 896
?<=?, 896
?>?, 896
?>=?, 896
?*?, 896
?+?, 896
?-?, 896
?.?, 896
?=?, 896
?~=?, 896
#?, 896
any?, 896
coerce, 896
concat, 896
construct, 896
convert, 896
copy, 896
copyInto, 896
count, 896
cross, 896
delete, 896
dimension, 896
dot, 896
elt, 896
empty, 896
empty?, 896
entries, 896
entry?, 896
eq?, 896
eval, 896
every?, 896
extend, 896
fill, 896
find, 896
finiteAggregate, 897
first, 896
hash, 896
index?, 896
indices, 896
insert, 896
latex, 896
length, 896
less?, 896
magnitude, 896
map, 896
max, 896
maxIndex, 896
member?, 896
members, 896
merge, 896
min, 896
minIndex, 896
more?, 896
new, 896
outerProduct, 896
parts, 896
point, 896
position, 896
qelt, 896
qsetelt, 896
reduce, 896
remove, 896
removeDuplicates, 896
reverse, 896
sample, 896
select, 896
setelt, 896
shallowlyMutable, 897
size?, 896
sort, 896
sorted?, 896

swap, 896
zero, 896
PTRANFN, 42
acoshIfCan, 43
acosIfCan, 43
acothIfCan, 43
acotIfCan, 43
acschIfCan, 43
acscIfCan, 43
asechIfCan, 43
asecIfCan, 43
asinhIfCan, 43
asinIfCan, 43
atanhIfCan, 43
atanIfCan, 43
coshIfCan, 43
cosIfCan, 43
cothIfCan, 43
cotIfCan, 43
cschIfCan, 43
cscIfCan, 43
expIfCan, 43
logIfCan, 43
nthRootIfCan, 43
sechIfCan, 43
secIfCan, 43
sinhIfCan, 43
sinIfCan, 43
tanhIfCan, 43
tanIfCan, 43
puiseux
UPXSCCA, 1574
purelyAlgebraic?
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
purelyAlgebraicLeadingMonomial?
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
purelyTranscendental?
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
qelt
A1AGG, 687
ALAGG, 824
ARR2CAT, 359
BTAGG, 734
DIRPCAT, 1004
ELAGG, 550
ELTAGG, 77
FLAGG, 558
IXAGG, 320
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
PTCAT, 896
RMATCAT, 903
SMATCAT, 1085
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
VECTCAT, 811
QFCAT, 1376
-?, 1380
?<?, 1380
?<=?, 1380
?>?, 1380
?>=?, 1380
?**?, 1380
?*?, 1380
?+?, 1380
?-?, 1380
?.?, 1380
?/?, 1380
?=?, 1380
?^?, 1380
?~=?, 1380
?quo?, 1380
?rem?, 1380
0, 1380
1, 1380
abs, 1380
associates?, 1380
canonicalsClosed, 1380

canonicalUnitNormal, 1380
ceiling, 1380
characteristic, 1380
charthRoot, 1380
coerce, 1380
commutative("*"), 1380
conditionP, 1380
convert, 1380
D, 1380
denom, 1380
denominator, 1380
differentiate, 1380
divide, 1380
euclideanSize, 1380
eval, 1380
expressIdealMember, 1380
exquo, 1380
extendedEuclidean, 1380
factor, 1380
factorPolynomial, 1380
factorSquareFreePolynomial, 1380
floor, 1380
fractionPart, 1380
gcd, 1380
gcdPolynomial, 1380
hash, 1380
init, 1380
inv, 1380
latex, 1380
lcm, 1380
leftUnitary, 1380
map, 1380
max, 1380
min, 1380
multiEuclidean, 1380
negative?, 1380
nextItem, 1380
noZeroDivisors, 1380
numer, 1380
numerator, 1380
one?, 1380
patternMatch, 1380
positive?, 1380
prime?, 1380
principalIdeal, 1380
random, 1380
recip, 1380
reducedSystem, 1380
retract, 1380
retractIfCan, 1380
rightUnitary, 1380
sample, 1380
sign, 1380
sizeLess?, 1380
solveLinearPolynomialEquation, 1380
squareFree, 1380
squareFreePart, 1380
squareFreePolynomial, 1380
subtractIfCan, 1380
unit?, 1380
unitCanonical, 1380
unitNormal, 1380
unitsKnown, 1380
wholePart, 1380
zero?, 1380
qinterval
INTCAT, 1153
QUAGG, 486
?=?, 488
?~=?, 488
#?, 488
any?, 488
back, 488
bag, 488
coerce, 488
copy, 488
count, 488
dequeue, 488
empty, 488
empty?, 488
enqueue, 488
eq?, 488
eval, 488
every?, 488
extract, 488
finiteAggregate, 488
front, 488
hash, 488
insert, 488
inspect, 488
latex, 488
length, 488

less?, 488
map, 488
member?, 488
members, 488
more?, 488
parts, 488
rotate, 488
sample, 488
shallowlyMutable, 488
size?, 488
quasiComponent
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
quasiMonic?
RPOLCAT, 1414
quasiRegular
XFALG, 994
XPOLYC, 1096
quasiRegular?
XFALG, 994
XPOLYC, 1096
QUATCAT, 1071
-?, 1073
?<?, 1073
?<=?, 1073
?>?, 1073
?>=?, 1073
?**?, 1073
?*?, 1073
?+?, 1073
?-?, 1073
?.?, 1073
?=?, 1073
?^?, 1073
?~=?, 1073
0, 1073
1, 1073
abs, 1073
characteristic, 1073
charthRoot, 1073
coerce, 1073
conjugate, 1073
convert, 1073
D, 1073
differentiate, 1073
eval, 1073
hash, 1073
imagI, 1073
imagJ, 1073
imagK, 1073
inv, 1073
latex, 1073
leftUnitary, 1074
map, 1073
max, 1073
min, 1073
norm, 1073
noZeroDivisors, 1073
one?, 1073
quatern, 1073
rational, 1073
rational?, 1073
rationalIfCan, 1073
real, 1073
recip, 1073
reducedSystem, 1073
retract, 1073
retractIfCan, 1073
rightUnitary, 1074
sample, 1073
subtractIfCan, 1073
unitsKnown, 1073
zero?, 1073
quatern
QUATCAT, 1073
QuaternionCategory, 1071
QueueAggregate, 486
quoByVar
UTSCAT, 1281
quotientByP
PADICCT, 1251
QuotientFieldCategory, 1376
quotValuation
BLMETCT, 128

RADCAT, 52
?**?, 53
nthRoot, 53
sqrt, 53

RadicalCategory, 52
ramified?
    FFCAT, 1647
ramifiedAtInfinity?
    FFCAT, 1647
ramifMult
    BLMETCT, 128
random
    COMPCAT, 1625
    DIRPCAT, 1004
    FAXF, 1587
    FFCAT, 1647
    FFIELDC, 1521
    FINITE, 169
    FSAGG, 624
    INS, 1231
    MONOGEN, 1602
    OC, 1062
    ORDFIN, 337
    PACFFC, 1548
    QFCAT, 1380
    ULSCCAT, 1559
rank
    COMPCAT, 1625
    FFCAT, 1647
    FINAALG, 1022
    FINRALG, 1331
    FRAMALG, 1541
    FRNAALG, 1118
    MATCAT, 437
    MONOGEN, 1602
    RMATCAT, 903
    SMATCAT, 1085
rational
    COMPCAT, 1625
    INS, 1231
    OC, 1062
    QUATCAT, 1073
rational?
    AFSPCAT, 273
    COMPCAT, 1625
    INS, 1231
    OC, 1062
    PRSPCAT, 346
    QUATCAT, 1073
rationalFunction
    ULSCAT, 1452
    ULSCCAT, 1559
rationalIfCan
    COMPCAT, 1625
    INS, 1231
    OC, 1062
    QUATCAT, 1073
rationalPoint?
    FFCAT, 1647
rationalPoints
    FFCAT, 1647
rationalPower
    UPXSCCA, 1574
RCAGG, 351
    ?.value, 353
    ?=?, 353
    ?~=?, 353
    #?, 353
    any?, 353
    child?, 353
    children, 353
    coerce, 353
    copy, 353
    count, 353
    cyclic?, 353
    distance, 353
    empty, 353
    empty?, 353
    eq?, 353
    eval, 353
    every?, 353
    hash, 353
    latex, 353
    leaf?, 353
    leaves, 353
    less?, 353
    map, 353
    member?, 353
    members, 353
    more?, 353
    node?, 353
    nodes, 353
    parts, 353
    sample, 353
    setchildren, 353
    setelt, 353

setvalue, 353
shallowlyMutable, 353
size?, 353
value, 353
RCFIELD, 1390
-?, 1392
?<?, 1392
?<=?, 1392
?>?, 1392
?>=?, 1392
?**?, 1392
?*?, 1392
?+?, 1392
?-?, 1392
?/?, 1392
?=?, 1392
?^?, 1392
?~=?, 1392
?quo?, 1392
?rem?, 1392
0, 1392
1, 1392
abs, 1392
allRootsOf, 1392
approximate, 1392
associates?, 1392
canonicalsClosed, 1393
canonicalUnitNormal, 1392
characteristic, 1392
coerce, 1392
commutative("*"), 1393
divide, 1392
euclideanSize, 1392
expressIdealMember, 1392
exquo, 1392
extendedEuclidean, 1392
factor, 1392
gcd, 1392
gcdPolynomial, 1392
hash, 1392
inv, 1392
latex, 1392
lcm, 1392
leftUnitary, 1393
mainDefiningPolynomial, 1392
mainForm, 1392
mainValue, 1392
max, 1392
min, 1392
multiEuclidean, 1392
negative?, 1392
noZeroDivisors, 1392
nthRoot, 1392
one?, 1392
positive?, 1392
prime?, 1392
principalIdeal, 1392
recip, 1392
rename, 1392
retract, 1392
retractIfCan, 1392
rightUnitary, 1393
rootOf, 1392
sample, 1392
sign, 1392
sizeLess?, 1392
sqrt, 1392
squareFree, 1392
squareFreePart, 1392
subtractIfCan, 1392
unit?, 1392
unitCanonical, 1392
unitNormal, 1392
unitsKnown, 1393
zero?, 1392
readable?
FNCAT, 173
REAL, 106
convert, 107
real
COMPCAT, 1625
OC, 1062
QUATCAT, 1073
RealClosedField, 1390
RealConstant, 106
RealNumberSystem, 1400
RealRootCharacterizationCategory, 230
recip
ACF, 1298
ACFS, 1500
ALGEBRA, 949
AMR, 1103

CHARNZ, 835
CHARZ, 840
COMPCAT, 1625
COMRING, 845
DIFEXT, 955
DIFRING, 850
DIRPCAT, 1004
DIVRING, 1014
DPOLCAT, 1310
ENTIRER, 855
EUCDOM, 1187
FAMR, 1143
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FINAALG, 1022
FINRALG, 1331
FLINEXP, 962
FMTC, 1110
FPC, 1325
FPS, 1532
FRAMALG, 1541
FRNAALG, 1118
FS, 1341
GCDDOM, 1131
GROUP, 400
INS, 1231
INTCAT, 1153
INTDOM, 1049
LALG, 865
LINEXP, 871
LOCPOWC, 1243
LODOCAT, 973
MLO, 1055
MONADWU, 327
MONOGEN, 1602
MONOID, 332
MTSCAT, 1196
NASRING, 742
OC, 1062
OINTDOM, 1137
ORDMON, 462
ORDRING, 881
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PDRING, 887
PERMCAT, 582
PFECAT, 1204
PID, 1168
POLYCAT, 1260
PSCAT, 1161
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1392
RING, 772
RNS, 1403
RPOLCAT, 1414
RRCC, 231
SMATCAT, 1085
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
XALG, 942
XF, 1513
XFALG, 994
XPOLYC, 1096
RectangularMatrixCategory, 901
RecursiveAggregate, 351
RecursivePolynomialCategory, 1408
reduce
A1AGG, 687
ALAGG, 824
BTAGG, 734
CLAGG, 289
COMPCAT, 1625
DIAGG, 535
DIOPS, 387
ELAGG, 550
FDIVCAT, 618
FFCAT, 1647
FLAGG, 558
FSAGG, 624

KDAGG, 632
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MDAGG, 572
MONOGEN, 1602
MSETAGG, 675
NTSCAT, 749
OMSAGG, 765
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PLACESC, 341
PSETCAT, 468
PTCAT, 896
RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
VECTCAT, 811
reduceBasisAtInfinity
FFCAT, 1647
reduceByQuasiMonic
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
reduced?
NTSCAT, 749
RPOLCAT, 1414
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
reducedSystem
ACFS, 1500
COMPCAT, 1625
DIRPCAT, 1004
DPOLCAT, 1310
FFCAT, 1647
FLINEXP, 962
FS, 1341
INS, 1231
LINEXP, 871
MONOGEN, 1602
POLYCAT, 1260
QFCAT, 1380
QUATCAT, 1073
RPOLCAT, 1414
SMATCAT, 1085
ULSCCAT, 1559
UPOLYC, 1477
reductum
AMR, 1103
DPOLCAT, 1310
FAMR, 1143
FMCAT, 859
IDPC, 189
LOCPOWC, 1243
LODOCAT, 973
MLO, 1055
MTSCAT, 1196
OREPCAT, 931
POLYCAT, 1260
PSCAT, 1161
RPOLCAT, 1414
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
regularRepresentation
COMPCAT, 1625
FFCAT, 1647
FINRALG, 1331
FRAMALG, 1541
MONOGEN, 1602
RegularTriangularSetCategory, 700
relativeApprox
RRCC, 231
remainder
NTSCAT, 749
PSETCAT, 468

RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
remove
A1AGG, 687
ALAGG, 824
BTAGG, 734
CLAGG, 289
DIAGG, 535
DIOPS, 387
ELAGG, 550
FLAGG, 558
FSAGG, 624
KDAGG, 632
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MDAGG, 572
MSETAGG, 675
NTSCAT, 749
OMSAGG, 765
PSETCAT, 468
PTCAT, 896
RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
VECTCAT, 811
removeConjugate
AFSPCAT, 273
PRSPCAT, 346
removeDuplicates
A1AGG, 687
ALAGG, 824
BTAGG, 734
CLAGG, 289
DIAGG, 535
DIOPS, 387
ELAGG, 550
FLAGG, 558
FSAGG, 624
KDAGG, 632
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MDAGG, 572
MSETAGG, 675
NTSCAT, 749
OMSAGG, 765
PSETCAT, 468
PTCAT, 896
RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
VECTCAT, 811
removeFirstZeroes
LOCPOWC, 1243
removeZero
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
removeZeroes
LOCPOWC, 1243
ULSCCAT, 1559
rename
RCFIELD, 1392
replace
SRAGG, 789
STRICAT, 923
representationType
COMPCAT, 1625
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
MONOGEN, 1602
PACFFC, 1548
represents
COMPCAT, 1625

FAXF, 1587
FFCAT, 1647
FINAALG, 1022
FINRALG, 1331
FRAMALG, 1541
FRNAALG, 1118
MONOGEN, 1602
rest
ALAGG, 824
LSAGG, 663
LZSTAGG, 640
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
STAGG, 588
TSETCAT, 598
URAGG, 509
resultant
DPOLCAT, 1310
POLYCAT, 1260
RPOLCAT, 1414
UPOLYC, 1477
RETRACT, 55
coerce, 57
retract, 57
retractIfCan, 57
retract
ACFS, 1500
COMPCAT, 1625
DIRPCAT, 1004
DIVCAT, 1181
DPOLCAT, 1310
DVARCAT, 296
ES, 302
FAMONC, 565
FAMR, 1143
FAXF, 1587
FFCAT, 1647
FMCAT, 859
FMFUN, 146
FMTC, 1110
FORTFN, 137
FPS, 1532
FRETRCT, 89
FS, 1341
FVFUN, 155
GRALG, 314
INS, 1231
INTCAT, 1153
LODOCAT, 973
MONOGEN, 1602
NTSCAT, 749
OC, 1062
OREPCAT, 931
PACEXTC, 1668
PACRATC, 1613
POLYCAT, 1260
PSETCAT, 468
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1392
RETRACT, 57
RNS, 1403
RPOLCAT, 1414
RSETCAT, 703
SFRTCAT, 779
SMATCAT, 1085
SNTSCAT, 914
TSETCAT, 598
ULSCCAT, 1559
UPOLYC, 1477
UPXSCCA, 1574
XF, 1513
XFALG, 994
XPOLYC, 1096
RetractableTo, 55
retractIfCan
ACFS, 1500
COMPCAT, 1625
DIRPCAT, 1004
DIVCAT, 1181
DPOLCAT, 1310
DVARCAT, 296
ES, 302
FAMONC, 565
FAMR, 1143
FAXF, 1587
FFCAT, 1647
FMCAT, 859
FMFUN, 146
FMTC, 1110

FORTFN, 137
FPS, 1532
FRETRCT, 89
FS, 1341
FVFUN, 155
GRALG, 314
INS, 1231
INTCAT, 1153
LODOCAT, 973
MONOGEN, 1602
NTSCAT, 749
OC, 1062
OREPCAT, 931
PACEXTC, 1668
PACRATC, 1613
POLYCAT, 1260
PSETCAT, 468
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1392
RETRACT, 57
RNS, 1403
RPOLCAT, 1414
RSETCAT, 703
SFRTCAT, 779
SMATCAT, 1085
SNTSCAT, 914
TSETCAT, 598
ULSCCAT, 1559
UPOLYC, 1477
UPXSCCA, 1574
XF, 1513
XFALG, 994
XPOLYC, 1096
reverse
A1AGG, 687
ALAGG, 824
BTAGG, 734
FLAGG, 558
LSAGG, 663
PTCAT, 896
SRAGG, 789
STRICAT, 923
VECTCAT, 811
rewriteIdealWithHeadRemainder
NTSCAT, 749
PSETCAT, 468
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
rewriteIdealWithRemainder
NTSCAT, 749
PSETCAT, 468
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
rewriteSetWithReduction
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
right
BRAGG, 374
BTCAT, 528
rightAlternative?
FINAALG, 1022
FRNAALG, 1118
rightCharacteristicPolynomial
FINAALG, 1022
FRNAALG, 1118
rightDiscriminant
FINAALG, 1022
FRNAALG, 1118
rightDivide
LODOCAT, 973
OREPCAT, 931
rightExactQuotient
LODOCAT, 973
OREPCAT, 931
rightExtendedGcd
LODOCAT, 973
OREPCAT, 931
rightGcd
LODOCAT, 973
OREPCAT, 931
rightLcm
LODOCAT, 973
OREPCAT, 931
rightMinimalPolynomial

FINAALG, 1022
FRNAALG, 1118
RightModule, 715
rightNorm
FINAALG, 1022
FRNAALG, 1118
rightPower
FINAALG, 1022
FRNAALG, 1118
MONAD, 198
MONADWU, 327
NAALG, 982
NARNG, 681
NASRING, 742
rightQuotient
LODOCAT, 973
OREPCAT, 931
rightRankPolynomial
FRNAALG, 1118
rightRecip
FINAALG, 1022
FRNAALG, 1118
MONADWU, 327
NASRING, 742
rightRegularRepresentation
FINAALG, 1022
FRNAALG, 1118
rightRemainder
LODOCAT, 973
OREPCAT, 931
rightTrace
FINAALG, 1022
FRNAALG, 1118
rightTraceMatrix
FINAALG, 1022
FRNAALG, 1118
rightTrim
SRAGG, 789
STRICAT, 923
rightUnit
FINAALG, 1022
FRNAALG, 1118
rightUnitary
ACF, 1299
ACFS, 1500
ALGEBRA, 949
AMR, 1103
ATTREG, 11
BMODULE, 728
COMPCAT, 1626
COMRING, 845
DIRPCAT, 1005
DIVRING, 1015
DPOLCAT, 1311
ENTIRER, 855
EUCDOM, 1187
FAMR, 1144
FAXF, 1588
FFCAT, 1648
FFIELDC, 1521
FIELD, 1224
FINAALG, 1022
FINRALG, 1331
FLALG, 1043
FMCAT, 860
FMTC, 1110
FPC, 1326
FPS, 1533
FRAMALG, 1541
FRNAALG, 1118
FS, 1342
GCDDOM, 1131
INS, 1231
INTCAT, 1153
INTDOM, 1049
LIECAT, 968
LOCPOWC, 1243
LODOCAT, 974
MLO, 1055
MODULE, 876
MONOGEN, 1603
MTSCAT, 1196
NAALG, 982
OC, 1062
OINTDOM, 1137
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1549
PACPERC, 1372
PACRATC, 1614
PADICCT, 1251
PFECAT, 1204

PID, 1168
POLYCAT, 1261
PSCAT, 1161
QFCAT, 1380
QUATCAT, 1074
RCFIELD, 1393
RMATCAT, 904
RPOLCAT, 1415
SMATCAT, 1085
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1560
UPOLYC, 1478
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
VSPACE, 989
XALG, 942
XF, 1513
XFALG, 994
XPOLYC, 1096
rightUnits
FINAALG, 1022
FRNAALG, 1118
RING, 770
-?, 772
?**?, 772
?*?, 772
?+?, 772
?-?, 772
?=?, 772
?^?, 772
?~=?, 772
0, 772
1, 772
characteristic, 772
coerce, 772
hash, 772
latex, 772
one?, 772
recip, 772
sample, 772
subtractIfCan, 772
unitsKnown, 772
zero?, 772
Ring, 770
RittWuCompare
RPOLCAT, 1414
RMATCAT, 901
-?, 903
?*?, 903
?+?, 903
?-?, 903
?/?, 903
?=?, 903
?~=?, 903
#?, 903
0, 903
antisymmetric?, 903
any?, 903
coerce, 903
column, 903
copy, 903
count, 903
diagonal?, 903
elt, 903
empty, 903
empty?, 903
eq?, 903
eval, 903
every?, 903
exquo, 903
finiteAggregate, 903
hash, 903
latex, 903
leftUnitary, 903
less?, 903
listOfLists, 903
map, 903
matrix, 903
maxColIndex, 903
maxRowIndex, 903
member?, 903
members, 903
minColIndex, 903
minRowIndex, 903
more?, 903
ncols, 903
nrows, 903
nullity, 903
nullSpace, 903

parts, 903
qelt, 903
rank, 903
rightUnitary, 904
row, 903
rowEchelon, 903
sample, 903
size?, 903
square?, 903
subtractIfCan, 903
symmetric?, 903
zero?, 903
RMODULE, 715
-?, 716
?*?, 716
?+?, 716
?-?, 716
?=?, 716
?~=?, 716
0, 716
coerce, 716
hash, 716
latex, 716
sample, 716
subtractIfCan, 716
zero?, 716
RNG, 719
-?, 720
?**?, 720
?*?, 720
?+?, 720
?-?, 720
?=?, 720
?^?, 720
?~=?, 720
0, 720
coerce, 720
hash, 720
latex, 720
sample, 720
subtractIfCan, 720
zero?, 720
Rng, 719
RNS, 1400
-?, 1403
?<?, 1403
?<=?, 1403
?>?, 1403
?>=?, 1403
?**?, 1403
?*?, 1403
?+?, 1403
?-?, 1403
?/?, 1403
?=?, 1403
?^?, 1403
?~=?, 1403
?quo?, 1403
?rem?, 1403
0, 1403
1, 1403
abs, 1403
associates?, 1403
ceiling, 1403
characteristic, 1403
coerce, 1403
convert, 1403
divide, 1403
euclideanSize, 1403
expressIdealMember, 1403
exquo, 1403
extendedEuclidean, 1403
factor, 1403
floor, 1403
fractionPart, 1403
gcd, 1403
gcdPolynomial, 1403
hash, 1403
inv, 1403
latex, 1403
lcm, 1403
max, 1403
min, 1403
multiEuclidean, 1403
negative?, 1403
norm, 1403
nthRoot, 1403
one?, 1403
patternMatch, 1403
positive?, 1403
prime?, 1403
principalIdeal, 1403

recip, 1403
retract, 1403
retractIfCan, 1403
round, 1403
sample, 1403
sign, 1403
sizeLess?, 1403
sqrt, 1403
squareFree, 1403
squareFreePart, 1403
subtractIfCan, 1403
truncate, 1403
unit?, 1403
unitCanonical, 1403
unitNormal, 1403
wholePart, 1403
zero?, 1403
root
PADICCT, 1251
rootOf
ACF, 1298
ACFS, 1500
RCFIELD, 1392
RRCC, 231
rootsOf
ACF, 1298
ACFS, 1500
roughBase?
NTSCAT, 749
PSETCAT, 468
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
roughEqualIdeals?
NTSCAT, 749
PSETCAT, 468
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
roughSubIdeal?
NTSCAT, 749
PSETCAT, 468
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
roughUnitIdeal?
NTSCAT, 749
PSETCAT, 468
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
round
FPS, 1532
RNS, 1403
row
ARR2CAT, 359
MATCAT, 437
RMATCAT, 903
SMATCAT, 1085
rowEchelon
MATCAT, 437
RMATCAT, 903
SMATCAT, 1085
RPOLCAT, 1408
-?, 1414
?<?, 1414
?<=?, 1414
?>?, 1414
?>=?, 1414
?**?, 1414
?*?, 1414
?+?, 1414
?-?, 1414
?/?, 1414
?=?, 1414
?^?, 1414
?~=?, 1414
0, 1414
1, 1414
associates?, 1414
binomThmExpt, 1414
canonicalUnitNormal, 1415
characteristic, 1414
charthRoot, 1414
coefficient, 1414
coefficients, 1414
coerce, 1414
commutative("*"), 1415

conditionP, 1414
convert, 1414
D, 1414
deepestInitial, 1414
deepestTail, 1414
degree, 1414
differentiate, 1414
discriminant, 1414
eval, 1414
exactQuotient, 1414
exquo, 1414
extendedSubResultantGcd, 1414
factor, 1414
factorPolynomial, 1414
factorSquareFreePolynomial, 1414
gcd, 1414
gcdPolynomial, 1414
ground, 1414
ground?, 1414
halfExtendedSubResultantGcd1, 1414
halfExtendedSubResultantGcd2, 1414
hash, 1414
head, 1414
headReduce, 1414
headReduced?, 1414
infRittWu?, 1414
init, 1414
initiallyReduce, 1414
initiallyReduced?, 1414
isExpt, 1414
isPlus, 1414
isTimes, 1414
iteratedInitials, 1414
lastSubResultant, 1414
latex, 1414
LazardQuotient, 1414
LazardQuotient2, 1414
lazyPquo, 1414
lazyPrem, 1414
lazyPremWithDefault, 1414
lazyPseudoDivide, 1414
lazyResidueClass, 1414
lcm, 1414
leadingCoefficient, 1414
leadingMonomial, 1414
leastMonomial, 1414
leftUnitary, 1415
mainCoefficients, 1414
mainContent, 1414
mainMonomial, 1414
mainPrimitivePart, 1414
mainSquareFreePart, 1414
mainVariable, 1414
map, 1414
mapExponents, 1414
max, 1414
mdeg, 1414
min, 1414
minimumDegree, 1414
monic?, 1414
monicDivide, 1414
monicModulo, 1414
monomial, 1414
monomial?, 1414
monomials, 1414
multivariate, 1414
mvar, 1414
nextsubResultant2, 1414
normalized?, 1414
noZeroDivisors, 1415
numberOfMonomials, 1414
one?, 1414
patternMatch, 1414
pomopo, 1414
pquo, 1414
prem, 1414
prime?, 1414
primitiveMonomials, 1414
primitivePart, 1414
primPartElseUnitCanonical, 1414
pseudoDivide, 1414
quasiMonic?, 1414
recip, 1414
reduced?, 1414
reducedSystem, 1414
reductum, 1414
resultant, 1414
retract, 1414
retractIfCan, 1414
rightUnitary, 1415
RittWuCompare, 1414
sample, 1414

solveLinearPolynomialEquation, 1414
squareFree, 1414
squareFreePart, 1414
squareFreePolynomial, 1414
subResultantChain, 1414
subResultantGcd, 1414
subtractIfCan, 1414
supRittWu?, 1414
tail, 1414
totalDegree, 1414
unit?, 1414
unitCanonical, 1414
unitNormal, 1414
unitsKnown, 1415
univariate, 1414
variables, 1414
zero?, 1414
rquo
FLALG, 1043
XFALG, 994
XPOLYC, 1096
RRCC, 230
?=?, 231
? =?, 231
allRootsOf, 231
approximate, 231
coerce, 231
definingPolynomial, 231
hash, 231
latex, 231
negative?, 231
positive?, 231
recip, 231
relativeApprox, 231
rootOf, 231
sign, 231
zero?, 231
RSETCAT, 700
?=?, 703
?~=?, 703
#?, 703
algebraicCoefficients?, 703
algebraicVariables, 703
any?, 703
augment, 703
autoReduced?, 703
basicSet, 703
coerce, 703
coHeight, 703
collect, 703
collectQuasiMonic, 703
collectUnder, 703
collectUpper, 703
construct, 703
convert, 703
copy, 703
count, 703
degree, 703
empty, 703
empty?, 703
eq?, 703
eval, 703
every?, 703
extend, 703
extendIfCan, 703
find, 703
finiteAggregate, 703
first, 703
hash, 703
headReduce, 703
headReduced?, 703
headRemainder, 703
infRittWu?, 703
initiallyReduce, 703
initiallyReduced?, 703
initials, 703
internalAugment, 703
intersect, 703
invertible?, 703
invertibleElseSplit?, 703
invertibleSet, 703
last, 703
lastSubResultant, 703
lastSubResultantElseSplit, 703
latex, 703
less?, 703
mainVariable?, 703
mainVariables, 703
map, 703
member?, 703
members, 703
more?, 703

mvar, 703
normalized?, 703
parts, 703
purelyAlgebraic?, 703
purelyAlgebraicLeadingMonomial?, 703
purelyTranscendental?, 703
quasiComponent, 703
reduce, 703
reduceByQuasiMonic, 703
reduced?, 703
remainder, 703
remove, 703
removeDuplicates, 703
removeZero, 703
rest, 703
retract, 703
retractIfCan, 703
rewriteIdealWithHeadRemainder, 703
rewriteIdealWithRemainder, 703
rewriteSetWithReduction, 703
roughBase?, 703
roughEqualIdeals?, 703
roughSubIdeal?, 703
roughUnitIdeal?, 703
sample, 703
select, 703
shallowlyMutable, 703
size?, 703
sort, 703
squareFreePart, 703
stronglyReduce, 703
stronglyReduced?, 703
triangular?, 703
trivialIdeal?, 703
variables, 703
zeroSetSplit, 703
zeroSetSplitIntoTriangularSystems, 703

rst
LZSTAGG, 640

sample
A1AGG, 687
ABELGRP, 522
ABELMON, 269
ACF, 1298
ACFS, 1500
AGG, 70
ALAGG, 824
ALGEBRA, 949
AMR, 1103
ARR2CAT, 359
BGAGG, 279
BMODULE, 728
BRAGG, 374
BTAGG, 734
BTCAT, 528
CABMON, 381
CHARNZ, 835
CHARZ, 840
CLAGG, 289
COMPCAT, 1625
COMRING, 845
DIAGG, 535
DIFEXT, 955
DIFRING, 850
DIOPS, 387
DIRPCAT, 1004
DIVCAT, 1181
DIVRING, 1014
DLAGG, 394
DPOLCAT, 1310
DQAGG, 542
DSTRCAT, 133
ELAGG, 550
ENTIRER, 855
EUCDOM, 1187
FAMONC, 565
FAMR, 1143
FAXF, 1587
FDIVCAT, 618
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FINAALG, 1022
FINRALG, 1331
FLAGG, 558
FLALG, 1043
FLINEXP, 962
FMCAT, 859
FMTC, 1110
FPC, 1325
FPS, 1532

FRAMALG, 1541
FRNAALG, 1118
FS, 1341
FSAGG, 624
GCDDOM, 1131
GROUP, 400
HOAGG, 183
INS, 1231
INTCAT, 1153
INTDOM, 1049
IXAGG, 320
KDAGG, 632
LALG, 865
LIECAT, 968
LINEXP, 871
LMODULE, 657
LNAGG, 407
LOCPOWC, 1243
LODOCAT, 973
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
MDAGG, 572
MLO, 1055
MODULE, 876
MONOGEN, 1602
MONOID, 332
MSETAGG, 675
MTSCAT, 1196
NAALG, 982
NARNG, 681
NASRING, 742
NTSCAT, 749
OAGROUP, 756
OAMON, 577
OAMONS, 760
OASGP, 458
OC, 1062
OCAMON, 697
OINTDOM, 1137
OMSAGG, 765
ORDMON, 462
ORDRING, 881
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PDRING, 887
PERMCAT, 582
PFECAT, 1204
PID, 1168
POLYCAT, 1260
PRQAGG, 482
PSCAT, 1161
PSETCAT, 468
PTCAT, 896
QFCAT, 1380
QUAGG, 488
QUATCAT, 1073
RCAGG, 353
RCFIELD, 1392
RING, 772
RMATCAT, 903
RMODULE, 716
RNG, 720
RNS, 1403
RPOLCAT, 1414
RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SKAGG, 502
SMATCAT, 1085
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
URAGG, 509
UTSCAT, 1281
VECTCAT, 811
VSPACE, 989
XALG, 942

XF, 1513
XFALG, 994
XPOLYC, 1096
sbt
LOCPOWC, 1243
scalarMatrix
MATCAT, 437
SMATCAT, 1085
search
ALAGG, 824
KDAGG, 632
TBAGG, 800
sec
COMPCAT, 1625
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
TRIGCAT, 64
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
sech
COMPCAT, 1625
HYPCAT, 33
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
sechIfCan
PTRANFN, 43
secIfCan
PTRANFN, 43
second
ALAGG, 824
LSAGG, 663
LZSTAGG, 640
STAGG, 588
URAGG, 509
SEGCAT, 109
?..?, 110
BY, 110
convert, 110
hi, 110
high, 110
incr, 110
lo, 110
low, 110
segment, 110
segment
SEGCAT, 110
SEGXCAT, 236
SegmentCategory, 109
SegmentExpansionCategory, 235
SEGXCAT, 235
?..?, 236
BY, 236
convert, 236
expand, 236
hi, 236
high, 236
incr, 236
lo, 236
low, 236
map, 236
segment, 236
select
A1AGG, 687
ALAGG, 824
BTAGG, 734
CLAGG, 289
DIAGG, 535
DIOPS, 387
ELAGG, 550
FLAGG, 558
FSAGG, 624
KDAGG, 632
LSAGG, 663
LZSTAGG, 640
MDAGG, 572
MSETAGG, 675
NTSCAT, 749
OMSAGG, 765
PSETCAT, 468
PTCAT, 896

RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SNTSCAT, 914
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
VECTCAT, 811
SemiGroup, 239
separant
DPOLCAT, 1310
separate
UPOLYC, 1477
series
LOCPOWC, 1243
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
set
FSAGG, 624
MSETAGG, 675
OMSAGG, 765
SETAGG, 494
SETAGG, 492
?<?, 494
?=?, 494
?~=?, 494
#?, 494
any?, 494
brace, 494
coerce, 494
construct, 494
convert, 494
copy, 494
count, 494
difference, 494
empty, 494
empty?, 494
eq?, 494
eval, 494
every?, 494
find, 494
hash, 494
intersect, 494
latex, 494
less?, 494
map, 494
member?, 494
members, 494
more?, 494
partiallyOrderedSet, 494
parts, 494
reduce, 494
remove, 494
removeDuplicates, 494
sample, 494
select, 494
set, 494
size?, 494
subset?, 494
symmetricDifference, 494
union, 494
SetAggregate, 492
SETCAT, 113
?=?, 115
?~=?, 115
coerce, 115
hash, 115
latex, 115
SETCATD, 243
?=?, 244
?~=?, 244
coerce, 244
degree, 244
hash, 244
latex, 244
SetCategory, 113
SetCategoryWithDegree, 243
setelt
A1AGG, 687
AFSPCAT, 273
ALAGG, 824
ARR2CAT, 359
BRAGG, 374
BTAGG, 734
BTCAT, 528
DIRPCAT, 1004
DLAGG, 394
DSTRCAT, 133

ELAGG, 550
ELTAGG, 77
FLAGG, 558
IXAGG, 320
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
PRSPCAT, 346
PTCAT, 896
RCAGG, 353
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
URAGG, 509
VECTCAT, 811
setFoundPlacesToEmpty
PLACESC, 341
setPosition
CACHESET, 284
SEXCAT, 246
?.?, 248
?=?, 248
?~=?, 248
#?, 248
atom?, 248
car, 248
cdr, 248
coerce, 248
convert, 248
destruct, 248
eq, 248
expr, 248
float, 248
float?, 248
hash, 248
integer, 248
integer?, 248
latex, 248
list?, 248
null?, 248
pair?, 248
string, 248
string?, 248
symbol, 248
symbol?, 248
SExpressionCategory, 246
SFRTCAT, 775
?=?, 779
?~=?, 779
#?, 779
algebraic?, 779
algebraicCoefficients?, 779
algebraicVariables, 779
any?, 779
augment, 779
autoReduced?, 779
basicSet, 779
coerce, 779
coHeight, 779
collect, 779
collectQuasiMonic, 779
collectUnder, 779
collectUpper, 779
construct, 779
convert, 779
copy, 779
count, 779
degree, 779
empty, 779
empty?, 779
eq?, 779
eval, 779
every?, 779
extend, 779
extendIfCan, 779
find, 779
finiteAggregate, 780
first, 779
hash, 779
headReduce, 779
headReduced?, 779
headRemainder, 779
infRittWu?, 779
initiallyReduce, 779
initiallyReduced?, 779
initials, 779
internalAugment, 779
intersect, 779
invertible?, 779
invertibleElseSplit?, 779

invertibleSet, 779
last, 779
lastSubResultant, 779
lastSubResultantElseSplit, 779
latex, 779
less?, 779
mainVariable?, 779
mainVariables, 779
map, 779
member?, 779
members, 779
more?, 779
mvar, 779
normalized?, 779
parts, 779
purelyAlgebraic?, 779
purelyAlgebraicLeadingMonomial?, 779
purelyTranscendental?, 779
quasiComponent, 779
reduce, 779
reduceByQuasiMonic, 779
reduced?, 779
remainder, 779
remove, 779
removeDuplicates, 779
removeZero, 779
rest, 779
retract, 779
retractIfCan, 779
rewriteIdealWithHeadRemainder, 779
rewriteIdealWithRemainder, 779
rewriteSetWithReduction, 779
roughBase?, 779
roughEqualIdeals?, 779
roughSubIdeal?, 779
roughUnitIdeal?, 779
sample, 779
select, 779
shallowlyMutable, 780
size?, 779
sort, 779
squareFreePart, 779
stronglyReduce, 779
stronglyReduced?, 779
triangular?, 779
trivialIdeal?, 779
variables, 779
zeroSetSplit, 779
zeroSetSplitIntoTriangularSystems, 779
SGROUP, 239
?**?, 240
?*?, 240
?=?, 240
?^?, 240
?~=?, 240
coerce, 240
hash, 240
latex, 240
sh
XFALG, 994
XPOLYC, 1096
shallowlyMutable
A1AGG, 687
ALAGG, 824
ARR2CAT, 359
ATTREG, 11
BGAGG, 279
BRAGG, 375
BTAGG, 734
BTCAT, 529
DIAGG, 535
DIOPS, 387
DLAGG, 394
DQAGG, 542
DSTRCAT, 133
ELAGG, 550
ELTAGG, 77
FLAGG, 558
FSAGG, 624
HOAGG, 183
IXAGG, 320
KDAGG, 632
LNAGG, 407
LSAGG, 663
MATCAT, 437
MDAGG, 572
MSETAGG, 675
NTSCAT, 750
OMSAGG, 765
PROAGG, 482
PTCAT, 897
QUAGG, 488

RCAGG, 353
RSETCAT, 703
SFRTCAT, 780
SKAGG, 502
SNTSCAT, 915
SRAGG, 790
STAGG, 589
STRICAT, 923
TBAGG, 800
TSETCAT, 598
URAGG, 509
VECTCAT, 811
shift
INS, 1231
LOCPOWC, 1243
shiftLeft
UPOLYC, 1477
shiftRight
UPOLYC, 1477
Si
LFCAT, 193
sign
DIRPCAT, 1004
FPS, 1532
INS, 1231
OINTDOM, 1137
ORDRING, 881
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
RRCC, 231
ULSCCAT, 1559
sin
COMPCAT, 1625
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
TRIGCAT, 64
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
singular?
FFCAT, 1647
singularAtInfinity?
FFCAT, 1647
sinh
COMPCAT, 1625
HYPCAT, 33
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
sinhIfCan
PTRANFN, 43
sinIfCan
PTRANFN, 43
size
COMPCAT, 1625
DIRPCAT, 1004
DIVCAT, 1181
FAMONC, 565
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FINITE, 169
FSAGG, 624
MONOGEN, 1602
OC, 1062
ORDFIN, 337
PACFFC, 1548
size?
A1AGG, 687
AGG, 70
ALAGG, 824
ARR2CAT, 359
BGAGG, 279
BRAGG, 374
BTAGG, 734
BTCAT, 528
CLAGG, 289
DIAGG, 535
DIOPS, 387
DIRPCAT, 1004
DLAGG, 394

DQAGG, 542
DSTRCAT, 133
ELAGG, 550
FLAGG, 558
FSAGG, 624
HOAGG, 183
IXAGG, 320
KDAGG, 632
LNAGG, 407
LSAGG, 663
LZSTAGG, 640
MATCAT, 437
MDAGG, 572
MSETAGG, 675
NTSCAT, 749
OMSAGG, 765
PRQAGG, 482
PSETCAT, 468
PTCAT, 896
QUAGG, 488
RCAGG, 353
RMATCAT, 903
RSETCAT, 703
SETAGG, 494
SFRTCAT, 779
SKAGG, 502
SMATCAT, 1085
SNTSCAT, 914
SRAGG, 789
STAGG, 588
STRICAT, 923
TBAGG, 800
TSETCAT, 598
URAGG, 509
VECTCAT, 811

sizeLess?
ACF, 1298
ACFS, 1500
COMPCAT, 1625
EUCDOM, 1187
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FPC, 1325
FPS, 1532
FS, 1341
INS, 1231
LOCPOWC, 1243
MONOGEN, 1602
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPXSCAT, 1464
UPXSCCA, 1574
XF, 1513

SKAGG, 500
?=?, 502
?~=?, 502
#?, 502
any?, 502
bag, 502
coerce, 502
copy, 502
count, 502
depth, 502
empty, 502
empty?, 502
eq?, 502
eval, 502
every?, 502
extract, 502
finiteAggregate, 502
hash, 502
insert, 502
inspect, 502
latex, 502
less?, 502
map, 502
member?, 502
members, 502
more?, 502
parts, 502
pop, 502

push, 502
sample, 502
shallowlyMutable, 502
size?, 502
top, 502
SMATCAT, 1082
-?, 1085
?**?, 1085
?*?, 1085
?+?, 1085
?-?, 1085
?/?, 1085
?=?, 1085
?^?, 1085
?~=?, 1085
#?, 1085
0, 1085
1, 1085
antisymmetric?, 1085
any?, 1085
characteristic, 1085
coerce, 1085
column, 1085
copy, 1085
count, 1085
D, 1085
determinant, 1085
diagonal, 1085
diagonal?, 1085
diagonalMatrix, 1085
diagonalProduct, 1085
differentiate, 1085
elt, 1085
empty, 1085
empty?, 1085
eq?, 1085
eval, 1085
every?, 1085
exquo, 1085
finiteAggregate, 1085
hash, 1085
inverse, 1085
latex, 1085
leftUnitary, 1085
less?, 1085
listOfLists, 1085
map, 1085
matrix, 1085
maxColIndex, 1085
maxRowIndex, 1085
member?, 1085
members, 1085
minColIndex, 1085
minordet, 1085
minRowIndex, 1085
more?, 1085
ncols, 1085
nrows, 1085
nullity, 1085
nullSpace, 1085
one?, 1085
parts, 1085
qelt, 1085
rank, 1085
recip, 1085
reducedSystem, 1085
retract, 1085
retractIfCan, 1085
rightUnitary, 1085
row, 1085
rowEchelon, 1085
sample, 1085
scalarMatrix, 1085
size?, 1085
square?, 1085
subtractIfCan, 1085
symmetric?, 1085
trace, 1085
unitsKnown, 1085
zero?, 1085
SNTSCAT, 910
?=?, 914
?~=?, 914
#?, 914
algebraic?, 914
algebraicCoefficients?, 914
algebraicVariables, 914
any?, 914
augment, 914
autoReduced?, 914
basicSet, 914
coerce, 914

coHeight, 914
collect, 914
collectQuasiMonic, 914
collectUnder, 914
collectUpper, 914
construct, 914
convert, 914
copy, 914
count, 914
degree, 914
empty, 914
empty?, 914
eq?, 914
eval, 914
every?, 914
extend, 914
extendIfCan, 914
find, 914
finiteAggregate, 915
first, 914
hash, 914
headReduce, 914
headReduced?, 914
headRemainder, 914
infRittWu?, 914
initiallyReduce, 914
initiallyReduced?, 914
initials, 914
internalAugment, 914
intersect, 914
invertible?, 914
invertibleElseSplit?, 914
invertibleSet, 914
last, 914
lastSubResultant, 914
lastSubResultantElseSplit, 914
latex, 914
less?, 914
mainVariable?, 914
mainVariables, 914
map, 914
member?, 914
members, 914
more?, 914
mvar, 914
normalized?, 914
parts, 914
purelyAlgebraic?, 914
purelyAlgebraicLeadingMonomial?, 914
purelyTranscendental?, 914
quasiComponent, 914
reduce, 914
reduceByQuasiMonic, 914
reduced?, 914
remainder, 914
remove, 914
removeDuplicates, 914
removeZero, 914
rest, 914
retract, 914
retractIfCan, 914
rewriteIdealWithHeadRemainder, 914
rewriteIdealWithRemainder, 914
rewriteSetWithReduction, 914
roughBase?, 914
roughEqualIdeals?, 914
roughSubIdeal?, 914
roughUnitIdeal?, 914
sample, 914
select, 914
shallowlyMutable, 915
size?, 914
sort, 914
squareFreePart, 914
stronglyReduce, 914
stronglyReduced?, 914
triangular?, 914
trivialIdeal?, 914
variables, 914
zeroSetSplit, 914
zeroSetSplitIntoTriangularSystems, 914
solveLinearPolynomialEquation
COMPCAT, 1625
DPOLCAT, 1310
PFECAT, 1204
POLYCAT, 1260
QFCAT, 1380
RPOLCAT, 1414
ULSCCAT, 1559
UPOLYC, 1477
someBasis
FINAALG, 1022

FRNAALG, 1118
sort
A1AGG, 687
ALAGG, 824
BTAGG, 734
FLAGG, 558
LSAGG, 663
NTSCAT, 749
PSETCAT, 468
PTCAT, 896
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
SRAGG, 789
STRICAT, 923
TSETCAT, 598
VECTCAT, 811
sorted?
A1AGG, 687
ALAGG, 824
BTAGG, 734
FLAGG, 558
LSAGG, 663
PTCAT, 896
SRAGG, 789
STRICAT, 923
VECTCAT, 811
SPACEC, 256
?=?, 258
?~=?, 258
check, 258
closedCurve, 258
closedCurve?, 258
coerce, 258
components, 258
composite, 258
composites, 258
copy, 258
create3Space, 258
curve, 258
curve?, 258
enterPointData, 258
hash, 258
latex, 258
lllip, 258
lllp, 258
llprop, 258
lp, 258
lprop, 258
merge, 258
mesh, 258
mesh?, 258
modifyPointData, 258
numberOfComponents, 258
numberOfComposites, 258
objects, 258
point, 258
point?, 258
polygon, 258
polygon?, 258
subspace, 258
SpecialFunctionCategory, 59
SPFCAT, 59
abs, 60
airyAi, 60
airyBi, 60
besselI, 60
besselJ, 60
besselK, 60
besselY, 60
Beta, 60
digamma, 60
Gamma, 60
polygamma, 60
split
DIVCAT, 1181
SRAGG, 789
STRICAT, 923
sqrt
ACF, 1298
ACFS, 1500
COMPCAT, 1625
FPS, 1532
INTCAT, 1153
MTSCAT, 1196
PADICCT, 1251
RADCAT, 53
RCFIELD, 1392
RNS, 1403
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464

UPXSCCA, 1574
UTSCAT, 1281
square?
MATCAT, 437
RMATCAT, 903
SMATCAT, 1085
squareFree
ACF, 1298
ACFS, 1500
COMPCAT, 1625
DPOLCAT, 1310
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FPC, 1325
FPS, 1532
FS, 1341
INS, 1231
LOCPOWC, 1243
MONOGEN, 1602
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PFECAT, 1204
POLYCAT, 1260
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
RPOLCAT, 1414
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPXSCAT, 1464
UPXSCCA, 1574
XF, 1513
SquareFreeNormalizedTriangularSetCategory, 910
squareFreePart
ACF, 1298
ACFS, 1500
COMPCAT, 1625
DPOLCAT, 1310
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FPC, 1325
FPS, 1532
FS, 1341
INS, 1231
LOCPOWC, 1243
MONOGEN, 1602
NTSCAT, 749
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PFECAT, 1204
POLYCAT, 1260
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
RPOLCAT, 1414
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPXSCAT, 1464
UPXSCCA, 1574
XF, 1513
squareFreePolynomial
COMPCAT, 1625
DPOLCAT, 1310
PFECAT, 1204
POLYCAT, 1260
QFCAT, 1380
RPOLCAT, 1414
ULSCCAT, 1559
UPOLYC, 1477
SquareFreeRegularTriangularSetCategory, 775
SquareMatrixCategory, 1082
squareTop
MATCAT, 437
SRAGG, 786
?<?, 789
?<=?, 789

?>?, 789
?>=?, 789
?.?, 789
?=?, 789
?~=?, 789
#?, 789
any?, 789
coerce, 789
concat, 789
construct, 789
convert, 789
copy, 789
copyInto, 789
count, 789
delete, 789
elt, 789
empty, 789
empty?, 789
entries, 789
entry?, 789
eq?, 789
eval, 789
every?, 789
fill, 789
find, 789
finiteAggregate, 790
first, 789
hash, 789
index?, 789
indices, 789
insert, 789
latex, 789
leftTrim, 789
less?, 789
lowerCase, 789
map, 789
match, 789
match?, 789
max, 789
maxIndex, 789
member?, 789
members, 789
merge, 789
min, 789
minIndex, 789
more?, 789
new, 789
parts, 789
position, 789
prefix?, 789
qelt, 789
qsetelt, 789
reduce, 789
remove, 789
removeDuplicates, 789
replace, 789
reverse, 789
rightTrim, 789
sample, 789
setelt, 789
shallowlyMutable, 790
size?, 789
sort, 789
sorted?, 789
split, 789
substring?, 789
suffix?, 789
swap, 789
trim, 789
upperCase, 789
StackAggregate, 500
STAGG, 585
?.?, 588
?.first, 588
?.last, 588
?.rest, 588
?.value, 588
?=?, 588
?~=?, 588
#?, 588
any?, 588
child?, 588
children, 588
coerce, 588
concat, 588
construct, 588
convert, 588
copy, 588
count, 588
cycleEntry, 588
cycleLength, 588
cycleSplit, 588

cycleTail, 588
cyclic?, 588
delete, 588
distance, 588
elt, 588
empty, 588
empty?, 588
entries, 588
entry?, 588
eq?, 588
eval, 588
every?, 588
explicitlyFinite?, 588
fill, 588
find, 588
first, 588
hash, 588
index?, 588
indices, 588
insert, 588
last, 588
latex, 588
leaf?, 588
leaves, 588
less?, 588
map, 588
maxIndex, 588
member?, 588
members, 588
minIndex, 588
more?, 588
new, 588
node?, 588
nodes, 588
parts, 588
possiblyInfinite?, 588
qelt, 588
qsetelt, 588
reduce, 588
remove, 588
removeDuplicates, 588
rest, 588
sample, 588
second, 588
select, 588
setchildren, 588
setelt, 588
setfirst, 588
setlast, 588
setrest, 588
setvalue, 588
shallowlyMutable, 589
size?, 588
split, 588
swap, 588
tail, 588
third, 588
value, 588
STEP, 252
?=?, 253
?~=?, 253
coerce, 253
hash, 253
init, 253
latex, 253
nextItem, 253
StepThrough, 252
StreamAggregate, 585
STRICAT, 920
?<?, 923
?<=?, 923
?>?, 923
?>=?, 923
?.?, 923
?=?, 923
?~=?, 923
#?, 923
any?, 923
coerce, 923
concat, 923
construct, 923
convert, 923
copy, 923
copyInto, 923
count, 923
delete, 923
elt, 923
empty, 923
empty?, 923
entries, 923
entry?, 923
eq?, 923

eval, 923
every?, 923
fill, 923
find, 923
finiteAggregate, 923
first, 923
hash, 923
index?, 923
indices, 923
insert, 923
latex, 923
leftTrim, 923
less?, 923
lowerCase, 923
map, 923
match, 923
match?, 923
max, 923
maxIndex, 923
member?, 923
members, 923
merge, 923
min, 923
minIndex, 923
more?, 923
new, 923
OMwrite, 923
parts, 923
position, 923
prefix?, 923
qelt, 923
qsetelt, 923
reduce, 923
remove, 923
removeDuplicates, 923
replace, 923
reverse, 923
rightTrim, 923
sample, 923
select, 923
setelt, 923
shallowlyMutable, 923
size?, 923
sort, 923
sorted?, 923
split, 923
string, 923
substring?, 923
suffix?, 923
swap, 923
trim, 923
upperCase, 923
string
SEXCAT, 248
STRICAT, 923
string?
SEXCAT, 248
StringAggregate, 786
StringCategory, 920
stronglyReduce
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
stronglyReduced?
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
structuralConstants
FINAALG, 1022
FRNAALG, 1118
subMatrix
MATCAT, 437
submod
INS, 1231
subMultV
INFCLCT, 1366
subResultantChain
RPOLCAT, 1414
subResultantGcd
RPOLCAT, 1414
UPOLYC, 1477
subset?
FSAGG, 624
MSETAGG, 675
OMSAGG, 765
SETAGG, 494
subspace
SPACEC, 258

subst
ACFS, 1500
ES, 302
FS, 1341
substring?
SRAGG, 789
STRICAT, 923
subtractIfCan
ABELGRP, 522
ACF, 1298
ACFS, 1500
ALGEBRA, 949
AMR, 1103
BMODULE, 728
CABMON, 381
CHARNZ, 835
CHARZ, 840
COMPCAT, 1625
COMRING, 845
DIFEXT, 955
DIFRING, 850
DIRPCAT, 1004
DIVCAT, 1181
DIVRING, 1014
DPOLCAT, 1310
ENTIRER, 855
EUCDOM, 1187
FAMONC, 565
FAMR, 1143
FAXF, 1587
FDIVCAT, 618
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FINAALG, 1022
FINRALG, 1331
FLALG, 1043
FLINEXP, 962
FMCAT, 859
FMTC, 1110
FPC, 1325
FPS, 1532
FRAMALG, 1541
FRNAALG, 1118
FS, 1341
GCDDOM, 1131
INS, 1231
INTCAT, 1153
INTDOM, 1049
LALG, 865
LIECAT, 968
LINEXP, 871
LMODULE, 657
LOCPOWC, 1243
LODOCAT, 973
MLO, 1055
MODULE, 876
MONOGEN, 1602
MTSCAT, 1196
NAALG, 982
NARNG, 681
NASRING, 742
OAGROUP, 756
OAMONS, 760
OC, 1062
OCAMON, 697
OINTDOM, 1137
ORDRING, 881
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PDRING, 887
PFECAT, 1204
PID, 1168
POLYCAT, 1260
PSCAT, 1161
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1392
RING, 772
RMATCAT, 903
RMODULE, 716
RNG, 720
RNS, 1403
RPOLCAT, 1414
SMATCAT, 1085
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559

UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
VSPACE, 989
XALG, 942
XF, 1513
XFALG, 994
XPOLYC, 1096
suffix?
SRAGG, 789
STRICAT, 923
summation
COMBOPC, 74
sup
DIRPCAT, 1004
INTCAT, 1153
OAMONS, 760
supp
DIVCAT, 1181
suppOfPole
DIVCAT, 1181
suppOfZero
DIVCAT, 1181
supRittWu?
RPOLCAT, 1414
symbNameV
INFCLCT, 1366
symbol
SEXCAT, 248
symbol?
SEXCAT, 248
symmetric?
MATCAT, 437
RMATCAT, 903
SMATCAT, 1085
symmetricDifference
FSAGG, 624
MSETAGG, 675
OMSAGG, 765
SETAGG, 494
symmetricPower
LODOCAT, 973
symmetricProduct
LODOCAT, 973
symmetricRemainder
INS, 1231
symmetricSquare
LODOCAT, 973

table
ALAGG, 824
TBAGG, 800
TableAggregate, 797
tableForDiscreteLogarithm
COMPCAT, 1625
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
MONOGEN, 1602
PACFFC, 1548
tail
ALAGG, 824
DLAGG, 394
LSAGG, 663
LZSTAGG, 640
RPOLCAT, 1414
STAGG, 588
URAGG, 509
tan
COMPCAT, 1625
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
TRIGCAT, 64
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
tanh
COMPCAT, 1625
HYPCAT, 33
INTCAT, 1153
LFCAT, 193
MTSCAT, 1196
TRANFUN, 118
ULSCAT, 1452
ULSCCAT, 1559
UPXSCAT, 1464

UPXSCCA, 1574
UTSCAT, 1281
tanhIfCan
PTRANFN, 43
tanIfCan
PTRANFN, 43
taylor
ULSCCAT, 1559
taylorIfCan
ULSCCAT, 1559
taylorRep
ULSCCAT, 1559
TBAGG, 797
?.?, 800
?=?, 800
?~=?, 800
#?, 800
any?, 800
bag, 800
coerce, 800
construct, 800
convert, 800
copy, 800
count, 800
dictionary, 800
elt, 800
empty, 800
empty?, 800
entries, 800
entry?, 800
eq?, 800
eval, 800
every?, 800
extract, 800
fill, 800
find, 800
finiteAggregate, 800
first, 800
hash, 800
index?, 800
indices, 800
insert, 800
inspect, 800
key?, 800
keys, 800
latex, 800
less?, 800
map, 800
maxIndex, 800
member?, 800
members, 800
minIndex, 800
more?, 800
parts, 800
qelt, 800
qsetelt, 800
reduce, 800
remove, 800
removeDuplicates, 800
sample, 800
search, 800
select, 800
setelt, 800
shallowlyMutable, 800
size?, 800
swap, 800
table, 800
terms
DIVCAT, 1181
FAMONC, 565
LOCPOWC, 1243
ULSCAT, 1452
ULSCCAT, 1559
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
third
ALAGG, 824
LSAGG, 663
LZSTAGG, 640
STAGG, 588
URAGG, 509
ThreeSpaceCategory, 256
top
DQAGG, 542
SKAGG, 502
totalDegree
DPOLCAT, 1310
POLYCAT, 1260
RPOLCAT, 1414
UPOLYC, 1477

tower
ACFS, 1500
ES, 302
FS, 1341
trace
COMPCAT, 1625
FAXF, 1587
FFCAT, 1647
FINRALG, 1331
FRAMALG, 1541
MONOGEN, 1602
SMATCAT, 1085
traceMatrix
COMPCAT, 1625
FFCAT, 1647
FINRALG, 1331
FRAMALG, 1541
MONOGEN, 1602
TRANFUN, 117
?**?, 118
acos, 118
acosh, 118
acot, 118
acoth, 118
acsc, 118
acsch, 118
asec, 118
asech, 118
asin, 118
asinh, 118
atan, 118
atanh, 118
cos, 118
cosh, 118
cot, 118
coth, 118
csc, 118
csch, 118
exp, 118
log, 118
pi, 118
sec, 118
sech, 118
sin, 118
sinh, 118
tan, 118
tanh, 118
transcendenceDegree
FAXF, 1587
PACEXTC, 1668
PACRATC, 1613
XF, 1513
transcendent?
FAXF, 1587
PACEXTC, 1668
PACRATC, 1613
XF, 1513
TranscendentalFunctionCategory, 117
transCoord
BLMETCT, 128
transpose
MATCAT, 437
tree
DSTRCAT, 133
triangular?
NTSCAT, 749
PSETCAT, 468
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
TriangularSetCategory, 595
TRIGCAT, 63
cos, 64
cot, 64
csc, 64
sec, 64
sin, 64
tan, 64
TrigonometricFunctionCategory, 63
trim
SRAGG, 789
STRICAT, 923
trivialIdeal?
NTSCAT, 749
PSETCAT, 468
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
trunc
FLALG, 1043

XPOLYC, 1096
truncate
FPS, 1532
LOCPOWC, 1243
RNS, 1403
ULSCAT, 1452
ULSCCAT, 1559
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
TSETCAT, 595
?=?, 598
?~=?, 598
#?, 598
algebraic?, 598
algebraicVariables, 598
any?, 598
autoReduced?, 598
basicSet, 598
coerce, 598
coHeight, 598
collect, 598
collectQuasiMonic, 598
collectUnder, 598
collectUpper, 598
construct, 598
convert, 598
copy, 598
count, 598
degree, 598
empty, 598
empty?, 598
eq?, 598
eval, 598
every?, 598
extend, 598
extendIfCan, 598
find, 598
finiteAggregate, 599
first, 598
hash, 598
headReduce, 598
headReduced?, 598
headRemainder, 598
infRittWu?, 598
initiallyReduce, 598
initiallyReduced?, 598
initials, 598
last, 598
latex, 598
less?, 598
mainVariable?, 598
mainVariables, 598
map, 598
member?, 598
members, 598
more?, 598
mvar, 598
normalized?, 598
parts, 598
quasiComponent, 598
reduce, 598
reduceByQuasiMonic, 598
reduced?, 598
remainder, 598
remove, 598
removeDuplicates, 598
removeZero, 598
rest, 598
retract, 598
retractIfCan, 598
rewriteIdealWithHeadRemainder, 598
rewriteIdealWithRemainder, 598
rewriteSetWithReduction, 598
roughBase?, 598
roughEqualIdeals?, 598
roughSubIdeal?, 598
roughUnitIdeal?, 598
sample, 598
select, 598
shallowlyMutable, 598
size?, 598
sort, 598
stronglyReduce, 598
stronglyReduced?, 598
triangular?, 598
trivialIdeal?, 598
variables, 598
zeroSetSplit, 598
zeroSetSplitIntoTriangularSystems, 598
TwoDimensionalArrayCategory, 357

TYPE, 66
Type, 66
type
    BLMETCT, 128

UFD, 1172
    -?, 1174
    ?**?, 1174
    ?*?, 1174
    ?+?, 1174
    ?-?, 1174
    ?=?, 1174
    ?^?, 1174
    ?~=?, 1174
    0, 1174
    1, 1174
    associates?, 1174
    characteristic, 1174
    coerce, 1174
    commutative("*"), 1174
    exquo, 1174
    factor, 1174
    gcd, 1174
    gcdPolynomial, 1174
    hash, 1174
    latex, 1174
    lcm, 1174
    leftUnitary, 1174
    noZeroDivisors, 1174
    one?, 1174
    prime?, 1174
    recip, 1174
    rightUnitary, 1174
    sample, 1174
    squareFree, 1174
    squareFreePart, 1174
    subtractIfCan, 1174
    unit?, 1174
    unitCanonical, 1174
    unitNormal, 1174
    unitsKnown, 1174
    zero?, 1174
ULSCAT, 1448
    -?, 1452
    ?**?, 1452
    ?*?, 1452
    ?+?, 1452
    ?-?, 1452
    ?.?, 1452
    ?/?, 1452
    ?=?, 1452
    ?^?, 1452
    ?~=?, 1452
    ?quo?, 1452
    ?rem?, 1452
    0, 1452
    1, 1452
    acos, 1452
    acosh, 1452
    acot, 1452
    acoth, 1452
    acsc, 1452
    acsch, 1452
    approximate, 1452
    asec, 1452
    asech, 1452
    asin, 1452
    asinh, 1452
    associates?, 1452
    atan, 1452
    atanh, 1452
    canonicalsClosed, 1452
    canonicalUnitNormal, 1452
    center, 1452
    characteristic, 1452
    charthRoot, 1452
    coefficient, 1452
    coerce, 1452
    commutative("*"), 1452
    complete, 1452
    cos, 1452
    cosh, 1452
    cot, 1452
    coth, 1452
    csc, 1452
    csch, 1452
    D, 1452
    degree, 1452
    differentiate, 1452
    divide, 1452
    euclideanSize, 1452
    eval, 1452

exp, 1452
expressIdealMember, 1452
exquo, 1452
extend, 1452
extendedEuclidean, 1452
factor, 1452
gcd, 1452
gcdPolynomial, 1452
hash, 1452
integrate, 1452
inv, 1452
latex, 1452
lcm, 1452
leadingCoefficient, 1452
leadingMonomial, 1452
leftUnitary, 1452
log, 1452
map, 1452
monomial, 1452
monomial?, 1452
multiEuclidean, 1452
multiplyCoefficients, 1452
multiplyExponents, 1452
noZeroDivisors, 1452
nthRoot, 1452
one?, 1452
order, 1452
pi, 1452
pole?, 1452
prime?, 1452
principalIdeal, 1452
rationalFunction, 1452
recip, 1452
reductum, 1452
rightUnitary, 1452
sample, 1452
sec, 1452
sech, 1452
series, 1452
sin, 1452
sinh, 1452
sizeLess?, 1452
sqrt, 1452
squareFree, 1452
squareFreePart, 1452
subtractIfCan, 1452
tan, 1452
tanh, 1452
terms, 1452
truncate, 1452
unit?, 1452
unitCanonical, 1452
unitNormal, 1452
unitsKnown, 1452
variable, 1452
variables, 1452
zero?, 1452
ULSCCAT, 1553
-?, 1559
?<?, 1559
?<=?, 1559
?>?, 1559
?>=?, 1559
?**?, 1559
?*?, 1559
?+?, 1559
?-?, 1559
?.?, 1559
?/?, 1559
?=?, 1559
?^?, 1559
?~=?, 1559
?quo?, 1559
?rem?, 1559
0, 1559
1, 1559
abs, 1559
acos, 1559
acosh, 1559
acot, 1559
acoth, 1559
acsc, 1559
acsch, 1559
approximate, 1559
asec, 1559
asech, 1559
asin, 1559
asinh, 1559
associates?, 1559
atan, 1559
atanh, 1559
canonicalsClosed, 1560

canonicalUnitNormal, 1560
ceiling, 1559
center, 1559
characteristic, 1559
charthRoot, 1559
coefficient, 1559
coerce, 1559
commutative("*"), 1560
complete, 1559
conditionP, 1559
convert, 1559
cos, 1559
cosh, 1559
cot, 1559
coth, 1559
csc, 1559
csch, 1559
D, 1559
degree, 1559
denom, 1559
denominator, 1559
differentiate, 1559
divide, 1559
euclideanSize, 1559
eval, 1559
exp, 1559
expressIdealMember, 1559
exquo, 1559
extend, 1559
extendedEuclidean, 1559
factor, 1559
factorPolynomial, 1559
factorSquareFreePolynomial, 1559
floor, 1559
fractionPart, 1559
gcd, 1559
gcdPolynomial, 1559
hash, 1559
init, 1559
integrate, 1559
inv, 1559
latex, 1559
laurent, 1559
lcm, 1559
leadingCoefficient, 1559
leadingMonomial, 1559
leftUnitary, 1560
log, 1559
map, 1559
max, 1559
min, 1559
monomial, 1559
monomial?, 1559
multiEuclidean, 1559
multiplyCoefficients, 1559
multiplyExponents, 1559
negative?, 1559
nextItem, 1559
noZeroDivisors, 1560
nthRoot, 1559
numer, 1559
numerator, 1559
one?, 1559
order, 1559
patternMatch, 1559
pi, 1559
pole?, 1559
positive?, 1559
prime?, 1559
principalIdeal, 1559
random, 1559
rationalFunction, 1559
recip, 1559
reducedSystem, 1559
reductum, 1559
removeZeroes, 1559
retract, 1559
retractIfCan, 1559
rightUnitary, 1560
sample, 1559
sec, 1559
sech, 1559
series, 1559
sign, 1559
sin, 1559
sinh, 1559
sizeLess?, 1559
solveLinearPolynomialEquation, 1559
sqrt, 1559
squareFree, 1559
squareFreePart, 1559
squareFreePolynomial, 1559

subtractIfCan, 1559
tan, 1559
tanh, 1559
taylor, 1559
taylorIfCan, 1559
taylorRep, 1559
terms, 1559
truncate, 1559
unit?, 1559
unitCanonical, 1559
unitNormal, 1559
unitsKnown, 1560
variable, 1559
variables, 1559
wholePart, 1559
zero?, 1559
UnaryRecursiveAggregate, 506
union
FSAGG, 624
MSETAGG, 675
OMSAGG, 765
SETAGG, 494
UniqueFactorizationDomain, 1172
unit
FINAALG, 1022
FRNAALG, 1118
unit?
ACF, 1298
ACFS, 1500
AMR, 1103
COMPCAT, 1625
DPOLCAT, 1310
EUCDOM, 1187
FAMR, 1143
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FMTC, 1110
FPC, 1325
FPS, 1532
FS, 1341
GCDDOM, 1131
INS, 1231
INTCAT, 1153
INTDOM, 1049
LOCPOWC, 1243
MONOGEN, 1602
MTSCAT, 1196
OINTDOM, 1137
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PFECAT, 1204
PID, 1168
POLYCAT, 1260
PSCAT, 1161
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
RPOLCAT, 1414
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
XF, 1513
unitCanonical
ACF, 1298
ACFS, 1500
AMR, 1103
COMPCAT, 1625
DPOLCAT, 1310
EUCDOM, 1187
FAMR, 1143
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FMTC, 1110
FPC, 1325
FPS, 1532
FS, 1341
GCDDOM, 1131
INS, 1231
INTCAT, 1153
INTDOM, 1049

LOCPOWC, 1243
MONOGEN, 1602
MTSCAT, 1196
OINTDOM, 1137
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PFECAT, 1204
PID, 1168
POLYCAT, 1260
PSCAT, 1161
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
RPOLCAT, 1414
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
XF, 1513
unitNormal
ACF, 1298
ACFS, 1500
AMR, 1103
COMPCAT, 1625
DPOLCAT, 1310
EUCDOM, 1187
FAMR, 1143
FAXF, 1587
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FMTC, 1110
FPC, 1325
FPS, 1532
FS, 1341
GCDDOM, 1131
INS, 1231
INTCAT, 1153
INTDOM, 1049
LOCPOWC, 1243
MONOGEN, 1602
MTSCAT, 1196
OINTDOM, 1137
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PFECAT, 1204
PID, 1168
POLYCAT, 1260
PSCAT, 1161
QFCAT, 1380
RCFIELD, 1392
RNS, 1403
RPOLCAT, 1414
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
XF, 1513
unitsKnown
ACF, 1299
ACFS, 1500
ALGEBRA, 949
AMR, 1103
ATTREG, 11, 582
CHARNZ, 835
CHARZ, 840
COMPCAT, 1626
COMRING, 845
DIFEXT, 955
DIFRING, 850
DIRPCAT, 1005
DIVRING, 1015
DPOLCAT, 1311
ENTIRER, 855
EUCDOM, 1187
FAMR, 1143
FAXF, 1587
FFCAT, 1648

FFIELDC, 1521
FIELD, 1224
FINAALG, 1022
FINRALG, 1331
FLINEXP, 962
FMTC, 1110
FPC, 1326
FPS, 1533
FRAMALG, 1541
FRNAALG, 1118
FS, 1341
GCDDOM, 1131
GROUP, 400
INS, 1231
INTCAT, 1153
INTDOM, 1049
LALG, 866
LINEXP, 871
LOCPOWC, 1243
LODOCAT, 973
MLO, 1055
MONOGEN, 1603
MTSCAT, 1196
OC, 1062
OINTDOM, 1137
ORDRING, 881
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1549
PACPERC, 1372
PACRATC, 1614
PADICCT, 1251
PDRING, 887
PFECAT, 1204
PID, 1168
POLYCAT, 1260
PSCAT, 1161
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1393
RING, 772
RPOLCAT, 1415
SMATCAT, 1085
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1560
UPOLYC, 1478
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
XALG, 942
XF, 1513
XFALG, 994
XPOLYC, 1096
unitVector
DIRPCAT, 1004
univariate
ACFS, 1500
DPOLCAT, 1310
FS, 1341
POLYCAT, 1260
RPOLCAT, 1414
UPOLYC, 1477
UnivariateLaurentSeriesCategory, 1448
UnivariateLaurentSeriesConstructorCategory, 1553
UnivariatePolynomialCategory, 1471
UnivariatePowerSeriesCategory, 1210
UnivariatePuiseuxSeriesCategory, 1460
UnivariatePuiseuxSeriesConstructorCategory, 1570
UnivariateSkewPolynomialCategory, 929
UnivariateTaylorSeriesCategory, 1277
universe
FSAGG, 624
unmakeSUP
UPOLYC, 1477
UPOLYC, 1471
-?, 1477
?<?, 1477
?<=?, 1477
?>?, 1477
?>=?, 1477
?**?, 1477
?*?, 1477
?+?, 1477
?-?, 1477
?.?, 1477
?/?, 1477
?=?, 1477
?^?, 1477

?~=?, 1477
?quo?, 1477
?rem?, 1477
0, 1477
1, 1477
additiveValuation, 1477
associates?, 1477
binomThmExpt, 1477
canonicalUnitNormal, 1478
characteristic, 1477
charthRoot, 1477
coefficient, 1477
coefficients, 1477
coerce, 1477
commutative("*"), 1477
composite, 1477
conditionP, 1477
content, 1477
convert, 1477
D, 1477
degree, 1477
differentiate, 1477
discriminant, 1477
divide, 1477
divideExponents, 1477
elt, 1477
euclideanSize, 1477
eval, 1477
expressIdealMember, 1477
exquo, 1477
extendedEuclidean, 1477
factor, 1477
factorPolynomial, 1477
factorSquareFreePolynomial, 1477
gcd, 1477
gcdPolynomial, 1477
ground, 1477
ground?, 1477
hash, 1477
init, 1477
integrate, 1477
isExpt, 1477
isPlus, 1477
isTimes, 1477
karatsubaDivide, 1477
latex, 1477
lcm, 1477
leadingCoefficient, 1477
leadingMonomial, 1477
leftUnitary, 1478
mainVariable, 1477
makeSUP, 1477
map, 1477
mapExponents, 1477
max, 1477
min, 1477
minimumDegree, 1477
monicDivide, 1477
monomial, 1477
monomial?, 1477
monomials, 1477
multiEuclidean, 1477
multiplyExponents, 1477
multivariate, 1477
nextItem, 1477
noZeroDivisors, 1477
numberOfMonomials, 1477
one?, 1477
order, 1477
patternMatch, 1477
pomopo, 1477
prime?, 1477
primitiveMonomials, 1477
primitivePart, 1477
principalIdeal, 1477
pseudoDivide, 1477
pseudoQuotient, 1477
pseudoRemainder, 1477
recip, 1477
reducedSystem, 1477
reductum, 1477
resultant, 1477
retract, 1477
retractIfCan, 1477
rightUnitary, 1478
sample, 1477
separate, 1477
shiftLeft, 1477
shiftRight, 1477
sizeLess?, 1477
solveLinearPolynomialEquation, 1477
squareFree, 1477

squareFreePart, 1477
squareFreePolynomial, 1477
subResultantGcd, 1477
subtractIfCan, 1477
totalDegree, 1477
unit?, 1477
unitCanonical, 1477
unitNormal, 1477
unitsKnown, 1478
univariate, 1477
unmakeSUP, 1477
variables, 1477
vectorise, 1477
zero?, 1477
upperCase
SRAGG, 789
STRICAT, 923
UPSCAT, 1210
-?, 1213
?**?, 1213
?*?, 1213
?+?, 1213
?-?, 1213
?.?, 1213
?/?, 1213
?=?, 1213
?^?, 1213
?~=?, 1213
0, 1213
1, 1213
approximate, 1213
associates?, 1213
center, 1213
characteristic, 1213
charthRoot, 1213
coefficient, 1213
coerce, 1213
commutative("*"), 1213
complete, 1213
D, 1213
degree, 1213
differentiate, 1213
eval, 1213
exquo, 1213
extend, 1213
hash, 1213
latex, 1213
leadingCoefficient, 1213
leadingMonomial, 1213
leftUnitary, 1213
map, 1213
monomial, 1213
monomial?, 1213
multiplyExponents, 1213
noZeroDivisors, 1213
one?, 1213
order, 1213
pole?, 1213
recip, 1213
reductum, 1213
rightUnitary, 1213
sample, 1213
subtractIfCan, 1213
terms, 1213
truncate, 1213
unit?, 1213
unitCanonical, 1213
unitNormal, 1213
unitsKnown, 1213
variable, 1213
variables, 1213
zero?, 1213
UPXSCAT, 1460
-?, 1464
?**?, 1464
?*?, 1464
?+?, 1464
?-?, 1464
?.?, 1464
?/?, 1464
?=?, 1464
?^?, 1464
?~=?, 1464
?quo?, 1464
?rem?, 1464
0, 1464
1, 1464
acos, 1464
acosh, 1464
acot, 1464
acoth, 1464
acsc, 1464

acsch, 1464
approximate, 1464
asec, 1464
asech, 1464
asin, 1464
asinh, 1464
associates?, 1464
atan, 1464
atanh, 1464
canonicalsClosed, 1464
canonicalUnitNormal, 1464
center, 1464
characteristic, 1464
charthRoot, 1464
coefficient, 1464
coerce, 1464
commutative("*"), 1464
complete, 1464
cos, 1464
cosh, 1464
cot, 1464
coth, 1464
csc, 1464
csch, 1464
D, 1464
degree, 1464
differentiate, 1464
divide, 1464
euclideanSize, 1464
eval, 1464
exp, 1464
expressIdealMember, 1464
exquo, 1464
extend, 1464
extendedEuclidean, 1464
factor, 1464
gcd, 1464
gcdPolynomial, 1464
hash, 1464
integrate, 1464
inv, 1464
latex, 1464
lcm, 1464
leadingCoefficient, 1464
leadingMonomial, 1464
leftUnitary, 1464
log, 1464
map, 1464
monomial, 1464
monomial?, 1464
multiEuclidean, 1464
multiplyExponents, 1464
noZeroDivisors, 1464
nthRoot, 1464
one?, 1464
order, 1464
pi, 1464
pole?, 1464
prime?, 1464
principalIdeal, 1464
recip, 1464
reductum, 1464
rightUnitary, 1464
sample, 1464
sec, 1464
sech, 1464
series, 1464
sin, 1464
sinh, 1464
sizeLess?, 1464
sqrt, 1464
squareFree, 1464
squareFreePart, 1464
subtractIfCan, 1464
tan, 1464
tanh, 1464
terms, 1464
truncate, 1464
unit?, 1464
unitCanonical, 1464
unitNormal, 1464
unitsKnown, 1464
variable, 1464
variables, 1464
zero?, 1464
UPXSCCA, 1570
-?, 1574
?**?, 1574
?*?, 1574
?+?, 1574
?-?, 1574
?.?, 1574

?/?, 1574
?=?, 1574
?^?, 1574
?~=?, 1574
?quo?, 1574
?rem?, 1574
0, 1574
1, 1574
acos, 1574
acosh, 1574
acot, 1574
acoth, 1574
acsc, 1574
acsch, 1574
approximate, 1574
asec, 1574
asech, 1574
asin, 1574
asinh, 1574
associates?, 1574
atan, 1574
atanh, 1574
canonicalsClosed, 1575
canonicalUnitNormal, 1575
center, 1574
characteristic, 1574
charthRoot, 1574
coefficient, 1574
coerce, 1574
commutative("*"), 1575
complete, 1574
cos, 1574
cosh, 1574
cot, 1574
coth, 1574
csc, 1574
csch, 1574
D, 1574
degree, 1574
differentiate, 1574
divide, 1574
euclideanSize, 1574
eval, 1574
exp, 1574
expressIdealMember, 1574
exquo, 1574
extend, 1574
extendedEuclidean, 1574
factor, 1574
gcd, 1574
gcdPolynomial, 1574
hash, 1574
integrate, 1574
inv, 1574
latex, 1574
laurent, 1574
laurentIfCan, 1574
laurentRep, 1574
lcm, 1574
leadingCoefficient, 1574
leadingMonomial, 1574
leftUnitary, 1574
log, 1574
map, 1574
monomial, 1574
monomial?, 1574
multiEuclidean, 1574
multiplyExponents, 1574
noZeroDivisors, 1575
nthRoot, 1574
one?, 1574
order, 1574
pi, 1574
pole?, 1574
prime?, 1574
principalIdeal, 1574
puiseux, 1574
rationalPower, 1574
recip, 1574
reductum, 1574
retract, 1574
retractIfCan, 1574
rightUnitary, 1574
sample, 1574
sec, 1574
sech, 1574
series, 1574
sin, 1574
sinh, 1574
sizeLess?, 1574
sqrt, 1574
squareFree, 1574

squareFreePart, 1574
subtractIfCan, 1574
tan, 1574
tanh, 1574
terms, 1574
truncate, 1574
unit?, 1574
unitCanonical, 1574
unitNormal, 1574
unitsKnown, 1574
variable, 1574
variables, 1574
zero?, 1574
URAGG, 506
?.first, 509
?.last, 509
?.rest, 509
?.value, 509
?=?, 509
?~=?, 509
#?, 509
any?, 509
child?, 509
children, 509
coerce, 509
concat, 509
copy, 509
count, 509
cycleEntry, 509
cycleLength, 509
cycleSplit, 509
cycleTail, 509
cyclic?, 509
distance, 509
empty, 509
empty?, 509
eq?, 509
eval, 509
every?, 509
finiteAggregate, 509
first, 509
hash, 509
last, 509
latex, 509
leaf?, 509
leaves, 509
less?, 509
map, 509
member?, 509
members, 509
more?, 509
node?, 509
nodes, 509
parts, 509
rest, 509
sample, 509
second, 509
setchildren, 509
setelt, 509
setfirst, 509
setlast, 509
setrest, 509
setvalue, 509
shallowlyMutable, 509
size?, 509
split, 509
tail, 509
third, 509
value, 509
UTSCAT, 1277
-?, 1281
?**?, 1281
?*?, 1281
?+?, 1281
?-?, 1281
?.?, 1281
?/?, 1281
?=?, 1281
?^?, 1281
?~=?, 1281
0, 1281
1, 1281
acos, 1281
acosh, 1281
acot, 1281
acoth, 1281
acsc, 1281
acsch, 1281
approximate, 1281
asec, 1281
asech, 1281
asin, 1281

asinh, 1281
associates?, 1281
atan, 1281
atanh, 1281
center, 1281
characteristic, 1281
charthRoot, 1281
coefficient, 1281
coefficients, 1281
coerce, 1281
commutative("*"), 1281
complete, 1281
cos, 1281
cosh, 1281
cot, 1281
coth, 1281
csc, 1281
csch, 1281
D, 1281
degree, 1281
differentiate, 1281
eval, 1281
exp, 1281
exquo, 1281
extend, 1281
hash, 1281
integrate, 1281
latex, 1281
leadingCoefficient, 1281
leadingMonomial, 1281
leftUnitary, 1281
log, 1281
map, 1281
monomial, 1281
monomial?, 1281
multiplyCoefficients, 1281
multiplyExponents, 1281
noZeroDivisors, 1281
nthRoot, 1281
one?, 1281
order, 1281
pi, 1281
pole?, 1281
polynomial, 1281
quoByVar, 1281
recip, 1281
reductum, 1281
rightUnitary, 1281
sample, 1281
sec, 1281
sech, 1281
series, 1281
sin, 1281
sinh, 1281
sqrt, 1281
subtractIfCan, 1281
tan, 1281
tanh, 1281
terms, 1281
truncate, 1281
unit?, 1281
unitCanonical, 1281
unitNormal, 1281
unitsKnown, 1281
variable, 1281
variables, 1281
zero?, 1281

value
ALAGG, 824
BRAGG, 374
BTCAT, 528
DLAGG, 394
DSTRCAT, 133
LSAGG, 663
LZSTAGG, 640
RCAGG, 353
STAGG, 588
URAGG, 509
variable
DVARCAT, 296
LOCPOWC, 1243
ULSCAT, 1452
ULSCCAT, 1559
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
variables
ACFS, 1500
DPOLCAT, 1310
FS, 1341

LOCPOWC, 1243
MTSCAT, 1196
NTSCAT, 749
POLYCAT, 1260
PSCAT, 1161
PSETCAT, 468
RPOLCAT, 1414
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
varList
FLALG, 1043
XFALG, 994
XPOLYC, 1096
VECTCAT, 808
-?, 811
?<?, 811
?<=?, 811
?>?, 811
?>=?, 811
?*?, 811
?+?, 811
?-?, 811
?.?, 811
?=?, 811
?~=?, 811
#?, 811
any?, 811
coerce, 811
concat, 811
construct, 811
convert, 811
copy, 811
copyInto, 811
count, 811
cross, 811
delete, 811
dot, 811
elt, 811
empty, 811
empty?, 811
entries, 811
entry?, 811
eq?, 811
eval, 811
every?, 811
fill, 811
find, 811
finiteAggregate, 811
first, 811
hash, 811
index?, 811
indices, 811
insert, 811
latex, 811
length, 811
less?, 811
magnitude, 811
map, 811
max, 811
maxIndex, 811
member?, 811
members, 811
merge, 811
min, 811
minIndex, 811
more?, 811
new, 811
outerProduct, 811
parts, 811
position, 811
qelt, 811
qsetelt, 811
reduce, 811
remove, 811
removeDuplicates, 811
reverse, 811
sample, 811
select, 811
setelt, 811
shallowlyMutable, 811
size?, 811
sort, 811
sorted?, 811

swap, 811
zero, 811
VectorCategory, 808
vectorise
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
UPOLYC, 1477
VectorSpace, 987
vertConcat
MATCAT, 437
VSPACE, 987
-?, 989
?*?, 989
?+?, 989
?-?, 989
?/?, 989
?=?, 989
?~=?, 989
0, 989
coerce, 989
dimension, 989
hash, 989
latex, 989
leftUnitary, 989
rightUnitary, 989
sample, 989
subtractIfCan, 989
zero?, 989

weight
DPOLCAT, 1310
DVARCAT, 296
weights
DPOLCAT, 1310
wholePart
FPS, 1532
QFCAT, 1380
RNS, 1403
ULSCCAT, 1559
width
INTCAT, 1153
writable?
FNCAT, 173

XALG, 940
-?, 942
?**?, 942
?*?, 942
?+?, 942
?-?, 942
?=?, 942
?^?, 942
?~=?, 942
0, 942
1, 942
characteristic, 942
coerce, 942
hash, 942
latex, 942
leftUnitary, 942
one?, 942
recip, 942
rightUnitary, 942
sample, 942
subtractIfCan, 942
unitsKnown, 942
zero?, 942
XAlgebra, 940
XF, 1510
-?, 1513
?**?, 1513
?*?, 1513
?+?, 1513
?-?, 1513
?/?, 1513
?=?, 1513
?^?, 1513
?~=?, 1513
?quo?, 1513
?rem?, 1513
0, 1513
1, 1513
algebraic?, 1513
associates?, 1513
canonicalsClosed, 1513
canonicalUnitNormal, 1513
characteristic, 1513
charthRoot, 1513
coerce, 1513
commutative("*"), 1513

degree, 1513
dimension, 1513
discreteLog, 1513
divide, 1513
euclideanSize, 1513
expressIdealMember, 1513
exquo, 1513
extendedEuclidean, 1513
extensionDegree, 1513
factor, 1513
Frobenius, 1513
gcd, 1513
gcdPolynomial, 1513
hash, 1513
inGroundField?, 1513
inv, 1513
latex, 1513
lcm, 1513
leftUnitary, 1513
multiEuclidean, 1513
noZeroDivisors, 1513
one?, 1513
order, 1513
prime?, 1513
primeFrbenius, 1513
principalIdeal, 1513
recip, 1513
retract, 1513
retractIfCan, 1513
rightUnitary, 1513
sample, 1513
sizeLess?, 1513
squareFree, 1513
squareFreePart, 1513
subtractIfCan, 1513
transcendenceDegree, 1513
transcendent?, 1513
unit?, 1513
unitCanonical, 1513
unitNormal, 1513
unitsKnown, 1513
zero?, 1513
XFALG, 992
-?, 994
?**?, 994
?*?, 994
?+?, 994
?-?, 994
?=?, 994
?^?, 994
?~=?, 994
0, 994
1, 994
characteristic, 994
coef, 994
coerce, 994
commutative("*"), 994
constant, 994
constant?, 994
hash, 994
latex, 994
leftUnitary, 994
lquo, 994
map, 994
mindeg, 994
mindegTerm, 994
mirror, 994
monom, 994
monomial?, 994
noZeroDivisors, 994
one?, 994
quasiRegular, 994
quasiRegular?, 994
recip, 994
retract, 994
retractIfCan, 994
rightUnitary, 994
rquo, 994
sample, 994
sh, 994
subtractIfCan, 994
unitsKnown, 994
varList, 994
zero?, 994
XFreeAlgebra, 992
xor
BTAGG, 734
XPOLYC, 1094
-?, 1096
?**?, 1096
?*?, 1096
?+?, 1096

?-?, 1096
?=?, 1096
?^?, 1096
?~=?, 1096
0, 1096
1, 1096
characteristic, 1096
coef, 1096
coerce, 1096
constant, 1096
constant?, 1096
degree, 1096
hash, 1096
latex, 1096
leftUnitary, 1096
lquo, 1096
map, 1096
maxdeg, 1096
mindeg, 1096
mindegTerm, 1096
mirror, 1096
monom, 1096
monomial?, 1096
noZeroDivisors, 1096
one?, 1096
quasiRegular, 1096
quasiRegular?, 1096
recip, 1096
retract, 1096
retractIfCan, 1096
rightUnitary, 1096
rquo, 1096
sample, 1096
sh, 1096
subtractIfCan, 1096
trunc, 1096
unitsKnown, 1096
varList, 1096
zero?, 1096
XPolynomialsCat, 1094
xRange
PPCURVE, 99
PSCURVE, 103

yCoordinates
FFCAT, 1647
yRange
PPCURVE, 99
PSCURVE, 103

zero
MATCAT, 437
PTCAT, 896
VECTCAT, 811
zero?
ABELGRP, 522
ABELMON, 269
ACF, 1298
ACFS, 1500
ALGEBRA, 949
AMR, 1103
BMODULE, 728
CABMON, 381
CHARNZ, 835
CHARZ, 840
COMPCAT, 1625
COMRING, 845
DIFEXT, 955
DIFRING, 850
DIRPCAT, 1004
DIVCAT, 1181
DIVRING, 1014
DPOLCAT, 1310
ENTIRER, 855
EUCDOM, 1187
FAMONC, 565
FAMR, 1143
FAXF, 1587
FDIVCAT, 618
FFCAT, 1647
FFIELDC, 1521
FIELD, 1224
FINAALG, 1022
FINRALG, 1331
FLALG, 1043
FLINEXP, 962
FMCAT, 859
FMTC, 1110
FPC, 1325
FPS, 1532
FRAMALG, 1541
FRNAALG, 1118

FS, 1341
GCDDOM, 1131
INS, 1231
INTCAT, 1153
INTDOM, 1049
LALG, 865
LIECAT, 968
LINEXP, 871
LMODULE, 657
LOCPOWC, 1243
LODOCAT, 973
MLO, 1055
MODULE, 876
MONOGEN, 1602
MTSCAT, 1196
NAALG, 982
NARNG, 681
NASRING, 742
OAGROUP, 756
OAMON, 577
OAMONS, 760
OASGP, 458
OC, 1062
OCAMON, 697
OINTDOM, 1137
ORDRING, 881
OREPCAT, 931
PACEXTC, 1668
PACFFC, 1548
PACPERC, 1371
PACRATC, 1613
PADICCT, 1251
PDRING, 887
PFECAT, 1204
PID, 1168
POLYCAT, 1260
PSCAT, 1161
QFCAT, 1380
QUATCAT, 1073
RCFIELD, 1392
RING, 772
RMATCAT, 903
RMODULE, 716
RNG, 720
RNS, 1403
RPOLCAT, 1414
RRCC, 231
SMATCAT, 1085
UFD, 1174
ULSCAT, 1452
ULSCCAT, 1559
UPOLYC, 1477
UPSCAT, 1213
UPXSCAT, 1464
UPXSCCA, 1574
UTSCAT, 1281
VSPACE, 989
XALG, 942
XF, 1513
XFALG, 994
XPOLYC, 1096
zeroOf
ACF, 1298
ACFS, 1500
zeroSetSplit
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
zeroSetSplitIntoTriangularSystems
NTSCAT, 749
RSETCAT, 703
SFRTCAT, 779
SNTSCAT, 914
TSETCAT, 598
zerosOf
ACF, 1298
ACFS, 1500
zRange
PSCURVE, 103