

# CoCoA5.2.1 Manual

John Abbott and Anna M. Bigatti

January 9, 2018



# Contents

<b>I</b>	<b>Alphabetical List of Commands</b>	<b>23</b>
<b>I-0</b>	<b>Special Characters</b>	<b>25</b>
I-0.1	operators, shortcuts . . . . .	25
<b>I-1</b>	<b>A</b>	<b>27</b>
I-1.1	abs . . . . .	27
I-1.2	adj . . . . .	27
I-1.3	AffHilbert [OBSOLESCENT] . . . . .	28
I-1.4	AffHilbertFn . . . . .	28
I-1.5	AffHilbertSeries . . . . .	28
I-1.6	AffPoincare [OBSOLESCENT] . . . . .	29
I-1.7	alias . . . . .	29
I-1.8	aliases . . . . .	30
I-1.9	AllReducedGroebnerBases [OBSOLESCENT] . . . . .	30
I-1.10	AlmostQR . . . . .	30
I-1.11	and . . . . .	31
I-1.12	append . . . . .	31
I-1.13	apply . . . . .	32
I-1.14	ApproxPointsNBM . . . . .	32
I-1.15	ApproxSolve . . . . .	33
I-1.16	AreGensMonomial . . . . .	34
I-1.17	AreGensSqFreeMonomial . . . . .	34
I-1.18	ascii . . . . .	35
I-1.19	AsINT . . . . .	35
I-1.20	AsRAT . . . . .	36
<b>I-2</b>	<b>B</b>	<b>37</b>
I-2.1	BaseRing . . . . .	37
I-2.2	BBasis5 . . . . .	37
I-2.3	BettiDiagram . . . . .	38
I-2.4	BettiMatrix . . . . .	38
I-2.5	BettiNumbers . . . . .	39
I-2.6	binomial . . . . .	39
I-2.7	BinomialRepr, BinomialReprShift . . . . .	40

I-2.8	block . . . . .	41
I-2.9	BlockMat . . . . .	41
I-2.10	BlockMat2x2 . . . . .	42
I-2.11	Bool01 . . . . .	42
I-2.12	break . . . . .	43
I-2.13	BringIn . . . . .	43
<b>I-3</b>	<b>C</b>	<b>45</b>
I-3.1	Call [OBSOLETE] . . . . .	45
I-3.2	CallOnGroebnerFanIdeals . . . . .	45
I-3.3	CanonicalHom . . . . .	46
I-3.4	CanonicalRepr . . . . .	47
I-3.5	CartesianProduct, CartesianProductList . . . . .	47
I-3.6	Cast [OBSOLETE] . . . . .	48
I-3.7	ceil . . . . .	48
I-3.8	CFApprox . . . . .	48
I-3.9	CFApproximants . . . . .	49
I-3.10	characteristic . . . . .	49
I-3.11	CharPoly . . . . .	49
I-3.12	ChebyshevPoly . . . . .	50
I-3.13	CheckArgTypes . . . . .	50
I-3.14	ciao . . . . .	51
I-3.15	ClearDenom . . . . .	51
I-3.16	close . . . . .	51
I-3.17	CloseLog . . . . .	51
I-3.18	CoCoA-4 mode . . . . .	52
I-3.19	CocoaLimits . . . . .	52
I-3.20	CocoaPackagePath . . . . .	53
I-3.21	codomain . . . . .	53
I-3.22	CoeffEmbeddingHom . . . . .	53
I-3.23	CoeffHeight . . . . .	54
I-3.24	coefficients . . . . .	54
I-3.25	CoefficientsWRT . . . . .	55
I-3.26	CoeffListWRT . . . . .	56
I-3.27	CoeffOfTerm . . . . .	56
I-3.28	CoeffRing . . . . .	56
I-3.29	ColMat . . . . .	57
I-3.30	colon . . . . .	57
I-3.31	ColumnVectors [OBSOLETE] . . . . .	58
I-3.32	CommonDenom . . . . .	58
I-3.33	Comp [OBSOLETE] . . . . .	58
I-3.34	Comparison Operators . . . . .	59
I-3.35	CompleteToOrd [OBSCOLESCENT] . . . . .	59

I-3.36	compts	59
I-3.37	ComputeElimFirst	60
I-3.38	concat	60
I-3.39	ConcatAntiDiag	60
I-3.40	ConcatDiag	61
I-3.41	ConcatHor	61
I-3.42	ConcatHorList	62
I-3.43	ConcatLists	62
I-3.44	ConcatVer	63
I-3.45	ConcatVerList	63
I-3.46	content	64
I-3.47	ContentFreeFactor	64
I-3.48	ContentWRT	65
I-3.49	ContFrac	65
I-3.50	ContFracToRat	65
I-3.51	continue	66
I-3.52	count	66
I-3.53	CpuTime	67
I-3.54	CRT	67
I-3.55	CRTPoly	68
I-3.56	CurrentRing	68
I-3.57	CurrentTypes	69
I-3.58	cyclotomic	69

<b>I-4</b>	<b>D</b>	<b>71</b>
I-4.1	dashes	71
I-4.2	date	71
I-4.3	DecimalStr	71
I-4.4	define	72
I-4.5	DefiningIdeal	74
I-4.6	deg	74
I-4.7	den	75
I-4.8	DensePoly	75
I-4.9	depth	75
I-4.10	deriv	76
I-4.11	DerivationAction	77
I-4.12	describe	77
I-4.13	det	78
I-4.14	DF	78
I-4.15	DiagMat	79
I-4.16	diff	79
I-4.17	dim	80
I-4.18	discriminant	80

I-4.19	distrib . . . . .	81
I-4.20	div . . . . .	81
I-4.21	DivAlg . . . . .	81
I-4.22	domain . . . . .	82
<b>I-5</b>	<b>E</b>	<b>83</b>
I-5.1	E_ [OBSOLETE] . . . . .	83
I-5.2	eigenfactors . . . . .	83
I-5.3	eigenvectors . . . . .	84
I-5.4	elim . . . . .	84
I-5.5	ElimHomogMat . . . . .	85
I-5.6	ElimMat . . . . .	85
I-5.7	EmbeddingHom . . . . .	86
I-5.8	EqSet . . . . .	86
I-5.9	Equality Test . . . . .	86
I-5.10	EquiIsoDec . . . . .	87
I-5.11	error . . . . .	87
I-5.12	eval . . . . .	88
I-5.13	EvalHilbertFn . . . . .	89
I-5.14	EvalQuasiPoly . . . . .	89
I-5.15	exit . . . . .	89
I-5.16	exponents . . . . .	90
I-5.17	Ext . . . . .	90
I-5.18	ExternalLibs . . . . .	92
<b>I-6</b>	<b>F</b>	<b>93</b>
I-6.1	factor . . . . .	93
I-6.2	FactorAlgExt . . . . .	94
I-6.3	factorial . . . . .	94
I-6.4	FactorMultiplicity . . . . .	94
I-6.5	FGLM5 . . . . .	95
I-6.6	fibonacci . . . . .	95
I-6.7	fields . . . . .	96
I-6.8	first . . . . .	96
I-6.9	FirstNonZero . . . . .	96
I-6.10	FirstNonZeroPosn . . . . .	97
I-6.11	flatten . . . . .	97
I-6.12	FloatApprox . . . . .	98
I-6.13	FloatStr . . . . .	98
I-6.14	floor . . . . .	99
I-6.15	FloorLog2, FloorLog10, FloorLogBase . . . . .	99
I-6.16	FloorSqrt . . . . .	100
I-6.17	for . . . . .	100

I-6.18	<code>foreach</code>	101
I-6.19	<code>format</code>	102
I-6.20	<code>FrbAlexanderDual</code>	102
I-6.21	<code>FrbAssociatedPrimes</code>	103
I-6.22	<code>FrbIrreducibleDecomposition</code>	103
I-6.23	<code>FrbMaximalStandardMonomials</code>	103
I-6.24	<code>FrbPrimaryDecomposition</code>	104
I-6.25	<code>FrobeniusMat</code>	104
I-6.26	<code>func</code>	105
I-6.27	<code>Function</code> [OBSOLETE]	105
I-6.28	<code>functions</code> [OBSOLETE]	105
I-6.29	<code>FVector</code>	106
<b>I-7</b>	<b>G</b>	<b>107</b>
I-7.1	<code>GBasis</code>	107
I-7.2	<code>GBasisTimeout</code>	107
I-7.3	<code>GBM</code>	108
I-7.4	<code>gcd</code>	108
I-7.5	<code>GCDFreeBasis</code>	109
I-7.6	<code>GenericPoints</code>	109
I-7.7	<code>GenRepr</code>	110
I-7.8	<code>gens</code>	110
I-7.9	<code>GensAsCols, GensAsRows</code>	111
I-7.10	<code>Get</code>	112
I-7.11	<code>GetCol</code>	112
I-7.12	<code>GetCols</code>	113
I-7.13	<code>GetEnv</code>	113
I-7.14	<code>GetErrMesg</code>	113
I-7.15	<code>GetRow</code>	114
I-7.16	<code>GetRows</code>	114
I-7.17	<code>GFanContainsPositiveVector</code>	114
I-7.18	<code>GFanGeneratorsOfLinealitySpace</code>	114
I-7.19	<code>GFanGeneratorsOfSpan</code>	115
I-7.20	<code>GFanGetAmbientDimension</code>	115
I-7.21	<code>GFanGetCodimension</code>	115
I-7.22	<code>GFanGetDimension</code>	115
I-7.23	<code>GFanGetDimensionOfLinealitySpace</code>	115
I-7.24	<code>GFanGetFacets</code>	115
I-7.25	<code>GFanGetImpliedEquations</code>	116
I-7.26	<code>GFanGetUniquePoint</code>	116
I-7.27	<code>GFanRelativeInteriorPoint</code>	116
I-7.28	<code>gin</code>	116
I-7.29	<code>GradingDim</code>	117

	I-7.30	GradingMat . . . . .	117
	I-7.31	graeffe . . . . .	118
	I-7.32	GraverBasis . . . . .	118
	I-7.33	GroebnerFanIdeals . . . . .	119
	I-7.34	GroebnerFanReducedGBases . . . . .	119
<b>I-8</b>	<b>H</b>		<b>121</b>
	I-8.1	HasGBasis . . . . .	121
	I-8.2	HColon . . . . .	121
	I-8.3	HermitePoly . . . . .	122
	I-8.4	HGBM . . . . .	122
	I-8.5	hilbert [OBSOLESCE] . . . . .	123
	I-8.6	HilbertBasisKer . . . . .	123
	I-8.7	HilbertFn . . . . .	123
	I-8.8	HilbertMat . . . . .	124
	I-8.9	HilbertPoly . . . . .	124
	I-8.10	HilbertSeries . . . . .	125
	I-8.11	HilbertSeriesMultiDeg . . . . .	126
	I-8.12	HilbertSeriesShifts . . . . .	127
	I-8.13	homog . . . . .	127
	I-8.14	HomogElimMat [OBSOLESCE] . . . . .	128
	I-8.15	HSaturation . . . . .	128
	I-8.16	HVector . . . . .	128
<b>I-9</b>	<b>I</b>		<b>131</b>
	I-9.1	ID [OBSOLETE] . . . . .	131
	I-9.2	ideal . . . . .	131
	I-9.3	IdealAndSeparatorsOfPoints . . . . .	132
	I-9.4	IdealAndSeparatorsOfProjectivePoints . . . . .	133
	I-9.5	IdealOfGBasis . . . . .	134
	I-9.6	IdealOfMinGens . . . . .	135
	I-9.7	IdealOfPoints . . . . .	135
	I-9.8	IdealOfProjectivePoints . . . . .	136
	I-9.9	IdentityMat . . . . .	137
	I-9.10	if . . . . .	137
	I-9.11	ILogBase . . . . .	138
	I-9.12	image [OBSOLESCE] . . . . .	138
	I-9.13	implicit . . . . .	138
	I-9.14	ImplicitHypersurface . . . . .	139
	I-9.15	ImplicitPlot . . . . .	139
	I-9.16	ImplicitPlotOn . . . . .	140
	I-9.17	ImportByRef, ImportByValue . . . . .	141
	I-9.18	in . . . . .	141

I-9.19	incr, decr	141
I-9.20	indent	142
I-9.21	indet	143
I-9.22	IndetIndex	143
I-9.23	IndetName	144
I-9.24	indets	144
I-9.25	IndetSubscripts	145
I-9.26	IndetSymbols	145
I-9.27	InducedHom	146
I-9.28	InitialIdeal	147
I-9.29	insert [OBSOLESCECENT]	147
I-9.30	Interpolate	148
I-9.31	interreduce, interreduced	148
I-9.32	intersection	149
I-9.33	IntersectionList	149
I-9.34	inverse	150
I-9.35	InverseSystem	150
I-9.36	IO.SprintTrunc	150
I-9.37	iroot	151
I-9.38	IsAntiSymmetric	151
I-9.39	IsCommutative	151
I-9.40	IsConstant	152
I-9.41	IsContained	152
I-9.42	IsCoprime	152
I-9.43	IsDefined	153
I-9.44	IsDiagonal	153
I-9.45	IsDivisible	153
I-9.46	IsElem	154
I-9.47	IsEven, IsOdd	154
I-9.48	IsFactorClosed	154
I-9.49	IsField	155
I-9.50	IsFiniteField	155
I-9.51	IsFractionField	155
I-9.52	IsHomog	156
I-9.53	IsIn	157
I-9.54	IsIndet	157
I-9.55	IsInImage	157
I-9.56	IsInjective	158
I-9.57	IsInRadical	158
I-9.58	IsInSubalgebra [OBSOLETE]	159
I-9.59	IsInteger	159
I-9.60	IsIntegralDomain	160
I-9.61	IsInvertible	160

I-9.62	IsIrred . . . . .	160
I-9.63	IsLexSegment . . . . .	160
I-9.64	IsMaximal . . . . .	161
I-9.65	IsMinusOne . . . . .	161
I-9.66	IsNumber [OBSOLETE] . . . . .	161
I-9.67	IsOne . . . . .	162
I-9.68	IsPolyRing . . . . .	162
I-9.69	IsPositiveGrading . . . . .	162
I-9.70	IsPrimary . . . . .	163
I-9.71	IsPrime . . . . .	163
I-9.72	IsProbPrime . . . . .	164
I-9.73	IsPthPower . . . . .	164
I-9.74	IsQQ . . . . .	164
I-9.75	isqrt . . . . .	165
I-9.76	IsQuotientRing . . . . .	165
I-9.77	IsRadical . . . . .	165
I-9.78	IsRational . . . . .	166
I-9.79	IsSqFree . . . . .	166
I-9.80	IsStable . . . . .	166
I-9.81	IsStdGraded . . . . .	167
I-9.82	IsStronglyStable . . . . .	167
I-9.83	IsSubset . . . . .	167
I-9.84	IsSurjective . . . . .	168
I-9.85	IsSymmetric . . . . .	168
I-9.86	IsTerm . . . . .	169
I-9.87	IsTermOrdering . . . . .	169
I-9.88	IsTree5 . . . . .	169
I-9.89	IsTrueGCDDomain . . . . .	170
I-9.90	IsZero . . . . .	170
I-9.91	IsZeroCol, IsZeroRow . . . . .	171
I-9.92	IsZeroDim . . . . .	171
I-9.93	IsZeroDivisor . . . . .	172
I-9.94	IsZZ . . . . .	172
I-9.95	It . . . . .	172
<b>I-10</b>	<b>J</b>	<b>175</b>
I-10.1	jacobian . . . . .	175
I-10.2	JanetBasis . . . . .	175
<b>I-11</b>	<b>K</b>	<b>177</b>
I-11.1	ker . . . . .	177
<b>I-12</b>	<b>L</b>	<b>179</b>
I-12.1	LaguerrePoly . . . . .	179

I-12.2	last	179
I-12.3	latex	180
I-12.4	LC	180
I-12.5	lcm	181
I-12.6	len	181
I-12.7	LexMat	182
I-12.8	LexSegmentIdeal	182
I-12.9	LF	183
I-12.10	LinearSimplify	183
I-12.11	LinKer	183
I-12.12	LinKerBasis	184
I-12.13	LinKerModP [OBSOLETE]	185
I-12.14	LinSol [OBSOLETE]	185
I-12.15	LinSolve	185
I-12.16	LM	186
I-12.17	log [OBSOLESCENT]	186
I-12.18	LogCardinality	187
I-12.19	LPosn	187
I-12.20	LPP	188
I-12.21	LT	188
<b>I-13</b>	<b>M</b>	<b>191</b>
I-13.1	MakeCheck	191
I-13.2	MakeMatByRows, MakeMatByCols	191
I-13.3	MakeSet	192
I-13.4	MakeTerm	192
I-13.5	MakeTermOrd	192
I-13.6	MantissaAndExponent10	193
I-13.7	MantissaAndExponent2	194
I-13.8	Manual	194
I-13.9	MapDown [OBSOLETE]	195
I-13.10	matrix	195
I-13.11	max	196
I-13.12	MaxBy	196
I-13.13	MayerVietorisTreeN1	197
I-13.14	min	197
I-13.15	MinBy	198
I-13.16	MinGens	198
I-13.17	MinGensGeneral [OBSOLESCENT]	199
I-13.18	minimalize [OBSOLESCENT]	199
I-13.19	minimalized [OBSOLESCENT]	199
I-13.20	MinimalPresentation	199
I-13.21	minors	200

I-13.22	MinPoly . . . . .	200
I-13.23	MinPolyQuot, MinPolyQuotDef, MinPolyQuotElim, MinPolyQuotMat . . . . .	200
I-13.24	MinPowerInIdeal . . . . .	201
I-13.25	MinSubsetOfGens . . . . .	202
I-13.26	mod . . . . .	202
I-13.27	Mod2Rat [OBSOLETE] . . . . .	203
I-13.28	ModuleElem . . . . .	203
I-13.29	ModuleOf . . . . .	203
I-13.30	monic . . . . .	204
I-13.31	monomials . . . . .	204
I-13.32	MonsInIdeal . . . . .	205
I-13.33	MSatLinSolve . . . . .	205
I-13.34	MultiplicationMat . . . . .	206
I-13.35	multiplicity . . . . .	207
<b>I-14</b>	<b>N</b>	<b>209</b>
I-14.1	NewFractionField . . . . .	209
I-14.2	NewFreeModule . . . . .	209
I-14.3	NewId [OBSOLETE] . . . . .	210
I-14.4	NewLine [OBSOLESCE] . . . . .	210
I-14.5	NewList . . . . .	210
I-14.6	NewMat . . . . .	211
I-14.7	NewMatFilled . . . . .	211
I-14.8	NewPolyRing . . . . .	212
I-14.9	NewQuotientRing . . . . .	212
I-14.10	NewRingFp . . . . .	213
I-14.11	NewRingTwinFloat . . . . .	213
I-14.12	NewWeylAlgebra . . . . .	214
I-14.13	NewZZmod . . . . .	214
I-14.14	NextPrime, PrevPrime . . . . .	215
I-14.15	NextProbPrime, PrevProbPrime . . . . .	215
I-14.16	NF . . . . .	215
I-14.17	NFsAreZero [OBSOLETE] . . . . .	216
I-14.18	NmzComputation . . . . .	216
I-14.19	NmzDiagInvariants . . . . .	217
I-14.20	NmzEhrhartRing . . . . .	217
I-14.21	NmzFiniteDiagInvariants . . . . .	218
I-14.22	NmzHilbertBasis . . . . .	218
I-14.23	NmzIntClosureMonIdeal . . . . .	219
I-14.24	NmzIntClosureToricRing . . . . .	219
I-14.25	NmzIntersectionValRings . . . . .	220
I-14.26	NmzNormalToricRing . . . . .	220
I-14.27	NmzSetVerbosityLevel . . . . .	220

I-14.28	NmzTorusInvariants	221
I-14.29	NmzVerbosityLevel	221
I-14.30	NonZero	221
I-14.31	not	222
I-14.32	NR	222
I-14.33	num	223
I-14.34	NumCols	223
I-14.35	NumCompts	223
I-14.36	NumGens	224
I-14.37	NumIndets	224
I-14.38	NumPartitions	224
I-14.39	NumRows	225
I-14.40	NumTerms	225
<b>I-15</b>	<b>O</b>	<b>227</b>
I-15.1	one	227
I-15.2	OpenIFile	227
I-15.3	OpenIString	228
I-15.4	OpenLog	228
I-15.5	OpenOFile	229
I-15.6	OpenOString	230
I-15.7	OpenSocket	230
I-15.8	Option [OBSOLETE]	231
I-15.9	or	231
I-15.10	OrdMat	231
<b>I-16</b>	<b>P</b>	<b>233</b>
I-16.1	packages	233
I-16.2	panel [OBSOLETE]	233
I-16.3	panels [OBSOLETE]	233
I-16.4	partitions	234
I-16.5	permutations	234
I-16.6	PerpIdealOfForm	234
I-16.7	pfaffian	235
I-16.8	PkgName	235
I-16.9	PlotPoints	236
I-16.10	PlotPointsOn	236
I-16.11	poincare [OBSOLESCENT]	237
I-16.12	PoincareMultiDeg [OBSOLETE]	237
I-16.13	PoincareShifts [OBSOLETE]	237
I-16.14	PolyAlgebraHom	237
I-16.15	PolyRingHom	238
I-16.16	PowerMod	238

I-16.17	PreImage [OBSOLESCECENT]	239
I-16.18	preimage0	239
I-16.19	PreprocessPts	239
I-16.20	PrevPrime, PrevProbPrime	240
I-16.21	PrevPrime, PrevProbPrime	241
I-16.22	PrimaryDecomposition	241
I-16.23	PrimaryDecomposition0	241
I-16.24	PrimaryDecompositionCore0	242
I-16.25	PrimaryDecompositionGTZ0	243
I-16.26	PrimaryHilbertSeries	243
I-16.27	PrimaryPoincare [OBSOLESCECENT]	244
I-16.28	PrimitiveRoot	244
I-16.29	print	245
I-16.30	print on	245
I-16.31	PrintBettiDiagram	246
I-16.32	PrintBettiMatrix	246
I-16.33	PrintBettiNumbers	247
I-16.34	println	247
I-16.35	PrintRes	248
I-16.36	PrintSectionalMatrix	248
I-16.37	product	249
I-16.38	protect	250
I-16.39	PthRoot	250
<b>I-17</b>	<b>Q</b>	<b>251</b>
I-17.1	QQ	251
I-17.2	QQEmbeddingHom	251
I-17.3	quit	252
I-17.4	QuotientBasis	252
I-17.5	QuotientBasisSorted	253
I-17.6	QuotientingHom	253
I-17.7	QZP	253
<b>I-18</b>	<b>R</b>	<b>255</b>
I-18.1	radical	255
I-18.2	RadicalOfUnmixed	255
I-18.3	random	256
I-18.4	randomize [OBSOLETE]	256
I-18.5	randomized [OBSOLETE]	256
I-18.6	RandomSparseNonSing01Mat	257
I-18.7	RandomSubset	257
I-18.8	RandomSubsetIndices	258
I-18.9	RandomTuple	258

I-18.10	<a href="#">RandomTupleIndices</a>	258
I-18.11	<a href="#">RandomUnimodularMat</a>	259
I-18.12	<a href="#">rank</a> [OBSOLESCENT]	259
I-18.13	<a href="#">RationalAffinePoints</a>	259
I-18.14	<a href="#">RationalProjectivePoints</a>	260
I-18.15	<a href="#">RationalSolve</a>	260
I-18.16	<a href="#">RatReconstructByContFrac</a> , <a href="#">RatReconstructByLattice</a>	260
I-18.17	<a href="#">RatReconstructPoly</a>	261
I-18.18	<a href="#">RatReconstructWithBounds</a>	262
I-18.19	<a href="#">ReadExpr</a> [OBSOLESCENT]	262
I-18.20	<a href="#">RealRootRefine</a>	262
I-18.21	<a href="#">RealRoots</a>	263
I-18.22	<a href="#">RealRootsApprox</a>	264
I-18.23	<a href="#">record</a>	264
I-18.24	<a href="#">record field selector</a>	265
I-18.25	<a href="#">ReducedGBasis</a>	265
I-18.26	<a href="#">ref</a>	265
I-18.27	<a href="#">RefineGCDFreeBasis</a>	266
I-18.28	<a href="#">reg</a>	266
I-18.29	<a href="#">RegularityIndex</a>	267
I-18.30	<a href="#">RelNotes</a>	268
I-18.31	<a href="#">ReloadMan</a>	268
I-18.32	<a href="#">remove</a>	268
I-18.33	<a href="#">repeat</a>	269
I-18.34	<a href="#">res</a>	269
I-18.35	<a href="#">reseed</a>	270
I-18.36	<a href="#">Reset</a> [OBSOLETE]	271
I-18.37	<a href="#">ResetPanels</a> [OBSOLETE]	271
I-18.38	<a href="#">resultant</a>	271
I-18.39	<a href="#">return</a>	271
I-18.40	<a href="#">reverse, reversed</a>	272
I-18.41	<a href="#">RevLexMat</a>	272
I-18.42	<a href="#">rgin</a>	273
I-18.43	<a href="#">RingElem</a>	273
I-18.44	<a href="#">RingEnv</a> [OBSOLETE]	274
I-18.45	<a href="#">RingID</a>	274
I-18.46	<a href="#">RingOf</a>	275
I-18.47	<a href="#">RingQQ</a>	276
I-18.48	<a href="#">RingQQt</a>	276
I-18.49	<a href="#">RingSet</a> [OBSOLETE]	276
I-18.50	<a href="#">RingsOf</a>	277
I-18.51	<a href="#">RingZZ</a>	277
I-18.52	<a href="#">rk</a>	278

I-18.53	RMap [OBSOLESCE	278
I-18.54	RootBound	278
I-18.55	round	279
I-18.56	RowMat	279
<b>I-19</b>	<b>S</b>	<b>281</b>
I-19.1	saturate	281
I-19.2	ScalarProduct	281
I-19.3	ScientificStr	282
I-19.4	SectionalMatrix	282
I-19.5	seed [OBSOLETE]	283
I-19.6	SeparatorsOfPoints	283
I-19.7	SeparatorsOfProjectivePoints	284
I-19.8	SetCol	285
I-19.9	SetRow	285
I-19.10	SetStackSize	286
I-19.11	SetVerbosityLevel	286
I-19.12	shape	287
I-19.13	sign	287
I-19.14	SimplestBinaryRatBetween	288
I-19.15	SimplestRatBetween	288
I-19.16	SimplexInfo	288
I-19.17	SimplicialHomology	289
I-19.18	size [OBSOLETE]	290
I-19.19	skip	290
I-19.20	SmallestNonDivisor	290
I-19.21	SmoothFactor	291
I-19.22	sort	291
I-19.23	SortBy	292
I-19.24	sorted	292
I-19.25	SortedBy	293
I-19.26	source	293
I-19.27	SourceRegion	294
I-19.28	spaces	294
I-19.29	sprint	294
I-19.30	SqFreeFactor	295
I-19.31	StableBBasis5	295
I-19.32	StableIdeal	296
I-19.33	StagedTrees	297
I-19.34	StarPrint, StarSprint	297
I-19.35	starting	298
I-19.36	StdBasis	298
I-19.37	StdDegLexMat	299

I-19.38	StdDegRevLexMat	299
I-19.39	StronglyStableIdeal	300
I-19.40	SubalgebraHom	300
I-19.41	SubalgebraMap [OBSOLETE]	300
I-19.42	SubalgebraRepr [OBSOLESCENT]	301
I-19.43	submat	301
I-19.44	submodule	301
I-19.45	SubmoduleCols, SubmoduleRows	302
I-19.46	SubmoduleOfMinGens	302
I-19.47	subsets	303
I-19.48	subst	303
I-19.49	sum	304
I-19.50	support	304
I-19.51	swap	305
I-19.52	SwapCols	305
I-19.53	SwapRows	306
I-19.54	sylvester	306
I-19.55	SymbolRange	307
I-19.56	SymmetricPolys	307
I-19.57	syz	307
I-19.58	SyzOfGens	308
<b>I-20</b>	<b>T</b>	<b>311</b>
I-20.1	tag	311
I-20.2	tagged	311
I-20.3	tail	312
I-20.4	TensorMat	312
I-20.5	TgCone	312
I-20.6	TimeFrom	313
I-20.7	TimeOfDay	313
I-20.8	TmpChainCanonicalHom	313
I-20.9	TmpNBM [OBSOLESCENT]	314
I-20.10	TopLevel	314
I-20.11	TopLevelFunctions	315
I-20.12	toric	315
I-20.13	transposed	316
I-20.14	try	316
I-20.15	tuples	317
I-20.16	Tutorial	317
I-20.17	Tutorial-01: manual	318
I-20.18	Tutorial-02: variables, assignment	318
I-20.19	Tutorial-03: arithmetic operators	319
I-20.20	Tutorial-04: printing	320

I-20.21	Tutorial-05: lists . . . . .	320
I-20.22	Tutorial-06: rings, polynomials, use command . . . . .	321
I-20.23	Tutorial-11: homomorphisms . . . . .	322
I-20.24	type . . . . .	322
<b>I-21</b>	<b>U</b>	<b>325</b>
I-21.1	UnivariateIndetIndex . . . . .	325
I-21.2	UniversalGBasis . . . . .	325
I-21.3	unprotect . . . . .	326
I-21.4	Unset [OBSOLETE] . . . . .	326
I-21.5	untagged . . . . .	326
I-21.6	use . . . . .	327
<b>I-22</b>	<b>V</b>	<b>329</b>
I-22.1	valuation [OBSOLETE] . . . . .	329
I-22.2	VerbosityLevel . . . . .	329
I-22.3	VersionInfo . . . . .	330
<b>I-23</b>	<b>W</b>	<b>331</b>
I-23.1	wdeg . . . . .	331
I-23.2	WeightsMatrix [OBSOLESCENT] . . . . .	331
I-23.3	while . . . . .	332
I-23.4	WithoutNth . . . . .	332
I-23.5	WLog [OBSOLETE] . . . . .	332
<b>I-24</b>	<b>X</b>	<b>335</b>
I-24.1	XelMat . . . . .	335
<b>I-25</b>	<b>Z</b>	<b>337</b>
I-25.1	zero . . . . .	337
I-25.2	ZeroMat . . . . .	337
I-25.3	ZPQ . . . . .	338
I-25.4	ZZ . . . . .	338
<b>II</b>	<b>The CoCoA Programming Language</b>	<b>341</b>
<b>II-1</b>	<b>Introduction to CoCoA Programming</b>	<b>343</b>
II-1.1	An Overview of CoCoA Programming . . . . .	343
II-1.2	All CoCoA commands . . . . .	343
<b>II-2</b>	<b>Language Elements</b>	<b>345</b>
II-2.1	Character Set and Special Symbols . . . . .	345
II-2.2	Identifiers . . . . .	345
II-2.3	Reserved Names . . . . .	346
II-2.4	Comments . . . . .	346

<b>II-3</b>	<b>Operators</b>	<b>347</b>
II-3.1	CoCoA Operators: introduction . . . . .	347
II-3.2	Algebraic Operators . . . . .	347
II-3.3	Relational Operators . . . . .	348
II-3.4	Selection Operators . . . . .	348
II-3.5	Range Operator . . . . .	348
<b>II-4</b>	<b>Evaluation and Assignment</b>	<b>349</b>
II-4.1	Evaluation . . . . .	349
II-4.2	Assignment . . . . .	349
<b>II-5</b>	<b>Flow Control: Conditional Statements and Loops</b>	<b>351</b>
II-5.1	Commands and Functions for Branching . . . . .	351
II-5.2	Commands and Functions for Loops . . . . .	351
<b>II-6</b>	<b>Verbosity and interrupt</b>	<b>353</b>
II-6.1	Introduction to verbosity and interrupt . . . . .	353
II-6.2	Commands and Functions implementing Verbosity . . . . .	353
II-6.3	Commands and Functions implementing interruption . . . . .	353
<b>II-7</b>	<b>Input/Output</b>	<b>355</b>
II-7.1	Introduction to IO . . . . .	355
II-7.2	Standard IO . . . . .	355
II-7.3	File IO . . . . .	355
II-7.4	String IO . . . . .	356
II-7.5	Commands and Functions for IO . . . . .	357
<b>II-8</b>	<b>CoCoA Packages</b>	<b>359</b>
II-8.1	Introduction to Packages . . . . .	359
II-8.2	First Example of a Package . . . . .	359
II-8.3	Package Essentials . . . . .	360
II-8.4	Global Aliases . . . . .	360
II-8.5	Sharing Your Package . . . . .	360
II-8.6	Commands and Functions for Packages . . . . .	360
II-8.7	Supported Packages . . . . .	361
II-8.8	Galois Package . . . . .	361
II-8.9	Integer Programming . . . . .	361
II-8.10	Algebra of Invariants . . . . .	361
II-8.11	Special Varieties . . . . .	361
II-8.12	Statistics . . . . .	362
II-8.13	Geometrical Theorem-Proving . . . . .	362
II-8.14	Typevectors . . . . .	362
II-8.15	Conductor . . . . .	362
II-8.16	Matrix Normal Form . . . . .	363

II-8.17	CantStop . . . . .	363
II-8.18	Control . . . . .	363
<b>II-9</b>	<b>Linked libraries</b>	<b>365</b>
II-9.1	CoCoALib . . . . .	365
II-9.2	GMP . . . . .	365
II-9.3	GSL . . . . .	365
II-9.4	Frobby . . . . .	365
II-9.5	MathSAT . . . . .	365
II-9.6	Normaliz . . . . .	365
<b>II-10</b>	<b>Migrating from CoCoA-4 and keeping up-to-date</b>	<b>367</b>
II-10.1	Changes in the CoCoA language . . . . .	367
II-10.2	Recent changes in the CoCoA-5 language . . . . .	368
II-10.3	Obsolete and obsolescent functions . . . . .	368
<b>III</b>	<b>CoCoA datatypes</b>	<b>371</b>
<b>III-1</b>	<b>BOOL</b>	<b>373</b>
III-1.1	Introduction to BOOL . . . . .	373
III-1.2	Commands and Functions for BOOL . . . . .	373
III-1.3	Commands and Functions returning BOOL . . . . .	373
<b>III-2</b>	<b>INT</b>	<b>375</b>
III-2.1	Introduction to INT . . . . .	375
III-2.2	Commands and Functions for INT . . . . .	375
III-2.3	Commands and Functions returning INT . . . . .	378
<b>III-3</b>	<b>RAT</b>	<b>381</b>
III-3.1	Introduction to RAT . . . . .	381
III-3.2	Commands and Functions for RAT . . . . .	381
III-3.3	Commands and Functions returning RAT . . . . .	382
<b>III-4</b>	<b>STRING</b>	<b>383</b>
III-4.1	String Literals . . . . .	383
III-4.2	String Operations . . . . .	383
III-4.3	Commands and Functions for STRING . . . . .	384
III-4.4	Commands and Functions returning STRING . . . . .	384
<b>III-5</b>	<b>LIST</b>	<b>387</b>
III-5.1	Introduction to LIST . . . . .	387
III-5.2	List Constructors . . . . .	388
III-5.3	Commands and Functions for LIST . . . . .	388
III-5.4	Commands and Functions returning LIST . . . . .	391

<b>III-6</b>	<b>RECORD</b>	<b>395</b>
III-6.1	Introduction to RECORD . . . . .	395
III-6.2	Commands and Functions for RECORD . . . . .	396
III-6.3	Commands and Functions returning RECORD . . . . .	396
<b>III-7</b>	<b>FUNCTION</b>	<b>399</b>
III-7.1	Introduction to FUNCTION . . . . .	399
III-7.2	FUNCTIONs are first class objects . . . . .	399
III-7.3	Commands and Functions for FUNCTION . . . . .	399
III-7.4	Commands and Functions returning FUNCTION . . . . .	400
<b>III-8</b>	<b>TYPE</b>	<b>401</b>
III-8.1	Commands and Functions for TYPE . . . . .	401
III-8.2	Commands and Functions returning TYPE . . . . .	401
<b>III-9</b>	<b>RING</b>	<b>403</b>
III-9.1	Introduction to RING . . . . .	403
III-9.2	Polynomial Rings . . . . .	403
III-9.3	Coefficient Rings . . . . .	404
III-9.4	Indeterminates . . . . .	404
III-9.5	Term Orderings . . . . .	404
III-9.6	Module Orderings . . . . .	405
III-9.7	Quotient Rings . . . . .	406
III-9.8	Commands and Functions for RING . . . . .	406
III-9.9	Commands and Functions returning RING . . . . .	407
<b>III-10</b>	<b>RINGHOM</b>	<b>409</b>
III-10.1	Introduction to RINGHOM . . . . .	409
III-10.2	Composition of RINGHOM . . . . .	409
III-10.3	Commands and Functions for RINGHOM . . . . .	410
III-10.4	Commands and Functions returning RINGHOM . . . . .	410
<b>III-11</b>	<b>RINGELEM</b>	<b>411</b>
III-11.1	Introduction to RINGELEM . . . . .	411
III-11.2	Evaluation of Polynomials . . . . .	412
III-11.3	Commands and Functions for RINGELEM . . . . .	412
III-11.4	Commands and Functions returning RINGELEM . . . . .	414
<b>III-12</b>	<b>IDEAL</b>	<b>417</b>
III-12.1	Commands and Functions for IDEAL . . . . .	417
III-12.2	Commands and Functions returning IDEAL . . . . .	419
<b>III-13</b>	<b>MAT</b>	<b>421</b>
III-13.1	Introduction to MAT . . . . .	421
III-13.2	Commands and Functions for MAT . . . . .	421

III-13.3	Commands and Functions returning MAT . . . . .	423
<b>III-14</b>	<b>MODULE</b>	<b>425</b>
III-14.1	Commands and Functions for MODULE . . . . .	425
III-14.2	Commands and Functions returning MODULE . . . . .	426
<b>III-15</b>	<b>MODULEELEM</b>	<b>427</b>
III-15.1	Introduction to MODULEELEM . . . . .	427
III-15.2	Commands and Functions for MODULEELEM . . . . .	427
III-15.3	Commands and Functions returning MODULEELEM . . . . .	428
<b>III-16</b>	<b>Creating new types</b>	<b>429</b>
III-16.1	Tagging an Object . . . . .	429
III-16.2	Printing a Tagged Object . . . . .	429
III-16.3	Commands and Functions for Tags . . . . .	430

## Part I

# Alphabetical List of Commands



# Chapter I-0

## Special Characters

### I-0.1 operators, shortcuts

	syntax
A := B	A:variable, B:OBJECT
A = B	A,B:OBJECT
A <> B	A,B:OBJECT
A < B	A,B:OBJECT
A >< B	A,B: LIST
A..B	A,B: INT or A,B:indeterminates
[...]	LIST
[.. ..]	LIST
L[N]	N: INT, L: LIST
S[N]	N: INT, S: STRING
M[i,j]	i,j: INT, M: MAT
R.F	R:RECORD and F field name
R ::= ...	R:variable
M : N	M, N: MODULE or IDEAL
R/I	R: RING, I: IDEAL
***E***	E:expression
?S	S: STRING
<< S	S: STRING

### Description

“A := B;” compute “B” then assign the result to “A” “A = B” test whether “A” and “B” are equal (see “Comparison Operators” (I-3.34 pg.59)) “A <> B” test whether “A” and “B” are not equal (see “Comparison Operators” (I-3.34 pg.59)) “A < B” test whether “A” is smaller than “B” (see “Comparison Operators” (I-3.34 pg.59)) “A >< B” equivalent to “CartesianProduct(A, B)”, “CartesianProductList([A, B])” is the “Range Operator” (II-3.5 pg.348) (see “List Constructors” (III-5.2 pg.388)) “[...]” build a new list (see “List Constructors” (III-5.2 pg.388)) “[..|..]” build a new list (see “List Constructors” (III-5.2 pg.388)) “L[N]” access “N”-th entry of list “L” (indexes start from 1) “S[N]” access “N”-th char of string “L” (indexes start from 1) “M[i,j]” access entry “i,j” in matrix “M” “R.F” “record field selector” (I-18.24 pg.265) for field named “F” of record “R” “R ::= ” for the special ring syntax (“NewPolyRing” (I-14.8 pg.212)) “M : N” equivalent to “colon(M, N)” “R/I” equivalent to “NewQuotientRing(R,I)” “\*\*\*E\*\*\*” interpret “E” in “CoCoA-4 mode” (I-3.18 pg.52) “\$<< S\$” OBSOLESCEnt equivalent to “source(S)”

“? string” prints the manual page for “string”, or a list of matching manual pages

**See Also:** colon(I-3.30 pg.57), List Constructors(III-5.2 pg.388), CartesianProduct, CartesianProductList(I-3.5 pg.47), NewPolyRing(I-14.8 pg.212), NewQuotientRing(I-14.9 pg.212), record field selector(I-18.24 pg.265), Range Operator(II-3.5 pg.348), source(I-19.26 pg.293), Manual(I-13.8 pg.194), Character Set and Special Sym-

bols([II-2.1](#) pg.[345](#)), CoCoA Operators([??](#) pg.[??](#))

# Chapter I-1

## A

### I-1.1 abs

syntax

```
abs(N: INT): INT
abs(N: RAT): RAT
abs(N: RINGELEM): RINGELEM
```

#### Description

This function returns the absolute value of “N”. If “N” is a “RINGELEM” then it must belong to an ordered ring.

example

```
/**/ abs(-3);
3

/**/ abs(-2/3);
2/3
```

### I-1.2 adj

syntax

```
adj(M: MAT): MAT
```

#### Description

This function returns the classical adjoint of the square matrix “M”.

example

```
/**/ use R := QQ[t,x,y,z];
/**/ adj(mat([[x,y,z],[t,y,x],[x,x^2,x*y]]));
matrix( /*RingWithID(44, "QQ[t,x,y,z]")*/
  [[-x^3 +x*y^2, -x*y^2 +x^2*z, x*y -y*z],
  [-t*x*y +x^2, x^2*y -x*z, -x^2 +t*z],
  [t*x^2 -x*y, -x^3 +x*y, -t*y +x*y]])

/**/ Z5 := NewRingFp(5);
/**/ adj(matrix(Z5, [[1,2],[3,1]]));
matrix( /*FFp(5)*/
  [[1, -2],
  [2, 1]])
```

**See Also:** `inverse`([I-9.34](#) pg.150)

## I-1.3 AffHilbert [OBSOLESCENT]

syntax

[OBSOLESCENT]

### Description

Renamed “AffHilbertFn” ([I-1.4](#) pg.28).

## I-1.4 AffHilbertFn

syntax

```
AffHilbertFn(R: RING): TAGGED("$hp.Hilbert")
AffHilbertFn(R: RING, N: INT): INT
```

### Description

The first form of this function computes the affine Hilbert function for “R”. The second form computes the “N”-th value of the affine Hilbert function. The weights of the indeterminates of “R” must all be 1. For evaluating of the Hilbert function repeatedly, use the function “EvalHilbertFn” ([I-5.13](#) pg.89) instead of “AffHilbertFn(R, N)” in order to speed up execution.

The coefficient ring must be a field.

example

```
/**/ use R ::= QQ[x,y,z];
/**/ AffHilbertFn(R/ideal(z^4-1, x*z^4-y-3));
H(0) = 1
H(1) = 3
H(t) = 4t - 2 for t >= 2
```

**See Also:** `AffHilbertSeries`([I-1.5](#) pg.28), `EvalHilbertFn`([I-5.13](#) pg.89), `HilbertPoly`([I-8.9](#) pg.124), `HVector`([I-8.16](#) pg.128), `HilbertSeries`([I-8.10](#) pg.125)

## I-1.5 AffHilbertSeries

syntax

```
AffHilbertSeries(TAGGED("Quotient")):TAGGED("$hp.PSeries")
```

### Description

This function computes the affine Hilbert-Poincare series of M. The grading must be a positive  $Z^1$ -grading (i.e. “GradingMat” ([I-7.30](#) pg.117) must have a single row with positive entries), and the ordering must be degree compatible. In the standard case, i.e. the weights of all indeterminates are 1, the result is simplified so that the power appearing in the denominator is the dimension of M + 1.

It used to be called “AffPoincare [OBSOLESCENT]” ([I-1.6](#) pg.29).

NOTES:

(i) the coefficient ring must be a field.

(ii) these functions produce tagged objects: they cannot safely be (non-)equality to other values.

For further details on affine Hilbert functions see the book: Kreuzer, Robbiano "Computer Commutative Algebra II", Section 5.6.

example

```
/**/ use R ::= QQ[x,y,z];
/**/ AffHilbertSeries(R/ideal(z^4-1, x*z^4-y-3));
(1 +x +x^2 +x^3) / (1-x)^2
```

**See Also:** AffHilbertFn(I-1.4 pg.28), HilbertSeries(I-8.10 pg.125)

## I-1.6 AffPoincare [OBSOLESCENT]

syntax

[OBSOLESCENT]

### Description

Renamed "AffHilbertSeries" (I-1.5 pg.28).

## I-1.7 alias

syntax

```
alias B_1,...,B_r
```

where each B\_i is a ‘‘{\it binding}’’ of the form: Identifier := \$PackageName

### Description

This function is for declaring both global and local aliases for package names. Recall that package names are meant to be long in order to avoid conflicts between the names of functions that are read into a CoCoA session. However, it is inconvenient to have to type out the long package name when referencing a function. So the user chooses an alias to take the place of the package name; the alias is just a means to avoid typing.

1. Global aliases. To avoid typing the full package name as a prefix to package functions, one may declare a short global alias during a CoCoA session. A list of the global aliases is produced by the function "aliases" (I-1.8 pg.30). For examples, see the chapter on packages in the manual, in particular the section, "Global Aliases" (II-8.4 pg.360). Online, enter "?global aliases".

2. Local aliases. A local alias has the same syntax as a global alias, however it appears inside a package definition. The local aliases work only inside the package and do not conflict with any global aliases already defined. In fact, in order to avoid conflicts, global aliases are not recognized within a package. For examples, again look in the chapter for packages.

example

```
/**/ alias LL := $abcd;
/**/ aliases();

Coclib      = $coclib
Approx      = $approx
(...)
TP          = $contrib/thmproving
TV          = $contrib/typevectors
LL          = $abcd
```

**See Also:** [aliases\(I-1.8 pg.30\)](#), [Introduction to Packages\(II-8.1 pg.359\)](#)

## I-1.8 aliases

syntax

```
aliases():TAGGED("aliases")
```

### Description

This function prints a list of global aliases for packages. Aliases are formed with the function “[alias](#)” ([I-1.7 pg.29](#)).

example

```
/**/ alias LL := $abcd;
/**/ aliases();

Coclib      = $coclib
Approx      = $approx
(...)
TP          = $contrib/thmproving
TV          = $contrib/typevectors
LL          = $abcd
```

**See Also:** [alias\(I-1.7 pg.29\)](#), [Introduction to Packages\(II-8.1 pg.359\)](#)

## I-1.9 AllReducedGroebnerBases [OBSOLESCENT]

syntax

```
GroebnerFanIdeals(I: IDEAL): LIST of IDEAL
```

### Description

Renamed as “[GroebnerFanIdeals](#)” ([I-7.33 pg.119](#)).

**See Also:** [GroebnerFanIdeals\(I-7.33 pg.119\)](#)

## I-1.10 AlmostQR

syntax

```
AlmostQR(M: MAT): RECORD
```

### Description

This function computes the decomposition of the matrix into an orthogonal and an upper triangular matrix with 1 on the diagonal. [“*orthogonal*” meaning that  $Q^T * Q$  is a diagonal matrix]

The auxiliary (possibly slow!) function “[Mat.SimplifySquareFactorsInAQR](#)” modifies “Q” and “R” in the decomposition so that the entries of the diagonal matrix  $Q^T * Q$  are squarefree rationals.

example

```
/**/ M := matrix([ [4, -2, 3], [3, 2, -2], [0, 0, 3] ]);
/**/ Dec := AlmostQR(M);
/**/ Dec;
```

```

record[Q := matrix(QQ,
  [[4, -42/25, 0],
   [3, 56/25, 0],
   [0, 0, 3]])
, R := matrix(QQ,
  [[1, -2/25, 6/25],
   [0, 1, -17/14],
   [0, 0, 1]])
]

/**/ $mat.SimplifySquareFactorsInAQR(ref Dec);
/**/ Dec;
record[Q := matrix(QQ,
  [[4/5, -3/5, 0],
   [3/5, 4/5, 0],
   [0, 0, 1]])
, R := matrix(QQ,
  [[5, -2/5, 6/5],
   [0, 14/5, -17/5],
   [0, 0, 3]])
, SqDiag := [1, 1, 1]]

```

**See Also:** [Matrix Normal Form](#)([II-8.16](#) pg.363)

## I-1.11 and

syntax

```
A and B      (where A, B: BOOL, return BOOL)
```

### Description

This operator represents the logical conjunction of “A” and “B”. CoCoA first evaluates “A”; if that gives “false” then the result is “false”, and “B” is not evaluated. Otherwise if “A” gives “true” then “B” is evaluated, and its value is the final result.

example

```

/**/ A := -1;
/**/ A >= 0 and FloorSqrt(A) < 10; --> calls FloorSqrt only if A >= 0
false

```

**See Also:** [or](#)([I-15.9](#) pg.231), [not](#)([I-14.31](#) pg.222)

## I-1.12 append

syntax

```
append(ref L: LIST, E: OBJECT)
```

### Description

Append the object “E” to the list “L”; this call returns nothing!

NOTE: the old CoCoA-4 syntax “Append(L, E)” is still allowed, but produces a warning; replace the call by “append(ref L, E)”.

example

```

/**/ use R := QQ[t,x,y,z];
/**/ L := [1,2,3];
/**/ append(ref L, 4);
/**/ L;
[1, 2, 3, 4]

```

**See Also:** [ref\(I-18.26 pg.265\)](#), [concat\(I-3.38 pg.60\)](#), [ConcatLists\(I-3.43 pg.62\)](#), [remove\(I-18.32 pg.268\)](#)

## I-1.13 apply

syntax

```

apply(phi: RINGHOM, X: RINGELEM): RINGELEM
apply(phi: RINGHOM, X: LIST): LIST
apply(phi: RINGHOM, X: MAT): MAT

```

### Description

Apply homomorphism “phi” to all elements in second argument “X” (“RINGELEM”, “LIST”, or “MAT”)

When “X” is of type “RINGELEM” this is equivalent to the natural syntax “phi(X)”.

example

```

/**/ use R := QQ[x,y,z];
/**/ S := QQ[x[1..3]];
/**/ phi := PolyAlgebraHom(R, S, indets(S));
/**/ apply(phi, [x^2-y, z-2]);
[x[1]^2 -x[2], x[3] -2]

/**/ apply(phi, x^2-y); -- same as phi(x^2-y)
x[1]^2 -x[2]
/**/ phi(x^2-y);
x[1]^2 -x[2]

```

**See Also:** [PolyAlgebraHom\(I-16.14 pg.237\)](#), [CanonicalHom\(I-3.3 pg.46\)](#)

## I-1.14 ApproxPointsNBM

syntax

```

ApproxPointsNBM(P: RING, Pts: MAT, Toler: MAT): RECORD

```

### Description

This function returns a record containing four fields: namely, “QuotientBasis”, “BBasis”, “AlmostVanishing” and “StableBBasisFound”.

The field “QuotientBasis” contains a factor-closed set of power-products, and the field “AlmostVanishing” contains a list of “*almost vanishing*” polynomials. If the cardinality of the field “QuotientBasis” is equal to the number of points, it is in fact a “*quotient basis*” of the ideal of points, and in this case a “*border basis*” founded on it is also returned in the field “BBasis” and the field “StableBBasisFound” is set to “true” (otherwise it is “false”).

The first argument is a list of points in k-dimensional space, and the second argument is the list of k positive tolerances (one for each dimension). So that the answer can be represented, the current ring must have at least k indeterminates; the term ordering is ignored as it plays no role in determining the border basis.

Verbosity range 80-95 (80, 90, 95).

Thanks to John Abbott and Maria-Laura Torrente for the implementation.

For a full description of the algorithms we refer to the paper C.Fassino “*Almost Vanishing Polynomials for Sets of Limited Precision Points*” (Journal of Symbolic Computation 45 (2010), 19–37).

example

```

/**/ P := QQ[x,y];
/**/ Eps := [0.1, 0.1];
/**/ Points := [[10, 0], [-10, 0], [0, 10], [0, -10], [7, 7], [-7, -7]];
/**/ indent(ApproxPointsNBM(P, mat(Points), RowMat(Eps)));
record[
  AlmostVanishing := [x^2 +(2/49)*x*y +y^2 -100,
                      x*y^2 +(49/51)*y^3 +(-4900/51)*y, y^4 +51*x*y -100*y^2],
  BBasis := [x^2 +(2/49)*x*y +y^2 -100, x*y^2 +(49/51)*y^3 +(-4900/51)*y,
             x^2*y +(49/51)*y^3 +(-4900/51)*y, y^4 +51*x*y -100*y^2, x*y^3 -49*x*y],
  QuotientBasis := [1, y, x, y^2, x*y, y^3],
  StableBBasisFound := true
]

```

**See Also:** IdealOfPoints(I-9.7 pg.135), StableBBasis5(I-19.31 pg.295)

## I-1.15 ApproxSolve

syntax

```
ApproxSolve(L: LIST of RINGELEM): LIST of LIST of RAT
```

### Description

This function returns the list of real solutions (points) of a 0-dimensional polynomial system “L”. The polynomials in “L” must have rational coefficients. Approximate coordinates are given for non-rational solutions.

Useful verbosity range 20–20.

See also “RationalSolve” (I-18.15 pg.260)

example

```

/**/ use QQ[x,y,z];
/**/ L := [x^3-y^2+z-1, x-2, (y-3)*(y+2)];
/**/ RationalSolve(L);
[[2, -2, -3], [2, 3, 2]]
/**/ ApproxSolve(L);
[[2, -2, -3], [2, 3, 2]]

/**/ L := [x^3-y^2+1, (y-3)*(y+2), z];
/**/ indent(ApproxSolve(L));
[
  [167001090947516369641767378634802431634869700965461961120334511774287062707365/11579208923731619542
  [2, 3, 0]
]

/**/ L := [x^3-y^2+z-1, x^2-2, (y-3)*(y+2)];
/**/ Pts := ApproxSolve(L);
--> [[17564737135690137373...
/**/ indent([[ DecimalStr(coord,10) | coord in pt] | pt in Pts]);
[
  ["1.4142135624", "-2.0000000000", "2.1715728753"],
  ["1.4142135624", "3.0000000000", "7.1715728753"],

```

```

["-1.4142135624", "-2.0000000000", "7.8284271247"],
["-1.4142135624", "3.0000000000", "12.8284271247"]
]

-- Verify we have an approximate answer:
/**/ indent([ [ FloatStr(eval(f, pt)) | f in L ] | pt in Pts]);
[
["-3.2567*10^(-76)", "-6.2932*10^(-77)", "2.3668*10^(-76)"],
["-1.3971*10^(-77)", "8.1808*10^(-78)", "2.5541*10^(-77)"],
["-3.7110*10^(-77)", "8.1808*10^(-78)", "2.5541*10^(-77)"],
["7.7208*10^(-77)", "3.2902*10^(-77)", "-1.2374*10^(-76)"]
]

```

**See Also:** [LinSolve\(I-12.15 pg.185\)](#), [RationalSolve\(I-18.15 pg.260\)](#)

## I-1.16 AreGensMonomial

— syntax —

```
AreGensMonomial(I: IDEAL): BOOL
```

### Description

This function checks if the “*given generators*” for “I” are monomial, and stores this information in “I”: this is useful if it has thousands of generators and we want to know if we can use special algorithms for monomial generators.

NOTE: this function return false if at least one generator in “gens(I)” is not monomial even if “*there exists*” another set of generators which are monomial.

— example —

```

/**/ use P := QQ[x[1..100]];
/**/ AreGensMonomial(ideal(x[1], x[1]+x[2]));
false

/**/ I := ideal(support(sum(indets(P))^3));
/**/ t0 := CpuTime(); AreGensMonomial(I); TimeFrom(t0);
true
0.040
/**/ t0 := CpuTime(); AreGensMonomial(I); TimeFrom(t0);
true
0.000

```

**See Also:** [AreGensSqFreeMonomial\(I-1.17 pg.34\)](#), [IsTerm\(I-9.86 pg.169\)](#), [HasGBasis\(I-8.1 pg.121\)](#)

## I-1.17 AreGensSqFreeMonomial

— syntax —

```
AreGensSqFreeMonomial(I: IDEAL): BOOL
```

### Description

This function checks if the “*given generators*” for “I” are monomial and square-free, and stores this information in “I”: this is useful if it has thousands of generators and we want to know if we can use special algorithms for square-free monomial generators.

NOTE: this function returns “true” only if the given generators (i.e. those from “gens(I)”) are all square-free monomials, regardless of whether “*there exists another*” set of generators which are all square-free monomials.

example

```
/**/ use P ::= QQ[x,y,z];
/**/ AreGensSqFreeMonomial(ideal(x, y));
true
/**/ AreGensSqFreeMonomial(ideal(x, x^2));
false
/**/ AreGensSqFreeMonomial(ideal(x, x+y));
false
```

**See Also:** AreGensMonomial(I-1.16 pg.34), IsSqFree(I-9.79 pg.166), HasGBasis(I-8.1 pg.121)

## I-1.18 *ascii*

syntax

```
ascii(N: INT): STRING
ascii(L: LIST of INT): STRING
ascii(S: STRING): LIST of INT
```

### Description

In the first form, “ascii” returns the character whose ASCII code is “N”.

In the second form, “ascii” returns the string whose characters, in order, have the ASCII codes listed in “L”.

The third form is the inverse of the second: it returns the ASCII codes of the characters in “S”.

example

```
/**/ ascii(97);
a

/**/ C := ascii("hello world");
/**/ C;
[104, 101, 108, 108, 111, 32, 119, 111, 114, 108, 100]

/**/ ascii(C);
hello world
```

## I-1.19 *AsINT*

syntax

```
AsINT(N: INT): INT
AsINT(N: RAT): INT
AsINT(N: RINGELEM): INT
```

### Description

If the argument is an integer value this function returns this value as an INT, otherwise it throws an error.

example

```
/**/ use P ::= QQ[x,y];
/**/ type(LC(3*x-y));
RINGELEM
/**/ type(AsINT(LC(3*x-y)));
INT
-- /**/ type(AsINT(LC((3/2)*x-y))); --> !!! ERROR !!! as expected
```

**See Also:** `AsRAT`([I-1.20](#) pg.36)

## I-1.20 `AsRAT`

syntax

```
AsRAT(N: INT): RAT
AsRAT(N: RAT): RAT
AsRAT(N: RINGELEM): RAT
```

### Description

If the argument is a rational value this function returns this value as a `RAT`, otherwise it throws an error. Note that if the argument is actually an integer the result is nevertheless a `RAT` (with denominator 1).

example

```
/**/ use P := QQ[x,y];
/**/ type(LC(3*x-y));
RINGELEM
/**/ type(AsRAT(LC(3*x-y)));
RAT
```

**See Also:** `AsINT`([I-1.19](#) pg.35)

# Chapter I-2

## B

### I-2.1 BaseRing

syntax

```
BaseRing(RmodI: (Quotient)RING): RING
BaseRing(K: (Fraction Field)RING): RING
```

#### Description

This function gives the "base ring" of a given ring; e.g. if "K" was constructed as the fraction field of "R" then "BaseRing(K)" produces "R". All rings in CoCoA are derived from "ZZ" via various steps; "BaseRing" gives the ring which is one step closer to "ZZ".

example

```
/**/ Fpx := ZZ/(7)[x];
/**/ Fp := BaseRing(Fpx); --> ZZ/(7)
/**/ BaseRing(Fp) = ZZ;
true
```

**See Also:** NewFractionField(I-14.1 pg.209), NewQuotientRing(I-14.9 pg.212), NewPolyRing(I-14.8 pg.212)

### I-2.2 BBasis5

syntax

```
BBasis5(I: IDEAL): LIST
```

#### Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This function is implemented in ApCoCoALib by Stefan Kaspar.

The function "BBasis5" calls the CoCoAServer to compute a Border Basis of zero dimensional ideal I.

example

```
/**/ use QQ[x, y], DegLex;
/**/ I := ideal([x^2, x*y + y^2]);
BBasis := BBasis5(I);
```

## I-2.3 BettiDiagram

syntax

BettiDiagram(X: IDEAL or (quotient)RING or MODULE)

### Description

This function computes the ("Macaulay-style") Betti diagram for "M".

example

```

/**/ use R ::= QQ[t,x,y,z];
/**/ I := ideal(x^2-y*t, x*y-z*t, x*y);
/**/ RES := res(I);
/**/ PrintRes(RES);
0 --> R(-5)^2 --> R(-4)^4 --> R(-2)^3
/**/ B := BettiDiagram(RES); indent(B);
record[
  Diagram := matrix(ZZ,
    [[3, 0, 0],
     [0, 4, 2]]),
  FirstShift := 2
]
/**/ PrintBettiDiagram(B);
      0      1      2
-----
2:      3      -      -
3:      -      4      2
-----
Tot:      3      4      2

```

**See Also:** BettiMatrix(I-2.4 pg.38), PrintRes(I-16.35 pg.248), PrintBettiDiagram(I-16.31 pg.246), PrintBettiMatrix(I-16.32 pg.246)

## I-2.4 BettiMatrix

syntax

BettiMatrix(M: IDEAL|MODULE|LISTResolution)

### Description

This function returns the Betti matrix for "M".

example

```

/**/ use R ::= QQ[t,x,y,z];
/**/ I := ideal(x^2-y*t, x*y-z*t, x*y);
/**/ PrintRes(I);
0 --> R^2(-5) --> R^4(-4) --> R^3(-2)
/**/ BettiMatrix(I);
matrix(ZZ,
  [[0, 0, 0],
   [3, 0, 0],
   [0, 0, 0],
   [0, 4, 0],
   [0, 0, 2]])

```

```

/**/ PrintBettiMatrix(I);
-- --> --> --
      0      0      0
      0      0      3
      0      0      0
      0      4      0
      2      0      0
-- --> --> --

```

**See Also:** [PrintRes\(I-16.35 pg.248\)](#), [PrintBettiDiagram\(I-16.31 pg.246\)](#)

## I-2.5 BettiNumbers

syntax

```
BettiNumbers(M: IDEAL|MODULE|Resolution): LIST
```

### Description

This function returns the Betti numbers for “M”.

example

```

/**/ M := MakeTermOrd(matrix([[5,5,5,1,1], [1,1,1,0,0]]));
/**/ P := NewPolyRing(QQ, "t[1],t[2],t[3],x,y", M, 2); -- ZZ^2-grading
/**/ use P;
/**/ I := ideal(t[1]^6 -t[3]^6, t[2]^6 -t[1]^5*t[3], t[1]*t[3]*x^8 -t[2]^2*y^8);
/**/ RES := res(P/I);
/**/ PrintRes(RES);
0 --> R[-78,-14] --> R[-48,-8]^2(+)R[-60,-12] --> R[-18,-2](+)R[-30,-6]^2 --> R

/**/ PrintBettiNumbers(RES); --> just prints in a readable way
----- 1 -----
----- 2 -----
      [18,  2]: 1
      [30,  6]: 2
----- 3 -----
      [48,  8]: 2
      [60, 12]: 1
----- 4 -----
      [78, 14]: 1
-----

/**/ BettiNumbers(RES); --> returns the value for further computations
[[], [[18,  2], 1], [[30,  6], 2]], [[48,  8], 2],
      [[60, 12], 1]], [[78, 14], 1]]

```

**See Also:** [PrintRes\(I-16.35 pg.248\)](#), [BettiDiagram\(I-2.3 pg.38\)](#), [BettiMatrix\(I-2.4 pg.38\)](#), [PrintBettiNumbers\(I-16.33 pg.247\)](#)

## I-2.6 binomial

syntax

```

binomial(N: INT, K: INT): INT
binomial(N: RINGELEM, K: INT): RINGELEM

```

## Description

This function computes the binomial coefficient, “ $N$  choose  $K$ ” according to the formula  $(N)(N-1)(N-2)\dots(N-K+1)/K!$

The same formula is used if  $N$  is a polynomial. The integer  $K$  cannot be negative.

example

```
/**/ binomial(4,2);
6

/**/ binomial(-4,3);
-20

/**/ use QQ[x,y];
/**/ binomial(x^2+2*y,3);
(1/6)*x^6 +x^4*y +(-1/2)*x^4 +2*x^2*y^2 -2*x^2*y +(4/3)*y^3 +(1/3)*x^2 -2*y^2 +(2/3)*y

/**/ It = (x^2+2*y)*(x^2+2*y-1)*(x^2+2*y-2)/6;
true
```

**See Also:** BinomialRepr, BinomialReprShift(I-2.7 pg.40)

## I-2.7 BinomialRepr, BinomialReprShift

syntax

```
BinomialRepr(N: INT, K: INT): LIST of INT
BinomialReprShift(N: INT, K: INT, Up: INT, Down: INT): INT
```

where  $N$  and  $K$  are positive.

## Description

The function “BinomialRepr” computes the “ $K$ ”-binomial representation of “ $N$ ”, also called Macaulay representation, i.e. the unique expression

$$N = \text{binomial}(N(K), K) + \text{binomial}(N(K-1), K-1) + \dots + \text{binomial}(N(L), L)$$

where  $N(K) > \dots > N(L) \geq 1$ , for some  $L$ . The value returned is the list “[ $N(t) \mid t \text{ in } 1..K$ ]” where  $N(t)=0$  for all  $t < L$ .

The function call “BinomialReprShift( $N, K, \text{up}, \text{down}$ )” computes the integer

$$\begin{aligned} & \text{binomial}(N(K) + \text{up}, K + \text{down}) + \\ & \text{binomial}(N(K-1) + \text{up}, (K-1) + \text{down}) + \\ & \dots + \\ & \text{binomial}(N(L) + \text{up}, L + \text{down}) \end{aligned}$$

It is useful in generalizations of Macaulay’s theorem characterizing Hilbert functions.

example

```
/**/ BinRep := BinomialRepr(13,4);
/**/ BinRep;
[1, 3, 4, 5]

/**/ BinomialReprShift(13,4,1,1);
16
```

**See Also:** binomial(I-2.6 pg.39)

## I-2.8 *block*

syntax

```
block C_1; ... ; C_n EndBlock;

where each C_i is a command.
```

### Description

The “**block**” command executes the commands as if they were one command. What this means in practice is that CoCoA will not print a string of dashes after executing each “**C<sub>i</sub>**”. Thus, “**Block**” is used on-the-fly and not inside user-defined functions. (It has nothing to do with declaration of local variables, for instance, as one might infer from some other computer languages.) The following example should make the use of “**Block**” clear:

example

```
/**/ Print "hello "; Print "world";
hello world
-----
/**/ Block
/**/   Print "hello ";
/**/   Print "world";
/**/ EndBlock;
hello world
-----
/**/ use QQ[x,y];
/**/ Block
/**/   PrintLn GCD([12, 24, 96]);
/**/   PrintLn LCM([12, 24, 96]);
/**/   PrintLn GCD([x+y, x^2-y^2]);
/**/   Print LCM([x+y, x^2-y^2]);
/**/ EndBlock;

12
96
x + y
x^2 - y^2
-----
```

## I-2.9 *BlockMat*

syntax

```
BlockMat(LIST of LIST of MAT: L): MAT
```

### Description

This function creates a block matrix from a LIST of rows of matrices. The following restrictions on the sizes of the matrices apply: in each row list: all matrices must have the same number of rows; for all row list: the total number of columns must be the same.

The function “**BlockMat2x2**” ([I-2.10 pg.42](#)) has a simpler syntax for a 2x2 block matrix.

example

```
/**/ A := RowMat([1,2,3,4]); B := RowMat([0,0]);
-- /**/ BlockMat2x2(A,B, B,A); --> !!! ERROR !!! as expected
/**/ BlockMat([[A,B], [B,A]]);
```

```
matrix(QQ,
  [[1, 2, 3, 4, 0, 0],
   [0, 0, 1, 2, 3, 4]])
/**/ BlockMat([[A,B], [RowMat(1..6)]]);
matrix(QQ,
  [[1, 2, 3, 4, 0, 0],
   [1, 2, 3, 4, 5, 6]])
```

**See Also:** ConcatHor(I-3.41 pg.61), ConcatVer(I-3.44 pg.63), ConcatHorList(I-3.42 pg.62), ConcatVerList(I-3.45 pg.63), ConcatDiag(I-3.40 pg.61), ConcatAntiDiag(I-3.39 pg.60), BlockMat2x2(I-2.10 pg.42)

## I-2.10 BlockMat2x2

syntax

```
BlockMat2x2(A: MAT,B: MAT,C: MAT,D: MAT): MAT
```

### Description

This function creates a block matrix. Each entry is a matrix. Given A, B, C, D matrices, then “BlockMat(A,B,C,D)” returns the matrix

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

The obvious restrictions on the sizes of the matrices apply:

“NumRows(A) = NumRows(B)” and “NumRows(C) = NumRows(D)”, and “NumCols(A) = NumCols(C)” and “NumCols(B) = NumCols(D)”.

The function “BlockMat” (I-2.9 pg.41) offers more flexibility, but with a heavier syntax.

example

```
/**/ A := matrix([[1,2,3], [4,5,6]]);
/**/ B := matrix([[1,2], [3,4]]);
/**/ C := matrix([[1,1,1], [2,2,2], [3,3,3]]);
/**/ D := matrix([[4,4], [5,5], [6,6]]);
/**/ BlockMat2x2(A,B, C,D);
matrix(QQ,
  [[1, 2, 3, 1, 2],
   [4, 5, 6, 3, 4],
   [1, 1, 1, 4, 4],
   [2, 2, 2, 5, 5],
   [3, 3, 3, 6, 6]])
```

**See Also:** ConcatHor(I-3.41 pg.61), ConcatVer(I-3.44 pg.63), ConcatDiag(I-3.40 pg.61), ConcatAntiDiag(I-3.39 pg.60), BlockMat(I-2.9 pg.41)

## I-2.11 Bool01

syntax

```
Bool01(B: BOOL): INT
```

### Description

This function converts a boolean to an integer using the convention: “false” becomes 0, and “true” becomes 1.

example

```

/**/ Id4 := matrix([[Bool01(i=j) | i in 1..4] | j in 1..4]);
/**/ Id4;
matrix(QQ,
  [[1, 0, 0, 0],
   [0, 1, 0, 0],
   [0, 0, 1, 0],
   [0, 0, 0, 1]])

```

## I-2.12 *break*

syntax

```
break
```

### Description

This command may be used only inside a loop statement (“for”, “foreach”, “repeat”, or “while”).

When executed, the entire current loop statement is terminated, and control passes to the command following the loop statement. If you just want to skip to the next iteration of the current loop statement use instead “continue” (I-3.51 pg.66).

In the case of nested loops “break” leaves just the innermost loop statement in which the “break” statement appears.

example

```

/**/ for i := 5 to 1 step -1 do
/**/   for j := 1 to 10 do
/**/     print j, " ";
/**/     if j = i then println; break; endif;
/**/   endfor;
/**/ endfor;
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1

```

**See Also:** continue(I-3.51 pg.66), return(I-18.39 pg.271), syntax(?? pg.??)

## I-2.13 *BringIn*

syntax

```
BringIn(E: OBJECT): OBJECT
```

### Description

This function maps a polynomial (or a list, matrix of these) into the current ring, preserving the names of the indeterminates.

This function is not implemented on ideals because might be misleading: one might expect that bringing an ideal from “ $\mathbb{R}[x,y]$ ” into “ $\mathbb{R}[x]$ ” means eliminating “ $y$ ”, while others might expect the ideal generated by mapping the generators. For example in the first case  $(x - y, x + y)$  returns the ideal  $(x)$ , in the second case returns an error. So, if you want to map the generators of the ideal type “`ideal(BringIn(gens(I)))`”.

– Changing characteristic from non-0 to 0 is NOT YET IMPLEMENTED in CoCoA-5 When mapping from a ring of finite characteristic to one of zero characteristic then consistent choices of image for the coefficients are made (i.e. if two coefficients are equal mod  $p$  then their images will be equal).

example

```

/**/ RR := QQ[x[1..4],z,y];
/**/ SS := ZZ[z,y,x[1..2]];
/**/ use RR;
/**/ F := (x[1]-y-z)^2; F;
x[1]^2 -2*x[1]*z +z^2 -2*x[1]*y +2*z*y +y^2

/**/ use SS;
/**/ BringIn(F);
z^2 +2*z*y +y^2 -2*z*x[1] -2*y*x[1] +x[1]^2

/**/ use R := QQ[x,y,z];
/**/ F := (1/2)*x^3 + (34/567)*x*y*z - 890; -- poly with rational coefficients
/**/ use S := ZZ/(101)[x,y,z];
/**/ BringIn(F);
-50*x^3 -19*x*y*z +19

```

**See Also:** PolyAlgebraHom([I-16.14](#) pg.237), apply([I-1.13](#) pg.32), QZP([I-17.7](#) pg.253), ZPQ([I-25.3](#) pg.338)

# Chapter I-3

## C

### I-3.1 Call [OBSOLETE]

syntax

```
[OBSOLETE]
```

#### Description

OBSOLETE: in CoCoA-5 functions can be used directly. See “FUNCTIONs are first class objects” ([III-7.2 pg.399](#)).

### I-3.2 CallOnGroebnerFanIdeals

syntax

```
CallOnGroebnerFanIdeals(I: IDEAL, fn: FUNCTION)
```

#### Description

Storing all the possible different (reduced) GBases in a Groebner fan is practicable only for a small example; larger ideals may have fans containing thousands or even millions of different Groebner bases. Typically we are interested only in those bases satisfying a certain property.

“CallOnGroebnerFanIdeals” calls the given function “fn” successively on the ideal “I” mapped into different polynomial rings so that the Groebner bases run through all possible distinct ones. This approach avoids storing all distinct possibilities in a big list.

Verbosity: see “GroebnerFanIdeals” ([I-7.33 pg.119](#)).

Note: using this needs a little technical ability, but might make the difference between getting an answer or fill up the computer’s memory.

example

```
/**/ -- print ord and GBasis if ideal I has GBases of length 3:
/**/ define PrintIfGBHasLen3(I)
/**/   if len(ReducedGBasis(I))=3 then
/**/     println OrdMat(RingOf(I));
/**/     indent(ReducedGBasis(I));
/**/   endif;
/**/ enddefine;

/**/ use R ::= QQ[a,b,c];
```

```

/**/ I := ideal(a^5+b^3+c^2-1, b^2+a^2+c-1, c^3+a^6+b^5-1);
/**/ SetVerbosityLevel(10);
/**/ CallOnGroebnerFanIdeals(I, PrintIfGBHasLen3);
*****
matrix(ZZ,
  [[3, 7, 7],
   [3, 6, 8],
   [0, 0, -1]])
[b^2+c+a^2-1,
 a^5+c^2-b*c-a^2*b+b-1,
 c^3+b*c^2+2*a^2*b*c+a^4*b-a*c^2+a*b*c+a^3*b-2*b*c-2*a^2*b-a*b+b+a-1]
*
matrix(ZZ,
  [[6, 7, 14],
   [6, 5, 15],
   [0, 0, -1]])
[c+b^2+a^2-1,
 -b^6-3*a^2*b^4-3*a^4*b^2+b^5+3*b^4+6*a^2*b^2+3*a^4-3*b^2-3*a^2,
 a^5+b^4+2*a^2*b^2+a^4+b^3-2*b^2-2*a^2]
*****

```

**See Also:** [GroebnerFanIdeals\(I-7.33 pg.119\)](#), [OrdMat\(I-15.10 pg.231\)](#), [RingOf\(I-18.46 pg.275\)](#)

### I-3.3 CanonicalHom

— syntax —

```
CanonicalHom(R: RING, S: RING): RINGHOM
```

#### Description

`CanonicalHom(R, S)` – where `R` and `S` are rings, gives the canonical homomorphism from `R` to `S`. Currently it works only on the most natural constructions:

```

ZZ -> S      QQ -> S
R -> R/I     R -> FractionFields(R)
R -> R[x[1..N]]

```

— example —

```

/**/ use R ::= QQ[x,y];
/**/ RmodI := NewQuotientRing(R, ideal(x^2-1));

/**/ phi := CanonicalHom(R, RmodI);
/**/ phi(x^3*y);
(x*y)
/**/ RingOf(It) = RmodI;
true

/**/ RingElem(RmodI, x^3*y); -- same as phi(x^3*y)
-- internally computes CanonicalHom

```

**See Also:** [NewFractionField\(I-14.1 pg.209\)](#), [NewQuotientRing\(I-14.9 pg.212\)](#), [NewPolyRing\(I-14.8 pg.212\)](#), [CanonicalHom\(I-3.3 pg.46\)](#), [CoeffEmbeddingHom\(I-3.22 pg.53\)](#), [QuotientingHom\(I-17.6 pg.253\)](#), [PolyAlgebraHom\(I-16.14 pg.237\)](#), [PolyRingHom\(I-16.15 pg.238\)](#)

## I-3.4 CanonicalRepr

syntax

```
CanonicalRepr(f: RINGELEM): RINGELEM
```

### Description

Given an element “f” in a quotient ring “R/I” this function returns a representative of “f” in “R”.

example

```
/**/ use R := QQ[a];
/**/ RmodI := R/ideal(a^2-2);
/**/ use RmodI;
/**/ a^3;
(2*a)
/**/ RingOf(a^3);
RingWithID(9, "RingWithID(7)/ideal(a^2 -2)")
/**/ CanonicalRepr(a^3);
2*a
/**/ RingOf(CanonicalRepr(a^3));
RingWithID(7, "QQ[a]")
```

**See Also:** NewQuotientRing(I-14.9 pg.212), DefiningIdeal(I-4.5 pg.74)

## I-3.5 CartesianProduct, CartesianProductList

syntax

```
CartesianProduct(L1: LIST, L2: LIST, L3: LIST, ...): LIST
CartesianProductList(L: LIST of LIST): LIST
L1 >< L2
L1 >< L2 >< ... >< Ln

where each Li is a LIST
```

### Description

This command returns the list whose elements form the Cartesian product of  $L_1, \dots, L_n$ .

For the N-fold product of a list with itself, one may use “tuples” (I-20.15 pg.317).

example

```
/**/ L1 := [1,2,3];
/**/ L2 := ["a","b"];
/**/ L1 >< L2 >< [5]; -- same as
/**/ CartesianProduct(L1, L2, [5]); -- same as
/**/ CartesianProductList([L1, L2, [5]]); -- this takes a list of lists
[[1, "a", 5], [1, "b", 5], [2, "a", 5], [2, "b", 5], [3, "a", 5], [3, "b", 5]]
-----
/**/ ChessBoard := (1..8)><(1..8); -- Need brackets around 1..8 otherwise
                                   -- we get a parse error.
```

Note that only “<>” is used for “not equal” in CoCoA.

**See Also:** CoCoA Operators(?? pg.??), operators, shortcuts(I-0.1 pg.25), tuples(I-20.15 pg.317)

## I-3.6 Cast [OBSOLETE]

syntax

[OBSOLETE]

### Description

[OBSOLETE] To cast INT, RAT, STRING to a polynomial (and more in general to a RINGELEM) use “RingElem” (I-18.43 pg.273).

To cast RINGELEM to INT, RAT use “AsINT” (I-1.19 pg.35), “AsRAT” (I-1.20 pg.36).

To cast LIST to MAT use “matrix” (I-13.10 pg.195). To cast MAT to LIST use “GetRows” (I-7.16 pg.114), “GetCols” (I-7.12 pg.113).

To cast a MODULEELEM to LIST use “compts” (I-3.36 pg.59).

To cast a MODULE to MAT use “GensAsCols, GensAsRows” (I-7.9 pg.111). To cast a MAT to MODULE use “SubmoduleCols, SubmoduleRows” (I-19.45 pg.302).

**See Also:** AsINT(I-1.19 pg.35), AsRAT(I-1.20 pg.36), gens(I-7.8 pg.110), GensAsCols, GensAsRows(I-7.9 pg.111), GetCols(I-7.12 pg.113), GetRows(I-7.16 pg.114), ideal(I-9.2 pg.131), matrix(I-13.10 pg.195), ModuleElem(I-13.28 pg.203), RingElem(I-18.43 pg.273), SubmoduleCols, SubmoduleRows(I-19.45 pg.302)

## I-3.7 ceil

syntax

ceil(X: RAT): INT

### Description

This function returns the least integer greater than or equal to “X”.

example

```

/**/  ceil(0.99);
1

/**/  ceil(0.01);
1

/**/  ceil(1);
1

/**/  ceil(-0.99);
0

```

**See Also:** floor(I-6.14 pg.99), round(I-18.55 pg.279), num(I-14.33 pg.223), den(I-4.7 pg.75)

## I-3.8 CFApprox

syntax

CFApprox(X: RAT, MaxRelErr: RAT): RAT

### Description

“CFApprox” finds the “*simplest*” continued fraction approximant to “X” which is within the maximum specified “*relative error*”.

example

```
/**/ CFApprox(1.414213, 10^(-2));
17/12
```

**See Also:** CFApproximants([I-3.9](#) pg.49), ContFrac([I-3.49](#) pg.65), SimplestRatBetween([I-19.15](#) pg.288)

## I-3.9 CFApproximants

syntax

```
CFApproximants(X: RAT): LIST of RAT
```

### Description

“CFApproximants” returns a list of all continued fraction approximants to the rational “X”.

example

```
/**/ CFApproximants(1.414213);
[1, 3/2, 7/5, 17/12, 41/29, 99/70, 239/169, 577/408, 816/577, 1393/985,
 6388/4517, 7781/5502, 14169/10019, 21950/15521, 36119/25540, 58069/41061,
 152257/107662, 210326/148723, 1414213/1000000]
```

**See Also:** CFApprox([I-3.8](#) pg.48), ContFrac([I-3.49](#) pg.65)

## I-3.10 characteristic

syntax

```
characteristic(R: RING): INT
```

### Description

This function returns the characteristic of the current ring, in the first case, or of the ring R, in the second.

example

```
/**/ use R ::= ZZ/(3)[t];
/**/ S ::= QQ[x,y];
/**/ characteristic(CurrentRing);
3

/**/ characteristic(S);
0
```

**See Also:** IsFiniteField([I-9.50](#) pg.155), LogCardinality([I-12.18](#) pg.187)

## I-3.11 CharPoly

syntax

```
CharPoly(M: MAT, X: RINGELEM): RINGELEM
```

### Description

This function returns the characteristic polynomial of “M”, square matrix, in the indeterminate “X”.

See also “MinPoly” ([I-13.22](#) pg.200).

example

```

/**/ use R ::= QQ[x];
/**/ CharPoly(matrix([[1,2,3],[4,5,6],[7,8,9]]), x);
x^3 -15*x^2 -18*x

```

**See Also:** [MinPoly\(I-13.22 pg.200\)](#)

## I-3.12 ChebyshevPoly

syntax

```

ChebyshevPoly(N: INT, X: RINGELEM): RINGELEM
ChebyshevPoly2(N: INT, X: RINGELEM): RINGELEM

```

### Description

The function “ChebyshevPoly” returns the Chebyshev polynomial (of 1st type) with index “N”, in the indeterminate “X”. The function “ChebyshevPoly2” returns the Chebyshev polynomial of 2nd type.

These functions also work if “X” is not an indeterminate: the result is then the evaluation of the polynomial at the given value.

example

```

/**/ use R ::= QQ[x];
/**/ ChebyshevPoly(3,x);
4*x^3 -3*x

```

**See Also:** [HermitePoly\(I-8.3 pg.122\)](#), [LaguerrePoly\(I-12.1 pg.179\)](#)

## I-3.13 CheckArgTypes

syntax

```

CheckArgTypes(Ltype: LIST of TYPE, Larg: LIST)

```

### Description

This function provides a basic type checking for user defined functions: it checks whether the types of the elements in the third argument, a list, correspond to the types in the second list. If so, it returns nothing, otherwise returns an error.

example

```

/**/ -- the following returns an error for the 2nd argument (INT)
/**/ -- CheckArgTypes([RAT, RINGELEM, MAT], [2/3, 20, LexMat(3)]);
--> ERROR: Arg 2 is INT but must be RINGELEM

/**/ -- the following returns nothing
/**/ CheckArgTypes([RAT, [INT,RAT,RINGELEM], MAT], [2/3, 20, LexMat(3)]);

/**/ -- an example of use for type checking
/**/ Define Pow(F, N)
/**/   CheckArgTypes([[INT,RAT,RINGELEM,IDEAL,MAT], INT], [F, N]);
/**/   Return F^N;
/**/ EndDefine; -- Pow
/**/ use QQ[x];
/**/ Pow(x, 3);

```

```
x^3
/**/  -- Pow(2, x); --> !!! ERROR !!! as expected
--> ERROR: Arg 2 is RINGELEM but must be INT
```

## I-3.14 *ciao*

syntax

```
ciao
```

### Description

This command is used to quit CoCoA. It may be used only at top level.

**See Also:** [exit\(I-5.15 pg.89\)](#), [quit\(I-17.3 pg.252\)](#)

## I-3.15 ClearDenom

syntax

```
ClearDenom(F: RINGELEM): RINGELEM
```

### Description

This function clears the denominators of the coefficients in a polynomial over QQ. It simply multiplies by the least common multiple of the denominators.

example

```
/**/  use QQ[x,y];
/**/  f := (2/3)*x + (4/5)*y;
/**/  ClearDenom(f);
10*x +12*y
```

## I-3.16 close

syntax

```
close(OUT: OSTREAM)
```

### Description

This function closes the output stream “OUT”.

example

```
/**/  file := OpenOFile("my-test"); -- open file for output from CoCoA
/**/  print "test" on file;  -- write to my-file
/**/  close(file);  -- close the output stream, "flushes" all output
```

**See Also:** [Introduction to IO\(II-7.1 pg.355\)](#)

## I-3.17 CloseLog

syntax

```
CloseLog(D: DEVICE)
```

## Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This function “OpenLog” ([I-15.4 pg.228](#)) opens the output device D and starts to record the output from a CoCoA session on D.

This function closes the device D and stops recording the CoCoA session on D.

**See Also:** OpenLog([I-15.4 pg.228](#))

## I-3.18 CoCoA-4 mode

syntax

```
*** E ***
```

where ‘‘\verb&E&’’ is a CoCoA-4 expression.

## Description

CoCoA-5 is not fully backward compatible with CoCoA-4, i.e. some CoCoA-4 programs will be rejected by CoCoA-5. CoCoA-4 mode helps ease the transition to CoCoA-5.

In CoCoA-4 it was not necessary to write explicitly the product between two indeterminates; in CoCoA-5 this is obligatory.

The expression “E” may also contain function calls, but only if the function names begin with a capital letter.

example

```
/**/ use QQ[x,y,z];
/**/ f := 2*x^2*y - 3*x*y*z - 4*y^2*z + 5*y*z^2 + 6*z^3;
/**/ g := ***2x^2y - 3xyz - 4y^2z + 5yz^2 + 6z^3***; --> C4 mode
/**/ f = g;
true
```

**See Also:** Changes in the CoCoA language([II-10.1 pg.367](#)), not([I-14.31 pg.222](#)), and([I-1.11 pg.31](#)), or([I-15.9 pg.231](#))

## I-3.19 CocoaLimits

syntax

```
CocoaLimits(): RECORD
```

## Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This function returns the maximum allowable characteristic of a CoCoA ring and the maximum allowable exponent in a CoCoA expression. These numbers may vary depending on the platform on which CoCoA is run.

example

```
CocoaLimits();
record[MaxChar := 32767, MaxExp := 2147483647]
-----
```

## I-3.20 CocoaPackagePath

— syntax —

```
CocoaPackagePath(): STRING
```

### Description

This function returns the path name of the directory containing the CoCoA libraries. It is platform dependent.

— example —

```
/**/ CocoaPackagePath();
/**/ Applications/CoCoA-5/packages
```

## I-3.21 codomain

— syntax —

```
codomain(phi: RINGHOM): RING
```

### Description

This function returns the codomain of the homomorphism “phi”

— example —

```
/**/ P := NewPolyRing(RingQQ(), "alpha,beta");
/**/ phi := CanonicalHom(RingZZ(), P);
/**/ codomain(phi);
RingWithID(4, "QQ[alpha,beta]")
/**/ psi := CoeffEmbeddingHom(P);
/**/ codomain(psi);
RingWithID(4, "QQ[alpha,beta]")
```

**See Also:** [codomain\(I-3.21 pg.53\)](#), [Commands and Functions for RINGHOM\(III-10.3 pg.410\)](#), [Commands and Functions returning RINGHOM\(III-10.4 pg.410\)](#)

## I-3.22 CoeffEmbeddingHom

— syntax —

```
CoeffEmbeddingHom(P: RING): RINGHOM
```

### Description

This function returns the coefficient embedding homomorphism of the polynomial ring “P”.

It is equivalent to (indeed it is called by) “[CanonicalHom\(CoeffRing\(P\), P\)](#)”.

— example —

```
/**/ use P := QQ[x,y];
/**/ phi := CoeffEmbeddingHom(P); -- phi: QQ -> P
/**/ f := 2*x+3*y;
/**/ f/phi(LC(f));
x + (3/2)*y
```

**See Also:** [CanonicalHom\(I-3.3 pg.46\)](#)

### I-3.23 CoeffHeight

syntax

```
CoeffHeight(F: RINGELEM): RINGELEM
```

#### Description

This function returns the maximum of the absolute values of the coefficients of “F”; naturally, the coefficient ring must be arithmetically ordered.

example

```
/**/ use P := QQ[x,y];
/**/ f := (2*x-3*y)^2;
/**/ f;
4*x^2 -12*x*y +9*y^2
/**/ CoeffHeight(f);
12
```

**See Also:** [RootBound\(I-18.54 pg.278\)](#)

### I-3.24 coefficients

syntax

```
coefficients(F: RINGELEM): LIST
coefficients(F: RINGELEM, S: LIST): LIST
```

#### Description

This function returns a list of coefficients of “F” which are elements of “`CoeffRing(RingOf(F))`”.

Called with one argument “F” it returns the list of all non-zero coefficients; the order being decreasing on the terms in “F” as determined by the term-ordering of “`RingOf(F)`”.

Called with two arguments “F,S” it returns the coefficients of the list of specified terms “S”; their order is determined by the list “S”. If a terms does not appear in “F” then the corresponding coefficient is 0.

The old form (CoCoA-4) “`Coefficients(F,x)`” for the coefficients of “F” w.r.t an indeterminate “x” is now replaced by the functions “`CoefficientsWRT`” ([I-3.25 pg.55](#)) and “`CoeffListWRT`” ([I-3.26 pg.56](#)).

example

```
/**/ use R := QQ[x,y,z];
/**/ F := 3*x^2*y + 5*y^2 - x*y;
/**/ Coeffs := coefficients(F); Coeffs; -- with one argument
[3, -1, 5]
/**/ phi := CoeffEmbeddingHom(RingOf(F));
/**/ F = ScalarProduct(apply(phi,Coeffs), support(F));
true

/**/ Skeleton := [1, x, y, z, x^2, x*y, y^2, y*z, z^2];
/**/ Coeffs := coefficients(F, Skeleton); Coeffs; -- with two arguments
[0, 0, 0, 0, 0, -1, 5, 0, 0]
/**/ ScalarProduct(apply(phi,Coeffs), Skeleton);
-x*y +5*y^2

/**/ L := CoefficientsWRT(F,[x,y,z]); indent(L); -- similar function
[
  record[PP := y^3, coeff := 5],
  record[PP := x^2*y, coeff := 3],
```

```

    record[PP := x*y^5, coeff := -1]
]
/**/ F = sum([X.coeff * X.PP | X in L]);
true

/**/ L := CoeffListWRT(F, y); L; -- similar function
[0, 3*x^2 -x, 5]
/**/ F = sum([L[d+1]*y^d | d in 0..(len(L)-1)]);
true

/**/ R3 := NewFreeModule(R,3);
/**/ V := ModuleElem(R3, [3*x^2+y, x-5*z^3, x+2*y]);
/**/ ConcatLists([coefficients(V[i]) | i in 1..NumCompts(V)]);
[3, 1, -5, 1, 1, 2]

```

**See Also:** Coefficient Rings([III-9.3](#) pg.404), CoefficientsWRT([I-3.25](#) pg.55), CoeffListWRT([I-3.26](#) pg.56), LC([I-12.4](#) pg.180), monomials([I-13.31](#) pg.204), support([I-19.50](#) pg.304)

## I-3.25 CoefficientsWRT

syntax

```

CoefficientsWRT(F: RINGELEM, X: RINGELEM): LIST of RECORD
CoefficientsWRT(F: RINGELEM, S: LIST of RINGELEM): LIST of RECORD

```

### Description

The first function returns the list of the coefficients and PPs of “F” seen as a polynomial in “X”; the second function does the same but viewing “F” as a polynomial in all the indeterminates in the set “S”.

Note that coefficients in the result are RINGELEM belonging to “RingOf(F)”.

example

```

/**/ use R := QQ[x,y,z];
/**/ f := x^3*z+x*y+x*z+y+2*z;
/**/ Cx := CoefficientsWRT(f, x); -- same as...
/**/ Cx := CoefficientsWRT(f, [x]);
/**/ indent(Cx);
[
  record[PP := x^3, coeff := z],
  record[PP := x, coeff := y +z],
  record[PP := 1, coeff := y +2*z]
]
/**/ f = sum([M.coeff * M.PP | M in Cx]);
true
/**/ Foreach M in Cx Do Print "  +(", M.coeff, ")*", M.PP; EndForeach;
  +(y +2*z)*1  +(y +z)*x  +(z)*x^3

/**/ Cxz := CoefficientsWRT(f, [x,z]);
/**/ indent(Cxz);
[
  record[PP := x^3*z, coeff := 1],
  record[PP := x*z, coeff := 1],
  record[PP := x, coeff := y],
  record[PP := z, coeff := 2],
  record[PP := 1, coeff := y]
]

```

**See Also:** Coefficient Rings([III-9.3](#) pg.404), CoeffListWRT([I-3.26](#) pg.56), coefficients([I-3.24](#) pg.54), CoeffOfTerm([I-3.27](#) pg.56), ContentWRT([I-3.48](#) pg.65), LC([I-12.4](#) pg.180), monomials([I-13.31](#) pg.204), support([I-19.50](#) pg.304)

## I-3.26 CoeffListWRT

— syntax —

CoeffListWRT(F: RINGELEM, X: RINGELEM): LIST of RINGELEM

### Description

This function returns the list of the coefficients of “F” seen as a univariate polynomial in “X”, an indeterminate or a list of indeterminates. All entries in the returned list are RingElems belonging to “RingOf(F)”.

Note that the returned list is “reversed” from the CoCoA-4 analogue “Coefficients(F,X)” thus to re-use old code you should call “reversed(CoeffListWRT(F,X))”.

— example —

```
/**/ use R := QQ[x,y,z];
/**/ F := 5*y^2 + (3*x^2-x)*y;
/**/ L := CoeffListWRT(F, y); Print L;
[0, 3*x^2 -x, 5]
/**/ F = sum([L[d+1]*y^d | d in 0..(len(L)-1)]);
true
```

**See Also:** coefficients([I-3.24](#) pg.54), CoefficientsWRT([I-3.25](#) pg.55)

## I-3.27 CoeffOfTerm

— syntax —

CoeffOfTerm(F: RINGELEM, T: RINGELEM): RINGELEM

### Description

This function returns the coefficient of the term “T” occurring in “F”. NOTE: In CoCoA 4 the order of the arguments was different.

— example —

```
/**/ use R := QQ[x,y,z];
/**/ F := 5*x*y^2 - 3*z^3;
/**/ CoeffOfTerm(F, x*y^2);
5
/**/ CoeffOfTerm(F, x^3);
0
/**/ CoeffOfTerm(F, z^3);
-3
```

**See Also:** coefficients([I-3.24](#) pg.54), LC([I-12.4](#) pg.180), exponents([I-5.16](#) pg.90), MakeTerm([I-13.4](#) pg.192), monomials([I-13.31](#) pg.204), support([I-19.50](#) pg.304)

## I-3.28 CoeffRing

— syntax —

CoeffRing(R: RING): RING

## Description

This function returns the ring of coefficients of a polynomial ring.

example

```
/**/ use R := QQ[x,y,z];
/**/ S := ZZ/(2)[a,b,c];
/**/ CoeffRing(R);
QQ

/**/ CoeffRing(S);
FFp(2)
```

**See Also:** [characteristic\(I-3.10 pg.49\)](#), [coefficients\(I-3.24 pg.54\)](#), [CurrentRing\(I-3.56 pg.68\)](#), [indets\(I-9.24 pg.144\)](#)

## I-3.29 ColMat

syntax

```
ColMat(L: LIST): MAT
ColMat(R: RING, L: LIST): MAT
```

## Description

This function returns the matrix whose only column consists of the elements of the list “L”.

The first form produces a matrix over “QQ” if all entries in “L” are of type “INT” or “RAT”. If “L” contains any entries of type RINGELEM then the matrix is over the ring these elements belong to.

The second form produces a matrix over “R”, and requires that the elements of “L” be “INT”, “RAT” or “RINGELEM” belonging to “R”.

example

```
/**/ ColMat([3,4,5]);
matrix(QQ,
  [[3],
   [4],
   [5]])

/**/ RingOf(It); -- default ring is QQ
QQ

/**/ ColMat(ZZ, [3,4,5]);
matrix(ZZ,
  [[3],
   [4],
   [5]])
/**/ RingOf(It);
ZZ
```

**See Also:** [matrix\(I-13.10 pg.195\)](#), [BlockMat\(I-2.9 pg.41\)](#), [DiagMat\(I-4.15 pg.79\)](#), [RowMat\(I-18.56 pg.279\)](#), [GensAsCols](#), [GensAsRows\(I-7.9 pg.111\)](#)

## I-3.30 colon

syntax

```
colon(M: IDEAL, N: IDEAL): IDEAL
colon(M: MODULE, N: MODULE): IDEAL
```

## Description

This function returns the quotient of  $M$  by  $N$ : the ideal of all polynomials  $F$  such that  $F \cdot G$  is in  $M$  for all  $G$  in  $N$ . The command “ $M : N$ ” is a shortcut for “`colon(M, N)`”.

See also “`HColon`” ([I-8.2 pg.121](#)) for non-homogeneous input.

example

```
/**/ use R := QQ[x,y];
/**/ ideal(x*y, x^2) : ideal(x);
ideal(y, x)

/**/ colon(ideal(x^2, x*y), ideal(x, x-y^2));
ideal(x)
```

**See Also:** `saturate`([I-19.1 pg.281](#)), `HSaturation`([I-8.15 pg.128](#)), `HColon`([I-8.2 pg.121](#))

## I-3.31 ColumnVectors [OBSOLETE]

syntax

[OBSOLETE]

## Description

[OBSOLETE] Essentially replaced by “`GensAsCols`, `GensAsRows`” ([I-7.9 pg.111](#)) and “`SubmoduleCols`, `SubmoduleRows`” ([I-19.45 pg.302](#)) **See Also:** `GensAsCols`, `GensAsRows`([I-7.9 pg.111](#)), `SubmoduleCols`, `SubmoduleRows`([I-19.45 pg.302](#))

## I-3.32 CommonDenom

syntax

`CommonDenom(RINGELEM f): RINGELEM`

## Description

This function returns a common denominator for the polynomial “ $f$ ”. The coefficient ring of the polynomial ring to which “ $f$ ” belongs must be a fraction field.

example

```
/**/ use P := QQ[x,y];
/**/ f := (1/4)*x+(1/6)*y;
/**/ CommonDenom(f);
12
/**/ CommonDenom(2*x);
1
```

**See Also:** `content`([I-3.46 pg.64](#)), `ContentWRT`([I-3.48 pg.65](#))

## I-3.33 Comp [OBSOLETE]

syntax

[OBSOLETE]

## Description

[OBSOLETE] please use “[...]” for accessing entries in a list by index, or the record field selector operator.

**See Also:** operators, shortcuts(I-0.1 pg.25), record field selector(I-18.24 pg.265)

## I-3.34 Comparison Operators

syntax

```
A < B
A > B
A <= B
A >= B
A = B
A <> B
return BOOL
```

## Description

These operators perform the corresponding comparison between “A” and “B” returning “true” or “false”; they will signal an error if “A” and “B” are not comparable.

example

```
/**/ 1 <> 2; -- "not equal"
true
/**/ "abc" < "def"; -- lex ordering for strings
true
```

**See Also:** Equality Test(I-5.9 pg.86), operators, shortcuts(I-0.1 pg.25)

## I-3.35 CompleteToOrd [OBSOLESCENT]

syntax

```
[OBSOLESCENT]
```

## Description

Renamed “MakeTermOrd” (I-13.5 pg.192).

## I-3.36 compts

syntax

```
compts(V: MODULEELEM): LIST
Comps(V: MODULEELEM): LIST
```

## Description

This function returns the list of components of a ModuleElem V. It is like converting a ModuleElem into a generic list. Note that a ModuleElem is a more structured object than a generic list.

example

```
/**/ use R ::= QQ[x,y,z];
/**/ R3 := NewFreeModule(R,3);
```

```

/**/ V := ModuleElem(R3, [3*x^2+4*y, 2*x-5*z^3, 2*x+2*y]); V;
[3*x^2 +4*y, -5*z^3 +2*x, 2*x +2*y]
/**/ type(V);
MODULEELEM

/**/ compts(V);
[3*x^2 +4*y, -5*z^3 +2*x, 2*x +2*y]
/**/ type(compts(V));
LIST

```

**See Also:** NumCompts(I-14.35 pg.223)

### I-3.37 ComputeElimFirst

syntax

```
ComputeElimFirst(I: IDEAL, PP: ElimIndsProd): RINGELEM
```

#### Description

This function is experimental and dangerous! No guarantees.

Manual is intentionally cryptic. If you think this function is useful for you, write email to Anna M. Bigatti.

example

```

/**/ use ZZ/(32003)[x,y,t];
/**/ ComputeElimFirst(t, ideal(x^2-t^2, y^3-t^3));
x^6 -y^6

/**/ Use ZZ/(32003)[x,y,t,h];
/**/ ComputeElimFirst(t, ideal(x*h-t^2, y*h^2-t^3));
x^3*h^2 -y^2*h^3

```

**See Also:** elim(I-5.4 pg.84)

### I-3.38 concat

syntax

```
concat(L_1: LIST,...,L_n: LIST): LIST
```

#### Description

This function returns the list obtained by concatenating the lists “L<sub>1</sub>, ..., L<sub>n</sub>”.

NOTE: to concatenate strings just use “+”.

example

```

/**/ concat([1,2,3],[4,5],[],[6]);
[1, 2, 3, 4, 5, 6]

```

**See Also:** append(I-1.12 pg.31), ConcatLists(I-3.43 pg.62), String Operations(III-4.2 pg.383)

### I-3.39 ConcatAntiDiag

syntax

```
ConcatAntiDiag(A: MAT, B: MAT): MAT
```

## Description

This function creates a simple block matrix. The two entries are matrices. ConcatAntiDiag(A, B) will return a matrix of the form

$$\begin{bmatrix} | & 0 & A & | \\ | & B & 0 & | \end{bmatrix}$$

example

```
/**/ A := mat([[1,2,3], [4,5,6]]);
/**/ B := mat([[101,102], [103,104]]);
/**/ ConcatAntiDiag(A, B);
matrix(QQ,
  [[0, 0, 1, 2, 3],
   [0, 0, 4, 5, 6],
   [101, 102, 0, 0, 0],
   [103, 104, 0, 0, 0]])
```

**See Also:** BlockMat([I-2.9](#) pg.41), ConcatDiag([I-3.40](#) pg.61), ConcatHor([I-3.41](#) pg.61), ConcatVer([I-3.44](#) pg.63)

## I-3.40 ConcatDiag

syntax

```
ConcatDiag(A: MAT, B: MAT): MAT
```

## Description

This function creates a simple block matrix. The two entries are matrices. ConcatDiag(A, B) will return a matrix of the form

$$\begin{bmatrix} | & A & 0 & | \\ | & 0 & B & | \end{bmatrix}$$

example

```
/**/ A := mat([[1,2,3], [4,5,6]]);
/**/ B := mat([[101,102], [103,104]]);
/**/ ConcatDiag(A, B);
matrix(QQ,
  [[1, 2, 3, 0, 0],
   [4, 5, 6, 0, 0],
   [0, 0, 0, 101, 102],
   [0, 0, 0, 103, 104]])
```

**See Also:** BlockMat([I-2.9](#) pg.41), ConcatAntiDiag([I-3.39](#) pg.60), ConcatHor([I-3.41](#) pg.61), ConcatVer([I-3.44](#) pg.63), DiagMat([I-4.15](#) pg.79)

## I-3.41 ConcatHor

syntax

```
ConcatHor(A: MAT, B: MAT): MAT
```

where A and B have the same number of rows

## Description

This function creates a simple block matrix. The two entries are matrices with the same number of rows. `ConcatHor(A, B)` will return a matrix of the form

$$\begin{bmatrix} | & A & B & | \end{bmatrix}$$

example

```
/**/ A := mat([[1,2,3], [4,5,6]]);
/**/ B := mat([[101,102], [103,104]]);
/**/ ConcatHor(A, B);
matrix(QQ,
  [[1, 2, 3, 101, 102],
   [4, 5, 6, 103, 104]])
```

**See Also:** [BlockMat\(I-2.9 pg.41\)](#), [MakeMatByRows](#), [MakeMatByCols\(I-13.2 pg.191\)](#), [ConcatAntiDiag\(I-3.39 pg.60\)](#), [ConcatDiag\(I-3.40 pg.61\)](#), [ConcatHorList\(I-3.42 pg.62\)](#), [ConcatVer\(I-3.44 pg.63\)](#), [RowMat\(I-18.56 pg.279\)](#)

## I-3.42 ConcatHorList

syntax

```
ConcatHorList(L: LIST of MAT): MAT
```

where the matrices in L have the same number of rows

## Description

This function creates a simple block matrix. The entries in the list are matrices with the same number of rows. `ConcatHorList(L)` will return a matrix of the form

$$\begin{bmatrix} | & L[1] & L[2] & \dots & L[N] & | \end{bmatrix}$$

example

```
/**/ L := [ mat([[1,2,3], [4,5,6]]), mat([[101,102], [103,104]]) ];
/**/ ConcatHorList(L);
matrix(QQ,
  [[1, 2, 3, 101, 102],
   [4, 5, 6, 103, 104]])
```

**See Also:** [BlockMat\(I-2.9 pg.41\)](#), [MakeMatByRows](#), [MakeMatByCols\(I-13.2 pg.191\)](#), [ConcatAntiDiag\(I-3.39 pg.60\)](#), [ConcatDiag\(I-3.40 pg.61\)](#), [ConcatHor\(I-3.41 pg.61\)](#), [ConcatVerList\(I-3.45 pg.63\)](#), [RowMat\(I-18.56 pg.279\)](#)

## I-3.43 ConcatLists

syntax

```
ConcatLists(L: LIST of LISTs): LIST
```

## Description

This function takes 1 argument, a list whose entries are lists, and returns the concatenation of its entries.

example

```

/**/ L := [[1,2],["abc","def"],[3,4]];
/**/ ConcatLists(L);
[1, 2, "abc", "def", 3, 4]

```

**See Also:** [append\(I-1.12 pg.31\)](#), [concat\(I-3.38 pg.60\)](#)

## I-3.44 ConcatVer

syntax

```

ConcatVer(A: MAT, B: MAT): MAT

```

where A and B have the same number of columns

### Description

This function creates a simple block matrix. The two entries are matrices with the same number of columns. ConcatVer(A, B) will return a matrix of the form

$$\begin{bmatrix} | & A & | \\ | & B & | \end{bmatrix}$$

example

```

/**/ A := mat([[1,2,3], [4,5,6]]);
/**/ B := mat([[101,102,103]]);
/**/ ConcatVer(A, B);
matrix(QQ,
  [[1, 2, 3],
   [4, 5, 6],
   [101, 102, 103]])

```

**See Also:** [BlockMat\(I-2.9 pg.41\)](#), [ColMat\(I-3.29 pg.57\)](#), [MakeMatByRows](#), [MakeMatByCols\(I-13.2 pg.191\)](#), [ConcatAntiDiag\(I-3.39 pg.60\)](#), [ConcatDiag\(I-3.40 pg.61\)](#), [ConcatHor\(I-3.41 pg.61\)](#), [ConcatVerList\(I-3.45 pg.63\)](#)

## I-3.45 ConcatVerList

syntax

```

ConcatVerList(L: LIST of MAT): MAT

```

where the matrices in L have the same number of columns

### Description

This function creates a simple block matrix. The entries in the list are matrices with the same number of columns. ConcatVer(L) will return a matrix of the form

$$\begin{bmatrix} | & L[1] & | \\ | & L[2] & | \\ | & \dots & | \end{bmatrix}$$

example

```

/**/ L := [ mat([[1,2,3], [4,5,6]]), mat([[101,102,103]]) ];
/**/ ConcatVerList(L);

```

```
matrix(QQ,
  [[1, 2, 3],
   [4, 5, 6],
   [101, 102, 103]])
```

**See Also:** BlockMat(I-2.9 pg.41), ColMat(I-3.29 pg.57), MakeMatByRows, MakeMatByCols(I-13.2 pg.191), ConcatAntiDiag(I-3.39 pg.60), ConcatDiag(I-3.40 pg.61), ConcatVer(I-3.44 pg.63), ConcatHorList(I-3.42 pg.62)

## I-3.46 content

syntax

```
content(F: RINGELEM): RINGELEM
```

### Description

This function returns the content of “F”. The returned value is a RingElem in “CoeffRing(RingOf(F))”.

If the coefficient ring is a true GCD domain, the result is the standard content (i.e. a gcd of its coefficients). If the coefficient ring is a fraction field then the result is fraction “c” such that “f/c” is a primitive polynomial.

example

```
/**/ use P := QQ[x,y,z];
/**/ F := 1234*x^3*z + 3456*x*y*z^3 + 5678*y^2*z;
/**/ content(F);
2
/**/ RingOf(It);
QQ
/**/ content(4*x/5 + 2);
2/5
```

**See Also:** ContentWRT(I-3.48 pg.65), coefficients(I-3.24 pg.54)

## I-3.47 ContentFreeFactor

syntax

```
ContentFreeFactor(F: RINGELEM): RECORD
```

### Description

This function returns a factorization of the multivariate polynomial “F” into (polynomial) content-free factors; it works by calling ContentWRT repeatedly. The multiplicities will always be 1.

A polynomial which is (polynomial) content-free means that all its irreducible factors involve all indeterminate appearing the polynomial itself.

example

```
/**/ use P := QQ[x,y,z];
/**/ f := 2*(x+1)*(y+2)*(x+y)^2*(x-y);
/**/ indent(ContentFreeFactor(f));
record[
  RemainingFactor := 2,
  factors := [y +2, x +1, x^3 +x^2*y -x*y^2 -y^3],
  multiplicities := [1, 1, 1]
]
```

**See Also:** ContentWRT(I-3.48 pg.65), factor(I-6.1 pg.93), SqFreeFactor(I-19.30 pg.295)

## I-3.48 ContentWRT

syntax

```
ContentWRT(F: RINGELEM, X: RINGELEM): RINGELEM
ContentWRT(F: RINGELEM, L: LIST of RINGELEM): RINGELEM
```

### Description

This function returns the (polynomial) content of “F” (i.e. a gcd of its coefficients) seen as a polynomial in the indeterminate “X”, or as a polynomial in all the indeterminates in “L”.

The returned value is a RingElem in RingOf(F).

example

```
/**/ use P := QQ[x,y,z];
/**/ f := x^3*z + x*y*z^3 + 2*z;
/**/ Cx := CoefficientsWRT(f, x);
/**/ indent(Cx);
[
  record[PP := 1, coeff := 2*z],
  record[PP := x, coeff := y*z^3],
  record[PP := x^3, coeff := z]
]
/**/ ContentWRT(f, x);
z
/**/ ContentWRT(f, [x]);
z
```

**See Also:** CoefficientsWRT(I-3.25 pg.55), content(I-3.46 pg.64), monomials(I-13.31 pg.204)

## I-3.49 ContFrac

syntax

```
ContFrac(X: RAT): LIST of INT
```

### Description

“ContFrac” returns a list of the continued fraction “*quotients*” for the given rational number “X”.

example

```
/**/ ContFrac(1.414213);
[1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 4, 1, 1, 1, 1, 1, 2, 1, 6]
```

**See Also:** CFApprox(I-3.8 pg.48), CFApproximants(I-3.9 pg.49), ContFracToRat(I-3.50 pg.65)

## I-3.50 ContFracToRat

syntax

```
ContFracToRat(L: LIST of INT): RAT
```

### Description

“ContFracToRat” returns the rational number equal to the continued fraction whose quotients are given as input. The quotients must all be integers, only the very first may be non-positive.

example

```
/**/ ContFracToRat([1, 2, 2, 2, 2, 2, 2, 2]);
577/408
```

**See Also:** ContFrac([I-3.49](#) pg.65), CFApprox([I-3.8](#) pg.48), CFApproximants([I-3.9](#) pg.49)

## I-3.51 continue

syntax

```
continue
```

### Description

This command must be used inside a loop statement (“for”, “foreach”, “repeat”, or “while”). When executed, the current loop iteration is terminated, and the control passes directly to the next iteration.

In the case of nested loops “continue” refers only to iterations of the innermost loop in which it appears; to affect loops outside the innermost one, you must use “break” ([I-2.12](#) pg.43) to break out of the current loop command.

example

```
/**/ for i := 5 to 1 step -1 do
/**/   for j := 1 to 4 do
/**/     if i = j then continue; endif;
/**/     print j, " ";
/**/   endfor;
/**/   println;
/**/ endfor;
1 2 3 4
1 2 3
1 2 4
1 3 4
2 3 4
```

**See Also:** break([I-2.12](#) pg.43), return([I-18.39](#) pg.271), syntax(?? pg.??)

## I-3.52 count

syntax

```
count(L: LIST, E: OBJECT): INT
```

### Description

This function counts the number of occurrences of the object E in the list L.

example

```
/**/ L := [1,2,3,2,[2,3]];
/**/ count(L,2);
2

/**/ count(L,[2,3]);
1

/**/ count(L,"a");
0
```

**See Also:** [distrib\(I-4.19 pg.81\)](#), [len\(I-12.6 pg.181\)](#)

## I-3.53 CpuTime

— syntax —

CpuTime(): RAT

### Description

This function returns a “RAT” whose value is the user CPU usage in seconds since the start of the program: this is the amount of time the processor has dedicated to your computation, and may be rather less than the real elapsed time if the computer is also busy with other tasks.

The most common usage is with “TimeFrom” ([I-20.6 pg.313](#)) as shown in the example; it automatically computes the time difference and returns it as decimal string.

— example —

```
/**/ StartTime := CpuTime(); -- time in seconds since the start (a RAT)
/**/ --
/**/ -- .... long computation ....
/**/ --
/**/ PrintLn "Computation time: ", TimeFrom(StartTime);

/**/ -- Alternative use: compute TimeTaken as a RAT
/**/ StartTime := CpuTime();
/**/ --
/**/ -- .... long computation ....
/**/ --
/**/ EndTime := CpuTime();
/**/ TimeTaken := EndTime - StartTime; --> ugly if printed directly
/**/ PrintLn "Computation time: ", DecimalStr(TimeTaken);
```

You can use “DecimalStr” ([I-4.3 pg.71](#)) to see the value of “CpuTime” in a more easily comprehensible form.

**See Also:** [TimeFrom\(I-20.6 pg.313\)](#), [DecimalStr\(I-4.3 pg.71\)](#)

## I-3.54 CRT

— syntax —

CRT(R1: INT, M1: INT, R2: INT, M2: INT): RECORD

### Description

This function combines two residue-modulus pairs “(R1,M1)” and “(R2,M2)” using the Chinese Remainder Theorem to produce a single residue-modulus pair “(R,M)” such that “R = R1 mod M1” and “R = R2 mod M2”, and “|R| < M”. The moduli “M1” and “M2” must be coprime (hence “M = M1\*M2”).

— example —

```
/**/ CRT(2,3, 4,5);
record[modulus := 15, residue := -1]
```

**See Also:** [CRTPoly\(I-3.55 pg.68\)](#), [RatReconstructByContFrac](#), [RatReconstructByLattice\(I-18.16 pg.260\)](#)

## I-3.55 CRTPoly

syntax

```
CRTPoly(f1: RINGELEM, M1: INT, f2: RINGELEM, M2: INT): RECORD
```

### Description

This function combines residue-modulus pairs “(f1,M1)” and “(f2,M2)” using the Chinese Remainder Theorem to produce a single residue-modulus pair “(f,M)” such that “f” is a polynomial (with coefficients in “QQ”), “f = f1 mod M1” and “f = f2 mod M2”, and all coefficients of “f” are smaller than “M”. The moduli “M1” and “M2” must be coprime (hence “M = M1\*M2”).

example

```
/**/ use QQ[x,y];
/**/ CRTPoly(x-y, 331, x+y, 10093);
record[modulus := 3340783, residue := x +676232*y]
/**/ mod(676232, 331);
330
/**/ mod(676232, 10093);
1
```

**See Also:** CRT([I-3.54](#) pg.67), RatReconstructPoly([I-18.17](#) pg.261)

## I-3.56 CurrentRing

syntax

```
CurrentRing
```

### Description

This is a top-level SYSTEM VARIABLE containing the current ring.

NOTE: in CoCoA-4 it used to be a function (namely “CurrentRing()”), now it is a top-level “*variable*” which needs to be imported in functions... but beware: this is to be considered BAD STYLE ;-)

example

```
/**/ use R ::= QQ[x,y];
/**/ use S ::= ZZ/(3)[t];
/**/ CurrentRing;
RingDistrMPolyClean(FFp(3), 1)

/**/ use R;
/**/ CurrentRing;
RingDistrMPolyClean(QQ, 2)

/**/ Define MyIndets1()
/**/   TopLevel CurrentRing; -- importing a top-level (global) variable
/**/   Return indets(CurrentRing);
/**/ EndDefine;

/**/ Define MyIndets2(val)
/**/   Return indets(RingOf(val)); -- cleaner: depends only on the argument
/**/ EndDefine;

/**/ MyIndets1();
[x, y]
```

```
/**/ MyIndets2(ideal(x));
[x, y]
```

**See Also:** [RingOf\(I-18.46 pg.275\)](#), [TopLevel\(I-20.10 pg.314\)](#)

## I-3.57 *CurrentTypes*

syntax

```
CurrentTypes(): LIST of TYPE
```

### Description

This function lists all CoCoA data types.

example

```
/**/ CurrentTypes();
[BOOL, ERROR, FUNCTION, ...]
```

## I-3.58 *cyclotomic*

syntax

```
cyclotomic(n: INT, x: RINGELEM): RINGELEM
```

### Description

This function computes the “**n**”-th cyclotomic polynomial (in the indeterminate “**x**”).

example

```
/**/ use QQ[z];
/**/ cyclotomic(4,z);
z^2 + 1
```



# Chapter I-4

## D

### I-4.1 dashes

syntax

```
dashes()
```

#### Description

This function returns a string of dashes:

example

```
/**/ dashes(); 1+1;
-----
2
```

### I-4.2 date

syntax

```
date() : INT
```

#### Description

This function returns the date.

Note that from version 5.0.4 the result is an INT and the date is in the form YYYYMMDD. See also “TimeOfDay” ([I-20.7 pg.313](#)).

example

```
/**/ date();
20130530
```

**See Also:** TimeOfDay([I-20.7 pg.313](#))

### I-4.3 DecimalStr

syntax

```
DecimalStr(X: INT|RAT|RINGELEM): STRING
DecimalStr(X: INT|RAT|RINGELEM, NumDigits: INT): STRING
```



```

/**/ strange();
1234
/**/ strange(1, 4, "a", 5, 10);
100
/**/ -- strange(1,2); --> !!! ERROR !!! as expected

```

2. SCOPE. Every variable defined or modified by the command sequence “C” is considered local to the function unless the variable is global or relative to a “ref” (I-18.26 pg.265) parameter.

See “ref” (I-18.26 pg.265) to learn about passing function arguments “*by reference*”, i.e. so that the function can change the value of an existing variable.

See “TopLevel” (I-20.10 pg.314) for the use of global variables.

example

```

/**/ Define Example_1(L)
/**/   L := L + 5;
/**/   Return L;
/**/ EndDefine;

/**/ L := 0;
/**/ Example_1(L);
5
/**/ L; -- L is unchanged despite the function call.
0

```

3. VARIABLE NUMBER OF PARAMETERS. It is also possible to have some optional arguments or a variable number of arguments:

example

```

-- OPTIONAL ARGUMENTS must be in the last positions

/**/ define deg0(f, opt x)
/**/   if f=0 then return 0; endif;
/**/   if IsDefined(x) then return deg(f,x); endif;
/**/   return deg(f);
/**/ enddefine;

/**/ use P ::= QQ[x,y,z];
/**/ deg0(zero(P));
0
/**/ deg0(x^2+y);
2
/**/ deg0(x^2+y, y);
1

-- VARIABLE number of ARGUMENTS

/**/ Define MySum(...) --> arguments are in the LIST "ARGV"
/**/   If len(ARGV) = 0 Then Return 12345; EndIf;
/**/   ans := 0;
/**/   Foreach N In ARGV Do ans := ans+N; EndForeach;
/**/   Return ans;
/**/ EndDefine;

/**/ MySum(1,2,3,4,5);
15
/**/ MySum();

```

12345

The old statement, “Help S;” is OBSOLETE!

**See Also:** [return\(I-18.39 pg.271\)](#), [TopLevel\(I-20.10 pg.314\)](#), [ref\(I-18.26 pg.265\)](#), [syntax\(?? pg.??\)](#)

## I-4.5 DefiningIdeal

syntax

DefiningIdeal(S: RING): IDEAL

### Description

When “S” is a quotient ring, say “S = R/I”, this function returns “I”, the ideal which defines “S”.

example

```

/**/ use R ::= QQ[x,y,z];
/**/ S := R/ideal(x);
/**/ DefiningIdeal(S);
ideal(x)

```

**See Also:** [CanonicalRepr\(I-3.4 pg.47\)](#), [InducedHom\(I-9.27 pg.146\)](#), [NewQuotientRing\(I-14.9 pg.212\)](#)

## I-4.6 deg

syntax

deg(F: RINGELEM): INT  
deg(F: RINGELEM, X: RINGELEM): INT

### Description

The first form of this function returns the “*standard degree*” of “F” (see “wdeg” ([I-23.1 pg.331](#)) for the “*weighted degree*”). The second form returns the exponent of the indeterminate “X” in “F”.

For the degree of a ring or quotient, see “multiplicity” ([I-13.35 pg.207](#)).

example

```

/**/ use R ::= QQ[x,y,z];
/**/ deg(x*y^2+y);
3

/**/ deg(x*y^2+y, x);
1

/**/ Ws := RowMat([2,3,1]);
/**/ P := NewPolyRing(QQ, "x,y,z", MakeTermOrd(Ws), 1);
/**/ use P;
/**/ deg(x*y^2+y);
3
/**/ wdeg(x*y^2+y);
[8]
/**/ deg(x*y^2+y, x);
1
/**/ deg(x*y^2+y, x);
2

```

**See Also:** [wdeg\(I-23.1 pg.331\)](#), [NewPolyRing\(I-14.8 pg.212\)](#), [multiplicity\(I-13.35 pg.207\)](#)

## I-4.7 den

syntax

```
den(X: INT|RAT): INT
den(X: RINGELEM): RINGELEM
```

### Description

This function returns the denominator of the argument “X”. If “X” is a “RINGELEM” in “FractionField(R)”, then “den(X)” is a RingElem in “R”.

NOTE: In CoCoA 4 the numerator and denominator could also be found using the suffixes “.Num” and “.Den”; this fragile syntax is now obsolete.

example

```
/**/ den(3);
1

/**/ P := QQ[x,y];
/**/ F := NewFractionField(P);
/**/ use F;
/**/ den(x/(x+y));
x +y
/**/ RingOf(It);
RingWithID(4, "QQ[x,y]")
```

**See Also:** num([I-14.33](#) pg.223)

## I-4.8 DensePoly

syntax

```
DensePoly(R: RING, N: INT): RINGELEM
```

### Description

This function returns the sum of all power-products of (standard) degree “N”.

example

```
/**/ use R := QQ[x,y];
/**/ DensePoly(R,3);
x^3 + x^2*y + x*y^2 + y^3

/**/ Weights := RowMat([2,3]);
/**/ P := NewPolyRing(QQ, "x,y", MakeTermOrd(Weights), 1);
/**/ use P;
/**/ DensePoly(P,1); // NOTE: standard degree!!
y +x
```

## I-4.9 depth

syntax

```
depth(I: IDEAL, M: TAGGED("Quotient")): INT
depth(RmodI: Quotient RING): INT
```

## Description

This function calculates the depth of  $M$  in the ideal  $I$ , i.e. the length of a maximal  $I$ -regular sequence in  $M$ . In the second form, where the ideal “ $I$ ” is not specified, it assumes that “ $I$ ” is the maximal ideal generated by the indeterminates, i.e. “`ideal(Indets())`”.

Note that if “ $M$ ” is homogeneous and “ $I$ ” is the maximal ideal, then it uses the Auslander-Buchsbaum formula “`depth_I(M) = N - pd(M)`” where “ $N$ ” is the number of indeterminates and “`pd`” is the projective dimension, otherwise (\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*) it returns “`min{N | Ext^N(R/I, M) <> 0}`” using the function “`Ext`” (I-5.17 pg.90).

### example

```

/**/ use P ::= QQ[x,y,z];
/**/ depth(P); -- the (x,y,z)-depth of the entire ring is 3
3

/**/ I := ideal(x^5,y^3,z^2);
/**/ depth(P/I);
0

//----- ***** NOT YET IMPLEMENTED ***** ----->>
N := Module([x^2,y], [x+z,0]);
depth(I, P^2/N); --- a max reg sequence would be (z^2,y^3)
2
-----
use P ::= QQ[x,y,z,t,u,v];
-- Cauchy-Riemann system in three complex vars!
N := Module([x,y], [-y,x], [z,t], [-t,z], [u,v], [-v,u]);
--- is it CM?
depth(P^2/N);
3
-----
dim(P^2/N);
3
-----
--- yes!

M := Module([x,y,z], [t,v,u]);
res(P^3/M);
0 --> P^2(-1) --> P^3
-----
depth(P^3/M); -- using Auslander Buchsbaum 6-1=5
5
-----
dim(P^3/M); -- not CM
6
-----
depth(ideal(x,y,z,t), P^2/N);
2
-----

```

**See Also:** `res`(I-18.34 pg.269), `Ext`(I-5.17 pg.90)

## I-4.10 deriv

### syntax

```
deriv(F: RINGELEM, X: RINGELEM): RINGELEM
```

## Description

This function returns the derivative of  $F$  with respect to the indeterminate  $X$ .

example

```

/**/ use R := QQ[x,y];
/**/ deriv(x*y^2, x);
y^2

/**/ define Jac(f) --> The Jacobian matrix for a polynomial.
/**/   return matrix([[deriv(f, X) | X in indets(RingOf(f))]]);
/**/ enddefine;

/**/ Jac(x*y^2);
matrix( /*RingDistrMPolyClean(QQ, 2)*/
  [[y^2, 2*x*y]])

/**/ FrF := NewFractionField(R);
/**/ use FrF;
/**/ deriv((x*y^2)/(x-1), x);
(-y^2)/(x^2 -2*x +1)

```

**See Also:** [jacobian\(I-10.1 pg.175\)](#)

## I-4.11 DerivationAction

syntax

```

DerivationAction(D: RINGELEM, P: RINGELEM)

```

## Description

Thanks to Enrico Carlini.

Given the polynomial “P” and the derivation “D”, this function computes the action of “D” on “P”.

For the sake of simplicity Forms/Polynomials and Derivations live in the same ring, the distinction between them is purely formal.

example

```

/**/ use R := QQ[x,y,z];
/**/ DerivationAction(x*y*z, x^3+x*y*z);
1

```

**See Also:** [InverseSystem\(I-9.35 pg.150\)](#), [PerpIdealOfForm\(I-16.6 pg.234\)](#)

## I-4.12 describe

syntax

```

describe X: OBJECT

```

## Description

This command gives some information about the object “X”. For instance, if “X” is a CoCoA-5 function, it prints out the definition, and if “X” is a package name (prefixed with a “\$”), it prints out the exported names.

example

```

/**/ Define succ(N) Return N+1; EndDefine;
/**/ describe succ;
Define succ(N) Return N+1 EndDefine

/**/ describe $latex;
The package $latex exports the following names:
* LaTeX
* latex

The package $latex also has the following non-exported members:
...

```

## I-4.13 det

syntax

```
det(M: MAT): RINGELEM
```

### Description

This function returns the determinant of the matrix “M”.

example

```

/**/ use R := QQ[x];
/**/ M := mat(R, [[x,x^2], [x,x^3]]);
/**/ det(M);
x^4 -x^3

/**/ det(mat(QQ, [[1,2], [0,5]]));
5

```

**See Also:** minors(I-13.21 pg.200)

## I-4.14 DF

syntax

```
DF(F: RINGELEM): RINGELEM
```

### Description

Same as “LF” (I-12.9 pg.183), but does not throw an error if the argument is zero or if the “GradingDim” (I-7.29 pg.117) of the polynomial ring is 0. As defined in Kreuzer-Robbiano book II (Definition 4.2.8).

example

```

/**/ use R := QQ[x,y];
/**/ DF(x^2 -x*y +2*x -1);
x^2 -x*y

/**/ use R := QQ[x,y], Lex; -- GradingDim is 0: everything is homogeneous
/**/ DF(x^2 -x*y +2*x -1);
x^2 -x*y +2*x -1

/**/ P := NewPolyRing(QQ, IndetSymbols(R), mat([[1,4],[1,0]]), 1);
/**/ use P;
/**/ DF(x^2 -x*y);

```

```
-x*y
/**/  DF(x^4 +x^2 -y);
x^4 -y
```

**See Also:** [LF\(I-12.9 pg.183\)](#)

## I-4.15 *DiagMat*

— *syntax* —

```
DiagMat(L: LIST): MAT
DiagMat(R: RING, L: LIST): MAT
```

### Description

This function returns the diagonal matrix whose diagonal are the elements of the list L.

— *example* —

```
/**/  DiagMat([3,4,5]);
matrix(
[
  [3, 0, 0],
  [0, 4, 0],
  [0, 0, 5]
])

/**/  DiagMat(QQ,[5,6,7]);
matrix(
[
  [5, 0, 0],
  [0, 6, 0],
  [0, 0, 7]
])

-- fast implementation for high powers of a diagonal matrix
/**/  Define PowerDiag(M, Exp)
/**/    If not(IsDiagonal(M)) Then
/**/      error("PowerDiag: matrix must be diagonal");
/**/    EndIf;
/**/    Return DiagMat([ M[I, I]^Exp | I in 1..NumRows(M) ]);
/**/  EndDefine;

/**/  PowerDiag(IdentityMat(QQ,3), 200000000);
matrix(QQ,
[[1, 0, 0],
 [0, 1, 0],
 [0, 0, 1]])
```

**See Also:** [BlockMat\(I-2.9 pg.41\)](#), [IsDiagonal\(I-9.44 pg.153\)](#), [ColMat\(I-3.29 pg.57\)](#), [RowMat\(I-18.56 pg.279\)](#)

## I-4.16 *diff*

— *syntax* —

```
diff(L: LIST, M: LIST): LIST
```

## Description

This function returns the list obtained by removing all the elements of M from L.

example

```
/**/ L := [1,2,3,2,[2,3]];
/**/ M := [1,2];
/**/ diff(L, M);
[3, [2, 3]]
```

**See Also:** [remove\(I-18.32 pg.268\)](#)

## I-4.17 dim

syntax

```
dim(R: RING or TAGGED("Quotient")): INT
```

## Description

This function computes the dimension of R.

The coefficient ring must be a field.

example

```
/**/ use R ::= QQ[x,y,z];
/**/ dim(R/ideal(x));
2

/**/ dim(R/ideal(y^2-x, x*z-y^3));
1
```

## I-4.18 discriminant

syntax

```
discriminant(F: RINGELEM): RINGELEM
discriminant(F: RINGELEM, X: RINGELEM): RINGELEM
```

## Description

This function computes the discriminant of a polynomial F (with respect to a given indeterminate X, if the polynomial is multivariate). If the polynomial is univariate then there is no need to specify which indeterminate to use.

The discriminant is defined as

$$(-1)^{(N*(N-1)/2)} \det(M) / M[1,1]$$

where  $M := \text{Sylvester}(F, \text{deriv}(F, X), X)$  and  $N := \deg(F, X)$ .

example

```
/**/ Use R ::= QQ[x,y];
/**/ discriminant(x^2+3*y^2, x);
-12*y^2

/**/ discriminant(x^2+3*y^2, y);
-12*x^2
```

```
/**/ discriminant((x+1)^20+2);
54975581388800000000000000000000
```

**See Also:** [resultant\(I-18.38 pg.271\)](#)

## I-4.19 distrib

— syntax —

```
distrib(L: LIST): LIST
```

### Description

For each object  $E$  of a list  $L$ , let  $N(E)$  be the number of times  $E$  occurs as a component of  $L$ . Then  $\text{Distrib}(L)$  returns the list whose components are  $[E, N(E)]$ .

— example —

```
/**/ distrib(["b","a","b",4,4,[1,2]]);
[["b", 2], ["a", 1], [4, 2], [[1, 2], 1]]
```

**See Also:** [count\(I-3.52 pg.66\)](#)

## I-4.20 div

— syntax —

```
div(N: INT, D: INT): INT
```

### Description

We define the quotient “ $Q$ ” and remainder “ $R$ ” to be integers which satisfy  $N = Q * D + R$  with  $0 \leq R < |D|$ . Then “ $\text{div}(N, D)$ ” returns “ $Q$ ” while “ $\text{mod}(N, D)$ ” returns “ $R$ ”.

NOTE: To perform the division algorithm on a polynomial use “ $\text{NR}$ ” ([I-14.32 pg.222](#)) (normal remainder) to find the remainder, or “ $\text{DivAlg}$ ” ([I-4.21 pg.81](#)) to get both quotients and remainder. To determine if a polynomial is in a given ideal or a vector is in a given module, use “ $\text{NF}$ ” ([I-14.16 pg.215](#)) or “ $\text{IsIn}$ ” ([I-9.53 pg.157](#)), and to find a representation in terms of the generators “ $\text{GenRepr}$ ” ([I-7.7 pg.110](#)).

— example —

```
/**/ div(10,3);
3
/**/ mod(10,3);
1
```

**See Also:** [DivAlg\(I-4.21 pg.81\)](#), [GenRepr\(I-7.7 pg.110\)](#), [NF\(I-14.16 pg.215\)](#), [NR\(I-14.32 pg.222\)](#), [mod\(I-13.26 pg.202\)](#)

## I-4.21 DivAlg

— syntax —

```
DivAlg(X: RINGELEM, L: LIST of RINGELEM): RECORD
DivAlg(X: MODULEELEM, L: LIST of MODULEELEM): RECORD
```

## Description

This function performs the division algorithm on  $X$  with respect to  $L$ . It returns a record with two fields: “Quotients” holding a list of polynomials, and “Remainder” holding the remainder of  $X$  upon division by  $L$ .

example

```

/**/ use R := QQ[x,y,z];
/**/ F := x^2*y + x*y^2 + y^2;
/**/ L := [x*y-1, y^2-1];
/**/ DivAlg(F, L);
record[quotients := [x + y, 1], remainder := x + y + 1]

/**/ D := It;
/**/ D.quotients;
[x + y, 1]
/**/ D.remainder;
x + y + 1
/**/ ScalarProduct(D.quotients, L) + D.remainder = F;
true

/**/ R2 := NewFreeModule(R,2);
/**/ V := ModuleElem(R2, [x^2+y^2+z^2, x*y*z]);
/**/ L := gens(SubmoduleRows(R2, mat([[x,y], [y,z], [z,x]])));
/**/ D := DivAlg(V, L);
/**/ indent(D);
record[
  quotients := [x, -z^2 + y + z, y*z - y],
  remainder := [z^2, z^3 - y*z - z^2]
]
/**/ sum([D.quotients[i]*L[i] | i in 1..len(L)]) + D.remainder;
[x^2 + y^2 + z^2, x*y*z]

```

**See Also:** [div\(I-4.20 pg.81\)](#), [mod\(I-13.26 pg.202\)](#), [GenRepr\(I-7.7 pg.110\)](#), [NF\(I-14.16 pg.215\)](#), [NR\(I-14.32 pg.222\)](#)

## I-4.22 domain

syntax

```
domain(phi: RINGHOM): RING
```

## Description

This function returns the domain of the homomorphism “phi”

example

```

/**/ P := NewPolyRing(RingQQ(), "alpha,beta");
/**/ phi := CanonicalHom(RingZZ(), P);
/**/ domain(phi);
ZZ
/**/ psi := CoeffEmbeddingHom(P);
/**/ domain(psi);
QQ

```

**See Also:** [codomain\(I-3.21 pg.53\)](#), [Commands and Functions for RINGHOM\(III-10.3 pg.410\)](#), [Commands and Functions returning RINGHOM\(III-10.4 pg.410\)](#)

# Chapter I-5

## E

### I-5.1 E\_ [OBSOLETE]

— syntax —

[OBSOLETE]

#### Description

[OBSOLETE] Essentially replaced by “gens” ([I-7.8 pg.110](#)) of a FreeModule.

— example —

```
/**/ use R := QQ[x,y,z];
/**/ R5 := NewFreeModule(R,5);
/**/ e := gens(R5);
/**/ e[2];
[0, 1, 0, 0, 0]
```

**See Also:** gens([I-7.8 pg.110](#)), GensAsCols, GensAsRows([I-7.9 pg.111](#))

### I-5.2 eigenfactors

— syntax —

eigenvectors(M: MAT, X: RINGELEM): LIST of RINGELEM

#### Description

“M” must be a square matrix, and “X” an indeterminate.

This function determines the eigenfactors of “M”, i.e. the irreducible factors of “CharPoly” ([I-3.11 pg.49](#)) of “M”.

— example —

```
/**/ use R := QQ[x];
/**/ M := mat([[0,2,0,0],[1,0,0,0],[0,0,0,2],[0,0,1,1]]);
/**/ eigenfactors(M, x);
[x+1, x-2, x^2-2]
```

## I-5.3 eigenvectors

syntax

```
eigenvectors(M: MAT, X: RINGELEM): LIST of RECORD
```

### Description

“M” must be a matrix of numbers, and “X” an indeterminate.

This function determines the eigenvalues of “M”, and for each eigenvalue gives a basis of the corresponding eigenspace – note that the basis is probably not orthogonal. For irrational eigenvalues, the minimal polynomial of the eigenvalue is given (as a polynomial in “X”), along with the eigenvectors expressed in terms of a root of the minimal polynomial (represented as “X”).

example

```
/**/ use R ::= QQ[x];
/**/ M := mat([[1,2,3],[4,5,6],[7,8,9]]);
/**/ eigenvectors(M, x);
[record[MinPoly := x, eigenspace := matrix(QQ,
  [[-1],
   [2],
   [-1]]],
record[MinPoly := x^2 -15*x -18,
eigenspace := [[1, (1/8)*x +1/4, (1/4)*x -1/2]]]
]

/**/ M := mat([[0,2,0,0],[1,0,0,0],[0,0,0,2],[0,0,1,0]]);
  eigenvectors(M, x); -- two irrational eigenvalues, each with eigenspace of dimension 2
[record[MinPoly := x^2 -2, eigenspace := [[1, (1/2)*x, 0, 0], [0, 0, 1, (1/2)*x]]]
```

## I-5.4 elim

syntax

```
elim(X: RINGELEM, M: IDEAL): IDEAL
elim(L: LIST, M: IDEAL): IDEAL
elim(X: RINGELEM, M: MODULE): MODULE
elim(L: LIST, M: MODULE): MODULE
```

### Description

This function returns the ideal or module obtained by eliminating the indeterminate “X”, or all indeterminates in “L”, from “M”. The coefficient ring needs to be a field.

As opposed to this function, there is also the “*modifier*”, “elim”, used when constructing a ring (see “Term Orderings” (III-9.5 pg.404)).

example

```
/**/ use R ::= QQ[t,x,y,z];
/**/ E := elim(t, ideal(t^15+t^6+t-x, t^5-y, t^3-z));
/**/ indent(E);
ideal(
  -z^5 +y^3,
  -y^4 -y*z^2 +x*y -z^2,
  -x*y^3*z -y^2*z^3 -x*z^3 +x^2*z -y^2 -y,
  -y^2*z^4 -x^2*y^3 -x*y^2*z^2 -y*z^4 -x^2*z^2 +x^3 -y^2*z -2*y*z -z,
  y^3*z^3 -x*z^3 +y^3 +y^2
)
```

```

/**/ use R ::= QQ[t,s,x,y,z,w];
/**/ t..x;
[t, s, x]

/**/ elim(t..x, ideal(t-x^2*z*w, x^2-t, y^2*t-w)); -- Note the use of t..x.
ideal(-z*w^2 + w)

/**/ use R ::= QQ[t[1..2], x[1..4]];
/**/ I := ideal(x[1]-t[1]^4, x[2]-t[1]^2*t[2], x[3]-t[1]*t[2]^3, x[4]-t[2]^4);
/**/ elim(indets(R,"t"), I);
ideal(x[2]^4 -x[1]^2*x[4], -x[3]^4 +x[1]*x[4]^3)

```

**See Also:** [Term Orderings\(III-9.5 pg.404\)](#)

## I-5.5 *ElimHomogMat*

[syntax](#)

```
ElimHomogMat(ElimInd: LIST, W: MAT): MAT
```

### Description

This function returns a matrix for a term ordering eliminating the indeterminates with indices in “ElimInd” for inputs which are homogeneous wrt the weights in the matrix “W”. If you don’t understand what this means, just use “ElimMat” ([I-5.6 pg.85](#)) :-)

NOTE: This function used to be called “HomogElimMat” up to version 5.1.4, and had swapped arguments.

[example](#)

```

/**/ ElimHomogMat([2,3], mat([[1,5,2]]));
matrix(ZZ,
  [[1, 5, 2],
   [0, 1, 1],
   [0, 0, -1]])

```

**See Also:** [elim\(I-5.4 pg.84\)](#), [ElimMat\(I-5.6 pg.85\)](#)

## I-5.6 *ElimMat*

[syntax](#)

```
ElimMat(ElimInd: LIST of INT, N: INT): MAT
ElimMat(ElimInd: LIST of INT, W: MAT): MAT
```

### Description

This function returns an “NxN” matrix representing a term ordering for eliminating the indeterminates with indices in “ElimInd”.

In the second form, a weight matrix “W” is given; these weights are placed immediately below the first elimination row.

NOTE: This function used to have swapped arguments up to version 5.1.4. (e.g. “ElimMat(3, [2,3])”)

[example](#)

```

/**/ ElimMat([2,3], 3);
matrix(ZZ,
  [[0, 1, 1],

```

```

    [1, 1, 1],
    [0, 0, -1]])

/**/ ElimMat([2,3], mat([[1,5,2]]));
matrix(ZZ,
  [[0, 1, 1],
   [1, 5, 2],
   [0, 0, -1]])

```

**See Also:** [elim\(I-5.4 pg.84\)](#), [ElimHomogMat\(I-5.5 pg.85\)](#), [NewPolyRing\(I-14.8 pg.212\)](#)

## I-5.7 EmbeddingHom

syntax

```
EmbeddingHom(K: RING): RINGHOM
```

### Description

This function returns the embedding homomorphism of the fraction field “K”.

example

```

/**/ use P := QQ[x,y];
/**/ K := NewFractionField(P);
/**/ phi := EmbeddingHom(K); -- phi: P -> K
/**/ f := 2*x+3*y;
/**/ phi(f);
2*x +3*y
/**/ RingOf(phi(f));
RingWithID(5, "FractionField(RingWithID(4))")

```

**See Also:** [CanonicalHom\(I-3.3 pg.46\)](#)

## I-5.8 EqSet

syntax

```
EqSet(L: LIST, M: LIST): BOOL
```

### Description

This function returns true if “L” equals “M” as sets, otherwise it returns false.

example

```

/**/ L := [1,2,2];
/**/ M := [2,1];
/**/ EqSet(L, M);
true

```

**See Also:** [intersection\(I-9.32 pg.149\)](#), [IntersectList\(?? pg.??\)](#), [IsSubset\(I-9.83 pg.167\)](#)

## I-5.9 Equality Test

syntax

```

A = B
A <> B

```

```
return BOOL
```

## Description

The first form returns “true” if “A” is equal to “B”, otherwise it returns “false” (or signals an error if they are not comparable). The second form is the same as “not (A=B)”.

example

```
/**/ 1=2;
false
/**/ 1<>2;
true
```

**See Also:** Comparison Operators(I-3.34 pg.59), operators, shortcuts(I-0.1 pg.25)

## I-5.10 EquiIsoDec

syntax

```
EquiIsoDec(I: IDEAL): LIST of IDEAL
```

## Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This function computes an equidimensional isoradical decomposition of I, i.e. a list of unmixed ideals  $I_1, \dots, I_k$  such that the radical of I is the intersection of the radicals of  $I_1, \dots, I_k$ . Redundancies are possible.

NOTE: at the moment, this implementation works only if the coefficient ring is the rationals or has large enough characteristic.

example

```
use R := QQ[x,y,z];
I := intersect(ideal(x-1,y-1,z-1), ideal(x-2,y-2)^2, ideal(x)^3);
H := EquiIsoDec(I);
H;
[ideal(x), ideal(z - 1, y - 1, x - 1), ideal(xy - y^2 - 2x + 2y, x^2 -
y^2 - 4x + 4y, y^2z - y^2 - 4yz + 4y + 4z - 4, y^3 - 5y^2 + 8y - 4, x
- 2)]
-----
T := [radical(J) | J in H];
S := IntersectionList(T);
radical(I) = S;
True
-----
```

**See Also:** PrimaryDecomposition(I-16.22 pg.241), radical(I-18.1 pg.255), RadicalOfUnmixed(I-18.2 pg.255)

## I-5.11 error

syntax

```
error(S: STRING): ERROR
```

## Description

This function throws an error containing the given message. For backward compatibility the function may also be called using the name “Error”

## example

```

/**/ Define T(N)
/**/   If type(N) <> INT Then error("Argument must be an integer."); EndIf;
/**/   Return mod(N,5);
/**/ EndDefine;

-- /**/ T(1/3); --> !!! ERROR !!! as expected
ERROR: Argument must be an integer.
  If type(N) <> INT Then error("Argument must be an integer."); EndIf;
  ~~~~~
CONTEXT: function T (previously defined at the prompt)
called at top-level

/**/ T(7);
2

```

**See Also:** [try\(I-20.14 pg.316\)](#), [GetErrMesg\(I-7.14 pg.113\)](#)

## I-5.12 eval

## syntax

```
eval(E: RINGELEM|MODULEELEM|LIST|MAT, L: LIST): OBJECT
```

### Description

This function substitutes “L[I]” for “[indet\(I\)](#)” in the expression “E” which must be of type POLY, MODULEELEM, LIST, or MAT. The evaluation takes place in the ring of “E”.

If “[len\(L\)](#)” is different from “[NumIndets\(\)](#)” then only the first N substitutions are performed, where N is the minimum of the two values.

For more general substitutions use “[subst](#)” ([I-19.48 pg.303](#)).

## example

```

/**/ use QQ[x,y];
/**/ eval(x^2+y, [2, 3]);
7
/**/ eval(x^2+y, [2]);
y +4

/**/ F := x*(x-1)*(x-2)*y*(y-1)*(y-2)/36;
/**/ P := [1/2, -2/3];
/**/ eval(F, P);
-5/162
/**/ eval([x+y,x-y], [2,1]);
[3, 1]
/**/ eval([x+y,x-y], [x^2,y^2]);
[x^2 + y^2, x^2 - y^2]
/**/ eval([x+y,x-y], [y]);
[2*y, 0]

```

**See Also:** [Evaluation of Polynomials\(III-11.2 pg.412\)](#), [PolyAlgebraHom\(I-16.14 pg.237\)](#), [subst\(I-19.48 pg.303\)](#)

## I-5.13 EvalHilbertFn

— syntax —

```
EvalHilbertFn(H:TAGGED("$hp.Hilbert"), N: INT): INT
```

### Description

This function evaluates the Hilbert function  $H$  at  $N$ . If  $H$  is the Hilbert function of a quotient  $R/I$ , then the value returned is the same as that returned by “HilbertFn( $R/I$ ,  $N$ )” but time is saved since the Hilbert function does not need to be recalculated at each call.

— example —

```
/**/ use R ::= QQ[w,x,y,z];
/**/ I := ideal(z^2-x*y, x*z^2+w^3);
/**/ H := HilbertFn(R/I);
/**/ H;
H(0) = 1
H(1) = 4
H(t) = 6t - 3   for t >= 2

/**/ EvalHilbertFn(H,1);
4
/**/ EvalHilbertFn(H,2);
9
```

**See Also:** HilbertFn(I-8.7 pg.123), HilbertPoly(I-8.9 pg.124)

## I-5.14 EvalQuasiPoly

— syntax —

```
EvalQuasiPoly(QP: LIST of RINGELEM, N: RINGELEM): RINGELEM
```

### Description

— example —

```
/**/ M := mat(ZZ, [ [0, 2, 1], [0, -2, 3], [2, -2, 3] ]);
/**/ Cinput := record[integral_closure := M, grading := mat([[1,1,1]]) ];
/**/ C := NmzComputation(Cinput, ["HilbertSeries"]);
/**/ CHQ := C.HilbertQuasiPolynomial;   indent(CHQ);
[
  (8/9)*t^2 +2*t +1,
  (8/9)*t^2 +(14/9)*t +5/9,
  (8/9)*t^2 +(16/9)*t +8/9
]
/**/ EvalQuasiPoly(CHQ,151);
20503
```

**See Also:** NmzComputation(I-14.18 pg.216), NmzHilbertBasis(I-14.22 pg.218), NmzNormalToricRing(I-14.26 pg.220), NmzIntClosureMonIdeal(I-14.23 pg.219), NmzSetVerbosityLevel(I-14.27 pg.220)

## I-5.15 exit

— syntax —

```
exit
```

## Description

This command is used to quit CoCoA. It may be used only at top level.

**See Also:** [ciao\(I-3.14 pg.51\)](#), [quit\(I-17.3 pg.252\)](#)

## I-5.16 exponents

syntax

```
exponents(F: RINGELEM): LIST of INT
```

## Description

This function returns the list of exponents of the leading term of “F”. The inverse function is “MakeTerm” ([I-13.4 pg.192](#)).

This function was called “log” up to version CoCoA-5.1.2.

example

```
/**/ use R ::= QQ[x,y,z];
/**/ F := x^3*y^2*z^5 + x^2*y + x*z^4;
/**/ exponents(F);
[3, 2, 5]
```

**See Also:** [LPP\(I-12.20 pg.188\)](#), [LT\(I-12.21 pg.188\)](#), [MakeTerm\(I-13.4 pg.192\)](#)

## I-5.17 Ext

syntax

```
Ext(I: INT, M:TAGGED("Quotient"), Q:TAGGED("Quotient")): TAGGED("Quotient")
Ext(I: LIST, M:TAGGED("Quotient"), Q:TAGGED("Quotient")): TAGGED("$ext.ExtList")
```

## Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

In the first form the function computes the “I”-th Ext module of “M” and “N”. It returns a presentation of  $Ext_R^I(M, N)$  as a quotient of a free module.

IMPORTANT: the only exception to the type of “M” or “N” (or even of the output) is when they are either a zero module or a free module. In these cases their type is indeed MOD.

It computes Ext via a presentation of the quotient of the two modules  $Ker(Phi*_I)$  and  $Im(Phi*_{I-1})$ , where

- $Phi_I$  is the “I”-th map in the free resolution of “M”
- $Phi*_I$  is the map  $Hom(Phi_I, N)$  in the dual of the free resolution.

The main differences with the previous version include:

- SHIFTS have been removed, consequently only standard homogeneous modules and quotients are supported
- as a consequence of 1), the type “Tagged(“Shifted”)” has been removed. Ext will just be a “Tagged(“Quotient”)”
- The former functions Presentation(), HomPresentation() and KerPresentation() have been removed
- The algorithm uses Res() to compute the maps needed, and not SyzOfGens anylonger, believed to cause troubles
- The function “Ext” always has THREE variables, see syntax...

In the second form the variable  $I$  is a LIST of nonnegative integers. In this case the function `Ext` prints all the `Ext` modules corresponding to the integers in  $I$ . The output is of special type “`Tagged("$ext.ExtList")`” which is basically just the list of pairs  $(J, \text{Ext}^J(M, N)) | J \in I$  in which the first element is an integer of “ $I$ ” and the second element is the corresponding `Ext` module.

VERY IMPORTANT: CoCoA cannot accept the ring “ $R$ ” as one of the inputs, so if you want to calculate the module  $\text{Ext}_R^I(M, R)$  you need to type something like

```
“Ext(I, M, ideal(1));”
```

or

```
“Ext(I, M, R^1);”
```

or

```
“Ext(I, M, R/ideal(0));”
```

NOTE: The input is pretty flexible in terms of what you can use for “ $M$ ” and “ $N$ ”. For example, they can be zero modules or free modules. See some examples below.

#### example

```

/**/ use R := QQ[x,y,z];
/**/ I := ideal(x^5, y^3, z^2);
/**/ ideal(R, []) : (I);
ideal(0)
-----
$hom.Hom(R^1/Module(I), R^1); -- from Hom package
Module([[0]])
-----
Ext(0, R/I, R^1); --- all those things should be isomorphic
Module([[0]])
-----
Ext(0..4, R/I, R/ideal(0)); -- another way to define the ring R as a quotient
Ext^0 = Module([[0]])

Ext^1 = Module([[0]])

Ext^2 = Module([[0]])

Ext^3 = R^1/Module([[x^5], [y^3], [z^2]])

Ext^4 = Module([[0]])

-----
N := Module([x^2,y], [x+z,0]);
Ext(0..4, R/I, R^2/N);
Ext^0 = Module([[0]])

Ext^1 = Module([[0]])

Ext^2 = R^2/Module([[0, x + z], [y, 0], [0, z^2], [z^2, 0], [0, y^3], [x^5, 0]])

Ext^3 = R^2/Module([[x + z, 0], [0, z^2], [z^2, 0], [y^3, 0], [0, x^5], [0, y]])

Ext^4 = Module([[0]])

-----

```

Since version 4.7.3 the output modules are presented minimally.

**See Also:** [res\(I-18.34 pg.269\)](#), [depth\(I-4.9 pg.75\)](#), [MinimalPresentation\(I-13.20 pg.199\)](#)

## I-5.18 ExternalLibs

— syntax —

```
ExternalLibs(): LIST of STRING
```

### Description

This function returns the list of the names of the linked libraries.

— example —

```
/**/ ExternalLibs();  
["BOOST", "FROBBY", "GSL", "NORMALIZ"]
```

**See Also:** Frobby([II-9.4 pg.365](#)), Normaliz([II-9.6 pg.365](#))

# Chapter I-6

## F

### I-6.1 factor

syntax

```
factor(F: RINGELEM): RECORD
```

#### Description

This function factorizes a polynomial into irreducibles in its ring of definition. Multivariate factorization is not yet supported over finite fields (but you can use “SqFreeFactor” (I-19.30 pg.295) and then “ContentFreeFactor” (I-3.47 pg.64) to obtain a partial factorization). To factorize an integer use “SmoothFactor” (I-19.21 pg.291). (For information about the algorithm, consult John Abbott’s papers)

For univariate polynomials with coefficients in an algebraic extension use “FactorAlgExt” (I-6.2 pg.94).

NOTE: in older versions of CoCoA-5 the field names were “Factors” and “Exponents”.

example

```
/**/ use R := QQ[x,y];
/**/ F := 4*x^8 + 4*x^6 + x^4 + 4*x^2 + 4;
/**/ FacInfo := factor(F);
/**/ indent(FacInfo);
record[
  RemainingFactor := 1,
  factors := [2*x^4-4*x^3+5*x^2-4*x+2, 2*x^4+4*x^3+5*x^2+4*x+2],
  multiplicities := [1, 1]
]
/**/ G := product([FacInfo.factors[i]^FacInfo.multiplicities[i]
/**/ | i in 1..len(FacInfo.factors)]);
/**/ F = G * FacInfo.RemainingFactor;
true

/**/ factor((8*x^2 +16*x +8)/27);
record[factors := [x +1], multiplicities := [2], RemainingFactor := 8/27]

/**/ factor(2*x^2-4); -- over a finite field the factors are monic
record[factors := [x^2 -2], multiplicities := [1], RemainingFactor := 2]
-----
```

**See Also:** FactorAlgExt(I-6.2 pg.94), SmoothFactor(I-19.21 pg.291), SqFreeFactor(I-19.30 pg.295), ContentFreeFactor(I-3.47 pg.64)



## Description

This function counts how many times the integer base “b” divides a given integer “N”. NOTE: “N” must be non-zero, and “b < 2”.

example

```
/**/ type(20/2);
INT
/**/ type(20/7);
RAT
/**/ FactorMultiplicity(2, 20);
2
/**/ FactorMultiplicity(5, 20);
1
/**/ FactorMultiplicity(7, 20);
0
```

**See Also:** [IsDivisible\(I-9.45 pg.153\)](#), [SmoothFactor\(I-19.21 pg.291\)](#)

## I-6.5 FGLM5

syntax

```
FGLM5(GBOld: LIST, M: MAT): LIST
```

## Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This function is implemented in ApCoCoALib by Stefan Kaspar.

The function “FGLM5” calls the CoCoAServer to perform a FGLM Groebner Basis conversion. Please note that the ideal generated by the given Groebner Basis must be zero-dimensional. The Groebner Basis contained in list GBOld will be converted into a Groebner Basis with respect to term ordering “Ord(M)”, i.e. M must be a matrix specifying a term ordering.

example

```
use QQ[x, y, z], DegRevLex;
GBOld := *** [z^4 - 3z^3 - 4yz + 2z^2 - y + 2z - 2, yz^2 + 2yz - 2z^2
+ 1, y^2 - 2yz + z^2 - z, x + y - z] ***;
M := LexMat(3);
GBNew := FGLM5(GBOld, M);
use QQ[x, y, z], Ord(M);
-- New basis (Lex)
BringIn(GBNew);
```

## I-6.6 fibonacci

syntax

```
fibonacci(N: INT): INT
```

## Description

This function returns the “N”-th fibonacci number.

example

```
/**/ fibonacci(0);
0
```

```
/**/ fibonacci(10);
55
```

## I-6.7 fields

syntax

```
fields(R: RECORD): LIST
```

### Description

This function returns a list of all of the fields of the record “R”. It is particularly useful when you want to know if a record field has been defined

example

```
/**/ rec := record[name := "David", number := 3728852, data := ["X","Y"] ];
/**/ fields(rec);
["data", "name", "number"]

/**/ rec.data;
["X", "Y"]

/**/ "surname" IsIn fields(rec);
false
```

**See Also:** record([I-18.23](#) pg.264)

## I-6.8 first

syntax

```
first(L: LIST): OBJECT
first(L: LIST, N: INT): OBJECT
```

### Description

In the first form the function returns the first element of the list L, same as “L[1]”. In the second form, it returns the list of the first N elements of L, same as “[ L[i] | i in 1..N ]”

example

```
/**/ L := [1,2,3,4,5];
/**/ first(L);
1

/**/ first(L,3);
[1, 2, 3]
```

**See Also:** last([I-12.2](#) pg.179)

## I-6.9 FirstNonZero

syntax

```
FirstNonZero(V: MODULEELEM): RINGELEM
```

## Description

This function returns the first non-zero entry of  $V$ . If it is handed a zero MODULEELEM then an error is signalled.

example

```
/**/ use R := QQ[x,y,z];
/**/ R5 := NewFreeModule(R,5);
/**/ V := ModuleElem(R5, [0, 0, x^2+y*z, 0, z^2]);

/**/ FirstNonZero(V);
x^2 +y*z

/**/ FirstNonZeroPosn(V);
2
```

**See Also:** FirstNonZeroPosn(I-6.10 pg.97), IsZero(I-9.90 pg.170), NonZero(I-14.30 pg.221)

## I-6.10 FirstNonZeroPosn

syntax

```
FirstNonZeroPosn(V: MODULEELEM): RINGELEM
```

## Description

This function returns the index of the first non-zero entry of  $V$ . If it is handed a zero MODULEELEM then an error is signalled.

example

```
/**/ use R := QQ[x,y,z];
/**/ R5 := NewFreeModule(R,5);
/**/ V := ModuleElem(R5, [0, 0, x^2+y*z, 0, z^2]);

/**/ FirstNonZero(V);
x^2 +y*z

/**/ FirstNonZeroPosn(V);
2
```

**See Also:** FirstNonZero(I-6.9 pg.96), IsZero(I-9.90 pg.170), NonZero(I-14.30 pg.221)

## I-6.11 flatten

syntax

```
flatten(L: LIST): LIST
flatten(L: LIST, N: INT): LIST
```

## Description

Components of lists may be lists themselves, i.e., lists may be nested. With one argument this function returns the list obtained from the list “ $L$ ” by removing all nesting, bringing all elements “*to the top level*”. With the optional second argument, “ $N$ ”, nesting is removed down “ $N$ ” levels. Thus, the elements of “ $M := \text{flatten}(L, 1)$ ” are formed as follows: go through the elements of “ $L$ ” one at a time; if an elements is not a list, add it to “ $M$ ”; if an element is a list, add all of its elements to “ $M$ ”. Recursively, “ $\text{Flatten}(L, N) = \text{Flatten}(\text{Flatten}(L, N-1), 1)$ ”. For “ $N$ ” large, depending on “ $L$ ”, “ $\text{Flatten}(L, N)$ ” gives the same result as “ $\text{Flatten}(L)$ ”.

example

```

/**/ flatten([1,["a","b",[2,3,4],"c","d"],5,6]);
[1, "a", "b", 2, 3, 4, "c", "d", 5, 6]

/**/ L := [1,2, [3,4], [5, [6,7,[8,9]]]];
/**/ flatten(L,1);
[1, 2, 3, 4, 5, [6, 7, [8, 9]]]

/**/ flatten(It,1);
[1, 2, 3, 4, 5, 6, 7, [8, 9]]

/**/ flatten(L,2); -- same as flatten(flatten(L,1),1)
[1, 2, 3, 4, 5, 6, 7, [8, 9]]

/**/ flatten(L,3); -- same as flatten(L)
[1, 2, 3, 4, 5, 6, 7, 8, 9]

```

## I-6.12 FloatApprox

syntax

```
FloatApprox(X: INT|RAT|RINGELEM, PrecBits: INT): RAT
```

### Description

This function computes an approximation of the form  $M * 2^E$  to a rational number “X” where the mantissa satisfies  $2^{(PrecBits - 1)} \leq |M| < 2^{PrecBits} - 1$  where *PrecBits* is the specified bit precision. It gives 0 when applied to 0.

The updated version of this function is not backward compatible with the old one; you must replace the 2nd arg by the number of bits you want in the mantissa (see “FloorLog2, FloorLog10, FloorLogBase” (I-6.15 pg.99)). The old fn is obsolescent and is now called “FloatApprox10”.

example

```

/**/ FloatApprox(1/3, 10);
683/2048

/**/ FloatApprox(1/3, 20);
699051/2097152

/**/ FloatApprox(123456789,8);
123207680

```

**See Also:** CFAprox(I-3.8 pg.48), FloatStr(I-6.13 pg.98), MantissaAndExponent2(I-13.7 pg.194)

## I-6.13 FloatStr

syntax

```
FloatStr(X: INT|RAT|RINGELEM): STRING
FloatStr(X: INT|RAT|RINGELEM, Prec: INT): STRING
```

### Description

This function produces a decimal string representation of the rational number “X”. The optional second argument “Prec” says how many significant decimal digits to produce; the default value is 5.

The aim is to produce an easily readable result.

example

```

/**/ FloatStr(2/3);      -- last printed digit is rounded
0.66667

/**/ FloatStr(7^510);    -- no arbitrary limit on exponent range
1.0000*10^431

/**/ FloatStr(1/81, 50); -- precision of mantissa specified by user
0.012345679012345679012345679012345679012345679012346

/**/ FloatStr(987654/321);
3076.8

```

**See Also:** [DecimalStr\(I-4.3 pg.71\)](#), [ScientificStr\(I-19.3 pg.282\)](#), [FloatApprox\(I-6.12 pg.98\)](#), [MantissaAndExponent10\(I-13.6 pg.193\)](#)

## I-6.14 floor

syntax

```

floor(X: RAT): INT

```

### Description

This function returns the greatest integer less than or equal to “X”.

example

```

/**/ floor(0.99);
0

/**/ floor(1.01);
1

/**/ floor(-1);
-1

/**/ floor(-0.01);
-1

```

**See Also:** [ceil\(I-3.7 pg.48\)](#), [round\(I-18.55 pg.279\)](#), [num\(I-14.33 pg.223\)](#), [den\(I-4.7 pg.75\)](#)

## I-6.15 FloorLog2, FloorLog10, FloorLogBase

syntax

```

FloorLog2(X: RAT): INT
FloorLog10(X: RAT): INT
FloorLogBase(X: RAT, base: INT): INT

```

### Description

These functions compute the integer part (floor) of the logarithm of a rational number in a given base. “FloorLog2(X)” is just shorthand for “FloorLogBase(X,2)”; similarly for “FloorLog10”.

NOTE: “FloorLog10” may be useful for formatted printing as it gives the number of digits (minus 1) in base 10.

NOTE: The signs of “X” and “base” are ignored.

These functions were called “ILogBase” up to version CoCoA-5.1.2.

example

```
/**/ FloorLog2(128);
7
/**/ FloorLog10(999); -- number of digits minus 1
2
/**/ FloorLogBase(128,2);
7
/**/ FloorLogBase(81.5,3);
4
```

## I-6.16 FloorSqrt

syntax

```
FloorSqrt(N: INT): INT
```

### Description

This function computes the square root of an integer. If the argument is not a perfect square it returns the integer part of the square root.

This function was called “isqrt” up to version CoCoA-5.1.2.

example

```
/**/ FloorSqrt(16);
4
/**/ FloorSqrt(99);
9
-- /**/ FloorSqrt(-1); --> !!! ERROR !!! as expected
ERROR: Value must be non-negative
FloorSqrt(-1);
^^^^^^^^^^
```

## I-6.17 for

syntax

```
For I := N_1 To N_2 Do C EndFor
For I := N_1 To N_2 Step D Do C EndFor
```

where I is the loop variable, N<sub>1</sub>, N<sub>2</sub>, and D are integer expressions,  
and C is a sequence of commands.

### Description

In the first form, the loop variable “I” is assigned the values “N<sub>1</sub>, N<sub>1</sub>+1, ..., N<sub>2</sub>” in succession. After each assignment, the command sequence “C” is executed. If “N<sub>2</sub> < N<sub>1</sub>”, then the command sequence “C” is not executed.

The second form is almost the same, except that “I” is assigned the values “N<sub>1</sub>, N<sub>1</sub>+D, N<sub>1</sub>+2D”, and so on, until the limit “N<sub>2</sub>” is passed. If “N<sub>2</sub> - N<sub>1</sub>” has opposite sign to “D”, then the command sequence “C” is not executed.

## example

```

/**/ for N := 1 to 5 do print 2^N, " "; endfor;
2 4 8 16 32

/**/ for n := 1 to 20 step 3 do print n, " "; endfor;
1 4 7 10 13 16 19

/**/ for N := 10 to 1 step -2 do print N, " "; endfor;
10 8 6 4 2

/**/ for N := 5 to 3 do print N, " "; endfor; -- no output

```

Loops can be nested.

## example

```

/**/ define MySort(ref L)
/**/   for i := 1 to len(L)-1 do
/**/     MaxPos := i;
/**/     for j := i+1 to len(L) do
/**/       if L[j] < L[MaxPos] then MaxPos := j; endif;
/**/     endfor;
/**/     if MaxPos <> i then
/**/       swap(ref L[i], ref L[MaxPos]);
/**/     endif;
/**/   endfor;
/**/ enddefine;

/**/ M := [5,3,1,4,2];
/**/ MySort(ref M);
/**/ M;
[1, 2, 3, 4, 5]

```

(Note that “ref L” is used so that the function can change the value of the variable referenced by “L”. See “ref”.)

**See Also:** [foreach\(I-6.18 pg.101\)](#), [repeat\(I-18.33 pg.269\)](#), [syntax\(?? pg.??\)](#), [while\(I-23.3 pg.332\)](#)

## I-6.18 foreach

## syntax

```
foreach X in L do CMDS endforeach
where “\verb&X&” is a loop variable, “\verb&L&” is a LIST
```

### Description

The loop variable “X” is assigned the value of each component of “L” in turn. After each assignment the command sequence “CMDS” is executed.

## example

```

/**/ foreach N in 1..10 do -- NOTE: 1..10 gives the list [1,2,...,10].
/**/   print N^2, " ";
/**/ endforeach;
1 4 9 16 25 36 49 64 81 100

/**/ use P := QQ[x,y,z];
/**/ f := x^2*y + 3*y^2*z - z^3;
/**/ J := [deriv(f, x) | x in indets(P)];

```

```

/**/ J;
[2*x*y, x^2 +6*y*z, 3*y^2 -3*z^2]

/**/ foreach g in J do
/**/   println g^2;
/**/ endforeach;
4*x^2*y^2
x^4 +12*x^2*y*z +36*y^2*z^2
9*y^4 -18*y^2*z^2 +9*z^4

```

**See Also:** for(I-6.17 pg.100), repeat(I-18.33 pg.269), syntax(?? pg.??), while(I-23.3 pg.332)

## I-6.19 format

— syntax —

```
format(E: OBJECT, N: INT): STRING
```

### Description

Like Sprint, this function converts the value of E into a string. If the string has fewer than N characters, then spaces are added to the front to make the length N.

— example —

```

/**/ L := [5^n | n in 0..7];
/**/ foreach F in L do print format(F,8); endforeach;
      1      5      25      125      625      3125      15625      78125

/**/ M := format(L,20);
/**/ M; -- "format" does not truncate
[1, 5, 25, 125, 625, 3125, 15625, 78125]
/**/ type(L);
LIST
/**/ type(M);
STRING

```

**See Also:** IO.SprintTrunc(I-9.36 pg.150), LaTeX(?? pg.??), sprint(I-19.29 pg.294)

## I-6.20 FrbAlexanderDual

— syntax —

```

FrbAlexanderDual(I: IDEAL): LIST
FrbAlexanderDual(I: IDEAL, T: RINGELEM): LIST

```

### Description

Using the “Frobby” (II-9.4 pg.365) library linked with CoCoALib. Thanks to Bjarke Roune.

— example —

```

/**/ use QQ[x,y,z];
/**/ I := ideal(x^2, x*y, y^2, z^2);
/**/ FrbAlexanderDual(I);
ideal(x^2*y*z, x*y^2*z)

/**/ FrbAlexanderDual(I, x^2*y^2*z^5);
ideal(x^2*y*z^4, x*y^2*z^4)

```

**See Also:** Frobyy([II-9.4](#) pg.365), FrbPrimaryDecomposition([I-6.24](#) pg.104), PrimaryDecomposition([I-16.22](#) pg.241)

## I-6.21 FrbAssociatedPrimes

syntax

```
FrbAssociatedPrimes(I: IDEAL): LIST
```

### Description

Using the “Frobyy” ([II-9.4](#) pg.365) C++ library by Bjarke Roune.

example

```
/**/ use R ::= QQ[x,y,z];
/**/ I := ideal(x^2, x*y, y^2, z^2);
/**/ FrbAssociatedPrimes(I);
[ideal(x, y, z)]
```

**See Also:** Frobyy([II-9.4](#) pg.365), FrbIrreducibleDecomposition([I-6.22](#) pg.103), FrbPrimaryDecomposition([I-6.24](#) pg.104)

## I-6.22 FrbIrreducibleDecomposition

syntax

```
FrbIrreducibleDecomposition(I: IDEAL): LIST
```

### Description

Using the “Frobyy” ([II-9.4](#) pg.365) C++ library by Bjarke Roune.

example

```
/**/ use R ::= QQ[x,y,z];
/**/ I := ideal(x^2, x*y, y^2, z^2);
/**/ FrbIrreducibleDecomposition(I);
[ideal(x, y^2, z^2), ideal(x^2, y, z^2)];

-- *** missing manual for these function: volunteers? ;- ) ***
FrbDimension
FrbMultigradedHilbertPoincareNumerator
FrbTotalDegreeHilbertPoincareNumerator
```

**See Also:** Frobyy([II-9.4](#) pg.365), FrbAssociatedPrimes([I-6.21](#) pg.103), FrbIrreducibleDecomposition([I-6.22](#) pg.103)

## I-6.23 FrbMaximalStandardMonomials

syntax

```
FrbMaximalStandardMonomials(I: IDEAL): LIST
```

### Description

Using the “Frobyy” ([II-9.4](#) pg.365) library linked with CoCoALib.

example

```

/**/ use QQ[x,y,z];
/**/ I := ideal(x^2, x*y, y^2, z^2);
/**/ FrbMaximalStandardMonomials(I);
ideal(y*z, x*z)

```

**See Also:** [Frobby](#)([II-9.4](#) pg.365)

## I-6.24 FrbPrimaryDecomposition

syntax

```

FrbPrimaryDecomposition(I: IDEAL): LIST

```

### Description

Using the “Frobby” ([II-9.4](#) pg.365) C++ library by Bjarke Roune.

example

```

/**/ use R ::= QQ[x,y,z];
/**/ I := ideal(x^2, x*y^2, z^2);
/**/ FrbPrimaryDecomposition(I);
[ideal(x^2, y^2, z^2), ideal(x, z^2)]

```

**See Also:** [Frobby](#)([II-9.4](#) pg.365), [FrbAssociatedPrimes](#)([I-6.21](#) pg.103), [FrbIrreducibleDecomposition](#)([I-6.22](#) pg.103), [PrimaryDecomposition](#)([I-16.22](#) pg.241)

## I-6.25 FrobeniusMat

syntax

```

FrobeniusMat(I: IDEAL): MAT
FrobeniusMat(I: IDEAL, QB: LIST): MAT

```

### Description

This function compute a matrix of the Frobenius Map (“ $f \mapsto f^q$ ”).

example

```

/**/ use P ::= ZZ/(5)[x,y,z], Lex;
/**/ I := ideal(y^2-x*z, z^2-x^2*y, x+y+z-1);

/**/ FrobeniusMat(I);
matrix( /*RingWithID(3, "FFp(5)")*/
  [[1, 0, 0, 0, 0, 0],
   [0, -2, 2, -2, -1, 0],
   [0, 1, 2, -1, 2, 0],
   [0, 2, 2, -1, -1, 0],
   [0, 1, 1, -1, -2, 1],
   [0, 0, 0, 0, 0, 0]])

/**/ FrobeniusMat(I, [z^4, z^3, z^2, z, y, 1]);
matrix( /*RingWithID(3, "FFp(5)")*/
  [[-2, -1, 1, 1, 1, 0],
   [-1, -1, 2, 2, 0, 0],
   [2, -1, 2, 1, 0, 0],
   [-1, -2, 2, -2, 0, 0],

```

```
[0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 1]])
```

**See Also:** [IsIn\(I-9.53 pg.157\)](#), [IsSubset\(I-9.83 pg.167\)](#)

## I-6.26 *func*

syntax

```
Func ... EndFunc
  returns FUNCTION
```

### Description

This syntactic structure defines an anonymous function, i.e. a function without a name. Anonymous functions can be passed as parameters and assigned to variables. Note that “`Func...EndFunc`” can also be used inside function definitions.

example

```
/**/ square := Func(x) Return x^2; EndFunc;
/**/ square(3);
9

/**/ SortedBy(["zzz", "x", "yy"], Func(x,y) Return len(x)>len(y); EndFunc);
["zzz", "yy", "x"]
```

**See Also:** [define\(I-4.4 pg.72\)](#), [TopLevel\(I-20.10 pg.314\)](#), [ImportByRef](#), [ImportByValue\(I-9.17 pg.141\)](#)

## I-6.27 *Function* [OBSOLETE]

syntax

```
[OBSOLETE]
```

### Description

[OBSOLETE] In CoCoA-5 functions are “first class objects”, and so may be passed like any other value – the operator “`Function`” serves no purpose.

In CoCoA-4 it was possible to have a variable and a function with the same name; the operator “`Function`” was used to instruct CoCoA-4 to search for the function of the given name, e.g. to pass it as an argument to another function.

**See Also:** [FUNCTIONs are first class objects\(III-7.2 pg.399\)](#), [SortBy\(I-19.23 pg.292\)](#), [SortedBy\(I-19.25 pg.293\)](#)

## I-6.28 *functions* [OBSOLETE]

syntax

```
[OBSOLETE]
```

### Description

[OBSOLETE] please use the command “`describe`” ([I-4.12 pg.77](#)).

## I-6.29 FVector

— syntax —

`FVector(A: LIST): RECORD`

### Description

This function computes the f-Vector of a simplicial complex described by a list of top faces.

Package “GeomModelling”, by Elisa Palezzato.

— example —

```
/**/ use QQ[x[1..5]], DegLex;
/**/ L := [x[1]*x[2]*x[3], x[2]*x[3]*x[4], x[3]*x[4]*x[5]]; -- list top faces
/**/ FVector(L);
[1, 5, 7, 3]
```

**See Also:** SimplexInfo([I-19.16](#) pg.288)

# Chapter I-7

## G

### I-7.1 GBasis

syntax

```
GBasis(I: IDEAL|MODULE): LIST
```

#### Description

This function returns a list whose components form a Groebner basis for the ideal (or module) “I” with respect to the term-ordering of the polynomial ring of “I”.

If “I” is a variable then the result is stored in “I” for later use.

For the reduced Groebner basis, use the command “ReducedGBasis” ([I-18.25](#) pg.265).

The coefficient ring must be a field.

example

```
/**/ use R ::= QQ[x,y];
/**/ I := ideal(x^4-x^2, x^3-y);
/**/ GBasis(I);
[-x^2 +x*y, -x*y +y^2, y^3 -y]
```

**See Also:** GBasisTimeout([I-7.2](#) pg.107)

### I-7.2 GBasisTimeout

syntax

```
GBasisTimeout(I: IDEAL, TimeLimit: INT): LIST
GBasisTimeout(M: MODULE, TimeLimit: INT): LIST
```

#### Description

Same as “GBasis” ([I-7.1](#) pg.107), but it will stop with an error if the computation is not completed within the specified time limit (in seconds).

ONLY PARTIALLY IMPLEMENTED: only for ideals. It is not yet possible to resume the computation; you must restart it from the beginning.

For dealing with errors see “try” ([I-20.14](#) pg.316).

example

```
/**/ use R ::= QQ[t,x,y,z];
/**/ I := ideal(t^3-x, t^4-y, t^5-z);
/**/ J := I^5;
```

```
-- /**/ G := GBasisTimeout(J, 0.1); --> !!! ERROR !!! as expected
ERROR: Computation exceeded given time limit
/**/ J := I^5; G := GBasisTimeout(J, 10); --> OK
```

**See Also:** [GBasis\(I-7.1 pg.107\)](#), [try\(I-20.14 pg.316\)](#)

## I-7.3 GBM

— syntax —

```
GBM(L: LIST): IDEAL
```

### Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This function computes the intersection of ideals corresponding to zero-dimensional schemes: GBM is for affine schemes, and “HGBM” ([I-8.4 pg.122](#)) for projective schemes. The list L must be a list of ideals. The function “IntersectList” ([?? pg.??](#)) should be used for computing the intersection of a collection of general ideals.

The name GBM comes from the name of the algorithm used: Generalized Buchberger-Moeller.

— example —

```
/**/ use P := QQ[x,y,z];
/**/ I1 := IdealOfPoints(P, mat([[1,2,1], [0,1,0]])); -- a simple affine scheme
/**/ I2 := IdealOfPoints(P, mat([[1,1,1], [2,0,1]]))^2;-- another affine scheme

***** NOT YET IMPLEMENTED *****
      GBM([I1, I2]);                                -- intersect the ideals
ideal(xz + yz - z^2 - x - y + 1,
      z^3 - 2z^2 + z,
      yz^2 - 2yz - z^2 + y + 2z - 1,
      y^2z - y^2 - yz + y,
      xy^2 + y^3 - 2x^2 - 5xy - 5y^2 + 2z^2 + 8x + 10y - 4z - 6,
      x^2y - y^3 + 2x^2 + 2xy + 4y^2 - 3z^2 - 8x - 8y + 6z + 5,
      x^3 + y^3 - 7x^2 - 5xy - 4y^2 + 5z^2 + 16x + 10y - 10z - 7,
      y^4 - 2y^3 - 4x^2 - 8xy - 3y^2 + 4z^2 + 16x + 16y - 8z - 12)
-----
```

**See Also:** [IdealAndSeparatorsOfPoints\(I-9.3 pg.132\)](#), [IdealAndSeparatorsOfProjectivePoints\(I-9.4 pg.133\)](#), [IdealOfPoints\(I-9.7 pg.135\)](#), [IdealOfProjectivePoints\(I-9.8 pg.136\)](#), [HGBM\(I-8.4 pg.122\)](#)

## I-7.4 gcd

— syntax —

```
gcd(F: INT,G: INT): INT
gcd(L: LIST of INT): INT

gcd(F: RINGELEM, G: RINGELEM): RINGELEM
gcd(L: LIST of RINGELEM): RINGELEM
```

### Description

This function returns the greatest common divisor of “F<sub>1</sub>, . . . , F<sub>n</sub>” or of the elements in the list L. For the calculation of the GCDs and LCMs of polynomials, the coefficient ring must be a field.

example

```

/**/ use R := QQ[x,y];
/**/ F := x^2-y^2;
/**/ G := (x+y)^3;
/**/ gcd(F, G);
-x -y

/**/ gcd([3*4,3*8,6*16]);
12

```

**See Also:** [div\(I-4.20 pg.81\)](#), [mod\(I-13.26 pg.202\)](#), [lcm\(I-12.5 pg.181\)](#), [IsCoprime\(I-9.42 pg.152\)](#)

## I-7.5 GCDFreeBasis

syntax

```
GCDFreeBasis(L: LIST of INT): LIST of INT
```

### Description

This function returns a GCD free basis of a set of integers; you can think of this as the set of all numbers (except 1) obtainable by performing GCD and exact division operations.

Given a set  $N = [N_1, \dots, N_k]$  we seek a basis  $G = [G_1, \dots, G_s]$  such that each  $N_i$  is a product of powers of the  $G_j$ , and the  $G_j$  are pairwise coprime; the set  $G$  is called a GCD free basis for  $N$ . In general the set  $G$  is not uniquely defined.

example

```

/**/ GCDFreeBasis([factorial(20), factorial(10)]);
[46189, 4, 14175]

```

**See Also:** [gcd\(I-7.4 pg.108\)](#)

## I-7.6 GenericPoints

syntax

```

GenericPoints(R: RING, NumPoints: INT): LIST
GenericPoints(R: RING, NumPoints: INT, RandomRange: INT): LIST

```

### Description

“GenericPoints” returns a list of NumPoints generic projective points with integer coordinates; it is not guaranteed that these points are distinct. RandomRange specifies the largest value any coordinate may take. If the second argument is omitted, the largest value possible is 100 (or P-1 where P is the characteristic of the coefficient ring).

example

```

/**/ use R := QQ[x,y]; GenericPoints(R,7);
[[1, 0], [0, 1], [1, 1], [12, 59], [6, 63], [12, 80], [17, 63]]

/**/ GenericPoints(R,7,500);
[[1, 0], [0, 1], [1, 1], [220, 162], [206, 452], [98, 106], [403, 449]]

/**/ use R := ZZ/(5)[x,y,z];
/**/ GenericPoints(R,7);
[[1, 0, 0], [0, 1, 0], [0, 0, 1], [1, 1, 1], [2, 1, 1], [2, 2, 4], [3, 1, 3]]

```

```

/**/ GenericPoints(R,7,500);
[[1, 0, 0], [0, 1, 0], [0, 0, 1], [1, 1, 1], [1, 4, 2], [1, 3, 2], [2, 3, 3]]

```

## I-7.7 GenRepr

syntax

```

GenRepr(X: RINGELEM, I: IDEAL): LIST of RINGELEM
GenRepr(X: MODULEELEM, I: MODULE): LIST of RINGELEM

```

### Description

This function returns a list giving a representation of “X” in terms of “gens(I)”: if “X” is in “I”, then “GenRepr” returns a list “[F\_1, ..., F\_t]” such that (“[G\_1, ..., G\_t] = gens(I)”)

$$X = F_1 \cdot G_1 + \dots + F_t \cdot G_t.$$

If X is not in I, then “GenRepr” returns the empty list, [].

NOTE: for a representation in terms of “GBasis(I)” call “DivAlg(X, GBasis(I))” instead.

example

```

/**/ use R := QQ[x,y];
/**/ I := ideal(x*y -2, x^2 -x*y);
/**/ GenRepr(x -y, I);
[(-1/2)*x +(1/2)*y, (1/2)*y]
/**/ ScalarProduct(It, gens(I));
x -y
/**/ ReducedGBasisRepr(x -y, I);
[1, 0]
/**/ ScalarProduct(It, ReducedGBasis(I));
x -y

/**/ K := NewFractionField(NewPolyRing(QQ, "a"));
/**/ use R := K[x,y];
/**/ L := [x+y^2, x^2-x*y];
/**/ GenRepr((a-2)*L[1] - (x-a)*L[2], ideal(L));
[a -2, -x +a]

/**/ R3 := NewFreeModule(R,3);
/**/ V1 := ModuleElem(R3, [x, y, y^2]);
/**/ V2 := ModuleElem(R3, [x-y, 0, x^2]);
/**/ V := x^2*V1 - y^2*V2;
/**/ M := submodule(R3, [V1, V2]);
--/**/ GenRepr(V, M); -- ***** NOT YET IMPLEMENTED *****
--[x^2, -y^2]

```

**See Also:** DivAlg(I-4.21 pg.81), IsIn(I-9.53 pg.157), NF(I-14.16 pg.215), syz(I-19.57 pg.307), SyzOfGens(I-19.58 pg.308)

## I-7.8 gens

syntax

```

gens(I: IDEAL): LIST
gens(M: MODULE): LIST

```

## Description

This function returns a list of polynomials which generate the ideal I or the module M. The list is not necessarily minimal.

example

```

/**/ use R ::= QQ[x,y,z];
/**/ I := ideal(y^2-x^3, x*y);
/**/ gens(I);
[-x^3 +y^2, x*y]

/**/ gens(I^2);
[x^6 -2x^3*y^2 +y^4, -x^4*y +x*y^3, x^2*y^2]

/**/ R3 := NewFreeModule(R, 3);
/**/ e := gens(R3); // canonical basis
/**/ e[2];
[0, 1, 0]

/**/ M := SubmoduleRows(R3, mat([[x,y,z], [x-1,0,z]]));
/**/ gens(M);
[[x, y, z], [x -1, 0, z]]
/**/ shape(It);
[MODULEELEM, MODULEELEM]
/**/ GensAsRows(M);
matrix( /*RingDistrMPolyClean(QQ, 3)*/
  [[x, y, z],
   [x -1, 0, z]])

```

**See Also:** GensAsCols, GensAsRows(I-7.9 pg.111), IdealOfMinGens(I-9.6 pg.135), SubmoduleOfMinGens(I-19.46 pg.302)

## I-7.9 GensAsCols, GensAsRows

syntax

```

GensAsRows(M: MODULE): MAT
GensAsCols(M: MODULE): MAT

```

## Description

These functions returns a matrix which generate the module M with the components as row (or columns) of a matrix.

The generators are not necessarily minimal.

example

```

/**/ use R ::= QQ[x,y,z];
/**/ R3 := NewFreeModule(R, 3);
/**/ L := [[x,y,z], [x-1,0,z]];
/**/ M := SubmoduleRows(R3, mat(L));
/**/ gens(M);
[[x, y, z], [x -1, 0, z]]
/**/ shape(It);
[MODULEELEM, MODULEELEM]

/**/ GensAsRows(M);
matrix( /*RingDistrMPolyClean(QQ, 3)*/
  [[x, y, z],

```

```

    [x -1, 0, z]])

/**/ GensAsCols(M);
matrix( /*RingDistrMPolyClean(QQ, 3)*/
    [[x, x -1],
     [y, 0],
     [z, z]])

```

**See Also:** gens(I-7.8 pg.110), SubmoduleCols, SubmoduleRows(I-19.45 pg.302)

## I-7.10 Get

— syntax —

```
Get(D: DEVICE, N: INT): LIST of INT
```

### Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This function reads N characters from D and returns the list of their ASCII codes.

— example —

```

D := OpenIFile("io.cpkg"); -- open the file "io.cpkg"
Get(D,10); -- get the first 10 characters
[45, 45, 32, 105, 111, 100, 101, 118, 46, 112]
-----
ascii(It); convert the ASCII code to characters
-- iodev.p
-----
ascii(Get(D,10)); -- get the next 10 characters and convert
kg : 0.1 :
-----
Close(D);

```

The instruction “Get(DEV.STDIN,3)”, for instance, will read 3 characters typed in by the user. Clever use of this function can be used to prompt a user for input to a function, although it is usually easier for functions to take input directly as arguments. NOTE: this function does not work properly under the GUI Interface.

**See Also:** Introduction to IO(II-7.1 pg.355), OpenIFile(I-15.2 pg.227), OpenOFile(I-15.5 pg.229), OpenIString(I-15.3 pg.228), OpenOString(I-15.6 pg.230)

## I-7.11 GetCol

— syntax —

```
GetCol(M: MAT, K: INT): LIST
```

### Description

This function makes a list containing the entries of the “K”-th column of “M”.

— example —

```

/**/ M := mat([[1,2], [3,4]]);
/**/ GetCol(M,2);
[2, 4]

```

**See Also:** GetRow(I-7.15 pg.114), GetCols(I-7.12 pg.113)

## I-7.12 GetCols

syntax

```
GetCols(M: MAT): LIST of LIST
```

### Description

This function produces a list of lists containing the columns of “M”.

example

```
/**/ M := mat([[1,2], [3,4]]);
/**/ GetCols(M);
[[1, 3], [2, 4]]
```

**See Also:** GetCol([I-7.11](#) pg.112), GetRows([I-7.16](#) pg.114)

## I-7.13 GetEnv

syntax

```
GetEnv(S: STRING): STRING
```

### Description

This function returns the value of system shell variables

example

```
/**/ GetEnv("HOME");
/Users/bigatti

/**/ GetEnv("COCOARC");
/Users/bigatti/.cocoarc

/**/ GetEnv("COCOA_PACKAGES");
/Applications/CoCoA-4.7/packages
```

## I-7.14 GetErrMsg

syntax

```
GetErrMsg(E: ERROR): STRING
```

### Description

This function returns the string containing the error message associated with an error.

example

```
/**/ ErrMsg := "";

Try
  F := 1/0;
UponError E Do
  ErrMsg := GetErrMsg(E);
EndTry; -- no error is thrown with Try .. UponError .. EndTry

/**/ ErrMsg;
Division by zero or by a zero-divisor
```

**See Also:** [try\(I-20.14 pg.316\)](#), [error\(I-5.11 pg.87\)](#)

## I-7.15 GetRow

syntax

```
GetRow(M: MAT, K: INT): LIST
```

### Description

This function makes a list containing the entries of the “K”-th row of “M”.

example

```
/**/ M := mat([[1,2], [3,4]]);
/**/ GetRow(M,2);
[3, 4]
```

**See Also:** [GetRows\(I-7.16 pg.114\)](#), [SetRow\(I-19.9 pg.285\)](#)

## I-7.16 GetRows

syntax

```
GetRows(M: MAT): LIST of LIST
```

### Description

This function produces a list of lists containing the rows of “M”.

example

```
/**/ M := mat([[1,2], [3,4]]);
/**/ GetRows(M);
[[1, 2], [3, 4]]
```

**See Also:** [GetRow\(I-7.15 pg.114\)](#)

## I-7.17 GFanContainsPositiveVector

syntax

```
GFanContainsPositiveVector(EqMat: MAT, IneqMat: MAT): INT
```

### Description

...to do..

## I-7.18 GFanGeneratorsOfLinealitySpace

syntax

```
GFanGeneratorsOfLinealitySpace(EqMat: MAT, IneqMat: MAT): MAT
```

### Description

...to do..

## I-7.19 GFanGeneratorsOfSpan

syntax

```
GFanGeneratorsOfSpan(EqMat: MAT, IneqMat: MAT): MAT
```

### Description

...to do..

## I-7.20 GFanGetAmbientDimension

syntax

```
GFanGetAmbientDimension(EqMat: MAT, IneqMat: MAT): INT
```

### Description

...to do..

## I-7.21 GFanGetCodimension

syntax

```
GFanGetCodimension(EqMat: MAT, IneqMat: MAT): INT
```

### Description

...to do..

## I-7.22 GFanGetDimension

syntax

```
GFanGetDimension(EqMat: MAT, IneqMat: MAT): INT
```

### Description

...to do..

## I-7.23 GFanGetDimensionOfLinealitySpace

syntax

```
GFanGetDimensionOfLinealitySpace(EqMat: MAT, IneqMat: MAT): INT
```

### Description

...to do..

## I-7.24 GFanGetFacets

syntax

```
GFan(EqMat: MAT, IneqMat: MAT): MAT
```

## Description

...to do..

### I-7.25 GFanGetImpliedEquations

syntax

```
GFanGetImpliedEquations(EqMat: MAT, IneqMat: MAT): MAT
```

## Description

...to do..

### I-7.26 GFanGetUniquePoint

syntax

```
GFanGetUniquePoint(EqMat: MAT, IneqMat: MAT): MAT
```

## Description

...to do..

### I-7.27 GFanRelativeInteriorPoint

syntax

```
GFanRelativeInteriorPoint(EqMat: MAT, IneqMat: MAT): MAT
```

## Description

These function returns a matrix ...

example

```
/**/ GFanRelativeInteriorPoint(matrix([[1,2,3]]), matrix([[1,0,2],[2,-1,-1]]));
matrix(ZZ,
  [[2],
   [5],
   [-1]])
```

### I-7.28 gin

syntax

```
gin(I: IDEAL): IDEAL
```

## Description

These functions return the [probabilistic] gin (generic initial ideal) of the ideal “I”. It is obtained by computing twice the leading term ideal of  $g(I)$ , where  $g$  is a random change of coordinates with integer coefficients in the range  $[-10^6, 10^6]$  using TwinFloats (see “NewRingTwinFloat” (I-14.11 pg.213)) to allow a much wider range of coefficients than a direct computation over the rationals (use second argument to see the TwinFloat precision needed).

See “rgin” (I-18.42 pg.273) for computing wrt DegRevLex independently of the current ring.

Verbosity:

with verbosity “ $\geq 50$ ” it prints

the two random changes of coordinates used

and the “NewRingTwinFloat” ([I-14.11](#) pg.213) precision used.

example

```

/**/ use R ::= QQ[x,y,z];
/**/ gin(ideal(y^2-x*z, x^2*z-y*z^2)); -- computed twice using TwinFloats
ideal(x^2, x*y^2, y^4)

/**/ SetVerbosityLevel(50); --> get some internal progress information
/**/ gin(ideal(y^7-x^4*z^3, x^5*z-y*z^5));
RandIdeal: g = [
  -439946*x,
  742383*x -909613*y,
  129429*x +49607*y +832207*z
]
TryPrecisions: -- trying with FloatPrecision 64
RandIdeal: g = [
  -187890*x,
  -127769*x +272107*y,
  377492*x +778394*y -547019*z
]
TryPrecisions: -- trying with FloatPrecision 64
ideal(x^6, x^5*y^2, x^4*y^4, x^3*y^6, x^2*y^8, x*y^10, y^12)

```

**See Also:** NewRingTwinFloat([I-14.11](#) pg.213), rgin([I-18.42](#) pg.273)

## I-7.29 GradingDim

syntax

```
GradingDim(P): INT
```

### Description

This function returns the grading dimension of a polynomial ring, i.e. how many of the rows of the order matrix of “P” are to be taken as specifying the grading.

example

```

/**/ OrdM := MakeTermOrd(RowMat([2,3]));
/**/ P := NewPolyRing(QQ, "x,y", OrdM, 1);
/**/ GradingDim(P);
1

```

**See Also:** NewPolyRing([I-14.8](#) pg.212), GradingMat([I-7.30](#) pg.117)

## I-7.30 GradingMat

syntax

```
GradingMat(R: RING): MAT
```

### Description

This function returns the grading matrix (or weights matrix) for the polynomials ring “R”.

example

```

/**/ OrdM := MakeTermOrd(RowMat([2,3])); OrdM;
matrix(ZZ,
  [[2, 3],
   [0, -1]])
/**/ P := NewPolyRing(QQ, "x,y", OrdM, 1); -- GradingDim = 1
/**/ GradingMat(P);
matrix(ZZ,
  [[2, 3]])

/**/ use P;
/**/ deg(x*y);
2
/**/ wdeg(x*y);
[5]

```

**See Also:** [deg\(I-4.6 pg.74\)](#), [wdeg\(I-23.1 pg.331\)](#), [GradingDim\(I-7.29 pg.117\)](#), [NewPolyRing\(I-14.8 pg.212\)](#)

### I-7.31 graeffe

syntax

```

graeffe(R RINGELEM): RINGELEM
graeffe3(R RINGELEM): RINGELEM

```

#### Description

The function “**graeffe**” applies the graeffe transformation to the univariate polynomial “**F**”. The result is a univariate polynomial whose roots are the squares of the roots of “**F**”.

The similar function “**graeffe3**” produces the polynomial whose roots are the cubes of the roots of “**F**”

example

```

/**/ use P ::= QQ[x];
/**/ f := x^3-3*x^2+5*x-7;
/**/ graeffe(f);
x^3 +x^2 -17*x -49
/**/ graeffe3(f);
x^3 -3*x^2 -43*x -343

```

**See Also:** [RootBound\(I-18.54 pg.278\)](#)

### I-7.32 GraverBasis

syntax

```

GraverBasis(M: MAT): LIST of RINGELEM

```

#### Description

These function return the Graver basis, computed with “**toric**” ([I-20.12 pg.315](#)).

example

```

/**/ use P ::= ZZ/(2)[x,y,z];
/**/ toric(P, mat([[1,3,2]]));
ideal(-x^2 +z, x^3 -y)
/**/ GraverBasis(P, mat([[1,3,2]]));

```

```
[x^2 -z, x*z -y, x^3 -y, x*y -z^2, z^3 -y^2]
/**/ UniversalGBasis(toric(P, mat([[1,3,2]])));
[x*z -y, x*y -z^2, x^2 -z, z^3 -y^2, x^3 -y]
```

**See Also:** [toric\(I-20.12 pg.315\)](#), [HilbertBasisKer\(I-8.6 pg.123\)](#)

## I-7.33 GroebnerFanIdeals

syntax

```
GroebnerFanIdeals(I: IDEAL): LIST of IDEAL
```

### Description

Returns a LIST of ideals, one for each possible distinct reduced Groebner basis of “I”; these ideals are generated by their GBasis, and each is in a different ring; their associated term orderings lead to all the different possible reduced Groebner bases.

See “[CallOnGroebnerFanIdeals](#)” ([I-3.2 pg.45](#)) for a way of computing with all the various ideal (but without storing the whole list).

Verbosity:

“\*” with verbosity “>=10” (recursive, [CallOnRecursive](#))

“.” with verbosity “>=20” ([GetFlippableInequalities](#))

“maxdeg” with verbosity “>=80” ([GetFlippableInequalities](#))

timings with verbosity “>=90” ([GroebnerFanIdeals](#), [CallOnGroebnerFanIdeals](#))

This function used to be called “[AllReducedGroebnerBases](#)” up to version CoCoA-5.1.4, and used to return the ideals encoded with the same set of generators as “I” (now generated by [GBasis](#)).

example

```
/**/ use R ::= QQ[a,b,c];
/**/ I := ideal(b^3+c^2-1, b^2+a^2+c-1, a^2+b^3-1);
/**/ GF := GroebnerFanIdeals(I);
/**/ [ len(GBasis(I)) | I in GF];
[4, 4, 6, 6, 5, 6, 4, 4, 4, 3, 4, 3, 3, 3, 4, 3, 3]

-- The ideal in [Sturmfels, Example 3.9] has 360 marked reduced Groebner bases
/**/ use R ::= QQ[a,b,c];
/**/ I := ideal(a^5+b^3+c^2-1, b^2+a^2+c-1, c^3+a^6+b^5-1);
/**/ GF := GroebnerFanIdeals(I);
/**/ len(GF);
360
```

**See Also:** [OrdMat\(I-15.10 pg.231\)](#), [RingOf\(I-18.46 pg.275\)](#), [UniversalGBasis\(I-21.2 pg.325\)](#)

## I-7.34 GroebnerFanReducedGBases

syntax

```
GroebnerFanReducedGBases(I: IDEAL): LIST of IDEAL
```

## Description

This function returns all reduced GBases. Good only for visualizing very small examples. For further computations use “GroebnerFanIdeals” ([I-7.33](#) pg.119) or, even better, “CallOnGroebnerFanIdeals” ([I-3.2](#) pg.45).

— example —

```

/**/ use R := QQ[a,b,c];
/**/ I := ideal(b^2-1, a^2+c-1, c^2-b);
/**/ indent(GroebnerFanReducedGBases(I));
[
  [c^2 -b, b^2 -1, a^2 +c -1],
  [b -c^2, a^2 +c -1, c^4 -1],
  [c +a^2 -1, b -a^4 +2*a^2 -1, a^8 -4*a^6 +6*a^4 -4*a^2],
  [c +a^2 -1, b^2 -1, a^4 -b -2*a^2 +1]
]
```

**See Also:** CallOnGroebnerFanIdeals([I-3.2](#) pg.45), GroebnerFanIdeals([I-7.33](#) pg.119)

# Chapter I-8

## H

### I-8.1 HasGBasis

syntax

```
HasGBasis(I: IDEAL): BOOL
```

#### Description

After the “GBasis” (I-7.1 pg.107) of “I” is (explicitely or implicitly) computed, it is stored within “I” for future use. This function says whether the GBasis of “I” has been stored.

example

```
/**/ use R ::= QQ[x,y,z];
/**/ I := ideal(x^20 -x*y -1, x^10*y^10 -x*z -1, x^10*z^10 -x*z -1);
/**/ HasGBasis(I);
false
/**/ t0 := CpuTime(); GB := GBasis(I); TimeFrom(t0);
0.948
/**/ HasGBasis(I);
true
/**/ t0 := CpuTime(); GB := GBasis(I); TimeFrom(t0);
0.007
```

**See Also:** IdealOfGBasis(I-9.5 pg.134)

### I-8.2 HColon

syntax

```
HColon(M: IDEAL, N: IDEAL): IDEAL
```

#### Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

The function “colon” (I-3.30 pg.57) returns the quotient of M by N: the ideal of all polynomials F such that  $F \cdot G$  is in M for all G in N.

This function computes the same ideal using a Hilbert-driven algorithm. It differs from “colon” (I-3.30 pg.57) only when the input is non-homogeneous, in which case, “HColon” may be faster.

example

```
use R ::= QQ[x,y];
ideal(xy, x^2) : ideal(x);
```

```
ideal(y, x)
-----
colon(ideal(x^2, xy), ideal(x, x-y^2));
ideal(x)
-----
HColon(ideal(x^2, xy), ideal(x, x-y^2));
ideal(x)
-----
```

**See Also:** [HSaturation\(I-8.15 pg.128\)](#), [saturate\(I-19.1 pg.281\)](#), [HColon\(I-8.2 pg.121\)](#), [colon\(I-3.30 pg.57\)](#)

## I-8.3 HermitePoly

syntax

```
HermitePoly(N: INT, X: RINGELEM): RINGELEM
HermitePoly2(N: INT, X: RINGELEM): RINGELEM
```

### Description

The function “HermitePoly” returns the “N”-th Hermite polynomial (as used in physics); the function “HermitePoly2” returns the “N”-th Hermite polynomial (as used in probability).

These functions also work if “X” is not an indeterminate: the result is then the evaluation of the polynomial at the given value.

example

```
/**/ use R := QQ[x];
/**/ HermitePoly(3,x);
8*x^3 -12*x
```

**See Also:** [ChebyshevPoly\(I-3.12 pg.50\)](#), [LaguerrePoly\(I-12.1 pg.179\)](#)

## I-8.4 HGBM

syntax

```
HGBM(L: LIST): IDEAL
```

### Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This function computes the intersection of ideals corresponding to zero-dimensional schemes: “GBM” ([I-7.3 pg.108](#)) is for affine schemes, and HGBM for projective schemes. The list L must be a list of ideals. The function “IntersectList” ([?? pg.??](#)) should be used for computing the intersection of a collection of general ideals.

The name GBM comes from the name of the algorithm used: Generalized Buchberger-Moeller. The prefix H comes from Homogeneous since ideals of projective schemes are necessarily homogeneous.

example

```
use P := QQ[x[0..2]];
I1 := IdealOfProjectivePoints(P, [[1,2,1], [0,1,0]]); -- simple projective scheme
I2 := IdealOfProjectivePoints(P, [[1,1,1], [2,0,1]]^2; -- another projective scheme
HGBM([I1, I2]); -- intersect the ideals
ideal(x[0]^3 - x[0]x[1]^2 - 5x[0]^2x[2] + x[1]^2x[2] + 8x[0]x[2]^2 - 4x[2]^3,
x[0]^2x[1] + x[0]x[1]^2 - 3x[0]x[1]x[2] - x[1]^2x[2] + 2x[1]x[2]^2,
x[0]x[1]^3 - 2x[0]^2x[2]^2 - 5x[0]x[1]x[2]^2 - 4x[1]^2x[2]^2 +
```

```

8x[0]x[2]^3 + 10x[1]x[2]^3 - 8x[2]^4,
x[0]x[1]^2x[2] + x[1]^3x[2] - 2x[0]^2x[2]^2 - 5x[0]x[1]x[2]^2
- 5x[1]^2x[2]^2 + 8x[0]x[2]^3 + 10x[1]x[2]^3 - 8x[2]^4,
x[1]^4x[2] - 2x[1]^3x[2]^2 - 4x[0]^2x[2]^3 - 8x[0]x[1]x[2]^3
- 3x[1]^2x[2]^3 + 16x[0]x[2]^4 + 16x[1]x[2]^4 - 16x[2]^5)
-----

```

**See Also:** [IdealAndSeparatorsOfPoints\(I-9.3 pg.132\)](#), [IdealAndSeparatorsOfProjectivePoints\(I-9.4 pg.133\)](#), [IdealOfPoints\(I-9.7 pg.135\)](#), [IdealOfProjectivePoints\(I-9.8 pg.136\)](#), [GBM\(I-7.3 pg.108\)](#)

## I-8.5 hilbert [OBSOLESCENT]

— syntax —

```
[OBSOLESCENT]
```

### Description

Renamed as “HilbertFn” ([I-8.7 pg.123](#)).

## I-8.6 HilbertBasisKer

— syntax —

```
HilbertBasisKer(M: MAT): LIST
```

### Description

This function returns a list whose components are lists (of non-negative integers) representing the Hilbert basis for the monoid of elements with non-negative coordinates in the kernel of “M”, matrix over “ZZ”.

— example —

```

/**/ M := mat([[1,-2,3,4], [1, 0, 0, -1]]);
/**/ HilbertBasisKer(M);
[[0, 3, 2, 0], [1, 4, 1, 1], [2, 5, 0, 2]]

/**/ M * transposed(mat(It));
matrix(QQ,
  [[0, 0, 0],
   [0, 0, 0]])

```

**See Also:** [LinKerBasis\(I-12.12 pg.184\)](#)

## I-8.7 HilbertFn

— syntax —

```

HilbertFn(R: RING|IDEAL): TAGGED("$hp.Hilbert")
HilbertFn(R: RING|IDEAL, N: INT): INT

```

### Description

The first form of this function computes the Hilbert function for R. The second form computes the N-th value of the Hilbert function. The weights of the indeterminates of R must all be 1. If the input is not homogeneous,

the Hilbert function of the corresponding leading term (initial) ideal or module is calculated. For repeated evaluations of the Hilbert function, use “EvalHilbertFn” (I-5.13 pg.89) instead of “HilbertFn(R, N)” in order to speed up execution.

The coefficient ring must be a field.

example

```

/**/ use R := QQ[t,x,y,z];
/**/ HilbertFn(R/ideal(z^2-x*y, x*z^2+t^3));
H(0) = 1
H(1) = 4
H(t) = 6*t -3    for t >= 2

/**/ R2 := NewFreeModule(R, 2);
/**/ Mgens := matrix(R, [[x^3,y^3], [x*y^2,0], [0,z^3]]);
/**/ M := SubmoduleRows(R2, Mgens);
/**/ HilbertFn(M);
H(0) = 0
H(1) = 0
H(2) = 0
H(3) = 3
H(4) = 12
H(t) = (1/3)*t^3 +(3/2)*t^2 +(-101/6)*t +35    for t >= 5

/**/ HilbertFn(M,3);
3
/**/ HilbertFn(M,5);
30

```

**See Also:** EvalHilbertFn(I-5.13 pg.89), HilbertPoly(I-8.9 pg.124), HVector(I-8.16 pg.128), HilbertSeries(I-8.10 pg.125)

## I-8.8 HilbertMat

syntax

```
HilbertMat(N: INT): MAT over QQ
```

### Description

This function returns the “n”-by-“n” Hilbert matrix over “QQ”.

example

```

/**/ HilbertMat(3);
matrix(QQ,
  [[1, 1/2, 1/3],
   [1/2, 1/3, 1/4],
   [1/3, 1/4, 1/5]])

```

## I-8.9 HilbertPoly

syntax

```
HilbertPoly(R: RING or TAGGED("Quotient")): RINGELEM in the ring QQt.
```

## Description

This function returns the Hilbert polynomial for  $R$  as a polynomial in the standard CoCoA ring “ $QQt$ ” (=  $QQ[t]$ ).

The weights of the indeterminates of “ $R$ ” must all be 1, and the coefficient ring must be a field.

If the input is not homogeneous, the Hilbert polynomial of the corresponding leading term (initial) ideal or module is calculated. For the Hilbert \*function\*, see “HilbertFn” (I-8.7 pg.123).

example

```

/**/ use R ::= QQ[w,x,y,z];
/**/ I := ideal(z^2-x*y, x*z^2+w^3);
/**/ HilbertFn(R/I);
H(0) = 1
H(1) = 4
H(t) = 6*t-3   for t >= 2

/**/ F := HilbertPoly(R/I);
/**/ F; -- a polynomial in the ring Qt
6*t-3

/**/ T := indet(RingOf(F), 1);
/**/ subst(F, T, 3);
15

```

**See Also:** EvalHilbertFn(I-5.13 pg.89), HilbertFn(I-8.7 pg.123), HVector(I-8.16 pg.128), HilbertSeries(I-8.10 pg.125), RingQQt(I-18.48 pg.276)

## I-8.10 HilbertSeries

syntax

```
HilbertSeries(M: MODULE|IDEAL|RING):TAGGED("$hp.PSeries")
```

## Description

This function computes the Hilbert-Poincare series of “ $M$ ”. The input, “ $M$ ”, must be homogeneous (with respect to the first row of the weights matrix). In the standard case, i.e. the weights of all indeterminates are 1, the result is simplified so that the power appearing in the denominator is the dimension of “ $M$ ”.

NOTE: for the local case see “PrimaryHilbertSeries” (I-16.26 pg.243).

NOTES:

- (i) the coefficient ring must be a field.
- (ii) these functions produce tagged objects: they cannot safely be tested for (non-)equality to other values.

Starting from release 4.7.5 the input may also be an ideal.

For more information, see the article: A.M. Bigatti, “Computations of Hilbert-Poincare Series” J. Pure Appl. Algebra, 119/3 (1997), 237–253.

example

```

/**/ use R ::= QQ[t,x,y,z]; -- standard weights
/**/ HilbertSeries(R/ideal(R, []));
(1) / (1-t)^4

/**/ HilbertSeries(R/ideal(t^2, x, y^3));
(1 + 2*t + 2*t^2 + t^3) / (1-t)

```

```

/**/ R2 := NewFreeModule(R, 2); -- MODULE
/**/ M := SubmoduleRows(R2, matrix(R, [[x^2,0], [0,z^3]]));
/**/ HilbertSeries(M);
(t^2 + t^3) / (1-t)^4

-- /**/ HilbertSeries(R2/M); --***WORK IN PROGRESS***

/**/ Ws := RowMat([1,2,3,4]); -- weights and multigradings
/**/ P := NewPolyRing(QQ, "t,x,y,z", MakeTermOrd(Ws), 1);
/**/ use P;
/**/ HilbertSeries(P/ideal(t^2, x, y^3));
--- Non-simplified HilbertPoincare' Series ---
(1 - 2*t^2 + t^4 - t^9 + 2*t^11 - t^13) / ( (1-t)*(1-t^2)*(1-t^3)*(1-t^4) )

/**/ HilbertSeries(ideal(t^2, x, y^3));
--- Non-simplified HilbertPoincare' Series ---
(2*t^2 - t^4 + t^9 - 2*t^11 + t^13) / ( (1-t)*(1-t^2)*(1-t^3)*(1-t^4) )

/**/ Ws := mat([[1,2,3,4],[0,0,5,8]]);
/**/ P := NewPolyRing(QQ, "t,x,y,z", MakeTermOrd(Ws), 2);
/**/ use P;
/**/ HilbertSeries(P/ideal(t^2, x, y^3));
--- Non Simplified Pseries ---
(1 - 2*t[1]^2 + t[1]^4 - t[1]^9*t[2]^15 + 2*t[1]^11*t[2]^15 - t[1]^13*t[2]^15) / ( (1-t[1])^1*(1-t[1]^2)*(1-t[1]^3)*t[2]^15 )

/**/ Ws := mat([[1,2,3,4],[0,0,5,8]]);
/**/ P := NewPolyRing(QQ, "t,x,y,z", MakeTermOrd(Ws), 2);
/**/ use P;
/**/ HilbertSeries(P/ideal(t^2, y^3));
--- Non-simplified HilbertPoincare' Series ---
(1 - t[1]^2 - t[1]^9*t[2]^15 + t[1]^11*t[2]^15) /
((1-t[1])^1*(1-t[1]^2)*(1-t[1]^3*t[2]^5)*(1-t[1]^4*t[2]^8) )

```

**See Also:** [dim\(I-4.17 pg.80\)](#), [multiplicity\(I-13.35 pg.207\)](#), [HilbertFn\(I-8.7 pg.123\)](#), [HVector\(I-8.16 pg.128\)](#), [HilbertSeriesShifts\(I-8.12 pg.127\)](#), [HilbertSeriesMultiDeg\(I-8.11 pg.126\)](#), [GradingMat\(I-7.30 pg.117\)](#), [Primary-HilbertSeries\(I-16.26 pg.243\)](#)

## I-8.11 HilbertSeriesMultiDeg

syntax

```
HilbertSeriesMultiDeg(RmodI: RING, WM: MAT): TAGGED("$hp.PSeries")
```

### Description

This function computes the multigraded Hilbert-Poincare series of “RmodI” wrt the multigrading “WM”. The “I” must be homogeneous wrt the multigrading “WM”.

This function is only a handy shortcut to avoid creating the proper polynomial ring multi-graded with “WM”.

example

```

/**/ use R ::= QQ[x,y];
/**/ HilbertSeriesMultiDeg(R/ideal(Indets(R))^2, mat([[1,1]]));
(1 + 2*t) / (1-t)^0

/**/ HilbertSeriesMultiDeg(R/ideal(Indets(R))^2, mat([[1,0],[0,1]]));

```

```
--- Non-simplified HilbertPoincare' Series ---
(1 - t[2]^2 - t[1]*t[2] - t[1]^2 + t[1]*t[2]^2 + t[1]^2*t[2])
/ ( (1-t[1])*(1-t[2]) )
```

**See Also:** HilbertSeries([I-8.10](#) [pg.125](#))

## I-8.12 HilbertSeriesShifts

syntax

```
HilbertSeriesShifts(M: MODULE, ShiftsList: LIST):TAGGED("$hp.PSeries")
HilbertSeriesShifts(M: TAGGED("Quotient"), ShiftsList: LIST)
:TAGGED("$hp.PSeries")
```

### Description

This function computes the Hilbert-Poincare series (single-graded) module “M” with shifts “sh”.

This function is only a handy shortcut to avoid creating the proper free module with shifts “sh”.

NOTE: functions producing tagged objects cannot safely be compared for equality with other values.

For more information, see the article: A.M. Bigatti, ”Computations of Hilbert-Poincare Series” J. Pure Appl. Algebra, 119/3 (1997), 237–253.

example

```
/**/ use P := QQ[x,y,z];
/**/ F := NewFreeModule(P, ColMat([2,0])); -- P(-2) (+) P(0)
/**/ M := SubmoduleRows(F, mat([[x,y^3], [x-z,0]]));
/**/ HilbertSeries(M);
(2*t^3) / (1-t)^3
/**/ HilbertSeriesShifts(M, [3,1]);
(2*t^4) / (1-t)^3
```

**See Also:** dim([I-4.17](#) [pg.80](#)), HilbertFn([I-8.7](#) [pg.123](#)), HVector([I-8.16](#) [pg.128](#)), multiplicity([I-13.35](#) [pg.207](#)), GradingMat([I-7.30](#) [pg.117](#))

## I-8.13 homog

syntax

```
homog(V: RINGELEM, X: RINGELEM): RINGELEM
homog(V: MODULEELEM, X: RINGELEM): MODULEELEM
homog(L: LIST, X: RINGELEM): LIST
homog(I: IDEAL, X: RINGELEM): IDEAL
homog(M: MODULE, X: RINGELEM): MODULE
```

### Description

This function returns the homogenization of the first arg with respect to the indeterminate “X”, which must have weight 1. The elements of the list “L” are homogenized separately.

NOTE: For an ideal/module the result is the ideal/module containing the homogenizations of all elements (and not simply the homogenizations of the specific generators).

example

```
/**/ use R := QQ[x,y,z,w];
/**/ homog(x^3-y, w);
```

```

x^3 -y*w^2

/**/  homog([x^3-y, x^4-z], w);
[x^3 -y*w^2, x^4 -z*w^3]

/**/  I := ideal(x^3-y, x^4-z);
/**/  homog(I, w);    -- don't just get the homogenizations of
                      -- the generators of I
ideal(x*y -z*w, x^2*z -y^2*w, x^3 -y*w^2, y^3 -x*z^2)

```

**See Also:** [IsHomog\(I-9.52 pg.156\)](#)

## I-8.14 HomogElimMat [OBSOLESCENT]

syntax

[OBSOLESCENT]

### Description

Renamed to “ElimHomogMat” ([I-5.5 pg.85](#)).

## I-8.15 HSaturation

syntax

HSaturation(I: IDEAL, J: IDEAL): IDEAL

### Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This functions returns the saturation of I with respect to J: the ideal of polynomials F such that  $F \cdot G$  is in I for all G in  $J^d$  for some positive integer  $d$ .

It calculates the saturation using a Hilbert-driven algorithm. It differs from “[saturate](#)” ([I-19.1 pg.281](#)) only when the input is inhomogeneous, in which case, “[HSaturation](#)” may be faster.

The coefficient ring must be a field.

example

```

/**/  use R := QQ[x,y];
/**/  I := ideal(x^4-x, y*x-2*x);
/**/  saturate(I, ideal(x));
ideal(y -2, x^3 -1)

HSaturation(I, ideal(x)); -- ***** NOT YET IMPLEMENTED *****

```

**See Also:** [colon\(I-3.30 pg.57\)](#), [HColon\(I-8.2 pg.121\)](#), [saturate\(I-19.1 pg.281\)](#)

## I-8.16 HVector

syntax

HVector(M: RING or TAGGED("Quotient")): LIST

## Description

This function returns the h-vector of “M”, i.e. the coefficients of the numerator of the simplified Poincare series for “M”. “M” can be a module or a quotient.

The weights of the indeterminates of the polynomial ring of “M” must all be 1, and the coefficient ring must be a field.

If the input is not homogeneous, the Hilbert function of the corresponding leading term (initial) ideal or module is calculated.

example

```
/**/ use R := QQ[t,x,y,z];
/**/ HVector(R/ideal(x,y,z)^5);
[1, 3, 6, 10, 15]

/**/ HilbertSeries(R/ideal(x,y,z)^5);
(1 + 3t + 6t^2 + 10t^3 + 15t^4) / (1-t)
```

**See Also:** [HilbertFn\(I-8.7 pg.123\)](#), [HilbertSeries\(I-8.10 pg.125\)](#)



# Chapter I-9

## I

### I-9.1 ID [OBSOLETE]

syntax

[OBSOLETE]

#### Description

[OBSOLETE] renamed “RingID” (I-18.45 pg.274).

**See Also:** RingID(I-18.45 pg.274)

### I-9.2 ideal

syntax

```
ideal(g1: RINGELEM,...,gn: RINGELEM): IDEAL
ideal(L: LIST): IDEAL
ideal(R: RING, L: LIST): IDEAL
```

#### Description

The first form returns the ideal generated by “ $g_1, \dots, g_n$ ”. The second form returns the ideal generated by the polynomials in “ $L$ ” (a bit more flexible than the first form). The third is the same as the second but works also if “ $L = []$ ”.

example

```
/**/ use R := QQ[x,y,z];
/**/ I := ideal(x-y^2, x*y-z);
/**/ I;
ideal(-y^2 +x, x*y -z)

/**/ L := [x*y-z, x-y^2];
/**/ J := ideal(L); -- same as ideal(R, L)
/**/ I = J;
true

/**/ ideal(R, []);
ideal()
```

## I-9.3 IdealAndSeparatorsOfPoints

syntax

```

IdealAndSeparatorsOfPoints(Points: LIST): RECORD

```

where Points is a list of lists of coefficients representing a set of “*{it distinct}*” points in affine space.

### Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This function computes the results of “IdealOfPoints” (I-9.7 pg.135) and “SeparatorsOfPoints” (I-19.6 pg.283) together at a cost lower than making the two separate calls. The result is a record with three fields:

```

points      -- the points given as argument
ideal       -- the result of IdealOfPoints
separators  -- the result of SeparatorsOfPoints

```

Thus, if the result is stored in a variable with identifier X, then: X.points will be the input list of points; X.ideal will be the ideal of the set of points, with generators forming the reduced Groebner basis for the ideal; X.separators will be a list of polynomials whose i-th element will take the value 1 on the i-th point and 0 on the others.

NOTE:

- \* the current ring must have at least as many indeterminates as the dimension of the space in which the points lie;
- \* the base field for the space in which the points lie is taken to be the coefficient ring, which should be a field;
- \* in the polynomials returned, the first coordinate in the space is taken to correspond to the first indeterminate, the second to the second, and so on;
- \* if the number of points is large, say 100 or more, the returned value can be very large. To avoid possible problems when printing such values as a single item we recommend printing out the elements one at a time as in this example:

```

X := IdealAndSeparatorsOfPoints(Pts);
foreach g in gens(X.ideal) do
  println g;
endforeach;

```

For ideals and separators of points in projective space, see “IdealAndSeparatorsOfProjectivePoints” (I-9.4 pg.133).

example

```

use R := QQ[x,y];
Points := [[1, 2], [3, 4], [5, 6]];
X := IdealAndSeparatorsOfPoints(Points);
X.points;
[[1, 2], [3, 4], [5, 6]]
-----
X.ideal;
ideal(x - y + 1, y^3 - 12y^2 + 44y - 48)
-----
X.separators;
[1/8y^2 - 5/4y + 3, -1/4y^2 + 2y - 3, 1/8y^2 - 3/4y + 1]
-----

```

**See Also:** GBM(I-7.3 pg.108), HGBM(I-8.4 pg.122), GenericPoints(I-7.6 pg.109), IdealAndSeparatorsOfProjectivePoints(I-9.4 pg.133), IdealOfPoints(I-9.7 pg.135), IdealOfProjectivePoints(I-9.8 pg.136), Interpolate(I-9.30 pg.148), QuotientBasis(I-17.4 pg.252), SeparatorsOfPoints(I-19.6 pg.283), SeparatorsOfProjectivePoints(I-19.7 pg.284)

## I-9.4 IdealAndSeparatorsOfProjectivePoints

— syntax —

```

IdealAndSeparatorsOfProjectivePoints(Points: LIST): RECORD

where Points is a list of lists of coefficients representing a set of
“{\it distinct}” points in projective space.

```

### Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This function computes the results of “IdealOfProjectivePoints” (I-9.8 pg.136) and “SeparatorsOfProjectivePoints” (I-19.7 pg.284) together at a cost lower than making the two separate calls. The result is a record with three fields:

```

points      -- the points given as argument
ideal       -- the result of IdealOfProjectivePoints
separators  -- the result of SeparatorsOfProjectivePoints

```

Thus, if the result is stored in a variable with identifier X, then: X.ideal will be the ideal of the set of points, with generators forming a reduced Groebner basis for the ideal; X.separators will be a list of homogeneous polynomials whose i-th element will be non-zero (actually 1, using the given representatives for the coordinates of the points) on the i-th point and 0 on the others.

NOTE:

- \* the current ring must have at least one more indeterminate than the dimension of the projective space in which the points lie, i.e, at least as many indeterminates as the length of an element of the input, Points;
- \* the base field for the space in which the points lie is taken to be the coefficient ring, which should be a field;
- \* in the polynomials returned, the first coordinate in the space is taken to correspond to the first indeterminate, the second to the second, and so on;
- \* if the number of points is large, say 100 or more, the returned value can be very large. To avoid possible problems when printing such values as a single item we recommend printing out the elements one at a time as in this example:

```

X := IdealAndSeparatorsOfProjectivePoints(Pts);
foreach g in gens(X.ideal) do
  println g;
endforeach;

```

For ideals and separators of points in affine space, see “IdealAndSeparatorsOfPoints” (I-9.3 pg.132).

— example —

```

use R ::= QQ[x,y,z];
Points := [[0,0,1],[1/2,1,1],[0,1,0]];
X := IdealAndSeparatorsOfProjectivePoints(Points);
X.points;
[[0, 0, 1], [1, 1, 1], [0, 1, 0]]
-----
X.ideal;

```

```

ideal(x*z - (1/2)*y*z,  x*y - (1/2)*y*z,  x^2 - (1/4)*y*z,  y^2*z - y*z^2)
-----
X.separators;
[-2*x + z,  x,  -2*x + y]
-----

use R ::= QQ[t,x,y,z];
Pts := GenericPoints(20); -- 20 random points in projective 3-space
X := IdealAndSeparatorsOfProjectivePoints(Pts);
Len(Gens(X.Ideal)); -- number of generators in the ideal
17
-----
HilbertFn(R/X.Ideal);
H(0) = 1
H(1) = 4
H(2) = 10
H(t) = 20   for t >= 3
-----
F := X.Separators[3];
[Eval(F, P) | P in Pts];
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
-----
Res(R/X.Ideal); -- the resolution of the ideal
0 --> R^10(-6) --> R^24(-5) --> R^15(-4) --> R
-----

```

**See Also:** HGBM(I-8.4 pg.122), GBM(I-7.3 pg.108), GenericPoints(I-7.6 pg.109), IdealAndSeparatorsOfPoints(I-9.3 pg.132), IdealOfPoints(I-9.7 pg.135), IdealOfProjectivePoints(I-9.8 pg.136), Interpolate(I-9.30 pg.148), QuotientBasis(I-17.4 pg.252), SeparatorsOfPoints(I-19.6 pg.283), SeparatorsOfProjectivePoints(I-19.7 pg.284)

## I-9.5 IdealOfGBasis

— syntax —

```

IdealOfGBasis(I: IDEAL): IDEAL

```

### Description

After the “GBasis” (I-7.1 pg.107) of “I” is (explicitely or implicitly) computed, it is stored within “I” for future use. This function returns the ideal generated by the GBasis of “I”, and knows it is a GBasis.

— example —

```

/**/ use R ::= QQ[x,y,z];
/**/ I := ideal(x^10 -x*y -1, x^5*y^5 -x*z -1,  x^5*z^5 -x*z -1);
/**/ J1 := ideal(GBasis(I));
/**/ HasGBasis(J1);
false
/**/ J2 := IdealOfGBasis(I);
/**/ HasGBasis(J2);
true

```

**See Also:** HasGBasis(I-8.1 pg.121)

## I-9.6 IdealOfMinGens

— syntax —

```
IdealOfMinGens(I: IDEAL): IDEAL
```

### Description

It works only in the homogeneous case: for the inhomogeneous case see “MinSubsetOfGens” (I-13.25 pg.202).

This function returns the ideal generated by a minimal set of generators (i.e. with minimal cardinality) of “I”. The minimal set of generators is not necessarily a subset of the given generators.

The coefficient ring must be a field.

— example —

```
/**/ use R := QQ[x,y,z];
/**/ I := ideal(x^2-y^2, z^4-y^4, x^2-z^2);
/**/ IdealOfMinGens(I);
ideal(x^2 -z^2, y^2 -z^2)
/**/ HasGBasis(I);
true
```

**See Also:** MinGens(I-13.16 pg.198), MinSubsetOfGens(I-13.25 pg.202)

## I-9.7 IdealOfPoints

— syntax —

```
IdealOfPoints(P: RING, Points: MAT): IDEAL
where Points is a MAT of coefficients whose rows represent a set of
“{\it distinct}” points in affine space.
```

### Description

This function computes the reduced Groebner basis for the ideal of all polynomials which vanish at the given set of points. It returns the ideal generated by that Groebner basis.

NOTE:

- \* the current ring must have at least as many indeterminates as the dimension of the space in which the points lie, i.e, at least as many indeterminates as “NumCols(Points)”;
- \* the base field for the space in which the points lie is taken to be the coefficient ring, which should be a field;
- \* in the polynomials returned, the first coordinate in the space is taken to correspond to the first indeterminate, the second to the second, and so on;

For ideals of points in projective space, see “IdealOfProjectivePoints” (I-9.8 pg.136).

— example —

```
/**/ use P := QQ[x,y];
/**/ Points := mat([[1, 2], [3, 4], [5, 6]]);
/**/ I := IdealOfPoints(P, Points);
/**/ I;
ideal(x -y +1, y^3 -12*y^2 +44*y -48)

/**/ K := NewFractionField(NewPolyRing(QQ, "a"));
/**/ use K;
/**/ Points := mat([[1,2,0], [3,4,a], [5,1,6]]);
/**/ use P := K[x,y,z], Lex;
/**/ I := IdealOfPoints(P, Points);
```

```

/**/ indent(I);
ideal(
  z^3 +(-a -6)*z^2 +(6*a)*z,
  y +((-a -12)/(6*a^2 -36*a))*z^2 +((a^2 +72)/(6*a^2 -36*a))*z -2,
  x +((2*a -6)/(3*a^2 -18*a))*z^2 +((-2*a^2 +36)/(3*a^2 -18*a))*z -1
)

```

**See Also:** GBM(I-7.3 pg.108), HGBM(I-8.4 pg.122), GenericPoints(I-7.6 pg.109), IdealAndSeparatorsOfPoints(I-9.3 pg.132), IdealAndSeparatorsOfProjectivePoints(I-9.4 pg.133), IdealOfProjectivePoints(I-9.8 pg.136), Interpolate(I-9.30 pg.148), QuotientBasis(I-17.4 pg.252), SeparatorsOfPoints(I-19.6 pg.283), SeparatorsOfProjectivePoints(I-19.7 pg.284)

## I-9.8 IdealOfProjectivePoints

— syntax —

```

IdealOfPoints(P: RING, Points: MAT): IDEAL
where Points is a MAT of coefficients whose rows represent a set of
‘‘{\it distinct}’’ points in projective space.

```

### Description

This function computes the reduced Groebner basis for the ideal of all homogeneous polynomials which vanish at the given set of points. It returns the ideal generated by that Groebner basis.

NOTE:

- \* the current ring must have at least one more indeterminate than the dimension of the projective space in which the points lie, i.e, at least as many indeterminates as “NumCols(Points)”;
- \* the base field for the space in which the points lie is taken to be the coefficient ring, which should be a field;
- \* in the polynomials returned, the first coordinate in the space is taken to correspond to the first indeterminate, the second to the second, and so on;
- \* if the number of points is large, say 100 or more, the returned value can be very large. To avoid possible problems when printing such values as a single item we recommend printing out the elements one at a time as in this example:

```

I := IdealOfProjectivePoints(Pts);
foreach g in gens(I) do
  println g;
endforeach;

```

For ideals of points in affine space, see “IdealOfPoints” (I-9.7 pg.135).

— example —

```

/**/ use P := QQ[x,y,z];
/**/ I := IdealOfProjectivePoints(P, mat([[0,0,1],[1/2,1,1],[0,1,0]]));
/**/ I;
ideal(x*z +(-1/2)*y, x*y +(-1/2)*y, x^2 +(-1/4)*y, y^2*z -y)
/**/ gens(I); -- the reduced Groebner basis
[x*z +(-1/2)*y, x*y +(-1/2)*y, x^2 +(-1/4)*y, y^2*z -y]

```

**See Also:** GBM(I-7.3 pg.108), HGBM(I-8.4 pg.122), GenericPoints(I-7.6 pg.109), IdealAndSeparatorsOfPoints(I-9.3 pg.132), IdealAndSeparatorsOfProjectivePoints(I-9.4 pg.133), IdealOfPoints(I-9.7 pg.135), Interpolate(I-9.30 pg.148), QuotientBasis(I-17.4 pg.252), SeparatorsOfPoints(I-19.6 pg.283), SeparatorsOfProjectivePoints(I-19.7 pg.284)

## I-9.9 IdentityMat

syntax

```
IdentityMat(R: RING, N: INT): MAT
```

### Description

This function returns the NxN identity matrix with entries in “R”.

example

```
/**/ Id := IdentityMat(QQ,3); Id;
matrix(QQ,
  [[1, 0, 0],
   [0, 1, 0],
   [0, 0, 1]])
/**/ type(Id[1,1]);
RINGELEM
/**/ RingOf(Id[1,1]);
QQ
```

## I-9.10 if

syntax

```
If B_1 Then C_1 EndIf
If B_1 Then C_1 Else D EndIf
If B_1 Then C_1 Elif B_2 Then C_2 Elif ... EndIf
If B_1 Then C_1 Elif B_2 Then C_2 Elif ... Else D EndIf
```

where the B\_j are boolean expressions,  
and the C\_j and D are command sequences.

### Description

If “B\_n” is the first in the sequence of the “B\_j” to evaluate to “true”, then “C\_n” is executed. If none of the “B\_j” evaluates to “True”, then “D” is executed if present otherwise nothing is done.

The construct, “Elif B\_j Then C\_j” can be repeated any number of times.

NOTE: the obsolete keyword “Elsif” is no longer allowed.

example

```
/**/ Define MySign(A)
/**/   If A > 0 Then Return 1;
/**/   Elif A = 0 Then Return 0;
/**/   Else Return -1;
/**/   EndIf;
/**/ EndDefine;

/**/ MySign(3);
1
```

**See Also:** [syntax\(?? pg.??\)](#)

## I-9.11 ILogBase

syntax

[OBSOLESCENT]

### Description

Renamed to “FloorLog2, FloorLog10, FloorLogBase” (I-6.15 pg.99).

## I-9.12 image [OBSOLESCENT]

syntax

Image(V: OBJECT, F:TAGGED("RMap")): OBJECT

### Description

In CoCoA-5 homomorphisms are properly implemented as “RINGHOM” (III-10 pg.409). “Image” was the CoCoA-4 function mimicking homomorphisms, in particular “PolyAlgebraHom” (I-16.14 pg.237).

example

```
/**/ use D ::= QQ[x,y]; -- domain
/**/ f := x-y; -- a RINGELEM in D

/**/ use C ::= QQ[a,b,c]; -- codomain

/**/ -- the old trick
/**/ Phi := RMap(a, c^2-a*b); -- OBSOLESCENT
/**/ Image(f, Phi); -- OBSOLESCENT
a*b -c^2 +a

/**/ -- the proper call
/**/ phi := PolyAlgebraHom(D, C, [a, c^2-a*b]); -- a RINGHOM
/**/ phi(f);
a*b -c^2 +a
```

**See Also:** PolyAlgebraHom(I-16.14 pg.237), apply(I-1.13 pg.32), BringIn(I-2.13 pg.43), subst(I-19.48 pg.303)

## I-9.13 implicit

syntax

implicit(SubalgebraGens: LIST): IDEAL  
 implicit(R: RING, SubalgebraGens: LIST): IDEAL

### Description

This function returns the implicitization of the subalgebra generated by the list “SubalgebraGens”.

If provided with a ring “R”, the result is in “R”, otherwise it is in a newly created ring.

NOTE: Some cases have been optimized: if the input is a list of power-products then use “toric” (I-20.12 pg.315). if you know the answer is a hypersurface then use “ImplicitHypersurface” (I-9.14 pg.139).

## example

```

/**/ use S := QQ[s,t];
/**/ implicit([s^3, s^2*t, s*t^2, t^3]);
ideal(x[3]^2 -x[2]*x[4], x[2]*x[3] -x[1]*x[4], x[2]^2 -x[1]*x[3])

/**/ P := QQ[x,y,z,w];
/**/ implicit(P, [s^3, s^2*t, s*t^2, t^3]);
ideal(z^2 -y*w, y*z -x*w, y^2 -x*z)

```

**See Also:** [ImplicitHypersurface\(I-9.14 pg.139\)](#), [ker\(I-11.1 pg.177\)](#)

## I-9.14 ImplicitHypersurface

## syntax

```

ImplicitHypersurface(ParamDescr: LIST): RINGELEM
ImplicitHypersurface(ParamDescr: LIST, Algo: STRING): RINGELEM
ImplicitHypersurface(P: RING, ParamDescr: LIST): RINGELEM
ImplicitHypersurface(P: RING, ParamDescr: LIST, Algo: STRING): RINGELEM

```

### Description

This function returns the implicitization of the hypersurface parametrically described by the list “ParamDescr”. From version CoCoA-5.2.2 it works also for rational parametrization.

The algorithms are described in the JSC paper Abbott, Bigatti, Robbiano “*Implicitization of Hypersurfaces*”

If provided with a polynomial ring “P”, the result is in “P”, otherwise it is in a newly created ring.

Verbosity: 20-80-90.

NOTE: it assumes the input is a correct parametric description of a hypersurface in “ $K^{(\text{len}(\text{ParamDescr})+1)}$ ”!!

## example

```

/**/ P := QQ[x,y,z];
/**/ use S := QQ[s,t];
/**/ ImplicitHypersurface(P, [s^2, s*t, t^2]);
y^2 -x*z
/**/ ImplicitHypersurface(P, [s^2, s*t, t^2], "Direct");
y^2 -x*z
/**/ ImplicitHypersurface(P, [s^2, s*t, t^2], "ElimTH");
y^2 -x*z

/**/ -- Parametrization by rational functions
/**/ K := NewFractionField(RingQQt(1));
/**/ use K;
/**/ ParamDescr := [ (1-t^2)/(1+t^2), 2*t/(1+t^2) ];
/**/ ImplicitHypersurface(ParamDescr);
x[1]^2 +x[2]^2 -1

```

**See Also:** [implicit\(I-9.13 pg.138\)](#)

## I-9.15 ImplicitPlot

## syntax

```

ImplicitPlot(F: POLY, Xrange: LIST, Yrange: LIST)

```

## Description

This function evaluates the first argument, a bivariate polynomial, at a grid of points in the range given by the second and third arguments. The coordinates of the approximate zeroes are output to a file called "CoCoAPlot". See "ImplicitPlotOn" (I-9.16 pg.140) for outputting to another file.

This result can be plotted using your preferred plotting program. For example, start "gnuplot" and then give it the command

```
plot "CoCoAPlot"
```

to see the plot.

example

```
/**/ use R ::= QQ[x,y];
/**/ ImplicitPlot(x^2 + y^2 - 200^2, [-256,256], [-256,256]);
Plotting points...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
800 plotted points have been placed in the file CoCoAPlot
```

**See Also:** ImplicitPlotOn(I-9.16 pg.140), PlotPoints(I-16.9 pg.236)

## I-9.16 ImplicitPlotOn

syntax

```
ImplicitPlotOn(F: POLY, Xrange: LIST, Yrange: LIST, PlotFileName: STRING)
```

## Description

This function is the same as "ImplicitPlot" (I-9.15 pg.139) with a fourth argument giving the name of the file to print on.

Note that the last argument is a STRING, the name of the file, and not an OSTREAM, as for "print on" (I-16.30 pg.245).

example

```
/**/ use R ::= QQ[x,y];
/**/ F := x^2 + y^2 - 100;
/**/ G := ((x+y)^2-1)*(x^2-36);
/**/ H := ((64*y^2-36*x^2)*(36*y^2-64*x^2)*(100*x^2-y^2)-1) * F - 1000^2 * G;

/**/ ImplicitPlotOn(F, [-16,16], [-16,16], "PLOT-circle");
Plotting points...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
640 plotted points have been placed in the file circle

/**/ ImplicitPlotOn(G, [-16,16], [-16,16], "PLOT-lines");
Plotting points...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
1502 plotted points have been placed in the file lines

/**/ ImplicitPlotOn(H, [-16,16], [-16,16], "PLOT-curve");
Plotting points...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
2790 plotted points have been placed in the file curve
```

After having produced the plot files using CoCoA-4, start "gnuplot" and then give it the following commands:

```
plot "circle"
replot "lines"
replot "curve"
```

**See Also:** ImplicitPlot(I-9.15 pg.139), PlotPointsOn(I-16.10 pg.236)

## I-9.17 ImportByRef, ImportByValue

syntax

```
ImportByRef X;
ImportByValue X;
  where ‘\verb&X&’ is the name of a variable in the containing scope.
```

### Description

“*\*\*\*YOU PROBABLY SHOULDN’T USE THESE COMMANDS YET!\*\*\**” It seems that they can be used only inside anonymous functions (see “**func**” (I-6.26 pg.105)).

These commands “import” an external variable by reference or value. “**ImportByValue**” creates a local variable with the given name, and its initial value is taken from the variable of the same name in the context the anonymous function is defined. “**ImportByRef**” creates a reference to the named variable in the context where the anonymous function is defined.

NOTE: Package variables should be accessed directly (via their fully qualified names); they cannot be imported.

example

```
/**/ Define add(X)
/**/   AnonFn := Func(Y) ImportByValue X; Return X+Y; EndFunc;
/**/   Return AnonFn;
/**/ EndDefine;
/**/ add3 := add(3);
/**/ add3(2);
5
```

**See Also:** TopLevel(I-20.10 pg.314), func(I-6.26 pg.105)

## I-9.18 in

syntax

```
[X in L | B: BOOL]
[E | X in L]
[E | X in L and B]
  where L: LIST, B: BOOL, E: expression
  returns LIST
```

### Description

See “List Constructors” (III-5.2 pg.388) for a full description.

example

```
/**/ [N in 1..10 | IsPrime(N)];
[2, 3, 5, 7]

/**/ [N^2 | N in 1..10 and IsPrime(N)];
[4, 9, 25, 49]
```

**See Also:** List Constructors(III-5.2 pg.388), IsIn(I-9.53 pg.157)

## I-9.19 incr, decr

syntax

```
incr(ref X: INT)
decr(ref X: INT)
```

## Description

“incr(ref X)” adds 1 to the value of “X”. “decr(ref X)” subtracts 1 from the value of “X”.

These functions are useful when counting objects or adjusting pointers.

example

```
/**/ L := [(10^k-1)/9 | k in 1..99];
/**/ NPrimes := 0;
/**/ Foreach N in L Do
/**/   If IsPrime(N) Then incr(ref NPrimes); EndIf;
/**/ EndForeach;
/**/ PrintLn "The list L contains ", NPrimes, " primes.";
The list L contains 3 primes.
```

## I-9.20 indent

syntax

```
indent(X: OBJECT)
indent(X: OBJECT, N: INT)
```

## Description

This function prints the argument “X” splitting it into several lines: a “LIST” or “IDEAL” is printed one element per line, a “RECORD” one field per line.

The second optional argument is for setting the level of recursive indentation; it is useful for example when printing a list of records.

example

```
/**/ L := [1,2] >< [3,4];
/**/ L;
[[1, 3], [1, 4], [2, 3], [2, 4]]

/**/ indent(L);
[
  [1, 3],
  [1, 4],
  [2, 3],
  [2, 4]
]

/**/ indent(L,2);
[
  [
    1,
    3
  ],
  --( Further output )--
  [
    2,
    4
  ]
]
```

```

/**/ indent(record[B := 1, A := 2]);
record[
  A := 2,
  B := 1
]

```

**See Also:** [format\(I-6.19 pg.102\)](#)

## I-9.21 *indet*

syntax

```
indet(R: RING, N: INT): RINGELEM
```

### Description

This function returns the N-th indeterminate of the current ring.

example

```

/**/ use R ::= QQ[x,y,z];
/**/ indet(R, 2);
y

```

**See Also:** [IndetSubscripts\(I-9.25 pg.145\)](#), [IndetIndex\(I-9.22 pg.143\)](#), [IndetName\(I-9.23 pg.144\)](#), [indets\(I-9.24 pg.144\)](#), [NumIndets\(I-14.37 pg.224\)](#)

## I-9.22 *IndetIndex*

syntax

```
IndetIndex(X: RINGELEM): INT
```

### Description

This function returns the position in which the indeterminate is listed when the corresponding ring was created.

example

```

/**/ use R ::= QQ[x,y,z];
/**/ IndetIndex(y);
2

/**/ use R ::= QQ[x[1..2],y[1..2]];
/**/ Indets(R);
[x[1,1], x[1,2], x[2,1], x[2,2], y[1], y[2]]

/**/ IndetIndex(x[2,1]);
3

/**/ S ::= QQ[a,b,c];
/**/ IndetIndex(RingElem(S, "b"));
2

```

**See Also:** [indet\(I-9.21 pg.143\)](#), [IndetSubscripts\(I-9.25 pg.145\)](#), [IndetName\(I-9.23 pg.144\)](#), [indets\(I-9.24 pg.144\)](#), [NumIndets\(I-14.37 pg.224\)](#), [UnivariateIndetIndex\(I-21.1 pg.325\)](#)

## I-9.23 IndetName

— syntax —

```
IndetName(X: RINGELEM): STRING
```

### Description

This function returns the name of the indeterminate X as a string (i.e. the letter without the indices).

— example —

```
/**/ use R ::= QQ[x,y,z];
/**/ IndetName(indet(R, 2));
y

/**/ type(It);
STRING

/**/ use R ::= QQ[a, x[1..3]];
/**/ IndetName(Indet(R, 2));
x

/**/ indent(IndetSymbols(R));
[
  record[head := "a", indices := []],
  record[head := "x", indices := [1]],
  record[head := "x", indices := [2]],
  record[head := "x", indices := [3]]
]
```

**See Also:** [indet\(I-9.21 pg.143\)](#), [IndetSubscripts\(I-9.25 pg.145\)](#), [IndetIndex\(I-9.22 pg.143\)](#), [IndetSymbols\(I-9.26 pg.145\)](#), [NumIndets\(I-14.37 pg.224\)](#)

## I-9.24 indets

— syntax —

```
indets(R: RING): LIST
indets(R: RING, X: STRING): LIST
```

### Description

With one argument (a polynomial ring), this function returns the list of indeterminates of that polynomial ring. With two arguments (the second a STRING), it returns the list of all indeterminates whose name is the given string. The indeterminates in the list appear in order of increasing index (see the function “[IndetIndex](#)”).

This function used to be called “[IndetsCalled](#)” up to version CoCoA-5.0.3, and “[AllIndetsCalled](#)” in CoCoA-4.

Additionally, up to version 4.7.3 you could get this list just by giving the name, e.g. “`Use QQ[x[0..4]]; x;`” but this syntax is no longer allowed because it is ambiguous: “`x[2];`” is different from “`X := x; X[2];`”

— example —

```
/**/ S ::= QQ[x,y,z];
/**/ use R ::= QQ[a,b];
/**/ indets(CurrentRing);
[a, b]
/**/ indets(S);
[x, y, z]
/**/ indets(S, "x");
```

```
[x]
/**/ RingElem(S,"x"); -- works also if R is not a polynomial ring
x

/**/ use R := QQ[x[1..4],a[1..2,1..3]];
/**/ indets(R,"x");
[x[1], x[2], x[3], x[4]]
/**/ indets(R,"a");
[a[1,1], a[1,2], a[1,3], a[2,1], a[2,2], a[2,3]]
/**/ indets(R,"b"); -- empty list if no indet has a matching head
[]
```

**See Also:** [indet\(I-9.21 pg.143\)](#), [IndetSubscripts\(I-9.25 pg.145\)](#), [IndetIndex\(I-9.22 pg.143\)](#), [IndetName\(I-9.23 pg.144\)](#), [NumIndets\(I-14.37 pg.224\)](#)

## I-9.25 IndetSubscripts

syntax

```
IndetSubscripts(X: RINGELEM): LIST
```

### Description

This function returns the subscripts of the name of the argument, an indeterminate (used to be called “IndetInd”).

Please note the difference with “IndetIndex” ([I-9.22 pg.143](#)).

example

```
/**/ use R := QQ[x[1..3,1..2],y,z];
/**/ IndetSubscripts(x[3,2]);
[3, 2]
/**/ IndetSubscripts(y);
[]
/**/ IndetIndex(RingElem(R, ["x",3,2]));
6
/**/ IndetSubscripts(RingElem(R, ["x",3,2]));
[3, 2]
```

**See Also:** [indet\(I-9.21 pg.143\)](#), [IndetIndex\(I-9.22 pg.143\)](#), [IndetName\(I-9.23 pg.144\)](#), [IndetSymbols\(I-9.26 pg.145\)](#), [indets\(I-9.24 pg.144\)](#), [NumIndets\(I-14.37 pg.224\)](#)

## I-9.26 IndetSymbols

syntax

```
IndetSymbols(P: RING): RECORD
```

### Description

This function returns the list of the symbols in a polynomial ring. A symbol is a record “with” head (as “IndetName” ([I-9.23 pg.144](#))) and “indices” (as “IndetSubscripts” ([I-9.25 pg.145](#)))

example

```
/**/ use R := QQ[x,y,z];
/**/ indent(IndetSymbols(R));
[
```

```

    record[head := "x", indices := []],
    record[head := "y", indices := []],
    record[head := "z", indices := []]
]

/**/ use R := QQ[a, x[1..3]];
/**/ indent(IndetSymbols(R));
[
    record[head := "a", indices := []],
    record[head := "x", indices := [1]],
    record[head := "x", indices := [2]],
    record[head := "x", indices := [3]]
]
```

**See Also:** [indet\(I-9.21 pg.143\)](#), [IndetSubscripts\(I-9.25 pg.145\)](#), [IndetIndex\(I-9.22 pg.143\)](#), [IndetName\(I-9.23 pg.144\)](#), [NumIndets\(I-14.37 pg.224\)](#), [SymbolRange\(I-19.55 pg.307\)](#)

## I-9.27 InducedHom

syntax

```
InducedHom(RmodI: RING, phi: RINGHOM): RINGHOM
```

### Description

“InducedHom(RmodI, phi)” – where “RmodI” is a QuotientRing, and “phi” is a homomorphism “R --> S” (which must have “BaseRing(RmodI)” as its “domain” (I-4.22 pg.82), and whose “ker” (I-11.1 pg.177) must contain “DefiningIdeal(RmodI)”) gives the homomorphism “R/I --> S” induced by “phi”

“InducedHom(FrF, phi)” – may be partial where “FrF” is a FractionField, gives the homomorphism induced by “phi” (which must have the base ring of FrF as its domain). Note that the resulting homomorphism may be only partial (e.g. if ker(phi) is non-trivial, or if the codomain is not a field).

example

```

/**/ use R := QQ[x,y];
/**/ RmodI := NewQuotientRing(R, ideal(x^2-1));

/**/ use S := QQ[a,b,c];
/**/ SmodJ := NewQuotientRing(S, ideal(a^2-1));

/**/ phi := PolyAlgebraHom(R,S,[a,b]);
/**/ use R;
/**/ phi(x);
a
/**/ RingOf(phi(x)) = S;
true
/**/ psi := CanonicalHom(S,SmodJ)(phi); -- composition of homomorphisms
/**/ psi(x);
(a)
/**/ RingOf(psi(x)) = SmodJ;
true

/**/ theta := InducedHom(RmodI, psi);
/**/ use RmodI;
/**/ theta(x);
(a)
```

**See Also:** domain(I-4.22 pg.82), codomain(I-3.21 pg.53), Composition of RINGHOM(III-10.2 pg.409), BaseRing(I-2.1 pg.37), DefiningIdeal(I-4.5 pg.74), NewQuotientRing(I-14.9 pg.212), NewFractionField(I-14.1 pg.209), CanonicalHom(I-3.3 pg.46), PolyAlgebraHom(I-16.14 pg.237), PolyRingHom(I-16.15 pg.238)

## I-9.28 InitialIdeal

syntax

```
InitialIdeal(I: IDEAL, Inds: LIST): IDEAL
```

### Description

Let “Inds” be a subset of the set of indeterminates, and let 0 be the degree of the remaining indeterminates. The “*initial form with respect to Inds*” of a polynomial “f” is the homogeneous component of “f” of the lowest degree (in contrast with the “*leading form*”, see “LF” (I-12.9 pg.183), “DF” (I-4.14 pg.78)). The “*initial ideal*” of the ideal “I” is the ideal generated by the initial forms of all polynomials in “I”.

If “Inds” is the set of all indeterminates then the initial ideal is also called the “*tangent cone*” of “I” (“TgCone” (I-20.5 pg.312)).

The implementation is based on Lazard’s method (see Kreuzer-Robbiano, Computational Commutative Algebra 2, pg.463).

example

```
/**/ use R ::= QQ[x,y];
/**/ I := ideal(x^3 + x^2 - y^2);
/**/ InitialIdeal(I, [x,y]);
ideal(x^2 - y^2)
/**/ TgCone(I);
ideal(x^2 - y^2)

/**/ use R ::= QQ[x,y];
/**/ I := ideal(x^2 + x*y);
/**/ InitialIdeal(I, [x,y]);
ideal(x^2 + x*y)
/**/ InitialIdeal(I, [x]);
ideal(x*y)
```

**See Also:** TgCone(I-20.5 pg.312), PrimaryHilbertSeries(I-16.26 pg.243)

## I-9.29 insert [OBSOLESCE]

syntax

```
[OBSOLESCE] insert(ref L: LIST, N: INT, E: OBJECT)
```

### Description

This function inserts “E” into “L” as the “N”-th component. Kept just for backward compatibility, it is strongly discouraged for its intrinsic inefficiency. See “append” (I-1.12 pg.31).

example

```
/**/ L := ["a","b","d","e"];
/**/ insert(ref L,3,"c");
/**/ L;
["a", "b", "c", "d", "e"]
```

**See Also:** append(I-1.12 pg.31), remove(I-18.32 pg.268)

## I-9.30 Interpolate

— syntax —

```
Interpolate(Points: LIST, Values: LIST): RINGELEM
```

where `Points` is a list of lists of coefficients representing a set of ‘‘{\it distinct}’’ points and `Values` is a list of the same size containing numbers from the coefficient ring.

### Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This function returns a multivariate polynomial which takes given values at a given set of points.

NOTE:

- \* the current ring must have at least as many indeterminates as the dimension of the space in which the points lie;
- \* the base field for the space in which the points lie is taken to be the coefficient ring, which should be a field;
- \* in the polynomials returned, the first coordinate in the space is taken to correspond to the first indeterminate, the second to the second, and so on;
- \* if the number of points is large, say 100 or more, the returned value can be very large. To avoid possible problems when printing such values as a single item we recommend printing out the elements one at a time as in this example:

— example —

```
X := Interpolate(Pts, Vals);
foreach element in X do
  println element;
endforeach;

use QQ[x,y];
Points := [[1/2, 2], [3/4, 4], [5, 6/11], [-1/2, -2]];
Values := [1/2, 1/3, 1/5, -1/2];
F := Interpolate(Points, Values);
F;
-46849/834000y^2 - 1547/52125x + 13418/52125y + 46849/208500
-----
[Eval(F, P) | P in Points] = Values;  -- check
True
-----
```

## I-9.31 interreduce, interreduced

— syntax —

```
interreduce(ref L: LIST of RINGELEM)
interreduced(L: LIST of RINGELEM): LIST of RINGELEM
```

### Description

These functions reduce each polynomial using the other polynomials as reduction rules. The process terminates when each is in normal form with respect to the others. The function “**interreduce**” takes a variable containing a list and overwrites that variable with the interreduced list. The second returns an interreduced list without affecting its arguments.

## example

```

/**/ use QQ[x,y,z];
/**/ interreduced([x^3-x*y^2+y*z, x*y, z]);
[x*y, z, x^3]

/**/ L := [x^3-x*y^2+y*z, x*y, z];
/**/ interreduce(ref L);
/**/ L;
[x*y, z, x^3]

```

## I-9.32 intersection

## syntax

```

intersection(A: LIST, B: LIST): LIST
intersection(A: IDEAL, B: LIST): LIST
intersection(A: LIST, B: IDEAL): LIST
intersection(A: IDEAL, B: IDEAL): IDEAL

```

### Description

This function returns the intersection of “A” and “B”.

The coefficient ring must be a field.

NOTE: To compute the intersection of ideals corresponding to zero-dimensional schemes, see the commands “GBM” (I-7.3 pg.108) and “HGBM” (I-8.4 pg.122).

## example

```

/**/ use R := QQ[x,y,z];
/**/ intersection(ideal(x,y,z), ideal(x*y));
ideal(x*y)

/**/ intersection(["a","b","c"], ["b","c","d"]);
["b", "c"]
-----

```

**See Also:** GBM(I-7.3 pg.108), HGBM(I-8.4 pg.122), IntersectList(?? pg.??)

## I-9.33 IntersectionList

## syntax

```

IntersectionList(L: LIST of LIST): LIST
IntersectionList(L: LIST of IDEAL): IDEAL
IntersectionList(L: LIST of MODULE): MODULE

```

### Description

This function returns the intersection of all elements in “L”. Generalizes “intersection” (I-9.32 pg.149).

## example

```

/**/ use R := QQ[x,y,z];
/**/ Points := [[0,0],[1,0],[0,1],[1,1]]; -- a list of points in the plane
/**/ IntersectionList([ideal(x-P[1]*z, y-P[2]*z) | P in Points]);
ideal(y^2 - y*z, x^2 - x*z)

```

```
/**/ IntersectionList([ 1..7, 3..10, 0..5 ]);
[3, 4, 5]
```

**See Also:** [intersection\(I-9.32 pg.149\)](#), [IdealOfProjectivePoints\(I-9.8 pg.136\)](#), [IdealOfPoints\(I-9.7 pg.135\)](#), [HGBM\(I-8.4 pg.122\)](#), [intersection\(I-9.32 pg.149\)](#)

## I-9.34 inverse

syntax

```
inverse(X: MAT): MAT
```

### Description

This function computes the multiplicative inverse of its argument. It is included for use when writing `inverse(X)` comes more naturally than writing “ $X^{-1}$ ”, though both notations are functionally equivalent.

example

```
/**/ inverse(mat([[1,2], [3,4]]));
matrix(QQ,
  [[-2, 1],
   [3/2, -1/2]])
```

**See Also:** [adj\(I-1.2 pg.27\)](#)

## I-9.35 InverseSystem

syntax

```
InverseSystem(I: IDEAL, D: INT): LIST of RINGELEM
```

### Description

Thanks to Enrico Carlini.

Given an ideal of derivations “I”, and an integer “D”, this function computes the degree “D” part of the inverse system of “I”.

For the sake of simplicity Forms/Polynomials and Derivations live in the same ring, the distinction between them is purely formal.

example

```
/**/ use QQ[x,y,z];
/**/ InverseSystem(ideal(x^3+x*y*z), 3);
[z^3, y*z^2, x*z^2, y^2*z, x^2*z, y^3, x*y^2, x^2*y, x^3 - 6*x*y*z]
```

**See Also:** [DerivationAction\(I-4.11 pg.77\)](#), [PerpIdealOfForm\(I-16.6 pg.234\)](#)

## I-9.36 IO.SprintTrunc

syntax

```
$io.SprintTrunc(E: OBJECT, N: INT): STRING
```

### Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This function works like “**sprint**” (I-19.29 pg.294), turning the value of the expression *E* into a string, but if the string has length greater than *N*-1, it is truncated and the string “...” is concatenated. This function is useful in formatting reports of results.

example

```
use R := QQ[x,y];
I := ideal(x,y);
$io.SprintTrunc(I,4);
Idea...
-----
```

**See Also:** [format\(I-6.19 pg.102\)](#), [sprint\(I-19.29 pg.294\)](#)

## I-9.37 *iroot*

syntax

```
iroot(N: INT, R: INT): INT
```

### Description

This function computes the “*R*”-th root of the integer “*N*”. If the argument is not a perfect “*R*”-th power it returns the integer part of the root.

example

```
/**/ iroot(25, 2);
5
/**/ iroot(99, 3);
4
/**/ iroot(-1, 3);
-1
```

**See Also:** [FloorLog2](#), [FloorLog10](#), [FloorLogBase\(I-6.15 pg.99\)](#)

## I-9.38 *IsAntiSymmetric*

syntax

```
IsAntiSymmetric(M: MAT): BOOL
```

### Description

This function tests whether the square matrix “*M*” is anti-symmetric.

example

```
/**/ M := mat([[0, 1, 2], [-1, 0, 3], [-2, -3, 0]]);
/**/ IsAntiSymmetric(M);
true
```

**See Also:** [IsSymmetric\(I-9.85 pg.168\)](#)

## I-9.39 *IsCommutative*

syntax

```
IsCommutative(R: RING): BOOL
```

## Description

This function tests whether the ring “R” is commutative.

example

```
/**/ IsCommutative(ZZ);
true
/**/ IsCommutative(NewWeylAlgebra(ZZ,"x,y"));
false
```

**See Also:** NewWeylAlgebra([I-14.12](#) pg.214)

## I-9.40 IsConstant

syntax

```
IsConstant(X: RINGELEM): BOOL
```

## Description

This function tests whether the value of a RINGELEM of a polynomial ring actually lies in the image of the coefficient ring. It is equivalent to checking that the degree in each indeterminate is 0.

example

```
/**/ QQx ::= QQ[x];
/**/ use QQx[y,z];
/**/ IsConstant(y+1);
false
/**/ IsConstant(x+1);
true
```

**See Also:** indets([I-9.24](#) pg.144)

## I-9.41 IsContained

syntax

```
IsContained(A: IDEAL, B: IDEAL): BOOL
IsContained(A: MODULE, B: MODULE): BOOL
```

## Description

This function tests whether A is contained in B. Was “<=” in CoCoA-4: this syntax is no longer supported.

example

```
/**/ use QQ[x,y,z];
/**/ IsContained(ideal(x), ideal(x+y, x-y));
true
```

**See Also:** IsIn([I-9.53](#) pg.157), IsSubset([I-9.83](#) pg.167)

## I-9.42 IsCoprime

syntax

```
IsCoprime(t1: RINGELEM, t2: RINGELEM): BOOL
```

## Description

This function tests whether two POWER-PRODUCTS “t1” and “t2” are coprime. (“.. is coprime with ..”).

example

```
/**/ use QQ[x,y,z];
/**/ IsCoprime(x*y, y*z);
false
/**/ IsCoprime(x*y, z);
true
```

**See Also:** IsDivisible(I-9.45 pg.153)

## I-9.43 IsDefined

syntax

```
IsDefined(E)
```

## Description

This function returns true if “E” is defined, otherwise it returns false. Typically, it is used to check if a name has already been assigned.

To know if a field in a record has been assigned use “fields” (I-6.7 pg.96).

example

```
/**/ IsDefined(MyVariable);
false

/**/ MyVariable := 3;
/**/ IsDefined(MyVariable);
true
```

**See Also:** fields(I-6.7 pg.96)

## I-9.44 IsDiagonal

syntax

```
IsDiagonal(M: MAT): BOOL
```

## Description

This function tests whether the square matrix M is diagonal.

example

```
/**/ M := mat([[0, 1, 2],[-1, 0, 3],[-2, -3, 0]]);
/**/ IsDiagonal(M);
false
```

**See Also:** IsSymmetric(I-9.85 pg.168), DiagMat(I-4.15 pg.79)

## I-9.45 IsDivisible

syntax

```
IsDivisible(A: RINGELEM, B: RINGELEM): BOOL
```

## Description

This function says whether “A” is divisible by “B”; it returns “**true**” if so, otherwise “**false**”.

example

```
/**/ use QQ[x,y,z];
/**/ IsDivisible(x, x^2*(y-1));
false
/**/ IsDivisible(x^2*(y-1), x);
true
```

**See Also:** FactorMultiplicity([I-6.4](#) pg.94), IsCoprime([I-9.42](#) pg.152)

## I-9.46 IsElem

syntax

```
IsElem(A: RINGELEM, B: IDEAL): BOOL
IsElem(A: MODULEELEM, B: MODULE): BOOL
```

## Description

This function tests whether A is an element of B. Same as the command “**IsIn**” ([I-9.53](#) pg.157), but works on fewer types: it is in CoCoA-5 for compatibility with the C++ function in CoCoALib.

example

```
/**/ use QQ[x,y,z];
/**/ IsElem(x, ideal(x+y, x-y));
true

/**/ x IsIn ideal(x+y, x-y);
true
```

**See Also:** IsIn([I-9.53](#) pg.157)

## I-9.47 IsEven, IsOdd

syntax

```
IsEven(N: INT): BOOL
IsOdd(N: INT): BOOL
```

## Description

These functions test whether an integer is even or odd.

example

```
/**/ IsEven(3);
false
/**/ IsOdd(3);
true
```

**See Also:** IsZero([I-9.90](#) pg.170)

## I-9.48 IsFactorClosed

syntax

```
IsFactorClosed(L: LIST of power products): BOOL
```

## Description

A set of power products is factor closed iff it contains every factor of every one of its elements. This function checks whether the given set is factor closed (also known as "order-ideal"). It is an error if the list "L" is empty.

example

```
/**/ use P ::= QQ[x,y,z];
/**/ IsFactorClosed([1, x, x^2]);
true
/**/ IsFactorClosed([one(P), y^2]);
false
```

**See Also:** [QuotientBasis\(I-17.4 pg.252\)](#), [LT\(I-12.21 pg.188\)](#), [ApproxPointsNBM\(I-1.14 pg.32\)](#), [IsStronglyStable\(I-9.82 pg.167\)](#)

## I-9.49 IsField

syntax

```
IsField(R: RING): BOOL
```

## Description

This function tests whether a ring is a field.

example

```
/**/ IsField(ZZ);
false
/**/ IsField(QQ);
true
```

**See Also:** [IsFiniteField\(I-9.50 pg.155\)](#)

## I-9.50 IsFiniteField

syntax

```
IsFiniteField(R: RING): BOOL
```

## Description

This function tests whether a ring is a finite field.

example

```
/**/ IsFiniteField(ZZ);
false
/**/ IsFiniteField(QQ);
false
/**/ Fp:=ZZ/(7); IsFiniteField(Fp);
true
```

**See Also:** [IsField\(I-9.49 pg.155\)](#), [IsPthPower\(I-9.73 pg.164\)](#), [LogCardinality\(I-12.18 pg.187\)](#), [PthRoot\(I-16.39 pg.250\)](#)

## I-9.51 IsFractionField

syntax

```
IsFractionField(R: RING): BOOL
```

## Description

This function tests whether the ring “R” is a fraction field, i.e. is “QQ” ([I-17.1 pg.251](#)) or has been constructed with “NewFractionField” ([I-14.1 pg.209](#)).

example

```
/**/ use R := QQ[x,y];
/**/ K := NewFractionField(R);
/**/ IsFractionField(K);
true;
/**/ BaseRing(K);
RingWithID(3, "QQ[x,y]")
```

**See Also:** NewFractionField([I-14.1 pg.209](#)), BaseRing([I-2.1 pg.37](#))

## I-9.52 IsHomog

syntax

```
IsHomog(F: RINGELEM|MODULEELEM): BOOL
IsHomog(L: LIST): BOOL
IsHomog(I: IDEAL|MODULE): BOOL
```

## Description

The first form of this function returns True if F is homogeneous. The second form returns True if every element of L is homogeneous. Otherwise, they return False. The third form returns True if the ideal/module can be generated by homogeneous elements, and False if not. Homogeneity is with respect to the first row of the weights matrix.

NOTE: when the grading dimension is 0 everything is trivially true. For safety reasons (from version CoCoALib-5.0.3) “IsHomog” throws an error in this case, e.g. “IsHomog(x-1)” gives error instead of a possibly misleading “true”.

example

```
/**/ use R := QQ[x,y];
/**/ IsHomog(x^2-x*y);
true

/**/ IsHomog(x-y^2);
false

/**/ IsHomog([x^2-x*y, x-y^2]);
false

/**/ R := NewPolyRing(QQ, "x,y", mat([[2,3],[1,2]]), 1);
/**/ use R;
/**/ IsHomog(x^3*y^2+y^4);
true

/**/ R := NewPolyRing(QQ, "x,y", mat([[2,3],[1,2]]), 2);
/**/ use R;
/**/ IsHomog(x^3*y^2+y^4);
false

/**/ use R := QQ[x,y];
/**/ IsHomog(ideal(x^2+y,y));
true
```

```

/**/ use R ::= QQ[x,y], Lex; -- note: GradingDim = 0
-- /**/ IsHomog(x-1); -- !!! ERROR !!! as expected, instead of "true"

```

**See Also:** [deg\(I-4.6 pg.74\)](#), [wdeg\(I-23.1 pg.331\)](#)

## I-9.53 *IsIn*

syntax

```

X IsIn Y          (return BOOL)

```

### Description

The semantics of “*IsIn*” is explained in the following table:

-----			
OBJECT	IsIn	LIST	checks if the list contains the object.
POLY	IsIn	IDEAL	checks for ideal membership.
MODULEELEM	IsIn	MODULE	checks for module membership.
STRING	IsIn	STRING	checks if the first string is a substring
			of the second one.
-----			
IsIn operator			

## I-9.54 *IsIndet*

syntax

```

IsIndet(X: RINGELEM): BOOL

```

### Description

This function tests whether “*X*” is an indeterminate. If so, it returns “**true**”; otherwise it returns “**false**”. An error is signalled if “*X*” is not a “*RINGELEM*” or if “*RingOf(X)*” is not a polynomial ring.

example

```

/**/ use QQ[x,y,z];
/**/ IsIndet(x);
true
/**/ IsIndet(x-1);
false

```

## I-9.55 *IsInImage*

syntax

```

IsInImage(phi: RINGHOM, f: RINGELEM): BOOL

```

### Description

This function checks if “*f*” is in the image of “*phi*” (better use “*preimage0*” ([I-16.18 pg.239](#)) directly).

example

```

/**/ QQxyz := QQ[x,y,z];
/**/ QQab  := QQ[a,b];

/**/ use QQab;
/**/ phi := PolyAlgebraHom(QQxyz, QQab, [a+1, a*b+3, b^2]);

/**/ IsInImage(phi, b);
false
/**/ preimage0(phi, b);
0

```

**See Also:** [IsSurjective\(I-9.84 pg.168\)](#), [preimage0\(I-16.18 pg.239\)](#)

## I-9.56 IsInjective

syntax

```
IsInjective(phi: RINGHOM): BOOL
```

### Description

This function checks if a RINGHOM is injective.

example

```

/**/ QQxyz := QQ[x,y,z];
/**/ QQab  := QQ[a,b];

/**/ use QQab;
/**/ phi := PolyAlgebraHom(QQxyz, QQab, [a+1, a*b+3, b^2]);
/**/ IsInjective(phi);
false
/**/ ker(phi);
ideal(-x^2*z + y^2 + 2*x*z - 6*y - z + 9)
/**/ IsSurjective(phi);
false

/**/ use QQab;
/**/ preimage0(phi, b);
0

/**/ preimage0(phi, a^2);
x^2 - 2*x + 1

/**/ phi(RingElem(QQxyz, "x^2 - 2*x + 1"));
a^2
/**/ phi(RingElem(QQxyz, "x^2 - 2*x + 1 + (-x^2*z + y^2 + 2*x*z - 6*y - z + 9)"));
a^2

```

**See Also:** [ker\(I-11.1 pg.177\)](#), [preimage0\(I-16.18 pg.239\)](#), [IsSurjective\(I-9.84 pg.168\)](#)

## I-9.57 IsInRadical

syntax

```
IsInRadical(F: RINGELEM, I: IDEAL): BOOL
IsInRadical(J: IDEAL, I: IDEAL): BOOL
```

## Description

This function tests whether the first argument, a polynomial or an ideal, is contained in the radical of the second argument, an ideal.

This function is much faster than asking “F IsIn Radical(I);”.

example

```

/**/ use QQ[x,y,z];
/**/ I := ideal(x^6*y^4, z);
/**/ IsInRadical(x*y, I);
true
/**/ IsInRadical(ideal(x,y), I);
false
/**/ MinPowerInIdeal(x*y, I);
6

```

**See Also:** [MinPowerInIdeal\(I-13.24 pg.201\)](#), [radical\(I-18.1 pg.255\)](#)

## I-9.58 IsInSubalgebra [OBSOLETE]

syntax

```

[OBSOLETE]

```

## Description

See “SubalgebraHom” ([I-19.40 pg.300](#)).

**See Also:** [SubalgebraHom\(I-19.40 pg.300\)](#)

## I-9.59 IsInteger

syntax

```

IsInteger(ref n: INT, f: RINGELEM): BOOL

```

## Description

This function tests whether the argument “f” is integer and convert it into an “INT”. To convert “f” straight away use “AsINT(f)”.

example

```

/**/ use R ::= QQ[x];
/**/ f := x-x-3; f; type(f);
-3
RINGELEM
/**/ IsInteger(ref a, x-x-3);
true
/**/ a; type(a);
-3
INT

```

**See Also:** [AsINT\(I-1.19 pg.35\)](#), [AsRAT\(I-1.20 pg.36\)](#), [IsRational\(I-9.78 pg.166\)](#)

## I-9.60 IsIntegralDomain

syntax

```
IsIntegralDomain(R: RING): BOOL
```

### Description

This function tests whether the ring “R” is integral.

example

```
/**/ IsIntegralDomain(ZZ);
true
/**/ IsIntegralDomain(NewZZmod(6));
false
```

**See Also:** IsField([I-9.49](#) pg.155)

## I-9.61 IsInvertible

syntax

```
IsInvertible(f: RINGELEM): BOOL
```

### Description

This function tests whether the argument “f” is invertible in “RingOf(f)”.

example

```
/**/ use R := QQ[x];
/**/ Q := R/ideal(x^2+1);
/**/ use Q;
/**/ IsInvertible(x-2);
true
/**/ 1/(x-2);
((-1/5)*x -2/5)
```

## I-9.62 IsIrred

syntax

```
IsIrred(f: RINGELEM): BOOL
```

### Description

This function tests whether the argument “f” is irreducible in “RingOf(f)”.

example

```
/**/ FFp := NewRingFp(7);
/**/ use FFp[x];
/**/ f := x^9+x+1;
/**/ IsIrred(f);
true
```

## I-9.63 IsLexSegment

syntax

```
IsLexSegment(I: IDEAL): BOOL
```

## Description

This function tests whether the monomial ideal  $I$  is a lex-segment ideal.

example

```
/**/ use R ::= QQ[x,y,z];
/**/ I := ideal(x*y^3, y^4, x^3, x^2*y, x^2*z);
/**/ IsLexSegment(I);
false
```

**See Also:** [IsStable\(I-9.80 pg.166\)](#), [IsStronglyStable\(I-9.82 pg.167\)](#), [LexSegmentIdeal\(I-12.8 pg.182\)](#)

## I-9.64 IsMaximal

syntax

```
IsMaximal(I: IDEAL): BOOL
```

## Description

This function determines whether an ideal is maximal. (Not yet implemented for small characteristic)

example

```
/**/ use P ::= QQ[x,y,z];
/**/ IsMaximal(ideal(x^2+1, (x-5*y+4*z), z^3+z-1));
true
/**/ IsMaximal(ideal(x^2+1, (x-5*y+4*z)*(y-3), z^3+z-1));
false
```

**See Also:** [IsPrimary\(I-9.70 pg.163\)](#), [IsRadical\(I-9.77 pg.165\)](#)

## I-9.65 IsMinusOne

syntax

```
IsOne(X: OBJECT): BOOL
```

## Description

This function tests whether its argument is  $-1$ ; the argument can be of almost any type for which “ $-1$ ” makes sense.

example

```
/**/ IsMinusOne(23);
false
/**/ IsMinusOne(3/-3);
true
/**/ use R ::= QQ[x,y,z];
/**/ IsMinusOne(-x/x);
true
```

**See Also:** [IsOne\(I-9.67 pg.162\)](#)

## I-9.66 IsNumber [OBSOLETE]

syntax

```
[OBSOLETE]
```

## Description

[OBSOLETE] See “IsInteger”, “IsRational”

## I-9.67 IsOne

syntax

```
IsOne(X: OBJECT): BOOL
```

## Description

This function tests whether its argument is one; the argument can be of almost any type for which “one” makes sense.

example

```
/**/ IsOne(23);
false
/**/ IsOne(3/3);
true
/**/ use R ::= QQ[x,y,z];
/**/ IsOne(1);
false
/**/ IsOne(ideal(x^2, x^2-1));
false
```

**See Also:** IsEven, IsOdd([I-9.47](#) pg.154), one([I-15.1](#) pg.227), IsZero([I-9.90](#) pg.170)

## I-9.68 IsPolyRing

syntax

```
IsPolyRing(R: RING): BOOL
```

## Description

This function tests whether its argument is a polynomial ring.

example

```
/**/ use P ::= QQ[x,y];
/**/ IsPolyRing(P);
true
/**/ PmodI := NewQuotientRing(P,ideal([x])); // NO, but isom to QQ[y]
false
/**/ IsPolyRing(QQ);
false
```

**See Also:** IsQuotientRing([I-9.76](#) pg.165), NewPolyRing([I-14.8](#) pg.212)

## I-9.69 IsPositiveGrading

syntax

```
IsPositiveGrading(M: MAT): BOOL
IsPositiveGrading(M: MAT,N: INT): BOOL
```

## Description

This function determines whether a matrix of integers defines a positive grading, i.e. each column has a non-zero entry and its first non-zero entry is positive.

NOTE: it also requires that the matrix has maximal rank (from version CoCoA-5.1.3).

example

```
/**/ IsPositiveGrading(LexMat(5));
true
/**/ IsPositiveGrading(submat(LexMat(5),1..3,1..5)); --only the first 3 rows
false
/**/ IsPositiveGrading(mat([[0,2,3], [1, -1, 0]]));
true
/**/ IsPositiveGrading(mat([[1,1], [0,-1]]));
true
/**/ IsPositiveGrading(mat([[1,1], [0,-1], [-1, 0]])); --not maximal rank
false
```

**See Also:** HilbertSeriesMultiDeg([I-8.11](#) pg.126)

## I-9.70 IsPrimary

syntax

```
IsPrimary(I: IDEAL): BOOL
```

## Description

This function determines whether an ideal is primary.

example

```
/**/ use P ::= QQ[x,y,z];
/**/ IsPrimary(ideal(x^2, y^2, z));
true
/**/ IsPrimary(ideal(x*(x-1), y^2, z));
false
```

**See Also:** PrimaryDecomposition0([I-16.23](#) pg.241), IsMaximal([I-9.64](#) pg.161), IsRadical([I-9.77](#) pg.165)

## I-9.71 IsPrime

syntax

```
IsPrime(N: INT): BOOL
```

## Description

This function determines whether a positive integer is prime; if N is not positive, an error is signalled. This function may be extremely slow when N is a large prime; in practice it is usually better to call “IsProbPrime” ([I-9.72](#) pg.164).

For the curious: currently, the function first performs a probabilistic check (Miller-Rabin), if that passes, it then verifies primality (via the Lucas test).

example

```
/**/ IsPrime(32003);
true
```

```
/**/ IsPrime(10^100);
false
```

**See Also:** [IsProbPrime\(I-9.72 pg.164\)](#), [NextPrime\(?? pg.??\)](#)

## I-9.72 IsProbPrime

syntax

```
IsProbPrime(N: INT): BOOL
```

### Description

This function returns True if its integer argument passes a fairly stringent primality test; otherwise it returns False. There is a very small chance of the function returning True even though the argument is composite; if it returns False, we are certain that the argument is composite. Some people call it a compositeness test.

example

```
/**/ IsProbPrime(2);
true

/**/ IsProbPrime(1111111111111111111);
true

/**/ [N in 1..1111 | IsProbPrime((10^N-1)/9)]; -- only five values are known
[2, 19, 23, 317, 1031] -- next might be 49081
```

**See Also:** [IsPrime\(I-9.71 pg.163\)](#), [NextProbPrime\(?? pg.??\)](#)

## I-9.73 IsPthPower

syntax

```
IsPthPower(X: RINGELEM): BOOL
```

### Description

This function determines whether a polynomial over a finite field (of char p) is a p-th power. If the coefficient ring is not a finite field then an error is signalled.

example

```
/**/ use ZZ/(7)[x];
/**/ IsPthPower(x^7+3);
true
/**/ IsPthPower(x^6+3);
false
```

**See Also:** [IsFiniteField\(I-9.50 pg.155\)](#), [PthRoot\(I-16.39 pg.250\)](#)

## I-9.74 IsQQ

syntax

```
IsQQ(R: RING): BOOL
```

## Description

This function tests whether a ring is the ring of rationals.

example

```
/**/ R := QQ[x,y];
/**/ IsQQ(CoeffRing(R));
true;
```

**See Also:** [QQ\(I-17.1 pg.251\)](#), [RingQQ\(I-18.47 pg.276\)](#), [IsZZ\(I-9.94 pg.172\)](#)

## I-9.75 isqrt

syntax

[OBSOLESCENT]

## Description

Renamed to “FloorSqrt” ([I-6.16 pg.100](#)).

## I-9.76 IsQuotientRing

syntax

```
IsQuotientRing(R: RING): BOOL
```

## Description

This function tests whether a ring is a quotient ring; it returns “true” if the ring is a quotient ring.

example

```
/**/ use R := QQ[x,y];
/**/ S := R/ideal(x);
/**/ IsQuotientRing(S);
true;
```

**See Also:** [DefiningIdeal\(I-4.5 pg.74\)](#)

## I-9.77 IsRadical

syntax

```
IsRadical(I: IDEAL): BOOL
```

## Description

This function tests whether the IDEAL “I” is radical. Currently works only for 0-dimensional ideals.

example

```
/**/ use R := QQ[x,y,z];
/**/ I := ideal(x^2-1, y^2-2, z^3);
/**/ IsRadical(I);
false
/**/ I := ideal(x^2-1, y^2-2, z^3-3);
/**/ IsRadical(I);
true
```

**See Also:** [radical\(I-18.1 pg.255\)](#), [IsMaximal\(I-9.64 pg.161\)](#), [IsPrimary\(I-9.70 pg.163\)](#)

## I-9.78 IsRational

— syntax —

```
IsRational(ref n: RAT, f: RINGELEM): BOOL
```

### Description

This function tests whether the argument “f” is rational and convert it into a “RAT”. To convert “f” straight away use “AsRAT(f)”.

— example —

```
/**/ use R := QQ[x];
/**/ f := x-x-3; f; type(f);
-3
RINGELEM
/**/ IsRational(ref a, x-x-3);
true
/**/ a; type(a);
-3
RAT
```

**See Also:** [AsINT\(I-1.19 pg.35\)](#), [AsRAT\(I-1.20 pg.36\)](#), [IsInteger\(I-9.59 pg.159\)](#)

## I-9.79 IsSqFree

— syntax —

```
IsSqFree(f: RINGELEM): BOOL
IsSqFree(n: INT): BOOL
```

### Description

This function tests whether the argument is square-free. A ring elem “f” must belong to a polynomial ring over a field. Currently, it may wrongly declare as square-free an integer “n” with a repeated large prime factor.

— example —

```
/**/ use R := QQ[x];
/**/ IsSqFree(x^2);
false
/**/ IsSqFree(101);
true
```

**See Also:** [radical\(I-18.1 pg.255\)](#), [factor\(I-6.1 pg.93\)](#)

## I-9.80 IsStable

— syntax —

```
IsStable(I: IDEAL): BOOL
```

### Description

This function tests whether the monomial ideal I is stable.

example

```

/**/ use R := QQ[x,y,z];
/**/ I := ideal(x*y^3, y^4, x^3, x^2*y, x^2*z);
/**/ IsStable(I);
true

```

**See Also:** IsLexSegment([I-9.63](#) pg.160), IsStronglyStable([I-9.82](#) pg.167), LexSegmentIdeal([I-12.8](#) pg.182)

## I-9.81 IsStdGraded

syntax

```
IsStdGraded(P: RING): BOOL
```

### Description

This function tests whether “P” is standard graded, “*i.e.*” “GradingDim” is 1 and all indeterminates in “P” have degree 1.

example

```

/**/ P := QQ[x,y,z];
/**/ IsStdGraded(P);
true
/**/ P := QQ[x,y,z], lex;
/**/ IsStdGraded(P);
false
/**/ P := NewPolyRing(QQ, "x,y", mat([[2,3],[1,2]]), 1);
/**/ IsStdGraded(P);
false

```

**See Also:** NewPolyRing([I-14.8](#) pg.212), wdeg([I-23.1](#) pg.331)

## I-9.82 IsStronglyStable

syntax

```
IsStronglyStable(I: IDEAL): BOOL
```

### Description

This function tests whether the monomial ideal I is strongly stable (Borel-fixed in characteristic 0).

example

```

/**/ use R := QQ[x,y,z];
/**/ I := ideal(x*y^3, y^4, x^3, x^2*y, x^2*z);
/**/ IsStronglyStable(I);
true

```

**See Also:** IsLexSegment([I-9.63](#) pg.160), IsStable([I-9.80](#) pg.166)

## I-9.83 IsSubset

syntax

```
IsSubset(L: LIST, M: LIST): BOOL
```

## Description

This function returns “true” if “MakeSet(L)” is contained in “MakeSet(M)”; otherwise it returns “false”.

example

```
/**/ IsSubset([1,1,2],[1,2,3,"a"]);
true
/**/ IsSubset([1,2],["a","b"]);
false
/**/ IsSubset([], [1,2]);
true
```

**See Also:** IsContained(I-9.41 pg.152), IsIn(I-9.53 pg.157), EqSet(I-5.8 pg.86), MakeSet(I-13.3 pg.192), subsets(I-19.47 pg.303)

## I-9.84 IsSurjective

syntax

```
IsSurjective(phi: RINGHOM): BOOL
```

## Description

This function checks if a RINGHOM is surjective.

example

```
/**/ QQxyz ::= QQ[x,y,z];
/**/ QQab  ::= QQ[a,b];

/**/ use QQab;
/**/ phi := PolyAlgebraHom(QQxyz, QQab, [a+1, a*b+3, b^2]);
ideal(-x^2*z +y^2 +2*x*z -6*y -z +9)
/**/ IsSurjective(phi);
false

/**/ preimage0(phi, b);
0
```

**See Also:** ker(I-11.1 pg.177), IsInjective(I-9.56 pg.158), preimage0(I-16.18 pg.239)

## I-9.85 IsSymmetric

syntax

```
IsSymmetric(M: MAT): BOOL
```

## Description

This function tests whether the square matrix “M” is symmetric.

example

```
/**/ M := mat([[1, 2, 3], [2, 4, 5], [3, 5, 6]]);
/**/ IsSymmetric(M);
true
```

**See Also:** IsAntiSymmetric(I-9.38 pg.151)

## I-9.86 IsTerm

syntax

```
IsTerm(X: RINGELEM|MODULEELEM): BOOL
```

### Description

The function determines whether X is a term. For a polynomial, a “*term*” is a power-product, namely, a product of indeterminates. Thus,  $x * y^2 * z$  is a term, while  $4 * x * y^2 * z$  and  $x * y + z^3$  are not. For a vector, a term is a power-product times a standard basis vector, for instance  $(0, x * y^2 * z, 0)$ .

example

```
/**/ use R ::= QQ[x,y,z];
/**/ IsTerm(x+y^2);
false

/**/ IsTerm(x^3*y*z^2);
true

/**/ IsTerm(5*x^3*y*z^2);
false

/**/ R2 := NewFreeModule(R,2);
--/**/ IsTerm(ModuleElem(R2, [0,x*z])); --***WORK IN PROGRESS***
--true

--/**/ IsTerm(ModuleElem(R2, [x,y])); --***WORK IN PROGRESS***
--false
```

## I-9.87 IsTermOrdering

syntax

```
IsTermOrdering(M: MAT): BOOL
```

### Description

This function determines whether a square matrix defines a term-ordering, i.e. if its determinant is non-zero and if for each column the first nonnegative entry is positive.

example

```
/**/ IsTermOrdering(LexMat(5));
true

/**/ IsTermOrdering(StdDegRevLexMat(5));
true

/**/ IsTermOrdering(RevLexMat(5));
false
```

**See Also:** [NewPolyRing\(I-14.8 pg.212\)](#), [OrdMat\(I-15.10 pg.231\)](#)

## I-9.88 IsTree5

syntax

```
IsTree5(L: LIST): [BOOL, LIST ]
IsTree5(L: LIST, "NOOPT"): [BOOL, LIST]
```

```
IsTree5(L: LIST, "OPT"): [BOOL, LIST]
IsTree5(L: LIST, "CS_NOOPT"): [BOOL, LIST]
IsTree5(L: LIST, "CS_OPT"): [BOOL, LIST]
```

## Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This function is implemented in CoCoALib.

This function tests whether the facet complex described by the list L of square free power products is a tree, plus a list which:

- is empty if L is a tree
- contains three elements of a cycle of L if L is not a tree.

Four options “NOOPT”, “OPT”, “CS\_NOOPT”, “CS\_OPT” are available as second argument, specifying different algorithms; the default is “CS\_OPT”.

For a full description of the algorithms we refer to the paper by M. Caboara, S. Faridi, and P. Selinger, “Simplicial cycles and the computation of simplicial trees”, Journal of Symbolic Computation, vol.42/1-2, pp.77-88 (2006).

example

```
use R := QQ[x,y,z,t];
D := [x*y, y*z, z*t, t*x];
IsTree5(D);
[False, [xy, xt, yt]]
-----
IsTree5([xy, yz, zt]);
[True, [ ]]
```

## I-9.89 IsTrueGCDDomain

syntax

```
IsTrueGCDDomain(R: RING): BOOL
```

## Description

This function tests whether a ring is a (true) GCD domain but not a field. CoCoA can compute GCDs of elements of a true GCD domain.

example

```
/**/ IsTrueGCDDomain(ZZ);
true
/**/ IsTrueGCDDomain(QQ);
false
```

**See Also:** IsField(I-9.49 pg.155)

## I-9.90 IsZero

syntax

```
IsZero(X: OBJECT): BOOL
```

## Description

This function tests whether its argument is zero; the argument can be of almost any type for which “zero” makes sense.

example

```

/**/ IsZero(23);
false
/**/ IsZero(3-3);
true
/**/ use R ::= QQ[x,y,z];
/**/ IsZero(x^2+3*y-1);
false
/**/ IsZero(ideal(x^2,x*y^3));
false
/**/ F := NewFreeModule(R, 3);
/**/ zero(F);
[0, 0, 0]
/**/ IsZero(zero(F));
true
/**/ IsZero(matrix([[0,0,0], [0,0,0]]));
true

```

**See Also:** IsEven, IsOdd(I-9.47 pg.154), IsOne(I-9.67 pg.162), zero(I-25.1 pg.337), ZeroMat(I-25.2 pg.337)

## I-9.91 IsZeroCol, IsZeroRow

syntax

```

IsZeroCol(M: MAT, N: INT): BOOL
IsZeroRow(M: MAT, N: INT): BOOL

```

## Description

This function tests whether all entries in the “N”-th column(row) of “M” are zero.

example

```

/**/ IsZeroRow(matrix([[1,0,0], [0,0,0]]), 1);
false
/**/ IsZeroCol(matrix([[1,0,0], [0,0,0]]), 2);
true

```

**See Also:** IsZero(I-9.90 pg.170), ZeroMat(I-25.2 pg.337)

## I-9.92 IsZeroDim

syntax

```

IsZeroDim(I: IDEAL): BOOL

```

## Description

This function tests whether its argument is zero-dimensional.

example

```

/**/ use QQ[x,y,z];
/**/ IsZeroDim(ideal(x));
false

```

```

/**/ IsZeroDim(ideal(x^3, y^4-x ,z-3));
true
/**/ IsZeroDim(ideal(x^2, x*y^3));
false

```

**See Also:** [dim\(I-4.17 pg.80\)](#)

## I-9.93 IsZeroDivisor

— syntax —

```
IsZeroDivisor(X: RINGELEM): BOOL
```

### Description

This function tests whether its argument is a zero-divisor.

— example —

```

/**/ use P := QQ[x,y,z];
/**/ R := NewQuotientRing(P, ideal(x*y));
/**/ IsZeroDivisor(RingElem(R,x));
true
/**/ colon(ideal(zero(R)), ideal(RingElem(R,x)));
ideal((y))

```

**See Also:** [colon\(I-3.30 pg.57\)](#)

## I-9.94 IsZZ

— syntax —

```
IsZZ(R: RING): BOOL
```

### Description

This function tests whether a ring is the ring of integers.

— example —

```

/**/ R := QQ[x,y];
/**/ IsZZ(CoeffRing(R));
false
/**/ IsZZ(BaseRing(CoeffRing(R)));
true

```

**See Also:** [ZZ\(I-25.4 pg.338\)](#), [RingZZ\(I-18.51 pg.277\)](#), [IsQQ\(I-9.74 pg.164\)](#)

## I-9.95 It

— syntax —

```
It
```

### Description

“It” is a top-level SYSTEM VARIABLE containing the last result computed but not assigned. It is the CoCoA equivalent to GAP’s “last”.

When CoCoA evaluates a “*standalone expression*”, the result is assigned to the system variable named “It” (and then printed as if in a “println” ([I-16.34 pg.247](#)) command). You may use “It” in expressions just like any other variable.

example

```
/**/ 1+1; -- standalone expression ==> result is saved in "It".
2
/**/ It;
2

/**/ It+1;
3
/**/ It;
3

/**/ X := 17; -- assignment is not a standalone expression, "It" is unchanged
/**/ It;
3
/**/ X+It;
20
```

**See Also:** [print\(I-16.29 pg.245\)](#), [println\(I-16.34 pg.247\)](#), [Evaluation and Assignment\(II-4 pg.349\)](#)



# Chapter I-10

## J

### I-10.1 jacobian

syntax

```
jacobian(L: LIST of RINGELEM): MAT
```

#### Description

This function returns the Jacobian matrix of the polynomials in “L” with respect to all the indeterminates of the current ring.

example

```
/**/ use R ::= QQ[x,y];
/**/ L := [x-y, x^2-y, x^3-y^2];
/**/ jacobian(L);
matrix( /*RingDistrMPolyClean(QQ, 2)*/
  [[1, -1],
   [2*x, -1],
   [3*x^2, -2*y]])
```

### I-10.2 JanetBasis

syntax

```
JanetBasis(I: IDEAL): LIST of RINGELEM
```

#### Description

Thanks to Mario Albert.

This function returns the Janet basis of an ideal.

example

```
/**/ use R ::= QQ[x,y,z];
/**/ L := [x-y, x^2-z+1, x^3-y^2];
/**/ JanetBasis(ideal(L));
[x -y, z^2 -3*z +2, y*z -y -z +1, y^2 -z +1]
```

**See Also:** GBasis([I-7.1](#) pg.107)



# Chapter I-11

## K

### I-11.1 ker

syntax

```
ker(phi: RINGHOM): IDEAL
```

#### Description

This function returns the kernel of a homomorphism.

example

```
/**/ R := QQ[x,y,z,w];
/**/ Use S := QQ[s,t];
/**/ phi := PolyAlgebraHom(R, S, [s^3, s^2*t, s*t^2, t^3]);
/**/ ker(phi);
ideal(z^2 -y*t, y*z -x*t, y^2 -x*z)

/**/ SmodJ := NewQuotientRing(S, ideal(RingElem(S,"t+s")));
/**/ use SmodJ;
/**/ psi := PolyAlgebraHom(R, SmodJ, [s^3, s^2*t, s*t^2, t^3]);
/**/ ker(psi);
ideal(x +w, y -w, z +w)

/**/ RmodI := NewQuotientRing(R, ideal(RingElem(R,"x+y")));
/**/ ker(InducedHom(RmodI, psi));
ideal((-y +w), (y -w), (z +w))
```

**See Also:** [preimage0\(I-16.18 pg.239\)](#), [IsInjective\(I-9.56 pg.158\)](#), [IsSurjective\(I-9.84 pg.168\)](#)



# Chapter I-12

## L

### I-12.1 LaguerrePoly

— syntax —

```
LaguerrePoly(N: INT, X: RINGELEM): RINGELEM
```

#### Description

The function “LaguerrePoly” returns the “N”-th Laguerre polynomial multiplied by “factorial(N)” (so that the coefficients are integers).

This function also works if “X” is not an indeterminate: the result is then the evaluation of the polynomial at the given value.

— example —

```
/**/ use R := QQ[x];
/**/ LaguerrePoly(3,x);
-x^3 +9*x^2 -18*x +6
```

**See Also:** ChebyshevPoly([I-3.12](#) pg.50), HermitePoly([I-8.3](#) pg.122)

### I-12.2 last

— syntax —

```
last(L: LIST): OBJECT
last(L: LIST, N: INT): OBJECT
```

#### Description

In the first form, the function returns the last element of L. In the second form, it returns the list of the last N elements of L.

The CoCoA equivalent to GAP “last” is the variable “It” ([I-9.95](#) pg.172).

— example —

```
/**/ L := [1,2,3,4,5];
/**/ last(L);
5

/**/ last(L,3);
[3, 4, 5]
```

**See Also:** first([I-6.8](#) pg.96), tail([I-20.3](#) pg.312), It([I-9.95](#) pg.172)

## I-12.3 latex

— syntax —

```
latex(X: OBJECT): STRING
```

### Description

This function returns a string containing the argument formatted in LaTeX. It can also be called with the name “LaTeX”.

For typesetting ideals this function assumes use of a LaTeX macro like this:

```
\newcommand{\ideal}[1]{\langle #1 \rangle}
```

— example —

```
/**/ use R := QQ[x,y,z];
/**/ F := x^10 +2*y^12*z^3;
/**/ latex(F);
2 y^{12} z^3 +x^{10}

/**/ M := mat([[1,2],[3,4]]);
/**/ latex(M);
\left( \begin{array}{cc}
1 & 2 \\
3 & 4 \end{array} \right)

/**/ R := QQ[x,y,z];
/**/ latex(ideal(x^2, y+z));
\ideal{x^2 ,
y +z}

/**/ P := NewFractionField(R);
/**/ use P;
/**/ F := (x+y)/(1-z)^3;
/**/ latex(F);
\frac{-x -y}{z^3 -3 z^2 +3 z -1}
```

**See Also:** [format\(I-6.19 pg.102\)](#), [sprint\(I-19.29 pg.294\)](#)

## I-12.4 LC

— syntax —

```
LC(F: RINGELEM|MODULEELEM): RINGELEM
```

### Description

This function returns the leading coefficient of F, as determined by the term-ordering of the ring to which F belongs.

— example —

```
/**/ use R := QQ[x,y];
/**/ LC(x +3*x^2 -5*y^2);
3

/**/ F := NewFreeModule(R,3);
/**/ LC(ModuleElem(F, [0, 5*y+6*x^2, y^2]));
6
```

**See Also:** [coefficients\(I-3.24 pg.54\)](#), [CoeffOfTerm\(I-3.27 pg.56\)](#), [LT\(I-12.21 pg.188\)](#)

## I-12.5 lcm

syntax

```
lcm(N: INT, M: INT): INT
lcm(L: LIST of INT): INT

lcm(F: RINGELEM, G: RINGELEM): RINGELEM
lcm(L: LIST of RINGELEM): RINGELEM
```

### Description

This function returns the least common multiple of “F<sub>1</sub>, ..., F<sub>n</sub>” or of the elements in the list L. For the calculation of the GCDs and LCMs of polynomials, the coefficient ring must be a field.

example

```
/**/ use R ::= QQ[x,y];
/**/ F := x^2-y^2;
/**/ G := (x+y)^3;
/**/ lcm(F, G);
-x^4 -2*x^3*y +2*x*y^3 +y^4

/**/ IsDivisible(F*G, It);
true

/**/ lcm(F, G) * gcd(F,G) = F*G;
true

/**/ lcm([3*4,3*8,6*16]);
96
```

**See Also:** [div\(I-4.20 pg.81\)](#), [mod\(I-13.26 pg.202\)](#), [gcd\(I-7.4 pg.108\)](#)

## I-12.6 len

syntax

```
len(E: STRING|LIST): INT
```

### Description

This function returns the “*length*” of an object, as summarized in the table below:

-----	
type	length
-----	
STRING	number of bytes in the string
LIST	number of items in the list
-----	

The function ‘‘\verb&len&’’

example

```
/**/ len( [2,3,4] );
3
```

```
/**/ len( "string" );
6
```

Previously “len” could be applied to other types too; this is no longer supported. See “NumCompts” (I-14.35 pg.223) for module elements, “NumRows” (I-14.39 pg.225) for matrices, and “NumTerms” (I-14.40 pg.225) for polynomials.

**See Also:** count(I-3.52 pg.66), NumCompts(I-14.35 pg.223), NumRows(I-14.39 pg.225), NumTerms(I-14.40 pg.225)

## I-12.7 LexMat

syntax

```
LexMat(N: INT): MAT
```

### Description

This function return the matrix defining the standard term-ordering ”lex”.

example

```
/**/ LexMat(3);
matrix(ZZ,
  [[1, 0, 0],
   [0, 1, 0],
   [0, 0, 1]
])
```

**See Also:** Term Orderings(III-9.5 pg.404), StdDegLexMat(I-19.37 pg.299), StdDegRevLexMat(I-19.38 pg.299), RevLexMat(I-18.41 pg.272), XelMat(I-24.1 pg.335)

## I-12.8 LexSegmentIdeal

syntax

```
LexSegmentIdeal(L: LIST of power-products): IDEAL
LexSegmentIdeal(I: IDEAL): IDEAL
```

### Description

If the argument is a LIST of power-products “L”, this function returns the smallest lex-segment ideal containing the power-products in “L”.

If it is an IDEAL “I”, it returns the lex-segment ideal having the same Hilbert function as “I”.

example

```
/**/ use R := QQ[x,y,z];
/**/ LexSegmentIdeal([y^3]);
ideal(y^3, x*z^2, x*y*z, x*y^2, x^2*z, x^2*y, x^3)
/**/ LexSegmentIdeal(ideal(y^3));
ideal(x^3)
```

**See Also:** IsLexSegment(I-9.63 pg.160), StableIdeal(I-19.32 pg.296), StronglyStableIdeal(I-19.39 pg.300)

## I-12.9 LF

— syntax —

```
LF(I: IDEAL): IDEAL
LF(F: RINGELEM): RINGELEM
```

### Description

For a polynomial “F” this function returns the leading form, i.e. the sum of all summands having highest degree. It throws an error if the argument is zero or if the “GradingDim” (I-7.29 pg.117) of the polynomial ring is 0 (use “DF” (I-4.14 pg.78) to allow these cases).

For an ideal “I” this function returns the ideal of all the “LF(f)” for “f in I”. It throws an error if the “GradingDim” (I-7.29 pg.117) of the polynomial ring is 0.

— example —

```
/**/ use R := QQ[x,y];
/**/ LF(x^2 -x*y +2*x -1);
x^2 -x*y

/**/ use R := QQ[x,y], Lex; -- GradingDim is 0: everything is homogeneous
-- /**/ LF(x-1); --> !!! ERROR !!! as expected: instead of x-1

/**/ P := NewPolyRing(QQ, IndetSymbols(R), mat([[1,4],[1,0]]), 1);
/**/ Use P;
/**/ LF(x^2 - x*y);
-x*y
/**/ LF(x^4 + x^2 - y);
x^4 -y
```

**See Also:** DF(I-4.14 pg.78), IsHomog(I-9.52 pg.156), LC(I-12.4 pg.180), LM(I-12.16 pg.186), LPP(I-12.20 pg.188), LT(I-12.21 pg.188)

## I-12.10 LinearSimplify

— syntax —

```
LinearSimplify(F: RINGELEM): RECORD
```

### Description

This function returns a “RECORD[LinearChange, SimplePoly]” where “LinearChange” is a linear change of variable and “SimplePoly” is simple (in a heuristic sense). The composition “SimplePoly(LinearChange)” is equal the univariate polynomial “F”.

— example —

```
/**/ use QQ[x];
/**/ LinearSimplify((123*x-456)^9-1);
record[LinearChange := 123*x - 456, SimplePoly := x^9 - 1]

/**/ LinearSimplify(x^9-1); -- the heuristic finds no useful simplification
record[LinearChange := x, SimplePoly := x^9 - 1]
```

## I-12.11 LinKer

— syntax —

```
LinKer(M: MAT): MAT
```

## Description

This function returns a matrix whose columns represent a basis for the kernel of “M”. Calling the function twice on the same input will not necessarily produce the same output, though in each case, a basis for the kernel is produced.

This function works only on matrices whose entries are in a field (from version CoCoA-5.0.3). The CoCoA-4 function returning a ZZ-basis for the kernel of “M” is not yet implemented.

The output as it was given by CoCoA-4 (the basis of the ker) is now given by “LinKerBasis” (I-12.12 pg.184). See also “HilbertBasisKer” (I-8.6 pg.123).

### example

```

/**/ M := mat([[1,2,3,4],[5,6,7,8],[9,10,11,12]]);
/**/ LinKer(M);
matrix(QQ,
  [[-1, -2],
   [2, 3],
   [-1, 0],
   [0, -1]])

/**/ M*It;
matrix(QQ,
  [[0, 0],
   [0, 0],
   [0, 0]])

```

**See Also:** LinKerBasis(I-12.12 pg.184), LinSolve(I-12.15 pg.185), HilbertBasisKer(I-8.6 pg.123)

## I-12.12 LinKerBasis

### syntax

```

LinKerBasis(M: MAT): LIST of RINGELEM
LinKerBasis(L: LIST): LIST of RINGELEM

```

## Description

This function returns a list whose components are lists representing a basis for the kernel of the matrix “M” or for the solutions of the linear system given by linear polynomials “L”. Calling the function twice on the same input will not necessarily produce the same output, though in each case, a basis for the kernel is produced.

This function works only on matrices whose entries are in a field (from version CoCoA-5.0.3). The CoCoA-4 function returning a ZZ-basis for the kernel of “M” is not yet implemented.

### example

```

/**/ use QQ[x,y,z];
/**/ L := [x+y+z, y-z];
/**/ LinKerBasis(L);
[[-2, 1, 1]]

/**/ M := mat([[1,2,3,4],[5,6,7,8],[9,10,11,12]]);
/**/ LinKerBasis(M);
[[-1, 2, -1, 0], [-2, 3, 0, -1]]

/**/ K := NewFractionField(NewPolyRing(QQ, "a,b"));
/**/ use K;
/**/ M := mat([[1,2,3,a],[5,6,7,a*b]]);
/**/ LinKerBasis(M);
[[-1, 2, -1, 0], [(a*b - 3*a)/2, (-a*b + 5*a)/4, 0, -1]]

```

**See Also:** LinKer(I-12.11 pg.183), LinSolve(I-12.15 pg.185)

## I-12.13 LinKerModP [OBSOLETE]

— syntax —

[OBSOLETE]

### Description

[OBSOLETE] In CoCoA-4 it was difficult to map a matrix into “ZZ/(p)”. Now, in CoCoA-5, we can map the matrix and then call directly “LinKer” (I-12.11 pg.183) and “LinKerBasis” (I-12.12 pg.184).

— example —

```

/**/ use ZZ/(7);
/**/ M := mat([[1,2,3,4],[5,6,7,8],[9,10,11,12]]); --> by default over QQ
/**/ LinKerBasis(M);
[[-1, 2, -1, 0], [-2, 3, 0, -1]]

/**/ LinKerBasis(matrix(NewRingFp(3), M)); --> map M into ZZ/(3)
[[-1, -1, -1, 0], [1, 0, 0, -1]]

/**/ LinKer(matrix(CurrentRing, M)); --> map M into CurrentRing ZZ/(7)
matrix( /*RingWithID(9, "FFp(7)")*/
  [[-1, -2],
   [2, 3],
   [-1, 0],
   [0, -1]])
/**/ matrix(CurrentRing, M) * It;
matrix( /*RingWithID(9, "FFp(7)")*/
  [[0, 0],
   [0, 0],
   [0, 0]]

```

**See Also:** LinKer(I-12.11 pg.183), LinKerBasis(I-12.12 pg.184), LinSolve(I-12.15 pg.185)

## I-12.14 LinSol [OBSOLETE]

— syntax —

[OBSOLETE] use LinSolve

### Description

[OBSOLETE] Replaced by “LinSolve” (I-12.15 pg.185) **See Also:** LinSolve(I-12.15 pg.185)

## I-12.15 LinSolve

— syntax —

LinSolve(M: MAT, RHS: MAT): MAT

## Description

This function finds a solution “X” to the matrix equation “ $M \cdot X = \text{RHS}$ ”. If more than one solution exists, it returns just one of them. If no solution exists then it produces a 0-by-0 matrix. To find all solutions, compute the kernel of “M” using the function “LinKer” (I-12.11 pg.183).

NOTE: an easy way of converting a list into a column matrix (for the second argument) is to use the function “ColMat” (I-3.29 pg.57).

example

```

/**/ M := mat([[3,1,4],[1,5,9],[2,6,5]]);
/**/ L := [123,456,789];
/**/ LinSolve(M, ColMat(L));
mat([
  [199/5],
  [742/5],
  [-181/5]
])

/**/ M*It;
mat([
  [123],
  [456],
  [789]
])
-----

```

**See Also:** ColMat(I-3.29 pg.57), LinKer(I-12.11 pg.183)

## I-12.16 LM

syntax

```

LM(X: RINGELEM): RINGELEM
LM(X: MODULEELEM): MODULEELEM

```

## Description

This function returns the leading monomial of “X”. The monomial includes the coefficient. To get the leading term of “P”, (which does not include the coefficient), use “LT” (I-12.21 pg.188).

example

```

/**/ use R := QQ[x,y];
/**/ LM(3*x^2*y + y);
3*x^2*y

```

**See Also:** LC(I-12.4 pg.180), LF(I-12.9 pg.183), LPP(I-12.20 pg.188), LT(I-12.21 pg.188)

## I-12.17 log [OBSOLESCENT]

syntax

```

[OBSOLESCENT]

```

## Description

Renamed to “exponents” (I-5.16 pg.90).

## I-12.18 LogCardinality

— syntax —

```
LogCardinality(Fp: RING): INT
```

### Description

This function returns the extension degree of a finite field over its prime field, or equivalently the log (base p) of its cardinality.

— example —

```
/**/ Fp := ZZ/(7);
/**/ use Fpx := Fp[x];
/**/ Fq := Fpx/ideal(x^2+1);
/**/ LogCardinality(Fq);
2
```

**See Also:** IsFiniteField([I-9.50](#) pg.155), characteristic([I-3.10](#) pg.49)

## I-12.19 LPosn

— syntax —

```
LPosn(V: MODULEELEM): INT
```

### Description

This function returns the position of the leading power-product of “V”.

This function used to be called “LPos” up to version 5.0.3.

— example —

```
/**/ use R := QQ[x,y,z]; -- the default term-ordering is DegRevLex
/**/ R4 := NewFreeModule(R,4); -- the default module ordering is TOPos
/**/ LPosn(ModuleElem(R4, [0, x, y^2, x^2]));
4
/**/ LPP(ModuleElem(R4, [0, x, y^2, x^2]));
x^2
/**/ LT(ModuleElem(R4, [0, x, y^2, x^2]));
[0, 0, 0, x^2]

use R := QQ[x,y], PosTo;
LT(Vector(x,y^2));
Vector(x, 0)
-----
LPP(Vector(x,y^2));
x
-----
LPosn(Vector(x,y^2));
1
-----
```

**See Also:** LF([I-12.9](#) pg.183), LM([I-12.16](#) pg.186), LPP([I-12.20](#) pg.188), LT([I-12.21](#) pg.188)

## I-12.20 LPP

syntax

```
LPP(X: RINGELEM): RINGELEM
LPP(X: MODULEELEM): RINGELEM
```

### Description

This function returns the leading power-product of “X”; it discards information about which component the power-product appears in.

example

```
/**/ use R := QQ[x,y];
/**/ LPP(3*x^2*y+y); -- LPP is the same as LT for polynomials
x^2*y

-- Note the difference between LPP and LT for MODULEELEM.
/**/ R4 := NewFreeModule(R,4); -- the default module ordering is TOPos
/**/ LPP(ModuleElem(R4, [0, x, y^2, x^2]));
x^2
/**/ LT(ModuleElem(R4, [0, x, y^2, x^2]));
[0, 0, 0, x^2]
/**/ LPosn(ModuleElem(R4, [0, x, y^2, x^2]));
4
```

**See Also:** LC(I-12.4 pg.180), LF(I-12.9 pg.183), LM(I-12.16 pg.186), LPosn(I-12.19 pg.187), LT(I-12.21 pg.188)

## I-12.21 LT

syntax

```
LT(I: RINGELEM): RINGELEM
LT(I: IDEAL): IDEAL
LT(I: MODULEELEM): MODULEELEM
LT(I: MODULE): MODULE
```

### Description

If “E” is a polynomial this function returns the leading term of the polynomial “E” with respect to the term-ordering of the polynomial ring of “E”.

For the leading monomial, which includes the coefficient, use “LM” (I-12.16 pg.186).

example

```
/**/ use R := QQ[x,y,z]; -- the default term-ordering is DegRevLex
/**/ LT(y^2-x*z);
y^2

/**/ use R := QQ[x,y,z], Lex;
/**/ LT(y^2-x*z);
x*z
```

If “E” is a MODULEELEM, “LT(E)” gives the leading term of “E” with respect to the module term-ordering of “E”. For the leading monomial, which includes the coefficient, use “LM” (I-12.16 pg.186).

example

```
/**/ use R := QQ[x,y];
/**/ R3 := NewFreeModule(R,3);
```

```

/**/ LT(ModuleElem(R3, [0, x, y^2]));
[0, 0, y^2]

```

If “E” is an ideal or module, “LT(E)” returns the ideal or module generated by the leading terms of all elements of E, sometimes called the “*initial*” ideal or module.

example

```

/**/ use R := QQ[x,y,z];
/**/ I := ideal(x-y, x-z^2);
/**/ LT(I);
ideal(x, z^2)

```

**See Also:** LC(I-12.4 pg.180), LF(I-12.9 pg.183), LM(I-12.16 pg.186), LPP(I-12.20 pg.188), Module Orderings(III-9.6 pg.405), Term Orderings(III-9.5 pg.404)



# Chapter I-13

## M

### I-13.1 MakeCheck

\_\_\_\_\_ syntax \_\_\_\_\_

```
MakeCheck()
```

#### Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This function run a series of tests on the whole system. To get a reliable result you should run this on a “*just opened*” CoCoA because some printouts may mysteriously add some empty spaces which will result in an, apparent, failure of some tests.

\_\_\_\_\_ example \_\_\_\_\_

```
MakeCheck();
```

### I-13.2 MakeMatByRows, MakeMatByCols

\_\_\_\_\_ syntax \_\_\_\_\_

```
MakeMatByRows(R: INT, C: INT, L: LIST): MAT  
MakeMatByCols(R: INT, C: INT, L: LIST): MAT
```

#### Description

These functions create an “ $R \times C$ ” matrix from the list “ $L$ ”. The first argument “ $R$ ” is the number of rows, and the second “ $C$ ” is the number of columns. It is an error if the length of “ $L$ ” is not “ $R \times C$ ”.

The ring of the matrix is determined from the ring containing the elements of “ $L$ ”. If “ $L$ ” contains only integers/rationals then the matrix is over “ $QQ$ ”.

\_\_\_\_\_ example \_\_\_\_\_

```
/**/ MakeMatByRows(2, 10, 1..20);  
matrix(QQ,  
  [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10],  
   [11, 12, 13, 14, 15, 16, 17, 18, 19, 20]])  
  
/**/ MakeMatByCols(2, 10, 1..20);  
matrix(QQ,  
  [[1, 3, 5, 7, 9, 11, 13, 15, 17, 19],  
   [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]])
```

**See Also:** BlockMat([I-2.9 pg.41](#)), matrix([I-13.10 pg.195](#)), NewMat([I-14.6 pg.211](#)), ColMat([I-3.29 pg.57](#)), RowMat([I-18.56 pg.279](#)), DiagMat([I-4.15 pg.79](#)), ConcatHor([I-3.41 pg.61](#)), ConcatVer([I-3.44 pg.63](#))

### I-13.3 MakeSet

syntax

```
MakeSet(L: LIST): LIST
```

#### Description

This function returns a list obtained by removing duplicates from “L”.

example

```
/**/ MakeSet([2,2,2,1,2,1,1,3,3]);
[2, 1, 3]
```

NOTE: to test two sets for equality use the function “EqSet” ([I-5.8 pg.86](#)) instead of a normal equality test (because the latter yields false if the elements are in a different order).

**See Also:** EqSet([I-5.8 pg.86](#)), intersection([I-9.32 pg.149](#)), IntersectList([?? pg.??](#)), remove([I-18.32 pg.268](#))

### I-13.4 MakeTerm

syntax

```
MakeTerm(R: RING, L: LIST of INT): RINGELEM
```

#### Description

This function returns the power-product in “R” whose list of exponents is “L”. It is the inverse of “exponents” ([I-5.16 pg.90](#)). The length of “L” must be equal to the number of indeterminates in “R”.

This function was called “LogToTerm” up to version CoCoA-5.1.2.

example

```
/**/ use R := QQ[x,y,z];
/**/ MakeTerm(R, [2,3,5]);
x^2*y^3*z^5

/**/ exponents(It);
[2, 3, 5]
```

**See Also:** exponents([I-5.16 pg.90](#))

### I-13.5 MakeTermOrd

syntax

```
MakeTermOrd(M: MAT): MAT
MakeTermOrd(M: MAT, GrDim: INT): MAT
```

#### Description

This function returns a (square) matrix of integers defining a term ordering; the output matrix is built from “M”. The first “GrDim” rows are left unchanged; if “GrDim” is not given then it is taken to be 0.

The input matrix “M” must have rational entries; the first “GrDim” rows must have non-negative integer entries, and they must be linearly independent.

If “M” is not square then rows are appended or eliminated as necessary.

NOTE: this function was called “CompleteToOrd” up to version CoCoA-5.1.2.

#### example

```

/**/ M := matrix([[1,2,3,4]]);
/**/ MakeTermOrd(M);
[[1, 2, 3, 4],
 [0, 0, 0, -1],
 [0, 0, -1, 0],
 [0, -1, 0, 0]]

/**/ MakeTermOrd(RowMat([1,2,0,0]));
[[1, 2, 0, 0],
 [0, 0, 1, 1],
 [0, 0, 0, -1],
 [0, -1, 0, 0]]

/**/ MakeTermOrd(matrix([[1,2,0,0],[0,0,3,4]]),2);
matrix(ZZ,
[[1, 2, 0, 0],
 [0, 0, 3, 4],
 [0, 0, 0, -1],
 [0, -1, 0, 0]])

```

**See Also:** LexMat(I-12.7 pg.182), RevLexMat(I-18.41 pg.272), StdDegLexMat(I-19.37 pg.299), StdDegRevLexMat(I-19.38 pg.299), NewPolyRing(I-14.8 pg.212)

## I-13.6 MantissaAndExponent10

#### syntax

```
MantissaAndExponent10(X: INT|RAT, Prec: INT): RECORD
```

### Description

This function converts a rational number into a “RECORD” with components named “exponent”, “mantissa” and “NumDigits”.

If “X=0”, all fields of the record are set to zero.

For non-zero “X” the fields give the best representation of the form  $M * 10^E$  where “M” has “Prec” decimal digits. The value of “NumDigits” is simply “Prec”. The value of “exponent” is “FloorLog10(X)”, plus 1 if the mantissa “overflows”. The value of “mantissa” is an integer “M” satisfying  $10^{(Prec-1)} \leq |M| < 10^{Prec}$ .

#### example

```

/**/ MantissaAndExponent10(1/2,3);      -- 1/2 = 5.00*10^(-1)
record[NumDigits := 3, exponent := -1, mantissa := 500]

/**/ MantissaAndExponent10(0.99999, 4); -- 0.99999 rounds up to give 1.000
record[NumDigits := 4, exponent := 0, mantissa := 1000]

```

**See Also:** AsINT(I-1.19 pg.35), AsRAT(I-1.20 pg.36), DecimalStr(I-4.3 pg.71), FloatApprox(I-6.12 pg.98), FloatStr(I-6.13 pg.98), FloorLog2, FloorLog10, FloorLogBase(I-6.15 pg.99), MantissaAndExponent2(I-13.7 pg.194), ScientificStr(I-19.3 pg.282)

## I-13.7 MantissaAndExponent2

— syntax —

```
MantissaAndExponent2(X: INT|RAT, Prec: INT): RECORD
MantissaAndExponent2(X: RINGELEM): RECORD
```

### Description

The first form of this function converts an integer or rational number into a “RECORD” with components named “exponent”, “mantissa” and “NumDigits”.

If “X=0”, all fields of the record are set to zero.

For non-zero “X” the fields give the best representation of the form  $M * 2^E$  where “M” has “Prec” bits. The value of “NumDigits” is simply “Prec”. The value of “exponent” is “FloorLog2(X,2)”, plus 1 if the mantissa “overflows”. The value of “mantissa” is an integer “M” satisfying  $2^{(Prec-1)} \leq |M| < 2^{Prec} - 1$

The second form of this function applies to elements of a “twin-float” ring. In this case the “precision” is determined directly from the twin-float value; since twin-float arithmetic is based on a randomized heuristic, repeating a computation may give a slightly different result (and this can be seen in the output of “MantissaAndExponent2”).

— example —

```
/**/ MantissaAndExponent2(1/2,8);      -- 1/2 = 128*2^(-8)
record[NumDigits := 8, exponent := -1, mantissa := 128]

/**/ MantissaAndExponent2(65535, 10);  -- rounds up
record[NumDigits := 10, exponent := 16, mantissa := 512]
```

**See Also:** AsINT(I-1.19 pg.35), AsRAT(I-1.20 pg.36), FloatApprox(I-6.12 pg.98), FloorLog2, FloorLog10, FloorLogBase(I-6.15 pg.99), MantissaAndExponent10(I-13.6 pg.193), NewRingTwinFloat(I-14.11 pg.213)

## I-13.8 Manual

— syntax —

```
? key
?? key
```

### Description

These operators are used to search the online help system for information matching a keyword (introduced in CoCoA 4.2).

The commands have the form “?key” and “??key” where “key” is a literal string without quotes. The search for the “key” is case insensitive and ignores blank space before or after “key”. Also, the semicolon usually required at the end of a line of CoCoA input is optional.

The search system is fairly simple. The searching algorithm looks through the title and keywords of each manual page. A page matches if “key” appears as a (case-insensitive) substring of the title/keywords.

The “??” form prints the list of all matches. The “?” form prints the page matching exactly if there is one, otherwise it prints the list of all matches.

— example —

```
/**/ ?approxs
-----[ ApproxSolve ]-----
--> ApproxSolve(L: LIST of RINGELEM): LIST of LIST of RAT
```

This function returns the list of real solutions (points) of a

```

... ..
/**/ ?approx
-----< No exact match for "approx" >-----
All 9 matches for "approx":
? ApproxPointsNBM
? ApproxSolve
... ..

```

## I-13.9 MapDown [OBSOLETE]

syntax

[OBSOLETE]

### Description

[OBSOLETE] See “AsINT” (I-1.19 pg.35), “AsRAT” (I-1.20 pg.36).

## I-13.10 matrix

syntax

```

matrix(L: LIST): MAT
matrix(R: RING, L: LIST): MAT
matrix(R: RING, M: MAT): MAT

```

### Description

This function returns a matrix in the ring “R”.

When the input is “L”, a “*rectangular*” LIST of LIST of RINGELEM all in “R” (or INT, or RAT). When the ring is not specified it “guesses” the right ring; if all elements are INT or RAT the resulting matrix is in QQ.

The third form is equivalent to “`apply(CanonicalHom(RingOf(M),R), M)`” (See “`apply`” (I-1.13 pg.32), “`CanonicalHom`” (I-3.3 pg.46)).

example

```

/**/ L := [[1,2],[3,4]];
/**/ mat(L);
matrix(QQ,
  [[1, 2],
   [3, 4]])
/**/ mat(ZZ,L);
matrix(ZZ,
  [[1, 2],
   [3, 4]])

/**/ P := QQ[x,y];
/**/ M := mat(P,L); print M;
matrix( /*RingWithID(3, "QQ[x,y]")*/
  [[1, 2],
   [3, 4]])
/**/ RingOf(M);
RingWithID(3, "QQ[x,y]")

```

```

/**/ M := IdentityMat(ZZ,2);
/**/ matrix(QQ, M);
matrix(QQ,
  [[1, 0],
   [0, 1]])

```

**See Also:** NewMat(I-14.6 pg.211), ColMat(I-3.29 pg.57), RowMat(I-18.56 pg.279), DiagMat(I-4.15 pg.79), MakeMatByRows, MakeMatByCols(I-13.2 pg.191), ConcatHor(I-3.41 pg.61), ConcatVer(I-3.44 pg.63), BlockMat(I-2.9 pg.41), Commands and Functions for MAT(III-13.2 pg.421)

## I-13.11 max

— syntax —

```

max(E_1: OBJECT, ..., E_n: OBJECT): OBJECT
max(L: LIST): OBJECT

```

### Description

In the first form, this function returns a maximum of  $E_1, \dots, E_n$ . In the second form, it returns a maximum of the objects in the list “L”.

— example —

```

/**/ max([1,2,3]);
3
/**/ max(1,2,3);
3

/**/ use R ::= QQ[x,y,z];
/**/ max(x^3*z, x^2*y^2); -- x^2y^2 > x^3z in the default ordering, DegRevLex
x^2*y^2
/**/ min(x^3*z, x^2*y^2);
x^3*z

/**/ use R ::= QQ[x,y,z], DegLex;
/**/ max(x^3*z, x^2*y^2); -- x^3z < x^2y^2 in DegLex
x^3*z

```

**See Also:** min(I-13.14 pg.197), Relational Operators(II-3.3 pg.348)

## I-13.12 MaxBy

— syntax —

```

MaxBy(L: LIST, LessThanFunc: FUNCTION)

```

### Description

This function return a maximum of the elements of the list in L with respect to the comparisons made by LessThanFunc.

The comparison function LessThanFunc takes two arguments and returns true if the first argument is less than the second, otherwise it returns false.

NOTE: to call MaxBy(L, LessThanFunc) inside a function you will need to make the name LessThanFunc accessible using TopLevel LessThanFunc;

NOTE: if both LessThanFunc(A, B) and LessThanFunc(B, A) return true, then A and B are viewed as being equal.

example

```

/**/ Define ByLength(S, T)  -- define the sorting function
/**/   Return len(S) < len(T);
/**/ EndDefine;

/**/ L := ["bird", "mouse", "cat", "elephant"];
/**/ MaxBy(L, ByLength);
elephant

/**/ use QQ[x,y];
/**/ Define ByLPP(S, T)  return LPP(S) < LPP(T);  EndDefine;
/**/ L := [x^5 -1, x^2*y -y -3];
/**/ MaxBy(L, ByLPP);
x^5 -1

```

**See Also:** MinBy([I-13.15](#) pg.198), func([I-6.26](#) pg.105), SortedBy([I-19.25](#) pg.293), TopLevel([I-20.10](#) pg.314)

## I-13.13 MayerVietorisTreeN1

syntax

```

MayerVietorisTreeN1(I: IDEAL): INT

```

### Description

Implemented in CoCoALib by Eduardo Saenz-de-Cabezón.

This function returns the list of multidegrees “M” such that the N-1st Betti number of a monomial ideal “I” at multidegree “M” is not zero. It is computed via a version of its Mayer-Vietoris tree.

The length of this list is the number of irreducible components of I, the number of maximal standard monomials, and the number of generators of its Alexander Dual.

example

```

/**/ use QQ[x,y,z];
/**/ I := ideal(x, y, z)^2;
/**/ MayerVietorisTreeN1(I);
[x^2*y*z, x*y^2*z, x*y*z^2]

```

**See Also:** Frobbby([II-9.4](#) pg.365)

## I-13.14 min

syntax

```

min(E_1: OBJECT,...,E_n: OBJECT): OBJECT
min(L: LIST): OBJECT

```

### Description

In the first form, this function returns a minimum of  $E_1, \dots, E_n$ . In the second form, it returns a minimum of the objects in the list “L”.

See more examples in “max” ([I-13.11](#) pg.196).

example

```
/**/ min([1,2,3]);
1
/**/ min(1,2,3);
1
```

**See Also:** [max\(I-13.11 pg.196\)](#), [Relational Operators\(II-3.3 pg.348\)](#)

## I-13.15 MinBy

syntax

```
MinBy(L: LIST, LessThanFunc: FUNCTION)
```

### Description

This function return a minimum of the elements of the list in L with respect to the comparisons made by LessThanFunc. (see “MaxBy” ([I-13.12 pg.196](#)) for details)

example

```
/**/ Define ByLength(S, T)    -- define the sorting function
/**/   Return len(S) < len(T);
/**/ EndDefine;

/**/ L := ["bird", "mouse", "cat", "elephant"];
/**/ MinBy(L, ByLength);
cat
```

**See Also:** [MaxBy\(I-13.12 pg.196\)](#), [func\(I-6.26 pg.105\)](#), [SortedBy\(I-19.25 pg.293\)](#), [TopLevel\(I-20.10 pg.314\)](#)

## I-13.16 MinGens

syntax

```
MinGens(M: IDEAL|MODULE): LIST
```

### Description

If “M” is a homogeneous ideal or module, this function returns a list of minimal generators for “M” (not necessarily a subset of “gens(M)”).

For non-homogeneous input use “MinSubsetOfGens” ([I-13.25 pg.202](#)).

NOTE: the coefficient ring must be a field.

example

```
/**/ use R := QQ[x,y,z];
/**/ I := ideal(x-y, (x-y)^4, z+y, (z+y)^2);
/**/ MinGens(I);
[y + z, x + z]

/**/ R3 := NewFreeModule(R, 3);
/**/ MGens := matrix(R, [[x,y,z], [x^2,0,z^2], [2*x^2,x*y,z^2+x*z]]);
/**/ M := SubmoduleRows(R3, MGens);
/**/ gens(M);
[[x, y, z], [x^2, 0, z^2], [2*x^2, x*y, x*z + z^2]]
```

```
/**/ MinGens(M);
[[x, y, z], [0, x*y, x*z -z^2]]
```

**See Also:** [IdealOfMinGens\(I-9.6 pg.135\)](#), [SubmoduleOfMinGens\(I-19.46 pg.302\)](#), [MinSubsetOfGens\(I-13.25 pg.202\)](#)

## I-13.17 MinGensGeneral [OBSOLESCENT]

syntax

[OBSOLESCENT]

### Description

Just renamed “[MinSubsetOfGens\(I-13.25 pg.202\)](#)” (more expressive).

**See Also:** [MinSubsetOfGens\(I-13.25 pg.202\)](#)

## I-13.18 minimize [OBSOLESCENT]

syntax

```
[OBSOLESCENT]
minimize(ref X: IDEAL)
minimize(ref X: MODULE)
```

### Description

[OBSOLESCENT] use “[IdealOfMinGens\(I-9.6 pg.135\)](#)” and “[SubmoduleOfMinGens\(I-19.46 pg.302\)](#)”.

**See Also:** [IdealOfMinGens\(I-9.6 pg.135\)](#), [SubmoduleOfMinGens\(I-19.46 pg.302\)](#), [MinGens\(I-13.16 pg.198\)](#), [MinSubsetOfGens\(I-13.25 pg.202\)](#)

## I-13.19 minimized [OBSOLESCENT]

syntax

[OBSOLESCENT]

### Description

Renamed “[IdealOfMinGens\(I-9.6 pg.135\)](#)” and “[SubmoduleOfMinGens\(I-19.46 pg.302\)](#)”.

**See Also:** [IdealOfMinGens\(I-9.6 pg.135\)](#), [SubmoduleOfMinGens\(I-19.46 pg.302\)](#)

## I-13.20 MinimalPresentation

syntax

```
MinimalPresentation(Q:TAGGED):TAGGED

where Q is a quotient module of the type R^s/M
```

## Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

Given a quotient module of the type  $R^s/M$ , or a zero module, this function computes an isomorphic quotient,  $R^t/N$ , minimally presented [using the algorithm in Kreuzer-Robbiano II].

example

```
use R := QQ[x,y,z];
MinimalPresentation(R^3/Module([[x,1,1], [x,2,2]]));
R^2/Module([[x, 0]])
-----
```

## I-13.21 minors

syntax

```
minors(M: MAT, N: INT): LIST
```

## Description

This function returns the list of all determinants of  $N \times N$  submatrices of  $M$ .

example

```
/**/ M := mat([[1,2,3], [-1,2,4]]);
/**/ minors(M, 2);
[4, 7, 2]
```

**See Also:** [det\(I-4.13 pg.78\)](#)

## I-13.22 MinPoly

syntax

```
MinPoly(M: MAT, X: RINGELEM): RINGELEM
```

## Description

Thanks to Maria-Laura Torrente.

This function returns the minimal polynomial of the matrix “M” in the indeterminate “X” (with “M” a square matrix whose entries lie in the coefficient ring of “X”, or in the same ring as “X” but not dependent on “X”). See also “CharPoly” ([I-3.11 pg.49](#)).

example

```
/**/ use R := QQ[x];
/**/ MinPoly(matrix([[0,0,1], [0,0,0], [0,0,0]]), x);
x^2
/**/ CharPoly(matrix([[0,0,1], [0,0,0], [0,0,0]]), x);
x^3
```

**See Also:** [CharPoly\(I-3.11 pg.49\)](#)

## I-13.23 MinPolyQuot, MinPolyQuotDef, MinPolyQuotElim, MinPolyQuotMat

syntax

```
MinPolyQuot(f: RINGELEM, I: IDEAL, z: RINGELEM): RINGELEM
MinPolyQuotDef(f: RINGELEM, I: IDEAL, z: RINGELEM): RINGELEM
```

```
MinPolyQuotElim(f: RINGELEM, I: IDEAL, z: RINGELEM): RINGELEM
MinPolyQuotMat(f: RINGELEM, I: IDEAL, z: RINGELEM): RINGELEM
```

## Description

This functions return the minimal polynomial (in the indeterminate “z”) of the element “f” modulo the 0-dimensional ideal “I”.

See article Abbott, Bigatti, Palezzato, Robbiano ”Computing and Using Minimal Polynomials” (“<https://arxiv.org/abs/1702.07262>”)

Verbosity: “MinPolyQuot” uses modular methods when coefficients are in “QQ” (I-17.1 pg.251). At level 80 it lists all primes used indicating any which are ”bad”.

### example

```
/**/ use P := QQ[x,y];
/**/ I := IdealOfPoints(P, mat([[1,2], [3,4], [5,6]]));
/**/ MinPolyQuotDef(x,I,x); -- the smallest x-univariate poly in I
x^3 -9*x^2 +23*x -15
/**/ indent(factor(It));
record[
  RemainingFactor := 1,
  factors := [x -1, x -3, x -5],
  multiplicities := [1, 1, 1]
]

/**/ f := x+y;
/**/ I := ideal(x^2, y^2);
/**/ MinPolyQuotDef(f,I,x);
x^3
/**/ subst(It, x, f) isin I;
true

/**/ use QQ[x,y];
/**/ I := ideal(x^3-5,y^2-3);
/**/ f := x+y;
/**/ SetVerbosityLevel(80);
/**/ MinPolyQuot(f, I, x);
1: prime is 32009
2: prime is 32027
x^6 -9*x^4 -10*x^3 +27*x^2 -90*x -2

---- this is how to use an indet in another ring:
/**/ QQt := RingQQt(1);
/**/ MinPolyQuotDef(f, I, indet(RingQQt(1),1));
t^3
```

**See Also:** MinPoly(I-13.22 pg.200)

## I-13.24 MinPowerInIdeal

### syntax

```
MinPowerInIdeal(F: RINGELEM, I: IDEAL): INT
```

## Description

This function returns the minimum power of  $F$ , the first argument, in the ideal  $I$ , the second argument. If  $F$  is not in the radical  $I$  then -1 is returned.

example

```
/**/ use QQ[x,y,z];
/**/ I := ideal(x^6*y^4, z);
/**/ IsInRadical(x*y, I);
true

/**/ MinPowerInIdeal(x*y, I);
6
```

**See Also:** [IsInRadical\(I-9.57 pg.158\)](#), [radical\(I-18.1 pg.255\)](#)

## I-13.25 MinSubsetOfGens

syntax

```
MinSubsetOfGens(M: IDEAL|MODULE): LIST
```

## Description

This function returns a subset “S” of “gens(M)” which is minimal in the sense that no proper subset of “S” generates “M”.

NOTE: in general there might be other subsets with smaller cardinality.

If “M” is a homogeneous ideal or module, the function “MinGens” ([I-13.16 pg.198](#)) is much faster (but may return a generating set which is not a subset of “gens(M)”).

The coefficient ring must be a field.

example

```
/**/ use R ::= QQ[x,y,z];
/**/ I := ideal(x-1, (x-y)^4, z+y, (z+y)^2);
/**/ MinSubsetOfGens(I);
[x -1, x^4 -4*x^3*y +6*x^2*y^2 -4*x*y^3 +y^4, y +z]
```

**See Also:** [MinGens\(I-13.16 pg.198\)](#), [IdealOfMinGens\(I-9.6 pg.135\)](#), [SubmoduleOfMinGens\(I-19.46 pg.302\)](#)

## I-13.26 mod

syntax

```
mod(N: INT, D: INT): INT
```

## Description

We define the quotient “Q” and remainder “R” to be integers which satisfy  $N = Q * D + R$  with  $0 \leq R < |D|$ . Then “div(N, D)” returns “Q” while “mod(N, D)” returns “R”.

NOTE: To perform the division algorithm on a polynomial, use “NR” ([I-14.32 pg.222](#)) (normal remainder) to find the remainder, or “DivAlg” ([I-4.21 pg.81](#)) to get both the quotients and the remainder.

example

```
/**/ div(10,3);
3
/**/ mod(10,3);
1
```

**See Also:** [div\(I-4.20 pg.81\)](#), [DivAlg\(I-4.21 pg.81\)](#), [GenRepr\(I-7.7 pg.110\)](#), [NF\(I-14.16 pg.215\)](#), [NR\(I-14.32 pg.222\)](#)

## I-13.27 Mod2Rat [OBSOLETE]

[OBSOLETE] syntax

### Description

[OBSOLETE] See “RatReconstructWithBounds” ([I-18.18 pg.262](#)).

## I-13.28 ModuleElem

syntax  
ModuleElem(M: MODULE, L: LIST): MODULEELEM

### Description

This function returns the MODULEELEM (called “Vector” in CoCoA-4) in the module “M” whose components are the components of the list L.

example

```

/**/ use R ::= QQ[x];
/**/ R3 := NewFreeModule(R,3);
/**/ V := ModuleElem(R3, [1, x, x^2]); V;
[1, x, x^2]
/**/ type(V);
MODULEELEM
/**/ zero(R3);
[0, 0, 0]
```

**See Also:** [SubmoduleCols](#), [SubmoduleRows\(I-19.45 pg.302\)](#)

## I-13.29 ModuleOf

syntax  
ModuleOf(M: MODULE): MODULE

### Description

This function returns the module on which the object E is defined.

NOTE: A module contains many information and two separate rings, even when defined with the same commands, are not “equal”. When a module is printed only a few informations are shown, so different modules might look the same.

example

```

/**/ use R ::= QQ[x];
/**/ R3 := NewFreeModule(R,3);
/**/ V := ModuleElem(R3, [1, x, x^2]); V;
[1, x, x^2]
/**/ type(V);
```

```

MODULEELEM
/**/ ModuleOf(V) = R3;
true
/**/ ModuleOf(V);
FreeModule(RingDistrMPolyClean(QQ, 1), 3)

```

**See Also:** `submodule`([I-19.44](#) pg.301)

## I-13.30 monic

— syntax —

```
monic(F: RINGELEM): RINGELEM
```

### Description

This function returns “F” divided by its leading coefficient (see “LC” ([I-12.4](#) pg.180)) or, if “F” is zero, it returns just zero.

— example —

```

/**/ use R := QQ[x,y];
/**/ F := 4*x^5-y^2;
/**/ monic(F);
x^5 +(-1/4)*y^2

/**/ use R := ZZ[x,y];
/**/ F := 4*x^5-y^2;
-- /**/ monic(F); --> !!! ERROR !!! as expected
ERROR: Inexact division
monic(L); -- can't invert coefficients over ZZ
~~~~~

/**/ use P := ZZ/(5)[x,y];
/**/ F := 2*x^2+4*y^3;
/**/ monic(F);
y^3 -2*x^2

```

**See Also:** `LC`([I-12.4](#) pg.180)

## I-13.31 monomials

— syntax —

```
monomials(F: RINGELEM|MODULEELEM): LIST
```

### Description

This function returns the list of monomials of F. The function “support” ([I-19.50](#) pg.304) returns the list of terms (monomials without coefficients).

— example —

```

/**/ use R := QQ[x,y];
/**/ F := 3*x^2*y +5*y^3 -x*y^5;
/**/ monomials(F);
[-x*y^5, 3*x^2*y, 5*y^3]

/**/ support(F);

```

```
[x*y^5, x^2*y, y^3]

Monomials(Vector(3*x^2*y+y, 5*x*y+4)); --***WORK IN PROGRESS***
[Vector(3x^2y, 0), Vector(0, 5xy), Vector(y, 0), Vector(0, 4)]
```

**See Also:** [coefficients\(I-3.24 pg.54\)](#), [support\(I-19.50 pg.304\)](#)

## I-13.32 *MonsInIdeal*

syntax

```
MonsInIdeal(I: IDEAL): IDEAL
```

### Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This function returns the ideal generated by all monomials in the original ideal I.

example

```
use R := QQ[x,y,z];
I := ideal(xy^3+z^2, y^5-z^3, xz-y^2-x^3, x^4-xz^2+y^3);
MonsInIdeal(I);
ideal(z^3, yz^2, x^2z^2, x^5z, x^4yz, x^5y, x^2y^2z, x^7, x^4y^2,
      xy^3z, y^4z, xy^4, x^3y^3, y^5)
-----
```

## I-13.33 *MSatLinSolve*

syntax

```
MSatLinSolve(Env: RECORD): MATRIX
```

### Description

This function calls “MathSAT” ([II-9.5 pg.365](#)) implementation of the simplex method.

The work for this communication has been supported by the European Union’s Horizon 2020 research and innovation programme under grant agreement No H2020-FETOPEN-2015-CSA 712689: SC-square “<http://www.sc-square.org>”.

example

```
/**/ system :=
/**/   record[leq0 := matrix([[1,2,3, 4], //   x +2*y +3*z +4 <= 0
/**/                               [9,8,7, 0]]), // 9*x +8*y +7*z   <= 0
/**/   neq0 := matrix([[1,0,0, 0]]) //   x               <> 0
/**/   ];
/**/ sol := MSatLinSolve(system); sol;
matrix(QQ,
  [[1],
   [4/5],
   [-11/5]])
/**/ // verify:
/**/ sol1 := ConcatVer(sol, matrix([[1]])); //--> [[1], [4/5], [-11/5], [1]]
/**/ system.leq0 * sol1; // is <= 0
matrix(QQ,
  [[0],
   [0]])
```

```

/**/ system.neq0 * sol1; // is <> 0
matrix(QQ,
  [[1]])
/**/ // now we add new constraints:
/**/ system.eq0 := RowMat([1,1,0, 4]); // x +y +4 = 0
/**/ system.lt0 := RowMat([0,1,0, 0]); // y < 0
/**/ sol := MSatLinSolve(system); sol;
matrix(QQ,
  [[-2],
   [-2],
   [-2/7]])
/**/ // verify:
/**/ sol1 := ConcatVer(sol, RowMat([1])); //--> [[-2], [-2], [-2/7], [1]]
/**/ system.leq0 * sol1; // <= 0
matrix(QQ,
  [[-20/7],
   [-36]])
/**/ system.neq0 * sol1; // <> 0
matrix(QQ,
  [[-2]])
/**/ system.eq0 * sol1; // = 0
matrix(QQ,
  [[0]])
/**/ system.lt0 * sol1; // < 0
matrix(QQ,
  [[-2]])

```

**See Also:** [LinSolve\(I-12.15 pg.185\)](#), [MathSAT\(II-9.5 pg.365\)](#)

## I-13.34 MultiplicationMat

syntax

```

MultiplicationMat(X: RINGELEM, I: IDEAL): MAT
MultiplicationMat(X: RINGELEM, I: IDEAL, QB: LIST): MAT

```

### Description

This function computes the multiplication matrix of a ringelem “f” modulo a zero-dimensional ideal “I” with respect to a quotient basis of “I”.

In the second form it is computed with respect to the given quotient basis “QB”.

example

```

/**/ use QQ[x,y];
/**/ I := ideal(x*y +y^2 -x -4*y +3, x^2 -y^2 -4*x +2*y +3, y^3 -4*y^2 +5*y -2);
/**/ MultiplicationMat(x, I);
matrix(QQ,
  [[0, -3, -5, -3],
   [0, 4, 6, -2],
   [0, -1, -1, 1],
   [1, 1, 1, 4]])

/**/ MultiplicationMat(x, I, [one(CurrentRing), x, y, y^2]);
matrix(QQ,
  [[0, -3, -3, -5],
   [1, 4, 1, 1],

```

```
[0, -2, 4, 6],  
[0, 1, -1, -1]])
```

## I-13.35 *multiplicity*

— *syntax* —

```
multiplicity(R: RING): INT
```

### Description

This function computes the multiplicity (or degree) of  $M$ , i.e., the leading coefficient of the Hilbert polynomial multiplied by the factorial of the degree of the Hilbert polynomial.  $M$  can be a module or a quotient.

— *example* —

```
/**/ use R := QQ[t,x,y,z];  
/**/ multiplicity(R/ideal(x,y,z)^5);  
35
```

**See Also:** [HilbertFn\(I-8.7 pg.123\)](#), [HilbertSeries\(I-8.10 pg.125\)](#), [HVector\(I-8.16 pg.128\)](#)



# Chapter I-14

## N

### I-14.1 NewFractionField

syntax

```
NewFractionField(R: RING): RING
```

#### Description

NOTE: calling twice “NewFractionField” will produce two different rings, even with identical input: equality test is performed on the pointers. See “RingID” (I-18.45 pg.274).

example

```
/**/ K := NewFractionField(NewPolyRing(QQ, "a,b"));
/**/ use K;
/**/ M := mat([[1,2,3,a],[5,6,7,a*b]]);
/**/ LinKerBasis(M);
[[-1, 2, -1, 0], [(a*b -3*a)/2, (-a*b +5*a)/4, 0, -1]]
```

**See Also:** NewQuotientRing(I-14.9 pg.212), RingID(I-18.45 pg.274), den(I-4.7 pg.75), num(I-14.33 pg.223)

### I-14.2 NewFreeModule

syntax

```
NewFreeModule(R: RING, N: INT): MODULE
NewFreeModule(R: RING, Shifts: MAT): MODULE
```

#### Description

This function returns a free module which can be used as any programming variable.

NOTE: as for rings, calling twice “NewFreeModule” will produce two different modules, even with identical input: equality test is performed on the pointers.

This function does accept shifts from version CoCoA-5.0.4.

example

```
/**/ use R ::= QQ[x,y];
/**/ F := NewFreeModule(R, 3);
/**/ zero(F);
[0, 0, 0]
/**/ type(zero(F)); -- is NOT a LIST
MODULEELEM
/**/ gens(F);
```

```
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]

/**/ F := NewFreeModule(R, matrix([[1],[2],[3]])); -- shifts
/**/ [wdeg(e) | e in gens(F)];
[[1], [2], [3]]
```

**See Also:** [BaseRing\(I-2.1 pg.37\)](#), [RingOf\(I-18.46 pg.275\)](#)

### I-14.3 NewId [OBSOLETE]

syntax

```
[OBSOLETE]
```

#### Description

[OBSOLETE]

### I-14.4 NewLine [OBSOLESCENT]

syntax

```
NewLine(): STRING
```

#### Description

This function is “*OBSOLESCENT*” and exists only for backward compatibility with old CoCoA code. It returns a string containing just a newline; in CoCoA-5 it is simpler to write “`\n`”.

example

```
/**/ str1 := "Line 1" + NewLine() + "Line 2"; --> old CoCoA-4 way
/**/ str2 := "Line 1\nLine 2";               --> more compact in CoCoA-5
/**/ str1 = str2;
True
/**/ Print str2;
Line 1
Line2
```

**See Also:** [String Literals\(III-4.1 pg.383\)](#), [println\(I-16.34 pg.247\)](#), [ascii\(I-1.18 pg.35\)](#)

### I-14.5 NewList

syntax

```
NewList(N: INT): LIST
NewList(N: INT, E: OBJECT): LIST
```

#### Description

The first form returns a list of length “N” filled with 0 (“INT”). The second form returns a list of length “N”, filled with copies of “E”.

example

```
/**/ NewList(4,"a");
["a", "a", "a", "a"]
```

```
/**/ NewList(4);
[0, 0, 0, 0]
```

**See Also:** List Constructors([III-5.2](#) pg.388)

## I-14.6 NewMat

syntax

```
NewMat(R: RING, M: INT, N: INT): MAT
```

### Description

This function is kept for CoCoA-4 nostalgia: better use “ZeroMat” ([I-25.2](#) pg.337).

example

```
/**/ use S := QQ[x,y,z];
/**/ NewMat(S,2,3);
matrix( /*RingDistrMPolyClean(QQ, 3)*/
  [[0, 0, 0],
   [0, 0, 0]])
/**/ ZeroMat(S,2,3);
matrix( /*RingDistrMPolyClean(QQ, 3)*/
  [[0, 0, 0],
   [0, 0, 0]])
```

**See Also:** matrix([I-13.10](#) pg.195), NewMatFilled([I-14.7](#) pg.211)

## I-14.7 NewMatFilled

syntax

```
NewMatFilled(M: INT, N: INT, Val: INT|RAT|RINGELEM): MAT
```

### Description

This function returns an “MxN” matrix, filled with “Val”. If “Val” is an integer or rational, the ring of the matrix is “QQ” ([I-17.1](#) pg.251).

example

```
/**/ use S := QQ[x,y,z];
/**/ NewMatFilled(1,3,x);
matrix( /*RingDistrMPolyClean(QQ, 3)*/
  [[x, x, x]])

/**/ NewMatFilled(1,3, 0);
matrix(QQ,
  [[0, 0, 0]])
/**/ ZeroMat(QQ, 1, 3); --> same as NewMatFilled(1,3, 0)
matrix(QQ,
  [[0, 0, 0]])
```

**See Also:** NewMat([I-14.6](#) pg.211), matrix([I-13.10](#) pg.195)

## I-14.8 NewPolyRing

### syntax

```
NewPolyRing(CoeffRing: RING, IndetNames: STRING/LIST): RING
NewPolyRing(CoeffRing: RING, IndetNames: STRING/LIST, OrdMat: MAT, GradingDim: INT): RING
```

### Description

This function returns a polynomial ring which can be used as any programming variable (assigned with “:=”).

The “:=” syntax starts the input method for a new polynomial ring, with the special interpretation of brackets and symbols (i.e. “R := QQ[x]” is not read the same way as “X := L[i]”). The pre-defined orderings for the “:=” syntax are “Lex” (no grading), “DegLex”, “DegRevLex” (standard grading). For more orderings use the “NewPolyRing” function call (see also “ElimMat” (I-5.6 pg.85)).

NOTE: calling “NewPolyRing” twice with the same arguments gives two “*different rings*”, therefore incompatible. See “RingID” (I-18.45 pg.274).

NOTE: the syntax with all indet names in one string is new in CoCoA-5.1.2.

### example

```
/**/ R := QQ[x,y,alpha]; -- is equivalent to
/**/ R := NewPolyRing(QQ, "x,y,alpha"); -- in "define/enddefine" use "RingQQ()"

/**/ R := QQ[x,y], DegRevLex; -- is equivalent to
/**/ R := NewPolyRing(QQ, "x,y", StdDegRevLexMat(2), 1);

/**/ OrdM := matrix([[2,3,1],[0,0,-1],[0,-1,0]]);
/**/ P := NewPolyRing(QQ, "x[1],x[2],x[9]", OrdM, 1); -- 3 indeterminates
/**/ [wdeg(X) | X in indets(P)];
[[2], [3], [1]]

/**/ P2 := NewPolyRing(RingZZ(), IndetSymbols(P)); -- same indet names as P
/**/ Indets(P2);
[x[1], x[2], x[9]]

/**/ P3 := NewPolyRing(P2, SymbolRange("alpha", -2,2));
/**/ indets(P3);
[alpha[-2], alpha[-1], alpha[0], alpha[1], alpha[2]]
```

**See Also:** ElimMat(I-5.6 pg.85), RingID(I-18.45 pg.274), IndetSymbols(I-9.26 pg.145), SymbolRange(I-19.55 pg.307), MakeTermOrd(I-13.5 pg.192), GradingDim(I-7.29 pg.117)

## I-14.9 NewQuotientRing

### syntax

```
NewQuotientRing(R: RING, I: IDEAL): RING
R/I
```

### Description

NOTE: calling twice “NewQuotientRing” will produce two different rings, even with identical input: equality test is performed on the pointers. See “RingID” (I-18.45 pg.274).

### example

```
/**/ use Qi := QQ[i];
/**/ CC := Qi/ideal(i^2+1); -- sort of ;-)
```

```

/**/ use CC[x];
/**/ (x+i)^2;
x^2 +(2*i)*x +(-1)

/**/ R ::= QQ[x,y,z];
/**/ S := NewQuotientRing(R, ideal(indet(R,1)-3));
/**/ use S;
/**/ (x+y)^2;
(y^2 +6*y +9)

```

**See Also:** RingID([I-18.45 pg.274](#)), QuotientBasis([I-17.4 pg.252](#)), NewFractionField([I-14.1 pg.209](#))

## I-14.10 NewRingFp

syntax

```
NewRingFp(P: INT): RING
```

### Description

Create a new small prime finite field with characteristic “P”.

NOTE: in ring definitions you can use the convenient notation “ZZ/(p)”

NOTE: calling twice “NewRingFp” will produce two different rings, even with identical input: equality test is performed on the pointers. See “RingID” ([I-18.45 pg.274](#)).

example

```

/**/ p := NextPrime(1000);
/**/ Fp := NewRingFp(p);
/**/ use Fp[x];
/**/ product([x-i | i in 1..p]);
x^1009 - x
/**/ use ZZ/(p)[x]; --> convenient shorthand in ring defn
/**/ product([x-i | i in 1..p]);
x^1009 - x

```

**See Also:** RingID([I-18.45 pg.274](#)), NewQuotientRing([I-14.9 pg.212](#))

## I-14.11 NewRingTwinFloat

syntax

```
NewRingTwinFloat(Prec: INT): RING
```

### Description

Create a new twin-float ring with bit precision “Prec”.

NOTE: calling twice “NewRingTwinFloat” will produce two different rings, even with identical arguments, since the equality test for rings is performed on the pointers: see manual entry for “RingID” ([I-18.45 pg.274](#)).

For more information see the article: John Abbott, “Twin-float arithmetic”, Journal of Symbolic Computation, Volume 47 (2012), 536–551.

example

```

/**/ RR32 := NewRingTwinFloat(32);
/**/ use RR32[x];
/**/ (3*x-1)/3;

```



---

```
NextPrime(N: INT): INT
PrevPrime(N: INT): INT
```

The first function computes the smallest prime number greater than “N”. If “N” is negative or too large then an error is signalled. The upper limit depends on the computer you are using; it is probably  $2^{31}$  or  $2^{63}$ .

The second function computes the greatest prime number smaller than “N”. If “N” is less than 3 or too large then an error is signalled.

```
example
/**/ NextPrime(1000);
1009
/**/ PrevPrime(1000);
997
```

---

syntax

```
NextProbPrime(N: INT): INT
PrevProbPrime(N: INT): INT
```

This function computes the smallest probable prime number greater than N. If N is negative, an error is generated. To be absolutely certain the number produced is prime, you must call IsPrime on it, but this may be very costly.

[illegible]

```

NF(F: RINGELEM, I: IDEAL): RINGELEM
NF(V: MODULEELEM, M: MODULE): MODULEELEM

```

The first function returns the normal form of  $F$  with respect to  $I$ . It also computes a Groebner basis of  $I$  if that basis has not been computed previously.

The second function returns the normal form of  $V$  with respect to  $M$ . It also computes a Groebner basis of  $M$  if that basis has not been computed previously.

Currently (v 5.0.3) only full reduction is computed: each monomial in the result cannot be reduced. CoCoA-4 allowed setting the flag FullRed (of the panel GROEBNER) to False so that only the leading term is reduced.

Currently (v 5.0.3) polynomial ideals are implemented only with coeffs in a field.

example

```

/**/ use R ::= QQ[x,y,z];
/**/ I := ideal(z);
/**/ NF(x^2+x*y+x*z+y^2+y*z+z^2, I);
x^2 +x*y +y^2

/**/ I := ideal(z-1);
/**/ NF(x^2+x*y+x*z+y^2+y*z+z^2, I);
x^2 +x*y +y^2 +x +y +1

```

**See Also:** DivAlg([I-4.21](#) pg.81), GenRepr([I-7.7](#) pg.110), IsIn([I-9.53](#) pg.157), NR([I-14.32](#) pg.222)

## I-14.17 NFsAreZero [OBSOLETE]

syntax

[OBSOLETE]

### Description

[OBSOLETE] “NFsAreZero(L, I)” is the same as “IsContained(ideal(L), I)”.

**See Also:** IsContained([I-9.41](#) pg.152), IsIn([I-9.53](#) pg.157), NF([I-14.16](#) pg.215)

## I-14.18 NmzComputation

syntax

```

NmzComputation(Cone: RECORD): RECORD
NmzComputation(Cone: RECORD, ToCompute: LIST): RECORD

```

### Description

“NmzComputation” provides direct access to libnormaliz. It faithfully reflects the internal structure of the libnormaliz design. Its first argument should be a record representing the cone. For the possible input options see the Normaliz documentation. With the second (optional) argument one can specify what should be computed. If it is omitted, everything that can be computed by libnormaliz will be computed.

example

```

/**/ Cone := record[ integral_closure := mat([[1,2],[2,1]]),
/**/               grading := mat([[2,1]])];
/**/ NC2 := NmzComputation(Cone, ["HilbertBasis", "SupportHyperplanes", "HilbertSeries"]);
/**/ indent(NC2);

record[
  Congruences := [],
  Deg1Elements := [],
  EmbeddingDim := 2,
  Equations := [],
  ExtremeRays := [[1, 2], [2, 1]],
  Generators := [[1, 2], [2, 1]],
  Grading := [2, 1],

```

```

HilbertBasis := [[1, 1], [1, 2], [2, 1]],
HilbertSeries := record[DenFactors := record[RemainingFactor := 1, factors := [-t + 1, -t^20 + 1], mul
IsDeg1HilbertBasis := false,
IsInhomogeneous := false,
IsIntegrallyClosed := false,
IsPointed := true,
Multiplicity := 3/20,
Rank := 2,
SupportHyperplanes := [[-1, 2], [2, -1]]
]

```

**See Also:** NmzIntClosureToricRing(I-14.24 pg.219), NmzNormalToricRing(I-14.26 pg.220), NmzIntClosureMonIdeal(I-14.23 pg.219), NmzEhrhartRing(I-14.20 pg.217), NmzTorusInvariants(I-14.28 pg.221), NmzFiniteDiagInvariants(I-14.21 pg.218), NmzDiagInvariants(I-14.19 pg.217), NmzIntersectionValRings(I-14.25 pg.220), NmzSetVerbosityLevel(I-14.27 pg.220)

## I-14.19 NmzDiagInvariants

syntax

```
NmzDiagInvariants(M: MAT, N: MAT, R: Ring): LIST of RINGELEM
```

### Description

This function computes the ring of invariants of a diagonalizable group  $D = TxG$  where  $T$  is a torus and  $G$  is a finite abelian group, both acting diagonally on the polynomial ring  $K[X_1, \dots, X_n]$ .

The group actions are specified by the input matrices “M” and “N”. The first matrix specifies the torus action, the second the action of the finite group. See NmzTorusInvariants or NmzFiniteDiagInvariants for more detail. The output is the monomial subalgebra of invariants in “R”.

example

```

/**/      use R:=QQ[x,y,z,w];
/**/      T := matrix([[ -1,-1,2,0],[1,1,-2,-1]]);
/**/      U := matrix([[1,1,1,1,5],[1,0,2,0,7]]);
/**/      NmzDiagInvariants(T,U,R);
[x^4*y^6*z^5, x^15*y^5*z^10, x*y^19*z^10, x^26*y^4*z^15, x^37*y^3*z^20, x^48*y^2*z^25, x^59*y*z^30, x^

```

**See Also:** NmzComputation(I-14.18 pg.216), NmzTorusInvariants(I-14.28 pg.221), NmzFiniteDiagInvariants(I-14.21 pg.218), NmzSetVerbosityLevel(I-14.27 pg.220)

## I-14.20 NmzEhrhartRing

syntax

```
NmzEhrhartRing(L: LIST of RINGELEM, s: RINGELEM): LIST of RINGELEM
```

### Description

The exponent vectors of the given monomials are considered as vertices of a lattice polytope “P”. The Ehrhart ring of a (lattice) polytope “P” is the monoid algebra defined by the monoid of lattice points in the cone over the polytope “P”; see the book by Bruns and Gubeladze, Polytopes, Rings, and K-theory, publ. Springer 2009, pp. 228–229.

The function returns the generators of the Ehrhart ring. It uses the indeterminate in the second argument as auxiliary indeterminate of the Ehrhart ring.

example

```

/**/      use R:=QQ[x,y,z,t];
/**/      NmzEhrhartRing([x^2,y^2,z^3],t);
[x^2*t, z^3*t, x*y*t, y^2*t]

```

**See Also:** [NmzComputation\(I-14.18 pg.216\)](#), [NmzHilbertBasis\(I-14.22 pg.218\)](#), [NmzNormalToricRing\(I-14.26 pg.220\)](#), [NmzIntClosureMonIdeal\(I-14.23 pg.219\)](#), [NmzSetVerbosityLevel\(I-14.27 pg.220\)](#)

## I-14.21 NmzFiniteDiagInvariants

syntax

```

NmzFiniteDiagInvariants(M: MAT, M: Ring): LIST of RINGELEM

```

### Description

This function computes the ring of invariants of a finite abelian group  $G$  acting diagonally on the surrounding polynomial ring  $K[X_1, \dots, X_n]$ .

The group is the direct product of cyclic groups generated by finitely many elements  $g_1, \dots, g_w$ . The element  $g_i$  acts on the indeterminate  $X_j$  by  $g_i(X_j) = l_i^{u_{ij}} X_j$  where  $l_i$  is a primitive root of unity of order equal to  $\text{ord}(g_i)$ .

The ring of invariants is generated by all monomials satisfying the system  $u_{i1}a_1 + \dots + u_{in}a_n$  and congruent to 0 mod  $\text{ord}(g_i)$ ,  $i = 1, \dots, w$ .

The input to the function is the  $w$  times  $(n+1)$  matrix “U” with rows  $u_{i1} \dots u_{in} \text{ord}(g_i)$ ,  $i = 1, \dots, w$ . The output is the monomial subalgebra of invariants  $R^G = \text{fin}R : g_i f = f \text{ for all } i = 1, \dots, w$ .

example

```

/**/      use R:=QQ[x,y,z,w];
/**/      U := matrix([[1,1,1,1,3],[1,0,2,0,4]]);
/**/      NmzFiniteDiagInvariants(U,R);
[x^2*z, z^2*w, y*z^2, x^12, y^3, z^6, w^3, x^8*w, x^4*w^2, y*w^2, x^8*y, x^4*y*w, y^2*w, x^4*y^2]

```

**See Also:** [NmzComputation\(I-14.18 pg.216\)](#), [NmzTorusInvariants\(I-14.28 pg.221\)](#), [NmzDiagInvariants\(I-14.19 pg.217\)](#), [NmzSetVerbosityLevel\(I-14.27 pg.220\)](#)

## I-14.22 NmzHilbertBasis

syntax

```

NmzHilbertBasis(M: MAT): MAT

```

### Description

Given a matrix “M”, this function returns a matrix whose rows represent the Hilbert-Gordan Basis for the monoid generated by the rows of “M”.

example

```

/**/      M := matrix([[0,1],[3,1]]);
/**/      NmzHilbertBasis(M);
--the Hilbert basis of the monoid generated by the vectors [0,1] and [3,1] is...
matrix(QQ,
  [[3, 1],
   [0, 1]])
-- ... ([3,1], [0,1])

-- CAREFUL!! Different result from...
/**/      HilbertBasisKer(M);

```

```
-- the Hilbert basis of M is the Hilbert basis of the monoid of
-- elements in the kernel of M, namely...
[]
-- ...no elements! (except the zero-element)
```

**See Also:** [HilbertBasisKer\(I-8.6 pg.123\)](#), [NmzComputation\(I-14.18 pg.216\)](#), [NmzNormalToricRing\(I-14.26 pg.220\)](#), [NmzIntClosureMonIdeal\(I-14.23 pg.219\)](#), [NmzSetVerbosityLevel\(I-14.27 pg.220\)](#)

## I-14.23 NmzIntClosureMonIdeal

syntax

```
NmzIntClosureMonRing(L: LIST of RINGELEM): LIST of RINGELEM
NmzIntClosureMonRing(L: LIST of RINGELEM, s: RINGELEM): LIST of RINGELEM,
```

### Description

Given a list “L” of power-products in a ring “R”, the function returns the generators of the integral closure of the ideal generated by “L”.

As second argument you can specify an indeterminate of the ring which is not used in the power-products. In this case the result is the normalisation of its Rees algebra (or Rees ring); see Bruns and Herzog, Cohen-Macaulay Rings, Cambridge University Press 1998, p. 182.

example

```
/**/      use R:=QQ[x,y,z,t];
/**/      NmzIntClosureMonIdeal([x^2,y^2,z^3]);
-- the integral closure of the ideal generated by x^2,y^2 and z^3 is...
[y^2, x^2, x*y, z^3, y*z^2, x*z^2]
-- ...the ideal generated by y^2, x^2, x*y, z^3, y*z^2 and x*z^2
/**/      NmzIntClosureMonIdeal([x^2,y^2,z^3],t);
-- and the complete rees algebra is generated by
[z, z^3*t, y, y*z^2*t, y^2*t, x, x*z^2*t, x*y*t, x^2*t]
```

**See Also:** [NmzComputation\(I-14.18 pg.216\)](#), [NmzHilbertBasis\(I-14.22 pg.218\)](#), [NmzNormalToricRing\(I-14.26 pg.220\)](#), [NmzEhrhartRing\(I-14.20 pg.217\)](#), [NmzSetVerbosityLevel\(I-14.27 pg.220\)](#)

## I-14.24 NmzIntClosureToricRing

syntax

```
NmzIntClosureToricRing(L: LIST of RINGELEM): LIST of RINGELEM
```

### Description

Given a list L of power-products in a ring R, the function returns the generators of the integral closure of the algebra generated by the list.

example

```
/**/      use R:=QQ[x,y,t];
/**/      NmzIntClosureToricRing([x^3,x^2*y,y^3]);
-- the integral closure of QQ[x^3, x^2*y, y^3] is...
[y,x]
-- ... QQ[y, x]
```

**See Also:** [NmzComputation\(I-14.18 pg.216\)](#), [NmzHilbertBasis\(I-14.22 pg.218\)](#), [NmzNormalToricRing\(I-14.26 pg.220\)](#), [NmzIntClosureMonIdeal\(I-14.23 pg.219\)](#), [NmzEhrhartRing\(I-14.20 pg.217\)](#), [NmzSetVerbosityLevel\(I-14.27 pg.220\)](#)

## I-14.25 NmzIntersectionValRings

syntax

`NmzIntersectionValRings(M: MAT, M: Ring): LIST of RINGELEM`

### Description

A discrete monomial valuation  $v$  on  $R = K[X_1, \dots, X_n]$  is determined by the values  $v(X_j)$  of the indeterminates. This function computes the subalgebra  $S = \text{fin}R : v_i(f) \geq 0, i = 1, \dots, r$  that is the intersection of the valuation rings of the given valuations  $v_1, \dots, v_r$ , i.e. it consists of all elements of  $R$  that have a nonnegative value for all  $r$  valuations. It takes as input the matrix  $V = (v_i(X_j))$  whose rows correspond to the values of the indeterminates.

example

```
/**/      use R:=QQ[x,y,z,w];
/**/      V := matrix([[0,1,2,3],[-1,1,2,1]]);
/**/      NmzIntersectionValRings(V,R);
[y, z, w, x*y, x^2*z, x*w, x*z]
```

**See Also:** [NmzComputation\(I-14.18 pg.216\)](#), [NmzSetVerbosityLevel\(I-14.27 pg.220\)](#)

## I-14.26 NmzNormalToricRing

syntax

`NmzNormalToricRing(L: LIST of RINGELEM): LIST of RINGELEM`

### Description

Given a list “L” of power-products in a ring  $R$ , the function returns the generators of the normalization of the algebra generated by the list.

example

```
/**/      use R:=QQ[x,y,t];
-- We compute the normalization of QQ[x^3, x^2*y, y^3]
/**/      NmzNormalToricRing([x^3, x^2*y, y^3]);
[y^3, x^2*y, x^3, x*y^2]
--> answer is QQ[y^3, x^2*y, x^3, x*y^2]
```

**See Also:** [NmzComputation\(I-14.18 pg.216\)](#), [NmzHilbertBasis\(I-14.22 pg.218\)](#), [NmzIntClosureToricRing\(I-14.24 pg.219\)](#), [NmzIntClosureMonIdeal\(I-14.23 pg.219\)](#), [NmzSetVerbosityLevel\(I-14.27 pg.220\)](#)

## I-14.27 NmzSetVerbosityLevel

syntax

`NmzSetVerbosityLevel(v: INT)`

### Description

Set the verbosity level for the external library Normaliz: level 0 means no “verbosity”, any positive value activates “verbosity”.

NOTE: this is completely independent of CoCoA's own verbosity level!

example

```
/**/ NmzSetVerbosityLevel(1);
```

**See Also:** SetVerbosityLevel([I-19.11](#) pg.286), NmzVerbosityLevel([I-14.29](#) pg.221)

## I-14.28 NmzTorusInvariants

syntax

```
NmzTorusInvariants(M: MAT, R: Ring): LIST of RINGELEM
```

### Description

Let “ $T=(K^*)^r$ ” be the  $r$ -dimensional torus acting on the polynomial ring “ $R=K[X_1, \dots, X_n]$ ” diagonally. Such an action can be described as follows: there are integers  $a_{ij}, i = 1, \dots, r, j = 1, \dots, n$  such that  $(l_1, \dots, l_r)$  in “ $T$ ” acts by the substitution  $X_j$  maps to  $l_1^{a_{1j}} * \dots * l_r^{a_{rj}} * X_j$  for  $j=1, \dots, n$ .

The function takes the matrix  $M = (a_{ij})$  and the ring “ $R$ ” as input. It computes the ring of invariants  $R^T = \text{fin}R | f = f \text{ for all } l \text{ in } T$ .

example

```
/**/ use R:=QQ[x,y,z,w];
/**/ T := matrix([[-1,-1,2,0],[1,1,-2,-1]]);
/**/ NmzTorusInvariants(T,R);
[x^2*z, x*y*z, y^2*z]
```

**See Also:** NmzComputation([I-14.18](#) pg.216), NmzDiagInvariants([I-14.19](#) pg.217), NmzFiniteDiagInvariants([I-14.21](#) pg.218), NmzSetVerbosityLevel([I-14.27](#) pg.220)

## I-14.29 NmzVerbosityLevel

syntax

```
NmzVerbosityLevel(): INT
```

### Description

Returns the verbosity level for the external library Normaliz: value is 0 or 1 according as Normaliz verbosity was inactive or active.

NOTE: this is completely independent of CoCoA's own verbosity level!

example

```
/**/ NmzVerbosityLevel();
0
```

**See Also:** VerbosityLevel([I-22.2](#) pg.329), NmzSetVerbosityLevel([I-14.27](#) pg.220)

## I-14.30 NonZero

syntax

```
NonZero(L: LIST|MODULEELEM): LIST
```

## Description

This function returns the list obtained by removing the zeroes from L.

example

```
/**/ use R ::= QQ[x,y,z];
/**/ NonZero([0,0,3, ideal(y),0]);
[3, ideal(y)]
```

**See Also:** FirstNonZero([I-6.9](#) pg.96), FirstNonZeroPosn([I-6.10](#) pg.97), IsZero([I-9.90](#) pg.170)

## I-14.31 not

syntax

```
not(A: BOOL): BOOL
```

## Description

This function negates a boolean: i.e. if “A” gives “true” then “not(A)” gives “false”, and vice versa.

Note that from CoCoA-5.1 “not” is a function, so its argument must be between brackets!

example

```
/**/ [n in 1..10 | not(IsPrime(n))];
[1,4,6,8,9]
```

**See Also:** and([I-1.11](#) pg.31), or([I-15.9](#) pg.231)

## I-14.32 NR

syntax

```
NR(X: RINGELEM, L: LIST of RINGELEM): RINGELEM
NR(X: MODULEELEM, L: LIST of MODULEELEM): MODULEELEM
```

## Description

This function returns the normal remainder of X with respect to L, i.e., it returns the remainder from the division algorithm. To get both the quotients and the remainder, use “DivAlg” ([I-4.21](#) pg.81).

Note that if the list does not form a Groebner basis, the remainder may not be zero even if X is in the ideal or module generated by L (use “GenRepr” ([I-7.7](#) pg.110) or “NF” ([I-14.16](#) pg.215) instead).

Currently (v 5.0.3) the internal code for computing NF(F, I) and NR(F, GBasis(I)) is identical, but the second is slower just for the overhead in interpreting a possibly long list of polynomials.

example

```
/**/ use R ::= QQ[x,y,z];
/**/ F := x^2*y + x*y^2 + y^2;
/**/ NR(F, [x*y-1, y^2-1]);
x + y + 1

// NOT YET IMPLEMENTED for MODULEELEM
```

**See Also:** DivAlg([I-4.21](#) pg.81), GenRepr([I-7.7](#) pg.110), NF([I-14.16](#) pg.215)

## I-14.33 num

syntax

```
num(N: INT): INT
num(N: RAT): INT
num(N: RINGELEM): RINGELEM
```

### Description

This function returns the numerator of “N”.

The OBSOLETE fragile syntax in CoCoA 4 “N.Num” and “N.Den” is no longer supported.

example

```
/**/ num(3);
3

/**/ P := QQ[x,y];
/**/ F := NewFractionField(P);
/**/ use F;
/**/ num(x/(x+y));
x
```

**See Also:** den([I-4.7](#) pg.75)

## I-14.34 NumCols

syntax

```
NumCols(M: MAT): INT
```

### Description

This function returns the number of columns in a matrix.

example

```
/**/ M := mat([[1,2,3], [4,5,6]]);
/**/ NumCols(M);
3
```

**See Also:** matrix([I-13.10](#) pg.195), NumRows([I-14.39](#) pg.225)

## I-14.35 NumCompts

syntax

```
NumCompts(X: MODULEELEM|MODULE): INT
```

### Description

If “X” is a “MODULEELEM”, it returns the number of components of “X”. If “X” is a “MODULE”, it returns the rank of the free module in which “X” is defined.

This function used to be called “NumComps” in CoCoA-4.

example

```
/**/ use R := QQ[x,y];
/**/ R2 := NewFreeModule(R, 3);
/**/ M := SubmoduleRows(R2, matrix(R, mat([[x,0,y], [x^2+y^2,x^2,3]])));
```

```

/**/ NumCompts(M);
3
/**/ NumCompts(gens(M)[1]);
3

```

**See Also:** [len\(I-12.6 pg.181\)](#)

## I-14.36 NumGens

— syntax —

```
NumGens(I: IDEAL): INT
```

### Description

This function returns the number of generators of “I”. This is more direct, therefore efficient, than writing “len(gens(I))”, because it does not create the temporary list “gens(I)”.

— example —

```

/**/ use R := QQ[x,y,z];
/**/ I := ideal(indets(R))^40;
/**/ NumGens(I);
861

```

**See Also:** [len\(I-12.6 pg.181\)](#)

## I-14.37 NumIndets

— syntax —

```
NumIndets(R: RING): INT
```

### Description

This function returns the number of indeterminates of the ring “R”.

— example —

```

/**/ S := QQ[x,y];
/**/ R := QQ[x,y,z];
/**/ NumIndets(R);
3
/**/ NumIndets(S);
2

```

**See Also:** [indet\(I-9.21 pg.143\)](#), [IndetSubscripts\(I-9.25 pg.145\)](#), [IndetIndex\(I-9.22 pg.143\)](#), [IndetName\(I-9.23 pg.144\)](#), [indets\(I-9.24 pg.144\)](#)

## I-14.38 NumPartitions

— syntax —

```
NumPartitions(N: INT): INT
```

### Description

This function returns the number of partitions of a non-negative integer, i.e. the number of distinct ways of writing “N” as a sum of positive integers.

example

```

/**/ NumPartitions(2); -- 2 and 1+1
2
/**/ NumPartitions(5);
7

```

## I-14.39 NumRows

syntax

```

NumRows(M: MAT): INT

```

### Description

This function returns the number of rows in a matrix.

example

```

/**/ M := mat([[1,2,3], [4,5,6]]);
/**/ NumRows(M);
2

```

**See Also:** [matrix\(I-13.10 pg.195\)](#), [NumCols\(I-14.34 pg.223\)](#)

## I-14.40 NumTerms

syntax

```

NumTerms(F: RINGELEM): INT

```

### Description

This function returns the number of terms in a polynomial.

example

```

/**/ use R := QQ[x,y,z];
/**/ NumTerms((x+y+z)^5) = binomial(3+5-1, 5);
true

```

**See Also:** [len\(I-12.6 pg.181\)](#)



# Chapter I-15

## O

### I-15.1 one

syntax

```
one(R: RING): RINGELEM
```

#### Description

This function return the multiplicative identity of a ring. For when you want to force the integer “1” to be a “RINGELEM”.

example

```
/**/ P := ZZ/(101)[x,y,z];
/**/ N := 1; Print N, " of type ", type(N);
1 of type INT
/**/ N := one(P); Print N, " of type ", type(N);
1 of type RINGELEM
/**/ N := 300*1; Print N, " of type ", type(N);
300 of type INT
/**/ N := 300*one(P); Print N, " of type ", type(N);
-3 of type RINGELEM
```

**See Also:** [zero\(I-25.1 pg.337\)](#)

### I-15.2 OpenIFile

syntax

```
OpenIFile(S: STRING): DEVICE
```

#### Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This function opens the file with name S for input. Input from that file can then be read with “Get” ([I-7.10 pg.112](#)).

NOTE: it is better to use “source” ([I-19.26 pg.293](#)) to read CoCoA commands from a file.

example

```
D := OpenOFile("my-test"); -- open "my-test" for output from CoCoA
Print "hello world" On D; -- print string into "mytest"
Close(D);
```

```

D := OpenIFile("my-test"); -- open "my-test" for input to CoCoA
Get(D,3); -- get the first three characters (in Ascii code)
[104, 101, 108]
-----
ascii(It); -- convert the ASCII code into characters
hel
-----
Close(D);

```

**See Also:** [close\(I-3.16 pg.51\)](#), [Introduction to IO\(II-7.1 pg.355\)](#), [OpenOFile\(I-15.5 pg.229\)](#), [OpenIString\(I-15.3 pg.228\)](#), [OpenOString\(I-15.6 pg.230\)](#), [OpenSocket\(I-15.7 pg.230\)](#), [source\(I-19.26 pg.293\)](#)

## I-15.3 OpenIString

syntax

```

OpenIString(S: STRING, T: STRING): DEVICE
OpenOString(S: STRING): OSTREAM

```

### Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This function open strings for input. The string S serves as the name of the device opened for input or output; one may use the empty string. “OpenIString” is used to read input from the string T with the help of “Get” ([I-7.10 pg.112](#)).

example

```

S := "hello world";
D := OpenIString("", S); -- open the string S for input to CoCoA
L := Get(D,7); -- read 7 characters from the string
L; -- ASCII code
[104, 101, 108, 108, 111, 32, 119]
-----
ascii(L); -- convert ASCII code to characters
hello w
-----
Close(D); -- close device D

```

**See Also:** [close\(I-3.16 pg.51\)](#), [Introduction to IO\(II-7.1 pg.355\)](#), [OpenOString\(I-15.6 pg.230\)](#), [OpenIFile\(I-15.2 pg.227\)](#), [OpenOFile\(I-15.5 pg.229\)](#), [source\(I-19.26 pg.293\)](#), [sprintf\(I-19.29 pg.294\)](#)

## I-15.4 OpenLog

syntax

```

OpenLog(D: DEVICE)

```

### Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This function opens the output device D and starts to record the output from a CoCoA session on D. The “CloseLog” ([I-3.17 pg.51](#)) closes the device D and stops recording the CoCoA session on D.

At present the choices for the device D are an output file (see “OpenOFile” (I-15.5 pg.229)) or an output string (see “OpenOString” (I-15.6 pg.230)). Several output devices may be open at a time. If the panel option “Echo” is set to True, both the input and output of the CoCoA session are logged; otherwise, just the output is logged.

## example

```
D := OpenOFile("MySession");
OpenLog(D);
1+1;
2
-----
G := 1;
Set Echo;
2+2;
2 + 2
4
-----
F := 2;
F := 2
CloseLog(D);
CloseLog(D)
UnSet Echo;
SET(Echo, False)

-- The contents of "MySession":
2
-----
2 + 2
4
-----
F := 2
CloseLog(D)
```

**See Also:** Introduction to IO(II-7.1 pg.355), OpenIFile(I-15.2 pg.227), OpenOFile(I-15.5 pg.229), OpenIString(I-15.3 pg.228), OpenOString(I-15.6 pg.230)

## I-15.5 OpenOFile

## syntax

```
OpenOFile(S: STRING): OSTREAM
OpenOFile(S: STRING, "w" or "W"): OSTREAM
```

### Description

This function opens the file with name “S”—creating it if it does not already exist—for output. It immediately erases the file with name “S”. Then the function “print on” (I-16.30 pg.245) is then used for appending output to “S” until it is closed.

## example

```
/**/ file := OpenOFile("my-test"); -- open "my-test" for output from CoCoA
/**/ print "hello world! " on file; -- print string into "my-test"
/**/ print " test" on file;         -- append to the file "my-test"
/**/ close(file); -- close the file
/**/ file := OpenOFile("my-test"); -- clear "my-test" and re-open it
/**/ print "goodbye" on file; -- "mytest" now consists only of the string "goodbye"
/**/ close(file);
```

**See Also:** [close\(I-3.16 pg.51\)](#), [Introduction to IO\(II-7.1 pg.355\)](#), [OpenIFile\(I-15.2 pg.227\)](#), [OpenIString\(I-15.3 pg.228\)](#), [OpenOString\(I-15.6 pg.230\)](#), [source\(I-19.26 pg.293\)](#)

## I-15.6 OpenOString

syntax

```
OpenOString(): OSTREAM
```

### Description

This function opens strings for output. “OpenOString” is used to write to a string with the help of “print on” ([I-16.30 pg.245](#)). use “close” ([I-3.16 pg.51](#)) to obtain the string of characters ”printed on” the ostream.

example

```
/**/ str := OpenOString(); -- open a string for output from CoCoA
/**/ L := [1,2,3]; -- a list
/**/ print L on str; -- print to str
/**/ str;
<out-stream>
-----
/**/ close(str); -- gives a string containing the output sent to str
[1, 2, 3]
```

**See Also:** [close\(I-3.16 pg.51\)](#), [Introduction to IO\(II-7.1 pg.355\)](#), [OpenIFile\(I-15.2 pg.227\)](#), [OpenOFile\(I-15.5 pg.229\)](#), [OpenIString\(I-15.3 pg.228\)](#), [source\(I-19.26 pg.293\)](#), [sprintf\(I-19.29 pg.294\)](#)

## I-15.7 OpenSocket

syntax

```
OpenSocket(Machine: STRING, Port: STRING): DEVICE
```

### Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This function opens a client socket (I/O) connection. It requires the name of the machine with the server socket and the port number (expressed as a STRING).

CoCoA-4 communicates with the CoCoAServer via socket which, by default, runs on “localhost” on port “0xc0c0”. To change these settings redefine in your “userinit.coc” or “.cocoarc” the variables

```
MEMORY.CoCoAServerMachine := "localhost";
MEMORY.CoCoAServerPort := "0xc0c0";
```

example

```
D := OpenSocket("localhost", "10000");
Print 100^6 On D;
Source D;
Close(D);
```

**See Also:** [Introduction to IO\(II-7.1 pg.355\)](#), [OpenIFile\(I-15.2 pg.227\)](#), [OpenOFile\(I-15.5 pg.229\)](#), [OpenIString\(I-15.3 pg.228\)](#), [OpenOString\(I-15.6 pg.230\)](#)

## I-15.8 Option [OBSOLETE]

[OBSOLETE]

syntax

### Description

[OBSOLETE]

## I-15.9 or

syntax

A or B (where A, B: BOOL, return BOOL)

### Description

This operator represents the logical disjunction of “A” and “B”. CoCoA first evaluates “A”; if that gives “true” then the result is “true”, and “B” is not evaluated. Otherwise, if “A” gives “false” then “B” is evaluated, and its value is the final result.

example

```
/**/ Define IsUnsuitable(X)
/**/   Return X < 0 or FloorSqrt(X) >= 2^16;
/**/ EndDefine;
/**/ IsUnsuitable(-9);
true
/**/ IsUnsuitable(9);
false
```

**See Also:** and(I-1.11 pg.31), not(I-14.31 pg.222)

## I-15.10 OrdMat

syntax

OrdMat(R: RING): MAT

### Description

This function returns a matrix which describes the term-ordering of the ring “R”.

example

```
/**/ use S ::= QQ[x,y,z];
/**/ M := mat([ [1,2,3], [3,4,5], [0,0,1]]);
/**/ P := NewPolyRing(CoeffRing(S), IndetSymbols(S), M, 2);
/**/ GradingDim(P);
2
/**/ OrdMat(P);
matrix(QQ,
  [[1, 2, 3],
   [3, 4, 5],
   [0, 0, 1]])
/**/ GradingDim(S);
```

```
1
/**/ OrdMat(S);
matrix(QQ,
  [[1, 1, 1],
   [0, 0, -1],
   [0, -1, 0]])
```

**See Also:** StdDegLexMat([I-19.37](#) pg.299), StdDegRevLexMat([I-19.38](#) pg.299), LexMat([I-12.7](#) pg.182), RevLexMat([I-18.41](#) pg.272), XelMat([I-24.1](#) pg.335), elim([I-5.4](#) pg.84), GradingDim([I-7.29](#) pg.117), Term Orderings([III-9.5](#) pg.404), NewPolyRing([I-14.8](#) pg.212)

# Chapter I-16

## P

### I-16.1 packages

syntax

```
packages(): LIST of STRING
```

#### Description

This function returns the names of the loaded packages as a list of strings.  
The old CoCoA-4 names “\$user” and “\$builtin” are no longer used.

example

```
/**/ packages();  
["$BackwardCompatible", "$BringIn", (...)]
```

**See Also:** CoCoA Packages([II-8 pg.359](#)), Supported Packages([II-8.7 pg.361](#))

### I-16.2 panel [OBSOLETE]

syntax

```
[OBSOLETE]
```

#### Description

[OBSOLETE]

### I-16.3 panels [OBSOLETE]

syntax

```
[OBSOLETE]
```

#### Description

[OBSOLETE]

## I-16.4 partitions

syntax

```
partitions(N: INT): LIST
```

### Description

This function returns all integer partitions of N, positive integer

example

```
/**/ partitions(3);
[[3], [1, 2], [1, 1, 1]]
```

**See Also:** subsets([I-19.47 pg.303](#)), tuples([I-20.15 pg.317](#))

## I-16.5 permutations

syntax

```
permutations(L: LIST): LIST
```

### Description

This function computes all permutations of the entries of a list (set). If “L” has repeated elements it will return repeated elements.

example

```
/**/ permutations(3..5);
[[3, 4, 5], [3, 5, 4], [4, 3, 5], [4, 5, 3], [5, 3, 4], [5, 4, 3]]

/**/ permutations([2, 2, 5]);
[[2, 2, 5], [2, 5, 2], [2, 2, 5], [2, 5, 2], [5, 2, 2], [5, 2, 2]]

/**/ MakeSet(permutations([2, 2, 5]));
[[2, 2, 5], [2, 5, 2], [5, 2, 2]]
```

**See Also:** subsets([I-19.47 pg.303](#)), tuples([I-20.15 pg.317](#))

## I-16.6 PerpIdealOfForm

syntax

```
PerpIdealOfForm(F: RINGELEM): IDEAL
```

### Description

Thanks to Enrico Carlini.

Given a form “F” computes the ideal of derivations killing it.

For the sake of simplicity Forms/Polynomials and Derivations live in the same ring, the distinction between them is purely formal.

example

```
/**/ use R := QQ[x,y,z];
/**/ PerpIdealOfForm(x^3+x*y*z);
ideal(z^2, y^2, x^2 -6*y*z)
```

```

/**/ HilbertFn(R/It);
H(0) = 1
H(1) = 3
H(2) = 3
H(3) = 1
H(t) = 0   for t >= 4

```

**See Also:** [InverseSystem\(I-9.35 pg.150\)](#), [DerivationAction\(I-4.11 pg.77\)](#)

## I-16.7 *pfaffian*

— syntax —

```
pfaffian(M: MAT): RINGELEM
```

### Description

This function returns the Pfaffian of M.

— example —

```

/**/ use R ::= QQ[x,y];
/**/ pfaffian(mat([[0,y],[-y,0]]));
y

```

**See Also:** [det\(I-4.13 pg.78\)](#)

## I-16.8 *PkgName*

— syntax —

```

PkgName(): STRING
S.PkgName(): STRING

```

where S is the identifier or alias for a package.

### Description

This function returns the (long) name of a package. The first form returns “\$coclib” and the second returns the name of the package whose name or alias is S. This function is useful as a shorthand, when S is an alias, for the full name a package.

— example —

```

GB.PkgName();
$gb
-----
$gb.PkgName();
$gb
-----
PkgName();
$coclib
-----

```

## I-16.9 PlotPoints

syntax

```
PlotPoints(L: LIST of points)
```

### Description

This function outputs the coordinates of the points (with two components) to a file called "CoCoAPlot". See "PlotPointsOn" (I-16.10 pg.236) for outputting to another file.

This result can be plotted using your preferred plotting program. For example, start "gnuplot" and then give it the command

```
plot "CoCoAPlot"
```

to see the plot.

example

```
/**/ PlotPoints([ [X, X^2-X+14] | X in -10..10]);
Plotting points...100%
21 plotted points have been placed in the file CoCoAPlot
```

**See Also:** ImplicitPlot(I-9.15 pg.139), PlotPointsOn(I-16.10 pg.236)

## I-16.10 PlotPointsOn

syntax

```
PlotPointsOn(L: LIST of points, S: STRING)
```

### Description

This function is the same as "PlotPoints" (I-16.9 pg.236) with a second argument giving the name of the file to print on.

Note that the last argument is a STRING, the name of the file, and not an OSTREAM, as for "print on" (I-16.30 pg.245).

example

```
/**/ PlotPointsOn([ [1/(X+1/2), X^2-X+14] | X in -10..10], "PLOT-points");
Plotting points...100%
21 plotted points have been placed in the file points

/**/ use QQ[x,y];
/**/ ImplicitPlotOn(x^2*y -(59/4)*x^2 +2*x -1, [-3,3], [0,250], "PLOT-curve");
Plotting points...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
735 plotted points have been placed in the file "PLOT-curve"
```

After having produced the plot files using CoCoA, start "gnuplot" and then give it the following commands:

```
plot "PLOT-curve"
replot "points"
```

**See Also:** ImplicitPlot(I-9.15 pg.139), PlotPoints(I-16.9 pg.236)

## I-16.11 `poincare` [OBSOLESCENT]

syntax

```
[OBSOLESCENT]poincare(M: RING|IDEAL):TAGGED("$hp.PSeries")
```

### Description

(sorry Poincare' for the lower-case: here we follow the naming convention “*single name goes lower-case*”)

[OBSOLESCENT] Now called “HilbertSeries” (I-8.10 pg.125).

**See Also:** HilbertSeries(I-8.10 pg.125)

## I-16.12 `PoincareMultiDeg` [OBSOLETE]

syntax

```
[OBSOLETE]
```

### Description

[OBSOLETE] now called “HilbertSeriesMultiDeg” (I-8.11 pg.126)

## I-16.13 `PoincareShifts` [OBSOLETE]

syntax

```
[OBSOLETE]
```

### Description

[OBSOLETE] now called “HilbertSeriesShifts” (I-8.12 pg.127)

## I-16.14 `PolyAlgebraHom`

syntax

```
PolyAlgebraHom(Domain: RING, Codomain: RING, images: LIST): RINGHOM
```

### Description

This function creates the homomorphism of (polynomial) algebras from “R” to “S” with the same ring of coefficients. This is uniquely defined by the images of the indeterminates of “R” which are specified by the entries of “images”.

This is a cleaner mathematical implementation of the function “image [OBSOLESCENT]” (I-9.12 pg.138) in CoCoA-4.

example

```
/**/ use R := QQ[x,y,z];
/**/ S := QQ[x[1..3]];
/**/ phi := PolyAlgebraHom(R, S, indets(S));
/**/ phi(x^2-y);
x[1]^2 -x[2]
```

```

/**/ S := QQ[a];
/**/ phi := PolyAlgebraHom(R, S, [RingElem(S,"a"),1,0]);
/**/ phi(x^2-y);
a^2 -1

/**/ phi := PolyAlgebraHom(R, QQ, [2,1,0]); --> evaluate at [2,1,0]
/**/ phi(x^2-y);
3

```

**See Also:** [apply\(I-1.13 pg.32\)](#), [CanonicalHom\(I-3.3 pg.46\)](#)

## I-16.15 PolyRingHom

— syntax —

```
PolyRingHom(R: RING, S: RING, CoeffHom: RINGHOM, images: LIST): RINGHOM
```

### Description

This function creates the homomorphism of (polynomial) algebras between “R” and “S”. The homomorphism is uniquely defined by the images of the indeterminates of “R” and the homomorphism mapping “CoeffRing(R)” into “S”.

— example —

```

/**/ R := QQ[x,y];
/**/ S := QQ[a,b,c];
/**/ SmodJ := NewQuotientRing(S, ideal(RingElem(S,"a")^2-1));

/**/ use SmodJ;
/**/ phi := PolyRingHom(R, SmodJ, CanonicalHom(QQ,SmodJ), [a,b]);
/**/ use R;
/**/ phi(x);
(a)

```

**See Also:** [apply\(I-1.13 pg.32\)](#), [CanonicalHom\(I-3.3 pg.46\)](#)

## I-16.16 PowerMod

— syntax —

```
PowerMod(A: INT, B: INT, M: INT): INT
```

### Description

This function calculates efficiently an integer power modulo a given modulus. Thus “PowerMod(A, B, M)” is equal to “mod(A^B, M)”, but the former is computed faster. “B” must be non-negative.

— example —

```

/**/ PowerMod(12345,41041,41041); -- 41041 is a Carmichael number
12345

/**/ PowerMod(123456789,987654321,32003); -- cannot compute 123456789^987654321 directly
2332

```

## I-16.17 PreImage [OBSOLESCENT]

syntax

[OBSOLESCENT]

### Description

Changed into “preimage0” (I-16.18 pg.239).

**See Also:** preimage0(I-16.18 pg.239)

## I-16.18 preimage0

syntax

preimage0(phi: RINGHOM, f: RINGELEM): RECORD

### Description

This function returns a preimage of “f” via “phi”; if “f” is not in the image of “phi”, the function returns 0.

example

```

/**/ QQxyz ::= QQ[x,y,z];
/**/ QQab  ::= QQ[a,b];

/**/ use QQab;
/**/ phi := PolyAlgebraHom(QQxyz, QQab, [a+1, a*b+3, b^2]);
/**/ IsInjective(phi);
false
/**/ ker(phi);
ideal(-x^2*z +y^2 +2*x*z -6*y -z +9)
/**/ IsSurjective(phi);
false

/**/ use QQab;
/**/ preimage0(phi, b);
0

/**/ preimage0(phi, a^2);
x^2 -2*x +1
/**/ phi(It);
a^2

```

**See Also:** ker(I-11.1 pg.177), IsSurjective(I-9.84 pg.168)

## I-16.19 PreprocessPts

syntax

PreprocessPts(Pts: MAT, Toler: MAT): RECORD  
PreprocessPtsGrid(Pts: MAT, Toler: MAT): RECORD  
PreprocessPtsAggr(Pts: MAT, Toler: MAT): RECORD  
PreprocessPtsSubDiv(Pts: MAT, Toler: MAT): RECORD

## Description

Thanks to Maria-Laura Torrente.

These functions detect groupings of close points, and choose a single representative for them (which lies within the given tolerance of each original point); the result is the list of these representatives, and the number of original points associated to each representative.

The first argument is a matrix whose rows represent a set of points in k-dimensional space, and the second argument is row-matrix of k positive tolerances (one for each dimension).

The return value is a record containing two fields: “**NewPoints**” contains a matrix whose rows represent a list of “*well-separated*” points, and “**weights**” which contains the number of input points associated to each output point.

There are three underlying algorithms: “**Grid**” is fast but crude; “**Subdiv**” works best when the original points are densely packed (so the result will be a small list); finally “**Aggr**” is best suited to situations where the original points are less densely packed.

The function “**PreprocessPts**” automatically chooses between “**Subdiv**” and “**Aggr**” with the aim of minimising computation time. Note that the “**Aggr**” and “**Subdiv**” methods regard the tolerances as being slightly flexible.

For a full description of the algorithms we refer to the paper J.Abbott, C.Fassino, L.Torrente “*Thinning Out Redundant Empirical Data*” (published in Mathematics in Computer Science, 2007).

### example

```
/**/ Pts := matrix([[-1,0],[0,0],[1,0],[99,1],[99,0],[99,-1]]);
/**/ Toler := RowMat([3,3]);
/**/ PreprocessPts(Pts, Toler);
record[NewPoints := matrix(QQ,
  [[99, 0],
   [0, 0]]), weights := [3, 3]]

/**/ PreprocessPts(Pts, RowMat([0.8,0.8]));
record[NewPoints := matrix(QQ,
  [[-1/2, 0],
   [1, 0],
   [99, 1/2],
   [99, -1]]), weights := [2, 1, 2, 1]]

/**/ PreprocessPtsAggr(Pts, RowMat([0.9,0.9])); -- exhibits tolerance flex
record[NewPoints := matrix(QQ,
  [[0, 0],
   [99, 0]]), weights := [3, 3]]
```

## I-16.20 PrevPrime, PrevProbPrime

### syntax

```
NextPrime(N: INT): INT
PrevPrime(N: INT): INT
```

## Description

The first function computes the smallest prime number greater than “N”. If “N” is negative or too large then an error is signalled. The upper limit depends on the computer you are using; it is probably  $2^31$  or  $2^63$ .

The second function computes the greatest prime number smaller than “N”. If “N” is less than 3 or too large then an error is signalled.

### example

```
/**/ NextPrime(1000);
```

```
1009
/**/ PrevPrime(1000);
997
```

**See Also:** IsPrime([I-9.71](#) pg.163), NextProbPrime, PrevProbPrime([I-14.15](#) pg.215)

## I-16.21 PrevPrime, PrevProbPrime

syntax

```
PrevPrime(N: INT): INT
PrevProbPrime(N: INT): INT
```

### Description

See “NextPrime, PrevPrime” ([I-14.14](#) pg.215), “NextProbPrime, PrevProbPrime” ([I-14.15](#) pg.215).

example

```
/**/ PrevPrime(100);
97
/**/ PrevProbPrime(10000000000);
9999999967
```

**See Also:** IsPrime([I-9.71](#) pg.163), IsProbPrime([I-9.72](#) pg.164), NextPrime, PrevPrime([I-14.14](#) pg.215), NextProbPrime, PrevProbPrime([I-14.15](#) pg.215)

## I-16.22 PrimaryDecomposition

syntax

```
PrimaryDecomposition(I: IDEAL): LIST of IDEAL
```

### Description

This function returns the primary decomposition of the ideal I. Currently it responds ONLY for squarefree monomial ideals (using the Alexander dual technique). See “PrimaryDecomposition0” ([I-16.23](#) pg.241) for 0-dimensional ideals and “FrbPrimaryDecomposition” ([I-6.24](#) pg.104) for monomial ideals.

example

```
/**/ use P ::= QQ[x,y,z];
/**/ PrimaryDecomposition(ideal(x*y, y*z, z*x));
[ideal(y, z), ideal(x, z), ideal(x, y)]
```

**See Also:** PrimaryDecomposition0([I-16.23](#) pg.241), PrimaryDecompositionGTZ0([I-16.25](#) pg.243), FrbPrimaryDecomposition([I-6.24](#) pg.104), EquiIsoDec([I-5.10](#) pg.87)

## I-16.23 PrimaryDecomposition0

syntax

```
PrimaryDecomposition0(I: IDEAL): LIST of IDEAL
```

### Description

This function returns the primary decomposition of the 0-dimensional ideal “I”. (Monico-Robbiano-Kreuzer probabilistic algorithm, Gao-Wan-Wang finite characteristic algorithm)

Implemented by Elisa Palezzato and Anna M. Bigatti.

#### example

```

/**/ use QQ[x,y,z];
/**/ PD := PrimaryDecomposition0(ideal(x -z, y^2 -1, z^2)); PD;
[ideal(y +1, x -z, y^2 -1, z^2), ideal(y -1, x -z, y^2 -1, z^2)]

/**/ [IdealOfGBasis(Q) | Q in PD]; // remove some redundant generators
[ideal(y +1, x -z, z^2), ideal(y -1, x -z, z^2)]

/**/ use ZZ/(2)[x,y,z];
/**/ PD := PrimaryDecomposition0(ideal(x +z, y^2 +1, z^2)); PD;
[ideal(x +z, y^2 +1, z^2)]
/**/ IsPrimary(PD[1]);
true

```

NOTE: this function behaved like “PrimaryDecompositionCore0” up to version CoCoA-5.1.4.

**See Also:** PrimaryDecompositionGTZ0(I-16.25 pg.243), PrimaryDecompositionCore0(I-16.24 pg.242), PrimaryDecomposition(I-16.22 pg.241)

## I-16.24 PrimaryDecompositionCore0

#### syntax

```

PrimaryDecompositionCore0(I: IDEAL): RECORD
PrimaryDecompositionCore0(I: IDEAL, opt f: RINGELEM): RECORD

```

### Description

This function is the core partial step for the primary decomposition of a 0-dimensional ideal “I”: see “PrimaryDecomposition0” (I-16.23 pg.241) for the certified and reduced result.

The result is a record containing a (partial) decomposition and a “BOOL” stating whether or not the decomposition is certified to be the primary decomposition.

The second optional argument is for specifying the “*splitting*”, the polynomial whose minimal polynomial (see “MinPolyQuot” (?? pg.??)) is used for splitting “I” into components. See the paper Abbott, Bigatti, Palezzato, Robbiano “*Minimal polynomials and applications*” (work in progress).

Implementation by Elisa Palezzato.

NOTE: this function was called “PrimaryDecomposition0” up to version CoCoA-5.1.4.

#### example

```

/**/ use R ::= QQ[x,y,z];
/**/ PD := PrimaryDecompositionCore0(ideal(x-z, y^2-1, z^2));
/**/ indent(PD); -- decomposition is correct, but not certified
record[
  IsCertified := false,
  PrDec_I := [
    ideal(16*y^2 +40*y*z +25*z^2 +32*y +40*z +16, x -z, y^2 -1, z^2),
    ideal(16*y^2 +40*y*z +25*z^2 -32*y -40*z +16, x -z, y^2 -1, z^2)
  ]
]
/**/ PrimaryDecomposition0(ideal(x-z, y^2-1, z^2));
[ideal(y +1, x -z, z^2), ideal(y -1, x -z, z^2)]

/**/ use ZZ/(2)[x,y,z];
/**/ PD := PrimaryDecompositionCore0(ideal(x-z, y^2-1, z^2));
/**/ indent(PD);

```

```
record[
  IsCertified := true,
  PrDec_I := [ideal(x +z, y^2 +1, z^2)]
]
```

**See Also:** PrimaryDecomposition0(I-16.23 pg.241), PrimaryDecompositionGTZ0(I-16.25 pg.243)

## I-16.25 PrimaryDecompositionGTZ0

— syntax —

```
PrimaryDecompositionGTZ0(I: IDEAL): LIST of IDEAL
```

### Description

This function returns the primary decomposition of the 0-dimensional ideal “I” (using the GTZ algorithm). It will be improved and extended in future versions of CoCoA.

Implemented by Luis David Garcia (updated to CoCoA-5 by Anna M. Bigatti).

— example —

```
/**/ use R ::= QQ[x,y,z];
/**/ PD := PrimaryDecompositionGTZ0(ideal(x-z, y^2-1, z^2));
/**/ indent(PD);
```

**See Also:** PrimaryDecomposition0(I-16.23 pg.241)

## I-16.26 PrimaryHilbertSeries

— syntax —

```
PrimaryHilbertSeries(I: IDEAL, Q: IDEAL): TAGGED("PSeries")
```

### Description

Let “P” be a polynomial ring, “M” the maximal ideal in “P” generated by the indeterminates, and “(Q+I)/I” a primary ideal for “M/I”. This function computes the Hilbert-Poincare’ series of “(P/I)/((Q+I)/I)”.

— example —

```
/**/ use S ::= QQ[x,y,z];
/**/ I := ideal(x^3-y*z, y^2-x*z, z^2-x^2*y);
/**/ Q := ideal(y, z);
/**/ PS := PrimaryHilbertSeries(I, Q); PS;

/**/ use S ::= ZZ/(32003)[x,y,z,w];
/**/ I := ***Ideal(
/**/   x^5 - yz, y^4 - xz^2, xy^3 - zw, x^2z - yw,
/**/   y^2z^2 - w^3, y^3z - x^2w^2, x^3w - z^2, xyw^2 - z^3,
/**/   x^3y^2 - w^2, xz^4 - y^2w^3, yz^5 - xw^5, y^3w^5 - z^7,
/**/   x^2w^7 - z^8, z^9 - yw^8)***;
/**/ Q := ideal(x, y, z);
/**/ PS := PrimaryHilbertSeries(I, Q); PS;
/**/ $hp.PSerToHilbert(PS);

/**/ use S ::= ZZ/(32003)[x,y,z];
/**/ I := ideal(S, []); -- ideal in S with no generators
```

```

/**/ Q := ideal(x, y, z^2);
/**/ PS := PrimaryHilbertSeries(I, Q); PS;
/**/ $hp.PSerToHilbert(PS);
/**/ HV := $hp.PSerHVector(PS); -- the H-vector associated to PS

/**/ use S := ZZ/(32003)[x,y,z,w];
/**/ I := ***Ideal(-yz + xw, z^3 - yw^2, -xz^2 + y^2w, -y^3 + x^2z)***;
/**/ Q := ideal(x, y, z^2, w^3);
/**/ PS := PrimaryHilbertSeries(I, Q); PS;
/**/ $hp.PSerToHilbert(PS);
/**/ HV := $hp.PSerHVector(PS);
/**/ $primary.E(0, HV);
/**/ [ $primary.E(J,HV) | J in 0..($hp.PSerDim(PS)-2) ];
/**/ [ $primary.E(J,HV) | J in 0..(len(HV)-1) ];

/**/ use S := ZZ/(32003)[x,y,z,w];
/**/ I := ***Ideal(x^3-y^7, x^2y - xw^3-z^6)***;
/**/ Q := ideal(x, y, z, w);
/**/ PS := PrimaryHilbertSeries(I, Q); PS;
/**/ $hp.PSerToHilbert(PS);
/**/ HV := $hp.PSerHVector(PS);
/**/ [ $primary.E(J,HV) | J in 0..($hp.PSerDim(PS)-2) ];
/**/ [ $primary.E(J,HV) | J in 0..(len(HV)-1) ];

/**/ use S := ZZ/(32003)[x,y,z];
/**/ I := ideal(z^3);
/**/ Q := ideal(x^2, y^2, x*z, y*z);
/**/ PS := PrimaryHilbertSeries(I, Q); PS;
/**/ $hp.PSerToHilbert(PS);
/**/ HV := $hp.PSerHVector(PS);
/**/ [ $primary.E(J,HV) | J in 0..($hp.PSerDim(PS)-2) ];
/**/ [ $primary.E(J,HV) | J in 0..(len(HV)-1) ];

```

**See Also:** [InitialIdeal\(I-9.28 pg.147\)](#), [TgCone\(I-20.5 pg.312\)](#)

## I-16.27 PrimaryPoincare [OBSOLESCE]

syntax

[OBSOLESCE]

### Description

Renamed “PrimaryHilbertSeries” ([I-16.26 pg.243](#)).

**See Also:** [PrimaryHilbertSeries\(I-16.26 pg.243\)](#)

## I-16.28 PrimitiveRoot

syntax

PrimitiveRoot(P: INT): INT

## Description

Find a primitive root modulo the prime “P”, i.e. a generator of the cyclic multiplicative group of non-zero integers mod “P”.

Currently, the function produces the least positive primitive root.

example

```
/**/ PrimitiveRoot(17551561);
97
/**/ PrimitiveRoot(4111);
12;
```

**See Also:** [IsPrime\(I-9.71 pg.163\)](#)

## I-16.29 *print*

syntax

```
print E_1, ..., E_n
```

## Description

This command displays the value of each of the expressions “E\_i”. To insert a newline write “\n”.

The similar command “println” ([I-16.34 pg.247](#)) is equivalent to “print” with a final newline.

example

```
/**/ for I := 1 To 10 Do print I^2, " "; endfor;
1 4 9 16 25 36 49 64 81 100

/**/ print "a\nb";
a
b
```

**See Also:** [print on\(I-16.30 pg.245\)](#), [println\(I-16.34 pg.247\)](#), [format\(I-6.19 pg.102\)](#), [LaTeX\(?? pg.??\)](#), [StarPrint](#), [StarSprint\(I-19.34 pg.297\)](#), [syntax\(?? pg.??\)](#)

## I-16.30 *print on*

syntax

```
print E: OBJECT on OUT: OSTREAM
```

## Description

This command prints the value of expression “E” to the output stream “OUT”. Currently, the command can be used to print to files, strings, or the CoCoA window. In the first two cases, the appropriate device must be opened with “OpenOFile” ([I-15.5 pg.229](#)) or “OpenOString” ([I-15.6 pg.230](#)).

example

```
/**/ file := OpenOFile("my-test"); -- open "my-test" for output from CoCoA
/**/ println "hello world" on file; -- print string into "mytest"
/**/ close(file); -- close the file
```

See “OpenOFile” ([I-15.5 pg.229](#)) for an example using output strings. For printing to the CoCoA window, just use “println E”.

**See Also:** Introduction to IO([II-7.1](#) pg.355), OpenIFile([I-15.2](#) pg.227), OpenOFile([I-15.5](#) pg.229), OpenIString([I-15.3](#) pg.228), OpenOString([I-15.6](#) pg.230), print([I-16.29](#) pg.245), println([I-16.34](#) pg.247), syntax([??](#) pg.??)

## I-16.31 PrintBettiDiagram

— syntax —

```
PrintBettiDiagram(X: IDEAL or (quotient)RING or MODULE)
PrintBettiDiagram(X: LIST(res) or RECORD(diagram))
```

### Description

This function prints the ("Macaulay-style") Betti diagram for "M".

— example —

```
/**/ use R ::= QQ[t,x,y,z];
/**/ I := ideal(x^2-y*t, x*y-z*t, x*y);
/**/ RES := res(I);
/**/ PrintRes(RES);
0 --> R(-5)^2 --> R(-4)^4 --> R(-2)^3
/**/ B := BettiDiagram(RES); indent(B);
record[
  Diagram := matrix(ZZ,
    [[3, 0, 0],
     [0, 4, 2]]),
  FirstShift := 2
]
/**/ PrintBettiDiagram(RES); -- same as PrintBettiDiagram(I or B)
      0      1      2
-----
2:      3      -      -
3:      -      4      2
-----
Tot:      3      4      2
```

**See Also:** BettiDiagram([I-2.3](#) pg.38), BettiMatrix([I-2.4](#) pg.38), PrintRes([I-16.35](#) pg.248), PrintBettiMatrix([I-16.32](#) pg.246)

## I-16.32 PrintBettiMatrix

— syntax —

```
PrintBettiMatrix(M: IDEAL|MODULE|Resolution)
```

### Description

This function prints the Betti matrix for M.

— example —

```
/**/ use R ::= QQ[t,x,y,z];
/**/ I := ideal(x^2-y*t, x*y-z*t, x*y);
/**/ PrintRes(I);
0 --> R^2(-5) --> R^4(-4) --> R^3(-2)
-----
```

```

/**/ PrintBettiMatrix(I);
  0   0   0
  0   0   3
  0   0   0
  0   4   0
  2   0   0

```

**See Also:** [PrintRes\(I-16.35 pg.248\)](#), [PrintBettiDiagram\(I-16.31 pg.246\)](#), [PrintBettiNumbers\(I-16.33 pg.247\)](#)

## I-16.33 PrintBettiNumbers

syntax

```
PrintBettiNumbers(M: IDEAL|MODULE|Resolution)
```

### Description

This function prints the Betti numbers for “M”.

example

```

/**/ M := MakeTermOrd(matrix([[5,5,5,1,1], [1,1,1,0,0]]));
/**/ P := NewPolyRing(QQ, "t[1],t[2],t[3],x,y", M, 2); -- ZZ^2-grading
/**/ use P;
/**/ I := ideal(t[1]^6 -t[3]^6, t[2]^6 -t[1]^5*t[3], t[1]*t[3]*x^8 -t[2]^2*y^8);
/**/ RES := res(P/I);
/**/ PrintRes(RES);
0 --> R[-78,-14] --> R[-48,-8]^2(+)R[-60,-12] --> R[-18,-2](+)R[-30,-6]^2 --> R

/**/ PrintBettiNumbers(RES); --> just prints in a readable way
----- 1 -----
----- 2 -----
  [18,  2]: 1
  [30,  6]: 2
----- 3 -----
  [48,  8]: 2
  [60, 12]: 1
----- 4 -----
  [78, 14]: 1
-----

/**/ BettiNumbers(RES); --> returns the value for further computations
[[], [[18, 2], 1], [[30, 6], 2], [[48, 8], 2],
 [[60, 12], 1], [[78, 14], 1]]

```

**See Also:** [PrintRes\(I-16.35 pg.248\)](#), [PrintBettiDiagram\(I-16.31 pg.246\)](#), [PrintBettiMatrix\(I-16.32 pg.246\)](#), [BettiNumbers\(I-2.5 pg.39\)](#)

## I-16.34 println

syntax

```
println E_1,...,E_n
PrintLn E_1,...,E_n
```

## Description

This command is equivalent to “`print`” ([I-16.29 pg.245](#)) with a final newline; in other words, it prints the values of its arguments, then moves the cursor to the next line.

example

```
/**/ for i := 1 to 3 do print i; endfor;
123

/**/ for i := 1 to 3 do println i; endfor;
1
2
3
```

**See Also:** `LaTeX`([?? pg.??](#)), `print`([I-16.29 pg.245](#)), `print on`([I-16.30 pg.245](#)), `syntax`([?? pg.??](#))

## I-16.35 PrintRes

syntax

```
PrintRes(M)
```

## Description

This function prints the minimal free resolution of “`M`”. (see “`res`” ([I-18.34 pg.269](#))).

example

```
/**/ use R ::= QQ[x,y,z];
/**/ I := ideal(x, y, z^2);
/**/ RES := res(I);
/**/ PrintRes(I); -- recomputes resolution
0 --> R(-4) --> R(-2)(+)R(-3)^2 --> R(-1)^2(+)R(-2)
/**/ PrintRes(RES); -- just prints RES
0 --> R(-4) --> R(-2)(+)R(-3)^2 --> R(-1)^2(+)R(-2)
/**/ PrintBettiDiagram(RES); -- just prints the BettiDiagram for RES
      0      1      2
-----
1:      2      1      -
2:      1      2      1
-----
Tot:      3      3      1
```

**See Also:** `PrintBettiDiagram`([I-16.31 pg.246](#)), `PrintBettiMatrix`([I-16.32 pg.246](#)), `res`([I-18.34 pg.269](#))

## I-16.36 PrintSectionalMatrix

syntax

```
PrintSectionalMatrix(I: IDEAL): MAT
PrintSectionalMatrix(P/I: RING): MAT
```

## Description

This function prints the sectional matrix of “`I`” or “`P/I`”. See “`SectionalMatrix`” ([I-19.4 pg.282](#)) for more information.

## example

```

/**/ use P := QQ[x,y,z];
/**/ I := ideal(x^4 -x*y^3,  x*y -z^2,  x*z^2 -y^3);
/**/ SectionalMatrix(P/I);
matrix(ZZ,
  [[1, 1, 0, 0, 0, 0, 0, 0],
   [1, 2, 2, 1, 0, 0, 0, 0],
   [1, 3, 5, 6, 5, 3, 2, 2]])

/**/ PrintSectionalMatrix(P/I);
0 1 2 3 4 5 6 7
- - - - - - - -
1 1 0 0 0 0 0 0
1 2 2 1 0 0 0 0
1 3 5 6 5 3 2 2

/**/ PrintSectionalMatrix(I);
0 1 2 3 4 5 6 7
- - - - - - - -
0 0 1 1 1 1 1 1
0 0 1 3 5 6 7 8
0 0 1 4 10 18 26 34

```

**See Also:** [SectionalMatrix\(I-19.4 pg.282\)](#)

## I-16.37 product

## syntax

```

product(L: LIST): OBJECT
product(L: LIST, InitVal: OBJECT): OBJECT

```

### Description

This function returns the product of the objects in the list “L” (together with “InitVal”, if specified). When writing a program, if the list “L” may be empty, you must specify “InitVal”.

## example

```

/**/ use R := QQ[x,y];
/**/ product([3, x, y^2]);
3*x*y^2

/**/ product(1..40) = factorial(40);
true

/**/ product([]); -- gives 1 of type INT
1
/**/ product([], y);
y
/**/ product([3, x], y);
3*x*y

```

**See Also:** [Algebraic Operators\(II-3.2 pg.347\)](#), [sum\(I-19.49 pg.304\)](#)

## I-16.38 protect

— syntax —

```
protect X;
protect X : reason;
  where reason: STRING
```

### Description

This command protects the variable “X” from being assigned to. Attempting to assign to it will produce an error; if a “reason” (STRING) was given it is printed in the error message.

— example —

```
/**/ MaxSize := 99;
/**/ protect MaxSize : "size limit for fast computation";
-- /**/ MaxSize := 1000; --> !!! ERROR !!! as expected
ERROR: Cannot set "MaxSize" (size limit for fast computation)

/**/ unprotect MaxSize; --> remove protection, X may be assigned to now
/**/ MaxSize := 1000; --> OK
```

**See Also:** unprotect([I-21.3](#) pg.326)

## I-16.39 PthRoot

— syntax —

```
PthRoot(X: RINGELEM): RINGELEM
```

### Description

This function returns the p-th root of a polynomial over a finite field. If no p-th root exists then an error is signalled. p is the characteristic of the field.

— example —

```
/**/ use R ::= ZZ/(7)[x,y];
/**/ F := x^7-y^14+3;
/**/ PthRoot(F);
-y^2+x+3
```

**See Also:** IsFiniteField([I-9.50](#) pg.155), IsPthPower([I-9.73](#) pg.164)

# Chapter I-17

## Q

### I-17.1 QQ

— syntax —

QQ

#### Description

This system variable is constant; its value is the field of rationals. Its name is protected so that it cannot be re-assigned to any other value.

Please note: this is a “*variable*”, so in “**define/define**” use “**RingQQ**” ([I-18.47 pg.276](#)) instead (or import it with “**TopLevel**” ([I-20.10 pg.314](#))).

— example —

```
/**/ use QQ;

/**/ type(5);
INT
/**/ type(RingElem(QQ, 5));
RINGELEM

/**/ QQ = RingQQ();
true
```

**See Also:** ZZ([I-25.4 pg.338](#)), NewQuotientRing([I-14.9 pg.212](#)), RingQQ([I-18.47 pg.276](#)), TopLevel([I-20.10 pg.314](#))

### I-17.2 QQEmbeddingHom

— syntax —

QQEmbeddingHom(R: RING): RINGHOM

#### Description

This function returns the homomorphism “QQ --> R”. This is useful for changing the ring of coefficients.

NOTE: this is a partial homomorphism when “R” has finite characteristic.

— example —

```
/**/ use QQxy := QQ[x,y];
/**/ f := (2/3)*x +5*y;
```

```

/**/ FFpxy := ZZ/(101)[x,y];
/**/ QQEmbeddingHom(FFpxy) (LC(f));
-33
/**/ phi := PolyRingHom(QQxy, FFpxy, QQEmbeddingHom(FFpxy), indets(FFpxy));
/**/ phi(f);
-33*x +5*y

/**/ RRxy := NewPolyRing(NewRingTwinFloat(64), "x,y");
/**/ phi := PolyRingHom(QQxy, RRxy, QQEmbeddingHom(RRxy), indets(RRxy));
/**/ phi(f);
2/3*x +5*y

```

**See Also:** [CanonicalHom\(I-3.3 pg.46\)](#)

## I-17.3 quit

— syntax —

```
quit
```

### Description

This command is used to quit CoCoA. It may be used only at top level.

**See Also:** [ciao\(I-3.14 pg.51\)](#), [exit\(I-5.15 pg.89\)](#)

## I-17.4 QuotientBasis

— syntax —

```
QuotientBasis(I: IDEAL): LIST
```

### Description

This function determines a vector space basis (of power products) for the quotient space associated to a zero-dimensional ideal. That is, if  $R$  is a polynomial ring with field of coefficients  $k$ , and  $I$  is a zero-dimensional ideal in  $R$  then  $\text{QuotientBasis}(I)$  is a set of power products forming a  $k$ -vector space basis of  $R/I$ .

The actual set of power products chosen depends on the term ordering in the ring  $R$ : the power products chosen are those not divisible by the leading term of any member of the reduced Groebner basis of  $I$  (and consequently they form a factor-closed set).

The power-products in the result are sorted in increasing lex ordering. See “[QuotientBasisSorted](#)” ([I-17.5 pg.253](#)) for sorting them according to the term-ordering of the ring.

— example —

```

/**/ use P := QQ[x,y,z];
/**/ I := intersection(ideal(x,y,z)^2, ideal(x-1, y+1, z)^2);
/**/ QB := QuotientBasis(I);
/**/ QB; -- power-products underneath the reduced GBasis of I
[1, z, y, y*z, y^2, y^3, x, x*y]

```

**See Also:** [IdealOfPoints\(I-9.7 pg.135\)](#), [IsFactorClosed\(I-9.48 pg.154\)](#)

## I-17.5 QuotientBasisSorted

syntax

```
QuotientBasisSorted(I: IDEAL): LIST
```

### Description

This function determines a vector space basis (of power products) for the quotient space associated to a zero-dimensional ideal. It is the same as “QuotientBasis” (I-17.4 pg.252), but sorted in increasing order according to the term-ordering of the ring.

example

```
/**/ use P := QQ[x,y,z];
/**/ I := intersection(ideal(x,y,z)^2, ideal(x-1, y+1, z)^2);
/**/ QBS := QuotientBasisSorted(I);  QBS;
[1, z, y, x, y*z, y^2, x*y, y^3]

/**/ QB := QuotientBasis(I);  QB;
[1, z, y, y*z, y^2, y^3, x, x*y]
```

**See Also:** QuotientBasis(I-17.4 pg.252)

## I-17.6 QuotientingHom

syntax

```
QuotientingHom(P: RING): RINGHOM
```

### Description

This function returns the projection homomorphism of a ring “R” into a quotient ring “R/I”.

It is equivalent to (indeed it is called by) “CanonicalHom(BaseRing(RModI), RModI)”.

example

```
/**/ use P := QQ[i];
/**/ f := i^3;
/**/ CC := P/ideal(i^2+1);
/**/ phi := QuotientingHom(CC);  -- phi: P -> CC
/**/ RingOf(phi(f));
/**/ phi(f)^2;
1
```

**See Also:** CanonicalHom(I-3.3 pg.46)

## I-17.7 QZP

syntax

```
QZP(F: RINGELEM): RINGELEM
QZP(F: LIST of POLY): LIST of POLY
QZP(I: IDEAL): IDEAL
```

### Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

The functions “QZP” and “ZPQ” ([I-25.3 pg.338](#)) map polynomials and ideals of other rings into ones of the current ring. When mapping from one ring to another, one of the rings must have coefficients in the rational numbers and the other must have coefficients in a finite field. The indeterminates in both rings must be identical.

The function “QZP” maps polynomials with rational coefficients to polynomials with coefficients in a finite field; the function “ZPQ” ([I-25.3 pg.338](#)) does the reverse, mapping a polynomial with finite field coefficients into one with rational (actually, integer) coefficients. The function “ZPQ” ([I-25.3 pg.338](#)) is not uniquely defined mathematically, and currently for each coefficient the least non-negative equivalent integer is chosen. Users should not rely on this choice, though any change will be documented.

example

```

use R := QQ[x,y,z];
F := 1/2*x^3 + 34/567*x*y*z - 890; -- a poly with rational coefficients
use S := ZZ/(101)[x,y,z];
QZP(F);                               -- compute its image with coeffs in ZZ/(101)
-50x^3 - 19xyz + 19
-----
G := It;
use R;
ZPQ(G);                               -- now map that result back to QQ[x,y,z]
-- it is NOT the same as F...
51x^3 + 82xyz + 19
-----
H := It;
F - H;                               -- ... but the difference is divisible by 101
-101/2x^3 - 46460/567xyz - 909
-----
use S;
QZP(H) - G;                          -- F and H have the same image in ZZ/(101)[x,y,z]
0
-----

```

**See Also:** Introduction to RINGHOM([III-10.1 pg.409](#)), BringIn([I-2.13 pg.43](#))

# Chapter I-18

## R

### I-18.1 radical

syntax

```
radical(N: INT): INT
radical(X: RINGELEM): RINGELEM
radical(I: IDEAL): IDEAL
```

#### Description

This function computes the radical of its argument. For integers and ring elements this means the product of the distinct irreducibles dividing the argument (sometimes called "square-free"). For ideals it computes the radical ideal using the algorithm described in the paper

M. Caboara, P. Conti and C. Traverso: "*Yet Another Ideal Decomposition Algorithm.*" Proc. AAECC-12, pp 39-54, 1997, Lecture Notes in Computer Science, n.1255 Springer-Verlag.

NOTE: at the moment, this implementation works only if the coefficient ring is the rationals or has large enough characteristic.

example

```
/**/ radical(99);
33
/**/ use R := QQ[x,y];
/**/ radical((x -y)^3 * (x +y));
x^2 -y^2
/**/ I := ideal(x,y)^3;
/**/ radical(I);
ideal(y, x)
```

**See Also:** [IsInRadical\(I-9.57 pg.158\)](#), [EquiIsoDec\(I-5.10 pg.87\)](#), [RadicalOfUnmixed\(I-18.2 pg.255\)](#), [SqFreeFactor\(I-19.30 pg.295\)](#)

### I-18.2 RadicalOfUnmixed

syntax

```
RadicalOfUnmixed(I: IDEAL): IDEAL
```

#### Description

This function computes the radical of an unmixed ideal.

NOTE: at the moment, this implementation works only if the coefficient ring is the rationals or has large enough characteristic.

example

```
/**/ use R ::= QQ[x,y];
/**/ I := ideal(x^2 - y^2 - 4*x + 4*y, x - 2);
/**/ RadicalOfUnmixed(I);
ideal(x^2 -y^2 -4*x +4*y, x -2, y -2)
/**/ interreduced(gens(It)); -- the result may not be in its simplest form
[y -2, x -2]
```

**See Also:** [EquiIsoDec\(I-5.10 pg.87\)](#), [radical\(I-18.1 pg.255\)](#)

## I-18.3 random

syntax

```
random(X: INT, Y: INT): INT
```

### Description

The function returns a random integer between “X” and “Y”, inclusive. The range  $|X - Y|$  should be less than  $2^3$  to ensure a uniform distribution.

NOTE: every time you restart CoCoA the sequence of random numbers will be the same (as happens in many programming languages). You can change this by using “reseed” ([I-18.35 pg.270](#)).

example

```
/**/ random(1,100);
6

/**/ random(-10^4,0);
-3263
```

**See Also:** [RandomSubset\(I-18.7 pg.257\)](#), [RandomSubsetIndices\(I-18.8 pg.258\)](#), [RandomTuple\(I-18.9 pg.258\)](#), [RandomTupleIndices\(I-18.10 pg.258\)](#), [reseed\(I-18.35 pg.270\)](#)

## I-18.4 randomize [OBSOLETE]

syntax

```
[OBSOLETE]
```

### Description

OBSOLETE: you can do “f := sum([random(-99,99)\*t | t in support(f)])” instead.

**See Also:** [random\(I-18.3 pg.256\)](#)

## I-18.5 randomized [OBSOLETE]

syntax

```
OBSOLETE
```

## Description

OBSOLETE: you can do “`sum([random(-99,99)*t | t in support(f)]);`” instead.

**See Also:** [random\(I-18.3 pg.256\)](#)

## I-18.6 RandomSparseNonSing01Mat

syntax

```
RandomSparseNonSing01Mat(R: RING, N: INT): MAT
```

## Description

This function returns a random “N”-by-“N” sparse matrix (with entries 0 or 1) having non-zero determinant.

example

```
/**/ RandomSparseNonSing01Mat(ZZ, 3);
matrix(ZZ,
  [[1, 1, 0],
   [0, 1, 0],
   [0, 0, 1]])
```

**See Also:** [random\(I-18.3 pg.256\)](#), [RandomUnimodularMat\(I-18.11 pg.259\)](#)

## I-18.7 RandomSubset

syntax

```
RandomSubset(L: LIST, K: INT): LIST
```

## Description

The function returns a random subset of “L” of cardinality “K”. This function can be quite useful for testing properties on some subsets of a large list when testing on all of them would be unfeasible in time and memory (see also “[subsets](#)” ([I-19.47 pg.303](#))).

NOTE: the resulting list is sorted as in “L”.

example

```
/**/ RandomSubset(["a","b","c","d","e","f","g","h"], 5);
["a", "c", "d", "f", "h"]

/**/ indent([RandomSubset(1..1000, 10) | i in 1..4]);
[
  [160, 182, 215, 219, 349, 588, 628, 811, 886, 905],
  [23, 103, 315, 451, 531, 539, 571, 846, 858, 876],
  [24, 230, 240, 278, 380, 421, 495, 505, 665, 788],
  [81, 274, 299, 378, 414, 616, 828, 844, 870, 946]
]

/**/ binomial(1000, 10); --> too many to fit in memory ;-
263409560461970212832400
```

**See Also:** [random\(I-18.3 pg.256\)](#), [subsets\(I-19.47 pg.303\)](#), [RandomSubsetIndices\(I-18.8 pg.258\)](#), [RandomTuple\(I-18.9 pg.258\)](#), [RandomTupleIndices\(I-18.10 pg.258\)](#)

## I-18.8 RandomSubsetIndices

syntax

```
RandomSubsetIndices(N: INT, K: INT): LIST
```

### Description

The function returns a random subset of “1..N” of cardinality “K”. See also “RandomSubset” ([I-18.7 pg.257](#)).

NOTE: the resulting list is sorted.

example

```
/**/ RandomSubsetIndices(10, 5);  
[1, 3, 4, 6, 8]
```

**See Also:** [random\(I-18.3 pg.256\)](#), [subsets\(I-19.47 pg.303\)](#), [RandomSubset\(I-18.7 pg.257\)](#), [RandomTuple\(I-18.9 pg.258\)](#), [RandomTupleIndices\(I-18.10 pg.258\)](#)

## I-18.9 RandomTuple

syntax

```
RandomTuple(L: LIST, K: INT): LIST
```

### Description

The function returns a random tuple of “L” of cardinality “K”. This function can be quite useful for testing properties on some tuples of a large list when testing on all of them would be unfeasible in time and memory (see also “tuples” ([I-20.15 pg.317](#))).

example

```
/**/ RandomTuple(["a","b","c","d","e","f","g","h"], 5);  
["b", "b", "h", "g", "e"]  
  
/**/ indent([RandomTuple(-9..9, 10) | i in 1..4]);  
[  
  [-5, -3, 1, 8, -4, 6, -5, -7, -1, 1],  
  [-5, -8, 0, -9, -2, -1, 3, -6, 6, -3],  
  [-2, -2, -9, -5, 0, -6, 2, -6, -5, -8],  
  [-2, -4, 4, 3, -3, 5, 3, -1, 8, -7]  
]
```

**See Also:** [random\(I-18.3 pg.256\)](#), [subsets\(I-19.47 pg.303\)](#), [RandomSubset\(I-18.7 pg.257\)](#), [RandomSubsetIndices\(I-18.8 pg.258\)](#), [RandomTupleIndices\(I-18.10 pg.258\)](#)

## I-18.10 RandomTupleIndices

syntax

```
RandomTupleIndices(N: INT, K: INT): LIST
```

### Description

The function returns a random tuple of “1..N” of cardinality “K”. See also “RandomTuple” ([I-18.9 pg.258](#)).

example

```
/**/ RandomTupleIndices(32003, 10);  
[4987, 13034, 10044, 7148, 11122, 1144, 21264, 5379, 2934, 7015]
```

**See Also:** [random\(I-18.3 pg.256\)](#), [subsets\(I-19.47 pg.303\)](#), [RandomSubset\(I-18.7 pg.257\)](#), [RandomSubsetIndices\(I-18.8 pg.258\)](#), [RandomTuple\(I-18.9 pg.258\)](#)

## I-18.11 RandomUnimodularMat

syntax

```
RandomUnimodularMat(R: RING, N: INT): MAT
RandomUnimodularMat(R: RING, N: INT, Nitters: INT): MAT
```

### Description

The function returns a random “N”-by-“N” matrix with integer entries, and determinant +1. The matrix is over the ring “R”. To obtain a unimodular matrix with determinant +1 or -1, just multiply the first row by -1 with probability 1/2.

The optional 3rd argument says how many internal iterations to perform: the algorithm starts with an identity matrix, then on each iteration simply adds or subtracts one random row to another random row. The default number of iterations is currently “25\*N”.

example

```
/**/ RandomUnimodularMat(ZZ, 3);
matrix(ZZ,
  [[-684, -2919, -769],
   [1054, 4498, 1185],
   [-519, -2215, -584]])
```

**See Also:** [random\(I-18.3 pg.256\)](#)

## I-18.12 rank [OBSOLESCENT]

syntax

```
[OBSOLESCENT]
```

### Description

See “rk” ([I-18.52 pg.278](#))

## I-18.13 RationalAffinePoints

syntax

```
RationalAffinePoints(L: LIST of RINGELEM): LIST of LIST of RINGELEM
```

### Description

This function returns the list of affine rational solutions (points) of a 0-dimensional polynomial system “L”. See also “RationalSolve” ([I-18.15 pg.260](#))

example

```
/**/ use QQ[x,y,z];
/**/ L := [x^3-y^2+z-1, x-2, (y-3)*(y+2)];
/**/ RationalAffinePoints(L);
[[2, -2, -3], [2, 3, 2]]
```

**See Also:** [LinSolve\(I-12.15 pg.185\)](#), [RationalSolve\(I-18.15 pg.260\)](#)

## I-18.14 RationalProjectivePoints

syntax

```
RationalProjectivePoints(L: LIST of RINGELEM): LIST of LIST of RINGELEM
```

### Description

This function returns the list of projective rational solutions (points) of a 0-dimensional polynomial system “L”. See also the function “RationalSolve” (I-18.15 pg.260)

example

```
/**/ use QQ[x,y,z];
/**/ L := [x^3-y^2*x, x-2*z];
/**/ RationalProjectivePoints(L);
[[0, 1, 0], [1, -1, 1/2], [1, 1, 1/2]]
```

**See Also:** LinSolve(I-12.15 pg.185), RationalSolve(I-18.15 pg.260)

## I-18.15 RationalSolve

syntax

```
RationalSolve(L: LIST of RINGELEM): LIST of LIST of RINGELEM
```

### Description

This function returns the list of rational solutions (points) of a 0-dimensional polynomial system “L” (see also “ApproxSolve” (I-1.15 pg.33)). Tries to be clever: if some indeterminates do not appear in “L” they are ignored, if the polynomials in “L” are homogeneous it returns the projective points.

example

```
/**/ use QQ[x,y,z];
/**/ L := [x^3-y^2+z-1, x-2, (y-3)*(y+2)];
/**/ RationalSolve(L);
[[2, -2, -3], [2, 3, 2]]

/**/ L := [x^3-y^2+z-1, x^2-2, (y-3)*(y+2)];
/**/ RationalSolve(L);
[]
/**/ len(ApproxSolve(L));
4

/**/ L := [x^3-y^2+1, (y-3)*(y+2)]; -- ignores the indeterminate z
/**/ RationalSolve(L);
[[2, 3]]
```

**See Also:** ApproxSolve(I-1.15 pg.33), LinSolve(I-12.15 pg.185), RationalAffinePoints(I-18.13 pg.259), RationalProjectivePoints(I-18.14 pg.260)

## I-18.16 RatReconstructByContFrac, RatReconstructByLattice

syntax

```
RatReconstructByContFrac(X: INT, M: INT): RECORD
RatReconstructByContFrac(X: INT, M: INT, threshold: INT): RECORD
RatReconstructByLattice(X: INT, M: INT): RECORD
RatReconstructByLattice(X: INT, M: INT, threshold: INT): RECORD
```

## Description

These functions attempt to reconstruct rational numbers from a modular image “ $X \bmod M$ ”.

The algorithms are “*fault-tolerant*”: they will succeed provided that “ $X$ ” is correct modulo a sufficiently large factor of “ $M$ ”.

The result is a record: the boolean field “*failed*” is “*true*” if no “*convincing*” result was found; otherwise it is “*false*”, and a second field, called “*ReconstructedRat*”, contains the value reconstructed.

An optional third argument, “*threshold*”, determines what “*convincing*” means: a higher value gives a more reliable answer, but may need a larger modulus before the answer is found.

There are two different underlying heuristic algorithms: a faster one based on continued fractions, and a slower one based on 2-dimensional lattice reduction. See the JSC paper by John Abbott: “Fault-tolerant modular reconstruction of rational numbers”, “<http://www.sciencedirect.com/science/article/pii/S0747717116300773>”; a near-final version is at “<http://arxiv.org/abs/1303.2965>”.

NOTE: so that the heuristic can work, the modulus must be a bit larger than strictly necessary; indeed, reconstruction will always fail if “ $M$ ” is too small.

### example

```
/**/ X := 333333333;
/**/ M := 10^10;
/**/ RatReconstructByContFrac(X,M);
record[ReconstructedRat := -1/3, failed := false]

/**/ X := 3141592654;
/**/ M := 10^10;
/**/ RatReconstructByContFrac(X,M);
record[failed := true]
```

**See Also:** RatReconstructWithBounds(I-18.18 pg.262), CRT(I-3.54 pg.67)

## I-18.17 RatReconstructPoly

### syntax

```
RatReconstructPoly(f1: RINGELEM, M1: INT): RINGELEM
```

## Description

This function attempts to reconstruct the rational coefficients of a polynomial “ $f$ ” from a modular image “ $f \bmod M$ ”. The algorithm is fault-tolerant: it will succeed provided that the coefficients in “ $f$ ” are correct modulo a sufficiently large factor of “ $M$ ”.

NOTE: so that the heuristic can work, the modulus must be larger than strictly necessary; indeed, reconstruction always fails if “ $M$ ” is small.

### example

```
/**/ use QQ[x,y];
/**/ RatReconstructPoly(-341269321*x^2 +255951993*y, 1023807973);
(10/3)*x^2 +(-1/4)*y

/**/ RingElem(NewPolyRing(NewRingFp(32003),"x,y"), "(10/3)*x^2 +(-1/4)*y");
10671*x^2 -8001*y
/**/ RingElem(NewPolyRing(NewRingFp(31991),"x,y"), "(10/3)*x^2 +(-1/4)*y");
10667*x^2 -7998*y

/**/ CRTPoly(10671*x^2 -8001*y, 32003, 10667*x^2 -7998*y, 31991);
record[modulus := 1023807973, residue := -341269321*x^2 +255951993*y]
```

**See Also:** CRTPoly(I-3.55 pg.68), RatReconstructByContFrac, RatReconstructByLattice(I-18.16 pg.260)

## I-18.18 RatReconstructWithBounds

syntax

```
RatReconstructWithBounds(e: INT, P: INT, Q: INT, res: LIST of INT, mod: LIST of INT): RECORD
```

### Description

This function attempts to reconstruct a rational number from a collection of residue-modulus pairs “(res[i],mod[i])”. The function also requires the input of three bounds: “e” is an upper bound on the number of bad moduli, and “P” and “Q” are upper bounds for (respectively the numerator and denominator of) the rational to be reconstructed.

The result is a record: the boolean field “failed” is “true” if no result exists; otherwise it is “false”, and a second field, called “ReconstructedRat”, contains the value reconstructed.

example

```
/**/ moduli := [11,13,15,17,19];
/**/ residues := [-2, -5, 0, 7, 4];
/**/ RatReconstructWithBounds(1,10,10,residues,moduli);
record[ReconstructedRat := 1/5, failed := false]

/**/ RatReconstructWithBounds(0,10,10,residues,moduli);
record[failed := true]
```

**See Also:** CRT(I-3.54 pg.67), RatReconstructByContFrac, RatReconstructByLattice(I-18.16 pg.260)

## I-18.19 ReadExpr [OBSOLESCENT]

syntax

```
ReadExpr(R: RING, expr: STRING): RINGELEM
RingElem(R: RING, expr: STRING): RINGELEM
```

### Description

This function reads a “RINGELEM” expression from a “STRING”.

NOTE: from version 5.2.0 “RingElem” (I-18.43 pg.273) does the same, and more.

**See Also:** RingElem(I-18.43 pg.273), sprint(I-19.29 pg.294)

## I-18.20 RealRootRefine

syntax

```
RealRootRefine(Root: RECORD, Precision: RAT): RECORD
```

### Description

This function computes a refinement of a real root of a univariate polynomial over QQ to the desired precision (width of isolating interval). The starting root must be a record produced by “RealRoots” (I-18.21 pg.263).

example

```
/**/ use QQ[x];
/**/ RR := RealRoots(x^2-2);
/**/ RealRootRefine(RR[1], 1/2);
```

```
record[CoeffList := [-1, 0, 2], inf := -3/2, sup := -5/4]

/**/ RR := [RealRootRefine(Root, 10^(-20)) | Root in RR];
/**/ FloatStr(RR[1].inf);
-1.414213562*10^0
```

**See Also:** [RealRoots\(I-18.21 pg.263\)](#), [RealRootsApprox\(I-18.22 pg.264\)](#), [RootBound\(I-18.54 pg.278\)](#)

## I-18.21 RealRoots

— syntax —

```
RealRoots(F: RINGELEM): LIST
RealRoots(F: RINGELEM, Precision: RAT): LIST
RealRoots(F: RINGELEM, Precision: RAT, Interval:[RAT, RAT]): LIST
```

### Description

This function computes isolating intervals for the real roots of a univariate polynomial over  $\mathbb{Q}\mathbb{Q}$ . It returns the list of the real roots, where a root is represented as a record containing either the exact root (if the fields “**inf**” and “**sup**” are equal), or an open interval (inf, sup) containing the root. A third field (called **CoeffList**) has an obscure meaning.

An optional second argument specifies the maximum width an isolating interval may have. An optional third argument specifies a closed interval in which to search for roots.

The interval represented by a root record may be refined by using the function “**RealRootRefine**” ([I-18.20 pg.262](#)). The function “**RealRootsApprox**” ([I-18.22 pg.264](#)) may be more useful to you: it produces rational approximations to the real roots (but these cannot later be refined).

— example —

```
/**/ use QQ[x];
/**/ indent(RealRoots(x^2-2));
[
  record[CoeffList := [-1, 0, 2], inf := -4, sup := 0],
  record[CoeffList := [1, 0, -2], inf := 0, sup := 4]
]

/**/ RR := RealRoots((x^2-2)*(x-1), 10^(-5));
/**/ FloatStr(RR[1].inf); -- left end of interval
-1.414213562*10^0

/**/ FloatStr(RR[1].sup); -- right end of interval
-1.414213561*10^0

/**/ RR := RealRoots(x^2-2, 10^(-20), [0, 2]);

/**/ RR[1].inf; -- incomprehensible
60153992292001127886258443119406264231/42535295865117307932921825928971026432
/**/ FloatStr(RR[1].inf, 20); -- comprehensible
1.4142135623730950488*10^0
```

**See Also:** [RealRootRefine\(I-18.20 pg.262\)](#), [RealRootsApprox\(I-18.22 pg.264\)](#), [RootBound\(I-18.54 pg.278\)](#)

## I-18.22 RealRootsApprox

syntax

```
RealRootsApprox(F: RINGELEM): LIST
RealRootsApprox(F: RINGELEM, Precision: RAT): LIST
RealRootsApprox(F: RINGELEM, Precision: RAT, Interval:[RAT, RAT]): LIST
```

### Description

This function computes rational approximations to the real roots of a univariate polynomial (with rational coefficients).

An optional second argument specifies the maximum separation between the approximations produced and the corresponding exact root. An optional third argument specifies a closed interval in which to search for roots.

example

```
/**/ use QQ[x];
/**/ RealRootsApprox(x^2-2);
[-3037000499/2147483648, 3037000499/2147483648]

/**/ RR := RealRootsApprox(x^2-2, 10^(-15), [0, 2]);
/**/ RR;
[6521908912666391107/4611686018427387904]

/**/ FloatStr(RR[1], 15);
1.41421356237310*10^0
```

**See Also:** [RealRoots\(I-18.21 pg.263\)](#), [RootBound\(I-18.54 pg.278\)](#)

## I-18.23 record

syntax

```
record[F_1 := OBJECT,..., F_n := OBJECT]
  where each F_i is a field name
  returns RECORD
```

### Description

This constructor creates a record with fields called “F<sub>1</sub>”, ..., “F<sub>n</sub>”. The empty record is given by “`record[]`”.

Records in CoCoA are “*open*” in the sense that new fields may be added after the record is first defined. The names allowed for the fields are the same as those allowed for variables.

The dot operator is used to access the fields in a record.

example

```
/**/ P := record[height := 10, width := 5];
/**/ P.height * P.width;
50

/**/ P.area := It; --> creates a new field called "area"
/**/ P;
record[area := 50, height := 10, width := 5]
```

**See Also:** [record field selector\(I-18.24 pg.265\)](#), [fields\(I-6.7 pg.96\)](#)

## I-18.24 record field selector

syntax

```
R.FieldName
R["FieldName"]
  where R is a RECORD
```

### Description

A record is a data structure containing named entries. They are created using the command “**record**” (I-18.23 pg.264). Each entry may be selected using the “dot operator”, or equivalently a string index.

example

```
/**/ rec := record[name := "David", year := 1961];
/**/ rec.name;
David

/**/ rec.year := 1849;           --> change value of a field
/**/ rec.surname := "Copperfield"; --> create a new field
/**/ rec["year"]; -- alternative syntax
1849

/**/ foreach F in fields(rec) do print rec[F]; endforeach;
DavidCopperfield1849
```

**See Also:** [record\(I-18.23 pg.264\)](#)

## I-18.25 ReducedGBasis

syntax

```
ReducedGBasis(I: IDEAL): LIST of RINGELEM
ReducedGBasis(I: MODULE): LIST of MODULEELEM
```

### Description

This function returns a list whose components form a reduced Groebner basis for the ideal (or module) “I” with respect to the term-ordering of the polynomial ring of “I”.

example

```
/**/ use R ::= QQ[x,y];
/**/ I := ideal(x^4-x^2, x^3-y);
/**/ ReducedGBasis(I);
[x*y -y^2, x^2 -y^2, y^3 -y]
```

**See Also:** [GBasis\(I-7.1 pg.107\)](#)

## I-18.26 ref

syntax

```
ref X
  where X is the identifier of a CoCoA variable.
```

## Description

The keyword “**ref**” is used to pass a parameter “by reference” to a function which may modify its value (e.g. “**append**” (I-1.12 pg.31)). The keyword “**ref**” alerts the programmer to the possibility that the value may be changed during the call.

To write a new function which can modify some parameters use the same keyword “**ref**” to identify which formal parameters are to be passed by reference. The following example illustrates the difference between passing by reference and passing by value.

### example

```

/**/ Define CallByRef(ref L) -- "call by reference": The variable referred
/**/   L := "new value";      -- to by L is changed.
/**/ EndDefine;
/**/ M := "old value";
/**/ CallByRef(ref M); -- here "ref" recalls that M might change
/**/ PrintLn M;
new value

/**/ Define CallByVal(L) -- "call by value": The value of L is passed to
/**/   L := "new value"; -- the function.
/**/   Return L;
/**/ EndDefine;
/**/ L := "old value";
/**/ CallByVal(L);
new value

/**/ PrintLn L;
old value

```

**See Also:** [define\(I-4.4 pg.72\)](#)

## I-18.27 RefineGCDFreeBasis

### syntax

```
RefineGCDFreeBasis(B: LIST of INT, N: INT): LIST of INT
```

## Description

This function computes a refined GCD free basis by adjoining a given integer to it. The value returned is [NewB, N2] where NewB is the refined basis and N2 is the part of N coprime to every element of B.

### example

```

/**/ B := GCDFreeBasis([Fact(10), binomial(20,10)]); B;
[14175, 4, 46189]

/**/ RefineGCDFreeBasis(B, 15);
[[7, 3, 5, 4, 46189], 1]

```

**See Also:** [GCDFreeBasis\(I-7.5 pg.109\)](#)

## I-18.28 reg

### syntax

```

reg(I: IDEAL): INT
reg(R: (Quotient)RING): INT

```

## Description

These functions computes the Castelnuovo-Mumford regularity of an ideal. The implementation of “Reg” using Bermejo-Gimenez Algorithm.

Implemented by Eduardo Saenz-de-Cabezón (updated to CoCoA-5 by Anna M. Bigatti).

NOTE: this is different from “RegularityIndex” (I-18.29 pg.267), the regularity of a Hilbert Function.

### example

```

/**/ use R := QQ[x,y,z];
/**/ I := ideal(x^3, y^2);
/**/ reg(I);
4
/**/ reg(R/I);
3
/**/ PrintRes(I);
0 --> R(-5) --> R(-2)(+)R(-3)

/**/ PrintBettiDiagram(I);
      0      1
-----
2:    1      -
3:    1      -
4:    -      1
-----
Tot:   2      1

```

**See Also:** `res`(I-18.34 pg.269), `PrintRes`(I-16.35 pg.248), `PrintBettiDiagram`(I-16.31 pg.246), `PrintBettiMatrix`(I-16.32 pg.246), `RegularityIndex`(I-18.29 pg.267)

## I-18.29 RegularityIndex

### syntax

```
RegularityIndex(R: RING or TAGGED("Quotient")): INT
```

## Description

This function computes the regularity index of a Hilbert function. The input might be expressed as a Hilbert function or as the corresponding Hilbert series (computed with standard weights).

### example

```

/**/ use R := QQ[x,y,z];
/**/ Quot := R/ideal(x^3, y^2);
/**/ HilbertFn(Quot);
H(0) = 1
H(1) = 3
H(2) = 5
H(t) = 6 for t >= 3
/**/ RegularityIndex(HilbertFn(Quot));
3
/**/ RegularityIndex(HilbertSeries(Quot));
3

```

**See Also:** `HilbertFn`(I-8.7 pg.123), `HilbertSeries`(I-8.10 pg.125)

### I-18.30 RelNotes

— syntax —

```
RelNotes()
RelNotes(N: INT)
```

#### Description

This function prints the release notes of the version you are running, of all versions, or of last “N” versions.

— example —

```
/**/ RelNotes(); --> last version
/**/ RelNotes(1); --> last version
/**/ RelNotes(3); --> last 3 versions
/**/ RelNotes(0); --> all versions
```

**See Also:** [VersionInfo\(I-22.3 pg.330\)](#)

### I-18.31 ReloadMan

— syntax —

```
ReloadMan()
```

#### Description

This function reloads the xml source of the manual “CoCoAHelp.xml” (in directory “CoCoAManual”) and recreates the internal manual index in a running CoCoA-5 (instead of closing and re-opening CoCoA...).

It is useful “*only for developers*” working on the manual and making substantial changes to “CoCoAHelp.xml”.

After adding a new entry the index needs updating, but if the change is just in the description of an existing entry (so the internal index is still valid) there is no need to reload the manual: the description is always searched in the current file.

— example —

```
/**/ ReloadMan();
```

### I-18.32 remove

— syntax —

```
remove(ref L:LIST, N: INT)
```

#### Description

This function removes the “N”-th component from “L”; it changes the value of “L”. Use the function “WithoutNth” ([I-23.4 pg.332](#)) to create a new list containing the elements of “L” except the “N”-th (without changing “L”).

— example —

```
/**/ use R := QQ[x,y,z];
/**/ L := indets(R);
/**/ L;
[x, y, z]
```

```

/**/  remove(ref L,2);
/**/  L;
[x, z]

```

**See Also:** WithoutNth(I-23.4 pg.332)

## I-18.33 repeat

— syntax —

```

repeat C until B
repeat C endrepeat
(where C is a sequence of commands and B is BOOL)

```

### Description

In the first form, the command sequence “C” is repeated until “B” evaluates to “false”. Unlike the “while” (I-23.3 pg.332) command, “C” is executed at least once. Note that there is no “endrepeat” following “B”.

— example —

```

/**/      Define GCD_Euclid(A, B)
/**/      Repeat
/**/          R := mod(A, B);
/**/          A := B;
/**/          B := R;
/**/      Until B = 0;
/**/      Return A;
/**/      EndDefine;

/**/  GCD_Euclid(6,15);
3

/**/  N := 0;
/**/  repeat
/**/      N := N+1;
/**/      PrintLn N;
/**/      If N > 5 Then Break; EndIf;
/**/  endrepeat;
1
2
3
4
5

```

**See Also:** for(I-6.17 pg.100), foreach(I-6.18 pg.101), syntax(?? pg.??), while(I-23.3 pg.332)

## I-18.34 res

— syntax —

```

res(M: IDEAL): LIST
res(M: MODULE): LIST

```

## Description

This function returns the minimal free resolution of “M”. “res” only works in the homogeneous context, and the coefficient ring must be a field.

NOTE: the current implementation (CoCoA-5.1.0) is very naive so it might be very slow (better slow than nothing?).

example

```

/**/ use R ::= QQ[x,y,z];
/**/ I := ideal(x, y, z^2);
/**/ PrintRes(R/I);
0 --> R(-4) --> R(-2)(+)R(-3)^2 --> R(-1)^2(+)R(-2) --> R
/**/ indent(Res(R/I),2);
[
  RingWithID(20, "RingWithID(3)/ideal(x, y, z^2)"),
  ideal(
    y,
    x,
    z^2
  ),
  SubmoduleRows(F, matrix([
    [x, -y, 0],
    [0, z^2, -x],
    [z^2, 0, -y]
  ])),
  SubmoduleRows(F, matrix([
    [z^2, y, -x]
  ]))
]
```

**See Also:** [PrintBettiDiagram\(I-16.31 pg.246\)](#), [PrintBettiMatrix\(I-16.32 pg.246\)](#), [PrintRes\(I-16.35 pg.248\)](#)

## I-18.35 reseed

syntax

```
reseed(N: INT): VOID
```

## Description

CoCoA offers a pseudo-random number generator (see “random” ([I-18.3 pg.256](#))); however, each time you start CoCoA it will produce the same random values. The procedure “reseed” puts the pseudo-random number generator into a state determined solely by the given seed value “N”. This procedure can be used to make CoCoA generate different random values on different runs, e.g. if reseeded with a value which depends on the current time.

example

```

/**/ reseed(123);
/**/ random(0,9);
7
/**/ reseed(1000000*date()+TimeOfDay()); --> time-dependent seed value
```

**See Also:** [random\(I-18.3 pg.256\)](#)

## I-18.36 Reset [OBSOLETE]

[OBSOLETE]

— syntax —

### Description

[OBSOLETE]

## I-18.37 ResetPanels [OBSOLETE]

[OBSOLETE]

— syntax —

### Description

[OBSOLETE]

## I-18.38 resultant

— syntax —

```
resultant(F: RINGELEM, G: RINGELEM, X: RINGELEM): RINGELEM
```

### Description

This function returns the resultant of the polynomials F and G with respect to the indeterminate X.

— example —

```
/**/ use R ::= QQ[p,q,x];
/**/ F := x^3+p*x-q;   G := deriv(F, x);
/**/ resultant(F, G, x);
4*p^3 +27*q^2
```

**See Also:** [discriminant\(I-4.18 pg.80\)](#), [sylvestre\(I-19.54 pg.306\)](#)

## I-18.39 return

— syntax —

```
return
return E
```

### Description

This command is used to exit from a procedure/function. The second form returns the value of the expression E to the user. As a safety measure all “**return**”s in a function/procedure must be of the same kind: either they all return a value (function) or none of them returns a value (procedure).

“**return**” has immediate effect even inside loops; compare with “**break**” ([I-2.12 pg.43](#)).

example

```

/**/ Define Rev(L) -- reverse a list
/**/   If len(L) < 2 Then Return L; EndIf;
/**/   M := Rev(Tail(L)); -- recursive function call
/**/   append(ref M, L[1]);
/**/   Return M;
/**/ EndDefine;

/**/ Rev([1,2,3,4]);
[4, 3, 2, 1]

```

**See Also:** [break\(I-2.12 pg.43\)](#), [define\(I-4.4 pg.72\)](#), [syntax\(?? pg.??\)](#)

## I-18.40 reverse, reversed

syntax

```

reverse(ref L: LIST)
reversed(L: LIST): LIST

```

### Description

The the function “**reverse**” reverses the order of the elements of the list in “L”; it changes the value of “L” and returns nothing. The function “**reversed**” returns the reversed list without changing “L”.

example

```

/**/ L := [1,2,3,4];
/**/ reverse(ref L);
/**/ L; -- L has been modified
[4, 3, 2, 1]

/**/ M := [1,2,3,4];
/**/ reversed(M); -- the reversed list is returned
[4, 3, 2, 1]

/**/ M; -- M has not been modified
[1, 2, 3, 4]

```

**See Also:** [sort\(I-19.22 pg.291\)](#), [sorted\(I-19.24 pg.292\)](#)

## I-18.41 RevLexMat

syntax

```

RevLexMat(N: INT): MAT

```

### Description

This function return the matrix defining a standard ordering (which is not a term-ordering!).

example

```

/**/ RevLexMat(3);
matrix(ZZ,
  [[0, 0, -1],
   [0, -1, 0],
   [-1, 0, 0]])

```

**See Also:** OrdMat(I-15.10 pg.231), Term Orderings(III-9.5 pg.404), StdDegRevLexMat(I-19.38 pg.299), StdDegLexMat(I-19.37 pg.299), LexMat(I-12.7 pg.182), XelMat(I-24.1 pg.335)

## I-18.42 rgin

syntax

```
rgin(I: IDEAL): IDEAL
```

### Description

These functions return the [probabilistic] rgin (generic initial ideal wrt StdDegRevLex) of the ideal “I”.

See “gin” (I-7.28 pg.116) for details and verbosity.

example

```
/**/ use R := QQ[x,y,z], DegLex;
/**/ I := ideal(y^2-x*z, x^2*z-y*z^2);
/**/ gin(I);
ideal(x^2, x*y^2, x*y*z^2, x*z^4, y^6)

/**/ rgin(I);
ideal(x^2, x*y^2, y^4)
```

**See Also:** gin(I-7.28 pg.116), LT(I-12.21 pg.188)

## I-18.43 RingElem

syntax

```
RingElem(R: RING, E: STRING): RINGELEM
RingElem(R: RING, E: RINGELEM): RINGELEM
RingElem(R: RING, E: INT): RINGELEM
RingElem(R: RING, E: RAT): RINGELEM
RingElem(R: RING, E:[STRING, INT, .., INT]): RINGELEM
```

### Description

This function converts the expression “E” into a RINGELEM in “R”, if possible. useful for operating with different rings.

If “E” is a “STRING” it reads “E” in “R”, (with no need for “use R”). The expression “E” may contain operations and parentheses, but no programming variables nor function calls. New from version 5.2.0; was previously called “ReadExpr” (?? pg.??).

If “E” is a “RINGELEM” it is equivalent to applying the “CanonicalHom” (I-3.3 pg.46) from “RingOf(E)” to “R” (for other homomorphisms see “RINGHOM” (III-10 pg.409)).

example

```
/**/ P := QQ[x,y]; S := QQ[x,y,z[1..4,3..7]]; K := NewFractionField(P);
/**/ QR := NewQuotientRing(P, ideal(RingElem(P,"x^2-3"), RingElem(P,"y^2-5")));

/**/ -- STRING
/**/ 7*RingElem(P, "x"); --> x as an element of P
2*x
/**/ 7*RingElem(S, "x"); --> x as an element of S
7*x
/**/ RingElem(S, "((7/3)*z[2,5] - 1)^2"); -- expr without function calls
49*z[2,5]^2 -14*z[2,5] +1
```

```

/**/ RingElem(K, "(x^2-x*y)/(x*y-y^2)");
x/y
/**/ f := RingElem(S, "(x+y)^3"); f;
x^3 +3*x^2*y +3*x*y^2 +y^3
/**/ RingElem(QR, sprint(f));
(18*x +14*y)
/**/ RingElem(NewPolyRing(NewRingFp(3), "x,y"), sprint(f));
x^3 +y^3

/**/ -- RINGELEM (via CanonicalHom)
/**/ use P;
/**/ f := 2*x-3;
-- /**/ f/LC(f); -- !!! ERROR !!! as expected: LC(F) in CoeffRing(P)
/**/ f/RingElem(P,LC(f));
x +1
-- /**/ 1/f; -- !!! ERROR !!! as expected: f in P is not invertible
/**/ 1/RingElem(K, f); -- f in K is invertible
1/x

/**/ use P ::= ZZ/(5)[x,y,z];
/**/ -- INT and RAT
/**/ RingElem(P, 7);
2
/**/ RingElem(P, 3/2);
-1

/**/ -- LIST
/**/ i := 2; j := 5; 7*RingElem(S, ["z",i,j]);
7*z[2,5]

```

**See Also:** RingOf(I-18.46 pg.275), AsINT(I-1.19 pg.35), AsRAT(I-1.20 pg.36), IndetName(I-9.23 pg.144), IndetSubscripts(I-9.25 pg.145), CanonicalHom(I-3.3 pg.46)

## I-18.44 RingEnv [OBSOLETE]

syntax

[OBSOLETE]

### Description

See “RingOf” (I-18.46 pg.275).

**See Also:** RingOf(I-18.46 pg.275)

## I-18.45 RingID

syntax

RingID(R: RING): INT

### Description

This function returns the identification number of the ring “R”. Two rings are considered equal if and only if they have the same ID: this means they have the same internal implementation.

This function was called “ID” in CoCoA-5.1.1.

example

```

/**/ R := QQ[x,y,z];
/**/ R;
RingWithID(9, "QQ[x,y,z]")
/**/ S := QQ[x,y,z];
/**/ R = S;
false
/**/ RingID(R);
7
/**/ RingID(S);
8
-- /**/ RingElem(R,"x") = RingElem(S, "x"); --> !!! ERROR !!! mixed rings

/**/ S := R; -- or S := RingOf( some element in R )
/**/ R = S;
true

```

**See Also:** [print\(I-16.29 pg.245\)](#), [println\(I-16.34 pg.247\)](#), [Evaluation and Assignment\(II-4 pg.349\)](#)

## I-18.46 RingOf

syntax

```
RingOf(E: RINGELEM|IDEAL|MAT|MODULE): RING
```

### Description

This function returns the ring on which the object “E” is defined.

NOTE: A ring contains many information and two separate rings, even when defined with the same commands, are not “equal”. Also, when a ring is printed only limited information is shown, so different rings might print out the same

example

```

/**/ use R := QQ[x,y,z];
/**/ I := ideal(x,y);
/**/ RingOf(I);
RingWithID(6, "QQ[x,y,z]")

/**/ RingOf(mat([[1,2],[3,4]]));
QQ

/**/ use Qabc := QQ[a,b,c];
/**/ F := a^2+b;
/**/ G := a*b+b^2;
/**/ use S := ZZ/(3)[x,y];
/**/ RingOf(F+G); -- F+G is computed in the ring of definition
RingWithID(7, "QQ[a,b,c]")
/**/ indets(RingOf(F));
[a, b, c]

```

**See Also:** [CurrentRing\(I-3.56 pg.68\)](#), [RingsOf\(I-18.50 pg.277\)](#), [BaseRing\(I-2.1 pg.37\)](#)

## I-18.47 RingQQ

syntax

`RingQQ(): RING`

### Description

This function returns the ring of rationals. It is particularly useful when you want to use “QQ” (which is a pre-defined top-level “*variable*”) inside a function.

NOTE: calling “RingQQ” twice gives the same identical ring, whereas calling “NewPolyRing” (I-14.8 pg.212) and “NewFractionField” (I-14.1 pg.209) return each time a new ring.

example

```

/**/  QQ = RingQQ();

/**/  Two := RingElem(RingQQ(), 2);  Two;
2
/**/  type(Two);
RINGELEM;
/**/  IsQQ(RingOf(Two));
true

```

**See Also:** TopLevel(I-20.10 pg.314), QQ(I-17.1 pg.251), RingQQt(I-18.48 pg.276), RingZZ(I-18.51 pg.277)

## I-18.48 RingQQt

syntax

`RingQQt(N: INT): RING`

### Description

This function returns a polynomial ring over “QQ” with indeterminates  $t[1] \dots t[N]$ . In particular “RingQQt(1)” is the polynomial ring in which Hilbert polynomials are defined.

NOTE: calling “RingQQt(5)” twice gives the same identical ring, whereas calling “NewPolyRing(RingQQ(), SymbolRange(“

example

```

/**/  QQt := RingQQt(3);  use QQt;
/**/  (t[1]+1)^3;
t[1]^3 +3*t[1]^2 +3*t[1] +1
/**/  indets(RingQQt(1));
[t]
/**/  indets(RingQQt(5));
[t[1], t[2], t[3], t[4], t[5]]

```

**See Also:** TopLevel(I-20.10 pg.314), RingQQ(I-18.47 pg.276), RingZZ(I-18.51 pg.277)

## I-18.49 RingSet [OBSOLETE]

syntax

`[OBSOLETE]`

## Description

This function has just been renamed “RingsOf” ([I-18.50 pg.277](#)) **See Also:** [RingsOf\(I-18.50 pg.277\)](#)

## I-18.50 RingsOf

syntax

RingsOf(E: LIST|RINGELEM|IDEAL|MAT|MODULE|MODULEELEM): LIST of RING and TYPE

## Description

This function returns the list of the RINGS (or types, if not dependent from a RING) on which the object E is dependent. Similar to “RingOf” ([I-18.46 pg.275](#)), this function also works on lists and returns the set of ring environments of all entries. ...needless to say that it may be quite slow on big inputs!

example

```

/**/ use R ::= QQ[x,y,z];
/**/ L1 := [x, y];
/**/ L2 := [x, y, 0, 5/4];
/**/ Z3 := NewRingFp(3);
/**/ use S ::= Z3[a,b];
/**/ RingsOf(L1);
[RingDistrMPolyClean(QQ, 3)]

/**/ RingsOf(L2);
[RingDistrMPolyClean(QQ, 3), INT, RAT]

/**/ RingsOf([L2, a+b]);
[RingDistrMPolyClean(QQ, 3), INT, RAT, RingDistrMPolyClean(FFp(3), 2)]

```

**See Also:** [CurrentRing\(I-3.56 pg.68\)](#), [RingOf\(I-18.46 pg.275\)](#)

## I-18.51 RingZZ

syntax

RingZZ(): RING

## Description

This function returns the ring of integers. It is useful when you want to use “ZZ” ([I-25.4 pg.338](#)) inside “define/enddefine”.

NOTE: calling “RingZZ” twice gives the same identical ring, whereas calling “NewPolyRing” ([I-14.8 pg.212](#)) or “NewFractionField” ([I-14.1 pg.209](#)) return each time a new ring.

example

```

/**/ Two := RingElem(RingZZ(), 2); Two;
2
/**/ type(Two);
RINGELEM;
/**/ IsZZ(RingOf(Two));
true
/**/ IsQQ(RingOf(Two));
false

```

**See Also:** [TopLevel\(I-20.10 pg.314\)](#), [RingQQt\(I-18.48 pg.276\)](#), [RingQQ\(I-18.47 pg.276\)](#), [ZZ\(I-25.4 pg.338\)](#)

## I-18.52 rk

syntax

```
rk(M: MAT): INT
rk(M: MODULE): INT
```

### Description

This function computes the rank of “M”. For a module “M” this is defined as the vector space dimension of the subspace generated by the generators of “M” over the quotient field of the base ring – contrast this with the function “NumCompts” ([I-14.35 pg.223](#)) which simply counts the number of components the module has.

example

```
/**/ use R := QQ[x,y,z];
/**/ rk(IdentityMat(R, 4));
4

rk(Module([x,y,z,0])); --***WORK IN PROGRESS***
1
-----
rk(Module([[1,2,3],[2,4,6]])); --***WORK IN PROGRESS***
1
-----
rk(Module([[1,2,3],[2,5,6]])); --***WORK IN PROGRESS***
2
-----
```

## I-18.53 RMap [OBSOLESCENT]

syntax

```
[OBSOLESCENT] RMap(L: LIST): TAGGED("RMap")
```

### Description

[OBSOLESCENT] related with “image [OBSOLESCENT]” ([I-9.12 pg.138](#)). In CoCoA-5 such homomorphisms are properly implemented as “PolyAlgebraHom” ([I-16.14 pg.237](#)).

## I-18.54 RootBound

syntax

```
RootBound(F: RINGELEM): RAT
RootBound(F: RINGELEM, N: INT): RAT
RootBound_Birkhoff(F: RINGELEM): RAT
RootBound_Cauchy(F: RINGELEM): RAT
RootBound_Lagrange(F: RINGELEM): RAT
RootBound_LMS(F: RINGELEM): RAT
```

## Description

The function “**RootBound**” computes a bound on the absolute values of the complex roots of a univariate polynomial with rational coefficients. In some cases you may get a better bound by applying first the transformation produced by the function “**LinearSimplify**”. The optional second argument says how many iterations of Graeffe’s transformation to apply; high numbers give better bounds but can take significantly more time. With just one argument, the number of iterations is determined heuristically.

The functions “**RootBound\_Birkhoff**”, “**RootBound\_Cauchy**”, “**RootBound\_Lagrange**” and “**RootBound\_LMS**” compute those bounds directly. You should normally use the function “**RootBound**” which computes all the bounds, and takes the smallest; it may also apply some Graeffe transformations.

example

```
/**/ use P := QQ[x];
/**/ RootBound(x^2-2);
363/256
```

**See Also:** [graeffe\(I-7.31 pg.118\)](#), [LinearSimplify\(I-12.10 pg.183\)](#), [RealRootRefine\(I-18.20 pg.262\)](#), [RealRoots\(I-18.21 pg.263\)](#)

## I-18.55 *round*

syntax

```
round(X: RAT): INT
```

## Description

This function rounds a rational to the nearest integer; halves are rounded towards zero.

example

```
/**/ round(4.56);
5

/**/ round(-1/2);
0
```

**See Also:** [num\(I-14.33 pg.223\)](#), [den\(I-4.7 pg.75\)](#), [floor\(I-6.14 pg.99\)](#), [ceil\(I-3.7 pg.48\)](#)

## I-18.56 *RowMat*

syntax

```
RowMat(L: LIST): MAT
RowMat(R: RING, L: LIST): MAT
```

## Description

This function return the matrix whose only row consists of the elements of the list “L”.

The first form produces a matrix over “QQ” if all entries in “L” are of type “INT” or “RAT”. If “L” contains any entries of type RINGELEM then the matrix is over the ring these elements belong to.

The second form produces a matrix over “R”, and requires that the elements of “L” be “INT”, “RAT” or “RINGELEM” belonging to “R”.

example

```
/**/ RowMat([3,4,5]);
matrix(QQ,
  [[3, 4, 5]])
```

```
/**/ RowMat(QQ,[5,6,7]);  
matrix(QQ,  
  [[5, 6, 7]])
```

**See Also:** BlockMat([I-2.9](#) pg.41), DiagMat([I-4.15](#) pg.79), ColMat([I-3.29](#) pg.57)

# Chapter I-19

## S

### I-19.1 saturate

syntax

```
saturate(I: IDEAL, J: IDEAL): IDEAL
```

#### Description

This function returns the saturation of I with respect to J: the ideal of polynomials F such that  $F \cdot G$  is in I for all G in  $J^d$  for some positive integer d.

The coefficient ring must be a field.

example

```
/**/ use R ::= QQ[x,y,z];
/**/ I := ideal(x-z, y-2*z);
/**/ J := ideal(x-2*z, y-z);
/**/ K := intersection(I, J); -- ideal of two points in the
                             -- projective plane
/**/ L := intersection(K, ideal(x,y,z)^3); -- add an irrelevant component
/**/ HilbertFn(R/L);
H(0) = 1
H(1) = 3
H(2) = 6
H(t) = 2 for t >= 3

/**/ saturate(L, ideal(x,y,z)) = K; -- saturating gets rid of the
                                   -- irrelevant component
true
```

**See Also:** [colon\(I-3.30 pg.57\)](#), [HColon\(I-8.2 pg.121\)](#), [HSaturation\(I-8.15 pg.128\)](#)

### I-19.2 ScalarProduct

syntax

```
ScalarProduct(L, M): OBJECT
  where each of L and M is of type MODULEELEM or LIST
```

#### Description

This function returns the sum of the product of the components of L and M; precisely  $(\text{len}(L)=\text{len}(M))$ :

$\text{ScalarProduct}(L, M) = \sum([L[I]*M[I] \mid I \text{ in } 1..\text{len}(L)])$ .

The function works whenever the product of the components of L and M are defined (see “Algebraic Operators” (II-3.2 pg.347)).

example

```
/**/ ScalarProduct([1,2,3], [5,0,-1]);
2

use R := QQ[x,y];
ScalarProduct([ideal(x,y), ideal(x^2-xy)], [x^2,y]);
ideal(x^3, x^2y, x^2y - xy^2)
-----
```

**See Also:** Algebraic Operators(II-3.2 pg.347)

## I-19.3 ScientificStr

syntax

```
ScientificStr(X: INT|RAT|RINGELEM): STRING
ScientificStr(X: INT|RAT|RINGELEM, Prec: INT): STRING
```

### Description

This function converts a rational number “X” into a (decimal) floating-point string. The optional second argument “Prec” says how many decimal digits to include in the mantissa; the default value is 5. Note that an exponent is always included.

example

```
/**/ ScientificStr(2/3);      -- last printed digit is rounded
6.6667*10^(-1)

/**/ ScientificStr(7^510);    -- no arbitrary limit on exponent range
1.0000*10^431

/**/ ScientificStr(1/81, 50); -- precision of mantissa specified by user
1.2345679012345679012345679012345679012345679012345679012346*10^(-2)

/**/ ScientificStr(1/2);      -- trailing zeroes are not suppressed
5.0000*10^(-1)
```

**See Also:** DecimalStr(I-4.3 pg.71), FloatStr(I-6.13 pg.98), FloatApprox(I-6.12 pg.98), MantissaAndExponent10(I-13.6 pg.193)

## I-19.4 SectionalMatrix

syntax

```
SectionalMatrix(I: IDEAL): type MAT
SectionalMatrix(PmodI: RING): type MAT
SectionalMatrix(I: IDEAL, bound:INT): type MAT
SectionalMatrix(PmodI: RING, bound:INT): type MAT
```

### Description

The definition of Hilbert function was extended in “Borel Sets and Sectional Matrices” to the bivariate function encoding the Hilbert functions of the generic hyperplane sections: the sectional matrix of “I” (I-9 pg.131) (homogenous ideal) or “PmodI” (?? pg.??).

The second argument makes a matrix with “bound” columns. The default value is “**reg(I)**” (since the rest of the matrix is obtained by the Persistence Theorem).

See articles Bigatti, Robbiano, “Borel Sets and Sectional Matrices”, and Bigatti, Palezzato, Torielli, “Sectional Matrices” (work in progress).

example

```

/**/ use P := QQ[x,y,z];
/**/ I := ideal(x^4 -x*y^3, x*y -z^2, x*z^2 -y^3);
/**/ SectionalMatrix(I);
matrix(ZZ,
  [[0, 0, 1, 1, 1, 1, 1, 1],
   [0, 0, 1, 3, 5, 6, 7, 8],
   [0, 0, 1, 4, 10, 18, 26, 34]])
/**/ SectionalMatrix(P/I);
matrix(ZZ,
  [[1, 1, 0, 0, 0, 0, 0, 0],
   [1, 2, 2, 1, 0, 0, 0, 0],
   [1, 3, 5, 6, 5, 3, 2, 2]])

/**/ SectionalMatrix(P/I, 10);
matrix(ZZ,
  [[1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
   [1, 2, 2, 1, 0, 0, 0, 0, 0, 0],
   [1, 3, 5, 6, 5, 3, 2, 2, 2, 2]])

```

## I-19.5 *seed* [OBSOLETE]

syntax

[OBSOLETE]

### Description

Replaced by “**reseed**” ([I-18.35](#) pg.270).

## I-19.6 *SeparatorsOfPoints*

syntax

**SeparatorsOfPoints**(Points: LIST): LIST

where Points is a list of lists of coefficients representing a set of “{it distinct}” points in affine space.

### Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This function computes separators for the points: that is, for each point a polynomial is determined whose value is 1 at that point and 0 at all the others. The separators yielded are reduced with respect to the reduced Groebner basis which would be found by “**IdealOfPoints**” ([I-9.7](#) pg.135).

NOTE:

\* the current ring must have at least as many indeterminates as the dimension of the space in which the points lie;

- \* the base field for the space in which the points lie is taken to be the coefficient ring, which should be a field;
- \* in the polynomials returned the first coordinate in the space is taken to correspond to the first indeterminate, the second to the second, and so on;
- \* the separators are in the same order as the points (i.e. the first separator is the one corresponding the first point, and so on);
- \* if the number of points is large, say 100 or more, the returned value can be very large. To avoid possible problems when printing such values as a single item we recommend printing out the elements one at a time as in this example:

```
S := SeparatorsOfPoints(Pts);
foreach sep in S do
  println sep;
endforeach;
```

For separators of points in projective space, see “[SeparatorsOfProjectivePoints](#)” ([I-19.7 pg.284](#)).

example

```
use R ::= QQ[x,y];
Points := [[1, 2], [3, 4], [5, 6]];
S := SeparatorsOfPoints(Points); -- compute the separators
S;
[1/8y^2 - 5/4y + 3, -1/4y^2 + 2y - 3, 1/8y^2 - 3/4y + 1]
-----
[[Eval(F, P) | P in Points] | F in S]; -- verify separators
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
-----
```

**See Also:** [GenericPoints](#)([I-7.6 pg.109](#)), [IdealAndSeparatorsOfPoints](#)([I-9.3 pg.132](#)), [IdealAndSeparatorsOfProjectivePoints](#)([I-9.4 pg.133](#)), [IdealOfPoints](#)([I-9.7 pg.135](#)), [IdealOfProjectivePoints](#)([I-9.8 pg.136](#)), [Interpolate](#)([I-9.30 pg.148](#)), [SeparatorsOfProjectivePoints](#)([I-19.7 pg.284](#))

## I-19.7 SeparatorsOfProjectivePoints

syntax

```
SeparatorsOfProjectivePoints(Points: LIST): LIST
```

where Points is a list of lists of coefficients representing a set of ‘‘{\it distinct}’’ points in projective space.

### Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

This function computes separators for the points: that is, for each point a homogeneous polynomial is determined whose value is non-zero at that point and zero at all the others. (Actually, choosing the values listed in Points as representatives for the homogeneous coordinates of the corresponding points in projective space, the non-zero value will be 1.) The separators yielded are reduced with respect to the reduced Groebner basis which would be found by “[IdealOfProjectivePoints](#)” ([I-9.8 pg.136](#)).

NOTE:

- \* the current ring must have at least one more indeterminate than the dimension of the projective space in which the points lie, i.e, at least as many indeterminates as the length of an element of the input, Points;
- \* the base field for the space in which the points lie is taken to be the coefficient ring, which should be a field;
- \* in the polynomials returned the first coordinate in the space is taken to correspond to the first indeterminate, the second to the second, and so on;

\* the separators are in the same order as the points (i.e. the first separator is the one corresponding the first point, and so on);

\* if the number of points is large, say 100 or more, the returned value can be very large. To avoid possible problems when printing such values as a single item we recommend printing out the elements one at a time as in this example:

```
S := SeparatorsOfProjectivePoints(Pts);
foreach sep in S do
  println sep;
endforeach;
```

For separators of points in affine space, see “[SeparatorsOfPoints](#)” ([I-19.6 pg.283](#)).

example

```
use R ::= QQ[x,y,z];
Points := [[0,0,1],[1/2,1,1],[0,1,0]];
S := SeparatorsOfProjectivePoints(Points);
S;
[-2x + z, 2x, -2x + y]
-----
[[Eval(F, P) | P in Points] | F in S]; -- verify separators
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
-----
```

**See Also:** [GenericPoints\(I-7.6 pg.109\)](#), [IdealAndSeparatorsOfPoints\(I-9.3 pg.132\)](#), [IdealAndSeparatorsOfProjectivePoints\(I-9.4 pg.133\)](#), [IdealOfPoints\(I-9.7 pg.135\)](#), [IdealOfProjectivePoints\(I-9.8 pg.136\)](#), [Interpolate\(I-9.30 pg.148\)](#), [SeparatorsOfPoints\(I-19.6 pg.283\)](#)

## I-19.8 SetCol

syntax

```
SetCol(ref M: MAT, i: INT, L: LIST)
```

### Description

This procedure sets the elements in “L” as entries of the “i”-th column in the matrix “M”; it returns nothing!

example

```
/**/ M := IdentityMat(QQ, 5);
/**/ SetCol(ref M, 1, [2,3,4,0,0]);
/**/ M;
matrix(QQ,
  [[2, 0, 0, 0, 0],
   [3, 1, 0, 0, 0],
   [4, 0, 1, 0, 0],
   [0, 0, 0, 1, 0],
   [0, 0, 0, 0, 1]])
```

**See Also:** [ref\(I-18.26 pg.265\)](#), [GetCol\(I-7.11 pg.112\)](#), [SetRow\(I-19.9 pg.285\)](#), [SwapCols\(I-19.52 pg.305\)](#)

## I-19.9 SetRow

syntax

```
SetRow(ref M: MAT, i: INT, L: LIST)
```

## Description

This procedure sets the elements in “L” as entries of the “i”-th row in the matrix “M”; it returns nothing!

example

```
/**/ M := IdentityMat(QQ, 5);
/**/ SetRow(ref M, 1, [2,3,4,0,0]);
/**/ M;
matrix(QQ,
  [[2, 3, 4, 0, 0],
   [0, 1, 0, 0, 0],
   [0, 0, 1, 0, 0],
   [0, 0, 0, 1, 0],
   [0, 0, 0, 0, 1]])
```

**See Also:** [ref\(I-18.26 pg.265\)](#), [GetRow\(I-7.15 pg.114\)](#), [SetCol\(I-19.8 pg.285\)](#), [SwapRows\(I-19.53 pg.306\)](#)

## I-19.10 SetStackSize

syntax

```
SetStackSize(NewSize: INT)
```

## Description

Secret and dangerous.

## I-19.11 SetVerbosityLevel

syntax

```
SetVerbosityLevel(N: INT)
```

## Description

This function sets the verbosity level: various functions defined in CoCoALib and in the CoCoA packages print out some internal progress messages when the global “VerbosityLevel” ([I-22.2 pg.329](#)) is higher than some value (see their specific manual entries for the relevant values, anyway not less than 10).

This may also be applied in user defined functions: values 1-9 may be used without triggering any verbosity from CoCoA.

example

```
/**/ SetVerbosityLevel(100);
/**/ use QQ[x,y,z]; GB := GBasis(ideal(x^3 -x*y +1, x*y^2 -y -2));
myDoGBasis[1]: New poly in GB: len(GB) = 1 len(pairs) = 1
myDoGBasis[1]: New poly in GB: len(GB) = 2 len(pairs) = 1
myDoGBasis[1]: New poly in GB: len(GB) = 3 len(pairs) = 2
myDoGBasis[1]: New poly in GB: len(GB) = 4 len(pairs) = 2
myDoGBasis[1]: New poly in GB: len(GB) = 5 len(pairs) = 2

/**/ SetVerbosityLevel(0); --> unset verbosity
```

**See Also:** [VerbosityLevel\(I-22.2 pg.329\)](#)

## I-19.12 *shape*

syntax

```
shape(E: LIST): LIST (of TYPE)
shape(E: RECORD): RECORD (of TYPE)
shape(E: OTHER): TYPE
```

where OTHER stands for a type which is not LIST, MAT, or RECORD.

### Description

This function returns the extended list of types involved in the expression E as outlined below:

`type(E) = LIST`

In this case, `Shape(E)` is the list whose *i*-th component is the type of the *i*-th component of E.

`type(E) = MAT`

In this case, `Shape(E)` is a matrix with (*i,j*)-th entry equal to the type of the (*i,j*)-th entry of E.

`type(E) = RECORD`

In this case, `Shape(E)` is a record whose fields are the types of the fields of E.

Otherwise, “`Shape(E)`” is the type of E.

example

```
/**/ use R ::= QQ[x];
/**/ L := [1,[1,"a"], x^2-x];
/**/ shape(L);
[INT, [INT, STRING], POLY]

/**/ R := record[name := "test", contents := L];
/**/ shape(R);
record[contents := [INT, [INT, STRING], POLY], name := STRING]

/**/ It.name;
STRING
```

There are undocumented functions, “`IsSubShape`” and “`IsSubShapeOfSome`”, for determining if the “`shape`” of a CoCoA expression is a “`subshape`” of another. To see the code for these functions, enter

```
Describe Function("$misc.IsSubShape");
Describe Function("$misc.IsSubShapeOfSome");
```

## I-19.13 *sign*

syntax

```
sign(X: INT|RAT): INT
```

### Description

This function returns -1 if  $X < 0$ , 0 if  $X = 0$ , and 1 if  $X > 0$ . X must be INT or RAT.

example

```

/**/  sign(123);
1

/**/  sign(-5/2);
-1

```

## I-19.14 SimplestBinaryRatBetween

syntax

```
SimplestBinaryRatBetween(A: RAT, B: RAT): RAT
```

### Description

This function finds the simplest binary rational in the closed interval with end points “A” and “B”. We define the simplest binary rational to be the rational number of the form “ $N \cdot 2^k$ ” where the integer “N” has the smallest possible absolute value. See also “SimplestRatBetween” (I-19.15 pg.288).

example

```

/**/  SimplestBinaryRatBetween(0.123, 0.456);
1/4

/**/  SimplestBinaryRatBetween(-3.14159, -2.71828);
-3

/**/  SimplestBinaryRatBetween(5,10); // contrast with SimplestRatBetween!
8

```

**See Also:** CFAprox(I-3.8 pg.48), FloatApprox(I-6.12 pg.98), SimplestRatBetween(I-19.15 pg.288)

## I-19.15 SimplestRatBetween

syntax

```
SimplestRatBetween(A: RAT, B: RAT): RAT
```

### Description

This function finds the simplest rational in the closed interval with end points “A” and “B”. See also SimplestBinaryRatBetween.

example

```

/**/  SimplestRatBetween(0.123, 0.456);
1/3

/**/  SimplestRatBetween(-3.14159, -2.71828);
-3

```

**See Also:** CFAprox(I-3.8 pg.48), FloatApprox(I-6.12 pg.98), SimplestBinaryRatBetween(I-19.14 pg.288)

## I-19.16 SimplexInfo

syntax

```
SimplexInfo(A: LIST): RECORD
```

## Description

This function compute the Stanley-Reisner ideal, the Alexander Dual complex and ideal of a simplicial complex described by a list of top faces.

Package “GeomModelling”, by Elisa Palezzato.

example

```
/**/ use QQ[x[1..5]], DegLex;
/**/ L := [x[1]*x[2]*x[3], x[2]*x[3]*x[4], x[3]*x[4]*x[5]]; -- list top faces
/**/ indent(SimplexInfo(L));
record[
  AlexanderDualCOMPLEX := [x[2]*x[3]*x[5], x[2]*x[3]*x[4], x[1]*x[3]*x[4]],
  AlexanderDualIdeal := ideal(x[4]*x[5], x[1]*x[5], x[1]*x[2]),
  Delta := [x[1]*x[2]*x[3], x[2]*x[3]*x[4], x[3]*x[4]*x[5]],
  StanleyReisnerIdeal := ideal(x[1]*x[4], x[1]*x[5], x[2]*x[5])
]
```

**See Also:** FVector(I-6.29 pg.106), SimplicialHomology(I-19.17 pg.289)

## I-19.17 SimplicialHomology

syntax

```
SimplicialHomology(A: LIST): RECORD
SimplicialHomology(A: LIST, B: LIST): RECORD
```

## Description

This function computes the simplicial homology of a simplicial complex described by a list of top faces. With 2nd argument only with the second list of vertices.

Package GeomModelling, by Elisa Palezzato.

example

```
/**/ use QQ[x[1..5]], DegLex; --> DegLex ?

/**/ L := [x[1]*x[2]*x[3], x[2]*x[3]*x[4], x[3]*x[4]*x[5]]; -- list top faces
/**/ indent(SimplicialHomology(L));
record[
  H_0 := record[betti := 1, lambda := []],
  H_i := [record[betti := 0, lambda := []]],
  H_max := record[betti := 0, lambda := []]
]
-- 1 connected component (beti in H_0)

/**/ L := [x[1]*x[2]*x[3], x[2]*x[3]*x[4]]; -- list top faces
/**/ -- indent(SimplicialHomology(L)); --!!! ERROR !!! as expected: missing x[5]

/**/ L := [x[1]*x[2]*x[3], x[2]*x[3]*x[4], x[5]];
/**/ indent(SimplicialHomology(L));
record[
  H_0 := record[betti := 2, lambda := []],
  H_i := [record[betti := 0, lambda := []]],
  H_max := record[betti := 0, lambda := []]
]
-- 2 connected components

/**/ L := [x[1]*x[2]*x[3], x[2]*x[3]*x[4]];
```

```

/**/ indent( SimplicialHomology(L, [x[1], x[2], x[3], x[4]]) );
record[
  H_0 := record[betti := 1, lambda := []],
  H_i := [record[betti := 0, lambda := []]],
  H_max := record[betti := 0, lambda := []]
]

```

## I-19.18 size [OBSOLETE]

syntax

[OBSOLETE]

### Description

[OBSOLETE] see “len” (I-12.6 pg.181).

**See Also:** len(I-12.6 pg.181)

## I-19.19 skip

syntax

skip

### Description

This command does nothing. I suppose it might be used to make the structure of a user-defined function more clear. It is probably at least as useful as the function “Tao”.

example

```

/**/ skip;

```

## I-19.20 SmallestNonDivisor

syntax

SmallestNonDivisor(N: INT): INT

### Description

This function finds the smallest prime which does not divide an integer. It simply tries dividing by all primes in increasing order until it finds one which does not divide “N”.

example

```

/**/ SmallestNonDivisor(factorial(100));
101

/**/ SmallestNonDivisor(-100);
3

```

**See Also:** IsPrime(I-9.71 pg.163), IsProbPrime(I-9.72 pg.164), SmoothFactor(I-19.21 pg.291)

## I-19.21 SmoothFactor

syntax

```
SmoothFactor(N: INT, MaxP: INT): RECORD
```

### Description

This function finds the small prime factors of an integer. It simply tries dividing by all primes up to the given bound “MaxP”. The result is a list of the prime factors found together with the unfactored part of N. Be careful about supplying large values for “MaxP” (e.g. greater than a million) as the function could take a very long time.

From version 5.0.4 the field are called “factors” and “multiplicities” instead of “Factors” and “Exponents” to comply with the naming conventions.

example

```
/**/ SmoothFactor(100,3);
record[factors := [2], multiplicities := [2], RemainingFactor := 25]

/**/ SmoothFactor(123456789,3700);
record[factors := [3, 3607], multiplicities := [2, 1], RemainingFactor := 3803]
```

**See Also:** [IsPrime\(I-9.71 pg.163\)](#), [IsProbPrime\(I-9.72 pg.164\)](#)

## I-19.22 sort

syntax

```
sort(V: LIST)
```

where V is a variable containing a list.

### Description

This function sorts the elements of the list in V with respect to the default comparisons related to their types; it overwrites V and returns NULL.

For more on the default comparisons, see “Relational Operators” ([II-3.3 pg.348](#)) in the chapter on operators. For more complicated sorting, see “SortBy” ([I-19.23 pg.292](#)), “SortedBy” ([I-19.25 pg.293](#)).

example

```
/**/ L := [3,2,1];
/**/ sort(ref L); -- this returns nothing and modifies L
/**/ L;
[1, 2, 3]

/**/ use R ::= QQ[x,y,z];
/**/ L := [x,y,z];
/**/ sort(ref L); -- this returns nothing and modifies L
/**/ L[1];
z

/**/ sorted([y, x, x^2]); -- this returns the sorted list
[y, x, x^2]
```

**See Also:** [Relational Operators\(II-3.3 pg.348\)](#), [sorted\(I-19.24 pg.292\)](#), [SortBy\(I-19.23 pg.292\)](#), [SortedBy\(I-19.25 pg.293\)](#)

## I-19.23 SortBy

syntax

```
SortBy(L: LIST, LessThanFunc: FUNCTION)
```

### Description

This function sorts the elements of the list in L in increasing order with respect to the comparisons made by LessThanFunc; it overwrites L and returns NULL.

The comparison function LessThanFunc takes two arguments and returns True if the first argument is less than the second, otherwise it returns False. The sorted list is in increasing order.

Note that to call SortBy(L, LessThanFunc) inside a function you will need to make the name LessThanFunc accessible using TopLevel LessThanFunc;

Note that if both LessThanFunc(A, B) and LessThanFunc(B, A) return true, then A and B are viewed as being equal.

example

```
/**/ Define LessThanLen(S, T)    -- define the sorting function
/**/   Return len(S) < len(T);
/**/ EndDefine;

/**/ L := ["bird", "mouse", "cat", "elephant"];
/**/ SortBy(ref L, LessThanLen);
/**/ L;
["cat", "bird", "mouse", "elephant"]
```

**See Also:** func(I-6.26 pg.105), sort(I-19.22 pg.291), sorted(I-19.24 pg.292), SortedBy(I-19.25 pg.293), TopLevel(I-20.10 pg.314)

## I-19.24 sorted

syntax

```
sorted(L: LIST): LIST
```

### Description

This function returns the list of the sorted elements of L without affecting L, itself.

For more on the default comparisons, see “Relational Operators” (II-3.3 pg.348) in the chapter on operators. For more complicated sorting, see “SortBy” (I-19.23 pg.292), “SortedBy” (I-19.25 pg.293).

example

```
/**/ L := [3,2,1];
/**/ sorted(L);
[1, 2, 3]

/**/ use R ::= QQ[x,y,z];
/**/ L := [x,y,z];
/**/ sorted(L);
[z, y, x]

/**/ sorted([y, x, z, x^2]);
[z, y, x, x^2]

/**/ sorted([3, 1, 1, 2]);
```

```
[1, 1, 2, 3]

/**/ sorted(["b","c","a"]);
["a", "b", "c"]
```

**See Also:** Relational Operators([II-3.3](#) pg.348), SortBy([I-19.23](#) pg.292), SortedBy([I-19.25](#) pg.293), sort([I-19.22](#) pg.291)

## I-19.25 SortedBy

— syntax —

```
SortedBy(L: LIST, F: FUNCTION): LIST
```

### Description

This function returns the list of the sorted elements of L without affecting L, itself. As for “SortBy” ([I-19.23](#) pg.292), the comparison function F takes two arguments and returns True if the first argument is less than the second, otherwise it returns False. The sorted list is in increasing order.

Note that if both F(A, B) and F(B, A) return True, then A and B are viewed as being equal.

— example —

```
/**/ Define LessByLength(S, T)    -- define the sorting function
/**/   Return len(S) < len(T);
/**/ EndDefine;

/**/ L := ["bird","mouse","cat", "elephant"];
/**/ sorted(L); -- default is alphabetical order
["bird", "cat", "elephant", "mouse"]
/**/ SortedBy(L, LessByLength);
["cat", "bird", "mouse", "elephant"]

/**/ L; -- L is not changed
["bird", "mouse", "cat", "elephant"]

/**/ SortBy(ref L, LessByLength); -- sort L in place, changing L
/**/ L;
["cat", "bird", "mouse", "elephant"]
```

**See Also:** func([I-6.26](#) pg.105), sort([I-19.22](#) pg.291), sorted([I-19.24](#) pg.292), SortBy([I-19.23](#) pg.292)

## I-19.26 source

— syntax —

```
source S: STRING
```

### Description

This command executes all CoCoA commands in the file or device named S. A typical use of “source” is to collect user-defined functions and variables in a text file, say, “MyFile.coc” and then execute:

```
source "MyFile.cocoa5";
```

or, equivalently, the obsolescent

```
<< "MyFile.cocoa5";
```

Functions and variables read in from a file in this way will erase functions and variables with identical names that may already exist. This can be avoided by using packages. Repeatedly used functions can be read into CoCoA at start-up by using “source” in the “userinit.coc” file.

**See Also:** Introduction to IO([II-7.1](#) pg.[355](#)), Introduction to Packages([II-8.1](#) pg.[359](#))

## I-19.27 SourceRegion

syntax

```
SourceRegion FromLine: INT,FromChar: INT To ToLine: INT,ToChar: INT In S: STRING
```

### Description

This command executes all CoCoA commands in the specified region of the given file. It is not really intended for manual use, but is used by the CoCoA UI.

```
SourceRegion FromLine,FromChar To ToLine,ToChar In "MyFile.cocoa5";
```

It is almost equivalent to copying the region into a temporary file, and then reading that file with the “source” command.

Line and char indexes start from 1; the region identified starts at the “from” line/character position and stops immediately before the “to” line/character position.

**See Also:** source([I-19.26](#) pg.[293](#))

## I-19.28 spaces

syntax

```
spaces(N: INT): STRING
```

### Description

This function returns a string consisting of N spaces.

example

```
/**/ L := "a" + Spaces(5) + "b";
/**/ L;
a      b
```

**See Also:** dashes([I-4.1](#) pg.[71](#))

## I-19.29 sprint

syntax

```
sprint(E: OBJECT): STRING
```

### Description

This function takes any CoCoA expression and converts its value to a string. One use is to check for extremely long output before printing in a CoCoA window.

example

```

/**/ use R := QQ[x,y];
/**/ I := ideal(x,y);
/**/ J := sprint(I);
/**/ I;
ideal(x, y)

/**/ J;          -- The output for I and J looks the same, but ...
ideal(x, y)

/**/ type(I); -- I is an ideal, and
IDEAL

/**/ type(J); -- J is just the string "ideal(x, y)".
STRING

/**/ J[1]; -- the 1st character of J
i

/**/ J[2]; -- the 2nd character of J
d

/**/ len(J); -- J has 11 characters
11

```

**See Also:** Introduction to IO([II-7.1](#) pg.355), IO.SprintTrunc([I-9.36](#) pg.150), print([I-16.29](#) pg.245), println([I-16.34](#) pg.247)

## I-19.30 SqFreeFactor

syntax

```
SqFreeFactor(F: RINGELEM): RECORD
```

### Description

Compute a factorization (of a polynomial) into coprime squarefree factors. The factorization may sometimes be finer than strictly necessary, i.e. two factors could have the same multiplicity.

example

```

/**/ use P := QQ[x,y,z];
/**/ f := (x^2-1)^2*(y*z+2*z)^3;
/**/ indent(SqFreeFactor(f));
record[
  RemainingFactor := 1,
  factors := [x^2 -1, y +2, z],
  multiplicities := [2, 3, 3]
]

```

**See Also:** radical([I-18.1](#) pg.255), factor([I-6.1](#) pg.93), ContentFreeFactor([I-3.47](#) pg.64)

## I-19.31 StableBBasis5

syntax

```

StableBBasis5(Pts: LIST, Toler: LIST): RECORD
StableBBasis5(Pts: LIST, Toler: LIST, Gamma: RAT): RECORD

```

## Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\* See “ApproxPointsNBM” (I-1.14 pg.32)

This function returns a record containing a “*stable order ideal*” of the ideal of points, and a list of “*almost vanishing*” polynomials. If the cardinality of the order ideal is equal to the number of points, it is in fact a “*quotient basis*”, and in this case a “*stable border basis*” founded on it is also returned. The boolean field “StableBBasisFound” is set to “true” if a stable border basis was found, otherwise “false”.

The first argument is a list of points in k-dimensional space, and the second argument is list of k positive tolerances (one for each dimension). The function builds the stable order ideal stepwise by testing, from a numerical point of view, the linear dependence of a set of vectors. So that the answer can be represented, the current ring must have at least k indeterminates; the term ordering is ignored as it plays no role in determining the border basis.

There is a third, optional argument: it is a real non negative number “Gamma” which is used for scaling the threshold on the admissible perturbation of the points. A value of “Gamma”  $\geq 1$  should be used. If no value is specified then by default “Gamma” = 0.1

For a full description of the algorithms we refer to the paper J.Abbott, C.Fassino, L.Torrente “*Stable Border Bases for Ideals of Points*” (to appear in JSC or arXiv:07062316).

### example

```
Pts := [[0.1,-1],[1,1],[2,3]];
Toler := [0.1,0.1];
StableBBasis5(Pts, Toler);
record[
  AlmostVanishing := [ (...) ],
  BBasis := [
    -3602879701896397/288230376151711744y^2 + x -
    32425917317067571/72057594037927936y -
    154923827181545063/288230376151711744,
    xy - 140512308373959475/288230376151711744y^2 -
    39631676720860365/72057594037927936y +
    10808639105689191/288230376151711744,
    y^3 - 3y^2 - y + 3,
    xy^2 - 580063632005319885/288230376151711744y^2 -
    32425917317067571/72057594037927936y +
    421536925121878425/288230376151711744],
  SOI := [1, y, y^2],
  StableBBasisFound := True]
-----
Toler := [0.6, 0.6]:
StableBBasis5(Pts, Toler);
record[AlmostVanishing := [.....], SOI := [1, y], StableBBasisFound := False]
-----
```

**See Also:** IdealOfPoints(I-9.7 pg.135), ApproxPointsNBM(I-1.14 pg.32)

## I-19.32 StableIdeal

### syntax

```
StableIdeal(L: LIST of power-products): IDEAL
```

## Description

This function returns the smallest stable ideal containing the power-products in “L” (see also “StronglyStableIdeal” (I-19.39 pg.300)).

example

```

/**/ use R ::= QQ[x,y,z];
/**/ L := [x*z^4, y^3];
/**/ StableIdeal(L);
ideal(x^2*z^3, x*y*z^3, x*z^4, x^3, x^2*y, x*y^2, y^3)

```

**See Also:** [IsStable\(I-9.80 pg.166\)](#), [LexSegmentIdeal\(I-12.8 pg.182\)](#), [StronglyStableIdeal\(I-19.39 pg.300\)](#)

## I-19.33 StagedTrees

syntax

```
StagedTrees(L: LIST of power-products): IDEAL
```

### Description

This function returns all possible staged trees associated to an interpolating polynomial.

See BigGoeRicSmi paper (TODO details)

example

```

/**/ use QQ[a,b,x,y,z,w];
/**/ c_T := a*(x*(z+w)+y) + b*(x+y);
/**/ trees := StagedTrees(c_T);
/**/ PrintTrees(trees);
-- number of trees = 2 -----
----- tree 1 -----
-- florets: [[x, y], [a, b], [z, w]]
<
x <
  a <
    z
    w
  b
y <
  a
  b
----- tree 2 -----
-- florets: [[a, b], [x, y], [z, w]]
<
a <
  x <
    z
    w
  y
b <
  x
  y
----- end trees -----

```

**See Also:** [IsStable\(I-9.80 pg.166\)](#), [LexSegmentIdeal\(I-12.8 pg.182\)](#), [StronglyStableIdeal\(I-19.39 pg.300\)](#)

## I-19.34 StarPrint, StarSprint

syntax

```

StarPrint(F: RINGELEM)
StarPrintFold(F: RINGELEM, LineWidth: INT)

```

```
StarPrint(F: RINGELEM): STRING
StarPrintFold(F: RINGELEM, LineWidth: INT): STRING
```

## Description

\*\*\* NOT YET IMPLEMENTED \*\*\*

These functions print the polynomial  $F$  with asterisks added to denote multiplications. They may be useful when transferring polynomials or rational functions from CoCoA to other mathematical software (e.g. Gap, Maple, Macaulay, Singular,..). “StarPrint” inserts newline characters (only between terms) with the aim of avoiding lines longer than 70 characters; the second argument to “StarPrintFold” is for specifying a different width limit; a non positive value is treated as meaning no limit. The “StarSprint” functions print the value into a string.

example

```
use R := QQ[x,y];
F := x^3+2xy-y^2;
StarPrint(F);
1*x^3 +2*x*y -1*y^2
-----
StarPrintFold(F,1); -- this will print one term per line
1*x^3
+2*x*y
-1*y^2
-----
D := OpenOFile("example");
Print StarPrint(F) On D; -- this will print F into the file "example"
Close(D);
-----
```

**See Also:** LaTeX(?? pg.??)

## I-19.35 starting

syntax

```
starting(S: STRING): LIST of RECORD
```

## Description

This function returns a list of all CoCoA functions starting with the string “S”. In general, this list will include undocumented commands. For these, one may find some information using “Describe Function(“Fn\_Name”)” or “Describe Function(“\$PackageName.Fn\_Name”)”.

example

```
/**/ indent(starting("Su"));
[
  record[IsExported := true, name := "$BackwardCompatible.Subsets"],
  record[IsExported := true, name := "$BackwardCompatible.Subst"],
  record[IsExported := true, name := "$BackwardCompatible.Sum"],
  record[IsExported := true, name := "$BackwardCompatible.Support"]
]
```

## I-19.36 StdBasis

syntax

```
StdBasis(I: IDEAL): LIST
```

## Description

A “*standard basis*” of the ideal “*I*” is a set of polynomials whose initial forms generate the “*tangent cone*” of “*I*” (see “*TgCone*” (I-20.5 pg.312) for more details).

The implementation is based on Lazard’s method (see Kreuzer-Robbiano, Computational Commutative Algebra 2, pg.463).

example

```
/**/ use R ::= QQ[x,y,z];
/**/ I := ideal(x^2*z-2*y, x^3+y^2-y*z);
StdBasis(I);
[(-1/2)*x^2*z +y, (1/2)*x^2*y*z +(-1/2)*x^2*z^2 +x^3]
/**/ TgCone(I);
ideal(y, x^3)
```

**See Also:** TgCone(I-20.5 pg.312), PrimaryHilbertSeries(I-16.26 pg.243)

## I-19.37 StdDegLexMat

syntax

```
StdDegLexMat(N: INT): MAT
```

## Description

This function return the matrix defining a standard term-ordering.

example

```
/**/ StdDegLexMat(3);
matrix(ZZ,
  [[1, 1, 1],
   [1, 0, 0],
   [0, 1, 0]])
```

**See Also:** OrdMat(I-15.10 pg.231), Term Orderings(III-9.5 pg.404), StdDegRevLexMat(I-19.38 pg.299), LexMat(I-12.7 pg.182), RevLexMat(I-18.41 pg.272), XelMat(I-24.1 pg.335)

## I-19.38 StdDegRevLexMat

syntax

```
StdDegRevLexMat(N: INT): MAT
```

## Description

This function return the matrix defining a standard term-ordering.

example

```
/**/ StdDegRevLexMat(3);
matrix(ZZ,
  [[1, 1, 1],
   [0, 0, -1],
   [0, -1, 0]])
```

**See Also:** OrdMat(I-15.10 pg.231), Term Orderings(III-9.5 pg.404), StdDegLexMat(I-19.37 pg.299), LexMat(I-12.7 pg.182), RevLexMat(I-18.41 pg.272), XelMat(I-24.1 pg.335)

## I-19.39 StronglyStableIdeal

syntax

```
StronglyStableIdeal(L: LIST of power-products): IDEAL
```

### Description

This function returns the smallest strongly stable ideal containing the power-products in L.

example

```
/**/ use R ::= QQ[x,y,z];
/**/ L := [x*y^2*z];
/**/ StableIdeal(L);
ideal(x^4, x^3*y, x^2*y^2, x*y^3, x*y^2*z)

/**/ StronglyStableIdeal(L);
ideal(x^4, x^3*y, x^2*y^2, x*y^3, x^3*z, x^2*y*z, x*y^2*z)
```

**See Also:** [IsStronglyStable\(I-9.82 pg.167\)](#), [LexSegmentIdeal\(I-12.8 pg.182\)](#), [StableIdeal\(I-19.32 pg.296\)](#)

## I-19.40 SubalgebraHom

syntax

```
SubalgebraHom(R: RING, L: LIST): RINGHOM
```

### Description

This function returns the homomorphism from “R” into (a representation of) the subalgebra generated by “L”.

example

```
/**/ use QQ[s,t];
/**/ L := [s^3, s^2*t, s*t^2, t^3];

/**/ ReprRing ::= QQ[x,y,z,w];
/**/ phi := SubalgebraHom(ReprRing, L);
/**/ phi;
RingHom(RingWithID(256, "QQ[x,y,z,w]") --> RingWithID(257, "QQ[s,t]")
  sending (x |--> s^3) & (y |--> s^2*t) & (z |--> s*t^2) & (w |--> t^3))

/**/ ker(phi); --> implicitization
ideal(z^2 -y*w, y*z -x*w, y^2 -x*z)

/**/ SubalgRepr := preimage0(phi, s^6*t^6); SubalgRepr;
x^2*w^2
/**/ phi(SubalgRepr);
s^6*t^6
```

**See Also:** [preimage0\(I-16.18 pg.239\)](#), [ker\(I-11.1 pg.177\)](#)

## I-19.41 SubalgebraMap [OBSOLETE]

syntax

```
[OBSOLETE]
```

## Description

[OBSOLETE] See “SubalgebraHom” (I-19.40 pg.300).

**See Also:** SubalgebraHom(I-19.40 pg.300)

## I-19.42 SubalgebraRepr [OBSOLESCENT]

syntax

[OBSOLESCENT]

## Description

[OBSOLESCENT]

**See Also:** SubalgebraHom(I-19.40 pg.300), preimage0(I-16.18 pg.239), ker(I-11.1 pg.177)

## I-19.43 submat

syntax

submat(M: MAT, R: LIST of INT, C: LIST of INT): MAT

## Description

This function returns the submatrix of “M” formed by the rows listed in “R” and the columns listed in “C”. If “M” is a list, it is interpreted as a matrix in the natural way.

example

```
/**/ M := mat([[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15]]);
/**/ submat(M,[1,3],[3..5]);
matrix(QQ,
  [[3, 4, 5],
   [13, 14, 15]])

/**/ M := mat([[1,2,3],[4,5,6]]);
/**/ submat(M,[2],[1,3]);
matrix(QQ,
  [[4, 6]])
```

**See Also:** minors(I-13.21 pg.200)

## I-19.44 submodule

syntax

submodule(L: LIST of MODULEELEM): MODULE  
submodule(F: MODULE, L: LIST of MODULEELEM): MODULE

## Description

The first form returns the ideal generated by “L”. The second is the same as the first but works also if “L = []”.

This function is not friendly if you write the input by hand: we suggest “SubmoduleCols, SubmoduleRows” (I-19.45 pg.302) for creating a module from the rows or columns of a matrix.

NOTE: the second argument is a LIST of MODULEELEM, not a LIST of LISTS of RINGELEM.

## example

```

/**/ use R ::= QQ[x,y,z];
/**/ R3 := NewFreeModule(R, 3);
/**/ L := [ModuleElem(R3, [x,y,z]), ModuleElem(R3, [x-1,0,z])];
/**/ M := submodule(R3, L); -- equivalent to
/**/ M := submodule(L);      -- (L not empty)
/**/ gens(M);
[[x, y, z], [x -1, 0, z]]

```

**See Also:** ModuleOf(I-13.29 pg.203), SubmoduleCols, SubmoduleRows(I-19.45 pg.302), GensAsCols, GensAsRows(I-7.9 pg.111), gens(I-7.8 pg.110)

## I-19.45 SubmoduleCols, SubmoduleRows

## syntax

```

SubmoduleCols(F: MODULE, M: MATRIX): MODULE
SubmoduleRows(F: MODULE, M: MATRIX): MODULE

```

### Description

The first (second) function returns the submodule of F generated by the module elements described by the columns (rows) in the matrix M (which might be empty).

Dimensions must be compatible.

## example

```

/**/ use R ::= QQ[x,y,z];
/**/ R3 := NewFreeModule(R, 3);
/**/ MGens := matrix(R, [[x,y,z], [x-1,0,z]]);

/**/ M := SubmoduleRows(R3, MGens);
/**/ gens(M);
[[x, y, z], [x -1, 0, z]]

-- /**/ M := SubmoduleCols(R3, MGens); --!!! ERROR !!! as expected: wrong length

/**/ M := SubmoduleCols(NewFreeModule(R,2), MGens);
/**/ gens(M);
[[x, x -1], [y, 0], [z, z]]

```

**See Also:** GensAsCols, GensAsRows(I-7.9 pg.111), submodule(I-19.44 pg.301), ModuleElem(I-13.28 pg.203)

## I-19.46 SubmoduleOfMinGens

## syntax

```

SubmoduleOfMinGens(M: MODULE): MODULE

```

### Description

It works only in the homogeneous case: for the inhomogeneous case see “MinSubsetOfGens” (I-13.25 pg.202).

This function returns the ideal generated by a minimal set of generators (i.e. with minimal cardinality) of “M”. The minimal set of generators is not necessarily a subset of the given generators.

Similar to “IdealOfMinGens” (I-9.6 pg.135).

## example

```

/**/ use R ::= QQ[x,y,z];
/**/ R3 := NewFreeModule(R, 3);
/**/ MGens := matrix(R, [[x,y,z], [x^2,0,z^2], [2*x^2,x*y,z^2+x*z]]);
/**/ M := SubmoduleRows(R3, MGens);
/**/ MGM := SubmoduleOfMinGens(M); indent(MGM);
SubmoduleRows(F, matrix([
  [x, y, z],
  [0, x*y, x*z -z^2]
]))

```

**See Also:** [MinGens\(I-13.16 pg.198\)](#), [MinSubsetOfGens\(I-13.25 pg.202\)](#)

## I-19.47 subsets

## syntax

```

subsets(S: LIST): LIST
subsets(S: LIST, N: INT): LIST

```

### Description

This function computes all sublists (subsets) of a list (set). If N is specified, it computes all sublists of cardinality N.

## example

```

/**/ subsets([1, 4, 7]);
[[ ], [7], [4], [4, 7], [1], [1, 7], [1, 4], [1, 4, 7]]

/**/ subsets([1, 4, 7], 2);
[[1, 4], [1, 7], [4, 7]]

/**/ subsets([2,3,3]); -- list with repeated entries
[[ ], [3], [3], [3, 3], [2], [2, 3], [2, 3], [2, 3, 3]]

/**/ subsets(MakeSet([2,3,3]));
[[ ], [3], [2], [2, 3]]

```

**See Also:** [IsSubset\(I-9.83 pg.167\)](#), [partitions\(I-16.4 pg.234\)](#), [permutations\(I-16.5 pg.234\)](#), [MakeSet\(I-13.3 pg.192\)](#), [tuples\(I-20.15 pg.317\)](#)

## I-19.48 subst

## syntax

```

subst(E: OBJECT, X, F): OBJECT
subst(E: OBJECT, [[X_1, F_1], ..., [X_r, F_r]]): OBJECT
  where each X or X_i is an indeterminate
  and each F or F_i is a RINGELEM

```

### Description

The first form of this function substitutes “F\_i” for “X\_i” in the expression E. The second form is a shorthand for the first in the case of a single indeterminate. When substituting for the indeterminates in order, it is easier to use “eval” ([I-5.12 pg.88](#)).

## example

```

/**/ use R ::= QQ[x,y,z,t];
/**/ F := x +y +z +t^2;
/**/ subst(F, x, -2);
t^2 +y +z -2

/**/ subst(F, [[x,x^2], [y,y^3], [z,t^5]]);
t^5 +y^3 +x^2 +t^2

/**/ eval(F, [x^2,y^3,t^5]); -- the same thing as above
t^5 +y^3 +x^2 +t^2

/**/ MySubst := [[y,1], [t, 3*z-x]];
/**/ subst(x*y*z*t, MySubst); -- substitute into the function x*y*z*t
-x^2*z +3*x*z^2

```

**See Also:** [eval\(I-5.12 pg.88\)](#), [Evaluation of Polynomials\(III-11.2 pg.412\)](#), [PolyAlgebraHom\(I-16.14 pg.237\)](#), [QZP\(I-17.7 pg.253\)](#), [RingElem\(I-18.43 pg.273\)](#), [ZPQ\(I-25.3 pg.338\)](#)

## I-19.49 sum

## syntax

```

sum(L: LIST): OBJECT
sum(L: LIST, InitVal: OBJECT): OBJECT

```

### Description

This function returns the sum of the objects in the list “L” (together with “InitVal”, if specified). When writing a program, if the list “L” may be empty, you must specify “InitVal”.

## example

```

/**/ use R ::= QQ[x,y];
/**/ sum([3, x, y^2]);
y^2 +x +3

/**/ sum(1..40) = binomial(41,2);
true

/**/ sum(["c","oc","oa"]);
cocoa

/**/ sum([]); -- gives 0 of type INT
0
/**/ sum([], ""); -- gives empty STRING

/**/ sum([], x); -- gives type RINGELEM
x

```

**See Also:** [Algebraic Operators\(II-3.2 pg.347\)](#), [product\(I-16.37 pg.249\)](#)

## I-19.50 support

## syntax

```

support(F: RINGELEM): LIST
support(F: MODULEELEM): LIST

```

## Description

This function returns the list of terms of  $F$ . To get a list of monomials, which includes coefficients, use “monomials” ([I-13.31 pg.204](#)).

example

```
/**/ use R ::= QQ[x,y];
/**/ F := 3*x^2-4*x*y+y^3+3;
/**/ support(F);
[y^3, x^2, x*y, 1]

/**/ monomials(F);
[y^3, 3*x^2, -4*x*y, 3]

// NOT YET IMPLEMENTED for MODULEELEM
```

**See Also:** [coefficients\(I-3.24 pg.54\)](#), [monomials\(I-13.31 pg.204\)](#)

## I-19.51 swap

syntax

```
swap(ref A: OBJECT, ref B: OBJECT)
```

## Description

This procedure swaps two values; it returns nothing!

example

```
/**/ A := 1;
/**/ B := 2;
/**/ swap(ref A, ref B);
/**/ PrintLn [A,B];
[2, 1]
```

**See Also:** [ref\(I-18.26 pg.265\)](#)

## I-19.52 SwapCols

syntax

```
SwapCols(ref M: MAT, i: INT, j: INT)
```

## Description

This procedure swaps the “i”-th and “j”-th columns in the matrix “M”; it returns nothing!

example

```
/**/ M := StdDegLexMat(5);
/**/ SwapCols(ref M, 1,5);
/**/ M;
matrix(QQ,
  [[1, 1, 1, 1, 1],
   [0, 0, 0, 0, 1],
   [0, 1, 0, 0, 0],
```

```
[0, 0, 1, 0, 0],
[0, 0, 0, 1, 0]])
```

**See Also:** [ref\(I-18.26 pg.265\)](#), [swap\(I-19.51 pg.305\)](#), [GetCol\(I-7.11 pg.112\)](#), [SetCol\(I-19.8 pg.285\)](#)

## I-19.53 SwapRows

syntax

```
SwapRows(ref M: MAT, i: INT, j: INT)
```

### Description

This procedure swaps the “i”-th and “j”-th rows in the matrix “M”; it returns nothing!

example

```
/**/ M := IdentityMat(QQ, 5);
/**/ SwapRows(ref M, 2,5);
/**/ M;
matrix(QQ,
[[1, 0, 0, 0, 0],
 [0, 0, 0, 0, 1],
 [0, 0, 1, 0, 0],
 [0, 0, 0, 1, 0],
 [0, 1, 0, 0, 0]])
```

**See Also:** [ref\(I-18.26 pg.265\)](#), [swap\(I-19.51 pg.305\)](#), [GetRow\(I-7.15 pg.114\)](#), [SetRow\(I-19.9 pg.285\)](#)

## I-19.54 sylvester

syntax

```
sylvester(F: RINGELEM, G: RINGELEM, X: RINGELEM): MAT
```

### Description

(sorry Sylvester for the lower-case: here we follow the naming convention “*single name goes lower-case*”)

This function returns the Sylvester matrix of the polynomials F and G with respect to the indeterminate X. This is the matrix used to calculate the resultant.

example

```
/**/ use R ::= QQ[p,q,x];
/**/ F := x^3+p*x-q; G := deriv(F, x);
/**/ sylvester(F, G, x);
matrix( /*RingWithID(36, "QQ[p,q,x]")*/
[[1, 0, p, -q, 0],
 [0, 1, 0, p, -q],
 [3, 0, p, 0, 0],
 [0, 3, 0, p, 0],
 [0, 0, 3, 0, p]])

/**/ det(sylvester(F, G, x)) = resultant(F, G, x);
true
```

**See Also:** [resultant\(I-18.38 pg.271\)](#)

## I-19.55 *SymbolRange*

— syntax —

```
SymbolRange(H: STRING, LO: INT, HI: INT): LIST of RINGELEM
SymbolRange(H: STRING, LO: LIST of INT, HI: LIST of INT): LIST of RINGELEM
```

### Description

This function returns the list of the symbols with a given head and a range of indices. A symbol is a record with head (as “IndetName” (I-9.23 pg.144)) and indices (as “IndetSubscripts” (I-9.25 pg.145))

— example —

```
/**/ indent(SymbolRange("x", 3, 5));
[
  record[head := "x", indices := [3]],
  record[head := "x", indices := [4]],
  record[head := "x", indices := [5]]
]
/**/ P := NewPolyRing(QQ, SymbolRange("x", 0,2));
/**/ indets(P);
[x[0], x[1], x[2]]

/**/ indent(SymbolRange("x", [3,1], [5,2]));
[
  record[head := "x", indices := [[3, 1]]],
  record[head := "x", indices := [[3, 2]]],
  record[head := "x", indices := [[4, 1]]],
  record[head := "x", indices := [[4, 2]]],
  record[head := "x", indices := [[5, 1]]],
  record[head := "x", indices := [[5, 2]]]
]
```

**See Also:** [indet\(I-9.21 pg.143\)](#), [IndetSubscripts\(I-9.25 pg.145\)](#), [IndetIndex\(I-9.22 pg.143\)](#), [IndetName\(I-9.23 pg.144\)](#), [NumIndets\(I-14.37 pg.224\)](#), [SymbolRange\(I-19.55 pg.307\)](#)

## I-19.56 *SymmetricPolys*

— syntax —

```
SymmetricPolys(P: RING): LIST of RINGELEM
```

### Description

This function returns the list of the homogeneous symmetric polynomials with square-free support.

— example —

```
/**/ use P ::= QQ[x,y,z];
/**/ SymmetricPolys(P);
[x +y +z, x*y +x*z +y*z, x*y*z]
```

## I-19.57 *syz*

— syntax —

```
Syz(L: LIST of RINGELEM): MODULE
Syz(M: IDEAL|MODULE, Index: INT): MODULE
```

## Description

In the first two forms this function computes the syzygy module of a list of polynomials or module elements. “SyzOfGens(I)” is the same as “Syz(gens(I))”.

In the last form this function returns the specified syzygy module of the minimal free resolution of M which must be homogeneous. As a side effect, it computes the Groebner basis of M. (\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*)

The coefficient ring must be a field.

example

```

/**/ use R := QQ[x,y,z];
/**/ indent(Syz([x^2-y-1, y^3-z, x^2-y, y^3-z]));
SubmoduleRows(F, matrix(
  [y^3 -z, 0, 0, -x^2 +y +1],
  [0, 1, 0, -1],
  [x^2 -y, 0, -x^2 +y +1, 0],
  [0, 0, y^3 -z, -x^2 +y]
))
-----
/**/ I := ideal(x, x, y);
/**/ syz(gens(I));
submodule(FreeModule(..), [[1, -1, 0], [0, y, -x]])
/**/ SyzOfGens(I);
submodule(FreeModule(..), [[1, -1, 0], [0, y, -x]])

Syz(I, 1);      -- NOT YET IMPLEMENTED
Module([[x, -y]])
-----
I := ideal(x^2-yz, xy-z^2, xyz);    -- NOT YET IMPLEMENTED
Syz(I,0);
Module([x^2 - yz], [xy - z^2], [xyz])
-----
Syz(I,1);      -- NOT YET IMPLEMENTED
Module([-x^2 + yz, xy - z^2, 0], [xz^2, -yz^2, -y^2 + xz], [z^3, 0,
-xy + z^2], [0, z^3, -x^2 + yz])
-----
Syz(I,2);
Module([0, z, -x, y], [-z^2, -x, y, -z])
-----
Syz(I,3);
Module([[0]])
-----
Res(I);
0 --> R(-6)^2 --> R(-4)(+)R(-5)^3 --> R(-2)^2(+)R(-3)
-----

```

**See Also:** [res\(I-18.34 pg.269\)](#), [SyzOfGens\(I-19.58 pg.308\)](#)

## I-19.58 SyzOfGens

syntax

```
SyzOfGens(M: IDEAL|MODULE): MODULE
```

## Description

If M is an ideal or submodule, this function calculates the syzygy module for the given set of generators of M.

If  $M$  is a quotient of a ring by an ideal  $I$  or a quotient of a free module by a submodule  $N$ , then this function calculates the syzygy module for the given set of generators of  $I$  or  $N$ , respectively.

“`SyzOfGens(I)`” is the same as “`Syz(gens(I))`”.

The coefficient ring must be a field.

example

```

/**/ use R := QQ[x,y,z];
/**/ I := ideal(x, y, x+y);
/**/ indent(SyzOfGens(I));
SubmoduleRows(F, matrix([
  [1, 1, -1],
  [0, x +y, -y]
]))

/**/ R3 := NewFreeModule(R, 3);
/**/ MGens := matrix(R, [[x,y,z], [x-y,0,z], [y^2,y^2,0]]);
/**/ indent(SyzOfGens(SubmoduleRows(R3, MGens)));
SubmoduleRows(F, matrix([
  [1, -1, -1]
]))

```

**See Also:** `syz`([I-19.57](#) pg.307)



# Chapter I-20

## T

### I-20.1 tag

syntax

```
tag(E: OBJECT): STRING
```

#### Description

If E is a tagged object, this function returns the tag of E; otherwise, it returns the empty string.

example

```
/**/ L := tagged(3,"MyTag");
/**/ type(L);
TAGGED("$TopLevel.MyTag")

/**/ tag(L);
MyTag
```

**See Also:** [Printing a Tagged Object\(III-16.2 pg.429\)](#), [tagged\(I-20.2 pg.311\)](#), [untagged\(I-21.5 pg.326\)](#)

### I-20.2 tagged

syntax

```
tagged(E: OBJECT, S: STRING): TAGGED(S)
```

#### Description

This first function returns the object E, tagged with the string S. Tagging is used for pretty printing of objects. See the reference listed below.

example

```
/**/ L := [1,2,3];
/**/ M := tagged(L,"MyTag");
/**/ type(L);
LIST

/**/ type(M);
TAGGED("$TopLevel.MyTag")
```

```
/**/ type(untagged(M));
LIST
```

**See Also:** Printing a Tagged Object([III-16.2](#) pg.429), tag([I-20.1](#) pg.311), untagged([I-21.5](#) pg.326)

## I-20.3 tail

syntax

```
tail(L: LIST): LIST
```

### Description

This function returns the list obtained from *L* by removing its first element. It cannot be applied to the empty list.

example

```
/**/ tail([1,2,3]);
[2, 3]
```

**See Also:** first([I-6.8](#) pg.96), last([I-12.2](#) pg.179)

## I-20.4 TensorMat

syntax

```
TensorMat(M: MATRIX, N: MATRIX): MAT
```

### Description

This function returns the tensor product of two matrices.

example

```
/**/ use R := QQ[x,y,z,w];
/**/ TensorMat(mat(R, [[1,-1],[2,-2],[3,-3]]), mat(R, [[x,y],[z,w]]));
matrix( /*RingWithID(42, "QQ[x,y,z,w]")*/
  [x, y, -x, -y],
  [z, w, -z, -w],
  [2*x, 2*y, -2*x, -2*y],
  [2*z, 2*w, -2*z, -2*w],
  [3*x, 3*y, -3*x, -3*y],
  [3*z, 3*w, -3*z, -3*w])
```

## I-20.5 TgCone

syntax

```
TgCone(I: IDEAL): IDEAL
```

### Description

The “*initial form*” of a polynomial “*f*” is the homogeneous component of “*f*” of the lowest degree (in contrast with the “*leading form*”, see “*LF*” ([I-12.9](#) pg.183), “*DF*” ([I-4.14](#) pg.78)).

The “*initial ideal*” of the ideal “*I*” is the ideal generated by the initial forms of all polynomials in “*I*”. It is also called “*tangent cone*” (which strictly is the variety defined by the initial ideal).

The implementation is based on Lazard’s method (see Kreuzer-Robbiano, Commutative Computer Algebra 2, pg.463).

example

```
/**/ use R ::= QQ[x,y,z];
/**/ TgCone(ideal(x^3-y));
ideal(y)
/**/ TgCone(ideal(x^3+x^2-y^2));
ideal(x^2 -y^2)

/**/ I := ideal(x^3-y*z, y^2-x*z, z^2-x^2*y);
/**/ TgCone(I); -- same as InitialIdeal(I, [x,y,z]);
ideal(z^2, y*z, y^2 -x*z)
```

**See Also:** InitialIdeal(I-9.28 pg.147), PrimaryHilbertSeries(I-16.26 pg.243)

## I-20.6 TimeFrom

syntax

```
TimeFrom(StartPoint: RAT): STRING
```

### Description

This function returns a string indicating the number of CPU seconds consumed since “StartPoint”; the value in “StartPoint” should be the value produced by the function “CpuTime” (I-3.53 pg.67) at the point where timing should commence.

example

```
/**/ t0 := CpuTime();
/**/ N := factorial(10000000);
/**/ PrintLn "Time to compute N: ",TimeFrom(t0);
Time to compute N: 7.538
```

## I-20.7 TimeOfDay

syntax

```
TimeOfDay(): INT
```

### Description

This function returns the current time as an INT in the form  $HHMMSS = HH * 10000 + MM * 100 + SS$ . Note that from version 5.0.4 this information is no longer given by the function “date” (I-4.2 pg.71).

example

```
/**/ TimeOfDay(); -- 09:08:13
90813
/**/ date(); -- 2013-05-30
20130530
```

**See Also:** date(I-4.2 pg.71)

## I-20.8 TmpChainCanonicalHom

syntax

```
TmpChainCanonicalHom(R: RING, S: RING): RINGHOM
```

## Description

Temporary - might change name/meaning...

example

```
/**/ use R ::= QQ[x,y];
/**/ RmodI := NewQuotientRing(R, ideal(x^2-1));

/**/ phi := TmpChainCanonicalHom(R, RmodI);
/**/ phi(x^3*y);
(x*y)
```

**See Also:** CanonicalHom(I-3.3 pg.46), RingElem(I-18.43 pg.273)

## I-20.9 TmpNBM [OBSOLESCENT]

syntax

[OBSOLESCENT]

## Description

Renamed “ApproxPointsNBM” (I-1.14 pg.32)

## I-20.10 TopLevel

syntax

```
TopLevel X;
  where ‘\verb&X&’ is the name of a top level variable or function.
```

## Description

This command makes a top-level variable accessible from inside a function. It is useful for making “QQ” (I-17.1 pg.251) and “ZZ” (I-25.4 pg.338) visible, and also if a top-level function is to be passed as a parameter (e.g. to the function “SortBy” (I-19.23 pg.292)).

The command may be used with any top-level variable, but it is poor style to use it for purposes other than those mentioned above.

NOTE: Package variables should be accessed directly (via their fully qualified names); the “TopLevel” command does not recognise them.

example

```
/**/ define BeautifulRing(N)
/**/   TopLevel QQ;
/**/   R ::= QQ[b[1..N]];
/**/   return R;
/**/ enddefine;

/**/ Define CompareLen(X,Y) Return len(X) < len(Y); EndDefine;

/**/ Define LongestName(ListOfNameAndValue)
/**/   TopLevel CompareLen; --> to pass it as paremeter to SortBy
/**/   names := [entry[1] | entry in ListOfNameAndValue];
```

```

/**/  SortBy(ref names, CompareLen);
/**/  Return last(names);
/**/  EndDefine;

/**/  L := [{"ABC",1}, {"XYZT",2}];
/**/  LongestName(L);
XYZT

```

**See Also:** [func\(I-6.26 pg.105\)](#), [ImportByRef](#), [ImportByValue\(I-9.17 pg.141\)](#)

## I-20.11 TopLevelFunctions

syntax

TopLevelFunctions(): LIST of FUNCTION

### Description

This function returns the list of all functions available at top-level

example

```

/**/  indent(TopLevelFunctions());
[
  record[IsExported := true, Name := "$BackwardCompatible.Abs"],
  record[IsExported := true, Name := "$BackwardCompatible.Append"],
  record[IsExported := true, Name := "$BackwardCompatible.Ascii"],
  ...

```

## I-20.12 toric

syntax

```

toric(I: IDEAL): IDEAL
toric(I: IDEAL, L: LIST of INDETS): IDEAL
toric(M: MAT|LIST of LIST): IDEAL

```

### Description

These functions return the saturation of an ideal,  $I$ , generated by binomials. In the first two cases,  $I$  is the ideal generated by the binomials in  $L$ . To describe the ideal in the last case, let  $K$  be the integral elements in the kernel of  $M$ . For each  $k$  in  $K$ , we can write  $k = k(+) - k(-)$  where the  $i$ -th component of  $k(+)$  is the  $i$ -th component of  $k$ , if positive, otherwise zero. Then  $I$  is the ideal generated by the binomials " $x^{k(+)} - x^{k(-)}$ " as  $k$  ranges over  $K$ .

NOTE: successive calls to this last form of the function may produce different generators for the saturation.

The first and third functions return the saturation of  $I$ . For the second function, if the saturation of  $I$  with respect to the variables in  $X$  happens to equal the saturation of  $I$ , then the saturation of  $I$  is returned. Otherwise, an ideal "containing" the saturation with respect to the given variables is returned. The point is that if one knows, a priori, that the saturation of  $I$  can be obtained by saturating with respect to a subset of the variables, the second function may be used to save time.

For more details, see the article: A.M. Bigatti, R. La Scala, L. Robbiano, "Computing Toric Ideals," Journal of Symbolic Computation, 27, 351-365 (1999). The article describes three different algorithms; the one implemented in CoCoA is "EATP". The first two examples below are motivated by B. Sturmfels, "Groebner Bases and Convex Polytopes," Chapter 6, p. 51. They count the number of homogeneous primitive partition identities of degrees 8 and 9.

## example

```

/**/ use QQ[x[1..8],y[1..8]];
/**/ HPPI8 := [x[1]^I*x[I+2]*y[2]^(I+1) -y[1]^I*y[I+2]*x[2]^(I+1) | I in 1..6];
/**/ BL := toric(ideal(HPPI8), [x[1],y[2]]);
/**/ len(gens(BL));
340

/**/ use QQ[x[1..9],y[1..9]];
/**/ HPPI9 := [x[1]^I*x[I+2]*y[2]^(I+1) -y[1]^I*y[I+2]*x[2]^(I+1) | I in 1..7];
/**/ BL := toric(ideal(HPPI9), [x[1],y[2]]);
/**/ len(gens(BL));
798

/**/ use R ::= QQ[x,y,z,w];
/**/ toric(ideal(x*z-y^2, x*w-y*z));
ideal(-y^2 +x*z, -y*z +x*w, z^2 -y*w)

/**/ toric(ideal(x*z-y^2, x*w-y*z), [y]);
ideal(-y^2 +x*z, -y*z +x*w, z^2 -y*w)

/**/ use R ::= QQ[x,y,z];
/**/ toric([[1,3,2],[3,4,8]]);
ideal(-x^16 +y^2*z^5)

/**/ toric(mat([[1,3,2],[3,4,8]]));
ideal(-x^16 +y^2*z^5)

```

## I-20.13 transposed

## syntax

```
transposed(M: MAT): MAT
```

### Description

This function returns the transpose of the matrix “M”.

## example

```

/**/ M := mat([[1,2,3],[4,5,6]]);
/**/ M;
matrix(QQ,
  [[1, 2, 3],
   [4, 5, 6]])

/**/ transposed(M);
matrix(QQ,
  [[1, 4],
   [2, 5],
   [3, 6]])

```

## I-20.14 try

## syntax

```
Try C1 UponError E Do C2 EndTry
  where C1, C2 are sequences of commands and E is a variable identifier.
```

## Description

Usually, when an error occurs during the execution of a command, the error is automatically propagated out of the nesting of the evaluation. This can be prevented with the use of “Try..UponError”.

If an error occurs during the execution of the commands “C1”, then it is captured by the command “UponError” and assigned to the variable “E”, and the commands “C2” are executed; the string inside “E” may be retrieved using “GetErrMsg” (I-7.14 pg.113). If no error occurs then the variable “E” and the commands “C2” are ignored.

example

```
/**/ -- Equality function allowing mixed types:
/**/ Define AreEqual(A,B)
/**/   Try
/**/     Return A = B;
/**/   UponError E Do
/**/     Return false;
/**/   EndTry;
/**/ EndDefine;

/**/ AreEqual(0, "zero");
false
/**/ AreEqual(1+2, 3);
true
```

**See Also:** error(I-5.11 pg.87), GetErrMsg(I-7.14 pg.113), syntax(?? pg.??)

## I-20.15 tuples

syntax

```
tuples(S: LIST, N: INT): LIST
```

## Description

This function computes all N-tuples with entries in S. It is equivalent to “S >< S >< ... >< S” [N times].

example

```
/**/ tuples([1, 4, 7], 2);
[[1, 1], [1, 4], [1, 7], [4, 1], [4, 4], [4, 7], [7, 1], [7, 4], [7, 7]]
```

**See Also:** CartesianProduct, CartesianProductList(I-3.5 pg.47), permutations(I-16.5 pg.234), subsets(I-19.47 pg.303)

## I-20.16 Tutorial

syntax

```
?tutorial
```

## Description

Basic Tutorial for CoCoA-5

Use the command “`ciao;`” to get out of CoCoA-5. It is important to type the semicolon “`;`” after the word “`ciao`”. As a rule, you should put a semicolon after every CoCoA-5 command. After receiving the “`ciao`” command, it may occasionally take a few seconds for CoCoA-5 to fully terminate itself.

If CoCoA is busy computing, it will not heed any further commands (including “`ciao;`”) until the computation ends. When CoCoA-5 is ready for a new command it prints out a prompt; if the previous input was incomplete, this is indicated by a different prompt.

If CoCoA-5 is taking too long with a computation you may “interrupt” it (i.e. forcibly end it prematurely); it may take a few seconds for CoCoA-5 to react after you give the interrupt signal. CoCoA-5 will print a prompt when the computation has been stopped; it is then ready to receive new commands. The correct way to interrupt a CoCoA computation depends on the user interface (and operating system).

If you are still stuck inside CoCoA, you can try “`*/ciao;`” instead; note the extra two characters “star” and “slash” at the start. You may need to type this twice.

**See Also:** Tutorial-01: manual(I-20.17 pg.318), Tutorial-02: variables, assignment(I-20.18 pg.318), Tutorial-03: arithmetic operators(I-20.19 pg.319), Tutorial-04: printing(I-20.20 pg.320), Tutorial-05: lists(I-20.21 pg.320), Tutorial-06: rings, polynomials, use command(I-20.22 pg.321), Tutorial-11: homomorphisms(I-20.23 pg.322)

## I-20.17 Tutorial-01: manual

— syntax —

`?tutorial`

### Description

Tutorial-01 for CoCoA-5

CoCoA-5 includes an on-line manual which explains what the various commands and functions do. To consult the manual you use “`?`” followed by a keyword; for instance to find out how to compute GCDs in CoCoA, you could type “`?gcd`”. This will print out the corresponding manual page. Notice at the bottom that there is usually a list of related manual pages (with “`?`” at the start so you can easily cut-and-paste to go to the indicated manual page).

If your keyword does not identify a unique manual page then you will see a list of manual entries which do contain the keyword; again each entry is preceded by “`?`” to make it quicker to use cut-and-paste.

A double-query will simply list the titles of all manual pages containing the keyword: for example try “`??gcd`”.

Unlike for normal commands, there is no need to type a semicolon at the end of a manual query (but you can type one if you want).

**See Also:** Tutorial(I-20.16 pg.317), Tutorial-02: variables, assignment(I-20.18 pg.318), Tutorial-03: arithmetic operators(I-20.19 pg.319), Tutorial-04: printing(I-20.20 pg.320), Tutorial-05: lists(I-20.21 pg.320), Tutorial-06: rings, polynomials, use command(I-20.22 pg.321), Tutorial-11: homomorphisms(I-20.23 pg.322)

## I-20.18 Tutorial-02: variables, assignment

— syntax —

`?tutorial`

### Description

Tutorial-02 for CoCoA-5

CoCoA-5 includes its own imperative programming language. Values you plan to use in future computations

need to be stored in variables; the act of storing a value in a variable is also called assignment. CoCoA-5 uses the colon-equals operator to indicate assignment, for instance “A := 13;” assigns 13 to the variable “A”.

A variable name must start with a letter, and may contain letters, digits, and the underscore character. We recommend using names which are mnemonic (but hopefully not too long).

The most basic types in CoCoA-5 are integers, rationals and strings. A number written in “decimal notation” is automatically converted to a rational number: for example “3.14” is converted into the fraction “157/50”.

#### example

```
/**/ A := 1;           // assign the integer 1 to the variable "A"
/**/ half := 1/2;      // assign rational 1/2 to the variable "half"
/**/ A := 0.333;       // assign the rational 333/1000 to "A"
                        // The previously stored value is overwritten.
/**/ mesg1 := "hi!";   // assign the string "hi!" to variable "mesg1"
```

**See Also:** Tutorial(I-20.16 pg.317), Tutorial-01: manual(I-20.17 pg.318), Tutorial-03: arithmetic operators(I-20.19 pg.319), Tutorial-04: printing(I-20.20 pg.320), Tutorial-05: lists(I-20.21 pg.320), Tutorial-06: rings, polynomials, use command(I-20.22 pg.321), Tutorial-11: homomorphisms(I-20.23 pg.322)

## I-20.19 Tutorial-03: arithmetic operators

### syntax

?tutorial

### Description

Tutorial-03 for CoCoA-5

CoCoA-5 includes its own imperative programming language. The language includes some fairly natural arithmetic operators (usually similar to other computer algebra systems).

The main peculiarities are: colon-equals representing assignment, less-than greater-than representing not-equals, and the words “and”, “or” and “not” representing the boolean operations.

To see a complete list of all infix operators, type the manual query “?operator”

#### example

```
/**/ A := 1;           // assign the integer 1 to the variable "A"
/**/ 1+2*3^4;          // addition, multiplication, power
163
/**/ 1-2/3;            // subtraction, division
1/3
/**/ (A > 0) and (A < 2); // greater-than, less-than, boolean "and"
true
/**/ (A >= 0) or (A <= 2); // greater-or-equal, less-or-equal, boolean "or"
true
/**/ (A = 1) or (A <> 2); // equal, not-equal
true
/**/ not(A > -1 and A < 3); // boolean "not"
false
```

**See Also:** Tutorial(I-20.16 pg.317), Tutorial-01: manual(I-20.17 pg.318), Tutorial-02: variables, assignment(I-20.18 pg.318), Tutorial-04: printing(I-20.20 pg.320), Tutorial-05: lists(I-20.21 pg.320), Tutorial-06: rings, polynomials, use command(I-20.22 pg.321), Tutorial-11: homomorphisms(I-20.23 pg.322)

## I-20.20 Tutorial-04: printing

?tutorial

syntax

### Description

Tutorial-04 for CoCoA-5

The way CoCoA-5 can show us the results of its computation is by (so-called) printing them on the screen.

At top level, if you type in an expression for a computation, CoCoA-5 will evaluate the expression, and then automatically print out the answer. This is convenient for interactive use.

Inside a function definition you must use explicitly a printing command: the two fundamental printing commands are “`println`” and “`print`”. The only difference between them is that the first command also prints out a newline at the end; this is usually what is desired.

The procedure “`indent`” is useful for printing out long lists of values: it prints a newline after each list entry (whereas “`print`” and “`println`” will print all entries on the same line).

To help comprehend the true size of large integers or rationals there is the function “`FloatStr`” which prints out the value using an easy-to-understand floating-point format.

example

```
/**/ 1+2; // an expression at top level
3
/**/ println 1; println 2;
1
2
/**/ print 1; print 2;
12
/**/ println [11,22];
[11, 22]
/**/ indent([11,22]);
[
  11,
  22
]
```

**See Also:** Tutorial(I-20.16 pg.317), Tutorial-01: manual(I-20.17 pg.318), Tutorial-02: variables, assignment(I-20.18 pg.318), Tutorial-03: arithmetic operators(I-20.19 pg.319), Tutorial-05: lists(I-20.21 pg.320), Tutorial-06: rings, polynomials, use command(I-20.22 pg.321), Tutorial-11: homomorphisms(I-20.23 pg.322)

## I-20.21 Tutorial-05: lists

?tutorial

syntax

### Description

Tutorial-05 for CoCoA-5

A convenient way of “putting together” many values in CoCoA-5 is to put them into a LIST. Though the CoCoA-5 name is LIST the data-structure more closely resembles a vector in C++ than a list in C++.

CoCoA-5 does not impose restrictions on the values a LIST may contain; nevertheless it usually makes most sense if the values are all of the same type (e.g. all integers, all polynomials).

A list may be created in several ways. The simplest is to write out the entries explicitly. Another is to start with an empty list, and the append new elements in a loop. There is also a convenient syntax inspired by the mathematical notation for sets.

If you have a list, you can find out how many elements it contains using the function “len”. You can also iterate over the elements of a list using the “foreach” loop command.

— example —

```

/**/ [2,3,5]; // list containing the elements 2,3,5 in that order
[2, 3, 5]
/**/ 1..5; // integer range
[1, 2, 3, 4, 5]
// Now create a list using "append":
/**/ L := []; // start with the empty list in L
/**/ for i := 1 to 5 do append(ref L, i^2); endfor;
/**/ println L;
[1, 4, 9, 16, 25]
/**/ [ n in L | IsEven(n)]; // list containing even values
[4, 16]
/**/ [ n^2 | n in L and n < 10];
[1, 16, 81]
/**/ len(L);
5
/**/ foreach n in L do print n, " "; endforeach;
1 4 9 16 25

```

**See Also:** Tutorial(I-20.16 pg.317), Tutorial-01: manual(I-20.17 pg.318), Tutorial-02: variables, assignment(I-20.18 pg.318), Tutorial-03: arithmetic operators(I-20.19 pg.319), Tutorial-04: printing(I-20.20 pg.320), Tutorial-06: rings, polynomials, use command(I-20.22 pg.321), Tutorial-11: homomorphisms(I-20.23 pg.322)

## I-20.22 Tutorial-06: rings, polynomials, use command

— syntax —

```
?tutorial
```

### Description

Tutorial-06 for CoCoA-5

When you want to do a computation in CoCoA-5, the first thing you need to do is tell CoCoA-5 in which ring to compute. The “use” (I-21.6 pg.327) command informs CoCoA-5 about this. The most convenient method does two things at once: it creates the polynomial ring, and then chooses that ring as the “current ring”.

Once the correct current ring has been selected, you may type in polynomials using a natural syntax (with the caveat that you must use “\*” to denote all products (e.g. between coefficients and indeterminates, or even between powers of indeterminates).

The most common coefficient fields are the rationals (denoted by “QQ”) and small prime finite fields (denoted by “ZZ/(p)”).

— example —

```

/**/ use P := QQ[x,y]; // polys in x,y with coefficients in QQ
/**/ (x+y)^2;
x^2 + 2*x*y + y^2
/**/ use ZZ/(2)[a,b]; // polys in a,b with coefficients in ZZ/(2)
/**/ (a+b)^2;
a^2 + b^2
/**/ use QQ[x,y,z],lex; // polys in x,y,z, coeffs in QQ, term order "lex"

```

```
/**/ x+y^2;
x + y^2
```

**See Also:** Tutorial(I-20.16 pg.317), Tutorial-01: manual(I-20.17 pg.318), Tutorial-02: variables, assignment(I-20.18 pg.318), Tutorial-03: arithmetic operators(I-20.19 pg.319), Tutorial-04: printing(I-20.20 pg.320), Tutorial-05: lists(I-20.21 pg.320), Tutorial-11: homomorphisms(I-20.23 pg.322)

## I-20.23 Tutorial-11: homomorphisms

— syntax —

```
?tutorial
```

### Description

Tutorial-11 for CoCoA-5

CoCoA-5 lets you create ring homomorphisms; these are useful for various purposes such as "moving" a value from one ring to another.

A homomorphism from a polynomial ring must state what the images of the indeterminates are; and, if the homomorphism is not an algebra homomorphism, how the coefficient ring is mapped.

— example —

```
/**/ P1 ::= QQ[x,y]; // polys in x,y with coefficients in QQ
/**/ P2 ::= QQ[a,b]; // polys in a,b with coefficients in ZZ/(2)
/**/ use P2; IndetImages := [a^2, b^3];
/**/ phi := PolyAlgebraHom(P1, P2, IndetImages);
/**/ use P1;
/**/ f := 2*x^2 + 3*y + 4;
/**/ phi(f);
2*a^4 + 3*b^3 + 4
```

**See Also:** Tutorial(I-20.16 pg.317), Tutorial-01: manual(I-20.17 pg.318), Tutorial-02: variables, assignment(I-20.18 pg.318), Tutorial-03: arithmetic operators(I-20.19 pg.319), Tutorial-04: printing(I-20.20 pg.320), Tutorial-05: lists(I-20.21 pg.320), Tutorial-06: rings, polynomials, use command(I-20.22 pg.321)

## I-20.24 type

— syntax —

```
type(E: OBJECT): TYPE
```

### Description

This function returns the data type of E.

— example —

```
/**/ L := [1,"a",2,"b",3,"c"];
/**/ [ X in L | type(X)=INT ];
[1, 2, 3]

/**/ type(type(INT)); -- Type returns a value of type TYPE
TYPE

/**/ CurrentTypes();
[BOOL, ERROR, FUNCTION, ...]
```

**See Also:** `CurrentTypes`([I-3.57](#) pg.69)



# Chapter I-21

## U

### I-21.1 UnivariateIndetIndex

syntax

```
UnivariateIndetIndex(F: RINGELEM): INT
```

#### Description

This function returns 0 if the polynomial F is not univariate otherwise it returns the indeterminate index of F.

NOTE: If F is a constant, it returns 1.

example

```
/**/ use R := QQ[x,y,z];
/**/ UnivariateIndetIndex(3*x^4-2*x-1);
1

/**/ UnivariateIndetIndex(x-y-1);
0

/**/ UnivariateIndetIndex(one(R));
1
```

**See Also:** [indet\(I-9.21 pg.143\)](#), [IndetSubscripts\(I-9.25 pg.145\)](#), [IndetIndex\(I-9.22 pg.143\)](#), [IndetName\(I-9.23 pg.144\)](#), [indets\(I-9.24 pg.144\)](#), [NumIndets\(I-14.37 pg.224\)](#)

### I-21.2 UniversalGBasis

syntax

```
UniversalGBasis(I: IDEAL): LIST of RINGELEM
```

#### Description

Returns a universal Groebner basis of the input IDEAL “I”.

This function was called “UniversalGroebnerBasis” up to version CoCoA-5.1.4.

example

```
-- The ideal generated by the 3x3 minors of 3x4 matrix of indeterminates
-- has 96 marked reduced Groebner bases
/**/ use R := QQ[a,b,c,d,e,f,g,h,i,j,k,l];
/**/ I := ideal(minors(mat([[a,b,c,d],[e,f,g,h],[i,j,k,l]]),3));
/**/ indent(UniversalGBasis(I));
```

```
[d*g*j -c*h*j -d*f*k +b*h*k +c*f*l -b*g*l,
 d*g*i -c*h*i -d*e*k +a*h*k +c*e*l -a*g*l,
 d*f*i -b*h*i -d*e*j +a*h*j +b*e*l -a*f*l,
 c*f*i -b*g*i -c*e*j +a*g*j +b*e*k -a*f*k
]
```

**See Also:** [GroebnerFanIdeals\(I-7.33 pg.119\)](#)

## I-21.3 unprotect

— syntax —

```
unprotect X;
```

### Description

This command undoes the effect of the “`protect`” ([I-16.38 pg.250](#)) command; once a variable has been unprotected, it may be assigned to freely.

— example —

```
/**/ X := 1;
/**/ protect X;    --> cannot assign to X henceforth
/**/
/**/ unprotect X;  --> remove protection, X may be assigned to now
/**/ X := 2;
```

**See Also:** `protect` ([I-16.38 pg.250](#))

## I-21.4 Unset [OBSOLETE]

— syntax —

```
[OBSOLETE]
```

### Description

[OBSOLETE]

## I-21.5 untagged

— syntax —

```
untagged(E:TAGGED_OBJECT):UNTAGGED_OBJECT
```

### Description

This function strips an object `E` of its tag, if any. “`@E`” is equivalent to “`untagged(E)`”.

Tags are used for pretty printing of objects. See the reference listed below.

— example —

```
/**/ L := [1,2,3];
/**/ M := tagged(L,"MyTag");
```

```

/**/ type(L);
LIST

/**/ type(M);
TAGGED("MyTag")

/**/ type(untagged(M));
LIST

```

**See Also:** Printing a Tagged Object([III-16.2](#) pg.429), tag([I-20.1](#) pg.311), tagged([I-20.2](#) pg.311)

## I-21.6 use

### syntax

```

use R
use RingDefn
use R ::= RingDefn

where R is a RING, and RingDefn is a ring definition.

```

### Description

This command works only at top-level; it makes a ring active, i.e. it makes that ring the “*current ring*”. The command will also let you create a new ring, and make it active immediately “`use NewR ::= RingDefn;`” where “`RingDefn`” is a ring definition; this is a shorthand for “`NewR ::= RingDefn; use NewR;`”

This command cannot be called inside a function, and it is never necessary (if you write clean programs ;-). See also “`RingElem`” ([I-18.43](#) pg.273) for reading elements without “`use`”. In CoCoA-5 you can define new rings, return rings, assign rings and pass rings as arguments (this was not possible in CoCoA-4).

### example

```

/**/ use S ::= QQ[x,y,z];
/**/ Print CurrentRing;
RingDistrMPolyClean(QQ, 3)
/**/ indets(CurrentRing);
[x, y, z]

/**/ use QQ[u]; -- can be used w/out a ring identifier
/**/ indets(CurrentRing);
[u]

/**/ define SumInAnotherRing(N)
/**/   K := NewRingTwinFloat(128); -- 128 bits of precision
/**/   P := K[x[1..N]], Lex;
/**/   return sum(indets(P));
/**/ enddefine;

/**/ SumInAnotherRing(4);
x[1] +x[2] +x[3] +x[4]
/**/ CoeffRing(RingOf(It));
RingTwinFloat(AccuracyBits=128, BufferBits=128, NoiseBits=32)

```

**See Also:** Introduction to RINGHOM([III-10.1](#) pg.409), CurrentRing([I-3.56](#) pg.68), RingOf([I-18.46](#) pg.275), RingElem([I-18.43](#) pg.273)



# Chapter I-22

## V

### I-22.1 valuation [OBSOLETE]

syntax

[OBSOLETE]

#### Description

Renamed “FactorMultiplicity” (I-6.4 pg.94).

**See Also:** FactorMultiplicity(I-6.4 pg.94)

### I-22.2 VerbosityLevel

syntax

VerbosityLevel(): INT

#### Description

This function returns the current CoCoA verbosity level.

User defined functions may check this value when deciding whether to print out some internal progress messages: normally printing should happen only if “VerbosityLevel()” is higher than some threshold value.

Various functions in CoCoALib and in the CoCoA-5 packages do this, but they all have threshold values greater than 9. So threshold values from 1 to 9 may be used in user functions without triggering any verbosity from CoCoA.

example

```
/**/ define SimpleFn(n)
/**/   if VerbosityLevel() >= 4 then
/**/     println "SimpleFn: input type is ", type(n);
/**/   endif;
/**/   return n^2;
/**/ enddefine;

/**/ SetVerbosityLevel(9);
/**/ SimpleFn(ideal(indets(CurrentRing)));
SimpleFn: input type is IDEAL
ideal(z^2, y*z, x*z, y^2, x*y, x^2)
```

```

/**/ SetVerbosityLevel(0); --> unset verbosity
/**/ SimpleFn(ideal(indets(CurrentRing)));
ideal(z^2, y*z, x*z, y^2, x*y, x^2)

```

**See Also:** SetVerbosityLevel([I-19.11](#) pg.286)

## I-22.3 VersionInfo

— syntax —

```
VersionInfo(): RECORD
```

### Description

This function returns a record with various information about CoCoA and CoCoALib (the mathematical core of CoCoA)

— example —

```

/**/ indent(VersionInfo());
record[
  CoCoALibVersion := "0.99***",
  CoCoAVersion := "5.*.*",
  CompilationDate := ....,
  ...

```

**See Also:** CoCoALib([II-9.1](#) pg.365), RelNotes([I-18.30](#) pg.268)

# Chapter I-23

## W

### I-23.1 wdeg

— syntax —

```
wdeg(F: RINGELEM): LIST
```

#### Description

This function returns the multi-weighted degree of F, as determined by the matrix weights of the polynomial ring of F. The function “deg” (I-4.6 pg.74) returns the standard degree.

NOTE: In CoCoA-4 “deg” (I-4.6 pg.74) gave the weight given by only the first row of the weights matrix.

— example —

```
/**/ M := matrix([[2,3,4], [1,0,2], [1,0,0]]);
/**/ P := NewPolyRing(QQ, "x,y,z", M, 1); -- GradingDim=1
/**/ use P;
/**/ wdeg(x*y^2+y);
[8]
/**/ P := NewPolyRing(QQ, "x,y,z", M, 2); -- GradingDim=2
/**/ use P;
/**/ wdeg(x*y^2+y);
[8, 1]
/**/ deg(x*y^2+y);
3

/**/ P4 := NewFreeModule(P,4); -- the default module ordering is TPos
/**/ wdeg(ModuleElem(P4, [0, x, y^2, x^2]));
[6, 0]

/**/ LT(ModuleElem(P4, [0, x, y^2, x^2]));
[0, 0, y^2, 0]
```

**See Also:** deg(I-4.6 pg.74), LF(I-12.9 pg.183)

### I-23.2 WeightsMatrix [OBSOLESCENT]

— syntax —

```
WeightsMatrix(R: RING): MAT
```

## Description

This function is now called “GradingMat” ([I-7.30 pg.117](#)).

**See Also:** [deg\(I-4.6 pg.74\)](#), [wdeg\(I-23.1 pg.331\)](#)

## I-23.3 while

syntax

```
While B Do C EndWhile
```

where B is a boolean expression and C is a sequence of commands.

## Description

The command sequence C is repeated until B evaluates to False.

example

```
/**/ N := 0;
/**/ while N <= 5 do
/**/   PrintLn 2, "^", N, " = ", 2^N;
/**/   N := N+1;
/**/ EndWhile;
2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
2^4 = 16
2^5 = 32
```

**See Also:** [for\(I-6.17 pg.100\)](#), [foreach\(I-6.18 pg.101\)](#), [repeat\(I-18.33 pg.269\)](#), [syntax\(?? pg.??\)](#)

## I-23.4 WithoutNth

syntax

```
WithoutNth(L: LIST, N: INT): LIST
```

## Description

This function returns the list obtained by removing the “N”-th component of the list “L”. The list “L” itself is not changed; compare with “[remove](#)” ([I-18.32 pg.268](#)).

example

```
/**/ L := [1,2,3,4,5];
/**/ WithoutNth(L,3);
[1, 2, 4, 5]
```

**See Also:** [remove\(I-18.32 pg.268\)](#)

## I-23.5 WLog [OBSOLETE]

syntax

```
[OBSOLETE]
```

**Description**

[OBSOLETE] This function returns the weighted list of exponents of the leading term of  $F$ , as determined by the first row of the weights matrix.

**See Also:** `exponents`([I-5.16](#) pg.90)



# Chapter I-24

## X

### I-24.1 **XelMat**

— **syntax** —

```
XelMat(N: INT): MAT
```

#### **Description**

This function return the matrix defining a standard term-ordering.

— **example** —

```
/**/ XelMat(3);  
matrix(ZZ,  
  [[0, 0, 1],  
   [0, 1, 0],  
   [1, 0, 0]])
```

**See Also:** OrdMat([I-15.10](#) pg.231), Term Orderings([III-9.5](#) pg.404), StdDegRevLexMat([I-19.38](#) pg.299), StdDegLexMat([I-19.37](#) pg.299), LexMat([I-12.7](#) pg.182), RevLexMat([I-18.41](#) pg.272)



# Chapter I-25

## Z

### I-25.1 zero

syntax

```
zero(R: RING): RINGELEM
```

#### Description

This function return the additive identity of a ring. For when you want to force the integer “0” to be a “RINGELEM”.

example

```
/**/ P := ZZ/(101)[x,y,z];

/**/ N := 0; Print N, " of type ", type(N);
0 of type INT
/**/ N := zero(P); Print N, " of type ", type(N);
0 of type RINGELEM
/**/ N := 300*0; Print N, " of type ", type(N);
0 of type INT
/**/ N := 300*zero(P); Print N, " of type ", type(N);
0 of type RINGELEM

/**/ F := NewFreeModule(P, 3);
/**/ zero(F);
[0, 0, 0]
```

**See Also:** [one\(I-15.1 pg.227\)](#), [IsZero\(I-9.90 pg.170\)](#)

### I-25.2 ZeroMat

syntax

```
ZeroMat(R: RING, NumRows: INT, NumCols: INT): MAT
```

#### Description

This function returns the “NumRows x NumCols” zero matrix with entries in “R”.

example

```
/**/ use R := QQ[x,y,z];
/**/ ZeroMat(QQ, 1, 3); --> same as NewMatFilled(1,3, 0)
matrix(QQ,
```

```

[[0, 0, 0]]
/**/ ZeroMat(R, 1, 3); --> same as NewMatFilled(1,3, zero(R))
matrix( /*RingDistrMPolyClean(QQ, 3)*/
[[0, 0, 0]]

```

**See Also:** `matrix`([I-13.10](#) pg.195), `IdentityMat`([I-9.9](#) pg.137), `NewMatFilled`([I-14.7](#) pg.211)

## I-25.3 ZPQ

syntax

```

ZPQ(F: RINGELEM): RINGELEM
ZPQ(F: LIST of RINGELEM): LIST of RINGELEM
ZPQ(I: IDEAL): IDEAL

```

### Description

\*\*\*\*\* NOT YET IMPLEMENTED \*\*\*\*\*

The function “ZPQ” maps a polynomial with finite field coefficients into one with rational (actually, integer) coefficients. It is not uniquely defined mathematically, and currently for each coefficient the least non-negative equivalent integer is chosen. Users should not rely on this choice, though any change will be documented.

See “QZP” ([I-17.7](#) pg.253) for more details.

example

```

use R := QQ[x,y,z];
F := 1/2*x^3 + 34/567*x*y*z - 890; -- a poly with rational coefficients
use S := ZZ/(101)[x,y,z];
QZP(F);                                -- compute its image with coeffs in ZZ/(101)
-50x^3 - 19xyz + 19
-----
G := It;
use R;
ZPQ(G);    -- now map that result back to QQ[x,y,z] it is NOT the same as F...
51x^3 + 82xyz + 19
-----

```

**See Also:** `BringIn`([I-2.13](#) pg.43)

## I-25.4 ZZ

syntax

```

ZZ

```

### Description

This system variable is constant; its value is the ring of integers. Its name is protected so that it cannot be re-assigned to any other value.

example

```

/**/ type(5);
INT
/**/ type(RingElem(ZZ, 5));
RINGELEM

```

```
/**/ P ::= ZZ/(101)[x,y,z]; -- coeffs in quotient ring
```

**See Also:** QQ([I-17.1](#) pg.[251](#)), Quotient Rings([III-9.7](#) pg.[406](#))



## Part II

# The CoCoA Programming Language



## Chapter II-1

# Introduction to CoCoA Programming

### II-1.1 An Overview of CoCoA Programming

The CoCoA system includes a full-fledged high level programming language, CoCoALanguage, complete with loops, branching, scoping of variables, and input/output control. The language is used whenever one issues commands during a CoCoA session. A sequence of commands may be stored in a text file and then read into a CoCoA session using the “`source`” (I-19.26 pg.293) command.

The most important construct in CoCoA programming is the user-defined function, created with “`define`” (I-4.4 pg.72). A user-defined function can take any number of arguments, of any types, perform CoCoA commands, and return values. Collections of these functions can be stored in text files, as mentioned in the preceding paragraph, or formed into CoCoA “*packages*”, to be made available for general use.

See “All CoCoA commands” (II-1.2 pg.343).

### II-1.2 All CoCoA commands

This is a complete list of all CoCoA commands:

<code>break</code>	break out of a loop command
<code>ciao</code>	quit CoCoA
<code>continue</code>	continue directly with next loop iteration
<code>define</code>	define a function
<code>describe</code>	information about an object
<code>exit</code>	quit CoCoA
<code>for</code>	loop command
<code>foreach</code>	loop command
<code>if</code>	conditional statement
<code>print</code>	print the value of an expression
<code>print on</code>	print to an output stream
<code>println</code>	print the value of an expression
<code>protect</code>	protect a variable from being overwritten
<code>quit</code>	quit CoCoA
<code>repeat</code>	loop command
<code>return</code>	exit from a function
<code>source</code>	read commands from a file or device
<code>SourceRegion</code>	read commands from a region in a file
<code>try</code>	try command sequence, catch any errors
<code>unprotect</code>	remove protection from a variable
<code>use</code>	command for making a ring active
<code>while</code>	loop command



## Chapter II-2

# Language Elements

### II-2.1 Character Set and Special Symbols

The CoCoA character set consists of the 26 lower case letters, the 26 upper case letters, the 10 digits and the special characters listed in the table below. Note that the special character “|” looks a bit different on some keyboards (its ASCII code is 124).

	blank	_	underscore	(	left parenthesis	
	+	plus	=	equal	)	right parenthesis
	-	minus	<	less than	[	left bracket
	*	asterisk	<	greater than	[	right bracket
	/	slash		vertical bar	'	single quote
	:	colon	.	period	" "	double quote
	^	caret	;	semicolon		
	,	comma	%	percent		

Special Characters

The character-groups listed in the table below are special symbols in CoCoA

	:=	assign	..	range	
	<<	input from	//	start line comment	
	<>	not equal	--	start line comment	
	><	Cartesian product	::=	ring definition	
	<=	less than or equal to	/*	start embedded comment	
	>=	greater than or equal to	*/	end embedded comment	

Special Character-groups

### II-2.2 Identifiers

There are two types of identifiers or names.

- \* Identifiers of ring indeterminates (see “NewPolyRing” (I-14.8 pg.212))
- \* Predefined or user-defined names (functions and CoCoALanguage variables).

**See Also:** Indeterminates(III-9.4 pg.404)

## II-2.3 Reserved Names

\*\*\*\*\* NOT YET UPDATED TO CoCoA-5: follow with care \*\*\*\*\*

The names in the following tables are reserved and cannot be used otherwise. The names in the first table are case insensitive (e.g. CLEAR, Clear and cLEaR are all reserved). The names in the second table are case sensitive.

...work in progress...

Alias	And	Block	Ciao	Define	
Describe	Do	Elif	Else	End	
EndBlock	EndTry		EndDefine	EndFor	
EndForeach	EndIf	EndPackage	EndRepeat	EndUsing	
EndWhile	Eof	False	For	Foreach	
Global	Help	If	In	IsIn	
NewLine	Not	On	Or	Package	
Print	PrintLn	Quit	Repeat	Record	
Return	Set	Skip	Source	Step	
Then	Time	To	True	Unset	
Until	use	Using	Var	While	
QQ	ZZ				

Case insensitive reserved names

BOOL	DegLex	DegRevLex	DEVICE	ERROR	
FUNCTION	IDEAL	INT	LIST	Lex	
MAT	MODULE	NULL	Null	PANEL	
POLY	PosTo	RAT	RATFUN	RING	
STRING	TAGGED	ToPos	TYPE	MODULEELEM	
Xel	ZMOD				

Case sensitive reserved names

## II-2.4 Comments

End-of-line comments in CoCoA start with either “--” or “//”; all text up to the end of the line is considered comment. CoCoA also allows embedded comments; these begin with the symbol “/\*” and end with the symbol “\*/”. CoCoA ignores the contents of a comment, and treats it as if it were just a space.

example

```
/**/ // This is an end-of-line comment
/**/ Print 1+1; -- a command followed by an end-of-comment
2
/**/ A := [1 /*x-coord*/, 2 /*y-coord*/ ]; --> embedded comments
```

# Chapter II-3

## Operators

### II-3.1 CoCoA Operators: introduction

In CoCoA there are 5 main types of operators: algebraic operators, relational operators, boolean operators, selection operators, and the range operator. There is also an n-ary operator “><” for forming Cartesian products of lists and an operator “:=” used in defining rings.

The meaning of an operator depends on the types of its operands; the “+” in the expression “A + B” represents the sum of polynomials, or of ideals, or of matrices, etc. according to the type of A and B.

**See Also:** operators, shortcuts(I-0.1 pg.25)

### II-3.2 Algebraic Operators

The algebraic operators are:

+ - \* / : ^

The following table shows which operations the system can perform between two objects of the same or of different types; the first column lists the type of the first operand and the first row lists the type of the second operand. So, for example, the symbol “:” in the box on the seventh row and fourth column means that it is possible to divide an ideal by a polynomial.

	INT	RAT	RINGELEM	MODULEELEM	IDEAL	MODULE	MAT	LIST
INT	+*/^	+*/	+*/	*	*	*	*	*
RAT	+*/^	+*/	+*/	*	*	*	*	*
RINGELEM	+*/^	+*/	+*/	*	*	*	*	*
MODULEELEM	*	*	*	+-				
IDEAL	*^	*	*		++:	*		
MODULE	*	*	*		*	+:		
MAT	*^	*	*				+-*	
LIST	*	*	*					+-

Algebraic operators

Remarks:

\* Let F and G be two polynomials. If F is a multiple of G, then F/G is the polynomial obtained from the division of F by G, otherwise F/G is a rational function (common factors are simplified). The functions “div” (I-4.20 pg.81) and “mod” (I-13.26 pg.202) can be used to get the quotient and the remainder of a polynomial division.

\* Let  $L_1$  and  $L_2$  be two lists of the same length. Then  $L_1 + L_2$  is the list obtained by adding  $L_1$  to  $L_2$  componentwise.

\* If  $I$  and  $J$  are both ideals or both modules, then  $I : J$  is the ideal consisting of all polynomials  $f$  such that  $fg$  is in  $I$  for all  $g$  in  $J$ .

### II-3.3 Relational Operators

**See Also:** Equality Test(I-5.9 pg.86), Comparison Operators(I-3.34 pg.59), IsIn(I-9.53 pg.157)

### II-3.4 Selection Operators

The selection operators are

`[]` .

Let  $N$  be of type INT and let  $L$  be of type STRING, MODULEELEM, LIST, or MAT. Then the meaning of  $L[N]$  depends on the type of  $L$  as explained in the following table:

Type of L	Meaning of L[N]
STRING	string consisting of the N-th character of L.
MODULEELEM	N-th component of L
LIST	N-th element of L
MAT	N-th element of L

Selection Operator

If  $N$  is an identifier and  $L$  is of type RECORD, then “ $L.N$ ” indicates the object contained in the field  $N$  of the record  $L$  (see “record” (I-18.23 pg.264)).

**See Also:** record(I-18.23 pg.264), List Constructors(III-5.2 pg.388)

### II-3.5 Range Operator

If “ $M$ ” and “ $N$ ” are of type “INT”, then the expression: “ $M \dots N$ ” returns

- \* the list “[ $M$ ,  $M+1$ , ...,  $N$ ]” if  $M \leq N$ ;
- \* the empty list, “[]”, otherwise.

NOTE: see example for how to select a sub-range of a list

NOTE: CoCoA does not allow “ $N..M$ ” to produce lists longer than  $10^7$  values.

If “ $x$ ” and “ $y$ ” are indeterminates in a ring, then “ $x \dots y$ ” gives the indeterminates between “ $x$ ” and “ $y$ ” in the order they appear in the definition of the ring.

example

```

/**/ 1..10;
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

/**/ use R ::= QQ[x,y,z,a,b,c,d];
/**/ z..c;
[z, a, b, c]

/**/ L := [11, 22, 33, 44, 55];
/**/ PartOfL := L[2]..L[4]; --> probably *NOT* what you want!
/**/ PartOfL := [ L[k] | k in 2..4 ]; --> OK, this is RIGHT!
```

**See Also:** CoCoA Operators(?? pg.??), List Constructors(III-5.2 pg.388), LIST(III-5 pg.387)

## Chapter II-4

# Evaluation and Assignment

### II-4.1 Evaluation

An expression is by itself a valid command. The effect of this command is that the expression is evaluated in the current ring and its value is displayed.

The evaluation of an expression in CoCoA is normally performed in a full recursive evaluation mode. Usually the result is the fully evaluated expression.

The result of the evaluation is automatically stored in the variable “It” ([I-9.95 pg.172](#)).

— example —

```
/**/ 2 + 2;  
4  
/**/ It + 3;  
7  
/**/ It;  
7  
/**/ X := 5;  
/**/ It;  
7
```

The command “X := 5” is an assignment, not an evaluation; so it does not change the value of the variable “It” ([I-9.95 pg.172](#)).

If an error occurs during the evaluation of an expression, then the evaluation is interrupted and the user is notified about the error.

### II-4.2 Assignment

An assignment command has the form

$$L := E$$

where L is a variable and E is an expression. The assignment command binds the result of the evaluation of the expression E to L in the working memory.

— example —

```
/**/ use R ::= QQ[t,x,y,z];  
/**/ I := ideal(x,y);  
  
/**/ M := 5; N := 8;  
/**/ T := M+N;  
/**/ T;
```

```
13
/**/ T := T+1;  -- note that T occurs on the right, also
/**/ T;
14

/**/ L := [1,2,3];
/**/ L[2] := L[3];
/**/ L;
[1, 3, 3]

/**/ P := record[F := x*z];
/**/ P.Degree := Deg(P.F);
/**/ P;
record[Degree := 2, F := x*z]
```

## Chapter II-5

# Flow Control: Conditional Statements and Loops

### II-5.1 Commands and Functions for Branching

The following are the CoCoA commands for constructing conditional statements:

`if` conditional statement

### II-5.2 Commands and Functions for Loops

The following are the commands and functions for loops:

<code>break</code>	break out of a loop command
<code>continue</code>	continue directly with next loop iteration
<code>for</code>	loop command
<code>foreach</code>	loop command
<code>repeat</code>	loop command
<code>return</code>	exit from a function
<code>while</code>	loop command



## Chapter II-6

# Verbosity and interrupt

### II-6.1 Introduction to verbosity and interrupt

Various functions defined in CoCoALib and in the CoCoA packages print out some internal progress when the global “`VerbosityLevel`” ([I-22.2](#) pg.329) is higher than some value (see their specific manual for the values, anyway not less than 10). See also “`SetVerbosityLevel`” ([I-19.11](#) pg.286).

A computation which appears to take too long may be interrupted by typing “`C-c`” (control-c), or, in Emacs, “`C-c C-c`”. The state of the memory after an interrupt is as it was before calling the interrupted function, thanks to the exception-safe design of CoCoALib.

### II-6.2 Commands and Functions implementing Verbosity

The following are the commands and functions implementing verbosity:

<code>ApproxPointsNBM</code>	Numerical Border Basis of ideal of points
<code>ApproxSolve</code>	Approximate real solutions for polynomial system
<code>CallOnGroebnerFanIdeals</code>	apply a function to Groebner fan ideals
<code>GBasis</code>	calculate a Groebner basis
<code>gin</code>	generic initial ideal
<code>GroebnerFanIdeals</code>	all reduced Groebner bases of an ideal
<code>ImplicitHypersurface</code>	implicitization of hypersurface
<code>MinPolyQuot, MinPolyQuotDef, MinPolyQuotElim, MinPolyQuotMat</code>	compute a minimal polynomial
<code>rgin</code>	generic initial ideal wrt StdDegRevLex
<code>SetVerbosityLevel</code>	set the verbosity level
<code>UniversalGBasis</code>	universal Groebner basis of the input ideal
<code>VerbosityLevel</code>	verbosity level

### II-6.3 Commands and Functions implementing interruption

All functions implemented in CoCoA language, i.e. all user-defined functions and those defined in packages, are interruptible.

Moreover the following CoCoALib functions are interruptible:

<code>GBasis</code>	calculate a Groebner basis
---------------------	----------------------------

ImplicitHypersurface

implicitization of hypersurface

MinPolyQuot, MinPolyQuotDef, MinPolyQuotElim, MinPolyQuotMat

compute a minimal polynomial

## Chapter II-7

# Input/Output

### II-7.1 Introduction to IO

\*\*\*\*\* NOT YET UPDATED TO CoCoA-5: follow with care \*\*\*\*\*

Input and output is implemented in CoCoA through the use of “*devices*”. At present, the official devices are: (1) standard IO (the CoCoA window), (2) text files, and (3) strings. What this means is that it is possible to read from or write to any of these places. The cases are discussed separately, below. Text files may be read verbatim or—with the “*source*” (I-19.26 pg.293) command—be executed as CoCoA commands.

### II-7.2 Standard IO

\*\*\*\*\* NOT YET UPDATED TO CoCoA-5: follow with care \*\*\*\*\*

Standard IO is what takes places normally when one interacts with CoCoA. CoCoA accepts and interprets strings typed in by the user and prints out expressions. If *E* is a CoCoA object, then the command

*E*;

causes the value of *E* to be printed to the CoCoA window. One may also use the functions “*print*” (I-16.29 pg.245) and “*println*” (I-16.34 pg.247) for more control over the format of the output.

The official devices that are being used here are “DEV.STDIN” and “DEV.OUT”. So for instance, the commands “*Get*” (I-7.10 pg.112) and “*print on*” (I-16.30 pg.245) can be used with the standard devices although they are really meant to be used with the other devices. “*Print E On DEV.OUT*” is synonymous with “*Print E*”. Also, one may use “*Get(DEV.STDIN,10)*”, for example, to get the next 10 characters typed in the CoCoA window. Thus, clever use of “*Get*” (I-7.10 pg.112) will allow your user-defined functions to prompt the user for input, but normal practice is to pass variables to a function as arguments to that function.

### II-7.3 File IO

\*\*\*\*\* NOT YET UPDATED TO CoCoA-5: follow with care \*\*\*\*\*

To print CoCoA output to a file, one first opens the file with “*OpenOFile*” (I-15.5 pg.229) then prints to the file using “*print on*” (I-16.30 pg.245).

To receive verbatim input from a file, one first opens the file with “*OpenIFile*” (I-15.2 pg.227), then gets characters from the file with “*Get*” (I-7.10 pg.112). Actually, “*Get*” (I-7.10 pg.112) gets a list of ASCII codes for the characters in the file. These can be converted to real characters using the function “*ascii*” (I-1.18 pg.35).

example

```
D := OpenOFile("my-file"); -- open text file with name "my-file",
                           -- creating it if necessary
```

```

Print "hello world" On D; -- append "hello world" to my-file
Close(D); -- close the file
D := OpenIFile("my-file"); -- open "my-file"
Get(D,10); -- get the first ten characters, in ASCII code
[104, 101, 108, 108, 111, 32, 119, 111, 114, 108]
-----
  ascii(It); -- convert the ASCII code
hello worl
-----
Close(D);

```

To read and execute a sequence of CoCoA commands from a text file, one uses the “source” (I-19.26 pg.293) command. For instance, if the file “MyFile.coc” contains a list of CoCoA commands, then

```
Source "MyFile.cocoa";
```

reads and executes the commands.

**See Also:** [ascii\(I-1.18 pg.35\)](#), [close\(I-3.16 pg.51\)](#), [Get\(I-7.10 pg.112\)](#), [OpenIFile\(I-15.2 pg.227\)](#), [OpenOFile\(I-15.5 pg.229\)](#), [OpenLog\(I-15.4 pg.228\)](#), [CloseLog\(I-3.17 pg.51\)](#), [print on\(I-16.30 pg.245\)](#), [source\(I-19.26 pg.293\)](#)

## II-7.4 String IO

\*\*\*\*\* NOT YET UPDATED TO CoCoA-5: follow with care \*\*\*\*\*

To print CoCoA output to a string, one may use “OpenOString” (I-15.6 pg.230) to “open” the string, then “print on” (I-16.30 pg.245) to write to it. To read from a string, one may open the string for input with “OpenIString” (I-15.3 pg.228) then get characters from it with “Get” (I-7.10 pg.112).

example

```

S := "hello world";
D := OpenIString("", S); -- open the string S for input to CoCoA
                        -- the first argument is just a name for the device
L := Get(D,7); -- read 7 characters from the string
L; -- ASCII code
[104, 101, 108, 108, 111, 32, 119]
-----
  ascii(L); -- convert ASCII code to characters
hello w
-----
Close(D); -- close device D
D := OpenOString(""); -- open a string for output from CoCoA
L := [1,2,3]; -- a list
Print L On D; -- print to D
D;
record[Name := "", Type := "OString", Protocol := "CoCoALanguage"]
-----
S := Cast(D, STRING); -- S is the string output printed on D
S; -- a string
[1, 2, 3]
Print " more characters" On D; -- append to the existing output string
Cast(D, STRING);
[1, 2, 3] more characters
-----

```

There are usually more direct ways to collect results in strings. For instance, if the output of a CoCoA command is not already of type STRING, one may convert it to a string using “sprint” (I-19.29 pg.294).

## II-7.5 Commands and Functions for IO

The following are commands and functions for input/output:

<code>block</code>	group several commands into a single command
<code>close</code>	close an output stream
<code>CloseLog</code>	close a log of a CoCoA session
<code>format</code>	convert object to formatted string
<code>Get</code>	read characters from a device
<code>IO.SprintTrunc</code>	convert to a string and truncate
<code>latex</code>	LaTeX formatting
<code>NewFreeModule</code>	create a new FreeModule
<code>NewLine</code> [OBSOLESCENT]	[OBSOLESCENT] string containing a newline
<code>OpenIFile</code>	open input file
<code>OpenIString</code>	open input string
<code>OpenLog</code>	open a log of a CoCoA session
<code>OpenOFile</code>	open output file
<code>OpenOString</code>	open output string
<code>OpenSocket</code>	open a socket connection
<code>print</code>	print the value of an expression
<code>print on</code>	print to an output stream
<code>println</code>	print the value of an expression
<code>source</code>	read commands from a file or device
<code>SourceRegion</code>	read commands from a region in a file
<code>sprint</code>	convert to a string
<code>tag</code>	returns the tag string of an object
<code>tagged</code>	tag an object for pretty printing
<code>untagged</code>	untag an object



## Chapter II-8

# CoCoA Packages

### II-8.1 Introduction to Packages

User-defined functions may be saved in separate files and read into a CoCoA session using the “**source**” (I-19.26 pg.293) command. If one sources several such files or, especially, if a file is to be made available for general use, a possible problem arises from conflicting function names. If two functions with the same name are read into a CoCoA session, only the one last read survives. To avoid this, functions may be collected in “*packages*”.

A CoCoA package is essentially a list of functions labeled with prefix.

Writing a package in CoCoA-5 is slightly different from how it was done in CoCoA-4 it is easier!).

**See Also:** `define`(I-4.4 pg.72), `source`(I-19.26 pg.293)

### II-8.2 First Example of a Package

The following is an example of a package. It could be typed into a window as-is during a CoCoA session, but we will assume that it is stored in a file in the CoCoA directory under the name “`one.cpkg5`”.

```
package $contrib/toypackage

export ToyTest;

define IsNumberOne(n)
  if n = 1 then return true; else return false; endif;
enddefine;

define ToyTest(n)
  if IsNumberOne(n) then
    print "The number 1";
  else
    print "Not the number 1";
  endif;
enddefine;

endpackage; -- of toypackage
```

Below is output from a CoCoA session in which this package was used:

```
-- read in the package:
Source "one.cpkg";
/* */ ToyTest(4); -- was exported
```

```

Not the number 1
/* */ IsNumberOne(4); -- !!! ERROR !!! as expected: wasn't exported
ERROR: Cannot find a variable named "IsNumberOne" in scope
IsNumberOne(4);
~~~~~

/* */ $contrib/toypackage.IsNumberOne(4);
false

```

## II-8.3 Package Essentials

A package begins with

```
Package $PackageName
```

and ends with

```
EndPackage;
```

“`PackageName`” is a string that will be used to identify the package. The dollar sign is required. The “`PackageName`” must be a valid identifier: i.e. start with a letter and comprise only letters, digits, slash and underscore; the name should be meaningful (and usually long, to avoid any risk of a name clash). We recommend using a name of the form “`contrib/subject`”.

All packages in the CoCoA directory “`packages`” are automatically loaded when starting CoCoA.

## II-8.4 Global Aliases

A global alias for a package is formed by using the command “`alias`” ([I-1.7 pg.29](#)) during a CoCoA session. NOTE: global aliases cannot be used in function definitions. This is to force independence of context. Inside a function, one must use the complete package name.

**See Also:** `alias`([I-1.7 pg.29](#)), `aliases`([I-1.8 pg.30](#))

## II-8.5 Sharing Your Package

If you create a package that others might find useful, please contact the CoCoA team by email at “`cocoa at dima.unige.it`”.

Include comments in the package that:

- \* explain the use of the package
- \* give the syntax, description, examples for exported functions.

## II-8.6 Commands and Functions for Packages

The following are commands and functions for packages:

<code>alias</code>	define aliases for package names
<code>aliases</code>	list of global aliases
<code>packages</code>	list of loaded packages
<code>PkgName</code>	returns the name of a package

## II-8.7 Supported Packages

Several packages are supported by the CoCoA team. These packages contain functions that are not built into CoCoA because they are of a more specialized or experimental nature.

Some functions which used to be defined in supported packages are now official functions in CoCoA-5.

## II-8.8 Galois Package

\*\*\*\*\* NOT YET UPDATED TO CoCoA-5: follow with care \*\*\*\*\*

```
TITLE      : galois.cpkg
DESCRIPTION : CoCoA package for computing in a cyclic algebraic
              extension
AUTHOR     : A. Bigatti, D.La Macchia, F.Rossi
```

```
-- Enter
      $contrib/galois.Man();
      to get a complete description of the package including a suggested alias.
```

## II-8.9 Integer Programming

\*\*\*\*\* NOT YET UPDATED TO CoCoA-5: follow with care \*\*\*\*\*

```
TITLE      : intprog.cpkg
DESCRIPTION : CoCoA package for applying toric ideals to integer
              programming
AUTHOR     : A. Bigatti
```

```
-- Enter
      $contrib/intprog.Man();
      to get a complete description of the package including a suggested alias.
```

## II-8.10 Algebra of Invariants

\*\*\*\*\* NOT YET UPDATED TO CoCoA-5: follow with care \*\*\*\*\*

```
TITLE      : invariants.cpkg
DESCRIPTION : CoCoA package for computing homogeneous generators of an
              algebra of invariants, and for testing invariance of a polynomial
AUTHOR     : A. Del Padrone
```

```
-- Enter
      $contrib/invariants.Man();
      to get a complete description of the package including a suggested alias.
```

## II-8.11 Special Varieties

\*\*\*\*\* NOT YET UPDATED TO CoCoA-5: follow with care \*\*\*\*\*

```

TITLE      : specvar.cpkg
DESCRIPTION : CoCoA package for computing the Hilbert-Poincare
              series of special varieties (Segre, Veronese, Rees).
AUTHORS    : A. Bigatti, L. Robbiano

-- Enter
    $contrib/specvar.Man();
    to get a complete description of the package including a suggested alias.

```

## II-8.12 Statistics

\*\*\*\*\* NOT YET UPDATED TO CoCoA-5: follow with care \*\*\*\*\*

```

TITLE      : stat.cpkg
DESCRIPTION : package for design of experiments in statistics
AUTHOR     : M. Caboara

-- Enter
    $contrib/stat.Man();
    to get a complete description of the package including a suggested alias.

```

## II-8.13 Geometrical Theorem-Proving

\*\*\*\*\* NOT YET UPDATED TO CoCoA-5: follow with care \*\*\*\*\*

```

TITLE      : thmproving.cpkg
DESCRIPTION : CoCoA package for geometrical theorem-proving in euclidean space
AUTHOR     : L. Bazzotti, G. Dalzotto

-- Enter
    $contrib/thmproving.Man();
    to get a complete description of the package including a suggested alias.

```

## II-8.14 Typevectors

\*\*\*\*\* NOT YET UPDATED TO CoCoA-5: follow with care \*\*\*\*\*

```

TITLE      : typevectors.cpkg
DESCRIPTION : CoCoA package for computing type-vectors associated to
              Hilbert functions of ideals of points
AUTHOR     : E.Carlini, M.Stewart

-- Enter
    $contrib/typevectors.Man();
    to get a complete description of the package including a suggested alias.

```

## II-8.15 Conductor

\*\*\*\*\* NOT YET UPDATED TO CoCoA-5: follow with care \*\*\*\*\*

```

TITLE      : conductor.cpkg

```

DESCRIPTION : CoCoA package for computing conductor sequence of points  
 AUTHOR : L.Bazzotti

```
-- Enter
    $contrib/conductor.Man();
to get a complete description of the package including a suggested alias.
```

## II-8.16 Matrix Normal Form

\*\*\*\*\* NOT YET UPDATED TO CoCoA-5: follow with care \*\*\*\*\*

TITLE : matrixnormalform.cpkg  
 DESCRIPTION : CoCoA package for computing normal forms of a matrix,  
               Smith Normal Form (PID)  
 AUTHOR : A.Bigatti, S.DeFrancisci

```
-- Enter
    $contrib/matrixnormalform.Man();
to get a complete description of the package including a suggested alias.
```

## II-8.17 CantStop

\*\*\*\*\* NOT YET UPDATED TO CoCoA-5: follow with care \*\*\*\*\*

TITLE : CantStop.cpkg  
 DESCRIPTION : CoCoA package for playing Can't Stop and studying strategies  
 AUTHOR : A.Bigatti

```
-- Enter
    $contrib/CantStop.Man();
to get a complete description of the package including a suggested alias.
```

## II-8.18 Control

\*\*\*\*\* NOT YET UPDATED TO CoCoA-5: follow with care \*\*\*\*\*

TITLE : control.cpkg  
 DESCRIPTION : CoCoA package for Geometric Control Theory  
 AUTHOR : M. Anderlucchi and M. Caboara

```
-- Enter
    $contrib/control.Man();
to get a complete description of the package including a suggested alias.
```



## Chapter II-9

# Linked libraries

### II-9.1 CoCoALib

CoCoALib “<http://cocoa.dima.unige.it/cocoalib>”.

CoCoALib is the mathematical core of CoCoA-5. It may be used directly as a C++ library.

### II-9.2 GMP

GMP - The GNU Multiple Precision Arithmetic Library “<https://gmplib.org>”

All arbitrary precision integer/rational/floating-point datatypes and operations are based on GMP.

### II-9.3 GSL

GSL - GNU Scientific Library “<http://www.gnu.org/software/gsl/>”

Some functions from GSL have been ported to CoCoA-5. There is no manual yet because it's work in progress.

### II-9.4 Frobbby

Frobbby - Computations With Monomial Ideals “<http://www.broune.com/frobbby>”

All functions starting with “Frb” are implemented in Frobbby.

### II-9.5 MathSAT

MathSAT - Satisfiability modulo theories (SMT) solver “<http://mathsat.fbk.eu/>”

All functions starting with “MSAT” are implemented in MathSAT.

### II-9.6 Normaliz

libNormaliz is a C++ library for computations with rational cones and affine monoids; full details may be found on the official Normaliz website “<https://www.normaliz.uni-osnabrueck.de>”

When CoCoA is compiled it is possible to incorporate also libNormaliz; if so, then many libNormaliz functions can be called from CoCoA-5. All CoCoA functions starting with “Nmz” are actually implemented in libNormaliz.



## Chapter II-10

# Migrating from CoCoA-4 and keeping up-to-date

### II-10.1 Changes in the CoCoA language

CoCoA-5 is largely, but not completely, backward-compatible with CoCoA-4. Some commands/functions have changed name; others have been removed or replaced. Here we give a little guidance to help update your CoCoA-4 programs to CoCoA-5/

The operator “Not” has been replaced by the function “not(...)”.

example

```
/*C4*/ If Not X IsIn L Then ... EndIf;  
/*C5*/ If not(X IsIn L) Then ... EndIf;
```

Several functions modify one of their arguments (e.g. “append” (I-1.12 pg.31), “sort” (I-19.22 pg.291)); CoCoA-5 wants these arguments to be identified with the new keyword “ref” (I-18.26 pg.265), and will issue a warning if you don’t do this (just to make sure you know that “L” will be modified).

example

```
/*C4*/ L := [1,2,3]; Append(L, 4);  
/*C5*/ L := [1,2,3]; append(ref L, 4);
```

Implicit multiplication has gone: either write “x\*y” instead of “xy” for every product, or use “CoCoA-4 mode” (I-3.18 pg.52).

example

```
/*C4*/ F := 3xyzt;  
/*C5*/ F := 3*x*y*z*t; OR F := ***3xyzt***;
```

Many CoCoA-4 functions would employ the “CurrentRing” implicitly (e.g. “NumIndets()”, “CoeffRing()”). They now require an explicit argument; you can pass “CurrentRing” as the argument, but inside a function you must make that system variable visible via the command “TopLevel” (I-20.10 pg.314).

example

```
/*C4*/ Define LastIndet() Return Last(Indets()); EndDefine;  
/*C5*/ Define LastIndet()  
    TopLevel CurrentRing;  
    Return last(indets(CurrentRing));  
EndDefine;
```

However, we encourage you to consider modifying your function so that it does not depend on “CurrentRing”; e.g. you can find out to which ring a value belongs by calling the function “RingOf” (I-18.46 pg.275).

example

```
/*C5*/ I := ideal(x,y^2); NumIndets(RingOf(I));
```

The function “LinKer” (I-12.11 pg.183) has been replaced by “LinKerBasis” (I-12.12 pg.184), and there is a new function called “LinKer” (I-12.11 pg.183) which produces a matrix.

More generally, see also the CoCoA-4 ”translation table” in the CoCoAManual directory or at

<http://cocoa.dima.unige.it/cocoalib/doc/CoCoATranslationTable.html>

**See Also:** CoCoA-4 mode(I-3.18 pg.52), TopLevel(I-20.10 pg.314), CurrentRing(I-3.56 pg.68), RingOf(I-18.46 pg.275)

## II-10.2 Recent changes in the CoCoA-5 language

There are a few changes in the language even from the first versions of CoCoA-5.

The operator “Not” has been replaced by the function “not(...)”.

example

```
/*5.0.9*/ If Not X IsIn L Then ... EndIf;
/*5.1.0*/ If not(X IsIn L) Then ... EndIf;
```

The anonymous function called “lambda” is now called “func” (I-6.26 pg.105).

example

```
/*5.0.9*/ square := Lambda(x) Return x^2; EndLambda;
/*5.1.0*/ square := Func(x) Return x^2; EndFunc;
```

**See Also:** not(I-14.31 pg.222), func(I-6.26 pg.105)

## II-10.3 Obsolete and obsolescent functions

As the language evolves some functions might become obsolete, maybe just more sensibly renamed. This is the list of such functions: see in the manual for reasons/updates.

Call [OBSOLETE]	[OBSOLETE] apply a function to given arguments
Cast [OBSOLETE]	[OBSOLETE] type conversion
ColumnVectors [OBSOLETE]	[OBSOLETE] list of module elements
Comp [OBSOLETE]	[OBSOLETE] access a component
E_ [OBSOLETE]	[OBSOLETE] vector of the canonical basis
Function [OBSOLETE]	[OBSOLETE]
functions [OBSOLETE]	[OBSOLETE] replaced by describe
ID [OBSOLETE]	[OBSOLETE] renamed RingID
IsInSubalgebra [OBSOLETE]	[OBSOLETE] check if one polynomial is in a subalgebra
IsNumber [OBSOLETE]	[OBSOLETE] checks if the argument is a number
LinKerModP [OBSOLETE]	[OBSOLETE] find the kernel of a matrix mod p
LinSol [OBSOLETE]	[OBSOLETE] find a solution to a linear system
MapDown [OBSOLETE]	[OBSOLETE] convert a constant polynomial to a number
Mod2Rat [OBSOLETE]	[OBSOLETE] reconstruct rationals from modular integers
NewId [OBSOLETE]	[OBSOLETE] create a new identifier
NFsAreZero [OBSOLETE]	[OBSOLETE] test if normal forms are zero
Option [OBSOLETE]	[OBSOLETE] status of a panel option
panel [OBSOLETE]	[OBSOLETE] print status of a panel's options
panels [OBSOLETE]	[OBSOLETE] list of CoCoA panels

PoincareMultiDeg [OBSOLETE]	[OBSOLETE]
PoincareShifts [OBSOLETE]	[OBSOLETE]
randomize [OBSOLETE]	[OBSOLETE] randomize the coefficients of a given polynomial
randomized [OBSOLETE]	[OBSOLETE] randomize the coefficients of a given polynomial
Reset [OBSOLETE]	[OBSOLETE] reset panels and random number seed to defaults
ResetPanels [OBSOLETE]	[OBSOLETE] reset panels to their default values
RingEnv [OBSOLETE]	[OBSOLETE] name of the ring environment
RingSet [OBSOLETE]	[OBSOLETE] renamed RingsOf
seed [OBSOLETE]	[OBSOLETE] replaced by reseed
size [OBSOLETE]	[OBSOLETE]
SubalgebraMap [OBSOLETE]	[OBSOLETE] algebra homomorphism representing a subalgebra
Unset [OBSOLETE]	[OBSOLETE] set and unset panel options
valuation [OBSOLETE]	[OBSOLETE]
WLog [OBSOLETE]	[OBSOLETE] weighted list of exponents

Some functions are obsolescent, that means that they are still usable but will be deleted in some future version of CoCoA (leaving some time to adapt to the replacing function).

AffHilbert [OBSOLESCE	[OBSOLESCE	renamed AffHilbertFn
AffPoincare [OBSOLESCE	[OBSOLESCE	Renamed AffHilbertSeries
AllReducedGroebnerBases [OBSOLESCE		all reduced Groebner bases of an ideal
CompleteToOrd [OBSOLESCE	[OBSOLESCE	renamed MakeTermOrd
hilbert [OBSOLESCE		the Hilbert-Poincare' function
HomogElimMat [OBSOLESCE	[OBSOLESCE	renamed to ElimHomogMat
image [OBSOLESCE	[OBSOLESCE	apply ring homomorphism
insert [OBSOLESCE	[OBSOLESCE	insert an object in a list
log [OBSOLESCE	[OBSOLESCE	renamed to exponents
MinGensGeneral [OBSOLESCE	[OBSOLESCE	renamed MinSubsetOfGens
minimalize [OBSOLESCE	[OBSOLESCE	
minimalized [OBSOLESCE	[OBSOLESCE	
NewLine [OBSOLESCE	[OBSOLESCE	string containing a newline
poincare [OBSOLESCE	[OBSOLESCE	the Hilbert-Poincare series
PreImage [OBSOLESCE	[OBSOLESCE	
PrimaryPoincare [OBSOLESCE	[OBSOLESCE	renamed PrimaryHilbertSeries
rank [OBSOLESCE	[OBSOLESCE	rank
ReadExpr [OBSOLESCE	[OBSOLESCE	renamed RingElem
RMap [OBSOLESCE	[OBSOLESCE	define ring homomorphism for function image
SubalgebraRepr [OBSOLESCE	[OBSOLESCE	representation of a polynomial as a subalgebra element
TmpNBM [OBSOLESCE	[OBSOLESCE	renamed ApproxPointsNBM
WeightsMatrix [OBSOLESCE	[OBSOLESCE	matrix of generalized weights for indeterminates



# Part III

## CoCoA datatypes



# Chapter III-1

## BOOL

### III-1.1 Introduction to BOOL

The two BOOL constants are “true” and “false”. (can also be written “TRUE”, “FALSE” and “True”, “False”) They are mainly used with the commands “if” ([I-9.10](#) pg.137) and “while” ([I-23.3](#) pg.332), etc., inside CoCoA programs.

The relational operators

= <> < <= > >=

return boolean constants (see “Relational Operators” ([II-3.3](#) pg.348)).

The boolean operators are “and” ([I-1.11](#) pg.31), “or” ([I-15.9](#) pg.231), “IsIn” ([I-9.53](#) pg.157). From version CoCoA-5.0.9 “not” ([I-14.31](#) pg.222) is a function (instead of an operator).

**See Also:** Relational Operators([II-3.3](#) pg.348), Commands and Functions for BOOL([III-1.2](#) pg.373), Commands and Functions returning BOOL([III-1.3](#) pg.373)

### III-1.2 Commands and Functions for BOOL

and	boolean ”and” operator
Bool01	Convert a boolean to an integer
in	list element selector in list constructor
IsPolyRing	test whether a ring is a polynomial ring
not	boolean ”not” operator
or	boolean ”or” operators

### III-1.3 Commands and Functions returning BOOL

and	boolean ”and” operator
AreGensMonomial	checks if given gens are monomial
AreGensSqFreeMonomial	checks if given gens are squarefree monomial
Comparison Operators	less than, greater than, ...
EqSet	checks if the set of elements in two lists are equal
Equality Test	test whether two values are equal or not
HasGBasis	checks if the argument has a pre-computed GBasis

IsAntiSymmetric	checks if a matrix is anti-symmetric
IsCommutative	test whether a ring is commutative
IsConstant	checks if a ringelem is in the coefficient ring
IsContained	checks if A is Contained in B
IsCoprime	checks if t1 is coprime with t2
IsDiagonal	checks if a matrix is diagonal
IsDivisible	checks if A is divisible by B
IsElem	checks if A is an element of B
IsEven, IsOdd	test whether an integer is even or odd
IsFactorClosed	test whether a list of PPs is factor closed
IsField	test whether a ring is a field
IsFiniteField	test whether a ring is a finite field
IsFractionField	test whether a ring is a fraction field
IsHomog	test whether given polynomials are homogeneous
IsIn	check if one object is contained in another
IsIndet	checks argument is an indeterminate
IsInImage	check if a RINGELEM is in image of RINGHOM
IsInjective	check if a RINGHOM is injective
IsInRadical	check if a polynomial (or ideal) is in a radical
IsInteger	check if a RINGELEM is integer
IsIntegralDomain	test whether a ring is integral
IsInvertible	check if a RINGELEM is invertible
IsIrred	check if a RINGELEM is irreducible
IsLexSegment	checks if an ideal is lex-segment
IsMaximal	maximality test
IsMinusOne	test whether an object is -1
IsOne	test whether an object is one
IsPolyRing	test whether a ring is a polynomial ring
IsPositiveGrading	check if a matrix defines a positive grading
IsPrimary	primary test
IsPrime	prime integer test
IsProbPrime	checks if an integer is a probable prime
IsPthPower	p-th power test
IsQQ	test whether a ring is the ring of rationals
IsQuotientRing	test whether a ring is a quotient ring
IsRadical	check if an IDEAL is radical
IsRational	check if a RINGELEM is rational
IsSqFree	check if an INT or RINGELEM is square-free
IsStable	checks if an ideal is stable
IsStdGraded	checks if the grading is standard
IsStronglyStable	checks if an ideal is strongly stable
IsSubset	checks if the elements of one list are a subset of another
IsSurjective	check if a RINGHOM is surjective
IsSymmetric	checks if a matrix is symmetric
IsTerm	checks if the argument is a term
IsTermOrdering	check if a matrix defines a term-ordering
IsTrueGCDDomain	test whether a ring is a true GCD domain
IsZero	test whether an object is zero
IsZeroCol, IsZeroRow	test whether a column(row) is zero
IsZeroDim	test whether an ideal is zero-dimensional
IsZeroDivisor	test whether a RINGELEM is a zero-divisor
IsZZ	test whether a ring is the ring of integers
not	boolean "not" operator
or	boolean "or" operators

# Chapter III-2

## INT

### III-2.1 Introduction to INT

There are two types of numbers recognized by CoCoA: integers (type “INT”), rationals (type “RAT”). (CoCoA-4 also had “ZMOD”, but CoCoA-5 can deal with more rings: see “NewRingFp” ([I-14.10](#) pg.213)). Numbers in CoCoA are handled with arbitrary precision. This means that the sizes of numbers are only limited by the amount of available memory. The basic numeric operations—addition (“+”), subtraction (“-”), multiplication (“\*”), division (“/”), exponentiation (“^”), and negation (“-”)—behave as one would expect. Be careful, two adjacent minus signs, “--”, start a comment in CoCoA.

example

```
/**/ N := 3;
/**/ -N;
-3
--N; <-- THIS IS A COMMENT (not C++ decrement)
```

**See Also:** Commands and Functions for INT([III-2.2](#) pg.375), Commands and Functions returning INT([III-2.3](#) pg.378)

### III-2.2 Commands and Functions for INT

abs	absolute value of a number
AffHilbertFn	the affine Hilbert function
ascii	convert between characters and ascii code
AsINT	convert into an INT
AsRAT	convert into a RAT
binomial	binomial coefficient
BinomialRepr, BinomialReprShift	binomial representation of integers
ChebyshevPoly	Orthogonal Polynomials: Chebyshev, Hermite, Laguerre
ContFracToRat	convert continued fraction to rational
CRT	Chinese Remainder Theorem
CRTPoly	Chinese Remainder Theorem on polynomial coefficients
cyclotomic	n-th cyclotomic polynomial
date	the date
DecimalStr	convert rational number to decimal string
den	denominator
DensePoly	the sum of all power-products of a given degree
div	quotient for integers
ElimMat	matrix for elimination ordering
EvalHilbertFn	evaluate the Hilbert function

exponents	the list of exponents of the leading term of a polynomial
Ext	presentation Ext modules as quotients of free modules
factorial	factorial function
FactorMultiplicity	multiplicity of a factor of an integer
fibonacci	n-th fibonacci number
first	the first N elements of a list
flatten	flatten a list
FloatApprox	approx. of rational number of the form $M * 2^E$
FloatStr	convert rational number to a decimal string
FloorLog2, FloorLog10, FloorLogBase	integer part of the logarithm
FloorSqrt	(truncated) square root of an integer
format	convert object to formatted string
GBasisTimeout	compute a Groebner basis with a timeout
gcd	greatest common divisor
GCDFreeBasis	determine (minimal) GCD free basis of a set of integers
GenericPoints	random projective points
Get	read characters from a device
GetCol	convert a column of a matrix into a list
GetRow	convert a row of a matrix into a list
HermitePoly	Orthogonal Polynomials: Chebyshev, Hermite, Laguerre
HilbertFn	the Hilbert function
HilbertMat	create a Hilbert matrix over QQ
IdentityMat	the identity matrix
incr, decr	increment/decrement a counter
indent	prints in a more readable way
indet	individual indeterminates
insert [OBSOLESCE]	[OBSOLESCE] insert an object in a list
InverseSystem	Inverse system of an ideal of derivations
IO.SprintTrunc	convert to a string and truncate
iroot	integer part of r-th root of an integer
IsEven, IsOdd	test whether an integer is even or odd
IsInteger	check if a RINGELEM is integer
IsMinusOne	test whether an object is -1
IsOne	test whether an object is one
IsPositiveGrading	check if a matrix defines a positive grading
IsPrime	prime integer test
IsProbPrime	checks if an integer is a probable prime
IsSqFree	check if an INT or RINGELEM is square-free
IsZero	test whether an object is zero
IsZeroCol, IsZeroRow	test whether a column(row) is zero
LaguerrePoly	Orthogonal Polynomials: Chebyshev, Hermite, Laguerre
last	the last N elements of a list
lcm	least common multiple
LexMat	matrices for std. term-orderings
MakeMatByRows, MakeMatByCols	convert a list into a matrix
MakeTerm	returns a monomial (power-product) with given exponents
MakeTermOrd	Make a term order matrix from a given matrix
MantissaAndExponent10	convert rational number to a float
MantissaAndExponent2	convert rational number to a binary float
max	a maximum element of a sequence or list
min	a minimum element of a sequence or list
minors	list of minor determinants of a matrix
mod	remainder for integers
NewFreeModule	create a new FreeModule
NewList	create a new list
NewMat	Zero matrix
NewMatFilled	matrix filled with value

NewPolyRing	create a new PolyRing
NewRingFp	create a new finite field
NewRingTwinFloat	create a new twin-float ring
NewZZmod	create a new finite ring (integers mod N)
NextPrime, PrevPrime	find the next largest prime number
NextProbPrime, PrevProbPrime	find the next largest probable prime number
NmzSetVerbosityLevel	Set the verbosity state for Normaliz
num	numerator
NumPartitions	number of partitions of an integer
operators, shortcuts	Special characters equivalent to commands
partitions	partitions of an integer
PowerMod	compute a modular power efficiently
PrevPrime, PrevProbPrime	find the next largest prime number
PrevPrime, PrevProbPrime	find the previous largest probable prime number
PrimitiveRoot	find a primitive root modulo a prime
product	the product of the elements of a list
radical	radical of an ideal
random	random integer
RandomSparseNonSing01Mat	random sparse non-singular (0,1) matrix
RandomSubset	random subset
RandomSubsetIndices	indices for random subset
RandomTuple	random tuple
RandomTupleIndices	indices for random tuples
RandomUnimodularMat	random unimodular matrix
RatReconstructByContFrac, RatReconstructByLattice	rational reconstruction from modular image
RatReconstructPoly	Rational reconstruction of polynomial coefficients
RatReconstructWithBounds	deterministic rational reconstruction from modular image
RefineGCDFreeBasis	refine an integer GCD free basis
remove	remove an object in a list
reseed	reseed the pseudo-random number generator
RevLexMat	matrices for std. term-orderings
RingElem	convert an expression into a RINGELEM
RingQQt	pre-defined polynomial rings
RootBound	bound on roots of a polynomial over QQ
ScientificStr	convert integer/rational to a floating-point string
SectionalMatrix	sectional matrix
SetCol	set a list as a row into a matrix
SetRow	set a list as a row into a matrix
SetStackSize	secret ;-)
SetVerbosityLevel	set the verbosity level
sign	the sign of a number
SmallestNonDivisor	find smallest prime which does not divide an integer
SmoothFactor	find small prime factors of an integer
SourceRegion	read commands from a region in a file
spaces	return a string of spaces
StarPrint, StarSprint	print polynomial with *'s for multiplications
StdDegLexMat	matrices for std. term-orderings
StdDegRevLexMat	matrices for std. term-orderings
submat	submatrix
subsets	returns all sublists of a list
sum	the sum of the elements of a list
SwapCols	swap two columns in a matrix
SwapRows	swap two rows in a matrix
SymbolRange	range of symbols for the indeterminates of a PolyRing
syz	syzygy modules
tuples	N-tuples
WithoutNth	removes the N-th component from a list

XelMat	matrices for std. term-orderings
ZeroMat	matrix filled with 0

### III-2.3 Commands and Functions returning INT

abs	absolute value of a number
AffHilbertFn	the affine Hilbert function
AsINT	convert into an INT
binomial	binomial coefficient
BinomialRepr, BinomialReprShift	binomial representation of integers
Bool01	Convert a boolean to an integer
ceil	round rational up to integer
characteristic	the characteristic of a ring
ContFrac	continued fraction quotients
count	count the objects in a list
deg	the standard degree of a polynomial or moduleelem
den	denominator
depth	Depth of a module
dim	the dimension of a ring or quotient object
div	quotient for integers
EvalHilbertFn	evaluate the Hilbert function
factorial	factorial function
FactorMultiplicity	multiplicity of a factor of an integer
fibonacci	n-th fibonacci number
floor	round rational down to integer
FloorLog2, FloorLog10, FloorLogBase	integer part of the logarithm
FloorSqrt	(truncated) square root of an integer
gcd	greatest common divisor
GFanContainsPositiveVector	...
GFanGetAmbientDimension	...
GFanGetCodimension	...
GFanGetDimension	...
GFanGetDimensionOfLinealitySpace	...
GradingDim	Number of components in weighted degree
HilbertFn	the Hilbert function
IndetIndex	index of an indeterminate
iroot	integer part of r-th root of an integer
lcm	least common multiple
len	the length of an object
LogCardinality	extension degree of a finite field
LPosn	the position of the leading power-product in a ModuleElem
MayerVietorisTreeN1	N-1st Betti multidegrees of monomial ideals using Mayer-Vietoris trees
MinPowerInIdeal	the minimum power of a polynomial is an ideal
mod	remainder for integers
multiplicity	the multiplicity (degree) of a ring or quotient object
NextPrime, PrevPrime	find the next largest prime number
NextProbPrime, PrevProbPrime	find the next largest probable prime number
NmzVerbosityLevel	Get the verbosity level for Normaliz
num	numerator
NumCols	number of columns in a matrix
NumCompts	the number of components
NumGens	number of generators

NumIndets	number of indeterminates
NumPartitions	number of partitions of an integer
NumRows	number of rows in a matrix
NumTerms	number of terms in a polynomial
PowerMod	compute a modular power efficiently
PrevPrime, PrevProbPrime	find the next largest prime number
PrevPrime, PrevProbPrime	find the previous largest probable prime number
PrimitiveRoot	find a primitive root modulo a prime
radical	radical of an ideal
random	random integer
reg	Castelnuovo-Mumford regularity of a module
RegularityIndex	regularity index of a Hilbert function or series
RingID	identification for ring
rk	rank of a matrix or module
round	round to integer
sign	the sign of a number
SmallestNonDivisor	find smallest prime which does not divide an integer
TimeOfDay	the current time
UnivariateIndetIndex	the index of the indeterminate of a univariate polynomial
VerbosityLevel	verbosity level



# Chapter III-3

## RAT

### III-3.1 Introduction to RAT

Rational numbers can be entered as fractions or as terminating decimals. CoCoA always converts a rational number into a fraction in lowest terms.

example

```
/**/ 3.8;  
19/5  
/**/ N := 4/8; N;  
1/2  
/**/ type(N);  
RAT
```

**See Also:** Commands and Functions for RAT([III-3.2 pg.381](#)), Commands and Functions returning RAT([III-3.3 pg.382](#))

### III-3.2 Commands and Functions for RAT

abs	absolute value of a number
AsINT	convert into an INT
AsRAT	convert into a RAT
ceil	round rational up to integer
CFApprox	continued fraction approximation
CFApproximants	continued fraction approximants
ContFrac	continued fraction quotients
DecimalStr	convert rational number to decimal string
den	denominator
FloatApprox	approx. of rational number of the form $M * 2^E$
FloatStr	convert rational number to a decimal string
floor	round rational down to integer
FloorLog2, FloorLog10, FloorLogBase	integer part of the logarithm
IsMinusOne	test whether an object is -1
IsOne	test whether an object is one
IsRational	check if a RINGELEM is rational
IsZero	test whether an object is zero
MantissaAndExponent10	convert rational number to a float
MantissaAndExponent2	convert rational number to a binary float
max	a maximum element of a sequence or list
min	a minimum element of a sequence or list

NewMatFilled	matrix filled with value
num	numerator
product	the product of the elements of a list
RealRootRefine	refine a real root of a univariate polynomial
RealRoots	computes the real roots of a univariate polynomial
RealRootsApprox	approximations to the real roots of a univariate poly
RingElem	convert an expression into a RINGELEM
round	round to integer
ScientificStr	convert integer/rational to a floating-point string
sign	the sign of a number
SimplestBinaryRatBetween	find simplest binary rational in a closed interval
SimplestRatBetween	find simplest rational in a closed interval
StableBBasis5	Stable Border Basis of ideal of points
sum	the sum of the elements of a list
TimeFrom	time elapsed since a given moment

### III-3.3 Commands and Functions returning RAT

abs	absolute value of a number
ApproxSolve	Approximate real solutions for polynomial system
AsRAT	convert into a RAT
CFApprox	continued fraction approximation
CFApproximants	continued fraction approximants
ContFracToRat	convert continued fraction to rational
CpuTime	Counts cpu time
FloatApprox	approx. of rational number of the form $M * 2^E$
RootBound	bound on roots of a polynomial over QQ
SimplestBinaryRatBetween	find simplest binary rational in a closed interval
SimplestRatBetween	find simplest rational in a closed interval

## Chapter III-4

# STRING

### III-4.1 String Literals

A string literal consists of a sequence of characters between double quotes (“...”).

example

```
/**/ PrintLn "The primes up to 10 are ", [n in 1..10 | IsPrime(n)];
The primes up to 10 are [2, 3, 5, 7]

/**/ Print "The quick brown fox", "jumped over the lazy dog.";
The quick brown foxjumped over the lazy dog
```

To put special characters in CoCoA string literals use the appropriate “*escape sequence*”. Here is a summary: “\” produces a double-quote character; “\n” produces a newline character; “\\” produces a backslash character; “\t” produces a TAB character; “\r” produces a carriage-return character.

example

```
/**/ Print "line 1\nline 2";
line 1
line 2
/**/ Print "A string containing \"quote marks\".";
A string containing "quote marks".
```

WARNING: CoCoA still accepts an “*obsolescent*” syntax for string literals (between single-quotes); do not use this!

**See Also:** String Operations(III-4.2 pg.383), sprint(I-19.29 pg.294), Commands and Functions for STRING(III-4.3 pg.384), Commands and Functions returning STRING(III-4.4 pg.384)

### III-4.2 String Operations

CoCoA offers only a few operations on strings: length, concatenation, comparison, substring containment and indexing.

example

```
/**/ str := "Hello" + "World!"; --> string concatenation
/**/ Print str;
HelloWorld!
/**/ len(str);           --> length in characters
11
/**/ "Abc" < str;        --> lexicographical comparison
true
```

```
/**/ str[1];          --> character indexing, indexes start from 1
H
```

The operator “IsIn” (I-9.53 pg.157) can be used to test if one string is a substring of another.

example

```
/**/ mesg := "Banana";
/**/ "ana" IsIn mesg;
true
/**/ "Ana" IsIn mesg; --> substring must be an exact match
false
```

**See Also:** String Literals(III-4.1 pg.383), ascii(I-1.18 pg.35), concat(I-3.38 pg.60), IsIn(I-9.53 pg.157), len(I-12.6 pg.181)

### III-4.3 Commands and Functions for STRING

ascii	convert between characters and ascii code
error	throw an error message
GetEnv	access shell variables
ImplicitHypersurface	implicitization of hypersurface
ImplicitPlotOn	outputs the zero locus of a bivariate polynomial to a file
indets	list of indeterminates in a PolyRing
IsIn	check if one object is contained in another
len	the length of an object
max	a maximum element of a sequence or list
min	a minimum element of a sequence or list
NewPolyRing	create a new PolyRing
NewWeylAlgebra	create a new Weyl Algebra
OpenIFile	open input file
OpenIString	open input string
OpenOFile	open output file
OpenSocket	open a socket connection
operators, shortcuts	Special characters equivalent to commands
PlotPointsOn	outputs the coordinates of the points to a file
protect	protect a variable from being overwritten
ReadExpr [OBSOLESCENT]	[OBSOLESCENT] renamed RingElem
RingElem	convert an expression into a RINGELEM
source	read commands from a file or device
SourceRegion	read commands from a region in a file
starting	list functions starting with a given string
SymbolRange	range of symbols for the indeterminates of a PolyRing
tagged	tag an object for pretty printing

### III-4.4 Commands and Functions returning STRING

ascii	convert between characters and ascii code
CocoaPackagePath	returns the path to the CoCoA packages
DecimalStr	convert rational number to decimal string
ExternalLibs	Linked external libraries

<code>FloatStr</code>	convert rational number to a decimal string
<code>format</code>	convert object to formatted string
<code>GetEnv</code>	access shell variables
<code>GetErrMesg</code>	returns the message associated with an error
<code>IndetName</code>	the name of an indeterminate
<code>IO.SprintTrunc</code>	convert to a string and truncate
<code>latex</code>	LaTeX formatting
<code>NewLine [OBSOLESCENT]</code>	[OBSOLESCENT] string containing a newline
<code>packages</code>	list of loaded packages
<code>PkgName</code>	returns the name of a package
<code>ScientificStr</code>	convert integer/rational to a floating-point string
<code>spaces</code>	return a string of spaces
<code>sprint</code>	convert to a string
<code>StarPrint, StarSprint</code>	print polynomial with *'s for multiplications
<code>tag</code>	returns the tag string of an object
<code>TimeFrom</code>	time elapsed since a given moment



# Chapter III-5

## LIST

### III-5.1 Introduction to LIST

A CoCoA list is a sequence of CoCoA objects between square brackets. See also “List Constructors” ([III-5.2 pg.388](#)). The objects may be of different types, though a well-designed algorithm will likely create lists of objects of a single type.

In particular, a list may contain other lists. The empty list is “[]”. We use square brackets to index into a list. If “L” is a non-empty list and “N” is an integer (between 1 and “`len(L)`”), then “L[N]” is the “N”-th component of “L”; indexes start from 1.

If “L” contains sublists, we can write “L[N<sub>1</sub>, N<sub>2</sub>, . . . , N<sub>s</sub>]” as shorthand for “L[N<sub>1</sub>][N<sub>2</sub>] . . . [N<sub>s</sub>]” (see the example below).

Lists are often used to build structured objects of type “MAT”, “MODULEELEM”, “IDEAL”, and “MODULE”.

example

```
/**/ use R ::= QQ[t,x,y,z];
/**/ L := [34*x+y^2, "a string", [], [True, False]]; -- a list
/**/ L[1]; -- the 1st component
y^2 +34*x
/**/ L[2];
a string
/**/ L[3];
[ ]
/**/ L[4]; -- The 4th component is a list, itself;
[true, false]
/**/ L[4][1]; -- its 1st component;
true
/**/ L[4,1]; -- the same.
true

/**/ [1,"a"]+[2,"b"]; -- NOTE: one may add lists if their components
[3, "ab"] -- are compatible (see "Algebraic Operators").

/**/ L := [x^2-y, t*y^2-z^3];
/**/ I := ideal(L);
/**/ I;
ideal(x^2 -y, t*y^2 -z^3)
```

**See Also:** [len\(I-12.6 pg.181\)](#), [List Constructors\(III-5.2 pg.388\)](#), [Commands and Functions for LIST\(III-5.3 pg.388\)](#), [Commands and Functions returning LIST\(III-5.4 pg.391\)](#)

## III-5.2 List Constructors

These operators create new lists.

```
A..B
[A,B,C,...]
[X in L: LIST | B: BOOL]: LIST
[E:expression | X in L]: LIST
[E:expression | X in L: LIST and B: BOOL]: LIST
```

“A..B” creates the list of integers from “A” to “B”, both ends are included.

“[A,B,C,...]” makes a list containing “A”, “B”, “C” and so on, in that order.

“[X in L | B]” makes a list of those elements in “L” for which condition “B” is true.

“[E | X in L]” evaluates the expression “E” for each “X” in “L”, and collects the results in a new list.

“[E | X in L and B]” evaluates the expression “E” for each “X” in “L” which satisfies the condition “B”, and collects the results in a new list.

example

```
/**/ []; --> empty list
[]
/**/ 1..4;
[1, 2, 3, 4]
/**/ [3,1,4,2];
[3, 1, 4, 2]
/**/ [N in 1..10 | IsPrime(N)];
[2, 3, 5, 7]
/**/ [N^2 | N in 1..4];
[1, 4, 9, 16]
/**/ [N^2 | N in 1..10 and IsPrime(N)];
[4, 9, 25, 49]
```

**See Also:** NewList(I-14.5 pg.210), append(I-1.12 pg.31), concat(I-3.38 pg.60), Range Operator(II-3.5 pg.348), CartesianProduct, CartesianProductList(I-3.5 pg.47)

## III-5.3 Commands and Functions for LIST

append	append an object to a list
apply	apply homomorphism
ApproxSolve	Approximate real solutions for polynomial system
ascii	convert between characters and ascii code
BettiMatrix	the matrix of the graded Betti numbers
CartesianProduct, CartesianProductList	Cartesian product of lists
CheckArgTypes	Check types in a list
coefficients	list of coefficients of a polynomial
CoefficientsWRT	list of coeffs and PPs of a poly wrt indet or list of indets
ColMat	single column matrix
concat	concatenate lists
ConcatHorList	create a simple block matrix
ConcatLists	concatenate a list of lists
ConcatVerList	create a simple block matrix
ContentWRT	content of a polynomial wrt and indet or a list of indets
ContFracToRat	convert continued fraction to rational
count	count the objects in a list
DiagMat	matrix with given diagonal

<code>diff</code>	returns the difference between two lists
<code>distrib</code>	the distribution of objects in a list
<code>DivAlg</code>	division algorithm
<code>elim</code>	eliminate variables
<code>ElimHomogMat</code>	matrix for elimination ordering
<code>ElimMat</code>	matrix for elimination ordering
<code>EqSet</code>	checks if the set of elements in two lists are equal
<code>eval</code>	substitute numbers or polynomials for indeterminates
<code>EvalQuasiPoly</code>	Evaluate a quasi-polynomial at an integer
<code>Ext</code>	presentation Ext modules as quotients of free modules
<code>FGLM5</code>	perform a FGLM Groebner Basis conversion
<code>first</code>	the first N elements of a list
<code>flatten</code>	flatten a list
<code>foreach</code>	loop command
<code>FrobeniusMat</code>	compute a matrix of the Frobenius Map
<code>FVector</code>	compute the f-vector of a top simplices list
<code>GBM</code>	intersection of ideals for zero-dimensional schemes
<code>gcd</code>	greatest common divisor
<code>GCDFreeBasis</code>	determine (minimal) GCD free basis of a set of integers
<code>HGBM</code>	intersection of ideals for zero-dimensional schemes
<code>HilbertSeriesShifts</code>	the Hilbert-Poincare series
<code>homog</code>	homogenize with respect to an indeterminate
<code>ideal</code>	ideal generated by list
<code>IdealAndSeparatorsOfPoints</code>	ideal and separators for affine points
<code>IdealAndSeparatorsOfProjectivePoints</code>	ideal and separators for points
<code>implicit</code>	implicitization
<code>ImplicitHypersurface</code>	implicitization of hypersurface
<code>ImplicitPlot</code>	outputs the zero locus of a bivariate polynomial to a file
<code>ImplicitPlotOn</code>	outputs the zero locus of a bivariate polynomial to a file
<code>in</code>	list element selector in list constructor
<code>InitialIdeal</code>	Initial ideal
<code>insert [OBSOLESCENT]</code>	[OBSOLESCENT] insert an object in a list
<code>Interpolate</code>	interpolating polynomial
<code>interreduce, interreduced</code>	interreduce a list of polynomials
<code>intersection</code>	intersect lists, ideals, or modules
<code>IntersectionList</code>	intersect lists, ideals, or modules
<code>IsFactorClosed</code>	test whether a list of PPs is factor closed
<code>IsHomog</code>	test whether given polynomials are homogeneous
<code>IsIn</code>	check if one object is contained in another
<code>IsSubset</code>	checks if the elements of one list are a subset of another
<code>IsTree5</code>	checks if a facet complex is a tree
<code>jacobian</code>	the Jacobian of a list of polynomials
<code>last</code>	the last N elements of a list
<code>lcm</code>	least common multiple
<code>len</code>	the length of an object
<code>LexSegmentIdeal</code>	lex-segment ideal containing L, or with the same HilbertFn as I
<code>LinKerBasis</code>	find the kernel of a matrix
<code>MakeMatByRows, MakeMatByCols</code>	convert a list into a matrix
<code>MakeSet</code>	remove duplicates from a list
<code>MakeTerm</code>	returns a monomial (power-product) with given exponents
<code>matrix</code>	convert a list into a matrix
<code>max</code>	a maximum element of a sequence or list
<code>MaxBy</code>	a maximum element of a list
<code>min</code>	a minimum element of a sequence or list
<code>MinBy</code>	a minimum element of a list
<code>ModuleElem</code>	create a module element
<code>MultiplicationMat</code>	the multiplication matrix of a ringelem

NewPolyRing	create a new PolyRing
NmzComputation	flexible access to Normaliz
NmzEhrhartRing	Computes the Ehrhart ring
NmzIntClosureMonIdeal	integral closure of a monomial ideal
NmzIntClosureToricRing	integral closure of a toric ring
NmzNormalToricRing	normalization of a toric ring
NonZero	remove zeroes from a list
NR	normal reduction
operators, shortcuts	Special characters equivalent to commands
permutations	returns all permutations of the entries of a list
PlotPoints	outputs the coordinates of the points to a file
PlotPointsOn	outputs the coordinates of the points to a file
PolyAlgebraHom	homomorphism of polynomial algebras
PolyRingHom	homomorphism of polynomial rings
PrintBettiDiagram	the diagram of the graded Betti numbers
product	the product of the elements of a list
QZP	change field for polynomials and ideals
RandomSubset	random subset
RandomTuple	random tuple
RationalAffinePoints	Affine rational solutions
RationalProjectivePoints	Projective rational solutions
RationalSolve	Rational solutions for polynomial system
RatReconstructWithBounds	deterministic rational reconstruction from modular image
RefineGCDFreeBasis	refine an integer GCD free basis
remove	remove an object in a list
reverse, reversed	reverse a list
RingsOf	list of the rings of an object
RMap [OBSOLESCE]	[OBSOLESCE] define ring homomorphism for function image
RowMat	single row matrix
ScalarProduct	scalar product
SeparatorsOfPoints	separators for affine points
SeparatorsOfProjectivePoints	separators for projective points
SetCol	set a list as a row into a matrix
SetRow	set a list as a row into a matrix
shape	extended list of types involved in an expression
SimplexInfo	Stanley-Reisner ideal, AlexanderDual complex, ideal of top simplices
SimplicialHomology	compute the simplicial homology of a top simplices list
sort	sort a list
SortBy	sort a list
sorted	sort a list
SortedBy	sort a list
StableBBasis5	Stable Border Basis of ideal of points
StableIdeal	stable ideal containing L
StagedTrees	staged trees from Statistics
StronglyStableIdeal	strongly stable ideal containing L
submat	submatrix
submodule	submodule generated by list
subsets	returns all sublists of a list
sum	the sum of the elements of a list
SymbolRange	range of symbols for the indeterminates of a PolyRing
syz	syzygy modules
tail	remove the first element of a list
toric	saturate toric ideals
tuples	N-tuples
WithoutNth	removes the N-th component from a list
ZPQ	change field for polynomials and ideals

## III-5.4 Commands and Functions returning LIST

AllReducedGroebnerBases [OBSOLESCE]	all reduced Groebner bases of an ideal
apply	apply homomorphism
ApproxSolve	Approximate real solutions for polynomial system
ascii	convert between characters and ascii code
BBasis5	Border Basis of zero dimensional ideal
BettiNumbers	(Multi-)graded Betti numbers
BinomialRepr, BinomialReprShift	binomial representation of integers
CartesianProduct, CartesianProductList	Cartesian product of lists
CFApproximants	continued fraction approximants
coefficients	list of coefficients of a polynomial
CoefficientsWRT	list of coeffs and PPs of a poly wrt indet or list of indets
CoeffListWRT	list of coefficients of a polynomial wrt and indet
compts	list of components of a ModuleElem
concat	concatenate lists
ConcatLists	concatenate a list of lists
ContFrac	continued fraction quotients
CurrentTypes	lists all data types
diff	returns the difference between two lists
distrib	the distribution of objects in a list
eigenfactors	eigenfactors of a matrix
eigenvectors	eigenvalues and eigenvectors of a matrix
EquiIsoDec	equidimensional isoradical decomposition
exponents	the list of exponents of the leading term of a polynomial
Externallibs	Linked external libraries
FGLM5	perform a FGLM Groebner Basis conversion
fields	list the fields of a record
flatten	flatten a list
FrbAlexanderDual	Alexander Dual of monomial ideals
FrbAssociatedPrimes	Associated primes of monomial ideals
FrbIrreducibleDecomposition	Irreducible decomposition of monomial ideals
FrbMaximalStandardMonomials	Maximal standard monomials of monomial ideals
FrbPrimaryDecomposition	Primary decomposition of monomial ideals
GBasis	calculate a Groebner basis
GBasisTimeout	compute a Groebner basis with a timeout
GCDFreeBasis	determine (minimal) GCD free basis of a set of integers
GenericPoints	random projective points
GenRepr	representation in terms of generators
gens	list of generators of an ideal
Get	read characters from a device
GetCol	convert a column of a matrix into a list
GetCols	convert a matrix into a list of lists
GetRow	convert a row of a matrix into a list
GetRows	convert a matrix into a list of lists
GraverBasis	Graver basis
GroebnerFanIdeals	all reduced Groebner bases of an ideal
GroebnerFanReducedGBases	Groebner fan reduced GBases
HilbertBasisKer	Hilbert basis for a monoid
homog	homogenize with respect to an indeterminate
HVector	the h-vector of a module or quotient object
in	list element selector in list constructor

<code>indets</code>	list of indeterminates in a PolyRing
<code>IndetSubscripts</code>	the index of an indeterminate
<code>interreduce, interreduced</code>	interreduce a list of polynomials
<code>intersection</code>	intersect lists, ideals, or modules
<code>IntersectionList</code>	intersect lists, ideals, or modules
<code>InverseSystem</code>	Inverse system of an ideal of derivations
<code>JanetBasis</code>	the Janet basis of an ideal
<code>LinKerBasis</code>	find the kernel of a matrix
<code>MakeSet</code>	remove duplicates from a list
<code>MinGens</code>	list of minimal generators
<code>minors</code>	list of minor determinants of a matrix
<code>MinSubsetOfGens</code>	list of minimal generators
<code>monomials</code>	the list of monomials of a polynomial
<code>NewList</code>	create a new list
<code>NmzDiagInvariants</code>	ring of invariants of a diagonalizable group action
<code>NmzEhrhartRing</code>	Computes the Ehrhart ring
<code>NmzFiniteDiagInvariants</code>	ring of invariants of a finite group action
<code>NmzIntClosureMonIdeal</code>	integral closure of a monomial ideal
<code>NmzIntClosureToricRing</code>	integral closure of a toric ring
<code>NmzIntersectionValRings</code>	intersection of ring of valuations
<code>NmzNormalToricRing</code>	normalization of a toric ring
<code>NmzTorusInvariants</code>	ring of invariants of torus action
<code>NonZero</code>	remove zeroes from a list
<code>packages</code>	list of loaded packages
<code>partitions</code>	partitions of an integer
<code>permutations</code>	returns all permutations of the entries of a list
<code>PrimaryDecomposition</code>	primary decomposition of an ideal
<code>PrimaryDecomposition0</code>	primary decomposition of a 0-dimensional ideal
<code>PrimaryDecompositionGTZO</code>	primary decomposition of a 0-dimensional ideal
<code>QuotientBasis</code>	vector space basis for zero-dimensional quotient rings
<code>QuotientBasisSorted</code>	vector space basis for zero-dimensional quotient rings
<code>QZP</code>	change field for polynomials and ideals
<code>RandomSubset</code>	random subset
<code>RandomSubsetIndices</code>	indices for random subset
<code>RandomTuple</code>	random tuple
<code>RandomTupleIndices</code>	indices for random tuples
<code>RationalAffinePoints</code>	Affine rational solutions
<code>RationalProjectivePoints</code>	Projective rational solutions
<code>RationalSolve</code>	Rational solutions for polynomial system
<code>RealRoots</code>	computes the real roots of a univariate polynomial
<code>RealRootsApprox</code>	approximations to the real roots of a univariate poly
<code>ReducedGBasis</code>	compute reduced Groebner basis
<code>RefineGCDFreeBasis</code>	refine an integer GCD free basis
<code>res</code>	free resolution
<code>reverse, reversed</code>	reverse a list
<code>RingsOf</code>	list of the rings of an object
<code>SeparatorsOfPoints</code>	separators for affine points
<code>SeparatorsOfProjectivePoints</code>	separators for projective points
<code>shape</code>	extended list of types involved in an expression
<code>sorted</code>	sort a list
<code>SortedBy</code>	sort a list
<code>starting</code>	list functions starting with a given string
<code>StdBasis</code>	Standard basis
<code>subsets</code>	returns all sublists of a list
<code>support</code>	the list of terms of a polynomial or module elem
<code>SymbolRange</code>	range of symbols for the indeterminates of a PolyRing
<code>SymmetricPolys</code>	list of symmetric polynomials

<code>tail</code>	remove the first element of a list
<code>TopLevelFunctions</code>	returns the functions available at top-level
<code>tuples</code>	N-tuples
<code>UniversalGBasis</code>	universal Groebner basis of the input ideal
<code>wdeg</code>	multi-degree of an polynomial
<code>WithoutNth</code>	removes the N-th component from a list
<code>ZPQ</code>	change field for polynomials and ideals



## Chapter III-6

# RECORD

### III-6.1 Introduction to RECORD

A record is a data type in CoCoA representing a list of bindings of the form “*name to object*”.

example

```
/**/ use R ::= QQ[x,y,z];
/**/ P := record[ I := ideal(x,y^2-z), F := x^2 + y, Misc := [1,3,4]];
/**/ P.I;
ideal(x, y^2 -z)
/**/ P["I"];
ideal(x, y^2 -z)

/**/ P.Misc;
[1, 3, 4]
/**/ P.Misc[2];
3

/**/ P.Date := "1/1/98";
/**/ indent(P);
record[
  Date := "1/1/98",
  F := x^2 +y,
  I := ideal(x, y^2 -z),
  Misc := [1, 3, 4]
]

/**/ P["Misc",3]; -- equivalent to P.Misc[3]
4
```

Each entry in a record is called a “*field*”. Note that records are “*open*” in the sense that their fields can be extended, as shown in the previous example. At present, there is no function for deleting fields from a record, one must rewrite the record, selecting the fields to retain:

example

```
/**/ P := record[A := 2, B := 3, C := 5, D := 7];
/**/ Q := record[];

Foreach F In Fields(P) Do
  If F <> "C" Then Q[F] := P[F]; EndIf;
EndForeach;

/**/ P := Q;
```

```
/**/ P;
record[A := 2, B := 3, D := 7]
```

**See Also:** Commands and Functions for RECORD(III-6.2 pg.396), Commands and Functions returning RECORD(III-6.3 pg.396)

## III-6.2 Commands and Functions for RECORD

CoefficientsWRT	list of coeffs and PPs of a poly wrt indet or list of indets
fields	list the fields of a record
MSatLinSolve	
NmzComputation	flexible access to Normaliz
PrintBettiDiagram	the diagram of the graded Betti numbers
RealRootRefine	refine a real root of a univariate polynomial
record field selector	select a field of a record
shape	extended list of types involved in an expression

## III-6.3 Commands and Functions returning RECORD

AlmostQR	QR decomposition of a matrix
ApproxPointsNBM	Numerical Border Basis of ideal of points
CocoaLimits	limits on exponents and ring characteristics
ContentFreeFactor	factorization of multivariate polynomial into content-free factors
CRT	Chinese Remainder Theorem
CRTPoly	Chinese Remainder Theorem on polynomial coefficients
DivAlg	division algorithm
eigenvectors	eigenvalues and eigenvectors of a matrix
factor	factor a polynomial
FVector	compute the f-vector of a top simplices list
IdealAndSeparatorsOfPoints	ideal and separators for affine points
IdealAndSeparatorsOfProjectivePoints	ideal and separators for points
IndetSymbols	the names of the indeterminates in a PolyRing
LinearSimplify	simplifying linear substitution for a univariate poly over QQ
MantissaAndExponent10	convert rational number to a float
MantissaAndExponent2	convert rational number to a binary float
NmzComputation	flexible access to Normaliz
preimage0	preimage of a RINGELEM
PreprocessPts	Reduce redundancy in a set of approximate points
PrimaryDecompositionCore0	primary decomposition of a 0-dimensional ideal
RatReconstructByContFrac, RatReconstructByLattice	rational reconstruction from modular image
RatReconstructWithBounds	deterministic rational reconstruction from modular image
RealRootRefine	refine a real root of a univariate polynomial
record	create a record
shape	extended list of types involved in an expression
SimplexInfo	Stanley-Reisner ideal, AlexanderDual complex, ideal of top simplices
SimplicialHomology	compute the simplicial homology of a top simplices list
SmoothFactor	find small prime factors of an integer
SqFreeFactor	compute a squarefree factorization
StableBBasis5	Stable Border Basis of ideal of points
starting	list functions starting with a given string

**VersionInfo**

version and info about CoCoA



## Chapter III-7

# FUNCTION

### III-7.1 Introduction to FUNCTION

The most important construct in CoCoA programming is the user-defined function. These functions take parameters, perform CoCoA commands, and return values. Collections of functions can be stored in text files and read into CoCoA sessions using “source” (I-19.26 pg.293). To prevent name conflicts of the type that are likely to arise if functions are to be made available for use by others, the functions can be collected in “*packages*”.

To learn about user functions, look up “define” (I-4.4 pg.72) (online, enter “?define”).

### III-7.2 FUNCTIONs are first class objects

In CoCoA-5 functions are “first class objects”, and so may be passed like any other value.

#### example

```
-- The following function MyMax takes a function LessThan as parameter,
-- and returns the maximum of X and Y w.r.t. the ordering defined by the
-- function LessThan.

/**/ Define MyMax(LessThan, X, Y)
/**/   If LessThan(X, Y) Then Return Y; Else Return X; EndIf;
/**/ EndDefine;

-- Let's use MyMax by giving two different orderings.

/**/ Define CompareLT(X, Y) Return LT(X) < LT(Y); EndDefine;
/**/ Define CompareLC(X, Y) Return LC(X) < LC(Y); EndDefine;

/**/ use R ::= QQ[x,y,z];
/**/ MyMax(CompareLC, 3*x-y, 5*z-2);
5*z -2
/**/ MyMax(CompareLT, 3*x-y, 5*z-2);
3*x -y
```

### III-7.3 Commands and Functions for FUNCTION

CallOnGroebnerFanIdeals	apply a function to Groebner fan ideals
MaxBy	a maximum element of a list
MinBy	a minimum element of a list
ref	passing function parameters by reference

<code>return</code>	exit from a function
<code>SortBy</code>	sort a list
<code>SortedBy</code>	sort a list
<code>TopLevel</code>	make a top level variable accessible

### III-7.4 Commands and Functions returning **FUNCTION**

<code>define</code>	define a function
<code>func</code>	Anonymous function
<code>TopLevelFunctions</code>	returns the functions available at top-level

## Chapter III-8

# TYPE

### III-8.1 Commands and Functions for TYPE

`describe`    information about an object

### III-8.2 Commands and Functions returning TYPE

<code>CurrentTypes</code>	lists all data types
<code>shape</code>	extended list of types involved in an expression
<code>type</code>	the data type of an expression



# Chapter III-9

## RING

### III-9.1 Introduction to RING

Rings, and especially polynomial rings, play a central role in CoCoA.

The user can define many rings, but at any time a “*current ring*” is active within the system.

Once a ring has been defined, the system can handle the following mathematical objects defined over that ring:

- \* elements of the ring
- \* ideals
- \* matrices
- \* lists of objects
- \* modules (submodules of a free module)
- \* rings

Variables containing ring-dependent objects such as polynomials, ideals, and matrices are “*labeled*” by their ring. Any operation on them is performed in their ring, independently of what the current ring is.

**See Also:** Polynomial Rings(III-9.2 pg.403), Commands and Functions for RING(III-9.8 pg.406), Commands and Functions returning RING(III-9.9 pg.407)

### III-9.2 Polynomial Rings

CoCoA starts with the default (polynomial) ring “ $R = \mathbb{Q}[x,y,z]$ ”. Polynomial rings are created with the function “NewPolyRing” (I-14.8 pg.212), but there is a special simplified syntax working in most cases: it must be preceded by the command “use” (I-21.6 pg.327) or by the symbol “ $::=$ ” (or both)

```
R ::= C[X:INDETS];          use C[X:INDETS];
R ::= C[X:INDETS], 0;        use C[X:INDETS], 0;
```

“R” is the identifier of a CoCoALanguage variable, “C” is a RING, “X” is an expression that defines the indeterminates, “0” is a pre-defined ordering (“lex”, “deglex”, “degrevlex”). The default ordering is DegRevLex.

After the ring is defined using the above syntax, it can be made to be the current ring with the command “use” (I-21.6 pg.327).

example

```
/**/ use R ::= QQ[a,b,c]; -- define and use the ring R
/**/ K := NewFractionField(R);
/**/ S := K[x,y], Lex;
/**/ CurrentRing; -- the current ring is still R
RingWithID(21, "QQ[a,b,c]")
/**/ use S; -- now the ring S is the current ring
```

```
/**/ CurrentRing;
RingWithID(23, "RingWithID(22)[x,y]")
```

**See Also:** [NewPolyRing\(I-14.8 pg.212\)](#), [CurrentRing\(I-3.56 pg.68\)](#)

### III-9.3 Coefficient Rings

The coefficient ring for a CoCoA polynomial ring may be any ring “R”:

1. ZZ: (arbitrarily large) integer numbers;
2. QQ: (arbitrarily large) rational numbers;
3. ZZ/(N);
4. R[a,b,c];
5. K(a,b,c); ....

The first two types of coefficients are based on the GNU-gmp library. Some operations work only when coefficients are in a field (a meaningful error message will be thrown). NOTE: inside “define/enddefine” the top-level variables “ZZ” ([I-25.4 pg.338](#)) and “QQ” ([I-17.1 pg.251](#)) are not directly visible. Use “RingZZ()” or “RingQQ()” instead (or import them with “TopLevel” ([I-20.10 pg.314](#))).

example

```
/**/ R ::= QQ[x,y];    R;
/**/ S ::= ZZ/(5)[t];  S;
/**/ -- NOTE: "::=" for special syntax C[X], "==" for normal function call
/**/ QQi ::= QQ[i];
/**/ K := NewQuotientRing(QQi, ideal(RingElem(QQi, "i^2+1")));
/**/ U ::= K[u,v];    U;
```

**See Also:** [CoeffRing\(I-3.28 pg.56\)](#)

### III-9.4 Indeterminates

An “indeterminate” is represented by an identifier followed by one or more integer indices. For example, “x”, “alpha[1]”, “x[1,2,3]” are legal (and different) indeterminates, as is “x[i, 2\*i+1]” if “i” is of type “INT”.

When creating a ring the indeterminates are listed separate by commas.

example

```
/**/ use R ::= QQ[x,y,z];
/**/ use R ::= QQ[x[1..2,4..8],y[1..3],u,v];
/**/ Indets(R);
[x[1,4], x[1,5], x[1,6], x[1,7], x[1,8], x[2,4], x[2,5], x[2,6],
x[2,7], x[2,8], y[1], y[2], y[3], u, v]
-----
```

### III-9.5 Term Orderings

Polynomials are always sorted with respect to the ordering of their base ring; this ordering is specified when the ring is created. All operations involving polynomials utilize and preserve this ordering. There are mnemonic keywords for some predefined term-orderings; otherwise a custom ordering defined by an “ordering matrix” can be specified when using the function “NewPolyRing” ([I-14.8 pg.212](#)).

The predefined term-orderings are:

- \* standard-degree reverse lexicographic: “DegRevLex” (default)

- \* standard-degree lexicographic: “DegLex”
- \* pure lexicographic: “Lex” (no grading)
- \* pure xel: “Xe1” (NOT YET IMPLEMENTED)

If the indeterminates are given in the order “ $x_1, \dots, x_n$ ”, then “ $x_1 > \dots > x_n$ ” with respect to Lex, and “ $x_1 < \dots < x_n$ ” with respect to Xel.

example

```
-- Specify the ordering when you create the ring:
/**/ P := QQ[x,y,z];          --> default is DegRevLex
/**/ P := QQ[x,y,z], DegRevLex; --> same as above
/**/ P := QQ[x,y,z], lex;
/**/ P := QQ[x,y,z], DegLex;
```

**See Also:** NewPolyRing(I-14.8 pg.212), OrdMat(I-15.10 pg.231), elim(I-5.4 pg.84)

## III-9.6 Module Orderings

\*\*\*\*\* NOT YET UPDATED TO CoCoA-5 \*\*\*\*\*

First we recall the definition of a module term-ordering. We assume that all our free modules have finite rank and are of the type  $M = R^r$  where  $R$  is a polynomial ring with  $n$  indeterminates. Let  $[e_i | i = 1, \dots, r]$  be the canonical basis of  $M$ . A “*term*” of  $M$  is an element of the form  $Te_i$  where  $T$  belongs to the set  $T(R)$  of the terms of  $R$ . Hence the set  $T(M)$ , of the terms of  $M$ , is in one-to-one correspondence with the Cartesian product,  $T(R) \times [1, \dots, r]$ .

A “*module term-ordering*” is defined as a total ordering  $>$  on  $T(M)$  such that for all “ $T, T_1, T_2$ ” in  $T(R)$ , with  $T$  not equal to 1, and for all  $i, j$  in  $1, \dots, r$ ,

- (1)  $T * T_1 * e_i > T_1 * e_i$
- (2)  $T_1 * e_i > T_2 * e_j \Rightarrow T * T_1 * e_i > T * T_2 * e_j$

Each term-ordering on the current ring induces several term-orderings on a free module. CoCoA allows the user to choose between the following:

- \* the ordering called “ToPos” (which is the default one) defined by:

$$T_1 * e_i > T_2 * e_j \Leftrightarrow \begin{array}{l} T_1 > T_2 \text{ in } R \\ \text{or, if } T_1 = T_2, i < j \end{array}$$

- \* the ordering called “PosTo” defined by:

$$T_1 * e_i > T_2 * e_j \Leftrightarrow \begin{array}{l} i < j \\ \text{or, if } i = j, T_1 > T_2 \text{ in } R \end{array}$$

The leading term of the vector  $(x, y^2)$  with respect to two different module term-orderings:

example

```
use R := QQ[x,y], ToPos;
LT(Vector(x,y^2));
Vector(0, y^2)
-----
use R := QQ[x,y], PosTo;
LT(Vector(x,y^2));
Vector(x, 0)
-----
```

## III-9.7 Quotient Rings

If “R” is a ring and “I” is an ideal (in “R”) then “R/I” creates the corresponding quotient ring. There is a convenient shorthand for quotients of “ZZ”.

example

```

/**/ use ZZ/(11)[x];
/**/ (x+3)^11;
x^11 + 3
/**/ use R ::= QQ[x,y];
/**/ I := ideal(x^3+y^3, x^2*y-y^2*x);
/**/ Q := R/I;
/**/ HilbertFn(Q); -- the Hilbert function for Q
H(0) = 1
H(1) = 2
H(2) = 3
H(3) = 2
H(4) = 1
H(t) = 0 for t >= 5

```

**See Also:** NewQuotientRing(I-14.9 pg.212)

## III-9.8 Commands and Functions for RING

AffHilbertFn	the affine Hilbert function
ApproxPointsNBM	Numerical Border Basis of ideal of points
BaseRing	the base ring of a ring
BettiDiagram	the diagram of the graded Betti numbers
CanonicalHom	canonical homomorphism
characteristic	the characteristic of a ring
CoeffEmbeddingHom	returns the coefficient embedding homomorphism of a polynomial ring
CoeffRing	the ring of coefficients of a polynomial ring
ColMat	single column matrix
DefiningIdeal	defining ideal of a quotient ring
DensePoly	the sum of all power-products of a given degree
depth	Depth of a module
DiagMat	matrix with given diagonal
dim	the dimension of a ring or quotient object
EmbeddingHom	returns the embedding homomorphism of a fraction field
GenericPoints	random projective points
GradingMat	matrix of generalized weights for indeterminates
HilbertFn	the Hilbert function
HilbertPoly	the Hilbert polynomial
HilbertSeries	the Hilbert-Poincare series
HilbertSeriesMultiDeg	the Hilbert-Poincare series wrt a multigrading
HVector	the h-vector of a module or quotient object
ideal	ideal generated by list
IdealOfPoints	ideal of a set of affine points
IdealOfProjectivePoints	ideal of a set of projective points
IdentityMat	the identity matrix
implicit	implicitization
ImplicitHypersurface	implicitization of hypersurface
indet	individual indeterminates
indets	list of indeterminates in a PolyRing
IndetSymbols	the names of the indeterminates in a PolyRing
InducedHom	homomorphism induced by a homomorphism

IsCommutative	test whether a ring is commutative
IsField	test whether a ring is a field
IsFiniteField	test whether a ring is a finite field
IsFractionField	test whether a ring is a fraction field
IsIntegralDomain	test whether a ring is integral
IsPolyRing	test whether a ring is a polynomial ring
IsQQ	test whether a ring is the ring of rationals
IsQuotientRing	test whether a ring is a quotient ring
IsStdGraded	checks if the grading is standard
IsTrueGCDDomain	test whether a ring is a true GCD domain
IsZZ	test whether a ring is the ring of integers
LogCardinality	extension degree of a finite field
MakeTerm	returns a monomial (power-product) with given exponents
matrix	convert a list into a matrix
multiplicity	the multiplicity (degree) of a ring or quotient object
NewFractionField	create a new fraction field
NewFreeModule	create a new FreeModule
NewMat	Zero matrix
NewPolyRing	create a new PolyRing
NewQuotientRing	create a new quotient ring
NewWeylAlgebra	create a new Weyl Algebra
NumIndets	number of indeterminates
one	one of a ring
operators, shortcuts	Special characters equivalent to commands
OrdMat	matrix defining a term-ordering
poincare [OBSOLESCE]	[OBSOLESCE] the Hilbert-Poincare series
PolyAlgebraHom	homomorphism of polynomial algebras
PolyRingHom	homomorphism of polynomial rings
PrintBettiDiagram	the diagram of the graded Betti numbers
PrintSectionalMatrix	print sectional matrix
QQEmbeddingHom	returns the homomorphism $QQ \rightarrow R$
QuotientingHom	returns the projection homomorphism into a quotient ring
RandomSparseNonSing01Mat	random sparse non-singular (0,1) matrix
RandomUnimodularMat	random unimodular matrix
ReadExpr [OBSOLESCE]	[OBSOLESCE] renamed RingElem
reg	Castelnuovo-Mumford regularity of a module
RegularityIndex	regularity index of a Hilbert function or series
RingElem	convert an expression into a RINGELEM
RingID	identification for ring
RowMat	single row matrix
SectionalMatrix	sectional matrix
SymmetricPolys	list of symmetric polynomials
TmpChainCanonicalHom	canonical homomorphism
WeightsMatrix [OBSOLESCE]	[OBSOLESCE] matrix of generalized weights for indeterminates
zero	zero of a ring
ZeroMat	matrix filled with 0

## III-9.9 Commands and Functions returning RING

BaseRing	the base ring of a ring
codomain	codomain of a homomorphism
CoeffRing	the ring of coefficients of a polynomial ring

<code>domain</code>	domain of a homomorphism
<code>NewFractionField</code>	create a new fraction field
<code>NewPolyRing</code>	create a new PolyRing
<code>NewQuotientRing</code>	create a new quotient ring
<code>NewRingFp</code>	create a new finite field
<code>NewRingTwinFloat</code>	create a new twin-float ring
<code>NewWeylAlgebra</code>	create a new Weyl Algebra
<code>NewZZmod</code>	create a new finite ring (integers mod N)
<code>RingOf</code>	the ring of the object
<code>RingQQ</code>	the ring of rationals
<code>RingQQt</code>	pre-defined polynomial rings
<code>RingsOf</code>	list of the rings of an object
<code>RingZZ</code>	the ring of integers

## Chapter III-10

# RINGHOM

### III-10.1 Introduction to RINGHOM

A variable “X” containing an INT or a RAT can be immediately used within any RING. But an object “X” of other types, such as RINGELEM, IDEAL, MAT,... can be used only within its own RING, “RingOf(X)”. Such an object can be mapped into another RING using a “RINGHOM”: *“think mathematically”* ;-)

Most likely, the only function you need to use is just “CanonicalHom” (I-3.3 pg.46) which returns the canonical homomorphism between two rings (if there is one). Given a RINGHOM “phi” just type “phi(x)” if “x” is a RINGELEM, “apply(phi, x)” if “x” is a LIST or MAT.

However, there are also a few handy shortcuts silently determining and applying a homomorphism: the functions “matrix” (I-13.10 pg.195) and “RingElem” (I-18.43 pg.273) map the argument into the given ring (e.g. “matrix(R, M)” maps “M” into a new matrix in the ring “R”). Another shortcut is “BringIn” (I-2.13 pg.43) (easy, but slow).

NOTE: all CoCoA functions should be smart enough to take into account the RING in which their value was defined, for example “GBasis” (I-7.1 pg.107), “LT” (I-12.21 pg.188), “wdeg” (I-23.1 pg.331),...

NOTE: “QZP”, “ZPQ” are NOT YET IMPLEMENTED.

**See Also:** Commands and Functions for RINGHOM (III-10.3 pg.410), Commands and Functions returning RINGHOM (III-10.4 pg.410), apply (I-1.13 pg.32), CanonicalHom (I-3.3 pg.46), PolyAlgebraHom (I-16.14 pg.237), PolyRingHom (I-16.15 pg.238), matrix (I-13.10 pg.195), RingElem (I-18.43 pg.273), BringIn (I-2.13 pg.43)

### III-10.2 Composition of RINGHOM

Two RINGHOM “phi: R-->S” and “psi: S-->T” can be composed.

```
example
/**/ R := NewPolyRing(QQ, "a");
/**/ S := NewFractionField(R); -- QQ(a)
/**/ T := NewPolyRing(S, "x");
/**/ phi := CanonicalHom(R,S);
/**/ psi := CanonicalHom(S,T);
/**/ theta := psi(phi);
/**/ theta(RingElem(R, "a^2+a-1"));
a^2 + a - 1
/**/ RingOf(theta(RingElem(R, "a^2+a-1"))) = T;
true
```

**See Also:** CanonicalHom (I-3.3 pg.46), InducedHom (I-9.27 pg.146)

### III-10.3 Commands and Functions for RINGHOM

<code>apply</code>	apply homomorphism
<code>codomain</code>	codomain of a homomorphism
<code>domain</code>	domain of a homomorphism
<code>InducedHom</code>	homomorphism induced by a homomorphism
<code>IsInImage</code>	check if a RINGELEM is in image of RINGHOM
<code>IsInjective</code>	check if a RINGHOM is injective
<code>IsSurjective</code>	check if a RINGHOM is surjective
<code>ker</code>	Kernel of a homomorphism
<code>PolyRingHom</code>	homomorphism of polynomial rings
<code>preimage0</code>	preimage of a RINGELEM

### III-10.4 Commands and Functions returning RINGHOM

<code>CanonicalHom</code>	canonical homomorphism
<code>CoeffEmbeddingHom</code>	returns the coefficient embedding homomorphism of a polynomial ring
<code>EmbeddingHom</code>	returns the embedding homomorphism of a fraction field
<code>InducedHom</code>	homomorphism induced by a homomorphism
<code>PolyAlgebraHom</code>	homomorphism of polynomial algebras
<code>PolyRingHom</code>	homomorphism of polynomial rings
<code>QQEmbeddingHom</code>	returns the homomorphism $\mathbb{Q}\mathbb{Q} \rightarrow_i \mathbb{R}$
<code>QuotientingHom</code>	returns the projection homomorphism into a quotient ring
<code>TmpChainCanonicalHom</code>	canonical homomorphism

# Chapter III-11

## RINGELEM

### III-11.1 Introduction to RINGELEM

An object of type RINGELEM in CoCoA represents an element of a ring.

To fix terminology about polynomials (elements of a polynomial ring): a polynomial is a sum of terms; each term is the product of a coefficient and power-product, and a power-product is a product of powers of indeterminates.

In English it is standard to use “*monomial*” to mean a power-product, however, in other languages, such as Italian, monomial connotes a power-product multiplied by a scalar. In the interest of world peace, we will use the term power-product in those cases where confusion may arise.

example

```
/**/ use P ::= QQ[x,y,z];
/**/ f := 3*x*y*z + x*y^2;
/**/ f;
x*y^2 + 3*x*y*z
-----
/**/ use P ::= QQ[x[1..5]];
/**/ sum([x[N]^2 | N in 1..5]);
x[1]^2 + x[2]^2 + x[3]^2 + x[4]^2 + x[5]^2
-----
```

CoCoA always keeps polynomials ordered with respect to the term-orderings of their corresponding rings.

The following algebraic operations on polynomials are supported:

$F^N$ ,  $+F$ ,  $-F$ ,  $F \cdot G$ ,  $F/G$  if  $G$  divides  $F$ ,  $F+G$ ,  $F-G$ ,

where  $F$ ,  $G$  are polynomials and  $N$  is an integer. The result may be a rational function.

example

```
/**/ use R ::= QQ[x,y,z];
/**/ F := x^2 + x*y;
/**/ G := x;
/**/ F/G;
x + y

-- /**/ F/(x+z); --> !!! ERROR !!! as expected: Inexact division

/**/ F^2;
x^4 + 2*x^3*y + x^2*y^2
```

## III-11.2 Evaluation of Polynomials

The cleanest and most efficient way to evaluate polynomials is defining the appropriate “PolyAlgebraHom” (I-16.14 pg.237). However there are some handy shortcuts: “subst” (I-19.48 pg.303) and “eval” (I-5.12 pg.88).

example

```

/**/ use R ::= QQ[x,y,z];
/**/ f := x+y+z; --> let x=2 and y=1 in f
/**/ phi := PolyAlgebraHom(R, R, [2,1,z]); phi(f);
z +3
/**/ eval(f, [2,1]);
z +3
/**/ subst(f, [[x,2],[y,1]]);
z +3

```

**See Also:** PolyAlgebraHom(I-16.14 pg.237), eval(I-5.12 pg.88), subst(I-19.48 pg.303)

## III-11.3 Commands and Functions for RINGELEM

abs  
 apply  
 ApproxSolve  
 AsINT  
 AsRAT  
 binomial  
 CanonicalRepr  
 CharPoly  
 ChebyshevPoly  
 ClearDenom  
 CoeffHeight  
 coefficients  
 CoefficientsWRT  
 CoeffListWRT  
 CoeffOfTerm  
 CommonDenom  
 content  
 ContentFreeFactor  
 ContentWRT  
 CRTPoly  
 cyclotomic  
 DecimalStr  
 deg  
 den  
 deriv  
 DerivationAction  
 DF  
 discriminant  
 DivAlg  
 eigenfactors  
 eigenvectors  
 elim  
 eval  
 EvalQuasiPoly  
 exponents  
 factor  
 FactorAlgExt

absolute value of a number  
 apply homomorphism  
 Approximate real solutions for polynomial system  
 convert into an INT  
 convert into a RAT  
 binomial coefficient  
 representative of a class in a quotient ring  
 characteristic polynomial of a matrix  
 Orthogonal Polynomials: Chebyshev, Hermite,  
 clear common denominator of a polynomial with  
 the maximum of the absolute values of the coefficients  
 list of coefficients of a polynomial  
 list of coeffs and PPs of a poly wrt indet or list  
 list of coefficients of a polynomial wrt and index  
 coefficient of a term of a polynomial  
 Common denominator of a polynomial with rational  
 content of a polynomial  
 factorization of multivariate polynomial into content  
 content of a polynomial wrt and indet or a list  
 Chinese Remainder Theorem on polynomial coefficients  
 n-th cyclotomic polynomial  
 convert rational number to decimal string  
 the standard degree of a polynomial or module  
 denominator  
 the derivative of a polynomial or rational function  
 Action of a derivation  
 the degree form of a polynomial  
 the discriminant of a polynomial  
 division algorithm  
 eigenfactors of a matrix  
 eigenvalues and eigenvectors of a matrix  
 eliminate variables  
 substitute numbers or polynomials for indeterminates  
 Evaluate a quasi-polynomial at an integer  
 the list of exponents of the leading term of a polynomial  
 factor a polynomial  
 factorization algebraic extensions

FloatApprox	approx. of rational number of the form $M * 2^E$
FloatStr	convert rational number to a decimal string
FrbAlexanderDual	Alexander Dual of monomial ideals
gcd	greatest common divisor
GenRepr	representation in terms of generators
graeffe	graeffe transformation (squares the roots)
HermitePoly	Orthogonal Polynomials: Chebyshev, Hermite,
homog	homogenize with respect to an indeterminate
ideal	ideal generated by list
IndetIndex	index of an indeterminate
IndetName	the name of an indeterminate
IndetSubscripts	the index of an indeterminate
interreduce, interreduced	interreduce a list of polynomials
IsConstant	checks if a ringelem is in the coefficient ring
IsCoprime	checks if t1 is coprime with t2
IsDivisible	checks if A is divisible by B
IsElem	checks if A is an element of B
IsHomog	test whether given polynomials are homogeneous
IsIn	check if one object is contained in another
IsIndet	checks argument is an indeterminate
IsInImage	check if a RINGELEM is in image of RINGHO
IsInRadical	check if a polynomial (or ideal) is in a radical
IsInteger	check if a RINGELEM is integer
IsInvertible	check if a RINGELEM is invertible
IsIrred	check if a RINGELEM is irreducible
IsMinusOne	test whether an object is -1
IsOne	test whether an object is one
IsPthPower	p-th power test
IsRational	check if a RINGELEM is rational
IsSqFree	check if an INT or RINGELEM is square-free
IsTerm	checks if the argument is a term
IsZero	test whether an object is zero
IsZeroDivisor	test whether a RINGELEM is a zero-divisor
jacobian	the Jacobian of a list of polynomials
LaguerrePoly	Orthogonal Polynomials: Chebyshev, Hermite,
LC	the leading coefficient of a polynomial or Modu
lcm	least common multiple
LF	the leading form of a polynomial or an ideal
LinearSimplify	simplifying linear substitution for a univariate p
LM	the leading monomial of a polynomial or Modu
LPP	the leading power-product of a polynomial or M
LT	the leading term of an object
MantissaAndExponent2	convert rational number to a binary float
max	a maximum element of a sequence or list
min	a minimum element of a sequence or list
MinPoly	minimal polynomial of a matrix
MinPolyQuot, MinPolyQuotDef, MinPolyQuotElim, MinPolyQuotMat	compute a minimal polynomial
MinPowerInIdeal	the minimum power of a polynomial is an ideal
monic	divide polynomials by their leading coefficients
monomials	the list of monomials of a polynomial
MultiplicationMat	the multiplication matrix of a ringelem
NewMatFilled	matrix filled with value
NF	normal form
NmzEhrhartRing	Computes the Ehrhart ring
NmzIntClosureMonIdeal	integral closure of a monomial ideal
NmzIntClosureToricRing	integral closure of a toric ring
NmzNormalToricRing	normalization of a toric ring

NR  
 num  
 NumTerms  
 PerpIdealOfForm  
 preimage0  
 PrimaryDecompositionCore0  
 product  
 PthRoot  
 QZP  
 radical  
 RationalAffinePoints  
 RationalProjectivePoints  
 RationalSolve  
 RatReconstructPoly  
 RealRoots  
 RealRootsApprox  
 resultant  
 RingElem  
 RingOf  
 RingsOf  
 RootBound  
 ScientificStr  
 SqFreeFactor  
 StarPrint, StarSprint  
 subst  
 sum  
 support  
 sylvester  
 syz  
 UnivariateIndetIndex  
 wdeg  
 ZPQ

normal reduction  
 numerator  
 number of terms in a polynomial  
 Ideal of derivations annihilating a form  
 preimage of a RINGELEM  
 primary decomposition of a 0-dimensional ideal  
 the product of the elements of a list  
 Compute p-th root  
 change field for polynomials and ideals  
 radical of an ideal  
 Affine rational solutions  
 Projective rational solutions  
 Rational solutions for polynomial system  
 Rational reconstruction of polynomial coefficients  
 computes the real roots of a univariate polynomial  
 approximations to the real roots of a univariate polynomial  
 the resultant of two polynomials  
 convert an expression into a RINGELEM  
 the ring of the object  
 list of the rings of an object  
 bound on roots of a polynomial over QQ  
 convert integer/rational to a floating-point string  
 compute a squarefree factorization  
 print polynomial with \*'s for multiplications  
 substitute values for indeterminates  
 the sum of the elements of a list  
 the list of terms of a polynomial or module element  
 the Sylvester matrix of two polynomials  
 syzygy modules  
 the index of the indeterminate of a univariate polynomial  
 multi-degree of an polynomial  
 change field for polynomials and ideals

### III-11.4 Commands and Functions returning RINGELEM

abs  
 apply  
 binomial  
 CanonicalRepr  
 CharPoly  
 ChebyshevPoly  
 ClearDenom  
 CoeffHeight  
 CoeffListWRT  
 CoeffOfTerm  
 CommonDenom  
 ComputeElimFirst  
 content  
 ContentWRT  
 cyclotomic  
 den  
 DensePoly

absolute value of a number  
 apply homomorphism  
 binomial coefficient  
 representative of a class in a quotient ring  
 characteristic polynomial of a matrix  
 Orthogonal Polynomials: Chebyshev, Hermite,  
 clear common denominator of a polynomial with  
 the maximum of the absolute values of the coefficients  
 list of coefficients of a polynomial wrt and indet  
 coefficient of a term of a polynomial  
 Common denominator of a polynomial with rational coefficients  
 ComputeElimFirst  
 content of a polynomial  
 content of a polynomial wrt and indet or a list of indeterminates  
 n-th cyclotomic polynomial  
 denominator  
 the sum of all power-products of a given degree

<code>deriv</code>	the derivative of a polynomial or rational function
<code>det</code>	the determinant of a matrix
<code>DF</code>	the degree form of a polynomial
<code>discriminant</code>	the discriminant of a polynomial
<code>eigenfactors</code>	eigenfactors of a matrix
<code>EvalQuasiPoly</code>	Evaluate a quasi-polynomial at an integer
<code>FirstNonZero</code>	the first non-zero entry in a MODULEELEM
<code>FirstNonZeroPosn</code>	the first non-zero entry in a MODULEELEM
<code>gcd</code>	greatest common divisor
<code>graeffe</code>	graeffe transformation (squares the roots)
<code>GraverBasis</code>	Graver basis
<code>HermitePoly</code>	Orthogonal Polynomials: Chebyshev, Hermite,
<code>HilbertPoly</code>	the Hilbert polynomial
<code>homog</code>	homogenize with respect to an indeterminate
<code>ImplicitHypersurface</code>	implicitization of hypersurface
<code>indet</code>	individual indeterminates
<code>Interpolate</code>	interpolating polynomial
<code>interreduce, interreduced</code>	interreduce a list of polynomials
<code>InverseSystem</code>	Inverse system of an ideal of derivations
<code>JanetBasis</code>	the Janet basis of an ideal
<code>LaguerrePoly</code>	Orthogonal Polynomials: Chebyshev, Hermite,
<code>LC</code>	the leading coefficient of a polynomial or Module
<code>lcm</code>	least common multiple
<code>LF</code>	the leading form of a polynomial or an ideal
<code>LinKerBasis</code>	find the kernel of a matrix
<code>LM</code>	the leading monomial of a polynomial or Module
<code>LPP</code>	the leading power-product of a polynomial or Module
<code>LT</code>	the leading term of an object
<code>MakeTerm</code>	returns a monomial (power-product) with given exponents
<code>MinPoly</code>	minimal polynomial of a matrix
<code>MinPolyQuot, MinPolyQuotDef, MinPolyQuotElim, MinPolyQuotMat</code>	compute a minimal polynomial
<code>monic</code>	divide polynomials by their leading coefficients
<code>NF</code>	normal form
<code>NmzDiagInvariants</code>	ring of invariants of a diagonalizable group action
<code>NmzEhrhartRing</code>	Computes the Ehrhart ring
<code>NmzFiniteDiagInvariants</code>	ring of invariants of a finite group action
<code>NmzIntClosureMonIdeal</code>	integral closure of a monomial ideal
<code>NmzIntClosureToricRing</code>	integral closure of a toric ring
<code>NmzIntersectionValRings</code>	intersection of ring of valuations
<code>NmzNormalToricRing</code>	normalization of a toric ring
<code>NmzTorusInvariants</code>	ring of invariants of torus action
<code>NR</code>	normal reduction
<code>num</code>	numerator
<code>one</code>	one of a ring
<code>pfaffian</code>	the Pfaffian of a skew-symmetric matrix
<code>PthRoot</code>	Compute p-th root
<code>QZP</code>	change field for polynomials and ideals
<code>radical</code>	radical of an ideal
<code>RationalAffinePoints</code>	Affine rational solutions
<code>RationalProjectivePoints</code>	Projective rational solutions
<code>RationalSolve</code>	Rational solutions for polynomial system
<code>RatReconstructPoly</code>	Rational reconstruction of polynomial coefficients
<code>ReadExpr [OBSOLESCENT]</code>	[OBSOLESCENT] renamed RingElem
<code>ReducedGBasis</code>	compute reduced Groebner basis
<code>resultant</code>	the resultant of two polynomials
<code>RingElem</code>	convert an expression into a RINGELEM
<code>SymbolRange</code>	range of symbols for the indeterminates of a Polynomial

SymmetricPolys  
UniversalGBasis  
zero  
ZPQ

list of symmetric polynomials  
universal Groebner basis of the input ideal  
zero of a ring  
change field for polynomials and ideals

# Chapter III-12

## IDEAL

### III-12.1 Commands and Functions for IDEAL

AllReducedGroebnerBases [OBSOLESCENT]  
AreGensMonomial  
AreGensSqFreeMonomial  
BBasis5  
BettiDiagram  
BettiMatrix  
BettiNumbers  
CallOnGroebnerFanIdeals  
colon  
ComputeElimFirst  
depth  
elim  
EquiIsoDec  
FrbAlexanderDual  
FrbAssociatedPrimes  
FrbIrreducibleDecomposition  
FrbMaximalStandardMonomials  
FrbPrimaryDecomposition  
FrobeniusMat  
GBasis  
GBasisTimeout  
GenRepr  
gens  
gin  
GroebnerFanIdeals  
GroebnerFanReducedGBases  
HasGBasis  
HColon  
HilbertFn  
HilbertSeries  
homog  
HSaturation  
IdealOfGBasis  
IdealOfMinGens  
InitialIdeal  
intersection  
IntersectionList  
InverseSystem

all reduced Groebner bases of an ideal  
checks if given gens are monomial  
checks if given gens are squarefree monomial  
Border Basis of zero dimensional ideal  
the diagram of the graded Betti numbers  
the matrix of the graded Betti numbers  
(Multi-)graded Betti numbers  
apply a function to Groebner fan ideals  
ideal or module quotient  
ComputeElimFirst  
Depth of a module  
eliminate variables  
equidimensional isoradical decomposition  
Alexander Dual of monomial ideals  
Associated primes of monomial ideals  
Irreducible decomposition of monomial ideals  
Maximal standard monomials of monomial ideals  
Primary decomposition of monomial ideals  
compute a matrix of the Frobenius Map  
calculate a Groebner basis  
compute a Groebner basis with a timeout  
representation in terms of generators  
list of generators of an ideal  
generic initial ideal  
all reduced Groebner bases of an ideal  
Groebner fan reduced GBases  
checks if the argument has a pre-computed GB  
ideal or module quotient  
the Hilbert function  
the Hilbert-Poincare series  
homogenize with respect to an indeterminate  
saturation of ideals  
ideal generated by GBasis  
ideal generated by minimal generators  
Initial ideal  
intersect lists, ideals, or modules  
intersect lists, ideals, or modules  
Inverse system of an ideal of derivations

IsContained	checks if A is Contained in B
IsElem	checks if A is an element of B
IsHomog	test whether given polynomials are homogeneous
IsIn	check if one object is contained in another
IsInRadical	check if a polynomial (or ideal) is in a radical
IsLexSegment	checks if an ideal is lex-segment
IsMaximal	maximality test
IsOne	test whether an object is one
IsPrimary	primary test
IsRadical	check if an IDEAL is radical
IsStable	checks if an ideal is stable
IsStronglyStable	checks if an ideal is strongly stable
IsZero	test whether an object is zero
IsZeroDim	test whether an ideal is zero-dimensional
JanetBasis	the Janet basis of an ideal
LexSegmentIdeal	lex-segment ideal containing L, or with the same
LF	the leading form of a polynomial or an ideal
LT	the leading term of an object
MayerVietorisTreeN1	N-1st Betti multidegrees of monomial ideals using
MinGens	list of minimal generators
minimalize [OBSOLESCE]	[OBSOLESCE]
MinPolyQuot, MinPolyQuotDef, MinPolyQuotElim, MinPolyQuotMat	compute a minimal polynomial
MinPowerInIdeal	the minimum power of a polynomial is an ideal
MinSubsetOfGens	list of minimal generators
MonsInIdeal	ideal generated by the monomials in an ideal
MultiplicationMat	the multiplication matrix of a ringelem
NewQuotientRing	create a new quotient ring
NF	normal form
NumGens	number of generators
operators, shortcuts	Special characters equivalent to commands
poincare [OBSOLESCE]	[OBSOLESCE] the Hilbert-Poincare series
PrimaryDecomposition	primary decomposition of an ideal
PrimaryDecomposition0	primary decomposition of a 0-dimensional ideal
PrimaryDecompositionCore0	primary decomposition of a 0-dimensional ideal
PrimaryDecompositionGTZ0	primary decomposition of a 0-dimensional ideal
PrimaryHilbertSeries	primary
PrintBettiDiagram	the diagram of the graded Betti numbers
PrintBettiMatrix	print the matrix of the graded Betti numbers
PrintBettiNumbers	print the (multi-)graded Betti numbers
PrintSectionalMatrix	print sectional matrix
product	the product of the elements of a list
QuotientBasis	vector space basis for zero-dimensional quotient
QuotientBasisSorted	vector space basis for zero-dimensional quotient
QZP	change field for polynomials and ideals
radical	radical of an ideal
RadicalOfUnmixed	radical of an unmixed ideal
ReducedGBasis	compute reduced Groebner basis
reg	Castelnuovo-Mumford regularity of a module
res	free resolution
rgin	generic initial ideal wrt StdDegRevLex
RingOf	the ring of the object
RingsOf	list of the rings of an object
saturate	saturation of ideals
SectionalMatrix	sectional matrix
StdBasis	Standard basis
sum	the sum of the elements of a list
syz	syzygy modules

SyzOfGens	syzygy module for a given set of generators
TgCone	tangent cone
toric	saturate toric ideals
UniversalGBasis	universal Groebner basis of the input ideal
ZPQ	change field for polynomials and ideals

## III-12.2 Commands and Functions returning IDEAL

colon	ideal or module quotient
DefiningIdeal	defining ideal of a quotient ring
elim	eliminate variables
EquiIsoDec	equidimensional isoradical decomposition
GBM	intersection of ideals for zero-dimensional schemes
gin	generic initial ideal
HColon	ideal or module quotient
HGBM	intersection of ideals for zero-dimensional schemes
homog	homogenize with respect to an indeterminate
HSaturation	saturation of ideals
ideal	ideal generated by list
IdealOfGBasis	ideal generated by GBasis
IdealOfMinGens	ideal generated by minimal generators
IdealOfPoints	ideal of a set of affine points
IdealOfProjectivePoints	ideal of a set of projective points
implicit	implicitization
InitialIdeal	Initial ideal
intersection	intersect lists, ideals, or modules
IntersectionList	intersect lists, ideals, or modules
ker	Kernel of a homomorphism
LexSegmentIdeal	lex-segment ideal containing L, or with the same HilbertFn as I
LF	the leading form of a polynomial or an ideal
LT	the leading term of an object
MonsInIdeal	ideal generated by the monomials in an ideal
PerpIdealOfForm	Ideal of derivations annihilating a form
PrimaryDecomposition	primary decomposition of an ideal
PrimaryDecomposition0	primary decomposition of a 0-dimensional ideal
PrimaryDecompositionGTZO	primary decomposition of a 0-dimensional ideal
QZP	change field for polynomials and ideals
radical	radical of an ideal
RadicalOfUnmixed	radical of an unmixed ideal
rgin	generic initial ideal wrt StdDegRevLex
saturate	saturation of ideals
StableIdeal	stable ideal containing L
StagedTrees	staged trees from Statistics
StronglyStableIdeal	strongly stable ideal containing L
TgCone	tangent cone
toric	saturate toric ideals
ZPQ	change field for polynomials and ideals



# Chapter III-13

## MAT

### III-13.1 Introduction to MAT

An  $m \times n$  matrix is represented in CoCoA by the list of its rows (see “`matrix`” ([I-13.10](#) pg.195)). The  $(A,B)$ -th entry of a matrix  $M$  is given by “ $M[A][B]$ ” or “ $M[A,B]$ ”.

The following operations are defined as one would expect for matrices

$M^A$ ,  $+M$ ,  $-N$ ,  $M+N$ ,  $M-N$ ,  $M*N$ ,  $F*M$ ,  $M*F$

where  $M$ ,  $N$  are matrices,  $A$  is a non-negative integer, and  $F$  is a polynomial, with the obvious restrictions on the dimensions of the matrices involved.

example

```
/**/ use R ::= QQ[x,y];
/**/ N := matrix(R, [[1,2],[3,4]]);
/**/ N[1,2];
2;

/**/ N^2;
matrix( /*RingWithID(3, "QQ[x,y]")*/
  [[7, 10],
   [15, 22]])

/**/ x * N;
matrix( /*RingWithID(3, "QQ[x,y]")*/
  [[x, 2*x],
   [3*x, 4*x]])

/**/ N + matrix([[x,x], [y,y]]);
matrix( /*RingWithID(3, "QQ[x,y]")*/
  [[x +1, x +2],
   [y +3, y +4]])
```

### III-13.2 Commands and Functions for MAT

<code>adj</code>	classical adjoint matrix (also known as adjugate)
<code>AlmostQR</code>	QR decomposition of a matrix
<code>apply</code>	apply homomorphism
<code>ApproxPointsNBM</code>	Numerical Border Basis of ideal of points
<code>BlockMat</code>	create a block matrix

BlockMat2x2	create a block matrix with 4 matrices
CharPoly	characteristic polynomial of a matrix
ConcatAntiDiag	create a simple block matrix
ConcatDiag	create a simple block matrix
ConcatHor	create a simple block matrix
ConcatHorList	create a simple block matrix
ConcatVer	create a simple block matrix
ConcatVerList	create a simple block matrix
det	the determinant of a matrix
eigenfactors	eigenfactors of a matrix
eigenvectors	eigenvalues and eigenvectors of a matrix
ElimHomogMat	matrix for elimination ordering
ElimMat	matrix for elimination ordering
eval	substitute numbers or polynomials for indeterminates
FGLM5	perform a FGLM Groebner Basis conversion
GetCol	convert a column of a matrix into a list
GetCols	convert a matrix into a list of lists
GetRow	convert a row of a matrix into a list
GetRows	convert a matrix into a list of lists
GFanContainsPositiveVector	...
GFanGeneratorsOfLinealitySpace	...
GFanGeneratorsOfSpan	...
GFanGetAmbientDimension	...
GFanGetCodimension	...
GFanGetDimension	...
GFanGetDimensionOfLinealitySpace	...
GFanGetFacets	...
GFanGetImpliedEquations	...
GFanGetUniquePoint	...
GFanRelativeInteriorPoint	relative interior point of a cone
GraverBasis	Graver basis
HilbertBasisKer	Hilbert basis for a monoid
HilbertSeriesMultiDeg	the Hilbert-Poincare series wrt a multigrading
IdealOfPoints	ideal of a set of affine points
IdealOfProjectivePoints	ideal of a set of projective points
inverse	multiplicative inverse of matrix
IsAntiSymmetric	checks if a matrix is anti-symmetric
IsDiagonal	checks if a matrix is diagonal
IsPositiveGrading	check if a matrix defines a positive grading
IsSymmetric	checks if a matrix is symmetric
IsTermOrdering	check if a matrix defines a term-ordering
IsZero	test whether an object is zero
IsZeroCol, IsZeroRow	test whether a column(row) is zero
LinKer	find the kernel of a matrix
LinKerBasis	find the kernel of a matrix
LinSolve	find a solution to a linear system
MakeTermOrd	Make a term order matrix from a given matrix
matrix	convert a list into a matrix
minors	list of minor determinants of a matrix
MinPoly	minimal polynomial of a matrix
NewFreeModule	create a new FreeModule
NewPolyRing	create a new PolyRing
NmzDiagInvariants	ring of invariants of a diagonalizable group action
NmzFiniteDiagInvariants	ring of invariants of a finite group action
NmzHilbertBasis	Hilbert Basis of a monoid
NmzIntersectionValRings	intersection of ring of valuations
NmzTorusInvariants	ring of invariants of torus action

NumCols	number of columns in a matrix
NumRows	number of rows in a matrix
operators, shortcuts	Special characters equivalent to commands
pfaffian	the Pfaffian of a skew-symmetric matrix
PreprocessPts	Reduce redundancy in a set of approximate points
product	the product of the elements of a list
RingOf	the ring of the object
RingsOf	list of the rings of an object
rk	rank of a matrix or module
SetCol	set a list as a row into a matrix
SetRow	set a list as a row into a matrix
submat	submatrix
sum	the sum of the elements of a list
SwapCols	swap two columns in a matrix
SwapRows	swap two rows in a matrix
toric	saturate toric ideals
transposed	the transposition of a matrix

### III-13.3 Commands and Functions returning MAT

adj	classical adjoint matrix (also known as adjugate)
apply	apply homomorphism
BlockMat	create a block matrix
BlockMat2x2	create a block matrix with 4 matrices
ColMat	single column matrix
ConcatAntiDiag	create a simple block matrix
ConcatDiag	create a simple block matrix
ConcatHor	create a simple block matrix
ConcatHorList	create a simple block matrix
ConcatVer	create a simple block matrix
ConcatVerList	create a simple block matrix
DiagMat	matrix with given diagonal
ElimHomogMat	matrix for elimination ordering
ElimMat	matrix for elimination ordering
FrobeniusMat	compute a matrix of the Frobenius Map
GensAsCols, GensAsRows	matrix of generators of a module
GFanGeneratorsOfLinealitySpace	...
GFanGeneratorsOfSpan	...
GFanGetFacets	...
GFanGetImpliedEquations	...
GFanGetUniquePoint	...
GFanRelativeInteriorPoint	relative interior point of a cone
GradingMat	matrix of generalized weights for indeterminates
HilbertMat	create a Hilbert matrix over $\mathbb{Q}\mathbb{Q}$
IdentityMat	the identity matrix
inverse	multiplicative inverse of matrix
jacobian	the Jacobian of a list of polynomials
LexMat	matrices for std. term-orderings
LinKer	find the kernel of a matrix
LinSolve	find a solution to a linear system
MakeMatByRows, MakeMatByCols	convert a list into a matrix
MakeTermOrd	Make a term order matrix from a given matrix

<code>matrix</code>	convert a list into a matrix
<code>MultiplicationMat</code>	the multiplication matrix of a ringelem
<code>NewMat</code>	Zero matrix
<code>NewMatFilled</code>	matrix filled with value
<code>NmzHilbertBasis</code>	Hilbert Basis of a monoid
<code>OrdMat</code>	matrix defining a term-ordering
<code>PrintSectionalMatrix</code>	print sectional matrix
<code>RandomSparseNonSing01Mat</code>	random sparse non-singular (0,1) matrix
<code>RandomUnimodularMat</code>	random unimodular matrix
<code>RevLexMat</code>	matrices for std. term-orderings
<code>RowMat</code>	single row matrix
<code>StdDegLexMat</code>	matrices for std. term-orderings
<code>StdDegRevLexMat</code>	matrices for std. term-orderings
<code>submat</code>	submatrix
<code>sylvester</code>	the Sylvester matrix of two polynomials
<code>TensorMat</code>	returns the tensor product of two matrices
<code>transposed</code>	the transposition of a matrix
<code>WeightsMatrix</code> [OBSOLESCENT]	[OBSOLESCENT] matrix of generalized weights for indeterminates
<code>XelMat</code>	matrices for std. term-orderings
<code>ZeroMat</code>	matrix filled with 0

# Chapter III-14

## MODULE

### III-14.1 Commands and Functions for MODULE

BettiDiagram	the diagram of the graded Betti numbers
BettiMatrix	the matrix of the graded Betti numbers
BettiNumbers	(Multi-)graded Betti numbers
colon	ideal or module quotient
elim	eliminate variables
GBasis	calculate a Groebner basis
GBasisTimeout	compute a Groebner basis with a timeout
GenRepr	representation in terms of generators
gens	list of generators of an ideal
GensAsCols, GensAsRows	matrix of generators of a module
HilbertSeries	the Hilbert-Poincare series
HilbertSeriesShifts	the Hilbert-Poincare series
homog	homogenize with respect to an indeterminate
IntersectionList	intersect lists, ideals, or modules
IsContained	checks if A is Contained in B
IsElem	checks if A is an element of B
IsHomog	test whether given polynomials are homogeneous
IsIn	check if one object is contained in another
IsZero	test whether an object is zero
LT	the leading term of an object
MinGens	list of minimal generators
minimalize [OBSOLESCE]	[OBSOLESCE]
MinSubsetOfGens	list of minimal generators
ModuleElem	create a module element
ModuleOf	the module environment of the object
NF	normal form
NumCompts	the number of components
operators, shortcuts	Special characters equivalent to commands
PrintBettiDiagram	the diagram of the graded Betti numbers
PrintBettiMatrix	print the matrix of the graded Betti numbers
PrintBettiNumbers	print the (multi-)graded Betti numbers
ReducedGBasis	compute reduced Groebner basis
res	free resolution
RingOf	the ring of the object
RingsOf	list of the rings of an object
rk	rank of a matrix or module
submodule	submodule generated by list
SubmoduleCols, SubmoduleRows	convert a matrix into a module

<code>SubmoduleOfMinGens</code>	submodule generated by minimal generators
<code>syz</code>	syzygy modules
<code>SyzOfGens</code>	syzygy module for a given set of generators

## III-14.2 Commands and Functions returning MODULE

<code>elim</code>	eliminate variables
<code>homog</code>	homogenize with respect to an indeterminate
<code>IntersectionList</code>	intersect lists, ideals, or modules
<code>LT</code>	the leading term of an object
<code>ModuleOf</code>	the module environment of the object
<code>NewFreeModule</code>	create a new FreeModule
<code>submodule</code>	submodule generated by list
<code>SubmoduleCols, SubmoduleRows</code>	convert a matrix into a module
<code>SubmoduleOfMinGens</code>	submodule generated by minimal generators
<code>syz</code>	syzygy modules
<code>SyzOfGens</code>	syzygy module for a given set of generators

## Chapter III-15

# MODULEELEM

### III-15.1 Introduction to MODULEELEM

An object of type MODULEELEM in CoCoA represents a module element; in CoCoA this usually means an element of the free module “ $P^r$ ”, where “ $P$ ” is a polynomial ring. For “ $v$ ” and “ $w$ ” MODULEELEM in the same MODULE, and “ $f$ ” RINGELEM in its base ring, the following are also MODULEELEM:

$+v$ ,  $-v$ ,  $f*v$ ,  $v*f$ ,  $v+w$ ,  $v-w$

See “ModuleElem” ([I-13.28](#) pg.203).

**See Also:** Commands and Functions for MODULEELEM([III-15.2](#) pg.427), Commands and Functions returning MODULEELEM([III-15.3](#) pg.428)

### III-15.2 Commands and Functions for MODULEELEM

compts	list of components of a ModuleElem
DivAlg	division algorithm
eval	substitute numbers or polynomials for indeterminates
FirstNonZero	the first non-zero entry in a MODULEELEM
FirstNonZeroPosn	the first non-zero entry in a MODULEELEM
GenRepr	representation in terms of generators
homog	homogenize with respect to an indeterminate
IsElem	checks if A is an element of B
IsHomog	test whether given polynomials are homogeneous
IsIn	check if one object is contained in another
IsTerm	checks if the argument is a term
IsZero	test whether an object is zero
LC	the leading coefficient of a polynomial or ModuleElem
LM	the leading monomial of a polynomial or ModuleElem
LPosn	the position of the leading power-product in a ModuleElem
LPP	the leading power-product of a polynomial or ModuleElem
LT	the leading term of an object
monomials	the list of monomials of a polynomial
NF	normal form
NonZero	remove zeroes from a list
NR	normal reduction
NumCompts	the number of components
product	the product of the elements of a list
RingsOf	list of the rings of an object
ScalarProduct	scalar product

<code>submodule</code>	submodule generated by list
<code>sum</code>	the sum of the elements of a list
<code>support</code>	the list of terms of a polynomial or moduleelem

### III-15.3 Commands and Functions returning MODULEELEM

<code>homog</code>	homogenize with respect to an indeterminate
<code>LM</code>	the leading monomial of a polynomial or ModuleElem
<code>LT</code>	the leading term of an object
<code>ModuleElem</code>	create a module element
<code>NF</code>	normal form
<code>NR</code>	normal reduction
<code>ReducedGBasis</code>	compute reduced Groebner basis

## Chapter III-16

# Creating new types

### III-16.1 Tagging an Object

If “E” is any CoCoA object and “S” a string, then the function “Tagged(E, S)” returns the object “E” tagged with the string “S”. The returned object is then of type “TAGGED(S)”. The function “tag” ([I-20.1 pg.311](#)) returns “S”, the tag string of an object, and the function “untagged” ([I-21.5 pg.326](#)) returns “E”, the original object, stripped of its tag.

This is the way to add a new type at run-time.

```
example
/**/ L := ["Dave", "March 14, 1959", 372];
/**/ M := Tagged(L, "MiscData"); -- L tagged with the string "MiscData"
/**/ type(L); -- L is a list
LIST
/**/ type(M); -- M is a tagged object
TAGGED("MiscData")
/**/ --M; -- Until a special print function is defined, the printing of M
-- is the same as L (with a WARNING)
--> WARNING: Cannot find "$BackwardCompatible.PrintTagged", so I'm implicitly untagging the value
--> ["Dave", "March 14, 1959", 372]
```

The next section explains how to define functions for pretty printing of tagged objects.

### III-16.2 Printing a Tagged Object

Suppose the object “E” is tagged with the string “S”. When one tries to print “E”—say with “Print E”—CoCoA looks for a user-defined function with name “Print\_S”. If no such function is available, CoCoA prints E as if it were not tagged, otherwise, it executes “Print\_S”.

```
example
/**/ L := ["Dave", "March 14", 1959, 372];
/**/ M := tagged(L, "MiscData");

/**/ Define SpecialPrinting(Dev, Obj)
/**/   Print Obj[1], "'s birthday is: ", Obj[2] on Dev;
/**/ EndDefine;

/**/ PrintTagged := record[MiscData := SpecialPrinting];

/**/ Print M;
Dave's birthday is: March 14
```

### III-16.3 Commands and Functions for Tags

The following are commands and functions involving tags:

<code>tag</code>	returns the tag string of an object
<code>tagged</code>	tag an object for pretty printing
<code>untagged</code>	untag an object