

CoCoA-5.4.1x Manual

John Abbott and Anna M. Bigatti

May 13, 2024

Contents

I	Alphabetical List of Commands	27
I-0	Special Characters	29
I-0.1	operators, shortcuts	29
I-1	A	31
I-1.1	abs	31
I-1.2	adj	31
I-1.3	AdjacentMinors	32
I-1.4	AffHilbert [OBSOLESCENT]	32
I-1.5	AffHilbertFn	32
I-1.6	AffHilbertSeries	32
I-1.7	AffPoincare [OBSOLESCENT]	33
I-1.8	alias	33
I-1.9	aliases	34
I-1.10	AllFactors	34
I-1.11	AllReducedGroebnerBases [OBSOLETE]	34
I-1.12	AlmostQR	34
I-1.13	and	35
I-1.14	append	35
I-1.15	apply [OBSOLESCENT]	36
I-1.16	ApproxPointsNBM	36
I-1.17	ApproxSolve	36
I-1.18	AreGensMonomial	37
I-1.19	AreGensSqFreeMonomial	38
I-1.20	ArrBettiNumbers	38
I-1.21	ArrBoolean	38
I-1.22	ArrBraid	39
I-1.23	ArrCatalanA	39
I-1.24	ArrCatalanB	39
I-1.25	ArrCatalanD	39
I-1.26	ArrCharPoly	39
I-1.27	ArrCone	40
I-1.28	ArrDeletion	40
I-1.29	ArrDerMod [OBSOLESCENT]	40

I-1.30	ArrDerModule	40
I-1.31	ArrExponents	41
I-1.32	ArrFlats	41
I-1.33	ArrGraphical	42
I-1.34	ArrLattice	42
I-1.35	ArrPoincarePoly	42
I-1.36	ArrRestriction	42
I-1.37	ArrShiA	43
I-1.38	ArrShiB	43
I-1.39	ArrShiCatalanA	43
I-1.40	ArrShiCatalanB	43
I-1.41	ArrShiCatalanD	44
I-1.42	ArrShiD	44
I-1.43	ArrSignedGraphical	44
I-1.44	ArrToMultiArr	44
I-1.45	ArrTuttePoly	45
I-1.46	ArrTypeB	45
I-1.47	ArrTypeD	45
I-1.48	ArtinianOrlikTeraoIdeal	46
I-1.49	ascii	46
I-1.50	AsINT	46
I-1.51	AsRAT	47
I-1.52	assert	47
I-2	B	49
I-2.1	BaseRing	49
I-2.2	BBasis5	49
I-2.3	BettiDiagram	49
I-2.4	BettiMatrix	50
I-2.5	BettiNumbers	51
I-2.6	binomial	51
I-2.7	BinomialRepr, BinomialReprShift	52
I-2.8	block	52
I-2.9	BlockMat	53
I-2.10	BlockMat2x2	53
I-2.11	Bool01	54
I-2.12	break	54
I-2.13	BringIn	55
I-3	C	57
I-3.1	Call [OBSOLETE]	57
I-3.2	CallOnGroebnerFanIdeals	57
I-3.3	CanonicalBasis	58

I-3.4	CanonicalHom	58
I-3.5	CanonicalRepr	59
I-3.6	CartesianProduct, CartesianProductList	59
I-3.7	Cast [OBSOLETE]	60
I-3.8	CatalanNumber	60
I-3.9	ceil	60
I-3.10	CFApprox	60
I-3.11	CFApproximants	61
I-3.12	ChainCanonicalHom	61
I-3.13	characteristic	61
I-3.14	CharPoly	62
I-3.15	ChebyshevPoly	62
I-3.16	CheckArgTypes	62
I-3.17	ciao	63
I-3.18	ClearDenom	63
I-3.19	close	63
I-3.20	CloseLog	64
I-3.21	CoCoA-4 mode	64
I-3.22	CocoaLimits	64
I-3.23	CocoaPackagePath	64
I-3.24	codomain	65
I-3.25	CoeffEmbeddingHom	65
I-3.26	CoeffHeight	65
I-3.27	coefficients	66
I-3.28	CoefficientsWRT	67
I-3.29	CoeffListWRT	67
I-3.30	CoeffListWRTSupport	68
I-3.31	CoeffOfTerm	68
I-3.32	CoeffRing	68
I-3.33	ColMat	69
I-3.34	colon	69
I-3.35	ColumnVectors [OBSOLETE]	70
I-3.36	CommonDenom	70
I-3.37	Comp [OBSOLETE]	70
I-3.38	CompleteToOrd [OBSOLETE]	70
I-3.39	compts	70
I-3.40	ComputeElimFirst	71
I-3.41	concat	71
I-3.42	ConcatAntiDiag	72
I-3.43	ConcatDiag	72
I-3.44	ConcatHor	72
I-3.45	ConcatHorList	73
I-3.46	ConcatLists	73

I-3.47	ConcatStrings	74
I-3.48	ConcatVer	74
I-3.49	ConcatVerList	74
I-3.50	ConstantCoeff	75
I-3.51	content	75
I-3.52	ContentFreeFactor	75
I-3.53	ContentWRT	76
I-3.54	ContFrac	76
I-3.55	ContFracToRat	77
I-3.56	continue	77
I-3.57	CoprimeFactor	77
I-3.58	CoprimeFactorBasis	78
I-3.59	count	78
I-3.60	covers	78
I-3.61	CpuTime	79
I-3.62	CRT	79
I-3.63	CRTPoly	80
I-3.64	CurrentRing	80
I-3.65	CurrentTypes	80
I-3.66	cyclotomic	81
I-3.67	CyclotomicFactorIndexes	81
I-3.68	CyclotomicIndex, CyclotomicTest	81
I-4	D	83
I-4.1	dashes	83
I-4.2	date	83
I-4.3	DecimalStr	83
I-4.4	define	84
I-4.5	DefiningIdeal	85
I-4.6	deg	86
I-4.7	den	86
I-4.8	DensePoly	87
I-4.9	DenSigma	87
I-4.10	depth	87
I-4.11	deriv	88
I-4.12	DerivationAction	89
I-4.13	describe	89
I-4.14	det	89
I-4.15	DF	90
I-4.16	DiagMat	90
I-4.17	DicksonPoly	91
I-4.18	diff	91
I-4.19	dim	91

	I-4.20	discriminant	91
	I-4.21	distrib	92
	I-4.22	div	92
	I-4.23	DivAlg	93
	I-4.24	domain	93
I-5	E		95
	I-5.1	E_ [OBSOLETE]	95
	I-5.2	eigenfactors	95
	I-5.3	eigenvectors	95
	I-5.4	ElapsedTime	96
	I-5.5	elim	96
	I-5.6	ElimHomogMat	97
	I-5.7	ElimMat	97
	I-5.8	EmbeddingHom	98
	I-5.9	EqSet	98
	I-5.10	Equality Operator	98
	I-5.11	EquiIsoDec	99
	I-5.12	error	99
	I-5.13	EulerTotient	99
	I-5.14	eval	100
	I-5.15	EvalHilbertFn	100
	I-5.16	EvalQuasiPoly	101
	I-5.17	exit	101
	I-5.18	exponents	101
	I-5.19	export	102
	I-5.20	Ext	102
	I-5.21	ExternalLibs	103
I-6	F		105
	I-6.1	factor	105
	I-6.2	FactorAlgExt [OBSOLESCENT]	106
	I-6.3	factorial	106
	I-6.4	FactorINT	106
	I-6.5	FactorMultiplicity	107
	I-6.6	FGLM5	107
	I-6.7	fibonacci	107
	I-6.8	fields	108
	I-6.9	first	108
	I-6.10	FirstCols, FirstRows	108
	I-6.11	FirstNonZero	109
	I-6.12	FirstNonZeroPosn	109
	I-6.13	FixedDivisor	109

I-6.14	flatten	110
I-6.15	FloatApprox	110
I-6.16	FloatStr	111
I-6.17	floor	111
I-6.18	FloorLog2, FloorLog10, FloorLogBase	112
I-6.19	FloorRoot	112
I-6.20	FloorSqrt	112
I-6.21	fold	113
I-6.22	FoldToListInput	113
I-6.23	for	113
I-6.24	foreach	114
I-6.25	format	115
I-6.26	FrbAlexanderDual	115
I-6.27	FrbAssociatedPrimes	116
I-6.28	FrbIrreducibleDecomposition	116
I-6.29	FrbMaximalStandardMonomials	116
I-6.30	FrbPrimaryDecomposition	117
I-6.31	FrobeniusMat	117
I-6.32	FrobeniusNormSq	117
I-6.33	func	118
I-6.34	Function [OBSOLETE]	118
I-6.35	functions [OBSOLETE]	118
I-6.36	FVector	118

I-7	G	121
I-7.1	GBasis	121
I-7.2	GBasis timeout	121
I-7.3	GBasisByHomog	122
I-7.4	GBM	122
I-7.5	gcd	122
I-7.6	GenericPoints	123
I-7.7	GenRepr	123
I-7.8	gens	124
I-7.9	GensAsCols, GensAsRows	125
I-7.10	GensJacobian	125
I-7.11	Get [OBSOLETE]	126
I-7.12	GetCol	126
I-7.13	GetCols	126
I-7.14	GetEnv	126
I-7.15	GetErrMesg	126
I-7.16	GetLine	127
I-7.17	GetRow	127
I-7.18	GetRows	127

I-7.19	GFanContainsPositiveVector	128
I-7.20	GFanGeneratorsOfLinealitySpace	128
I-7.21	GFanGeneratorsOfSpan	128
I-7.22	GFanGetAmbientDimension	128
I-7.23	GFanGetCodimension	128
I-7.24	GFanGetDimension	128
I-7.25	GFanGetDimensionOfLinealitySpace	129
I-7.26	GFanGetFacets	129
I-7.27	GFanGetImpliedEquations	129
I-7.28	GFanGetUniquePoint	129
I-7.29	GFanRelativeInteriorPoint	129
I-7.30	gin	129
I-7.31	GinJacobian	130
I-7.32	GradingDim	130
I-7.33	GradingMat	131
I-7.34	graeffe	131
I-7.35	GraverBasis	132
I-7.36	GroebnerFanIdeals	132
I-7.37	GroebnerFanReducedGBases	133
I-8	H	135
I-8.1	HadamardBoundSq	135
I-8.2	HasGBasis	135
I-8.3	HColon	136
I-8.4	HermitePoly	136
I-8.5	HGBM	136
I-8.6	hilbert [OBSOLESCECENT]	137
I-8.7	HilbertBasisKer	137
I-8.8	HilbertFn	137
I-8.9	HilbertMat	138
I-8.10	HilbertPoly	138
I-8.11	HilbertSeries	139
I-8.12	HilbertSeriesMultiDeg	140
I-8.13	HilbertSeriesShifts	140
I-8.14	homog	141
I-8.15	HomogCompt	141
I-8.16	HomogElimMat [OBSOLESCECENT]	142
I-8.17	HSaturation	142
I-8.18	HVector	142
I-9	I	143
I-9.1	ID [OBSOLETE]	143
I-9.2	ideal	143

I-9.3	<code>IdealAndSeparatorsOfPoints</code>	143
I-9.4	<code>IdealAndSeparatorsOfProjectivePoints</code>	144
I-9.5	<code>IdealOfGBasis</code>	146
I-9.6	<code>IdealOfMinGens</code>	146
I-9.7	<code>IdealOfPoints</code>	147
I-9.8	<code>IdealOfProjectivePoints</code>	147
I-9.9	<code>IdentityMat</code>	148
I-9.10	<code>if</code>	148
I-9.11	<code>ILogBase [OBSOLETE]</code>	149
I-9.12	<code>image [OBSOLESCENT]</code>	149
I-9.13	<code>implicit</code>	150
I-9.14	<code>ImplicitHypersurface</code>	150
I-9.15	<code>ImplicitPlot</code>	151
I-9.16	<code>ImplicitPlotOn</code>	151
I-9.17	<code>ImportByRef, ImportByValue</code>	152
I-9.18	<code>in</code>	152
I-9.19	<code>incr, decr</code>	152
I-9.20	<code>indent, IndentStr</code>	153
I-9.21	<code>indet</code>	153
I-9.22	<code>IndetIndex</code>	154
I-9.23	<code>IndetName</code>	154
I-9.24	<code>indets</code>	155
I-9.25	<code>IndetsProd</code>	155
I-9.26	<code>IndetSubscripts</code>	156
I-9.27	<code>IndetSymbols</code>	156
I-9.28	<code>InducedHom</code>	157
I-9.29	<code>InitialIdeal</code>	157
I-9.30	<code>insert [OBSOLESCENT]</code>	158
I-9.31	<code>Interpolate</code>	158
I-9.32	<code>interreduce</code>	159
I-9.33	<code>interreduced</code>	159
I-9.34	<code>intersection</code>	160
I-9.35	<code>IntersectionList</code>	160
I-9.36	<code>inverse</code>	161
I-9.37	<code>InverseSystem</code>	161
I-9.38	<code>InvTotient</code>	161
I-9.39	<code>IsAntiSymmetric</code>	162
I-9.40	<code>IsArrCentral</code>	162
I-9.41	<code>IsArrFree</code>	162
I-9.42	<code>IsAtEOF</code>	162
I-9.43	<code>IsCommutative</code>	163
I-9.44	<code>IsConstant</code>	163
I-9.45	<code>IsContained</code>	163

I-9.46	IsCoprime	164
I-9.47	IsDefined	164
I-9.48	IsDiagonal	164
I-9.49	IsDivisible	165
I-9.50	IsElem	165
I-9.51	IsEmpty	165
I-9.52	IsEven, IsOdd	166
I-9.53	IsEvenPoly, IsOddPoly	166
I-9.54	IsFactorClosed	166
I-9.55	IsField	167
I-9.56	IsFiniteField	167
I-9.57	IsFractionField	167
I-9.58	IsHomog	168
I-9.59	IsIn	168
I-9.60	IsIndet	169
I-9.61	IsIndetPosPower	169
I-9.62	IsInImage	169
I-9.63	IsInjective	170
I-9.64	IsInRadical	170
I-9.65	IsInSubalgebra [OBSOLETE]	171
I-9.66	IsInteger	171
I-9.67	IsIntegralDomain	171
I-9.68	IsInvertible	172
I-9.69	IsIrred	172
I-9.70	IsLattice	172
I-9.71	IsLexSegment	173
I-9.72	IsLRSDegenerate	173
I-9.73	IsLRSDegenerateOrder	173
I-9.74	IsMaximal	174
I-9.75	IsMinusOne	174
I-9.76	IsMultiArrFree	174
I-9.77	IsNumber [OBSOLETE]	175
I-9.78	IsOne	175
I-9.79	IsPalindromic	175
I-9.80	IsPolyRing	175
I-9.81	IsPosetGraded	176
I-9.82	IsPositiveGrading	176
I-9.83	IsPowerOf2	177
I-9.84	IsPrimary	177
I-9.85	IsPrime	177
I-9.86	IsPrimitivePoly	178
I-9.87	IsProbPrime	178
I-9.88	IsPthPower	178

I-9.89	IsQQ	179
I-9.90	isqrt [OBSOLETE]	179
I-9.91	IsQuotientRing	179
I-9.92	IsRadical	179
I-9.93	IsRational	179
I-9.94	IsSigmaGoodPrime	180
I-9.95	IsSqFree	180
I-9.96	IsSquare	181
I-9.97	IsStable	181
I-9.98	IsStdGraded	181
I-9.99	IsStronglyStable	181
I-9.100	IsSubset	182
I-9.101	IsSurjective	182
I-9.102	IsSymmetric	182
I-9.103	IsTerm	183
I-9.104	IsTermOrdering	183
I-9.105	IsTree5	184
I-9.106	IsTrueGCDDomain	184
I-9.107	IsZero	184
I-9.108	IsZeroCol, IsZeroRow	185
I-9.109	IsZeroDet	185
I-9.110	IsZeroDim	185
I-9.111	IsZeroDivisor	186
I-9.112	IsZZ	186
I-9.113	It	186
I-10	J	189
I-10.1	JacobianMat	189
I-10.2	JanetBasis	189
I-11	K	191
I-11.1	ker	191
I-11.2	KroneckerProd	191
I-11.3	KroneckerSymbol	192
I-12	L	193
I-12.1	LaguerrePoly	193
I-12.2	last	193
I-12.3	latex	193
I-12.4	LawrenceMat	194
I-12.5	LC	194
I-12.6	lcm	195
I-12.7	len	195
I-12.8	LexMat	196

I-12.9	LexSegmentIdeal	196
I-12.10	LF	196
I-12.11	LinearSimplify	197
I-12.12	LinKer	197
I-12.13	LinKerBasis	198
I-12.14	LinKerModP [OBSOLETE]	198
I-12.15	LinKerZZ	199
I-12.16	LinSol [OBSOLETE]	199
I-12.17	LinSolve	199
I-12.18	LM	200
I-12.19	log [OBSOLESCENT]	200
I-12.20	LogCardinality	200
I-12.21	LPosn	201
I-12.22	LPP	201
I-12.23	LRSDegeneracyOrder	202
I-12.24	LT	202
I-13	M	205
I-13.1	MakeCheck	205
I-13.2	MakeMatByRows, MakeMatByCols	205
I-13.3	MakeSet	206
I-13.4	MakeTerm	206
I-13.5	MakeTermOrd [OBSOLESCENT]	206
I-13.6	MakeTermOrdMat	206
I-13.7	MantissaAndExponent10	207
I-13.8	MantissaAndExponent2	207
I-13.9	Manual	208
I-13.10	MapDown [OBSOLETE]	208
I-13.11	matrix	209
I-13.12	max	209
I-13.13	MaxBy	210
I-13.14	MaxChains	210
I-13.15	MayerVietorisTreeN1	211
I-13.16	min	211
I-13.17	MinBy	211
I-13.18	MinGBoverZZ [PROTOTYPE]	212
I-13.19	MinGens	212
I-13.20	MinGensGeneral [OBSOLESCENT]	213
I-13.21	minimalize [OBSOLESCENT]	213
I-13.22	minimalized [OBSOLESCENT]	213
I-13.23	MinimalPresentation	213
I-13.24	minors	213
I-13.25	MinPoly	213

I-13.26	MinPolyQuot	214
I-13.27	MinPowerInIdeal	215
I-13.28	MinSubsetOfGens	215
I-13.29	mod	215
I-13.30	Mod2Rat [OBSOLETE]	216
I-13.31	ModuleElem	216
I-13.32	ModuleOf	216
I-13.33	moebius	217
I-13.34	MoebiusFn	217
I-13.35	monic	217
I-13.36	monomials	218
I-13.37	MonsInIdeal	218
I-13.38	MSatLinSolve	219
I-13.39	MultiArrDerMod [OBSOLESCE]	220
I-13.40	MultiArrDerModule	220
I-13.41	MultiArrExponents	220
I-13.42	MultiArrRestrictionZiegler	220
I-13.43	MultiArrToArr	221
I-13.44	MultiplicationMat	221
I-13.45	multiplicity	221
I-14	N	223
I-14.1	NewFractionField	223
I-14.2	NewFreeModule	223
I-14.3	NewFreeModuleForSyz	224
I-14.4	NewId [OBSOLETE]	224
I-14.5	NewLine [OBSOLESCE]	224
I-14.6	NewList	225
I-14.7	NewMat	225
I-14.8	NewMatFilled	225
I-14.9	NewPolyRing	226
I-14.10	NewPolyRingWeights	226
I-14.11	NewQuotientRing	227
I-14.12	NewRingFp [OBSOLESCE]	227
I-14.13	NewRingTwinFloat	227
I-14.14	NewWeylAlgebra	228
I-14.15	NewZZmod	228
I-14.16	NextPrime, NextProbPrime	228
I-14.17	NF	229
I-14.18	NFsAreZero [OBSOLETE]	229
I-14.19	NmzComputation	229
I-14.20	NmzDiagInvariants	230
I-14.21	NmzEhrhartRing	231

I-14.22	NmzFiniteDiagInvariants	231
I-14.23	NmzHilbertBasis	231
I-14.24	NmzHilbertBasisKer	232
I-14.25	NmzIntClosureMonIdeal	232
I-14.26	NmzIntClosureToricRing	233
I-14.27	NmzIntersectionValRings	233
I-14.28	NmzNormalToricRing	233
I-14.29	NmzSetVerbosityLevel	234
I-14.30	NmzTorusInvariants	234
I-14.31	NmzVerbosityLevel	234
I-14.32	NonZero	235
I-14.33	not	235
I-14.34	NR	235
I-14.35	num	236
I-14.36	NumBChambers	236
I-14.37	NumChambers	236
I-14.38	NumCols	236
I-14.39	NumCompts	237
I-14.40	NumGens	237
I-14.41	NumIndets	237
I-14.42	NumPartitions	238
I-14.43	NumRealRoots	238
I-14.44	NumRows	238
I-14.45	NumTerms	238
I-15	O	241
I-15.1	one	241
I-15.2	OpenIFile	241
I-15.3	OpenIString	242
I-15.4	OpenLog	242
I-15.5	OpenOFile	243
I-15.6	OpenOString	243
I-15.7	OpenSocket	244
I-15.8	Option [OBSOLETE]	244
I-15.9	or	244
I-15.10	Order Comparison Operators	245
I-15.11	OrdMat	245
I-15.12	OrlikTeraoIdeal	246
I-16	P	247
I-16.1	package	247
I-16.2	PackageOf	247
I-16.3	packages	248

I-16.4	panel [OBSOLETE]	248
I-16.5	panels [OBSOLETE]	248
I-16.6	partitions	248
I-16.7	permutations	248
I-16.8	PerpIdealOfForm	249
I-16.9	pfaffian	249
I-16.10	PkgName	249
I-16.11	PlayCantStop	250
I-16.12	PlotPoints	250
I-16.13	PlotPointsOn	251
I-16.14	poincare [OBSOLESCENT]	251
I-16.15	PoincareMultiDeg [OBSOLETE]	251
I-16.16	PoincareShifts [OBSOLETE]	251
I-16.17	PolyAlgebraHom	251
I-16.18	PolyRingHom	252
I-16.19	PosetCharPoly	252
I-16.20	PosetDual	253
I-16.21	PosetJoin	253
I-16.22	PosetMeet	253
I-16.23	PosetNRank	254
I-16.24	PosetPoincarePoly	254
I-16.25	PosetRank	255
I-16.26	power	255
I-16.27	PowerMod	255
I-16.28	PreImage [OBSOLESCENT]	256
I-16.29	preimage0	256
I-16.30	PreprocessPts	256
I-16.31	PrevPrime, PrevProbPrime	257
I-16.32	prim	257
I-16.33	PrimaryDecomposition	258
I-16.34	PrimaryDecomposition0 [OBSOLETE]	258
I-16.35	PrimaryDecompositionGTZ0	258
I-16.36	PrimaryHilbertSeries	259
I-16.37	PrimaryPoincare [OBSOLESCENT]	260
I-16.38	PrimitiveRoot	260
I-16.39	primorial	260
I-16.40	print	260
I-16.41	print on	261
I-16.42	PrintBettiDiagram	261
I-16.43	PrintBettiMatrix	262
I-16.44	PrintBettiNumbers	262
I-16.45	println	263
I-16.46	PrintRes	263

I-16.47	PrintSectionalMatrix	264
I-16.48	product	264
I-16.49	protect	265
I-16.50	PthRoot	265
I-17	Q	267
I-17.1	QQ	267
I-17.2	QQEmbeddingHom	267
I-17.3	quit	268
I-17.4	QuotientBasis	268
I-17.5	QuotientBasisSorted	268
I-17.6	QuotientingHom	269
I-17.7	QZP	269
I-18	R	271
I-18.1	radical	271
I-18.2	RadicalOfUnmixed	271
I-18.3	random	272
I-18.4	randomize [OBSOLETE]	272
I-18.5	randomized [OBSOLETE]	272
I-18.6	RandomLinearForm	272
I-18.7	RandomNBitPrime	273
I-18.8	RandomPermutation	273
I-18.9	RandomSmallPrime	273
I-18.10	RandomSparseNonSing01Mat	273
I-18.11	RandomSubset	274
I-18.12	RandomSubsetIndices	274
I-18.13	RandomTuple	275
I-18.14	RandomTupleIndices	275
I-18.15	RandomUnimodularMat	275
I-18.16	rank [OBSOLESCENT]	276
I-18.17	RationalSolve	276
I-18.18	RationalSolveHomog	276
I-18.19	RatReconstructByContFrac	277
I-18.20	RatReconstructByLattice	277
I-18.21	RatReconstructPoly	278
I-18.22	RatReconstructWithBounds	278
I-18.23	ReadExpr [OBSOLESCENT]	279
I-18.24	RealRootRefine	279
I-18.25	RealRoots	279
I-18.26	RealRootsApprox	280
I-18.27	record	280
I-18.28	record field selector	281

I-18.29	ReducedGBasis	281
I-18.30	ref	281
I-18.31	RefineGCDFreeBasis [OBSOLETE]	282
I-18.32	reg	282
I-18.33	RegularityIndex	283
I-18.34	RelNotes	283
I-18.35	ReloadMan	284
I-18.36	remove	284
I-18.37	repeat	284
I-18.38	res	285
I-18.39	reseed	286
I-18.40	Reset [OBSOLETE]	286
I-18.41	ResetPanels [OBSOLETE]	286
I-18.42	resultant	286
I-18.43	return	287
I-18.44	reverse, reversed	287
I-18.45	RevLexMat	287
I-18.46	rgin	288
I-18.47	RingElem	288
I-18.48	RingElemList, RingElems	289
I-18.49	RingEnv [OBSOLETE]	290
I-18.50	RingID	290
I-18.51	RingOf	290
I-18.52	RingQQ	291
I-18.53	RingQQt	291
I-18.54	RingSet [OBSOLETE]	291
I-18.55	RingsOf	292
I-18.56	RingZZ	292
I-18.57	rk	292
I-18.58	RMap [OBSOLESCENT]	293
I-18.59	RootBound	293
I-18.60	RootBoundTransform	293
I-18.61	round	294
I-18.62	RowMat	294
I-18.63	rref	294
I-19	S	297
I-19.1	SAGBI, SAGBIHomog	297
I-19.2	SatSAGBI	297
I-19.3	saturate	298
I-19.4	ScalarProduct	299
I-19.5	ScientificStr	299
I-19.6	SectionalMatrix	300

I-19.7	seed [OBSOLETE]	300
I-19.8	SeparatorsOfPoints	300
I-19.9	SeparatorsOfProjectivePoints	301
I-19.10	SetCol	302
I-19.11	SetEntry	303
I-19.12	SetRow	303
I-19.13	SetStackSize	303
I-19.14	SetVerbosityLevel	303
I-19.15	shape	304
I-19.16	sign	304
I-19.17	SimplestBinaryRatBetween	305
I-19.18	SimplestRatBetween	305
I-19.19	SimplexInfo	305
I-19.20	SimplicialHomology	306
I-19.21	singular value decomposition	307
I-19.22	size [OBSOLETE]	307
I-19.23	skip	307
I-19.24	SleepFor	307
I-19.25	SmallestNonDivisor	307
I-19.26	SmoothFactor [OBSCULESCENT]	308
I-19.27	SolomonTeraoIdeal	308
I-19.28	sort	308
I-19.29	SortBy	308
I-19.30	sorted	309
I-19.31	SortedBy	310
I-19.32	source	310
I-19.33	SourceRegion	310
I-19.34	spaces	311
I-19.35	sprint	311
I-19.36	SprintTrunc	311
I-19.37	SqFreeFactor	312
I-19.38	StableBBasis5	312
I-19.39	StableIdeal	313
I-19.40	StagedTrees	313
I-19.41	StandardInput	314
I-19.42	StandardOutput	314
I-19.43	StarRoot	315
I-19.44	starting	315
I-19.45	StdBasis	315
I-19.46	StdDegLexMat	316
I-19.47	StdDegRevLexMat	316
I-19.48	StronglyStableIdeal	316
I-19.49	SturmSeq	317

I-19.50	SubalgebraHom	317
I-19.51	SubalgebraMap [OBSOLETE]	317
I-19.52	SubalgebraMinGens	318
I-19.53	SubalgebraRepr [OBSOLESCENT]	318
I-19.54	submat	318
I-19.55	submodule	319
I-19.56	SubmoduleCols, SubmoduleRows	319
I-19.57	SubmoduleOfMinGens	319
I-19.58	subsets	320
I-19.59	subst	320
I-19.60	substring	321
I-19.61	sum	321
I-19.62	support	322
I-19.63	swap	322
I-19.64	SwapCols	322
I-19.65	SwapRows	323
I-19.66	SwinertonDyerPoly	323
I-19.67	SylvesterMat	323
I-19.68	SymbolRange	324
I-19.69	SymmetricPolys	324
I-19.70	SystemCommand	324
I-19.71	syz	325
I-19.72	SyzOfGens	326
I-20	T	327
I-20.1	tag	327
I-20.2	tagged	327
I-20.3	tail	328
I-20.4	TensorMat [OBSOLESCENT]	328
I-20.5	TgCone	328
I-20.6	ThmProve [PROTOTYPE]	328
I-20.7	TimeFrom	329
I-20.8	TimeOfDay	329
I-20.9	TmpNBM [OBSOLETE]	329
I-20.10	TopLevel	329
I-20.11	TopLevelFunctions	330
I-20.12	toric	330
I-20.13	transposed	331
I-20.14	try	332
I-20.15	tuples	332
I-20.16	TVecFromHF	332
I-20.17	TVecPoints	333
I-20.18	TVecPrintRes	333

	I-20.19	TVecToHF	333
	I-20.20	type	334
I-21	U		335
	I-21.1	UnivariateIndetIndex	335
	I-21.2	UniversalGBasis	335
	I-21.3	unprotect	336
	I-21.4	Unset [OBSOLETE]	336
	I-21.5	untagged	336
	I-21.6	use	336
I-22	V		339
	I-22.1	valuation [OBSOLETE]	339
	I-22.2	VerbosityLevel	339
	I-22.3	VersionInfo	340
I-23	W		341
	I-23.1	wdeg	341
	I-23.2	WeightsMatrix [OBSOLESCE]	341
	I-23.3	while	342
	I-23.4	WithoutNth	342
	I-23.5	WLog [OBSOLETE]	342
I-24	X		343
	I-24.1	XelMat	343
I-25	Z		345
	I-25.1	zero	345
	I-25.2	ZeroMat	345
	I-25.3	ZPQ	346
	I-25.4	ZZ	346
II	CoCoA Tutorials and Programming Language		347
II-1	CoCoA Tutorials		349
	II-1.1	Basic Tutorial for CoCoA-5	349
	II-1.2	Tutorial: manual	349
	II-1.3	Tutorial: Emacs UI (basic)	349
	II-1.4	Tutorial: variables, assignment	350
	II-1.5	Tutorial: arithmetic operators	350
	II-1.6	Tutorial: printing	351
	II-1.7	Tutorial: lists	351
	II-1.8	Tutorial: polynomial rings, use command	352
	II-1.9	Tutorial: polynomials	352

II-1.10	Tutorial: defining new functions	353
II-1.11	Tutorial: defining new functions (advanced)	353
II-1.12	Tutorial: homomorphisms	354
II-1.13	Tutorial: programming and debugging	354
II-1.14	Tutorial: feedback and reporting bugs	355
II-2	Introduction to CoCoA Programming	357
II-2.1	An Overview of CoCoA Programming	357
II-2.2	All CoCoA commands	357
II-3	Language Elements	359
II-3.1	Character Set and Special Symbols	359
II-3.2	Identifiers	359
II-3.3	Reserved Names	360
II-3.4	Comments	360
II-4	Operators	361
II-4.1	CoCoA Operators: introduction	361
II-4.2	Algebraic Operators	361
II-4.3	Relational Operators	362
II-4.4	Selection Operators	362
II-4.5	Range Operator	362
II-5	Evaluation and Assignment	365
II-5.1	Evaluation	365
II-5.2	Assignment	365
II-6	Flow Control: Conditional Statements and Loops	367
II-6.1	Commands and Functions for Branching	367
II-6.2	Commands and Functions for Loops	367
II-7	Verbosity and interrupt	369
II-7.1	Introduction to verbosity and interrupt	369
II-7.2	Commands and Functions implementing Verbosity	369
II-7.3	Commands and Functions implementing interruption	370
II-8	Input/Output	371
II-8.1	Introduction to IO	371
II-8.2	Standard IO	371
II-8.3	File IO	371
II-8.4	String IO	372
II-8.5	Commands and Functions for IO	372
II-9	CoCoA Packages	375
II-9.1	Introduction to Packages	375
II-9.2	First Example of a Package	375

II-9.3	Package essentials	376
II-9.4	Global Aliases	376
II-9.5	Sharing Your Package	376
II-9.6	Commands and Functions for Packages	376
II-9.7	Supported Packages	377
II-9.8	Galois Package	377
II-9.9	Integer Programming	377
II-9.10	Algebra of Invariants	377
II-9.11	Special Varieties	377
II-9.12	Statistics	378
II-9.13	Geometrical Theorem-Proving [PROTOTYPE]	378
II-9.14	Conductor	378
II-9.15	Matrix Normal Form	378
II-9.16	Control	379
II-10	Linked libraries	381
II-10.1	CoCoALib	381
II-10.2	GMP	381
II-10.3	GSL	381
II-10.4	Frobby	381
II-10.5	MathSAT	381
II-10.6	Normaliz	381
II-11	Migrating from CoCoA-4 and keeping up-to-date	383
II-11.1	Changes in the CoCoA language	383
II-11.2	Recent changes in the CoCoA-5 language	384
II-11.3	Obsolete and obsolescent functions	384
III	CoCoA datatypes	387
III-1	BOOL	389
III-1.1	Introduction to BOOL	389
III-1.2	Commands and Functions for BOOL	389
III-1.3	Commands and Functions returning BOOL	389
III-2	INT	393
III-2.1	Introduction to INT	393
III-2.2	Commands and Functions for INT	393
III-2.3	Commands and Functions returning INT	396
III-3	RAT	399
III-3.1	Introduction to RAT	399
III-3.2	Commands and Functions for RAT	399
III-3.3	Commands and Functions returning RAT	400

III-4	STRING	401
III-4.1	String Literals	401
III-4.2	String Operations	401
III-4.3	Commands and Functions for STRING	402
III-4.4	Commands and Functions returning STRING	403
III-5	LIST	405
III-5.1	Introduction to LIST	405
III-5.2	List Constructors	406
III-5.3	Commands and Functions for LIST	406
III-5.4	Commands and Functions returning LIST	410
III-6	RECORD	413
III-6.1	Introduction to RECORD	413
III-6.2	Commands and Functions for RECORD	414
III-6.3	Commands and Functions returning RECORD	414
III-7	FUNCTION	417
III-7.1	Introduction to FUNCTION	417
III-7.2	FUNCTIONs are first class objects	417
III-7.3	Commands and Functions for FUNCTION	417
III-7.4	Commands and Functions returning FUNCTION	418
III-8	TYPE	419
III-8.1	Commands and Functions for TYPE	419
III-8.2	Commands and Functions returning TYPE	419
III-9	RING	421
III-9.1	Introduction to RING	421
III-9.2	Polynomial Rings	421
III-9.3	Coefficient Rings	422
III-9.4	Indeterminates	422
III-9.5	Term Orderings	422
III-9.6	Module Orderings	423
III-9.7	Quotient Rings	424
III-9.8	Commands and Functions for RING	424
III-9.9	Commands and Functions returning RING	426
III-10	RINGHOM	427
III-10.1	Introduction to RINGHOM	427
III-10.2	Composition of RINGHOM	427
III-10.3	Commands and Functions for RINGHOM	428
III-10.4	Commands and Functions returning RINGHOM	428
III-11	RINGELEM	429

III-11.1	Introduction to RINGELEM	429
III-11.2	Evaluation of Polynomials	430
III-11.3	Commands and Functions for RINGELEM	430
III-11.4	Commands and Functions returning RINGELEM	433
III-12	IDEAL	435
III-12.1	Commands and Functions for IDEAL	435
III-12.2	Commands and Functions returning IDEAL	437
III-13	MAT	439
III-13.1	Introduction to MAT	439
III-13.2	Commands and Functions for MAT	439
III-13.3	Commands and Functions returning MAT	441
III-14	MODULE	443
III-14.1	Commands and Functions for MODULE	443
III-14.2	Commands and Functions returning MODULE	444
III-15	MODULEELEM	445
III-15.1	Introduction to MODULEELEM	445
III-15.2	Commands and Functions for MODULEELEM	445
III-15.3	Commands and Functions returning MODULEELEM	446
III-16	Creating new types	447
III-16.1	Tagging an Object	447
III-16.2	Printing a Tagged Object	447
III-16.3	Commands and Functions for Tags	448

Part I

Alphabetical List of Commands

Chapter I-0

Special Characters

I-0.1 operators, shortcuts

syntax		
A := B	A: variable, B: OBJECT	assignment
A ::= K[...]	A: variable, K: RING	assignment special syntax for rings
A = B	A,B: OBJECT	returns BOOL
A <> B	A,B: OBJECT	returns BOOL
A < B	A,B: OBJECT	returns BOOL
A <= B	A,B: OBJECT	returns BOOL
A > B	A,B: OBJECT	returns BOOL
A >= B	A,B: OBJECT	returns BOOL
A >< B	A,B: LIST	returns LIST
A..B	A,B: INT	returns LIST
A..B	A,B: RINGELEM (indets)	returns LIST
[...]		returns LIST
[... ...]		returns LIST
I : J	I, J: IDEAL	returns IDEAL
L[N]	L: LIST, N: INT	returns OBJECT
S[N]	S: STRING, N: INT	returns STRING
M[i,j]	M: MAT, i,j: INT	returns RINGELEM
R.F	R: RECORD and F field name	returns OBJECT
R/I	R: RING, I: IDEAL	returns RING
E	E: expression	returns OBJECT
?S	S: STRING	prints manual
<< S	S: STRING	[OBSOLESCENT]

“A := B;” compute “B” then assign the result to “A”

“A ::= <ring-spec>” for the special ring syntax (see “NewPolyRing” ([I-14.9 pg.226](#)))

“A = B” test whether “A” and “B” are equal (see “Order Comparison Operators” ([I-15.10 pg.245](#)))

“A <> B” test whether “A” and “B” are not equal (see “Order Comparison Operators” ([I-15.10 pg.245](#)))

“A < B” test whether “A” is smaller than “B” (see “Order Comparison Operators” ([I-15.10 pg.245](#)))

“A >< B” equivalent to “CartesianProduct(A, B)”, “CartesianProductList([A,B])”

“A..B” is the “Range Operator” ([II-4.5 pg.362](#)) (see “List Constructors” ([III-5.2 pg.406](#)))

“[...]” build a new list (see “List Constructors” ([III-5.2 pg.406](#)))

“[...|...]” build a new list (see “List Constructors” ([III-5.2 pg.406](#)))

“I : J” equivalent to “colon(I, J)”

“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-18.28 pg.281](#)) for field named `"F"` of record `"R"`

`"R/I"` equivalent to `"NewQuotientRing(R,I)"`

`"***E***"` interpret `"E"` in `"CoCoA-4 mode"` ([I-3.21 pg.64](#))

`"$<< S$"` OBSOLESCENT equivalent to `"source(S)"`

`"? string"` prints the manual page for `"string"`, or a list of matching manual pages

See Also: [colon\(I-3.34 pg.69\)](#), [Equality Operator\(I-5.10 pg.98\)](#), [Order Comparison Operators\(I-15.10 pg.245\)](#), [List Constructors\(III-5.2 pg.406\)](#), [CartesianProduct](#), [CartesianProductList\(I-3.6 pg.59\)](#), [NewPolyRing\(I-14.9 pg.226\)](#), [NewQuotientRing\(I-14.11 pg.227\)](#), [record field selector\(I-18.28 pg.281\)](#), [Range Operator\(II-4.5 pg.362\)](#), [source\(I-19.32 pg.310\)](#), [Manual\(I-13.9 pg.208\)](#), [Character Set and Special Symbols\(II-3.1 pg.359\)](#), [CoCoA Operators: introduction\(II-4.1 pg.361\)](#)

Chapter I-1

A

I-1.1 abs

syntax

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

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
```

See Also: [sign\(I-19.16 pg.304\)](#)

I-1.2 adj

syntax

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

This function returns the classical adjoint (aka *adjugate*) 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]])

/**/ FF_5 := NewZZmod(5);
/**/ adj(matrix(FF_5, [[1,2],[3,1]]));
matrix( /*ZZ/(5)*/
  [[1, -2],
  [2, 1]])
```

See Also: [inverse\(I-9.36 pg.161\)](#)

I-1.3 AdjacentMinors

syntax

```
minors(M: MAT, N: INT): LIST
```

This function returns the list of all adjacent minors (determinants of adjacent $N \times N$ submatrices) of M .

example

```
/**/ use P := QQ[x[1..4],y[1..4]];
/**/ M := matrix([indets(P,"x"), indets(P,"y")]); M;
matrix( /*RingWithID(14, "QQ[x[1],x[2],x[3],x[4],y[1],y[2],y[3],y[4]]")*/
  [[x[1], x[2], x[3], x[4]],
   [y[1], y[2], y[3], y[4]]])

/**/ AdjacentMinors(M,2);
[-x[2]*y[1] +x[1]*y[2], -x[3]*y[2] +x[2]*y[3], -x[4]*y[3] +x[3]*y[4]]
```

See Also: [det\(I-4.14 pg.89\)](#), [minors\(I-13.24 pg.213\)](#)

I-1.4 AffHilbert [OBSOLESCENT]

Renamed to “AffHilbertFn” ([I-1.5 pg.32](#)).

I-1.5 AffHilbertFn

syntax

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

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-5.15 pg.100](#)) 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\(I-1.6 pg.32\)](#), [EvalHilbertFn\(I-5.15 pg.100\)](#), [HilbertPoly\(I-8.10 pg.138\)](#), [HVector\(I-8.18 pg.142\)](#), [HilbertSeries\(I-8.11 pg.139\)](#)

I-1.6 AffHilbertSeries

syntax

```
AffHilbertSeries(R: (Poly or Quotient)RING): TAGGED("$hp.PSeries")
```

This function computes the affine Hilbert-Poincare series of “ M ”. The grading must be a positive Z^1 -grading (*i.e.* “GradingMat” ([I-7.33 pg.131](#)) 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”.

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-1.5 pg.32), HilbertSeries(I-8.11 pg.139)

I-1.7 AffPoincare [OBSOLESCENT]

Renamed to “AffHilbertSeries” (I-1.6 pg.32).

I-1.8 alias

syntax

```
alias B_1,...,B_r

where each B_i is a \textbf{binding} of the form: Identifier := $PackageName
```

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-1.9 pg.34). For examples, see the chapter on packages in the manual, in particular the section, “Global Aliases” (II-9.4 pg.376). 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-1.9 pg.34), Introduction to Packages(II-9.1 pg.375)

I-1.9 aliases

— syntax —

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

This function prints a list of global aliases for packages. Aliases are formed with the function “`alias`” ([I-1.8 pg.33](#)).

— example —

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

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

See Also: `alias`([I-1.8 pg.33](#)), `Introduction to Packages`([II-9.1 pg.375](#))

I-1.10 AllFactors

— syntax —

```
AllFactors(N: INT): INT
```

This function produces a list of all positive factors of the positive integer “`N`”; it is limited to “small” integers.

See Also: `FactorINT`([I-6.4 pg.106](#))

I-1.11 AllReducedGroebnerBases [OBSOLETE]

Renamed to “`GroebnerFanIdeals`” ([I-7.36 pg.132](#)).

I-1.12 AlmostQR

— syntax —

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

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);
/**/ indent(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);
/**/ indent(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-9.15](#) pg.378)

I-1.13 and

— syntax —

```

A and B
where A, B: BOOL; return BOOL
```

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.244), not([I-14.33](#) pg.235)

I-1.14 append

— syntax —

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

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-18.30](#) pg.281), concat([I-3.41](#) pg.71), ConcatLists([I-3.46](#) pg.73), remove([I-18.36](#) pg.284)

I-1.15 apply [OBSOLESCENT]

To apply a homomorphism “phi” to all elements in second argument “X” (RINGELEM, LIST, or MAT) just use normal **function call** syntax: “phi(X)”.

See also “Introduction to RINGHOM” (III-10.1 pg.427)

example

```
/**/ use R := QQ[x,y,z];
/**/ S := QQ[x[1..3]];
/**/ phi := PolyAlgebraHom(R, S, indets(S));
/**/ phi([x^2-y, z-2]);          -- was apply(phi, [x^2-y, z-2]) up to 5.3.3
[x[1]^2 -x[2], x[3] -2]
/**/ phi(mat([[x,1], [y,z]])); -- was apply(phi, mat(...)) up to 5.3.3
```

I-1.16 ApproxPointsNBM

syntax

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

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.

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.147), StableBBasis5(I-19.38 pg.312)

I-1.17 ApproxSolve

syntax

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

This function returns approximations to the real solutions (points) of a 0-dimensional polynomial system “L”: these approximations are given as a LIST of LISTS of rationals in the field “AffinePts”; the field “indets” gives the indets corresponding to the positions in the coordinates of each solution.

The polynomials in “L” must have rational coefficients. Approximate coordinates are given for non-rational solutions. A heuristic is used to determine the precision (which may vary depending on each solution point).

Useful verbosity range 20–20.

See also “RationalSolve” (I-18.17 pg.276) which finds all solutions all of whose coordinates are rational.

NOTE: up to version 5.3.2 the output was just the LIST of solutions.

example

```

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

/**/ L := [x^3-y^2+z-1, x^2-2, (y-3)*(y+2)];
/**/ AS := ApproxSolve(L);
--> use FloatStr or DecimalStr to make result more readable
/**/ indent([[ FloatStr(coord) | coord in pt ] | pt in AS.AffinePts]);
[
  ["1.4142", "-2.0000", "2.1716"],
  ["-1.4142", "-2.0000", "7.8284"],
  ["1.4142", "3.0000", "7.1716"],
  ["-1.4142", "3.0000", "12.828"]
]

-- Verify we have an approximate answer:
/**/ indent([ [ FloatStr(eval(f, pt)) | f in L ] | pt in AS.AffinePts]);
[
  ["7.1576*10^(-19)", "5.0612*10^(-19)", "0.0000"],
  ["-7.1576*10^(-19)", "5.0612*10^(-19)", "0.0000"],
  ["2.8208*10^(-19)", "1.9946*10^(-19)", "0.0000"],
  ["-2.8208*10^(-19)", "1.9946*10^(-19)", "0.0000"]
]

```

See Also: LinSolve(I-12.17 pg.199), RationalSolve(I-18.17 pg.276)

I-1.18 AreGensMonomial

syntax

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

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\(I-1.19 pg.38\)](#), [IsTerm\(I-9.103 pg.183\)](#), [HasGBasis\(I-8.2 pg.135\)](#)

I-1.19 AreGensSqFreeMonomial

syntax

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

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\(I-1.18 pg.37\)](#), [IsSqFree\(I-9.95 pg.180\)](#), [HasGBasis\(I-8.2 pg.135\)](#)

I-1.20 ArrBettiNumbers

syntax

```
ArrBettiNumbers(A: LIST): LIST
```

This function returns the list “[b₀, b₁, ..., b_l]” of the Betti numbers from the list “A” of hyperplanes in the arrangement.

example

```

/**/ use QQ[x,y];
/**/ A := [x, x-y, y];
/**/ ArrBettiNumbers(A);
[1, 3, 2]

```

I-1.21 ArrBoolean

syntax

```
ArrBoolean(S: RING, k: INT): LIST
```

This function constructs the boolean arrangement in the first “k” variables of the polynomial ring “S”.

example

```

/**/ use S:=QQ[x,y,z];
/**/ ArrBoolean(S, 3);
[x, y, z]

```

I-1.22 ArrBraid

— syntax —

```
ArrBraid(S: RING, k: INT): LIST
```

This function constructs the braid arrangement in the first k variables of the polynomial ring S .

— example —

```
/**/ use S:=QQ[x,y,z];
/**/ ArrBraid(S, 3);
[x -y,  x -z,  y -z]
```

I-1.23 ArrCatalanA

— syntax —

```
ArrCatalanA(S: RING, k: INT): LIST
```

This function constructs the Catalan arrangement of type A in the first k variables of the polynomial ring S .

— example —

```
/**/ use S:=QQ[x,y,z];
/**/ ArrCatalanA(S, 2);
[x -y,  x -y -1,  x -y +1]
```

I-1.24 ArrCatalanB

— syntax —

```
ArrCatalanB(S: RING, k: INT): LIST
```

This function constructs the Catalan arrangement of type B in the first k variables of the polynomial ring S .

— example —

```
/**/ use S:=QQ[x,y,z];
/**/ ArrCatalanB(S, 2);
[x,  y,  x -y,  x +y,  x -1,  y -1,  x -y -1,  x +y -1,  x +1,  y +1,  x -y +1,  x +y +1]
```

I-1.25 ArrCatalanD

— syntax —

```
ArrCatalanD(S: RING, k: INT): LIST
```

This function constructs the Catalan arrangement of type D in the first k variables of the polynomial ring S .

— example —

```
/**/ use S:=QQ[x,y,z];
/**/ ArrCatalanD(S, 2);
[x -y,  x +y,  x -y -1,  x +y -1,  x -y +1,  x +y +1]
```

I-1.26 ArrCharPoly

— syntax —

```
ArrCharPoly(A: LIST): RINGELEM
```

This function returns the characteristic polynomial from the list A of hyperplanes in the arrangement in the variable t .

example

```

/**/ use QQ[x,y];
/**/ A := [x, x-y, y];
/**/ ArrCharPoly(A);
t^2 -3*t +2
/**/ RingOf(ArrCharPoly(A));
RingWithID(4, "QQ[t]")

/**/ use ZZ/(5)[x,y];
/**/ A := [x, x-y, y];
/**/ ArrCharPoly(A);
t^2 -3*t +2
/**/ RingOf(ArrCharPoly(A));
RingWithID(4, "QQ[t]")

```

See Also: [ArrPoincarePoly\(I-1.35 pg.42\)](#), [PosetCharPoly\(I-16.19 pg.252\)](#)

I-1.27 ArrCone

syntax

```
ArrCone(A: LIST, t: RINGELEM): LIST
```

This function returns the list of hyperplanes of the cone of the list “A” of hyperplanes of an arrangement with respect to the indeterminate “t”.

example

```

/**/ use QQ[x,y,t];
/**/ A := [x, x-1, y];
/**/ ArrCone(A, t);
[x, x -t, y, t]

```

I-1.28 ArrDeletion

syntax

```
ArrDeletion(A: LIST, H: RINGELEM): LIST
```

This function returns the list of hyperplanes obtained by deleting the hyperplane “H” from the list “A” of hyperplanes.

example

```

/**/ use QQ[x,y];
/**/ A := [x, x-1, y];
/**/ ArrDeletion(A, x-1);
[x, y]

```

See Also: [ArrRestriction\(I-1.36 pg.42\)](#)

I-1.29 ArrDerMod [OBSOLESCENT]

Renamed to “ArrDerModule” ([I-1.30 pg.40](#)).

I-1.30 ArrDerModule

syntax

```
ArrDerModule(Q: RINGELEM): MAT
```


This function returns the matrix whose columns are a set of generators of the module of logarithmic derivations of an arrangement of hyperplanes described by its defining equation Q .

example

```

/**/ use QQ[x,y];
/**/ A := [x, x-y, y];
/**/ Q_A := product(A);
/**/ ArrDerModule(Q_A);
matrix( /*RingWithID(70, "QQ[x,y]")*/
  [[x, 0],
   [y, x*y -y^2]])

/**/ use QQ[x,y,z];
/**/ A := [x, x+z, y, 2*y-3*z];
/**/ ArrDerModule(product(A));
matrix( /*RingWithID(78, "QQ[x,y,z]")*/
  [[x, 0, 0, 0],
   [y, y^2 +(-3/2)*y*z, 0, x*y +y*z],
   [z, 0, x*y +(-3/2)*x*z +y*z +(-3/2)*z^2, x*z +z^2]])

```

See Also: ArrExponents(I-1.31 pg.41), IsArrFree(I-9.41 pg.162)

I-1.31 ArrExponents

syntax

```
ArrExponents(Q: RINGELEM): LIST
```

This function returns the list of exponents of a free arrangement of hyperplanes from its defining equation Q .

example

```

/**/ use QQ[x,y];
/**/ A := [x, x-y, y]; -- free
/**/ Q_A := product(A);
/**/ ArrExponents(Q_A);
[1, 2]

/**/ use QQ[x,y,z];
/**/ A := [x, x+z, y, 2*y-3*z]; -- not free
/**/ IsArrFree(product(A)); --> false
-- /**/ ArrExponents(product(A)); --> !!! ERROR !!! as expected, not free

```

See Also: ArrDerModule(I-1.30 pg.40), IsArrFree(I-9.41 pg.162)

I-1.32 ArrFlats

syntax

```
ArrFlats(A: LIST): LIST
```

This function returns the list of flats from the list “A” of hyperplanes in the arrangement.

example

```

/**/ use QQ[x,y];
/**/ A := [x, x-y, y];
/**/ ArrFlats(A);
[[ideal(0)], [ideal(x), ideal(x -y), ideal(y)], [ideal(x, y)]]

```

See Also: ArrLattice(I-1.34 pg.42)

I-1.33 ArrGraphical

syntax

```
ArrGraphical(S: RING, edgesG: LIST): LIST
```

This function constructs the graphical arrangement with respect to the list of edges edgesG of the graph G in the polynomial ring S.

example

```
/**/ use S:=QQ[x,y,z];
/**/ ArrGraphical(S, [[1,2],[1,3]]);
[x -y, x -z]

/**/ use S:=QQ[x[1..3]];
/**/ ArrGraphical(S, [[1,2],[1,3]]);
[x[1] -x[2], x[1] -x[3]]
```

I-1.34 ArrLattice

syntax

```
ArrLattice(A: LIST): LIST of LIST
```

This function returns the list of relations of the lattice from the list A of hyperplanes in the arrangement.

example

```
/**/ use QQ[x,y];
/**/ A := [x, x-y, y];
/**/ ArrLattice(A);
[[1, 2], [1, 3], [1, 4], [2, 5], [3, 5], [4, 5]]
```

See Also: ArrFlats([I-1.32](#) pg.41)

I-1.35 ArrPoincarePoly

syntax

```
ArrPoincarePoly(A: LIST): RINGELEM
```

This function returns the Poincare polynomial from the list A of hyperplanes in the arrangement in the variable t.

example

```
/**/ use QQ[x,y];
/**/ A := [x, x-y, y];
/**/ ArrPoincarePoly(A);
2*t^2 +3*t +1
```

See Also: PosetPoincarePoly([I-16.24](#) pg.254), ArrCharPoly([I-1.26](#) pg.39)

I-1.36 ArrRestriction

syntax

```
ArrRestriction(A: LIST, H: RINGELEM): LIST
```

This function returns the list of hyperplanes in the variables $[y[1], \dots, y[n]]$ obtained by restricting to the hyperplane H of the list A of hyperplanes of an arrangement.

example

```

/**/ use QQ[x,y,z];
/**/ A := [x, x-z, y-z, z];
/**/ ArrRestriction(A, y-z);
[y[1], y[1] -y[2], y[2]]

```

See Also: MultiArrRestrictionZiegler([I-13.42](#) pg.220)

I-1.37 ArrShiA

syntax

```
ArrShiA(S: RING, k: INT): LIST
```

This function constructs the Shi arrangement of type A in the first k variables of the polynomial ring S.

example

```

/**/ use S:=QQ[x,y,z];
/**/ ArrShiA(S, 3);
[x -y, x -z, y -z, x -y -1, x -z -1, y -z -1]

```

I-1.38 ArrShiB

syntax

```
ArrShiB(S: RING, k: INT): LIST
```

This function constructs the Shi arrangement of type B in the first k variables of the polynomial ring S.

example

```

/**/ use S:=QQ[x,y,z];
/**/ ArrShiB(S, 2);
[x, y, x -y, x +y, x -1, y -1, x -y -1, x +y -1]

```

I-1.39 ArrShiCatalanA

syntax

```
ArrShiCatalanA(S: RING, k: INT, L: LIST): LIST
```

This function constructs the Shi-Catalan arrangement of type A in the first k variables of the polynomial ring S with multiplicities L[1] and L[2].

example

```

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

```

I-1.40 ArrShiCatalanB

syntax

```
ArrShiCatalanB(S: RING, k: INT, L: LIST): LIST
```

This function constructs the Shi-Catalan arrangement of type B in the first k variables of the polynomial ring S with multiplicities L[1] and L[2].

example

```

/**/ use S:=QQ[x,y,z];
/**/ L:=[-1,2];
/**/ ArrShiCatalanB(S, 2, L);
[x, y, x -y, x +y, x -1, y -1, x -y -1, x +y -1, x +1, x +2, y +1, y +2, x -y +1, x -y +2]

```

I-1.41 ArrShiCatalanD

syntax

```
ArrShiCatalanD(S: RING, k: INT, L: LIST): LIST
```

This function constructs the Shi-Catalan arrangement of type D in the first k variables of the polynomial ring S with multiplicities L[1] and L[2].

example

```

/**/ use S:=QQ[x,y,z];
/**/ L:=[-1,2];
/**/ ArrShiCatalanD(S, 2, L);
[x -y, x +y, x -y -1, x +y -1, x -y +1, x -y +2, x +y +1, x +y +2]

```

I-1.42 ArrShiD

syntax

```
ArrShiD(S: RING, k: INT): LIST
```

This function constructs the Shi arrangement of type D in the first k variables of the polynomial ring S.

example

```

/**/ use S:=QQ[x,y,z];
/**/ ArrShiD(S, 2);
[x -y, x +y, x -y -1, x +y -1]

```

I-1.43 ArrSignedGraphical

syntax

```
ArrSignedGraphical(S: RING, PositiveEdgesG: LIST, NegativeEdgesG: LIST, loopsG: LIST): LIST
```

This function constructs the signed graphical arrangement with respect to the list of positive edges “PositiveEdgesG”, of negative edges “NegativeEdgesG” and of loops “loopsG” of the signed graph “G” in the polynomial ring “S”.

example

```

/**/ use S:=QQ[x,y,z];
/**/ ArrSignedGraphical(S, [[1,2],[1,3]],[[2,3]],[1,3]);
[x -y, x -z, y +z, x, z]

/**/ use S:=QQ[x[1..3]];
/**/ ArrSignedGraphical(S, [[1,2],[1,3]],[[2,3]],[1,3]);
[x[1] -x[2], x[1] -x[3], x[2] +x[3], x[1], x[3]]

```

I-1.44 ArrToMultiArr

syntax

```
ArrToMultiArr(A: LIST, L: LIST): LIST
```

This function constructs the multiarrangement obtained from the arrangement A with respect to the list of multiplicities L.

example

```
/**/ use QQ[x,y,z];
/**/ A := [x, y, z];
/**/ ArrToMultiArr(A, [1, 3, 2]);
[[x, 1], [y, 3], [z, 2]]
```

See Also: MultiArrToArr([I-13.43](#) pg.221), MultiArrRestrictionZiegler([I-13.42](#) pg.220)

I-1.45 ArrTuttePoly

syntax

```
ArrTuttePoly(A: LIST): RINGELEM
```

This function returns the Tutte polynomial from the list A of hyperplanes in the arrangement in the ring $\mathbb{Q}\mathbb{Q}[t[1], t[2]]$.

example

```
/**/ use QQ[x,y];
/**/ A := [x, x-y, y];
/**/ ArrTuttePoly(A);
t[1]^2 +t[1] +t[2]
```

See Also: ArrPoincarePoly([I-1.35](#) pg.42), ArrCharPoly([I-1.26](#) pg.39)

I-1.46 ArrTypeB

syntax

```
ArrTypeB(S: RING, k: INT): LIST
```

This function constructs the reflection arrangement of type B in the first k variables of the polynomial ring S.

example

```
/**/ use S:=QQ[x,y,z];
/**/ ArrTypeB(S, 3);
[x, y, z, x -y, x +y, x -z, x +z, y -z, y +z]
```

I-1.47 ArrTypeD

syntax

```
ArrTypeD(S: RING, k: INT): LIST
```

This function constructs the reflection arrangement of type D in the first k variables of the polynomial ring S.

example

```
/**/ use S:=QQ[x,y,z];
/**/ ArrTypeD(S, 3);
[x -y, x +y, x -z, x +z, y -z, y +z]
```

I-1.48 ArtinianOrlikTeraoIdeal

syntax

```
ArtinianOrlikTeraoIdeal(A: LIST): IDEAL
```

This function returns the artinian Orlik-Terao ideal of the list A of hyperplanes of an arrangement.

example

```
/**/ use QQ[x,y];
/**/ A := [x, y, x-y];
/**/ ArtinianOrlikTeraoIdeal(A);
ideal(y[1]*y[2] +y[1]*y[3] -y[2]*y[3], y[1]^2, y[2]^2, y[3]^2)
```

See Also: [SolomonTeraoIdeal\(I-19.27 pg.308\)](#), [OrlikTeraoIdeal\(I-15.12 pg.246\)](#)

I-1.49 ascii

syntax

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

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-1.50 AsINT

syntax

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

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\(I-1.51 pg.47\)](#)

I-1.51 AsRAT

— syntax —

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

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\(I-1.50 pg.46\)](#)

I-1.52 assert

— syntax —

```
assert(cond: BOOL)
```

This function **always evaluates** its argument. If the argument is not “**true**” then an exception is thrown. This function is intended for use only during development, and should not be used in released code.

— example —

```
/**/ define fn(X)
/**/   assert(X > 0);
/**/   return FloorSqrt(X);
/**/ enddefine;

/**/ fn(3);
1
-- /**/ fn(-3); --> !!! ERROR !!! "assertion failed", as expected
```

See Also: [error\(I-5.12 pg.99\)](#)

Chapter I-2

B

I-2.1 BaseRing

— syntax —

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

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”, if “K” was constructed as a quotient “R/I” then “BaseRing(K)” produces “R” (see also “DefiningIdeal” (I-4.5 pg.85)).

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.223), NewQuotientRing(I-14.11 pg.227), DefiningIdeal(I-4.5 pg.85), NewPolyRing(I-14.9 pg.226)

I-2.2 BBasis5

— syntax —

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

***** 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);
***** NOT YET IMPLEMENTED *****
BBasis := BBasis5(I);
```

I-2.3 BettiDiagram

— syntax —

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

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\(I-2.4 pg.50\)](#), [PrintRes\(I-16.46 pg.263\)](#), [PrintBettiDiagram\(I-16.42 pg.261\)](#), [PrintBettiMatrix\(I-16.43 pg.262\)](#)

I-2.4 BettiMatrix

syntax

```
BettiMatrix(M: IDEAL|MODULE|LISTResolution)
```

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\(I-16.46 pg.263\)](#), [PrintBettiDiagram\(I-16.42 pg.261\)](#)

I-2.5 BettiNumbers

syntax

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

This function returns the Betti numbers for “M”.

example

```
/**/ M := MakeTermOrdMat(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
-- Betti 0 -----
-- Betti 1 -----
[18, 2]: 1
[30, 6]: 2
-- Betti 2 -----
[48, 8]: 2
[60, 12]: 1
-- Betti 3 -----
[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\(I-16.46 pg.263\)](#), [BettiDiagram\(I-2.3 pg.49\)](#), [BettiMatrix\(I-2.4 pg.50\)](#), [PrintBettiNumbers\(I-16.44 pg.262\)](#)

I-2.6 binomial

syntax

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

This function computes the binomial coefficient, **N choose K** according to the formula $(N)(N-1)(N-2)\dots(N-K+1)/K!$ If “K < 0” or “K > abs(N)” the value is 0.

The same formula is used if “N” is a ring element; in this case it is an error if the integer “K” is negative. See the example below, to see how to compute “binomial(N,K)” where “N” is a rational.

example

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

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

/**/ N := 5/3; // want to compute binomial(N,2)
/**/ AsRAT(binomial(RingElem(QQ,N), 2)); // trick: convert N to elem of QQ
5/9
```

```

/**/ 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.52)

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.

```

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) | t in 1..K]” where $N(t)=0$ for all $t < L$.

The function call “BinomialReprShift(N,K,up,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.51)

I-2.8 block

— syntax —

```

block C_1; ... ; C_n EndBlock;

where each C_i is a command.

```

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_i”. 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
```

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 lists: the total number of columns must be the same.

The function “BlockMat2x2” (I-2.10 pg.53) 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-3.44 pg.72), ConcatVer(I-3.48 pg.74), ConcatHorList(I-3.45 pg.73), ConcatVerList(I-3.49 pg.74), ConcatDiag(I-3.43 pg.72), ConcatAntiDiag(I-3.42 pg.72), BlockMat2x2(I-2.10 pg.53)

I-2.10 BlockMat2x2

syntax

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

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

$$\begin{array}{|c|c|c|} \hline & A & B \\ \hline C & D & \\ \hline \end{array}$$

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.53) 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-3.44 pg.72), ConcatVer(I-3.48 pg.74), ConcatDiag(I-3.43 pg.72), ConcatAntiDiag(I-3.42 pg.72), BlockMat(I-2.9 pg.53)

I-2.11 Bool01

syntax

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

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
```

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-3.56 pg.77).

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\(I-3.56 pg.77\)](#), [return\(I-18.43 pg.287\)](#), All CoCoA commands([II-2.2 pg.357](#))

I-2.13 BringIn

— syntax —

```

BringIn(E: OBJECT): OBJECT
BringIn(R: RING, E: OBJECT): OBJECT

```

This function maps a polynomial (or a list, matrix of these) into the current ring, or the ring “R”, preserving the names of the indeterminates. Honestly, this is not-so-clean shortcut for creating and calling a homomorphism. (“Introduction to RINGHOM” ([III-10.1 pg.427](#)))

NOTE: this function is not implemented on IDEAL because might be misleading: one might expect that bringing an ideal from “K[x,y]” into “K[x]” means eliminating “y”. For this operation call “elim” ([I-5.5 pg.96](#)). Instead, if you want to map the generators of the ideal type “ideal(BringIn(R, 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
/**/  BringIn(SS, 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: [Introduction to RINGHOM\(III-10.1 pg.427\)](#), [PolyAlgebraHom\(I-16.17 pg.251\)](#), [QZP\(I-17.7 pg.269\)](#), [ZPQ\(I-25.3 pg.346\)](#)

Chapter I-3

C

I-3.1 Call [OBSOLETE]

[OBSOLETE] in CoCoA-5 functions can be used directly. See “FUNCTIONs are first class objects” ([III-7.2](#) pg.417).

I-3.2 CallOnGroebnerFanIdeals

syntax

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

Storing all the possible different (reduced) GBases in a Groebner fan is practicable only for small examples; 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-7.36](#) pg.132).

NOTE: using this needs a little technical ability, but might make the difference between getting an answer or filling up the RAM.

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-7.36 pg.132), OrdMat(I-15.11 pg.245), RingOf(I-18.51 pg.290)

I-3.3 CanonicalBasis

— syntax —

```
CanonicalBasis(F: MODULE): LIST of MODULEELEM
```

“CanonicalBasis(F)” return a list of the canonical basis elements of the free module “F”: *i.e.* the k-th element in the list is the unit vector with a 1 in the k-th coordinate.

— example —

```

/**/ use R := QQ[x,y];
/**/ F := NewFreeModule(R,2);
/**/ e := CanonicalBasis(F);
/**/ e[2];
[0, 1]

```

See Also: NewFreeModule(I-14.2 pg.223), gens(I-7.8 pg.124), GensAsCols, GensAsRows(I-7.9 pg.125)

I-3.4 CanonicalHom

— syntax —

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

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 -> FractionFiels(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
(x*y)

```

See Also: NewFractionField([I-14.1](#) pg.223), NewQuotientRing([I-14.11](#) pg.227), NewPolyRing([I-14.9](#) pg.226), CanonicalHom([I-3.4](#) pg.58), CoeffEmbeddingHom([I-3.25](#) pg.65), QuotientingHom([I-17.6](#) pg.269), PolyAlgebraHom([I-16.17](#) pg.251), PolyRingHom([I-16.18](#) pg.252)

I-3.5 CanonicalRepr

— syntax —

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

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-14.11](#) pg.227), DefiningIdeal([I-4.5](#) pg.85)

I-3.6 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
```

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.332).

— 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.
```

Be careful: in CoCoA the operator “><” denotes cartesian product, whereas the operator “<>” is used for **not equal**.

See Also: CoCoA Operators: introduction([II-4.1](#) pg.361), operators, shortcuts([I-0.1](#) pg.29), tuples([I-20.15](#) pg.332), Equality Operator([I-5.10](#) pg.98)

I-3.7 Cast [OBSOLETE]

This old function is now obsolete. These are the functions for casting:

To cast INT, RAT, STRING to a polynomial (and more in general to a RINGELEM) use “RingElem” (I-18.47 pg.288).

To cast RINGELEM to INT, RAT use “AsINT” (I-1.50 pg.46), “AsRAT” (I-1.51 pg.47).

To cast LIST to MAT use “matrix” (I-13.11 pg.209). To cast MAT to LIST use “GetRows” (I-7.18 pg.127), “GetCols” (I-7.13 pg.126).

To cast a MODULEELEM to LIST use “compts” (I-3.39 pg.70).

To cast a MODULE to MAT use “GensAsCols, GensAsRows” (I-7.9 pg.125). To cast a MAT to MODULE use “SubmoduleCols, SubmoduleRows” (I-19.56 pg.319).

See Also: AsINT(I-1.50 pg.46), AsRAT(I-1.51 pg.47), gens(I-7.8 pg.124), GensAsCols, GensAsRows(I-7.9 pg.125), GetCols(I-7.13 pg.126), GetRows(I-7.18 pg.127), ideal(I-9.2 pg.143), matrix(I-13.11 pg.209), ModuleElem(I-13.31 pg.216), RingElem(I-18.47 pg.288), SubmoduleCols, SubmoduleRows(I-19.56 pg.319)

I-3.8 CatalanNumber

syntax

```
CatalanNumber(N: INT): INT
```

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

example

```
/**/ CatalanNumber(3);
5
```

I-3.9 ceil

syntax

```
ceil(X: RAT): INT
```

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-6.17 pg.111), round(I-18.61 pg.294), num(I-14.35 pg.236), den(I-4.7 pg.86), div(I-4.22 pg.92)

I-3.10 CFApprox

syntax

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

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

example

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

See Also: CFAproximants(I-3.11 pg.61), ContFrac(I-3.54 pg.76), SimplestRatBetween(I-19.18 pg.305)

I-3.11 CFAproximants

syntax

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

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

example

```
/**/ CFAproximants(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: CFAprox(I-3.10 pg.60), ContFrac(I-3.54 pg.76)

I-3.12 ChainCanonicalHom

syntax

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

This function returns the canonical homomorphism from “R” to “S”, where “S” has been constructed from “R” with a chain of ring constructors.

example

```
/**/ use P ::= QQ[i];
/**/ K := NewQuotientRing(P, ideal(i^2+1));
/**/ R ::= K[x,y,z];

/**/ phi := ChainCanonicalHom(P, R);
/**/ RingElem(R, "x^2-i*y") / phi(RingElem(P, "(3*i-2)*(5-i)"));
((-17/338)*i -7/338)*x^2 +((7/338)*i -17/338)*y
```

See Also: CanonicalHom(I-3.4 pg.58), RingElem(I-18.47 pg.288)

I-3.13 characteristic

syntax

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

This function returns the characteristic of the ring “R”.

example

```
/**/ use R ::= ZZ/(3)[t];
/**/ S ::= QQ[x,y];
/**/ characteristic(CurrentRing);
3
/**/ S ::= QQ[x,y];
/**/ characteristic(S);
0
```

See Also: [IsFiniteField\(I-9.56 pg.167\)](#), [LogCardinality\(I-12.20 pg.200\)](#)

I-3.14 CharPoly

syntax

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

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

See also “MinPoly” ([I-13.25 pg.213](#)).

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\(I-13.25 pg.213\)](#)

I-3.15 ChebyshevPoly

syntax

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

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\(I-8.4 pg.136\)](#), [LaguerrePoly\(I-12.1 pg.193\)](#)

I-3.16 CheckArgTypes

syntax

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

This function provides a basic type checking for user defined functions: it checks whether the **TYPE**s 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 !!! as expected: 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: Arg 2 is RINGELEM but must be INT

```

I-3.17 *ciao*

syntax

```
ciao
```

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

See Also: [exit\(I-5.17 pg.101\)](#), [quit\(I-17.3 pg.268\)](#)

I-3.18 *ClearDenom*

syntax

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

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

```

See Also: [CommonDenom\(I-3.36 pg.70\)](#), [content\(I-3.51 pg.75\)](#), [prim\(I-16.32 pg.257\)](#)

I-3.19 *close*

syntax

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

This function closes the output stream “OUT”. If “OUT” refers to a file, this will **flush** any buffered output (no value is returned). If “OUT” came from “OpenOString” ([I-15.6 pg.243](#)), this will **return** the string which has been **printed** into the stream.

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

/**/  S := OpenOString();
/**/  print "abc", 123 on S;
/**/  close(S);
abc123

```

See Also: [Introduction to IO\(II-8.1 pg.371\)](#)

I-3.20 CloseLog

syntax

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

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

This function “OpenLog” ([I-15.4 pg.242](#)) 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\(I-15.4 pg.242\)](#)

I-3.21 CoCoA-4 mode

syntax

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

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

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\(II-11.1 pg.383\)](#), [not\(I-14.33 pg.235\)](#), [and\(I-1.13 pg.35\)](#), or [\(I-15.9 pg.244\)](#)

I-3.22 CocoaLimits

syntax

```
CocoaLimits(): RECORD
```

***** 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

```
***** NOT YET IMPLEMENTED *****
CocoaLimits();
record[MaxChar := 32767, MaxExp := 2147483647]
-----
```

I-3.23 CocoaPackagePath

syntax

```
CocoaPackagePath(): STRING
```


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

example

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

I-3.24 codomain

syntax

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

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-3.24](#) pg.65), Commands and Functions for `RINGHOM`([III-10.3](#) pg.428), Commands and Functions returning `RINGHOM`([III-10.4](#) pg.428)

I-3.25 CoeffEmbeddingHom

syntax

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

This function returns the coefficient embedding homomorphism of the polynomial ring “P”. “ConstantCoeff” ([I-3.50](#) pg.75) is a sort of inverse.

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: `ConstantCoeff`([I-3.50](#) pg.75), `CanonicalHom`([I-3.4](#) pg.58)

I-3.26 CoeffHeight

syntax

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

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\(I-18.59 pg.293\)](#)

I-3.27 coefficients

syntax

```

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

```

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” with respect to (WRT) an indeterminate “x” is now replaced by the functions “`CoefficientsWRT`” ([I-3.28 pg.67](#)) and “`CoeffListWRT`” ([I-3.29 pg.67](#)).

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(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(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.422), CoefficientsWRT([I-3.28](#) pg.67), CoeffListWRT([I-3.29](#) pg.67), LC([I-12.5](#) pg.194), monomials([I-13.36](#) pg.218), support([I-19.62](#) pg.322)

I-3.28 CoefficientsWRT

syntax

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

The first function returns the list of the coefficients and power-products of “F” seen as a polynomial in “X” (with respect to (WRT) “X”); the second function does the same but viewing “F” as a polynomial in all the indeterminates in the set “S”.

NOTE: 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.422), CoeffListWRT([I-3.29](#) pg.67), coefficients([I-3.27](#) pg.66), CoefOfTerm([I-3.31](#) pg.68), ContentWRT([I-3.53](#) pg.76), LC([I-12.5](#) pg.194), monomials([I-13.36](#) pg.218), support([I-19.62](#) pg.322)

I-3.29 CoeffListWRT

syntax

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

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

NOTE: 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\(I-3.27 pg.66\)](#), [CoefficientsWRT\(I-3.28 pg.67\)](#)

I-3.30 CoeffListWRTSupport

syntax

```
CoeffListWRTSupport(f: RINGELEM, basis: RINGELEM): LIST of RINGELEM
```

This function returns the list of the coefficients of “f” relative to the basis of power-products given by the support of “basis”. The elements of the list belong to the coefficient ring of the ring of “f”.

NOTE: the returned list is **reversed** with respect to the order of the power-products in “basis”.

example

```

/**/ use R ::= QQ[x,y,z];
/**/ f := 5*y^2 + (3*x-1)*y;
/**/ basis := (1+x+y)^2;
/**/ CoeffListWRTSupport(f, basis);
[0, -1, 0, 5, 3, 0]

```

See Also: [coefficients\(I-3.27 pg.66\)](#), [CoeffListWRT\(I-3.29 pg.67\)](#), [support\(I-19.62 pg.322\)](#)

I-3.31 CoeffOfTerm

syntax

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

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\(I-3.27 pg.66\)](#), [CoefficientsWRT\(I-3.28 pg.67\)](#), [CoeffListWRT\(I-3.29 pg.67\)](#), [LC\(I-12.5 pg.194\)](#), [exponents\(I-5.18 pg.101\)](#), [MakeTerm\(I-13.4 pg.206\)](#), [monomials\(I-13.36 pg.218\)](#), [support\(I-19.62 pg.322\)](#)

I-3.32 CoeffRing

syntax

```
CoeffRing(R: RING): RING
```

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\(I-3.13 pg.61\)](#), [coefficients\(I-3.27 pg.66\)](#), [CurrentRing\(I-3.64 pg.80\)](#), [indets\(I-9.24 pg.155\)](#)

I-3.33 ColMat

syntax

```

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

```

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\(I-13.11 pg.209\)](#), [BlockMat\(I-2.9 pg.53\)](#), [DiagMat\(I-4.16 pg.90\)](#), [RowMat\(I-18.62 pg.294\)](#), [GensAsCols](#), [GensAsRows\(I-7.9 pg.125\)](#)

I-3.34 colon

syntax

```

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

```

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

See also “HColon” ([I-8.3 pg.136](#)) 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\(I-19.3 pg.298\)](#), [HSaturation\(I-8.17 pg.142\)](#), [HColon\(I-8.3 pg.136\)](#)

I-3.35 ColumnVectors [OBSOLETE]

Essentially replaced by “GensAsCols, GensAsRows” ([I-7.9 pg.125](#)) and “SubmoduleCols, SubmoduleRows” ([I-19.56 pg.319](#))

I-3.36 CommonDenom

syntax

```

CommonDenom(f: RINGELEM): RINGELEM
CommonDenom(L: LIST of RINGELEM): RINGELEM

```

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
/**/ CommonDenom([2*x, y/3]);
3

```

See Also: [ClearDenom\(I-3.18 pg.63\)](#), [content\(I-3.51 pg.75\)](#), [ContentWRT\(I-3.53 pg.76\)](#), [prim\(I-16.32 pg.257\)](#)

I-3.37 Comp [OBSOLETE]

Please use the indexing operator “[...]” for accessing entries in a list by index, or to select fields from a record.

See Also: [operators, shortcuts\(I-0.1 pg.29\)](#), [record field selector\(I-18.28 pg.281\)](#)

I-3.38 CompleteToOrd [OBSOLETE]

Renamed to “MakeTermOrdMat” ([I-13.6 pg.206](#)).

I-3.39 compts

syntax

```

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

```

This function returns the list of components of the ModuleElem “V”. It is like converting the “ModuleElem” into a generic list.

NOTE: 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-14.39 pg.237)

I-3.40 ComputeElimFirst

— syntax —

```

ComputeElimFirst(X: RINGELEM, I: IDEAL): RINGELEM

```

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-5.5 pg.96)

I-3.41 concat

— syntax —

```

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

```

This function returns the list obtained by concatenating the lists “L₁, ..., L_n”.

NOTE: to concatenate strings use “ConcatStrings” (I-3.47 pg.74) or “str1+str2”.

— example —

```

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

```

See Also: append(I-1.14 pg.35), ConcatLists(I-3.46 pg.73), ConcatStrings(I-3.47 pg.74), String Operations(III-4.2 pg.401)

I-3.42 ConcatAntiDiag

syntax

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

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.53), `ConcatDiag`([I-3.43](#) pg.72), `ConcatHor`([I-3.44](#) pg.72), `ConcatVer`([I-3.48](#) pg.74)

I-3.43 ConcatDiag

syntax

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

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.53), `ConcatAntiDiag`([I-3.42](#) pg.72), `ConcatHor`([I-3.44](#) pg.72), `ConcatVer`([I-3.48](#) pg.74), `DiagMat`([I-4.16](#) pg.90)

I-3.44 ConcatHor

syntax

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

where A and B have the same number of rows

This function creates a horizontally stacked 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.53), MakeMatByRows, MakeMatByCols(I-13.2 pg.205), ConcatAntiDiag(I-3.42 pg.72), ConcatDiag(I-3.43 pg.72), ConcatHorList(I-3.45 pg.73), ConcatVer(I-3.48 pg.74), RowMat(I-18.62 pg.294)

I-3.45 ConcatHorList

syntax

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

where the matrices in L have the same number of rows

This function creates a horizontally stacked 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.53), MakeMatByRows, MakeMatByCols(I-13.2 pg.205), ConcatAntiDiag(I-3.42 pg.72), ConcatDiag(I-3.43 pg.72), ConcatHor(I-3.44 pg.72), ConcatVerList(I-3.49 pg.74), RowMat(I-18.62 pg.294)

I-3.46 ConcatLists

syntax

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

This function takes one 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-1.14 pg.35), concat(I-3.41 pg.71), flatten(I-6.14 pg.110)

I-3.47 ConcatStrings

syntax

```
ConcatStrings(L: LIST of STRING): STRING
```

This function returns the string obtained by concatenating the strings in “L”. This function is faster than “concat” (I-3.41 pg.71).

NOTE: to concatenate 2 strings use “str1 + str2”.

example

```
/**/ ConcatStrings(["abc", "def", "ghi"]);
abcdefghi
```

See Also: concat(I-3.41 pg.71), FoldToListInput(I-6.22 pg.113), String Operations(III-4.2 pg.401)

I-3.48 ConcatVer

syntax

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

where A and B have the same number of columns

This function creates a vertically stacked 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([[11,12,13]]);
/**/ ConcatVer(A, B);
matrix(QQ,
  [[1, 2, 3],
   [4, 5, 6],
   [11, 12, 13]])
```

See Also: BlockMat(I-2.9 pg.53), ColMat(I-3.33 pg.69), MakeMatByRows, MakeMatByCols(I-13.2 pg.205), ConcatAntiDiag(I-3.42 pg.72), ConcatDiag(I-3.43 pg.72), ConcatHor(I-3.44 pg.72), ConcatVerList(I-3.49 pg.74)

I-3.49 ConcatVerList

syntax

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

where the matrices in L have the same number of columns

This function creates a vertically stacked block matrix. The entries in the list are matrices with the same number of columns. “ConcatVerList(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([[11,12,13]]) ];
/**/ ConcatVerList(L);
matrix(QQ,
  [[1, 2, 3],
   [4, 5, 6],
   [11, 12, 13]])

```

See Also: BlockMat([I-2.9](#) pg.53), ColMat([I-3.33](#) pg.69), MakeMatByRows, MakeMatByCols([I-13.2](#) pg.205), ConcatAntiDiag([I-3.42](#) pg.72), ConcatDiag([I-3.43](#) pg.72), ConcatVer([I-3.48](#) pg.74), ConcatHorList([I-3.45](#) pg.73)

I-3.50 ConstantCoeff

syntax

```
ConstantCoeff(F: RINGELEM): RINGELEM
```

This function returns the constant coefficient of a polynomial. The result is in the coefficient ring, and may be zero.

example

```

/**/ use QQ[x,y];
/**/ f := x^3+3*x*y-4*y+5;
/**/ ConstantCoeff(f);
5

```

See Also: CoeffEmbeddingHom([I-3.25](#) pg.65), coefficients([I-3.27](#) pg.66), CoefficientsWRT([I-3.28](#) pg.67), CoeffListWRT([I-3.29](#) pg.67), LC([I-12.5](#) pg.194)

I-3.51 content

syntax

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

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 of a (true) GCD domain “R” then the result is a fraction “c” such that “f/c” is a primitive polynomial with coefficients in “R”.

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-3.53](#) pg.76), coefficients([I-3.27](#) pg.66), IsTrueGCDDomain([I-9.106](#) pg.184), prim([I-16.32](#) pg.257)

I-3.52 ContentFreeFactor

syntax

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

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 indeterminates appearing in 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-3.53 pg.76), factor(I-6.1 pg.105), SqFreeFactor(I-19.37 pg.312)

I-3.53 ContentWRT

syntax

```

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

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-3.28 pg.67), content(I-3.51 pg.75), monomials(I-13.36 pg.218)

I-3.54 ContFrac

syntax

```

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

“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, 1, 1, 4, 1, 1, 1, 1, 1, 2, 1, 6]
```

See Also: CFApprox(I-3.10 pg.60), CFApproximants(I-3.11 pg.61), ContFracToRat(I-3.55 pg.77)

I-3.55 ContFracToRat

syntax

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

“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-3.54](#) pg.76), CFApprox([I-3.10](#) pg.60), CFApproximants([I-3.11](#) pg.61)

I-3.56 continue

syntax

```
continue
```

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.54) 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.54), return([I-18.43](#) pg.287), All CoCoA commands([II-2.2](#) pg.357)

I-3.57 CoprimeFactor

syntax

```
CoprimeFactor(N: INT, b:INT): INT
```

This function returns the largest factor of “N” which is coprime to “b”. An error is signalled if “N” or “b” is zero.

example

```
/**/ CoprimeFactor(100, 5);
4
/**/ CoprimeFactor(100, 21);
100
```

See Also: gcd([I-7.5](#) pg.122), CoprimeFactorBasis([I-3.58](#) pg.78)

I-3.58 CoprimeFactorBasis

— syntax —

```
CoprimeFactorBasis(L: LIST of INT): LIST of INT
CoprimeFactorBasis(L: LIST of RINGELEM): LIST of RINGELEM
```

This function returns a coprime factor base for a set of integers or ring elements from a (true) GCD domain.

Given a set of values $N = [N_1, \dots, N_k]$ we seek a factor base $G = [G_1, \dots, G_s]$ of pairwise coprime values such that each N_i is a product of powers of the G_j .

In general there are many different such sets G for a given set N . Such sets are sometimes also called **GCD-free bases**.

The factor base produced by these functions may not be of least cardinality. A least cardinality base can be obtained by performing all possible GCD and exact division operations iteratively.

— example —

```
/**/ CoprimeFactorBasis([factorial(20), factorial(10)]);
[46189, 4, 14175]
/**/ use QQ[x,y];
/**/ CoprimeFactorBasis([x^2*y^4, x^3*y^6]);
[x*y^2]
```

See Also: [gcd\(I-7.5 pg.122\)](#)

I-3.59 count

— syntax —

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

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\(I-4.21 pg.92\)](#), [len\(I-12.7 pg.195\)](#)

I-3.60 covers

— syntax —

```
covers(relP: LIST): LIST of LIST
```

This function print a poset description from the list “relP” of the strict relations in a poset.

— example —

```
// POSET:
//      3   4
//      \ /
//      2
//      |
//      1
```

```

/**/ relP := [[1,2], [2,3], [2,4]];
/**/ covers(relP);
[[], [1], [1, 2], [1, 2]]

/**/ indent(covers(relP));
[
  [],      // elems < 1
  [1],     // elems < 2
  [1, 2],  // elems < 3
  [1, 2]   // elems < 4
]
```

I-3.61 CpuTime

— syntax —

CpuTime(): RAT

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-20.7 pg.329) 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.83) to see the value of “CpuTime” in a more easily comprehensible form.

See Also: TimeFrom(I-20.7 pg.329), ElapsedTime(I-5.4 pg.96), DecimalStr(I-4.3 pg.83)

I-3.62 CRT

— syntax —

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

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-3.63 pg.80), RatReconstructByContFrac(I-18.19 pg.277), RatReconstructByLattice(I-18.20 pg.277)

I-3.63 CRTPoly

syntax

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

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-3.62 pg.79), RatReconstructPoly(I-18.21 pg.278)

I-3.64 CurrentRing

syntax

```
CurrentRing: RING
```

This is a top-level **system variable** containing the current ring.

NOTE: we advise against using it inside functions (see “TopLevel” (I-20.10 pg.329)). Previously, in CoCoA-4 it was a function (namely “CurrentRing()”)

example

```
/**/ use R ::= QQ[x,y];
/**/ use S ::= ZZ/(3)[t];
/**/ CurrentRing;
RingWithID(5, "ZZ/(3)[t]")

/**/ use R;
/**/ CurrentRing;
RingWithID(3, "QQ[x,y]")
```

See Also: RingOf(I-18.51 pg.290), TopLevel(I-20.10 pg.329)

I-3.65 CurrentTypes

syntax

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

This function lists all CoCoA data types.

example

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


I-3.66 cyclotomic

syntax

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

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

example

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

See Also: CyclotomicIndex, CyclotomicTest(I-3.68 pg.81), CyclotomicFactorIndexes(I-3.67 pg.81)

I-3.67 CyclotomicFactorIndexes

syntax

```
CyclotomicFactorIndexes(f: RINGELEM): LIST of INT
CyclotomicFactors_BeukersSmyth(f: RINGELEM): RECORD
CyclotomicFactors(f: RINGELEM): RECORD
```

The function “CyclotomicFactorIndexes” computes a list of indexes of the cyclotomic factors of the univariate polynomial “f”; it may contain some false positives (typically 3, 4 and/or 6).

The function “CyclotomicFactors_BeukersSmyth” produces a factorization of the product of all cyclotomic factors in “f”; these factors are coprime.

The function “CyclotomicFactors” computes the cyclotomic factors of the univariate polynomial “f”; it can be slower than “CyclotomicFactorIndexes”.

example

```
/**/ use QQ[z];
/**/ f := (z^2+1)^2 * (z^4+z^2+1);
/**/ CyclotomicFactorIndexes(f);
[3, 4, 6]
/**/ CyclotomicFactors(f);
record[RemainingFactor:=1, factors := [z^2+z+1, z^2-z+1, z^2+1], multiplicities := [1, 1, 2]]
/**/ CyclotomicFactors_BeukersSmyth(f);
record[RemainingFactor:=1, factors := [z^4+z^2+1, z^2+1], multiplicities := [1, 2]]
```

See Also: cyclotomic(I-3.66 pg.81), CyclotomicIndex, CyclotomicTest(I-3.68 pg.81), factor(I-6.1 pg.105)

I-3.68 CyclotomicIndex, CyclotomicTest

syntax

```
CyclotomicIndex(F: RINGELEM): INT
CyclotomicTest(F: RINGELEM): INT
```

The function “CyclotomicTest” takes the given polynomial “F” and returns the index “n” if it is the “n”-th cyclotomic polynomial; otherwise, it returns “0”. To test whether a polynomial is cyclotomic, call “CyclotomicTest”, and check that the result is non-zero.

The function “CyclotomicIndex” is similar but faster. It is correct if the given polynomial is cyclotomic, but could potentially return a non-zero value even if the input polynomial is not cyclotomic.

example

```
/**/ use R ::= QQ[x];
/**/ CyclotomicTest(x^6 + x^3 + 1);
```

```
9
/**/ CyclotomicTest(x^4 -x^2 +2);
0
```

See Also: `cyclotomic`([I-3.66](#) pg.81), `CyclotomicFactorIndexes`([I-3.67](#) pg.81)

Chapter I-4

D

I-4.1 dashes

syntax

```
dashes()
```

This function returns a string of dashes:

example

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

I-4.2 date

syntax

```
date() : INT
```

This function returns the date.

NOTE: from CoCoA version 5.0.4 the result is an INT and the date is in the form YYYYMMDD (in decimal). See also “TimeOfDay” ([I-20.8 pg.329](#)).

example

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

See Also: TimeOfDay([I-20.8 pg.329](#))

I-4.3 DecimalStr

syntax

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

This function produces a decimal string representation of the rational number “X” with up to “NumDigits” digits after the decimal point. If not specified, the default number of digits is 3.

If “X” is a RINGELEM, it is automatically converted to a RAT.

example

```
/**/ DecimalStr(1/3);
0.333
```



```
5
/**/ N; --> N is unchanged despite the function call.
0
```

(3) VARIABLE NUMBER OF PARAMETERS. It is also possible to have some optional parameters or a variable number of parameters. For optional parameters see also “IsDefined” ([I-9.47 pg.164](#)).

example

```
-- (3a) OPTIONAL ARGUMENTS
--      These must be in the last position(s).

/**/  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

-----
-- (3b) VARIABLE NUMBER OF PARAMETERS

/**/  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 CoCoA-4 statement, “Help S;” is now OBSOLETE!

See Also: [return\(I-18.43 pg.287\)](#), [TopLevel\(I-20.10 pg.329\)](#), [ref\(I-18.30 pg.281\)](#), All CoCoA commands([II-2.2 pg.357](#))

I-4.5 DefiningIdeal

syntax

```
DefiningIdeal(S: RING): IDEAL
```

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-3.5 pg.59), InducedHom(I-9.28 pg.157), NewQuotientRing(I-14.11 pg.227), BaseRing(I-2.1 pg.49)

I-4.6 deg

— syntax —

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

The first form of this function returns the **standard degree** of “F” (see “wdeg” (I-23.1 pg.341) 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-13.45 pg.221).

— 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", MakeTermOrdMat(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, y);
2
```

See Also: wdeg(I-23.1 pg.341), exponents(I-5.18 pg.101), NewPolyRing(I-14.9 pg.226), multiplicity(I-13.45 pg.221)

I-4.7 den

— syntax —

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

These 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-14.35 pg.236)

I-4.8 DensePoly

— syntax —

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

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", MakeTermOrdMat(Weights), 1);
/**/ use P;
/**/ DensePoly(P,1); // NOTE: standard degree!!
y +x
```

I-4.9 DenSigma

— syntax —

```
DenSigma(I: IDEAL): RINGELEM
```

This function returns the sigma-denominator of the ideal “I”, that is the common denominator of all polynomials in the sigma-reduced GBasis.

See article Abbott, Bigatti, Robbiano **Ideals modulo p** (“<https://arxiv.org/abs/1801.06112>”)

— example —

```
/**/ use QQ[x,y,z];
/**/ I := ideal(2*x*y^2 -1, 3*x^3*y -1);
/**/ ReducedGBasis(I);
[x*y^2 -1/2, x^2 +(-2/3)*y, y^3 +(-3/4)*x]
/**/ DenSigma(I);
12
```

See Also: CommonDenom(I-3.36 pg.70), IsSigmaGoodPrime(I-9.94 pg.180)

I-4.10 depth

— syntax —

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

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: 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-5.20 pg.102).

```

----- 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\(I-18.38 pg.285\)](#), [Ext\(I-5.20 pg.102\)](#)

I-4.11 deriv

```

----- syntax -----
deriv(F: RINGELEM, X: RINGELEM): RINGELEM

```

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

/**/ FrF := NewFractionField(R);
/**/ use FrF;

```



```
/**/ deriv((x*y^2)/(x-1), x);
(-y^2)/(x^2 -2*x +1)
```

See Also: [JacobianMat\(I-10.1 pg.189\)](#)

I-4.12 DerivationAction

— syntax —

```
DerivationAction(D: RINGELEM, P: RINGELEM)
```

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\(I-9.37 pg.161\)](#), [PerpIdealOfForm\(I-16.8 pg.249\)](#)

I-4.13 describe

— syntax —

```
describe X: OBJECT
```

This command prints some descriptive information about the object “X”. For instance, if “X” is a package name (prefixed with a “\$”), it prints out the exported names.

— example —

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

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

See Also: [PackageOf\(I-16.2 pg.247\)](#), [starting\(I-19.44 pg.315\)](#)

I-4.14 det

— syntax —

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

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: [HadamardBoundSq\(I-8.1 pg.135\)](#), [minors\(I-13.24 pg.213\)](#)

I-4.15 DF

syntax

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

Same as “LF” ([I-12.10 pg.196](#)), but does not throw an error if the argument is zero or if the “GradingDim” ([I-7.32 pg.130](#)) 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\(I-12.10 pg.196\)](#)

I-4.16 DiagMat

syntax

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

This function returns the diagonal matrix whose diagonal are the elements of the list “L”. By default the matrix is over “QQ” ([I-17.1 pg.267](#)).

example

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

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

See Also: [BlockMat\(I-2.9 pg.53\)](#), [IsDiagonal\(I-9.48 pg.164\)](#), [ColMat\(I-3.33 pg.69\)](#), [RowMat\(I-18.62 pg.294\)](#)

I-4.17 DicksonPoly

syntax

```
DicksonPoly(X: RINGELEM, N: INT, ALPHA: RINGELEM): RINGELEM
DicksonPoly2(X: RINGELEM, N: INT, ALPHA: RINGELEM): RINGELEM
```

These functions return the Dickson polynomials (of 1st and 2nd type respectively) of degree “N” in “X” with parameter “ALPHA”.

example

```
/**/ use QQ[x, alpha];
/**/ DicksonPoly(x, 3, alpha);
x^3 -3*x*alpha
```

I-4.18 diff

syntax

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

This function returns the list obtained by removing all the elements of “M” from “L”. See “deriv” (I-4.11 pg.88) for the derivative.

example

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

See Also: [remove\(I-18.36 pg.284\)](#)

I-4.19 dim

syntax

```
dim(R: RING): INT
```

This function computes the dimension of the ring “R”, where “R” is a quotient of a polynomial ring. NB: the coefficient ring must be a field.

If “I” is a zero-dimensional ideal in the polynomial ring “P”, and you want to compute the vector-space dimension of “P/I” then use the function “multiplicity” (I-13.45 pg.221).

example

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

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

See Also: [multiplicity\(I-13.45 pg.221\)](#)

I-4.20 discriminant

syntax

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

These functions compute the discriminant of a non-constant polynomial “F” with respect to the indeterminate “X”; in the first form “F” must be univariate, and “X” is taken to be its indeterminate.

The discriminant is defined as

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

where “M := SylvesterMat(F, 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-18.42 pg.286)

I-4.21 distrib

syntax

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

For each object “E” of a list “L”, let N(E) be the number of times “E” occurs as a component of “L”. Then “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-3.59 pg.78)

I-4.22 div

syntax

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

This function computes the integer quotient of “N” divided by “D”; it is equal to “N/D” **rounded towards zero**. The companion function “mod” (I-13.29 pg.215) computes the corresponding remainder.

If we set “Q = div(N,D)” and “R = mod(N,D)” then $N = Q * D + R$.

NOTE: for polynomials use “NR” (I-14.34 pg.235) (remainder), “DivAlg” (I-4.23 pg.93) (quotients and remainder), “IsIn” (I-9.59 pg.168) (ideal membership).

example

```
/**/ div(10,3);
3
/**/ div(10,-3);
-3
```

See Also: mod(I-13.29 pg.215), DivAlg(I-4.23 pg.93), IsIn(I-9.59 pg.168), NR(I-14.34 pg.235), ceil(I-3.9 pg.60), floor(I-6.17 pg.111)

I-4.23 DivAlg

— syntax —

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

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\(I-4.22 pg.92\)](#), [mod\(I-13.29 pg.215\)](#), [GenRepr\(I-7.7 pg.123\)](#), [NF\(I-14.17 pg.229\)](#), [NR\(I-14.34 pg.235\)](#)

I-4.24 domain

— syntax —

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

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\(I-3.24 pg.65\)](#), [Commands and Functions for RINGHOM\(III-10.3 pg.428\)](#), [Commands and Functions returning RINGHOM\(III-10.4 pg.428\)](#)

Chapter I-5

E

I-5.1 E_ [OBSOLETE]

[OBSOLETE] replaced by “CanonicalBasis” ([I-3.3 pg.58](#)) of a FreeModule.

I-5.2 eigenfactors

syntax

```
eigenfactors(M: MAT, X: RINGELEM): LIST of RINGELEM
```

“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-3.14 pg.62](#)) of “M” as polynomials in “X”.

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]
```

See Also: [eigenvectors\(I-5.3 pg.95\)](#)

I-5.3 eigenvectors

syntax

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

“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]]]]

```

See Also: [eigenfactors\(I-5.2 pg.95\)](#)

I-5.4 ElapsedTime

syntax

```
ElapsedTime(): RAT
```

Use the function “CpuTime” ([I-3.61 pg.79](#)) to measure how long your program took to compute its result.

This function returns a “RAT” whose value is the **elapsed** time in seconds since the start of the program. We recommend using “DecimalStr” ([I-4.3 pg.83](#)) to print out the value in a way which is convenient to comprehend.

example

```

/**/ TimeBeforeSleep := ElapsedTime();
/**/ SleepFor(5);
/**/ TimeAfterSleep := ElapsedTime();
/**/ DecimalStr(TimeAfterSleep - TimeBeforeSleep);
5.001

```

See Also: [CpuTime\(I-3.61 pg.79\)](#), [DecimalStr\(I-4.3 pg.83\)](#)

I-5.5 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

```

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

As an alternative, there is the function “ElimMat” ([I-5.7 pg.97](#)) for creating elimination orderings – this is used inside the function “elim”.

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.422)

I-5.6 ElimHomogMat

syntax

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

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 are unsure what this means, just use “ElimMat” ([I-5.7](#) pg.97) :-)

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-5.5](#) pg.96), ElimMat([I-5.7](#) pg.97), NewPolyRing([I-14.9](#) pg.226)

I-5.7 ElimMat

syntax

```

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

```

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]])

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

```

```
[1, 5, 2],
[0, 0, -1]])
/**/ P := NewPolyRing(QQ, "x,y,z", ElimOrd, 0);
```

See Also: [elim\(I-5.5 pg.96\)](#), [ElimHomogMat\(I-5.6 pg.97\)](#), [NewPolyRing\(I-14.9 pg.226\)](#)

I-5.8 EmbeddingHom

syntax

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

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\(I-3.4 pg.58\)](#)

I-5.9 EqSet

syntax

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

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\(I-9.34 pg.160\)](#), [IntersectionList\(I-9.35 pg.160\)](#), [IsSubset\(I-9.100 pg.182\)](#)

I-5.10 Equality Operator

syntax

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

These operators test for equality or not-equality. The first form returns “true” if “A” is equal to “B”, otherwise it returns “false”. The second form is the same as “not(A=B)”. These operators signal an error if the types of “A” and “B” differ, or if they are ring elements belonging to different rings.

To compare two values using an ordering see “Order Comparison Operators” ([I-15.10 pg.245](#))

example

```

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

```

See Also: Order Comparison Operators(I-15.10 pg.245), operators, shortcuts(I-0.1 pg.29)

I-5.11 EquiIsoDec

syntax

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

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 := IntersectionList([ideal(x-1,y-1,z-1), ideal(x-2,y-2)^2, ideal(x)^3]);
/**/ H := $radical.EquiIsoDec(I); H;
[ideal(x), ideal(y -2, x^2 -4*x +4), ideal(z -1, x*y -y^2 -2*x +2*y, x^2 -y^2 -4*x +4*y, y^3 -5*y^2 +4*y -4)];
/**/ T := [radical(J) | J in H];
/**/ S := IntersectionList(T);
/**/ radical(I) = S;
true

```

See Also: PrimaryDecomposition(I-16.33 pg.258), radical(I-18.1 pg.271), RadicalOfUnmixed(I-18.2 pg.271)

I-5.12 error

syntax

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

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(7);
2

-- /**/ T(1/3); --> !!! ERROR !!! as expected: "Argument must be an integer."

```

See Also: try(I-20.14 pg.332), GetErrMesg(I-7.15 pg.126)

I-5.13 EulerTotient

syntax

```
EulerTotient(N: INT): INT
```

This function computes the Euler **totient** function (also sometimes called Euler **phi** function). The value is the number of integers from 1 to “N” which are coprime to “N”.

example

```
/**/ EulerTotient(100);
40
/**/ EulerTotient(30030);
5760
```

See Also: PrimitiveRoot(I-16.38 pg.260)

I-5.14 eval

syntax

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

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-19.59 pg.320).

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.430), PolyAlgebraHom(I-16.17 pg.251), subst(I-19.59 pg.320)

I-5.15 EvalHilbertFn

syntax

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

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-8.8 pg.137), HilbertPoly(I-8.10 pg.138)

I-5.16 EvalQuasiPoly

syntax

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

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-14.19 pg.229), NmzHilbertBasis(I-14.23 pg.231), NmzNormalToricRing(I-14.28 pg.233), NmzIntClosureMonIdeal(I-14.25 pg.232), NmzSetVerbosityLevel(I-14.29 pg.234)

I-5.17 exit

syntax

```
exit
```

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

See Also: ciao(I-3.17 pg.63), quit(I-17.3 pg.268)

I-5.18 exponents

syntax

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

This function returns the list of exponents of the **leading term** of “F”. The inverse function is “MakeTerm” (I-13.4 pg.206).

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-12.22 pg.201), LT(I-12.24 pg.202), MakeTerm(I-13.4 pg.206)

I-5.19 export

— syntax —

```
export FunctionName
```

This command makes a function defined in a package visible at top-level (without having to use the package prefix). See the manual entry for “First Example of a Package” (II-9.2 pg.375) for an example.

The “**export**” command exports **just a single name**; you must call it separately for each exported function. All calls to “**export**” must be at the start of the package (*i.e.* before the first function definition).

If you do not want to export any names from your package then you should export the keyword “**skip**” (to avoid a warning that no “**export**” commands were found).

See Also: define(I-4.4 pg.84), Package essentials(II-9.3 pg.376), protect(I-16.49 pg.265)

I-5.20 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")
```

***** 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 $Im(Phi *_I)$, 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 $(J, Ext^J(M, N)) | J \in I$ 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 $Ext_R^I(M, R)$ 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)
-----
***** NOT YET IMPLEMENTED *****
  $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\(I-18.38 pg.285\)](#), [depth\(I-4.10 pg.87\)](#), [MinimalPresentation\(I-13.23 pg.213\)](#)

I-5.21 ExternalLibs

syntax

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

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

example

```
/**/ ExternalLibs();  
["GMP", "Normaliz", "Frobby", "Gfan", "CDD", "MathSAT"]  
/**/ indent(VersionInfo().ExternalLibs);  
[  
  record[name := "GMP", version := "60102"],  
  record[name := "Normaliz", version := "30102"],  
  (...)  
]
```

See Also: [VersionInfo\(I-22.3 pg.340\)](#)

Chapter I-6

F

I-6.1 factor

syntax

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

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-19.37](#) pg.[312](#)) and then “[ContentFreeFactor](#)” ([I-3.52](#) pg.[75](#)) to obtain a partial factorization).

(For information about the algorithm, consult John Abbott’s papers, for univariate polynomials with coefficients in an algebraic extension, from version 5.2.3, consult E.Palezzato’s PhD thesis)

To factorize an integer use “[FactorINT](#)” ([I-6.4](#) pg.[106](#)).

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]

/**/ L := NewQuotientRing(NewPolyRing(QQ,"a,b"), "a^2-2, b^2-3");
/**/ use L[x,y];
/**/ indent(factor(x^2-2));
record[
  RemainingFactor := (1),
  factors := [x + (a), x + (-a)],
  multiplicities := [1, 1]
```

```

]

/**/ indent(factor(x^2-6));
record[
  RemainingFactor := (1),
  factors := [x +(-a*b), x +(a*b)],
  multiplicities := [1, 1]
]

```

See Also: `FactorINT`([I-6.4](#) pg.106), `SqFreeFactor`([I-19.37](#) pg.312), `ContentFreeFactor`([I-3.52](#) pg.75), `IsDivisible`([I-9.49](#) pg.165)

I-6.2 FactorAlgExt [OBSOLESCENT]

Now directly handled by “`factor`” ([I-6.1](#) pg.105).

I-6.3 factorial

— syntax —

```
factorial(N: INT): INT
```

This function returns the factorial of “N”, the product of all integers up to and including “N”.

— example —

```

/**/ factorial(5);
120

/**/ factorial(25);
15511210043330985984000000

```

See Also: `binomial`([I-2.6](#) pg.51), `primorial`([I-16.39](#) pg.260)

I-6.4 FactorINT

— syntax —

```

FactorINT(N: INT): RECORD
FactorINT_TrialDiv(N: INT, MaxPrime: INT): RECORD
FactorINT_PollardRho(N: INT, Nitters: INT): RECORD

```

These functions find small (usually prime) factors of an integer. “`FactorINT`” may take a very long time, and may produce only a partial factorization! “`FactorINT_TrialDiv`” does trial division by all primes up to “`MaxPrime`”; be wary of giving large values for “`MaxPrime`”. “`FactorINT_PollardRho`” performs up to “`Nitters`” iterations of the Pollard-Rho algorithm; the iterations stop as soon as a (non-trivial) factor is found. If no factor was found, the result is an empty factorization. Note that the factor found may not be prime! Be wary of specifying large values for “`Nitters`”.

From version 5.0.4, to comply with the naming conventions, the fields are called “`factors`” and “`multiplicities`”; previously they were “`Factors`” and “`Exponents`”.

— example —

```

/**/ FactorINT_TrialDiv(100,3);
record[factors := [2], multiplicities := [2], RemainingFactor := 25]

/**/ FactorINT(123456789);
record[factors := [3, 3607, 3803], multiplicities := [2, 1, 1], RemainingFactor := 1]

```

See Also: `IsPrime`([I-9.85](#) pg.177), `IsProbPrime`([I-9.87](#) pg.178), `CoprimeFactorBasis`([I-3.58](#) pg.78)

I-6.5 FactorMultiplicity

syntax

```
FactorMultiplicity(b: INT, N: INT): INT
```

This function counts how many times the integer base “b” divides the integer “N”.

NOTE: “N” must be non-zero, and “b > 1”.

example

```
/**/ FactorMultiplicity(2, 20); // largest k s.t. 2^k divides 20
2
/**/ FactorMultiplicity(5, 20); // largest k s.t. 5^k divides 20
1
/**/ FactorMultiplicity(7, 20); // largest k s.t. 7^k divides 20
0
```

See Also: IsDivisible(I-9.49 pg.165), FactorINT(I-6.4 pg.106)

I-6.6 FGLM5

syntax

```
FGLM5(GB0ld: LIST, M: MAT): LIST
```

***** 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 “GB0ld” 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

```
***** NOT YET IMPLEMENTED *****
use QQ[x, y, z], DegRevLex;
GB0ld := *** [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(GB0ld, M);
use QQ[x, y, z], Ord(M);
-- New basis (Lex)
BringIn(GBNew);
```

I-6.7 fibonacci

syntax

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

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

example

```
/**/ fibonacci(0);
0
/**/ fibonacci(10);
55
```

I-6.8 fields

syntax

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

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-18.27 pg.280), record field selector(I-18.28 pg.281)

I-6.9 first

syntax

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

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.193), tail(I-20.3 pg.328)

I-6.10 FirstCols, FirstRows

syntax

```
FirstCols(M: MAT, N: INT): MAT
FirstRows(M: MAT, N: INT): MAT
```

This function returns the submatrix of “M” formed by the first “N” columns (or rows).

example

```
/**/ M := mat([[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15]]);

/**/ FirstCols(M, 3); -- same as submat(M, 1..3, 1..3);
matrix(QQ,
  [[1, 2, 3],
   [6, 7, 8],
   [11, 12, 13]])
```

```

/**/ FirstRows(M, 2); -- same as submat(M, 1..2, 1..5);
matrix(QQ,
  [[1, 2, 3, 4, 5],
   [6, 7, 8, 9, 10]])

```

See Also: [submat\(I-19.54 pg.318\)](#)

I-6.11 FirstNonZero

— syntax —

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

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);
3

```

See Also: [FirstNonZeroPosn\(I-6.12 pg.109\)](#), [IsZero\(I-9.107 pg.184\)](#), [NonZero\(I-14.32 pg.235\)](#)

I-6.12 FirstNonZeroPosn

— syntax —

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

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);
3

```

See Also: [FirstNonZero\(I-6.11 pg.109\)](#), [IsZero\(I-9.107 pg.184\)](#), [NonZero\(I-14.32 pg.235\)](#)

I-6.13 FixedDivisor

— syntax —

```
FixedDivisor(F: RINGELEM): RINGELEM
```

This function returns the integer fixed divisor of the polynomial “*F*”; that is the gcd of all “*F*(*n*)” as “*n*” runs through all integers.

The polynomial “*F*” must have rational coefficients, and all values “*F*(*n*)” must be integers. Current bug: “*F*” must be univariate.

example

```
/**/ use R ::= QQ[x];
/**/ f := (x-1)*(x-2)*(x-3);
/**/ FixedDivisor(f);
3
```

See Also: `content`([I-3.51](#) pg.75)

I-6.14 `flatten`

syntax

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

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* := `flatten`(*L*,1)” are formed as follows: go through the elements of “*L*” one at a time; if an element is not a list, add it to “*M*”; if an element is a list, add all of its elements to “*M*”.

Higher levels are recursive: “`flatten`(*L*, *N*) = `flatten`(`flatten`(*L*, *N*-1),1)”. For “*N*” large enough “`flatten`(*L*, *N*)” gives the same result as “`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(L,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]
```

See Also: `ConcatLists`([I-3.46](#) pg.73)

I-6.15 `FloatApprox`

syntax

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

This function computes an approximation of the form $M * 2^E$ to a rational number “*X*” where the mantissa satisfies $2^{(PrecBits - 1)} \leq |M| < 2^{PrecBits}$ 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-6.18 pg.112](#))). 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\(I-3.10 pg.60\)](#), [FloatStr\(I-6.16 pg.111\)](#), [MantissaAndExponent2\(I-13.8 pg.207\)](#)

I-6.16 FloatStr

syntax

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

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

The aim is to produce an easily readable result. An exception may be thrown if “N” is very large.

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, 35); -- specify number of digits
0.012345679012345679012345679012345679
```

See Also: [DecimalStr\(I-4.3 pg.83\)](#), [ScientificStr\(I-19.5 pg.299\)](#), [FloatApprox\(I-6.15 pg.110\)](#), [MantissaAndExponent10\(I-13.7 pg.207\)](#)

I-6.17 floor

syntax

```
floor(X: RAT): INT
```

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\(I-3.9 pg.60\)](#), [round\(I-18.61 pg.294\)](#), [num\(I-14.35 pg.236\)](#), [den\(I-4.7 pg.86\)](#), [div\(I-4.22 pg.92\)](#), [FloorSqrt\(I-6.20 pg.112\)](#), [FloorRoot\(I-6.19 pg.112\)](#), [FloorLog2](#), [FloorLog10](#), [FloorLogBase\(I-6.18 pg.112\)](#)

I-6.18 FloorLog2, FloorLog10, FloorLogBase

syntax

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

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”. “FloorLog2” applied to “INT” is particularly quick.

NOTE: “FloorLog10” may be useful for formatted printing as it gives the number of digits (minus 1) in base 10; 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-6.19 FloorRoot

syntax

```
FloorRoot(N: INT, R: INT): INT
```

This function computes the “R”-th root of the non-negative integer “N”. If the argument is not a perfect “R”-th power it returns the integer part of the root (the floor of the real root).

example

```
/**/ FloorRoot(25, 2);
5
/**/ FloorRoot(99, 3);
4
```

See Also: [floor\(I-6.17 pg.111\)](#), [FloorSqrt\(I-6.20 pg.112\)](#)

I-6.20 FloorSqrt

syntax

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

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.


```

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

```

See Also: floor(I-6.17 pg.111), FloorRoot(I-6.19 pg.112)

I-6.21 fold

```
fold(S: STRING, W: INT): STRING
```

This function copies a string inserting a newline every “W” characters. Together with “**sprint**” (I-19.35 pg.311) it can be helpful for displaying a long result more comprehensibly. Also Emacs can become quite slow if there are long lines in the output: combination of “**sprint**” (I-19.35 pg.311) and “**fold**” can help in such cases.

```
example  
/**/ n := factorial(100);  
/**/ fold(sprint(n), 60);  
933262154439441526816992388562667004907159682643816214685929  
638952175999932299156089414639761565182862536979208272237582  
511852109168640000000000000000000000
```

See Also: `FoldToListInput`(I-6.22 pg.113), `sprint`(I-19.35 pg.311)

I-6.22 FoldToListInput

FoldToListInput(S: STRING, W: INT): STRING

This function transforms a string so that it looks like a list of short strings (each of length at most “w”). Similar to “fold”, but the result is ready to be read by CoCoA.

WARNING: does not handle **funny** characters cleverly.

There is also “`FoldToList`” which returns a list of strings; it is disappointingly slow (so deliberately not properly documented).

```
example
```

```
/**/ n := factorial(100);  
/**/ FoldToListInput(sprintf(n), 60);  
["933262154439441526816992388562667004907159682643816214685929",  
 "638952175999932299156089414639761565182862536979208272237582",  
 "511852109168640000000000000000000000"]
```

See Also: `ConcatStrings`(I-3.47 pg.74), `fold`(I-6.21 pg.113), `sprint`(I-19.35 pg.311)

I-6.23 for

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_1, N_2, and D are integer expressions, and C is a sequence of commands.

In the first form, the loop variable “I” is assigned the values “N₁, N₁+1, ..., N₂” in succession. After each assignment, the command sequence “C” is executed. If “N₂ < N₁”, then the command sequence “C” is not executed.

The second form is almost the same, except that “I” is assigned the values “N₁, N₁+D, N₁+2D”, and so on, until the limit “N₂” is passed. If “N₂ - N₁” 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: we used “ref L” so that the function can change the value of the variable referenced by “L”. See “ref” ([I-18.30 pg.281](#)).

See Also: [foreach\(I-6.24 pg.114\)](#), [repeat\(I-18.37 pg.284\)](#), [All CoCoA commands\(II-2.2 pg.357\)](#), [while\(I-23.3 pg.342\)](#)

I-6.24 foreach

syntax

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

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

NOTE: the variable “X” contains a **copy of the value** in the list; changing “X” will not change the list “L”; instead you must iterate over indexes into the list.

example

```
/**/ foreach N in 1..10 do -- recall that 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\(I-6.23 pg.113\)](#), [List Constructors\(III-5.2 pg.406\)](#), [repeat\(I-18.37 pg.284\)](#), All CoCoA commands([II-2.2 pg.357](#)), [while\(I-23.3 pg.342\)](#)

I-6.25 *format*

— syntax —

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

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: [SprintTrunc\(I-19.36 pg.311\)](#), [latex\(I-12.3 pg.193\)](#), [sprint\(I-19.35 pg.311\)](#)

I-6.26 *FrbAlexanderDual*

— syntax —

```

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

```

Using the “Frobby” ([II-10.4 pg.381](#)) 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: [Frobby](#)([II-10.4](#) pg.381), [FrbPrimaryDecomposition](#)([I-6.30](#) pg.117), [PrimaryDecomposition](#)([I-16.33](#) pg.258)

I-6.27 FrbAssociatedPrimes

— syntax —

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

Using the “Frobby” ([II-10.4](#) pg.381) 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: [Frobby](#)([II-10.4](#) pg.381), [FrbIrreducibleDecomposition](#)([I-6.28](#) pg.116), [FrbPrimaryDecomposition](#)([I-6.30](#) pg.117)

I-6.28 FrbIrreducibleDecomposition

— syntax —

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

Using the “Frobby” ([II-10.4](#) pg.381) 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: [Frobby](#)([II-10.4](#) pg.381), [FrbAssociatedPrimes](#)([I-6.27](#) pg.116), [FrbIrreducibleDecomposition](#)([I-6.28](#) pg.116)

I-6.29 FrbMaximalStandardMonomials

— syntax —

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

Using the “Frobby” ([II-10.4](#) pg.381) 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-10.4](#) pg.381)

I-6.30 FrbPrimaryDecomposition

syntax

```
FrbPrimaryDecomposition(I: IDEAL): LIST
```

Using the “Frobby” ([II-10.4 pg.381](#)) 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\(II-10.4 pg.381\)](#), [FrbAssociatedPrimes\(I-6.27 pg.116\)](#), [FrbIrreducibleDecomposition\(I-6.28 pg.116\)](#), [PrimaryDecomposition\(I-16.33 pg.258\)](#)

I-6.31 FrobeniusMat

syntax

```
FrobeniusMat(I: IDEAL): MAT
FrobeniusMat(I: IDEAL, QB: LIST): MAT
```

The Frobenius map on “ R/I ” sends “ $f \mapsto f^q$ ”. This function computes the matrix of the Frobenius Map in “ R/I ” where “ I ” is a zero-dimensional ideal. The second uses the given quotient basis, otherwise the Macaulay basis is used.

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\(I-9.59 pg.168\)](#), [IsSubset\(I-9.100 pg.182\)](#)

I-6.32 FrobeniusNormSq

syntax

```
FrobeniusNormSq(M: MAT): RINGELEM
```

This function computes the **square of** the Frobenius norm of “ M ”; namely, the sum of the squares of the matrix entries.

example

```
/**/ M := mat([[1,2],[3,4]]);
/**/ FrobeniusNormSq(M);
30
```

See Also: [HadamardBoundSq\(I-8.1 pg.135\)](#)

I-6.33 func

syntax

```
Func(ARGS) BODY EndFunc
returns FUNCTION
```

This syntactic structure defines an anonymous function, *i.e.* a function without a name. Anonymous functions can be passed as parameters, assigned to variables, and returned as values.

To access variables outside the anonymous function use the commands “[ImportByRef](#), [ImportByValue](#)” ([I-9.17 pg.152](#)) or “[TopLevel](#)” ([I-20.10 pg.329](#)).

example

```
/**/ square := Func(x) Return x^2; EndFunc;
/**/ square(3);
9

/**/ strings := ["x", "zzz", "yy"];
/**/ SortedBy(strings, Func(x,y) return len(x)>len(y); EndFunc);
["zzz", "yy", "x"]
```

See Also: [define\(I-4.4 pg.84\)](#), [TopLevel\(I-20.10 pg.329\)](#), [ImportByRef](#), [ImportByValue\(I-9.17 pg.152\)](#)

I-6.34 Function [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\(III-7.2 pg.417\)](#), [SortBy\(I-19.29 pg.308\)](#), [SortedBy\(I-19.31 pg.310\)](#)

I-6.35 functions [OBSOLETE]

Use “[describe](#)” ([I-4.13 pg.89](#)).

I-6.36 FVector

syntax

```
FVector(A: LIST): RECORD
```

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-19.19](#) pg.[305](#))

Chapter I-7

G

I-7.1 GBasis

syntax

```
GBasis(I: IDEAL|MODULE): LIST
```

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-18.29](#) pg.281).

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*y -y^2, x^2 -y^2, y^3 -y]
```

See Also: GBasis timeout([I-7.2](#) pg.121)

I-7.2 GBasis timeout

syntax

```
GBasis_timeout(I: IDEAL, TimeLimit: INT): LIST
GBasis_timeout(M: MODULE, TimeLimit: INT): LIST
```

Same as “GBasis” ([I-7.1](#) pg.121), 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.332).

example

```
/**/ use R := QQ[t,x,y,z];
/**/ I := ideal(t^3-x, t^4-y, t^5-z);
/**/ J := I^5;
-- /**/ G := GBasis_timeout(J, 0.1);
--> !!! ERROR !!! as expected: Computation exceeded given time limit

/**/ J := I^5; G := GBasis_timeout(J, 10); --> OK
```

See Also: GBasis([I-7.1](#) pg.121), try([I-20.14](#) pg.332)

I-7.3 GBasisByHomog

syntax

```
GBasisByHomog(I: IDEAL): LIST
```

Same as “GBasis” (I-7.1 pg.121), but it is computed by homogenizing the input generators, and then dehomogenizing the output.

From version 5.4.2 works with all term-orderings.

example

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

See Also: GBasis(I-7.1 pg.121)

I-7.4 GBM

syntax

```
GBM(L: LIST): IDEAL
```

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

This function computes the intersection of ideals corresponding to zero-dimensional schemes: GBM is for affine schemes, and “HGBM” (I-8.5 pg.136) for projective schemes. The list L must be a list of ideals. The function “IntersectionList” (I-9.35 pg.160) 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.143), IdealAndSeparatorsOfProjectivePoints(I-9.4 pg.144), IdealOfPoints(I-9.7 pg.147), IdealOfProjectivePoints(I-9.8 pg.147), HGBM(I-8.5 pg.136)

I-7.5 gcd

syntax

```
gcd(M: INT, N: INT): INT
gcd(L: LIST of INT): INT
```

```
gcd(F: RINGELEM, G: RINGELEM): RINGELEM
gcd(L: LIST of RINGELEM): RINGELEM
```

This function returns the greatest common divisor of its arguments 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\(I-4.22 pg.92\)](#), [mod\(I-13.29 pg.215\)](#), [lcm\(I-12.6 pg.195\)](#), [IsCoprime\(I-9.46 pg.164\)](#)

I-7.6 GenericPoints

syntax

```
GenericPoints(R: RING, NumPoints: INT): LIST
GenericPoints(R: RING, NumPoints: INT, RandomRange: INT): LIST
```

“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
```

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 “[F₁, ..., F_t]” such that (“[G₁, ..., 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))”.

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-4.23](#) pg.[93](#)), IsIn([I-9.59](#) pg.[168](#)), NF([I-14.17](#) pg.[229](#)), syz([I-19.71](#) pg.[325](#)), SyzOfGens([I-19.72](#) pg.[326](#))

I-7.8 gens

syntax

```

gens(I: IDEAL): LIST
gens(M: MODULE): LIST

```

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.125](#)), IdealOfMinGens([I-9.6 pg.146](#)), SubmoduleOfMinGens([I-19.57 pg.319](#))

I-7.9 GensAsCols, GensAsRows

— syntax —

```
GensAsRows(M: MODULE): MAT
GensAsCols(M: MODULE): MAT
```

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.124](#)), SubmoduleCols, SubmoduleRows([I-19.56 pg.319](#))

I-7.10 GensJacobian

— syntax —

```
GensJacobian(Q: RINGELEM): LIST
```

This function returns the list of generators of the Jacobian ideal of a polynomial Q.

— example —

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

I-7.11 Get [OBSOLETE]

Essentially replaced by “`GetLine`” ([I-7.16](#) pg.127).

I-7.12 GetCol

syntax

```
GetCol(M: MAT, K: INT): LIST
```

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-7.17](#) pg.127), `GetCols`([I-7.13](#) pg.126)

I-7.13 GetCols

syntax

```
GetCols(M: MAT): LIST of LIST
```

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-7.12](#) pg.126), `GetRows`([I-7.18](#) pg.127)

I-7.14 GetEnv

syntax

```
GetEnv(S: STRING): STRING
```

This function returns the value of system shell variable whose name is “S”.

CoCoA is normally started by its own shell script; so we recommend using “`export`” ([I-5.19](#) pg.102) to make your shell variables visible to CoCoA.

example

```
/**/ GetEnv("HOME");
/Users/bigatti

/**/ GetEnv("SHELL");
/bin/bash
```

See Also: `VersionInfo`([I-22.3](#) pg.340), `SystemCommand`([I-19.70](#) pg.324)

I-7.15 GetErrMsg

syntax

```
GetErrMsg(E: ERROR): STRING
```

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\(I-20.14 pg.332\)](#), [error\(I-5.12 pg.99\)](#)

I-7.16 *GetLine*

syntax

```

GetLine(IN: ISTREAM): STRING

```

This function reads a line of input from the in-stream “IN”, and returns the result as a string. The string does not contain the end-of-line character. The function “IsAtEOF” ([I-9.42 pg.162](#)) says whether the end of input has been reached.

example

```

/**/ Istring := OpenIString("one\ntwo");
/**/ GetLine(Istring); -- get the first line
one
/**/ GetLine(Istring); -- get next line
two
/**/ IsAtEOF(Istring);
true

```

See Also: [IsAtEOF\(I-9.42 pg.162\)](#), [OpenIFile\(I-15.2 pg.241\)](#), [OpenIString\(I-15.3 pg.242\)](#), [StandardInput\(I-19.41 pg.314\)](#)

I-7.17 *GetRow*

syntax

```

GetRow(M: MAT, K: INT): LIST

```

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\(I-7.18 pg.127\)](#), [SetRow\(I-19.12 pg.303\)](#)

I-7.18 *GetRows*

syntax

```

GetRows(M: MAT): LIST of LIST

```

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\(I-7.17 pg.127\)](#)

I-7.19 GFanContainsPositiveVector

— **syntax** —

```
GFanContainsPositiveVector(EqMat: MAT, IneqMat: MAT): INT
```

...to do..

I-7.20 GFanGeneratorsOfLinealitySpace

— **syntax** —

```
GFanGeneratorsOfLinealitySpace(EqMat: MAT, IneqMat: MAT): MAT
```

...to do..

I-7.21 GFanGeneratorsOfSpan

— **syntax** —

```
GFanGeneratorsOfSpan(EqMat: MAT, IneqMat: MAT): MAT
```

...to do..

I-7.22 GFanGetAmbientDimension

— **syntax** —

```
GFanGetAmbientDimension(EqMat: MAT, IneqMat: MAT): INT
```

...to do..

I-7.23 GFanGetCodimension

— **syntax** —

```
GFanGetCodimension(EqMat: MAT, IneqMat: MAT): INT
```

...to do..

I-7.24 GFanGetDimension

— **syntax** —

```
GFanGetDimension(EqMat: MAT, IneqMat: MAT): INT
```

...to do..

I-7.25 GFanGetDimensionOfLinealitySpace

syntax

```
GFanGetDimensionOfLinealitySpace(EqMat: MAT, IneqMat: MAT): INT
```

...to do..

I-7.26 GFanGetFacets

syntax

```
GFan(EqMat: MAT, IneqMat: MAT): MAT
```

...to do..

I-7.27 GFanGetImpliedEquations

syntax

```
GFanGetImpliedEquations(EqMat: MAT, IneqMat: MAT): MAT
```

...to do..

I-7.28 GFanGetUniquePoint

syntax

```
GFanGetUniquePoint(EqMat: MAT, IneqMat: MAT): MAT
```

...to do..

I-7.29 GFanRelativeInteriorPoint

syntax

```
GFanRelativeInteriorPoint(EqMat: MAT, IneqMat: MAT): MAT
```

This function returns a column matrix whose entries are the coordinates of a relative interior point of the cone described by “IneqMat”

example

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

I-7.30 gin

syntax

```
gin(I: IDEAL): IDEAL
```

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-14.13 pg.227)) 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-18.46 pg.288) for computing wrt DegRevLex independently of the current ring.

Verbosity:

with verbosity “ ≥ 50 ” it prints

the two random changes of coordinates used

and the “NewRingTwinFloat” (I-14.13 pg.227) 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));
[L50,RandIdeal]   change coord = [
  484819*x,
  -844426*x -7426*y,
  695955*x +168758*y -239080*z
]
[L50,TryPrecisions] -- trying with FloatPrecision 64
[L50,RandIdeal]   change coord = [
  304634*x,
  480790*x -499447*y,
  -732749*x -840921*y -466314*z
]
[L50,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-14.13 pg.227), rgin(I-18.46 pg.288)

I-7.31 GinJacobian

syntax

```
GensJacobian(Q: RINGELEM): IDEAL
```

This function returns the generic initial ideal of the Jacobian ideal of a polynomial Q.

example

```

/**/ use QQ[x,y,z];
/**/ Q := x^3;
/**/ GinJacobian(Q);
ideal(x^2)

```

I-7.32 GradingDim

syntax

```
GradingDim(P): INT
```

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 := MakeTermOrdMat(RowMat([2,3]));
/**/ P := NewPolyRing(QQ, "x,y", OrdM, 1);

```

```
/**/ GradingDim(P);
1
```

See Also: NewPolyRing(I-14.9 pg.226), GradingMat(I-7.33 pg.131)

I-7.33 GradingMat

syntax

```
GradingMat(R: RING): MAT
```

This function returns the grading matrix (or weights matrix) for the polynomials ring “R”. To define a ring with (multi)weights use “NewPolyRing” (I-14.9 pg.226).

example

```
/**/ OrdM := MakeTermOrdMat(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.86), wdeg(I-23.1 pg.341), GradingDim(I-7.32 pg.130), NewPolyRing(I-14.9 pg.226)

I-7.34 graeffe

syntax

```
graeffe(F RINGELEM): RINGELEM
graeffe3(F RINGELEM): RINGELEM
GraeffeN(F RINGELEM, N INT): RINGELEM
```

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 a polynomial whose roots are the cubes of the roots of “F”.

For more general cases, “GraeffeN” produces a polynomial whose roots are the “N”-th powers 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
/**/ GraeffeN(f, 7);
x^3 -115*x^2 -24187*x -823543
```

See Also: RootBound(I-18.59 pg.293)

I-7.35 GraverBasis

syntax

```
GraverBasis(M: MAT): LIST of RINGELEM
```

These function return the Graver basis, computed with “toric” (I-20.12 pg.330).

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.330), HilbertBasisKer(I-8.7 pg.137)

I-7.36 GroebnerFanIdeals

syntax

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

Returns a LIST of ideals, one for each possible distinct reduced Groebner basis of “I”; each ideal is in a different ring (one ring for each term-ordering), and has as its generators the corresponding GBasis.

See also “CallOnGroebnerFanIdeals” (I-3.2 pg.57) for a way of computing with each ideal in succession (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, 3, 4, 3, 3, 4, 3, 3]
/**/ OrdMat(RingOf(GF[1])); --> matrix of the term-ordering
matrix(ZZ,
  [[1, 1, 1],
   [0, 0, -1],
   [0, -1, 0]])

-- 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-15.11 pg.245), RingOf(I-18.51 pg.290), UniversalGBasis(I-21.2 pg.335)

I-7.37 GroebnerFanReducedGBases

— syntax —

GroebnerFanReducedGBases(I: IDEAL): LIST of IDEAL

This function returns a list of all reduced GBases. It is good for **visualizing** small examples; however, for further computations with the different bases it is better to use “GroebnerFanIdeals” ([I-7.36](#) pg.132) or “CallOnGroebnerFanIdeals” ([I-3.2](#) pg.57).

— 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.57), GroebnerFanIdeals([I-7.36](#) pg.132)

Chapter I-8

H

I-8.1 HadamardBoundSq

— syntax —

```
HadamardBoundSq(M: MAT): RECORD
```

This computes **square of** the Hadamard determinant bound (both for rows and for columns). The result is a record with two fields “ColBoundSq” and “RowBoundSq”. An error is signalled if “M” is not square.

The field names were changed in version 5.4.2.

— example —

```
/**/ M := mat([[1,2],[3,4]]);  
/**/ HadamardBoundSq(M);  
record[ColBoundSq := 200, RowBoundSq := 125]
```

See Also: [det\(I-4.14 pg.89\)](#)

I-8.2 HasGBasis

— syntax —

```
HasGBasis(I: IDEAL): BOOL
```

After the “GBasis” ([I-7.1 pg.121](#)) of “I” is (explicitly 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\(I-9.5 pg.146\)](#)

I-8.3 HColon

syntax

```
HColon(M: IDEAL, N: IDEAL): IDEAL
```

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

The function “colon” (I-3.34 pg.69) 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-3.34 pg.69) only when the input is non-homogeneous, in which case, “HColon” may be faster.

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)
***** NOT YET IMPLEMENTED *****
HColon(ideal(x^2, x*y), ideal(x, x-y^2));
ideal(x)
-----
```

See Also: HSaturation(I-8.17 pg.142), saturate(I-19.3 pg.298), HColon(I-8.3 pg.136), colon(I-3.34 pg.69)

I-8.4 HermitePoly

syntax

```
HermitePoly(N: INT, X: RINGELEM): RINGELEM
HermitePoly2(N: INT, X: RINGELEM): RINGELEM
```

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-3.15 pg.62), LaguerrePoly(I-12.1 pg.193)

I-8.5 HGBM

syntax

```
HGBM(L: LIST): IDEAL
```

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

This function computes the intersection of ideals corresponding to zero-dimensional schemes: “GBM” (I-7.4 pg.122) is for affine schemes, and HGBM for projective schemes. The list L must be a list of ideals. The function “IntersectionList” (I-9.35 pg.160) 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

```

**** NOT YET IMPLEMENTED ****
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\(I-9.3 pg.143\)](#), [IdealAndSeparatorsOfProjectivePoints\(I-9.4 pg.144\)](#), [IdealOfPoints\(I-9.7 pg.147\)](#), [IdealOfProjectivePoints\(I-9.8 pg.147\)](#), [GBM\(I-7.4 pg.122\)](#)

I-8.6 hilbert [OBSOLESCENT]

Renamed to “HilbertFn” ([I-8.8 pg.137](#)).

I-8.7 HilbertBasisKer

syntax

```
HilbertBasisKer(M: MAT): LIST
```

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\(I-12.13 pg.198\)](#), [NmzHilbertBasis\(I-14.23 pg.231\)](#)

I-8.8 HilbertFn

syntax

```

HilbertFn(R: RING|IDEAL): TAGGED("$hp.Hilbert")
HilbertFn(R: RING|IDEAL, N: INT): INT

```

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-5.15 pg.100](#)) 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-5.15 pg.100), HilbertPoly(I-8.10 pg.138), HVector(I-8.18 pg.142), HilbertSeries(I-8.11 pg.139)

I-8.9 HilbertMat

syntax

```
HilbertMat(N: INT): MAT over QQ
```

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-8.10 HilbertPoly

syntax

```
HilbertPoly(R: (Poly or Quotient)RING): RINGELEM in the ring QQt
```

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-8.8 pg.137).

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-5.15 pg.100), HilbertFn(I-8.8 pg.137), HVector(I-8.18 pg.142), HilbertSeries(I-8.11 pg.139), RingQQt(I-18.53 pg.291)

I-8.11 HilbertSeries

syntax

```
HilbertSeries(M: MODULE|IDEAL|RING): TAGGED("$hp.PSeries")
```

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-16.36 pg.259).

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", MakeTermOrdMat(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", MakeTermOrdMat(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", MakeTermOrdMat(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\(I-4.19 pg.91\)](#), [multiplicity\(I-13.45 pg.221\)](#), [HilbertFn\(I-8.8 pg.137\)](#), [HVector\(I-8.18 pg.142\)](#), [HilbertSeriesShifts\(I-8.13 pg.140\)](#), [HilbertSeriesMultiDeg\(I-8.12 pg.140\)](#), [GradingMat\(I-7.33 pg.131\)](#), [Primary-HilbertSeries\(I-16.36 pg.259\)](#)

I-8.12 HilbertSeriesMultiDeg

— syntax —

```
HilbertSeriesMultiDeg(RmodI: RING, WM: MAT): TAGGED("$hp.PSeries")
```

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\(I-8.11 pg.139\)](#)

I-8.13 HilbertSeriesShifts

— syntax —

```
HilbertSeriesShifts(M: MODULE, ShiftsList: LIST): TAGGED("$hp.PSeries")
```

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: tagged objects cannot be usefully compared for equality with untagged 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\(I-4.19 pg.91\)](#), [HilbertFn\(I-8.8 pg.137\)](#), [HVector\(I-8.18 pg.142\)](#), [multiplicity\(I-13.45 pg.221\)](#), [GradingMat\(I-7.33 pg.131\)](#)

I-8.14 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

```

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); -- not just ideal gen by 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\(I-9.58 pg.168\)](#)

I-8.15 HomogCompt

syntax

```
HomogCompt(f: RINGELEM, d: INT): RINGELEM
```

This function returns the homogeneous part of weighted degree “d” of “f”.

NOTE: currently works only if the weighted degrees are integers (gr.dim is 1).

example

```

/**/ use P := QQ[a,b,c];
/**/ HomogCompt(c^2-a+b+c+1, 1);
-a +b +c
/**/ OrdM := matrix([[2,3,1],[0,0,-1],[0,-1,0]]);
/**/ P := NewPolyRing(QQ, "a,b,c", OrdM, 1); -- 3 indeterminates

```

```

/**/ use P;
/**/ HomogCompt(c^2-a+b+c+1, 1);
c

```

See Also: IsHomog(I-9.58 pg.168), LF(I-12.10 pg.196), DF(I-4.15 pg.90), RandomLinearForm(I-18.6 pg.272)

I-8.16 HomogElimMat [OBSOLESCENT]

Renamed to “ElimHomogMat” (I-5.6 pg.97).

I-8.17 HSaturation

syntax

```

HSaturation(I: IDEAL, J: IDEAL): IDEAL

```

***** 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-19.3 pg.298) 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-3.34 pg.69), HColon(I-8.3 pg.136), saturate(I-19.3 pg.298)

I-8.18 HVector

syntax

```

HVector(M: (Poly or Quotient)RING): LIST
HVector(M: MODULE): LIST

```

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-8.8 pg.137), HilbertSeries(I-8.11 pg.139)

Chapter I-9

I

I-9.1 ID [OBSOLETE]

Renamed to “RingID” ([I-18.50](#) pg.290).

I-9.2 ideal

— syntax —

```
ideal(g1: RINGELEM,...,gn: RINGELEM): IDEAL
ideal(L: LIST): IDEAL
ideal(R: RING, L: LIST): IDEAL
```

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
`\textbf{distinct}` points in affine space.

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

This function computes the results of “[IdealOfPoints](#)” (I-9.7 pg.147) and “[SeparatorsOfPoints](#)” (I-19.8 pg.300) 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:

```
***** NOT YET IMPLEMENTED *****
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.144).

example

```
***** NOT YET IMPLEMENTED *****
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-7.4 pg.122), HGBM(I-8.5 pg.136), GenericPoints(I-7.6 pg.123), IdealAndSeparatorsOfProjectivePoints(I-9.4 pg.144), IdealOfPoints(I-9.7 pg.147), IdealOfProjectivePoints(I-9.8 pg.147), Interpolate(I-9.31 pg.158), QuotientBasis(I-17.4 pg.268), SeparatorsOfPoints(I-19.8 pg.300), SeparatorsOfProjectivePoints(I-19.9 pg.301)

I-9.4 IdealAndSeparatorsOfProjectivePoints

syntax

```
IdealAndSeparatorsOfProjectivePoints(Points: LIST): RECORD
```


where Points is a list of lists of coefficients representing a set of $\text{\textbf{distinct}}$ points in projective space.

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

This function computes the results of “IdealOfProjectivePoints” (I-9.8 pg.147) and “SeparatorsOfProjectivePoints” (I-19.9 pg.301) 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:

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

```

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.143).

example

```

***** NOT YET IMPLEMENTED *****
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]
-----
    Res(R/X.Ideal); -- the resolution of the ideal
0 --> R^10(-6) --> R^24(-5) --> R^15(-4) --> R
-----

```

See Also: HGBM(I-8.5 pg.136), GBM(I-7.4 pg.122), GenericPoints(I-7.6 pg.123), IdealAndSeparatorsOfPoints(I-9.3 pg.143), IdealOfPoints(I-9.7 pg.147), IdealOfProjectivePoints(I-9.8 pg.147), Interpolate(I-9.31 pg.158), QuotientBasis(I-17.4 pg.268), SeparatorsOfPoints(I-19.8 pg.300), SeparatorsOfProjectivePoints(I-19.9 pg.301)

I-9.5 IdealOfGBasis

syntax

```

IdealOfGBasis(I: IDEAL): IDEAL

```

After the “GBasis” (I-7.1 pg.121) of “I” is (explicitly 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-8.2 pg.135)

I-9.6 IdealOfMinGens

syntax

```

IdealOfMinGens(I: IDEAL): IDEAL

```

It works only in the homogeneous case: for the inhomogeneous case see “MinSubsetOfGens” (I-13.28 pg.215).

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\(I-13.19 pg.212\)](#), [MinSubsetOfGens\(I-13.28 pg.215\)](#)

I-9.7 IdealOfPoints

syntax

```

IdealOfPoints(P: RING, Points: MAT): IDEAL
where Points is a MAT of coefficients whose rows represent a set of
\textbf{distinct} points in affine space.

```

This function computes the reduced Groebner basis for the ideal of all polynomials which vanish at the given set of **distinct** 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.147](#)).

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\(I-7.4 pg.122\)](#), [HGBM\(I-8.5 pg.136\)](#), [GenericPoints\(I-7.6 pg.123\)](#), [IdealAndSeparatorsOfPoints\(I-9.3 pg.143\)](#), [IdealAndSeparatorsOfProjectivePoints\(I-9.4 pg.144\)](#), [IdealOfProjectivePoints\(I-9.8 pg.147\)](#), [Interpolate\(I-9.31 pg.158\)](#), [IsZeroDim\(I-9.110 pg.185\)](#), [QuotientBasis\(I-17.4 pg.268\)](#), [SeparatorsOfPoints\(I-19.8 pg.300\)](#), [SeparatorsOfProjectivePoints\(I-19.9 pg.301\)](#)

I-9.8 IdealOfProjectivePoints

syntax

```

IdealOfPoints(P: RING, Points: MAT): IDEAL
where Points is a MAT of coefficients whose rows represent a set of
\textbf{distinct} points in projective space.

```

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.147).

— example —

```
/**/ use P := QQ[x,y,z];
/**/ I := IdealOfProjectivePoints(P, mat([[0,0,1],[1/2,1,1],[0,1,0]]));
/**/ I; -- gens are the reduced Groebner basis
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)
```

See Also: GBM(I-7.4 pg.122), HGBM(I-8.5 pg.136), GenericPoints(I-7.6 pg.123), IdealAndSeparatorsOfPoints(I-9.3 pg.143), IdealAndSeparatorsOfProjectivePoints(I-9.4 pg.144), IdealOfPoints(I-9.7 pg.147), Interpolate(I-9.31 pg.158), QuotientBasis(I-17.4 pg.268), SeparatorsOfPoints(I-19.8 pg.300), SeparatorsOfProjectivePoints(I-19.9 pg.301)

I-9.9 IdentityMat

— syntax —

```
IdentityMat(R: RING, N: INT): MAT
```

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
```

See Also: ZeroMat(I-25.2 pg.345), NewMatFilled(I-14.8 pg.225)

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.

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 CoCoA-4 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: Bool01(I-2.11 pg.54), All CoCoA commands(II-2.2 pg.357)

I-9.11 ILogBase [OBSOLETE]

Renamed: see “FloorLog2, FloorLog10, FloorLogBase” (I-6.18 pg.112).

I-9.12 image [OBSOLESCENT]

In CoCoA-5 homomorphisms are properly implemented as “RINGHOM” (III-10 pg.427). “Image” was the CoCoA-4 function mimicking homomorphisms, in particular “PolyAlgebraHom” (I-16.17 pg.251).

example

```
/**/ use Dom := QQ[x,y]; -- domain
/**/ f := x-y; -- a RINGELEM in D

/**/ use Cod := 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(Dom, Cod, [a, c^2-a*b]); -- a RINGHOM
/**/ phi(f);
a*b -c^2 +a
/**/ phi([f, f^2]);
[a*b -c^2 +a, a^2*b^2 -2*a*b*c^2 +c^4 +2*a^2*b -2*a*c^2 +a^2]
```

See Also: PolyAlgebraHom(I-16.17 pg.251), Introduction to RINGHOM(III-10.1 pg.427), BringIn(I-2.13 pg.55), subst(I-19.59 pg.320)

I-9.13 implicit

— syntax —

```
implicit(SubalgebraGens: LIST): IDEAL
implicit(R: RING, SubalgebraGens: LIST): IDEAL
```

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.330). if you know the answer is a hypersurface then use “ImplicitHypersurface” (I-9.14 pg.150).

— 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.150), ker(I-11.1 pg.191)

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
```

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^(len(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.150)

I-9.15 ImplicitPlot

syntax

```
ImplicitPlot(F: POLY, Xrange: LIST, Yrange: LIST)
```

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.151) 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 ‘\verb&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\(I-9.16 pg.151\)](#), [PlotPoints\(I-16.12 pg.250\)](#)

I-9.16 ImplicitPlotOn

syntax

```
ImplicitPlotOn(F: POLY, Xrange: LIST, Yrange: LIST, PlotFileName: STRING)
```

This function is the same as “ImplicitPlot” (I-9.15 pg.151) with a fourth argument giving the name of the file to print on.

NOTE: the last argument is a “STRING”, the name of the file, and not an “OSTREAM”, as for “print on” (I-16.41 pg.261).

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\(I-9.15 pg.151\)](#), [PlotPointsOn\(I-16.13 pg.251\)](#)

I-9.17 ImportByRef, ImportByValue

syntax

```
ImportByRef X;
ImportByValue X;
where ‘\verb&X&’ is the name of a variable in the containing scope.
```

These commands can be used only inside anonymous functions (see “func” (I-6.33 pg.118)).

These commands **import** an external, non-top-level variable by reference or value. “ImportByValue” creates a variable inside the anon func with the given name, and its initial value is taken from the external variable of the same name in the context the anonymous function is defined. In contrast, “ImportByRef” creates a **reference** (from inside the anon func) to the named external variable in the context where the anon func is defined.

See “TopLevel” (I-20.10 pg.329) for accessing top-level variables.

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.329), func(I-6.33 pg.118)

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
```

See “List Constructors” (III-5.2 pg.406) 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.406), IsIn(I-9.59 pg.168)

I-9.19 incr, decr

syntax

```
incr(ref X: INT)
decr(ref X: INT)
```

“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, IndentStr*

syntax

```

indent(X: OBJECT)
indent(X: OBJECT, RecursionDepth: INT)
IndentStr(X: OBJECT): STRING
IndentStr(X: OBJECT, RecursionDepth: INT): STRING

```

These functions produce an easy-to-read version of the argument by splitting it into several lines: a “LIST” or “IDEAL” is printed one element per line, a “RECORD” one field per line.

The function “*indent*” prints the result on the screen, whereas “*IndentStr*” returns it as a “STRING”, useful for passing it to other functions (*e.g.* “*print IndentStr(L) on file*”).

The second optional argument is for setting the level of recursive indentation; *e.g.* it can be useful when printing a list of records.

example

```

/**/ use QQ[x,y];
/**/ R := record[I := ideal((x-1)^3, (y+2)^4), L := 5..7];
/**/ println R;
record[I := ideal(x^3 -3*x^2 +3*x -1, y^4 +8*y^3 +24*y^2 +32*y +16), L := [5, 6, 7]]

/**/ indent(R);
record[
  I := ideal(x^3 -3*x^2 +3*x -1, y^4 +8*y^3 +24*y^2 +32*y +16),
  L := [5, 6, 7]
]

/**/ indent(R, 2);
record[
  I := ideal(
    x^3 -3*x^2 +3*x -1,
    y^4 +8*y^3 +24*y^2 +32*y +16
  ),
  L := [
    5,
    6,
    7
  ]
]

```

See Also: [format\(I-6.25 pg.115\)](#), [fold\(I-6.21 pg.113\)](#), [print on\(I-16.41 pg.261\)](#)

I-9.21 *indet*

syntax

```

indet(R: RING, N: INT): RINGELEM

```

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\(I-9.26 pg.156\)](#), [IndetIndex\(I-9.22 pg.154\)](#), [IndetName\(I-9.23 pg.154\)](#), [indets\(I-9.24 pg.155\)](#), [NumIndets\(I-14.41 pg.237\)](#), [UnivariateIndetIndex\(I-21.1 pg.335\)](#)

I-9.22 IndetIndex

syntax

```
IndetIndex(X: RINGELEM): INT
```

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\(I-9.21 pg.153\)](#), [IndetSubscripts\(I-9.26 pg.156\)](#), [IndetName\(I-9.23 pg.154\)](#), [indets\(I-9.24 pg.155\)](#), [NumIndets\(I-14.41 pg.237\)](#), [UnivariateIndetIndex\(I-21.1 pg.335\)](#)

I-9.23 IndetName

syntax

```
IndetName(X: RINGELEM): STRING
```

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\(I-9.21 pg.153\)](#), [IndetSubscripts\(I-9.26 pg.156\)](#), [IndetIndex\(I-9.22 pg.154\)](#), [IndetSymbols\(I-9.27 pg.156\)](#), [NumIndets\(I-14.41 pg.237\)](#)

I-9.24 indets

— syntax —

```

indets(R: RING): LIST
indets(R: RING, X: STRING): LIST

```

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\(I-9.21 pg.153\)](#), [IndetSubscripts\(I-9.26 pg.156\)](#), [IndetIndex\(I-9.22 pg.154\)](#), [IndetName\(I-9.23 pg.154\)](#), [NumIndets\(I-14.41 pg.237\)](#)

I-9.25 IndetsProd

— syntax —

```

IndetsProd(f: RINGELEM): RINGELEM
IndetsProd(L: LIST of RINGELEM): RINGELEM

```

This function says which indeterminates are used in “f” (or “L”). The result is the product of all indeterminates which actually appear in “f” (or “L”).

example

```

/**/ use P ::= QQ[x,y,z];
/**/ IndetsProd(x^2-z);
x*z
/**/ IndetsProd([x^2-z, y]);
x*y*z

```

See Also: `indet`([I-9.21](#) pg.153), `IndetIndex`([I-9.22](#) pg.154), `support`([I-19.62](#) pg.322)

I-9.26 IndetSubscripts

syntax

```

IndetSubscripts(X: RINGELEM): LIST

```

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.154).

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.153), `IndetIndex`([I-9.22](#) pg.154), `IndetName`([I-9.23](#) pg.154), `IndetSymbols`([I-9.27](#) pg.156), `indets`([I-9.24](#) pg.155), `NumIndets`([I-14.41](#) pg.237)

I-9.27 IndetSymbols

syntax

```

IndetSymbols(P: RING): RECORD

```

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.154)) and “indices” (as “IndetSubscripts” ([I-9.26](#) pg.156))

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\(I-9.21 pg.153\)](#), [IndetSubscripts\(I-9.26 pg.156\)](#), [IndetIndex\(I-9.22 pg.154\)](#), [IndetName\(I-9.23 pg.154\)](#), [NumIndets\(I-14.41 pg.237\)](#), [SymbolRange\(I-19.68 pg.324\)](#)

I-9.28 InducedHom

— syntax —

```
InducedHom(RmodI: RING, phi: RINGHOM): RINGHOM
```

“InducedHom(RmodI, phi)” – where “RmodI” is a QuotientRing, and “phi” is a homomorphism “R → S” (which must have “BaseRing(RmodI)” as its “domain” ([I-4.24 pg.93](#)), and whose “ker” ([I-11.1 pg.191](#)) 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\(I-4.24 pg.93\)](#), [codomain\(I-3.24 pg.65\)](#), [Composition of RINGHOM\(III-10.2 pg.427\)](#), [BaseRing\(I-2.1 pg.49\)](#), [DefiningIdeal\(I-4.5 pg.85\)](#), [NewQuotientRing\(I-14.11 pg.227\)](#), [NewFractionField\(I-14.1 pg.223\)](#), [CanonicalHom\(I-3.4 pg.58\)](#), [PolyAlgebraHom\(I-16.17 pg.251\)](#), [PolyRingHom\(I-16.18 pg.252\)](#)

I-9.29 InitialIdeal

— syntax —

```
InitialIdeal(I: IDEAL, Inds: LIST): IDEAL
```

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-12.10 pg.196](#)), “DF” ([I-4.15 pg.90](#))). 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.328)).

The implementation is based on the method of Lazard (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.328), PrimaryHilbertSeries(I-16.36 pg.259)

I-9.30 insert [OBSOLESCENT]

syntax

```
[OBSOLESCENT] insert(ref L: LIST, N: INT, E: OBJECT)
```

In most cases you should use “append” (I-1.14 pg.35) rather than this function!

This function inserts “E” into “L” as the “N”-th component. Kept just for backward compatibility, it is **strongly discouraged** for its intrinsic inefficiency.

example

```

/**/ L := ["a","b","d","e"];
/**/ insert(ref L,3,"c");
/**/ L;
["a", "b", "c", "d", "e"]

```

See Also: append(I-1.14 pg.35), remove(I-18.36 pg.284)

I-9.31 Interpolate

syntax

```
Interpolate(Points: LIST, Values: LIST): RINGELEM
```

where Points is a list of lists of coefficients representing a set of `\textbf{distinct}` points and Values is a list of the same size containing numbers from the coefficient ring.

***** 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

```
***** NOT YET IMPLEMENTED *****
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-9.32 *interreduce*

syntax

```
interreduce(ref L: LIST of RINGELEM)
```

This function is the same as “*interreduced*”, but modifies the argument “L”, and returns nothing.

example

```
/**/ use QQ[x,y,z];
/**/ L := [x^3-x*y^2+y*z, x*y, z];
/**/ interreduce(ref L); -- returns nothing, modifies L
/**/ L;
[z, x*y, x^3]
```

See Also: *interreduced*([I-9.33](#) pg.159)

I-9.33 *interreduced*

syntax

```
interreduced(L: LIST of RINGELEM): LIST of RINGELEM
```

This function returns an interreduced list “L0” of polynomials, *i.e.* each polynomial in “L0” is fully reduced with respect to the others, such that “L” and “L0” generate the same ideal.

This is generally computed in several interreducing loops. In each loop each polynomial in the list is fully reduced with respect to the others, and this process terminates when a loop finds no possible reduction. Verbosity shows how many loops are performed.

example

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

```
[z, x*y, x^3]
/**/ L; -- unmodified
[x^3 -x*y^2 +y*z, x*y, z]

/**/ SetVerbosityLevel(90);
/**/ L := [x^3*y^3 -y, x^3*y^2 +x, x^3*y +y^2];
/**/ interreduced(L);
[D1,L90,interreduced] round 1
[D1,L90,interreduced] round 2
[D1,L90,interreduced] round 3
[D1,L90,interreduced] round 4
[D1,L90,interreduced] round 5
[-2*y, x]
```

I-9.34 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
```

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-7.4 pg.122) and “HGBM” (I-8.5 pg.136).

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-7.4 pg.122), HGBM(I-8.5 pg.136), IntersectionList(I-9.35 pg.160)

I-9.35 IntersectionList

syntax

```
IntersectionList(L: LIST of LIST): LIST
IntersectionList(L: LIST of IDEAL): IDEAL
IntersectionList(L: LIST of MODULE): MODULE
```

This function returns the intersection of all elements in “L”. Generalizes “intersection” (I-9.34 pg.160).

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\(I-9.34 pg.160\)](#), [IdealOfProjectivePoints\(I-9.8 pg.147\)](#), [IdealOfPoints\(I-9.7 pg.147\)](#), [HGBM\(I-8.5 pg.136\)](#), [intersection\(I-9.34 pg.160\)](#)

I-9.36 *inverse*

— syntax —

```
inverse(X: MAT): MAT
```

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\(I-1.2 pg.31\)](#)

I-9.37 *InverseSystem*

— syntax —

```
InverseSystem(I: IDEAL, D: INT): LIST of RINGELEM
```

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\(I-4.12 pg.89\)](#), [PerpIdealOfForm\(I-16.8 pg.249\)](#)

I-9.38 *InvTotient*

— syntax —

```
InvTotient(N: INT): LIST of INT
```

Return all preimages of “N” via the “EulerTotient” ([I-5.13 pg.99](#)) function. Error if “N” is not positive. Large values of “N” may trigger an “argument too large” error.

— example —

```
/**/ InvTotient(48);
[65, 104, 105, 112, 130, 140, 144, 156, 168, 180, 210]
/**/ InvTotient(50);
[]
```

See Also: [EulerTotient\(I-5.13 pg.99\)](#)

I-9.39 IsAntiSymmetric

syntax

```
IsAntiSymmetric(M: MAT): BOOL
```

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-9.102 pg.182)

I-9.40 IsArrCentral

syntax

```
IsArrCentral(A: LIST): BOOL
```

This function tests whether the arrangement is central from the list A of hyperplanes.

example

```
/**/ use QQ[x,y];
/**/ A := [x, x-y, y];
/**/ IsArrCentral(A);
true

/**/ A := [x, x-1, y];
/**/ IsArrCentral(A);
false
```

I-9.41 IsArrFree

syntax

```
IsArrFree(Q: RINGELEM): BOOL
```

This function tests whether the arrangement A is free from its defining equation Q.

example

```
/**/ use QQ[x,y];
/**/ A := [x, x-y, y];
/**/ Q_A := product(A);
/**/ IsArrFree(Q_A);
true

/**/ use QQ[x,y,z];
/**/ A := [x, x+z, y, 2*y-3*z];
/**/ IsArrFree(product(A));
false
```

See Also: ArrDerModule(I-1.30 pg.40), ArrExponents(I-1.31 pg.41)

I-9.42 IsAtEOF

syntax

```
IsAtEOF(in: ISTREAM): BOOL
```

This function tests whether the input stream “in” has reached end of input.

example

```
/**/ Istring := OpenIString("just one line");
/**/ GetLine(Istring);
just one line
/**/ IsAtEOF(Istring);
true
```

See Also: [GetLine\(I-7.16 pg.127\)](#), [OpenIFile\(I-15.2 pg.241\)](#), [OpenIString\(I-15.3 pg.242\)](#)

I-9.43 IsCommutative

syntax

```
IsCommutative(R: RING): BOOL
```

This function tests whether the ring “R” is commutative.

example

```
/**/ IsCommutative(ZZ);
true
/**/ IsCommutative(NewWeylAlgebra(ZZ,"x,y"));
false
```

See Also: [NewWeylAlgebra\(I-14.14 pg.228\)](#)

I-9.44 IsConstant

syntax

```
IsConstant(X: RINGELEM): BOOL
```

This function tests whether the value of “X” in a polynomial ring actually lies in the image of the coefficient ring. It is equivalent to checking that the degree is 0 (or the support comprises just the PP 1).

example

```
/**/ QQx ::= QQ[x];
/**/ use QQx[y,z];
/**/ IsConstant(y+1);
false
/**/ IsConstant(x+1);
true
```

See Also: [indets\(I-9.24 pg.155\)](#)

I-9.45 IsContained

syntax

```
IsContained(A: IDEAL, B: IDEAL): BOOL
IsContained(A: MODULE, B: MODULE): BOOL
```

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\(I-9.59 pg.168\)](#), [IsSubset\(I-9.100 pg.182\)](#)

I-9.46 IsCoprime

syntax

```
IsCoprime(F: INT, G: INT): BOOL
IsCoprime(F: RINGELEM, G: RINGELEM): BOOL
```

“IsCoprime” applies to two integers says whether they are coprime. It can also be applied to two ring elements, which must belong to the same ring. It works in the following situations: - they are elements of “ZZ” - they are **power-products** in a polynomial ring - they are in a polynomial ring whose coeff ring is a field. For polynomials over the integers, please compute the “gcd” (I-7.5 pg.122), and check the result (via “IsInvertible” (I-9.68 pg.172) or “IsConstant” (I-9.44 pg.163)).

example

```
/**/ use QQ[x,y,z];
/**/ IsCoprime(x*y, y*z);
false
/**/ IsCoprime(x*y, z);
true
```

See Also: gcd(I-7.5 pg.122), IsDivisible(I-9.49 pg.165)

I-9.47 IsDefined

syntax

```
IsDefined(E)
```

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-6.8 pg.108).

example

```
/**/ IsDefined(MyVariable);
false

/**/ MyVariable := 3;
/**/ IsDefined(MyVariable);
true
```

See Also: fields(I-6.8 pg.108)

I-9.48 IsDiagonal

syntax

```
IsDiagonal(M: MAT): BOOL
```

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-9.102 pg.182), DiagMat(I-4.16 pg.90)

I-9.49 IsDivisible

syntax

```
IsDivisible(A: RINGELEM, B: RINGELEM): BOOL
IsDivisible(A: INT, B: INT): BOOL
IsDivisible(A: RINGELEM, B: INT): BOOL
IsDivisible(A: INT, B: RINGELEM): BOOL
```

This function says whether “A” is divisible by “B”; it returns “true” if so, otherwise “false”. An exception is thrown if the test is in a field; to permit testing also field elements use “IsDivisible-AllowFields”.

example

```
/**/ use QQ[x,y,z];
/**/ IsDivisible(x, x^2*(y-1));
false
/**/ IsDivisible(x^2*(y-1), x);
true
/**/ IsDivisible(9,3);
true
/**/ IsDivisible(9,2);
false
/**/ IsDivisible(9*x,2); -- 9*x is in QQ[x], so divisible by any constant!!
true
/**/ use ZZ[x];
/**/ IsDivisible(9*x,2);
false
```

See Also: FactorMultiplicity(I-6.5 pg.107), IsCoprime(I-9.46 pg.164)

I-9.50 IsElem

syntax

```
IsElem(A: RINGELEM, B: IDEAL): BOOL
IsElem(A: MODULEELEM, B: MODULE): BOOL
```

This function tests whether “A” is an element of “B”. Same as the command “IsIn” (I-9.59 pg.168), 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-9.59 pg.168)

I-9.51 IsEmpty

syntax

```
IsEmpty(L: LIST): BOOL
```

This function tests whether “L” is empty; it is equivalent to “L = []”.

example

```

/**/ L := [1,2];
/**/ IsEmpty(L);
false
/**/ L = [];
false

```

See Also: List Constructors([III-5.2](#) pg.406)

I-9.52 IsEven, IsOdd

syntax

```

IsEven(N: INT): BOOL
IsOdd(N: INT): BOOL

```

These functions test whether an integer is even or odd.

example

```

/**/ IsEven(3);
false
/**/ IsOdd(3);
true

```

See Also: IsZero([I-9.107](#) pg.184)

I-9.53 IsEvenPoly, IsOddPoly

syntax

```

IsEvenPoly(F: RINGELEM): BOOL
IsOddPoly(F: RINGELEM): BOOL

```

These functions test whether a polynomial is even or odd (as a function). Of course, most polynomials are neither even nor odd.

example

```

/**/ use QQ[x];
/**/ f := x^2+1;
/**/ IsEvenPoly(f);
true
/**/ IsOddPoly(x*f);
true
/**/ IsEvenPoly(f+x);
false

```

I-9.54 IsFactorClosed

syntax

```

IsFactorClosed(L: LIST of power products): BOOL

```

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\(I-17.4 pg.268\)](#), [LT\(I-12.24 pg.202\)](#), [ApproxPointsNBM\(I-1.16 pg.36\)](#), [IsStronglyStable\(I-9.99 pg.181\)](#)

I-9.55 IsField

syntax

```
IsField(R: RING): BOOL
```

This function tests whether a ring is a field.

example

```
/**/ IsField(ZZ);
false
/**/ IsField(QQ);
true
```

See Also: [IsFiniteField\(I-9.56 pg.167\)](#)

I-9.56 IsFiniteField

syntax

```
IsFiniteField(R: RING): BOOL
```

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\(I-9.55 pg.167\)](#), [IsPthPower\(I-9.88 pg.178\)](#), [LogCardinality\(I-12.20 pg.200\)](#), [PthRoot\(I-16.50 pg.265\)](#)

I-9.57 IsFractionField

syntax

```
IsFractionField(R: RING): BOOL
```

This function tests whether the ring “R” is a fraction field, *i.e.* is “QQ” ([I-17.1 pg.267](#)) or has been constructed with “NewFractionField” ([I-14.1 pg.223](#)).

example

```
/**/ use R := QQ[x,y]; R;
RingWithID(201, "QQ[x,y]")
/**/ IsFractionField(R);
false
/**/ K := NewFractionField(R); K;
RingWithID(202, "FractionField(RingWithID(201))")
```

```

/**/ IsFractionField(K);
true
/**/ BaseRing(K);
RingWithID(201, "QQ[x,y]")

```

See Also: [NewFractionField\(I-14.1 pg.223\)](#), [BaseRing\(I-2.1 pg.49\)](#), [RingID\(I-18.50 pg.290\)](#)

I-9.58 IsHomog

syntax

```

IsHomog(F: RINGELEM|MODULEELEM): BOOL
IsHomog(L: LIST): BOOL
IsHomog(I: IDEAL|MODULE): BOOL

```

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.

NOTE: when the grading dimension is 0 everything is homogeneous. 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\(I-4.6 pg.86\)](#), [homog\(I-8.14 pg.141\)](#), [wdeg\(I-23.1 pg.341\)](#)

I-9.59 IsIn

syntax

```

X IsIn Y          (return BOOL)

```


The semantics of “**IsIn**” is explained in the following table:

	OBJECT	IsIn	LIST	checks if the list contains the object.
	RINGELEM	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.

NOTE: if RINGELEM (or MODULEELEM) is in an IDEAL (or MODULE), a representation in terms of the generators is given by ‘‘\verb&GenRepr&’’ (\ref{GenRepr} pg.\pageref{GenRepr})

See Also: [GenRepr\(I-7.7 pg.123\)](#)

I-9.60 **IsIndet**

— syntax —

```
IsIndet(X: RINGELEM): BOOL
```

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
```

See Also: [IsIndetPosPower\(I-9.61 pg.169\)](#)

I-9.61 **IsIndetPosPower**

— syntax —

```
IsIndetPosPower(X: RINGELEM): BOOL
```

This function tests whether “X” is a (positive) power of 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];
/**/ IsIndetPosPower(x);
true
/**/ IsIndetPosPower(x/x);
false
```

See Also: [IsIndet\(I-9.60 pg.169\)](#)

I-9.62 **IsInImage**

— syntax —

```
IsInImage(phi: RINGHOM, f: RINGELEM): BOOL
```

This function checks if “f” is in the image of “phi” (better use “preimage0” ([I-16.29 pg.256](#)) 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\(I-9.101 pg.182\)](#), [preimage0\(I-16.29 pg.256\)](#)

I-9.63 IsInjective

syntax

```
IsInjective(phi: RINGHOM): BOOL
```

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\(I-11.1 pg.191\)](#), [preimage0\(I-16.29 pg.256\)](#), [IsSurjective\(I-9.101 pg.182\)](#)

I-9.64 IsInRadical

syntax

```
IsInRadical(F: RINGELEM, I: IDEAL): BOOL
IsInRadical(J: IDEAL, I: IDEAL): BOOL
```

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-13.27 pg.215), radical(I-18.1 pg.271)

I-9.65 IsInSubalgebra [OBSOLETE]

See “SubalgebraHom” (I-19.50 pg.317).

I-9.66 IsInteger

syntax

```
IsInteger(ref n: INT, f: RINGELEM): BOOL
```

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-1.50 pg.46), AsRAT(I-1.51 pg.47), IsRational(I-9.93 pg.179)

I-9.67 IsIntegralDomain

syntax

```
IsIntegralDomain(R: RING): BOOL
```

This function tests whether the ring “R” is integral.

example

```
/**/ IsIntegralDomain(ZZ);
true
/**/ IsIntegralDomain(NewZZmod(6));
false
```

See Also: IsField(I-9.55 pg.167)

I-9.68 IsInvertible

syntax

```
IsInvertible(f: RINGELEM): BOOL
```

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-9.69 IsIrred

syntax

```
IsIrred(f: RINGELEM): BOOL
```

This function tests whether the argument “f” is irreducible in “RingOf(f)”. The coefficient ring must be a field.

example

```
/**/ FFp := NewZZmod(7);
/**/ use FFp[x];
/**/ f := x^9+x+1;
/**/ IsIrred(f);
true
```

I-9.70 IsLattice

syntax

```
IsLattice(relP: LIST): BOOL
```

This function tests whether the poset P is a lattice from the list “relP” of its strict relations.

example

```
// POSET:
//      3   4
//      \ /
//      2
//      |
//      1
/**/ relP := [[1, 2], [2, 3], [2, 4]];
/**/ IsLattice(relP);
false

// P:
//      5
//      / \
//      3   4
//      |   |
//      2   |
//      \ /
//      1
```

```

/**/ relP := [[1,2], [2,3], [1,4], [3,5], [4,5]];
/**/ IsLattice(relP);
true

```

See Also: IsPosetGraded(I-9.81 pg.176)

I-9.71 IsLexSegment

syntax

```
IsLexSegment(I: IDEAL): BOOL
```

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-9.97 pg.181), IsStronglyStable(I-9.99 pg.181), LexSegmentIdeal(I-12.9 pg.196)

I-9.72 IsLRSDegenerate

syntax

```
IsLRSDegenerate(F: RINGELEM): BOOL
```

This function takes the given polynomial “F” and checks the corresponding Linear Recurrence Sequence (LRS) for degeneracy. A LRS is “n”-degenerate iff its characteristic polynomial “F” has two distinct roots whose quotient is a primitive “n”-th root of unity. Use “LRSDegeneracyOrder” (I-12.23 pg.202) to find a value for “n”.

example

```

/**/ use R := QQ[x];
/**/ IsLRSDegenerate(x^4 +3*x^3 +2*x^2 -3*x +1);
true
/**/ IsLRSDegenerate(x^3 +x +2);
false

```

See Also: cyclotomic(I-3.66 pg.81), CyclotomicIndex, CyclotomicTest(I-3.68 pg.81), IsLRSDegenerateOrder(I-9.73 pg.173), LRSDegeneracyOrder(I-12.23 pg.202)

I-9.73 IsLRSDegenerateOrder

syntax

```
IsLRSDegenerateOrder(F: RINGELEM, n: INT): BOOL
```

This function takes the given polynomial “F” and checks the corresponding Linear Recurrence Sequence (LRS) for “n”-degeneracy. A LRS is “n”-degenerate iff its characteristic polynomial “F” has two distinct roots whose quotient is a primitive “n”-th root of unity.

example

```

/**/ use R := QQ[x];
/**/ IsLRSDegenerateOrder(x^4 +3*x^3 -9*x +9, 6);
true
/**/ IsLRSDegenerateOrder(x^4 +3*x^3 -9*x +9, 3);
false

```

See Also: [cyclotomic\(I-3.66 pg.81\)](#), [CyclotomicIndex](#), [CyclotomicTest\(I-3.68 pg.81\)](#), [IsLRSDegenerate\(I-9.72 pg.173\)](#), [LRSDegeneracyOrder\(I-12.23 pg.202\)](#)

I-9.74 IsMaximal

— syntax —

```
IsMaximal(I: IDEAL): BOOL
```

This function determines whether an ideal is maximal.

— example —

```
/**/ use P := QQ[x,y,z];
/**/ IsMaximal(ideal(x^2+1, z^3+z-1, x-5*y+4*z));
true
/**/ IsMaximal(ideal(x^2+1, z^3+z-1, (x-5*y+4*z)*(y-3)));
false
```

See Also: [IsPrimary\(I-9.84 pg.177\)](#), [IsRadical\(I-9.92 pg.179\)](#)

I-9.75 IsMinusOne

— syntax —

```
IsOne(X: OBJECT): BOOL
```

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\(I-9.78 pg.175\)](#)

I-9.76 IsMultiArrFree

— syntax —

```
IsMultiArrFree(MultiA: LIST): BOOL
```

This function tests whether the multiarrangement MultiA is free.

— example —

```
/**/ use QQ[x,y];
/**/ MultiA := [[x,1], [x-y,3], [y,2]];
/**/ IsMultiArrFree(MultiA);
true

/**/ use QQ[x,y,z];
/**/ MultiA := [[x,1], [x-y,3], [z,2], [x+y-z,3]];
/**/ IsMultiArrFree(MultiA);
false
```

See Also: [MultiArrDerModule\(I-13.40 pg.220\)](#), [MultiArrExponents\(I-13.41 pg.220\)](#)

I-9.77 IsNumber [OBSOLETE]

See “IsInteger”, “IsRational”

I-9.78 IsOne

syntax

```
IsOne(X: OBJECT): BOOL
```

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);
true
/**/ IsOne(ideal(x^2, x^2-1));
true
```

See Also: IsEven, IsOdd(I-9.52 pg.166), one(I-15.1 pg.241), IsZero(I-9.107 pg.184)

I-9.79 IsPalindromic

syntax

```
IsPalindromic(R: RINGELEM): BOOL
```

This function tests whether its argument (univariate poly) is palindromic.

example

```
/**/ use P ::= QQ[x,y];
/**/ IsPalindromic(x+1);
true
/**/ IsPalindromic(y^4-3*y^2+1);
true
/**/ IsPalindromic(x-1);
false
```

See Also: CyclotomicIndex, CyclotomicTest(I-3.68 pg.81)

I-9.80 IsPolyRing

syntax

```
IsPolyRing(R: RING): BOOL
```

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\(I-9.91 pg.179\)](#), [NewPolyRing\(I-14.9 pg.226\)](#)

I-9.81 IsPosetGraded

— syntax —

```
IsPosetGraded(relP: LIST): BOOL
```

This function tests whether the poset “P” is graded from the list “relP” of its strict relations.

— example —

```
// POSET:
//      3   4
//      \ /
//      2
//      |
//      1
/**/ relP := [[1, 2], [2, 3], [2, 4]];
/**/ IsPosetGraded(relP);
true

// P:
//      5
//      / \
//      3   4
//      |   |
//      2   |
//      \ /
//      1
/**/ relP := [[1,2], [2,3], [1,4], [3,5], [4,5]];
/**/ IsPosetGraded(relP);
false
```

I-9.82 IsPositiveGrading

— syntax —

```
IsPositiveGrading(M: MAT): BOOL
```

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-8.12](#) pg.140)

I-9.83 IsPowerOf2

syntax

```
IsPowerOf2(N: INT): BOOL
```

This function determines whether an integer is a (positive) power of 2. To find out which power of 2 use the function “FloorLog2” (see entry about “FloorLog2, FloorLog10, FloorLogBase” ([I-6.18](#) pg.112)).

example

```
/**/ IsPowerOf2(2);
true
/**/ IsPowerOf2(-2);
false
```

See Also: FloorLog2, FloorLog10, FloorLogBase([I-6.18](#) pg.112)

I-9.84 IsPrimary

syntax

```
IsPrimary(I: IDEAL): BOOL
```

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: PrimaryDecomposition([I-16.33](#) pg.258), IsMaximal([I-9.74](#) pg.174), IsRadical([I-9.92](#) pg.179)

I-9.85 IsPrime

syntax

```
IsPrime(N: INT): BOOL
```

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-9.87](#) pg.178).

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-9.87](#) pg.178), NextPrime, NextProbPrime([I-14.16](#) pg.228), PrevPrime, PrevProbPrime([I-16.31](#) pg.257)

I-9.86 IsPrimitivePoly

syntax

```
IsPrimitivePoly(F: RINGELEM): BOOL
```

This function determines whether a univariate polynomial over a finite field is **primitive** (in the field theory sense): each of its roots is a group generator of extension field.

For the notion of **primitive** meaning no common factor dividing the coefficients see “**content**” (I-3.51 pg.75).

example

```
/**/ use ZZ/(3)[x];
/**/ IsPrimitivePoly(x^3+2*x+1);
true
```

See Also: [content\(I-3.51 pg.75\)](#)

I-9.87 IsProbPrime

syntax

```
IsProbPrime(N: INT): BOOL
```

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 when the argument is composite; however, if it returns “**false**”, we are certain that the argument is composite. Some people call it a **compositeness test**.

This function cannot be interrupted, so be wary of inputs larger than about “ 2^{100000} ” (which already takes a few minutes).

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\(I-9.85 pg.177\)](#), [NextPrime](#), [NextProbPrime\(I-14.16 pg.228\)](#), [PrevPrime](#), [PrevProbPrime\(I-16.31 pg.257\)](#)

I-9.88 IsPthPower

syntax

```
IsPthPower(X: RINGELEM): BOOL
```

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\(I-9.56 pg.167\)](#), [PthRoot\(I-16.50 pg.265\)](#)

I-9.89 IsQQ

syntax

```
IsQQ(R: RING): BOOL
```

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.267), RingQQ(I-18.52 pg.291), IsZZ(I-9.112 pg.186)

I-9.90 isqrt [OBSOLETE]

Renamed to “FloorSqrt” (I-6.20 pg.112).

I-9.91 IsQuotientRing

syntax

```
IsQuotientRing(R: RING): BOOL
```

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.85)

I-9.92 IsRadical

syntax

```
IsRadical(I: IDEAL): BOOL
```

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.271), IsMaximal(I-9.74 pg.174), IsPrimary(I-9.84 pg.177)

I-9.93 IsRational

syntax

```
IsRational(ref n: RAT, f: RINGELEM): BOOL
```

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-1.50 pg.46), AsRAT(I-1.51 pg.47), IsInteger(I-9.66 pg.171)

I-9.94 IsSigmaGoodPrime

syntax

```
IsSigmaGoodPrime(p: INT, I: IDEAL): BOOL
```

This function checks if “p” is a sigma-good prime for “I”, that is if it is good for modular reduction of the sigma-reduced GBasis of “I”.

See article Abbott, Bigatti, Robbiano **Ideals modulo p** (“<https://arxiv.org/abs/1801.06112>”)

example

```
/**/ use QQ[x,y,z];
/**/ I := ideal(2*x*y^2 -1, 3*x^3*y -1);
/**/ ReducedGBasis(I);
[x*y^2 -1/2, x^2 +(-2/3)*y, y^3 +(-3/4)*x]
/**/ DenSigma(I);
12
/**/ IsSigmaGoodPrime(2,I);
false
/**/ IsSigmaGoodPrime(32003,I);
true
```

See Also: CommonDenom(I-3.36 pg.70), DenSigma(I-4.9 pg.87)

I-9.95 IsSqFree

syntax

```
IsSqFree(f: RINGELEM): BOOL
IsSqFree(N: INT): BOOL
```

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.271), factor(I-6.1 pg.105), FactorINT(I-6.4 pg.106)

I-9.96 IsSquare

syntax

```
IsSquare(N: INT): BOOL
```

This function tests whether the argument is a **perfect** square.

example

```
/**/ IsSquare(99);
false
/**/ IsSquare(100);
true
```

See Also: IsSqFree(I-9.95 pg.180), FactorINT(I-6.4 pg.106)

I-9.97 IsStable

syntax

```
IsStable(I: IDEAL): BOOL
```

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-9.71 pg.173), IsStronglyStable(I-9.99 pg.181), LexSegmentIdeal(I-12.9 pg.196)

I-9.98 IsStdGraded

syntax

```
IsStdGraded(P: RING): BOOL
```

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-14.9 pg.226), wdeg(I-23.1 pg.341)

I-9.99 IsStronglyStable

syntax

```
IsStronglyStable(I: IDEAL): BOOL
```

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\(I-9.71 pg.173\)](#), [IsStable\(I-9.97 pg.181\)](#)

I-9.100 IsSubset

— syntax —

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

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\(I-9.45 pg.163\)](#), [IsIn\(I-9.59 pg.168\)](#), [EqSet\(I-5.9 pg.98\)](#), [MakeSet\(I-13.3 pg.206\)](#), [subsets\(I-19.58 pg.320\)](#)

I-9.101 IsSurjective

— syntax —

```
IsSurjective(phi: RINGHOM): BOOL
```

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\(I-11.1 pg.191\)](#), [IsInjective\(I-9.63 pg.170\)](#), [preimage0\(I-16.29 pg.256\)](#)

I-9.102 IsSymmetric

— syntax —

```
IsSymmetric(M: MAT): BOOL
```

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-9.39 pg.162)

I-9.103 IsTerm

syntax

```
IsTerm(X: RINGELEM|MODULEELEM): BOOL
```

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-9.104 IsTermOrdering

syntax

```
IsTermOrdering(M: MAT): BOOL
```

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-14.9 pg.226), OrdMat(I-15.11 pg.245)

I-9.105 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]
```

***** 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

```
***** NOT YET IMPLEMENTED *****
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-9.106 IsTrueGCDDomain

syntax

```
IsTrueGCDDomain(R: RING): BOOL
```

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-9.55 pg.167)

I-9.107 IsZero

syntax

```
IsZero(X: OBJECT): BOOL
```

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-9.52 pg.166), IsOne(I-9.78 pg.175), zero(I-25.1 pg.345), ZeroMat(I-25.2 pg.345)

I-9.108 IsZeroCol, IsZeroRow

syntax

```

IsZeroCol(M: MAT, N: INT): BOOL
IsZeroRow(M: MAT, N: INT): BOOL

```

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-9.107 pg.184), ZeroMat(I-25.2 pg.345)

I-9.109 IsZeroDet

syntax

```

IsZeroDet(M: MAT): BOOL

```

This function tests whether the determinant of “M” is zero; it may sometimes be faster than computing the determinant.

example

```

/**/ M := matrix([[1,2], [3,4]]);
/**/ IsZeroDet(M);
false

```

See Also: det(I-4.14 pg.89), rk(I-18.57 pg.292), IsZero(I-9.107 pg.184)

I-9.110 IsZeroDim

syntax

```

IsZeroDim(I: IDEAL): BOOL

```

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: IdealOfPoints(I-9.7 pg.147), IdealOfProjectivePoints(I-9.8 pg.147), dim(I-4.19 pg.91)

I-9.111 IsZeroDivisor

syntax

```
IsZeroDivisor(X: RINGELEM): BOOL
```

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-3.34 pg.69)

I-9.112 IsZZ

syntax

```
IsZZ(R: RING): BOOL
```

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.346), RingZZ(I-18.56 pg.292), IsQQ(I-9.89 pg.179)

I-9.113 It

syntax

```
It
```

“It” is a top-level **system variable** containing the last result computed but not assigned. It is the CoCoA equivalent to “last” in GAP.

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-16.45 pg.263) 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\(I-16.40 pg.260\)](#), [println\(I-16.45 pg.263\)](#), [Evaluation and Assignment\(II-5 pg.365\)](#)

Chapter I-10

J

I-10.1 JacobianMat

[syntax](#)

```
JacobianMat(L: LIST of RINGELEM): MAT  
JacobianMat(L: LIST of RINGELEM, X: LIST of indets): MAT
```

This function returns the Jacobian matrix of the polynomials in the non empty list “L” with respect to all the indeterminates of the ring of “L” or, if specified, to the indeterminates in “X”.

[example](#)

```
/**/ use R := QQ[x,y];  
/**/ L := [x-y, x^2-y, x^3-y^2];  
/**/ JacobianMat(L);  
matrix( /*RingWithID(26, "QQ[x,y]")*/  
  [[1, -1],  
   [2*x, -1],  
   [3*x^2, -2*y]])  
/**/ JacobianMat(L,[y]);  
matrix( /*RingWithID(26, "QQ[x,y]")*/  
  [[-1],  
   [-1],  
   [-2*y]])
```

I-10.2 JanetBasis

[syntax](#)

```
JanetBasis(I: IDEAL): LIST of RINGELEM
```

This function returns the Janet basis of the ideal “I” as a list of polynomials..

Originally written by Mario Albert.

[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.121)

Chapter I-11

K

I-11.1 ker

syntax

```
ker(phi: RINGHOM): IDEAL
```

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*w, y*z -x*w, y^2 -x*z)

/**/ SmodJ := NewQuotientRing(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, "x+y");
/**/ ker(InducedHom(RmodI, psi));
ideal((z +w), (y -w))
```

See Also: [preimage0\(I-16.29 pg.256\)](#), [IsInjective\(I-9.63 pg.170\)](#), [IsSurjective\(I-9.101 pg.182\)](#), [implicit\(I-9.13 pg.150\)](#)

I-11.2 KroneckerProd

syntax

```
KroneckerProd(M: MATRIX, N: MATRIX): MAT
```

This function returns the Kronecker (tensor) product of two matrices.

example

```
/**/ use R := QQ[x,y,z,w];
/**/ KroneckerProd(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-11.3 KroneckerSymbol

syntax

```
KroneckerSymbol(R: INT, M: INT): INT
```

This function returns the kronecker symbol of “R” modulo “M”. If “M” is prime then this value is 0 when “R” is divisible by “M”, and otherwise is 1 when “R” is a square modulo “M” (*i.e.* a quadratic residue), and -1 when it is not a square.

example

```
/**/ KroneckerSymbol(1,3);
1
/**/ KroneckerSymbol(-1,7);
-1
```

See Also: [IsSquare\(I-9.96 pg.181\)](#)

Chapter I-12

L

I-12.1 LaguerrePoly

— syntax —

```
LaguerrePoly(N: INT, X: RINGELEM): RINGELEM
```

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-3.15](#) pg.62), HermitePoly([I-8.4](#) pg.136)

I-12.2 last

— syntax —

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

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**.

NOTE: The CoCoA equivalent to GAP “**last**” is the variable “**It**” ([I-9.113](#) pg.186).

— example —

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

/**/ last(L,3);
[3, 4, 5]
```

See Also: first([I-6.9](#) pg.108), tail([I-20.3](#) pg.328), It([I-9.113](#) pg.186)

I-12.3 latex

— syntax —

```
latex(X: OBJECT): STRING
```

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\(I-6.25 pg.115\)](#), [sprint\(I-19.35 pg.311\)](#)

I-12.4 LawrenceMat

syntax

```
LawrenceMat(M: MATRIX): MAT
```

This function returns the Lawrence lifting of the matrix “M”, defined as follows:

$$\begin{pmatrix} | & M & 0 & | \\ | & I & I & | \end{pmatrix}$$

example

```
/**/ LawrenceMat(mat([[1,2,3],[4,5,6]]));
matrix(QQ,
[[1, 2, 3, 0, 0, 0],
[4, 5, 6, 0, 0, 0],
[1, 0, 0, 1, 0, 0],
[0, 1, 0, 0, 1, 0],
[0, 0, 1, 0, 0, 1]])
```

I-12.5 LC

syntax

```
LC(F: RINGELEM|MODULEELEM): RINGELEM
```

This function returns the leading coefficient of “F”, as determined by the term-ordering of the ring to which “F” belongs. It is an error if “F” is zero.

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: [ConstantCoeff\(I-3.50 pg.75\)](#), [coefficients\(I-3.27 pg.66\)](#), [CoeffOfTerm\(I-3.31 pg.68\)](#), [LT\(I-12.24 pg.202\)](#)

I-12.6 lcm

syntax

```

lcm(M: INT, N: INT): INT
lcm(L: LIST of INT): INT

lcm(F: RINGELEM, G: RINGELEM): RINGELEM
lcm(L: LIST of RINGELEM): RINGELEM

```

This function returns the least common multiple of its arguments, 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\(I-4.22 pg.92\)](#), [mod\(I-13.29 pg.215\)](#), [gcd\(I-7.5 pg.122\)](#)

I-12.7 len

syntax

```

len(S: STRING): INT
len(L: LIST): INT

```

This function returns the **length** of a string or a list.

NOTE: previously “len” could be applied to other types too; this is no longer supported: see “NumCompts” ([I-14.39 pg.237](#)) for module elements, “NumRows” ([I-14.44 pg.238](#)) for matrices, “NumTerms” ([I-14.45 pg.238](#)) for polynomials.

example

```

/**/ len( [2,3,4] );
3
/**/ len( "string" );
6

```

See Also: [count\(I-3.59 pg.78\)](#), [NumCompts\(I-14.39 pg.237\)](#), [NumRows\(I-14.44 pg.238\)](#), [NumTerms\(I-14.45 pg.238\)](#)

I-12.8 LexMat

syntax

```
LexMat(N: INT): MAT
```

This function returns 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\(III-9.5 pg.422\)](#), [StdDegLexMat\(I-19.46 pg.316\)](#), [StdDegRevLexMat\(I-19.47 pg.316\)](#), [RevLexMat\(I-18.45 pg.287\)](#), [XelMat\(I-24.1 pg.343\)](#)

I-12.9 LexSegmentIdeal

syntax

```
LexSegmentIdeal(L: LIST of power-products): IDEAL
LexSegmentIdeal(I: IDEAL): IDEAL
```

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\(I-9.71 pg.173\)](#), [StableIdeal\(I-19.39 pg.313\)](#), [StronglyStableIdeal\(I-19.48 pg.316\)](#)

I-12.10 LF

syntax

```
LF(I: IDEAL): IDEAL
LF(F: RINGELEM): RINGELEM
```

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-7.32 pg.130](#)) of the polynomial ring is 0 (use “DF” ([I-4.15 pg.90](#)) 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-7.32 pg.130) 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-4.15 pg.90), IsHomog(I-9.58 pg.168), LC(I-12.5 pg.194), LM(I-12.18 pg.200), LPP(I-12.22 pg.201), LT(I-12.24 pg.202)

I-12.11 LinearSimplify

syntax

```
LinearSimplify(F: RINGELEM): RECORD
```

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-12.12 LinKer

syntax

```
LinKer(M: MAT): MAT
```

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). See “LinKerZZ” (I-12.15 pg.199) for computing a ZZ-basis for the kernel of “M”.

The output as it was given by CoCoA-4 (the basis of the ker) is now given by “LinKerBasis” (I-12.13 pg.198). See also “HilbertBasisKer” (I-8.7 pg.137).

To clarify: if “K := LinKer(M);” then “M*K” is a zero matrix.

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]])

/**/ IsZero(M*It);
true
```

See Also: LinKerBasis(I-12.13 pg.198), LinKerZZ(I-12.15 pg.199), LinSolve(I-12.17 pg.199), HilbertBasisKer(I-8.7 pg.137)

I-12.13 LinKerBasis

— syntax —

```
LinKerBasis(M: MAT): LIST of RINGELEM
LinKerBasis(L: LIST): LIST of RINGELEM
```

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”.

NOTE: 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.

NOTE: this function works only on matrices whose entries are in a field (from version CoCoA-5.0.3). See “LinKerZZ” (I-12.15 pg.199) for computing a ZZ-basis for the kernel of “M”.

— 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-12.12 pg.197), LinKerZZ(I-12.15 pg.199), LinSolve(I-12.17 pg.199)

I-12.14 LinKerModP [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-12.12 pg.197) and “LinKerBasis” (I-12.13 pg.198).

— 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(NewZZmod(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-12.12 pg.197), LinKerBasis(I-12.13 pg.198), LinSolve(I-12.17 pg.199)

I-12.15 LinKerZZ

syntax

```
LinKerZZ(M: MAT): MAT
```

This function expects a matrix “M” with integer or rational entries, and returns a matrix of integers whose columns represent a “ZZ”-module basis for the integer vectors in 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.

To clarify: if “K := LinKerZZ(M);” then “M*K” is a zero matrix.

example

```

/**/ LinKerZZ(RowMat([1,1/2,1/3]));
matrix(ZZ,
  [[1, 0],
   [0, 2],
   [-3, -3]])

```

See Also: LinKer(I-12.12 pg.197), LinKerBasis(I-12.13 pg.198), LinSolve(I-12.17 pg.199)

I-12.16 LinSol [OBSOLETE]

Renamed to “LinSolve” (I-12.17 pg.199).

I-12.17 LinSolve

syntax

```
LinSolve(M: MAT, RHS: MAT): MAT
```

This function finds a solution “X” to the matrix equation “M*X = 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 “LinKerBasis” (I-12.13 pg.198).

NOTE: an easy way of converting a list into a column matrix (for the second argument) is to use the function “ColMat” (I-3.33 pg.69).

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-3.33](#) pg.69), LinKer([I-12.12](#) pg.197), LinKerBasis([I-12.13](#) pg.198)

I-12.18 LM

— syntax —

```

LM(X: RINGELEM): RINGELEM
LM(X: MODULEELEM): MODULEELEM

```

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-12.24](#) pg.202).

— example —

```

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

```

See Also: LC([I-12.5](#) pg.194), LF([I-12.10](#) pg.196), LPP([I-12.22](#) pg.201), LT([I-12.24](#) pg.202)

I-12.19 log [OBSOLESCENT]

Renamed to “exponents” ([I-5.18](#) pg.101).

I-12.20 LogCardinality

— syntax —

```

LogCardinality(Fp: RING): INT

```

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-9.56](#) pg.167), characteristic([I-3.13](#) pg.61)

I-12.21 LPosn

syntax

```
LPosn(V: MODULEELEM): INT
```

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-12.10 pg.196), LM(I-12.18 pg.200), LPP(I-12.22 pg.201), LT(I-12.24 pg.202)

I-12.22 LPP

syntax

```
LPP(X: RINGELEM): RINGELEM
LPP(X: MODULEELEM): RINGELEM
```

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-12.5 pg.194), LF(I-12.10 pg.196), LM(I-12.18 pg.200), LPosn(I-12.21 pg.201), LT(I-12.24 pg.202)

I-12.23 LRSDegeneracyOrder

syntax

```
LRSDegeneracyOrder(f: RINGELEM):  INT
LRSDegeneracyOrders(f: RINGELEM):  LIST of INT
```

If “f” is a univariate polynomial with rational coefficients “LRSDegeneracyOrder” returns the least LRS-degeneracy order of “f”, or 0 to mean that “f” is not LRS-degenerate. Recall “f” is “k”-LRS-degenerate iff it has two distinct roots whose ratio is a primitive “k”-th root of unity. The function “LRSDegeneracyOrders” returns a list of all “k”; this function may take a long time.

Note: some polynomials are LRS-degenerate for several orders, for instance “cyclotomic(3,x)*cyclotomic(5,x)”.

In Cipu, Diouf, Mignotte the property was called just **degenerate**.

example

```
/**/ use R := QQ[x];
/**/ LRSDegeneracyOrder(x^6+3*x^5+6*x^4+6*x^3+9*x^2+9*x+3);
18

/**/ LRSDegeneracyOrder(x^2+x+2);
0
```

See Also: IsLRSDegenerate(I-9.72 pg.173), IsLRSDegenerateOrder(I-9.73 pg.173), cyclotomic(I-3.66 pg.81), CyclotomicIndex, CyclotomicTest(I-3.68 pg.81)

I-12.24 LT

syntax

```
LT(I: RINGELEM):  RINGELEM
LT(I: IDEAL):     IDEAL
LT(I: MODULEELEM): MODULEELEM
LT(I: MODULE):    MODULE
```

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-12.18 pg.200).

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-12.18 pg.200).

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-12.5](#) pg.[194](#)), LF([I-12.10](#) pg.[196](#)), LM([I-12.18](#) pg.[200](#)), LPP([I-12.22](#) pg.[201](#)), Module Orderings([III-9.6](#) pg.[423](#)), Term Orderings([III-9.5](#) pg.[422](#))

Chapter I-13

M

I-13.1 MakeCheck

— syntax —

```
MakeCheck()
```

***** 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 —

```
***** NOT YET IMPLEMENTED *****  
MakeCheck();
```

I-13.2 MakeMatByRows, MakeMatByCols

— syntax —

```
MakeMatByRows(R: INT, C: INT, L: LIST): MAT  
MakeMatByCols(R: INT, C: INT, L: LIST): MAT
```

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.53), matrix([I-13.11](#) pg.209), NewMat([I-14.7](#) pg.225), ColMat([I-3.33](#) pg.69), RowMat([I-18.62](#) pg.294), DiagMat([I-4.16](#) pg.90), ConcatHor([I-3.44](#) pg.72), ConcatVer([I-3.48](#) pg.74)

I-13.3 MakeSet

— syntax —

```
MakeSet(L: LIST): LIST
```

This function returns a list obtained by removing duplicates from “L”.

NOTE: to test two sets for equality use the function “EqSet” (I-5.9 pg.98) instead of a normal equality test (because the latter yields false if the elements are in a different order).

— example —

```
/**/ MakeSet([2,2,2,1,2,1,1,3,3]);
[2, 1, 3]
/**/ MakeSet([2,2,2,1,2,1,1,3,3]) = [1,2,3];
false
/**/ EqSet([2,2,2,1,2,1,1,3,3], [1,2,3]);
true
```

See Also: EqSet(I-5.9 pg.98), intersection(I-9.34 pg.160), IntersectionList(I-9.35 pg.160), remove(I-18.36 pg.284)

I-13.4 MakeTerm

— syntax —

```
MakeTerm(R: RING, L: LIST of INT): RINGELEM
```

This function returns the power-product in “R” whose list of exponents is “L”. It is the inverse of “exponents” (I-5.18 pg.101). 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-5.18 pg.101)

I-13.5 MakeTermOrd [OBSOLESCE]

Renamed to “MakeTermOrdMat” (I-13.6 pg.206) from version 5.3.4.

I-13.6 MakeTermOrdMat

— syntax —

```
MakeTermOrdMat(DegVec: MAT): MAT
MakeTermOrdMat(DegVec: MAT, GrDim: INT): MAT
```

This function returns a (square) matrix of integers defining a term ordering; the output matrix is built from “DegVec”. The first “GrDim” rows are left unchanged; if “GrDim” is not given then it is taken to be 0.

The input matrix “DegVec” must have rational entries; the first “GrDim” rows must have non-negative integer entries, and they must be linearly independent.

If “DegVec” 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

```

/**/ DegVec := matrix([[1,2,3,4]]);
/**/ MakeTermOrdMat(DegVec);
[[1, 2, 3, 4],
 [0, 0, 0, -1],
 [0, 0, -1, 0],
 [0, -1, 0, 0]]

/**/ MakeTermOrdMat(RowMat([1,2,0,0]));
[[1, 2, 0, 0],
 [0, 0, 1, 1],
 [0, 0, 0, -1],
 [0, -1, 0, 0]]

/**/ MakeTermOrdMat(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-12.8](#) pg.196), RevLexMat([I-18.45](#) pg.287), StdDegLexMat([I-19.46](#) pg.316), StdDegRevLexMat([I-19.47](#) pg.316), NewPolyRing([I-14.9](#) pg.226)

I-13.7 MantissaAndExponent10

syntax

```
MantissaAndExponent10(X: INT|RAT, Prec: INT): RECORD
```

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 $10^{(Prec-1)} \leq |M| < 10^P$

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-1.50](#) pg.46), AsRAT([I-1.51](#) pg.47), DecimalStr([I-4.3](#) pg.83), FloatApprox([I-6.15](#) pg.110), FloatStr([I-6.16](#) pg.111), FloorLog2, FloorLog10, FloorLogBase([I-6.18](#) pg.112), MantissaAndExponent2([I-13.8](#) pg.207), ScientificStr([I-19.5](#) pg.299)

I-13.8 MantissaAndExponent2

syntax

```

MantissaAndExponent2(X: INT|RAT, Prec: INT): RECORD
MantissaAndExponent2(X: RINGELEM): RECORD

```

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 “TwinFloat” 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-1.50 pg.46), AsRAT(I-1.51 pg.47), FloatApprox(I-6.15 pg.110), FloorLog2, FloorLog10, FloorLogBase(I-6.18 pg.112), MantissaAndExponent10(I-13.7 pg.207), NewRingTwinFloat(I-14.13 pg.227)

I-13.9 Manual

syntax

```
? key
?? key
```

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

```
/**/ ?approx
-----[ 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-13.10 MapDown [OBSOLETE]

See “AsINT” (I-1.50 pg.46), “AsRAT” (I-1.51 pg.47).

I-13.11 matrix

— syntax —

```
matrix(LofL: LIST): MAT
matrix(R: RING, LofL: LIST): MAT
matrix(R: RING, M: MAT): MAT
```

These functions construct a matrix. The 2nd and 3rd forms produce a matrix over the specified ring “R” (or error if the entries in “LofL” or “M” cannot be mapped into “R”).

In the first form, where the ring is not specified, CoCoA **guesses** the ring: if all elements of “LofL” are INT or RAT the resulting matrix is in “QQ” (otherwise it picks the biggest ring from the given elements and uses that).

The third form is equivalent to “CanonicalHom(RingOf(M),R)(M)” (See “CanonicalHom” (I-3.4 pg.58)).

“LofL” must be a **rectangular** LIST of LIST of RINGELEMENTS or INTs or RATs.

— 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); // promotion mapping: ZZ to QQ
matrix(QQ,
  [[1, 0],
   [0, 1]])
```

See Also: NewMat(I-14.7 pg.225), ColMat(I-3.33 pg.69), RowMat(I-18.62 pg.294), DiagMat(I-4.16 pg.90), MakeMatByRows, MakeMatByCols(I-13.2 pg.205), ConcatHor(I-3.44 pg.72), ConcatVer(I-3.48 pg.74), BlockMat(I-2.9 pg.53), Commands and Functions for MAT(III-13.2 pg.439)

I-13.12 max

— syntax —

```
max(E_1: OBJECT,...,E_n: OBJECT): OBJECT
max(L: LIST): OBJECT
```

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\(I-13.16 pg.211\)](#), [max\(I-13.12 pg.209\)](#), [MaxBy\(I-13.13 pg.210\)](#), [Relational Operators\(II-4.3 pg.362\)](#)

I-13.13 MaxBy

— syntax —

```
MaxBy(L: LIST, LessThanFunc: FUNCTION)
```

This function returns a maximum of the elements of the list “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: for example, when comparing two polynomials by their “LPP” only.

— 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^2*y -y -3, x^5 -1, x^5 -y^2 +3];
/**/ MaxBy(L, ByLPP);
x^5 -1

```

See Also: [MinBy\(I-13.17 pg.211\)](#), [func\(I-6.33 pg.118\)](#), [SortBy\(I-19.29 pg.308\)](#), [SortedBy\(I-19.31 pg.310\)](#), [TopLevel\(I-20.10 pg.329\)](#)

I-13.14 MaxChains

— syntax —

```
MaxChains(relP: LIST): LIST of LIST
```

This function returns the list of all maximal chains from the list “relP” of the strict relations in a poset.

example

```
// POSET:
//      3   4
//      \ /
//      2
//      |
//      1
/**/ relP := [[1, 2], [2, 3], [2, 4]];
/**/ MaxChains(relP);
[[1, 2, 3], [1, 2, 4]]
```

I-13.15 MayerVietorisTreeN1

syntax

```
MayerVietorisTreeN1(I: IDEAL): INT
```

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: Frobbys (II-10.4 pg.381)

I-13.16 min

syntax

```
min(E_1: OBJECT,...,E_n: OBJECT): OBJECT
min(L: LIST): OBJECT
```

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-13.12 pg.209).

example

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

See Also: max(I-13.12 pg.209), min(I-13.16 pg.211), MinBy(I-13.17 pg.211), MaxBy(I-13.13 pg.210), Relational Operators(II-4.3 pg.362)

I-13.17 MinBy

syntax

```
MinBy(L: LIST, LessThanFunc: FUNCTION): OBJECT
```

This function returns a minimum of the elements of the list “L” with respect to the comparisons made by “LessThanFunc”. (see “MaxBy” (I-13.13 pg.210) 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-13.13 pg.210), func(I-6.33 pg.118), SortBy(I-19.29 pg.308), SortedBy(I-19.31 pg.310), TopLevel(I-20.10 pg.329)

I-13.18 MinGBoverZZ [PROTOTYPE]

syntax

```
MinGBoverZZ(L: LIST of RINGELEM): LIST
```

This function returns a minimal Groebner basis of the ideal generated by “L”, which must be a list of polynomials from a ring with coefficient ring “ZZ” (I-25.4 pg.346). **This is just a prototype:** exact semantics, and interface are likely to change!

example

```
/**/ use ZZ[x,y];
/**/ MinGBoverZZ([x^2+y^2+2, x^3-y^3, x*y-2*x-3]);
[1460, 4*y +308, x +499*y +500, y^2 -6*x +4*y -3]
```

See Also: GBasis(I-7.1 pg.121)

I-13.19 MinGens

syntax

```
MinGens(M: IDEAL|MODULE): LIST
```

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-13.28 pg.215).

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.146), SubmoduleOfMinGens(I-19.57 pg.319), MinSubsetOfGens(I-13.28 pg.215)

I-13.20 *MinGensGeneral* [OBSOLESCENT]

Renamed to “*MinSubsetOfGens*” (I-13.28 pg.215) (more expressive).

I-13.21 *minimalize* [OBSOLESCENT]

Essentially replaced by “*IdealOfMinGens*” (I-9.6 pg.146) and “*SubmoduleOfMinGens*” (I-19.57 pg.319).

See Also: *MinGens*(I-13.19 pg.212), *MinSubsetOfGens*(I-13.28 pg.215)

I-13.22 *minimalized* [OBSOLESCENT]

Renamed: see “*IdealOfMinGens*” (I-9.6 pg.146) and “*SubmoduleOfMinGens*” (I-19.57 pg.319).

I-13.23 *MinimalPresentation*

syntax

```
MinimalPresentation(Q:TAGGED): TAGGED
```

where Q is a quotient module of the type R^s/M

***** 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

```
***** NOT YET IMPLEMENTED *****
use R := QQ[x,y,z];
MinimalPresentation(R^3/Module([[x,1,1], [x,2,2]]));
R^2/Module([[x, 0]])
-----
```

I-13.24 *minors*

syntax

```
minors(M: MAT, N: INT): LIST
```

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-4.14 pg.89), *AdjacentMinors*(I-1.3 pg.32)

I-13.25 *MinPoly*

syntax

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

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-3.14 pg.62).

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-3.14 pg.62)

I-13.26 MinPolyQuot

syntax

```
MinPolyQuot(f: RINGELEM, I: IDEAL, z: RINGELEM): RINGELEM
MinPolyQuot(f: RINGELEM, I: IDEAL, z: RINGELEM, VerificationLevel: INT): 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
```

These 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>”)

When coefficients are in “QQ” (I-17.1 pg.267), “MinPolyQuot” uses modular methods. If called with “VerificationLevel” equal to $n \geq 0$ the result polynomial is verified over “FF_p”, with n different primes (if n=0, not verified).

Verbosity: 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]]));
/**/ MinPolyQuot(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);
/**/ MinPolyQuot(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);
/**/ MinPolyQuot(f, I, indet(RingQQt(1),1));
t^6 -9*t^4 -10*t^3 +27*t^2 -90*t -2

```

See Also: [MinPoly\(I-13.25 pg.213\)](#)

I-13.27 *MinPowerInIdeal*

syntax

```
MinPowerInIdeal(F: RINGELEM, I: IDEAL): INT
```

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\(I-9.64 pg.170\)](#), [radical\(I-18.1 pg.271\)](#)

I-13.28 *MinSubsetOfGens*

syntax

```
MinSubsetOfGens(M: IDEAL|MODULE): LIST
```

This function returns a subset “S” of “gens(M)” which is **minimal** in the sense that no proper subset of “S” generates “M”.

NOTE: there no guarantee that “S” is the smallest possible subset which generates “M”.

If “M” is a homogeneous ideal or module, the function “MinGens” ([I-13.19 pg.212](#)) 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\(I-13.19 pg.212\)](#), [IdealOfMinGens\(I-9.6 pg.146\)](#), [SubmoduleOfMinGens\(I-19.57 pg.319\)](#)

I-13.29 *mod*

syntax

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

This function computes the remainder of integer division of “*N*” by “*D*”; the **remainder is zero or has the same sign as “*N*”**. If “*R*” is the remainder then “ $\text{abs}(R) < \text{abs}(D)$ ”.

If we set “ $Q = \text{div}(N, D)$ ” and “ $R = \text{mod}(N, D)$ ” then $N = Q * D + R$.

NOTE: for polynomials use “NR” (I-14.34 pg.235) (remainder), “DivAlg” (I-4.23 pg.93) (quotients and remainder), “IsIn” (I-9.59 pg.168) (ideal membership).

example

```

/**/ mod(10,3);
1
/**/ mod(-10,3);
-1
/**/ // How to test if two numbers are equivalent modulo M
/**/ A := 10; B := -5; M := 3;
/**/ mod(A-B,M) = 0;      // CORRECT!
true
/**/ mod(A,M) = mod(B,M); // WRONG!! Do not do this!
false

```

See Also: [div\(I-4.22 pg.92\)](#), [DivAlg\(I-4.23 pg.93\)](#), [IsIn\(I-9.59 pg.168\)](#), [NF\(I-14.17 pg.229\)](#), [NR\(I-14.34 pg.235\)](#)

I-13.30 Mod2Rat [OBSOLETE]

See “RatReconstructWithBounds” (I-18.22 pg.278).

I-13.31 ModuleElem

syntax

```
ModuleElem(M: MODULE, L: LIST): MODULEELEM
```

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\(I-19.56 pg.319\)](#)

I-13.32 ModuleOf

syntax

```
ModuleOf(V: MODULEELEM): MODULE
ModuleOf(M: MODULE): MODULE
```

The first form gives the free module to which “*V*” belongs; the second form gives the free module containing “*M*”.

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-19.55](#) pg.319)

I-13.33 moebius

syntax

```
moebius(relP: LIST): LIST of INT
```

This function returns a list “L” such that “L[i]” is the moebius function value of the node “i” in the poset from the list “relP” of the strict relations in the poset.

example

```

// POSET:
//      3   4
//      \ /
//      2
//      |
//      1
/**/ relP := [[1, 2], [2, 3], [2, 4]];
/**/ moebius(relP);
[1, -1, 0, 0]

```

I-13.34 MoebiusFn

syntax

```
MoebiusFn(N: INT): INT
```

This function returns the value of the Moebius function μ at the point “N”. It represents the sum of the primitive “N”-th roots of unity and is always -1, 0, or 1.

example

```

/**/ MoebiusFn(5);
-1
/**/ MoebiusFn(8);
0
/**/ MoebiusFn(10);
1

```

I-13.35 monic

syntax

```
monic(F: RINGELEM): RINGELEM
```

This function returns “F” divided by its leading coefficient (see “LC” (I-12.5 pg.194)). If “F” is zero, it throws an error.

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: cannot invert 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-12.5 pg.194), prim(I-16.32 pg.257), CommonDenom(I-3.36 pg.70)

I-13.36 monomials

syntax

```
monomials(F: RINGELEM|MODULEELEM): LIST
```

This function returns the list of monomials of F. The function “support” (I-19.62 pg.322) 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-3.27 pg.66), support(I-19.62 pg.322)

I-13.37 MonsInIdeal

syntax

```
MonsInIdeal(I: IDEAL): IDEAL
```

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

This function returns the ideal generated by all monomials in the original ideal I.

example

```

***** NOT YET IMPLEMENTED *****
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-13.38 MSatLinSolve

— syntax —

`MSatLinSolve(Env: RECORD): MATRIX`

This function calls the “MathSAT” ([II-10.5 pg.381](#)) implementation of the simplex method. The argument “Env” may contain the following fields: “leq0”, “neq0”, “eq0”, “lt0”; each field must have as value a matrix with rational entries, where each row of the matrix corresponds to a linear condition. The matrices must all have the same number of columns.

The work for this communication has been partly supported by the European Union Horizon 2020 research and innovation programme under grant agreement No H2020-FETOPEN-2015-CSA 712689: SC-square “<http://www.sc-square.org>”.

— example —

```

/**/ LinSys :=
/**/   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
/**/                               ];
/**/ soln := MSatLinSolve(LinSys); soln;
matrix(QQ,
  [[1],
   [4/5],
   [-11/5]])
/**/ // verify:
/**/ soln1 := ConcatVer(soln, matrix([[1]])); //-->  [[1], [4/5], [-11/5], [1]]
/**/ LinSys.leq0 * soln1; // is <= 0
matrix(QQ,
  [[0],
   [0]])
/**/ LinSys.neq0 * soln1; // is <> 0
matrix(QQ,
  [[1]])
/**/ // now we add new constraints:
/**/ LinSys.eq0 := RowMat([1,1,0, 4]); //   x +y +4 = 0
/**/ LinSys.lt0 := RowMat([0,1,0, 0]); //   y   < 0
/**/ soln := MSatLinSolve(LinSys); soln;
matrix(QQ,
  [[-2],
   [-2],
   [-2/7]])
/**/ // verify:
/**/ soln1 := ConcatVer(soln, RowMat([1])); //-->  [[-2], [-2], [-2/7], [1]]
/**/ LinSys.leq0 * soln1; // <= 0
matrix(QQ,
  [[-20/7],
   [-36]])
/**/ LinSys.neq0 * soln1; // <> 0
matrix(QQ,
  [[-2]])
/**/ LinSys.eq0 * soln1; // = 0
matrix(QQ,
  [[0]])
/**/ LinSys.lt0 * soln1; // < 0
matrix(QQ,
  [[-2]])

```

See Also: [LinSolve\(I-12.17 pg.199\)](#), [MathSAT\(II-10.5 pg.381\)](#)

I-13.39 MultiArrDerMod [OBSOLESCENT]

Renamed to “MultiArrDerModule” (I-13.40 pg.220).

I-13.40 MultiArrDerModule

syntax

```
MultiArrDerModule(MultiA: LIST): MAT
```

This function returns the matrix whose columns are a set of generators of the module of logarithmic derivations of a multiarrangement of hyperplanes.

example

```
/**/ use QQ[x,y];
/**/ MultiA := [[x,1], [x-y,3], [y,2]];
/**/ MultiArrDerModule(MultiA);
matrix( /*RingWithID(2200, "QQ[x,y]")*/
  [[x*y^2, x^3 -3*x^2*y],
   [x*y^2, -3*x*y^2 +y^3]])
```

See Also: MultiArrExponents(I-13.41 pg.220), IsMultiArrFree(I-9.76 pg.174)

I-13.41 MultiArrExponents

syntax

```
MultiArrExponents(MultiA: LIST): LIST
```

This function returns the list of exponents of a free multiarrangement of hyperplanes MultiA.

example

```
/**/ use QQ[x,y];
/**/ MultiA := [[x,1], [x-y,3], [y,2]]; -- free
/**/ MultiArrExponents(MultiA);
[3, 3]

/**/ use QQ[x,y,z];
/**/ MultiA := [[x,1], [x-y,3], [z,2], [x+y-z,3]]; -- not free
/**/ IsMultiArrFree(MultiA); --> false
-- /**/ MultiArrExponents(MultiA); --> !!! ERROR !!! as expected, not free
```

See Also: MultiArrDerModule(I-13.40 pg.220), IsMultiArrFree(I-9.76 pg.174)

I-13.42 MultiArrRestrictionZiegler

syntax

```
MultiArrRestrictionZiegler(A: LIST, H: RINGELEM): LIST
```

This function returns the Ziegler multirestriction of the arrangement of hyperplanes A with respect to its hyperplane H in the variables $[y[1], \dots, y[n]]$.

example

```
/**/ use QQ[x,y,z];
/**/ A := [x, x-z, y-z, z];
/**/ MultiArrRestrictionZiegler(A, z);
[[y[1], 2], [y[2], 1]]
```

See Also: ArrRestriction(I-1.36 pg.42), ArrToMultiArr(I-1.44 pg.44)

I-13.43 MultiArrToArr

syntax

```
MultiArrToArr(MultiA: LIST): LIST
```

This function constructs the underlying arrangement of the multiarrangement MultiA.

example

```
/**/ use QQ[x,y,z];
/**/ MultiA := [[x,1], [y,3], [z,2]];
/**/ MultiArrToArr(MultiA);
[x, y, z]
```

See Also: ArrToMultiArr([I-1.44](#) pg.44)

I-13.44 MultiplicationMat

syntax

```
MultiplicationMat(X: RINGELEM, I: IDEAL): MAT
MultiplicationMat(X: RINGELEM, I: IDEAL, QB: LIST): MAT
```

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-13.45 multiplicity

syntax

```
multiplicity(R: RING): INT
multiplicity(M: MODULE): INT
```

This function computes the multiplicity (or degree) of “R” or “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-8.8](#) pg.137), HilbertSeries([I-8.11](#) pg.139), HVector([I-8.18](#) pg.142)

Chapter I-14

N

I-14.1 NewFractionField

— syntax —

```
NewFractionField(R: RING): RING
```

Create the fraction field of “R”; “R” must be a true GCD domain.

NOTE: calling twice “NewFractionField” will produce two different rings, even with identical input: equality test is performed on the pointers. See “RingID” ([I-18.50](#) pg.290).

— 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-14.11](#) pg.227), BaseRing([I-2.1](#) pg.49), den([I-4.7](#) pg.86), num([I-14.35](#) pg.236), RingID([I-18.50](#) pg.290)

I-14.2 NewFreeModule

— syntax —

```
NewFreeModule(R: RING, N: INT): MODULE
NewFreeModule(R: RING, Shifts: MAT): MODULE
```

This function returns a free module of rank “N” over “R”; for the second form “N” is taken to be the number of columns in “Shifts”.

NOTE: as for rings, calling twice “NewFreeModule” will produce two different modules, even with identical input: equality test is performed on the pointers.

Starting from version CoCoA-5.0.4, this function does accept shifts.

— example —

```
/**/ use R ::= QQ[x,y];
/**/ F := NewFreeModule(R, 3);
/**/ zero(F);
[0, 0, 0]
/**/ type(zero(F)); -- is NOT a LIST
MODULEELEM
/**/ CanonicalBasis(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\(I-2.1 pg.49\)](#), [CanonicalBasis\(I-3.3 pg.58\)](#), [RingOf\(I-18.51 pg.290\)](#)

I-14.3 NewFreeModuleForSyz

syntax

```

NewFreeModuleForSyz(L: LIST of RINGELEM): MODULE

```

This function returns a free module of rank “len(L)” over the ring of “L” with the shifts given by the degrees in “L”.

NOTE: calling twice “NewFreeModuleForSyz” will produce different modules.

example

```

/**/ use QQ[x,y,z];
/**/ L := [x^2, 0, 0, y];

-- /**/ syz(L); --> !!! ERROR !!! as expected: 0 entries
-- for dealing with 0 entries we need to specify the FreeModule

/**/ FwShifts := NewFreeModuleForSyz([x^2, x^100, 1, y]);
/**/ [ wdeg(v) | v in gens(FwShifts) ];
[[2], [100], [0], [1]]
/**/ S := syz(FwShifts, L); indent(S); --> prints "F" for free module
SubmoduleRows(F, matrix([
  [0, 1, 0, 0],
  [0, 0, 1, 0],
  [y, 0, 0, -x^2]
]))
/**/ [ wdeg(v) | v in gens(S) ];
[[100], [0], [3]]

```

See Also: [NewFreeModule\(I-14.2 pg.223\)](#), [syz\(I-19.71 pg.325\)](#)

I-14.4 NewId [OBSOLETE]

[OBSOLETE]

I-14.5 NewLine [OBSOLESCENT]

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.401), println([I-16.45](#) pg.263), ascii([I-1.49](#) pg.46)

I-14.6 NewList

syntax

```
NewList(N: INT): LIST
NewList(N: INT, E: OBJECT): LIST
```

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.406)

I-14.7 NewMat

syntax

```
NewMat(R: RING, M: INT, N: INT): MAT
```

This function is kept only for backward-compatibility with CoCoA-4. Please use instead “ZeroMat” ([I-25.2](#) pg.345) or “NewMatFilled” ([I-14.8](#) pg.225).

See Also: matrix([I-13.11](#) pg.209), NewMatFilled([I-14.8](#) pg.225), ZeroMat([I-25.2](#) pg.345)

I-14.8 NewMatFilled

syntax

```
NewMatFilled(M: INT, N: INT, Val: INT|RAT|RINGELEM): MAT
```

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.267).

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-14.7](#) pg.225), ZeroMat([I-25.2](#) pg.345), matrix([I-13.11](#) pg.209)

I-14.9 NewPolyRing

syntax

```
NewPolyRing(CoeffRing: RING, IndetNames: STRING/LIST): RING
NewPolyRing(CoeffRing: RING, IndetNames: STRING/LIST, OrdMat: MAT, GradingDim: INT): RING
```

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 PP-orderings which can be specified with the “:=” syntax are “Lex” and “Xel” (no grading), “DegLex” and “DegRevLex” (std grading). For other PP-orderings you must use the “NewPolyRing” function call (see also “ElimMat” (I-5.7 pg.97)).

NOTE: calling “NewPolyRing” twice with the same arguments gives two **different rings**, therefore incompatible. See “RingID” (I-18.50 pg.290).

NOTE: the syntax with all indet names in one string is new in CoCoA-5.1.2.

example

```
/**/ 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]]

/**/ 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);

/**/ P3 := NewPolyRing(P2, SymbolRange("alpha", -2,2));
/**/ indets(P3);
[alpha[-2], alpha[-1], alpha[0], alpha[1], alpha[2]]
```

See Also: ElimMat(I-5.7 pg.97), MakeTermOrdMat(I-13.6 pg.206), OrdMat(I-15.11 pg.245), GradingDim(I-7.32 pg.130), IndetSymbols(I-9.27 pg.156), SymbolRange(I-19.68 pg.324), RingID(I-18.50 pg.290)

I-14.10 NewPolyRingWeights

syntax

```
NewPolyRing(CoeffRing: RING, IndetNames: STRING/LIST, W: MAT): RING
```

This function is similar to “NewPolyRing” (I-14.9 pg.226). It returns a polynomial ring with the given matrix of weights “W” (with “GradingDim” (I-7.32 pg.130) equal to “NumRows(W)”).

Currently (CoCoA-5.4.2) only non-negative entries are allowed in “W” (work in progress: Redmine 832).

example

```
/**/ W := matrix([[2,3,1]]);
/**/ P := NewPolyRingWeights(QQ, "a,b,c", W);
/**/ [wdeg(X) | X in indets(P)];
[[2], [3], [1]]

/**/ -- same as
/**/ P2 := NewPolyRing(QQ, "a,b,c", MakeTermOrdMat(W), 1);
/**/ [wdeg(X) | X in indets(P2)];
```

See Also: GradingDim(I-7.32 pg.130), GradingMat(I-7.33 pg.131), MakeTermOrdMat(I-13.6 pg.206), NewPolyRing(I-14.9 pg.226)

I-14.11 NewQuotientRing

syntax

```
NewQuotientRing(R: RING, I: IDEAL): RING
NewQuotientRing(R: RING, s: STRING): RING
R/I      -- R: RING, I: IDEAL
```

This function makes a new ring “R/I”. The syntax with the “STRING” is a shortcut for quotienting by the principal ideal generated by the RINGELEM it represents. This syntax is especially useful when the base ring is contextually created (see example below).

NOTE: calling twice “NewQuotientRing” will produce two different rings, even with identical input: equality test is performed on the pointers. See “RingID” (I-18.50 pg.290).

example

```
/**/ use Qi ::= QQ[i];
/**/ CC := Qi/ideal(i^2+1); -- sort of ;-)
/**/ use CC[x];
/**/ (x+i)^2; --> round brackets in output indicate class in CC
x^2 +(2*i)*x +(-1)

/**/ -- string shortcut
/**/ RmodI := NewQuotientRing(NewPolyRing(QQ, "x,y,z"), "y-3, z^2-5");
/**/ use RmodI;
/**/ (x+y)^2; --> round brackets in output indicate class in RmodI
(x^2 +6*x +9)
```

See Also: BaseRing(I-2.1 pg.49), QuotientBasis(I-17.4 pg.268), NewFractionField(I-14.1 pg.223), RingID(I-18.50 pg.290)

I-14.12 NewRingFp [OBSOLESCENT]

Please use “NewZZmod” (I-14.15 pg.228) instead.

See Also: NewZZmod(I-14.15 pg.228)

I-14.13 NewRingTwinFloat

syntax

```
NewRingTwinFloat(Prec: INT): RING
```

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-18.50 pg.290).

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;
x -0.33333333333333333333
```


The functions give errors if handed negative arguments. “NextProbPrime” uses “IsProbPrime” (I-9.87 pg.178) which cannot be interrupted, so be wary of inputs larger than about “2^100000” (which already takes a few minutes).

[illegible]

I-14.17 NF

```
NF(F: RINGELEM, I: IDEAL): RINGELEM
NF(V: MODULEELEM, M: MODULE): MODULEELEM
```

Currently polynomial ideals are implemented only with coeffs in a field.

```

/**/ 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

```

I-14.18 NFsAreZero [OBSOLETE]

I-14.19 NmzComputation

`NmzComputation(Cone: RECORD): RECORD`
`NmzComputation(Cone: RECORD, ToCompute: LIST): RECORD`

“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.

(sub-)list of fields of cone properties: ModuleGenerators, Generators, ExtremeRays, VerticesOfPolyhedron, Deg1Elements, OriginalMonoidGenerators, SupportHyperplanes, ExcludedFaces, HilbertSeries, Multiplicity, Grading, IsDeg1HilbertBasis, IsPointed, IsIntegrallyClosed, RecessionRank, AffineDim, ModuleRank, Dehomogenization.

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-14.26 pg.233), NmzNormalToricRing(I-14.28 pg.233), NmzIntClosureMonIdeal(I-14.25 pg.232), NmzEhrhartRing(I-14.21 pg.231), NmzTorusInvariants(I-14.30 pg.234), NmzFiniteDiagInvariants(I-14.22 pg.231), NmzDiagInvariants(I-14.20 pg.230), NmzIntersectionValRings(I-14.27 pg.233), NmzSetVerbosityLevel(I-14.29 pg.234)

I-14.20 NmzDiagInvariants

syntax

```
NmzDiagInvariants(M: MAT, N: MAT, R: Ring): LIST of RINGELEM
```

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-14.19 pg.229), NmzTorusInvariants(I-14.30 pg.234), NmzFiniteDiagInvariants(I-14.22 pg.231), NmzSetVerbosityLevel(I-14.29 pg.234)

I-14.21 NmzEhrhartRing

— syntax —

NmzEhrhartRing(L: LIST of RINGELEM, s: RINGELEM): LIST of RINGELEM

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-14.19 pg.229), NmzHilbertBasis(I-14.23 pg.231), NmzNormalToricRing(I-14.28 pg.233), NmzIntClosureMonIdeal(I-14.25 pg.232), NmzSetVerbosityLevel(I-14.29 pg.234)

I-14.22 NmzFiniteDiagInvariants

— syntax —

NmzFiniteDiagInvariants(M: MAT, M: Ring): LIST of RINGELEM

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 $R^G = \text{finR} : g_i f = f \text{ for all } i = 1, \dots, w$.

— 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-14.19 pg.229), NmzTorusInvariants(I-14.30 pg.234), NmzDiagInvariants(I-14.20 pg.230), NmzSetVerbosityLevel(I-14.29 pg.234)

I-14.23 NmzHilbertBasis

— syntax —

NmzHilbertBasis(M: MAT): MAT

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);
-- which is the Hilbert basis of the monoid of the kernel of M:
[]
-- ...no elements! (except the zero-element)

```

See Also: [HilbertBasisKer\(I-8.7 pg.137\)](#), [NmzComputation\(I-14.19 pg.229\)](#), [NmzNormalToricRing\(I-14.28 pg.233\)](#), [NmzIntClosureMonIdeal\(I-14.25 pg.232\)](#), [NmzSetVerbosityLevel\(I-14.29 pg.234\)](#)

I-14.24 NmzHilbertBasisKer

syntax

```

HilbertBasisKer(M: MAT): MAT

```

This function returns a matrix representing the Hilbert basis for the monoid of elements with non-negative coordinates in the kernel of “M”, matrix over “ZZ”.

NOTE: use with care! Interface might change in the next releases to make it compatible with “LinKer” ([I-12.12 pg.197](#)) or “HilbertBasisKer” ([I-8.7 pg.137](#)).

example

```

/**/ M := mat([[1,-2,3,4], [1, 0, 0, -1]]);
/**/ NmzHilbertBasisKer(M);
matrix(QQ,
  [[0, 3, 2, 0],
   [1, 4, 1, 1],
   [2, 5, 0, 2]])

/**/ M * transposed(It);
matrix(QQ,
  [[0, 0, 0],
   [0, 0, 0]])

```

See Also: [LinKerBasis\(I-12.13 pg.198\)](#), [NmzHilbertBasis\(I-14.23 pg.231\)](#)

I-14.25 NmzIntClosureMonIdeal

syntax

```

NmzIntClosureMonIdeal(L: LIST of RINGELEM): LIST of RINGELEM
NmzIntClosureMonIdeal(L: LIST of RINGELEM, s: RINGELEM): LIST of RINGELEM,

```

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-14.19 pg.229), NmzHilbertBasis(I-14.23 pg.231), NmzNormalToricRing(I-14.28 pg.233), NmzEhrhartRing(I-14.21 pg.231), NmzSetVerbosityLevel(I-14.29 pg.234)

I-14.26 NmzIntClosureToricRing

syntax

```
NmzIntClosureToricRing(L: LIST of RINGELEM): LIST of RINGELEM
```

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-14.19 pg.229), NmzHilbertBasis(I-14.23 pg.231), NmzNormalToricRing(I-14.28 pg.233), NmzIntClosureMonIdeal(I-14.25 pg.232), NmzEhrhartRing(I-14.21 pg.231), NmzSetVerbosityLevel(I-14.29 pg.234)

I-14.27 NmzIntersectionValRings

syntax

```
NmzIntersectionValRings(M: MAT, M: Ring): LIST of RINGELEM
```

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-14.19 pg.229), NmzSetVerbosityLevel(I-14.29 pg.234)

I-14.28 NmzNormalToricRing

syntax

```
NmzNormalToricRing(L: LIST of RINGELEM): LIST of RINGELEM
```

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-14.19 pg.229), NmzHilbertBasis(I-14.23 pg.231), NmzIntClosureToricRing(I-14.26 pg.233), NmzIntClosureMonIdeal(I-14.25 pg.232), NmzSetVerbosityLevel(I-14.29 pg.234)

I-14.29 NmzSetVerbosityLevel

syntax

```
NmzSetVerbosityLevel(v: INT)
```

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 the verbosity level for CoCoA.

example

```
/**/ NmzSetVerbosityLevel(1);
```

See Also: SetVerbosityLevel(I-19.14 pg.303), NmzVerbosityLevel(I-14.31 pg.234)

I-14.30 NmzTorusInvariants

syntax

```
NmzTorusInvariants(M: MAT, R: Ring): LIST of RINGELEM
```

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 $R^T = \text{fin}R|_{lf = f \text{ for all } l \in T}$.

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-14.19 pg.229), NmzDiagInvariants(I-14.20 pg.230), NmzFiniteDiagInvariants(I-14.22 pg.231), NmzSetVerbosityLevel(I-14.29 pg.234)

I-14.31 NmzVerbosityLevel

syntax

```
NmzVerbosityLevel(): INT
```

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 the verbosity level for CoCoA.

example

```
/**/ NmzVerbosityLevel();
0
```

See Also: [VerbosityLevel\(I-22.2 pg.339\)](#), [NmzSetVerbosityLevel\(I-14.29 pg.234\)](#)

I-14.32 NonZero

syntax

```
NonZero(L: LIST|MODULEELEM): LIST
```

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\(I-6.11 pg.109\)](#), [FirstNonZeroPosn\(I-6.12 pg.109\)](#), [IsZero\(I-9.107 pg.184\)](#)

I-14.33 not

syntax

```
not(A: BOOL): BOOL
```

This function negates a boolean: *i.e.* if “A” gives “true” then “not(A)” gives “false”, and vice versa.

NOTE: from CoCoA-5.1 “not” is a function, so its argument must be between round brackets!

example

```
/**/ [n in 1..10 | not(IsPrime(n))];
[1,4,6,8,9]
```

See Also: [and\(I-1.13 pg.35\)](#), [or\(I-15.9 pg.244\)](#)

I-14.34 NR

syntax

```
NR(X: RINGELEM, L: LIST of RINGELEM): RINGELEM
NR(X: MODULEELEM, L: LIST of MODULEELEM): MODULEELEM
```

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-4.23 pg.93](#)).

NOTE: this function does not compute a Groebner basis. If the list “L” does not form a Groebner basis then the remainder may not be zero even if “X” is in the ideal or module generated by “L” (use “NF” ([I-14.17 pg.229](#)) instead, or possibly “GenRepr” ([I-7.7 pg.123](#))).

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\(I-4.23 pg.93\)](#), [GenRepr\(I-7.7 pg.123\)](#), [NF\(I-14.17 pg.229\)](#)

I-14.35 num

syntax

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

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\(I-4.7 pg.86\)](#)

I-14.36 NumBChambers

syntax

```
NumBChambers(A: LIST): INT
```

This function returns the number of bounded chambers from the list A of hyperplanes in the arrangement.

example

```
/**/ use QQ[x,y];
/**/ A := [x, x-y, y];
/**/ NumBChambers(A);
0
```

See Also: [NumChambers\(I-14.37 pg.236\)](#)

I-14.37 NumChambers

syntax

```
NumChambers(A: LIST): INT
```

This function returns the number of chambers from the list A of hyperplanes in the arrangement.

example

```
/**/ use QQ[x,y];
/**/ A := [x, x-y, y];
/**/ NumChambers(A);
6
```

See Also: [NumBChambers\(I-14.36 pg.236\)](#)

I-14.38 NumCols

syntax

```
NumCols(M: MAT): INT
```

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\(I-13.11 pg.209\)](#), [NumRows\(I-14.44 pg.238\)](#)

I-14.39 NumCompts

syntax

```
NumCompts(X: MODULEELEM|MODULE): INT
```

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\(I-12.7 pg.195\)](#)

I-14.40 NumGens

syntax

```
NumGens(I: IDEAL): INT
```

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\(I-12.7 pg.195\)](#)

I-14.41 NumIndets

syntax

```
NumIndets(R: RING): INT
```

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\(I-9.21 pg.153\)](#), [IndetSubscripts\(I-9.26 pg.156\)](#), [IndetIndex\(I-9.22 pg.154\)](#), [IndetName\(I-9.23 pg.154\)](#), [indets\(I-9.24 pg.155\)](#)

I-14.42 NumPartitions

— syntax —

```
NumPartitions(N: INT): INT
```

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-14.43 NumRealRoots

— syntax —

```
NumRealRoots(F: RINGELEM): INT
```

This function returns the number of real roots the polynomial “F” has. The coefficients of “F” must be rational.

— example —

```

/**/ use QQ[x];
/**/ NumRealRoots(x^10 - (10^10*x-1)^2);
4

```

See Also: [SturmSeq\(I-19.49 pg.317\)](#), [RealRoots\(I-18.25 pg.279\)](#)

I-14.44 NumRows

— syntax —

```
NumRows(M: MAT): INT
```

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\(I-13.11 pg.209\)](#), [NumCols\(I-14.38 pg.236\)](#)

I-14.45 NumTerms

— syntax —

```
NumTerms(F: RINGELEM): INT
```

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: [support\(I-19.62 pg.322\)](#), [coefficients\(I-3.27 pg.66\)](#), [monomials\(I-13.36 pg.218\)](#), Tutorial: [polynomials\(II-1.9 pg.352\)](#), [len\(I-12.7 pg.195\)](#)

Chapter I-15

O

I-15.1 one

syntax

```
one(R: RING): RINGELEM
```

This function returns 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\(I-25.1 pg.345\)](#)

I-15.2 OpenFile

syntax

```
OpenFile(S: STRING): ISTREAM
```

This function opens the file with name “S” for input. Input from that file can then be read with “[GetLine\(I-7.16 pg.127\)](#)”.

NOTE: it is better to use “[source\(I-19.32 pg.310\)](#)” to read CoCoA commands from a file.

example

```
/**/ D := OpenOFile("my-test"); -- open "my-test" for output from CoCoA
/**/ Print "hello world\nhello!" On D; -- print string into "mytest"
/**/ Close(D);
/**/ D := OpenIFile("my-test"); -- open "my-test" for input to CoCoA
/**/ GetLine(D);
hello world
/**/ GetLine(D);
hello!
/**/ Close(D);
```

See Also: [close\(I-3.19 pg.63\)](#), [Introduction to IO\(II-8.1 pg.371\)](#), [OpenOFile\(I-15.5 pg.243\)](#), [OpenIString\(I-15.3 pg.242\)](#), [OpenOString\(I-15.6 pg.243\)](#), [OpenSocket\(I-15.7 pg.244\)](#), [source\(I-19.32 pg.310\)](#)

I-15.3 OpenIString

syntax

```
OpenIString(S: STRING): ISTREAM
OpenOString(S: STRING): OSTREAM
```

This function open strings for input. “OpenIString” is used to read input from the string “S” with the help of “GetLine” ([I-7.16 pg.127](#)).

example

```
/**/ S := "hello world\n1!\n2!\n3!";
/**/ D := OpenIString(S);
/**/ GetLine(D);
hello world
/**/ GetLine(D);
1!
/**/ GetLine(D);
2!
```

See Also: [close\(I-3.19 pg.63\)](#), [Introduction to IO\(II-8.1 pg.371\)](#), [OpenOString\(I-15.6 pg.243\)](#), [OpenIFile\(I-15.2 pg.241\)](#), [OpenOFile\(I-15.5 pg.243\)](#), [source\(I-19.32 pg.310\)](#), [sprintf\(I-19.35 pg.311\)](#)

I-15.4 OpenLog

syntax

```
OpenLog(D: DEVICE)
```

***** 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-3.20 pg.64](#)) 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.243](#))) or an output string (see “OpenOString” ([I-15.6 pg.243](#))). 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

```
***** NOT YET IMPLEMENTED *****
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-8.1](#) pg.[371](#)), OpenIFile([I-15.2](#) pg.[241](#)), OpenOFile([I-15.5](#) pg.[243](#)), OpenIString([I-15.3](#) pg.[242](#)), OpenOString([I-15.6](#) pg.[243](#))

I-15.5 OpenOFile

— syntax —

```

OpenOFile(S: STRING): OSTREAM
OpenOFile(S: STRING, "w" or "W"): OSTREAM

```

This function opens the file with name “S” for output; If the file exists, its contents are cleared; if it does not exist, it is created empty. Use the command “**print on**” ([I-16.41](#) pg.[261](#)) for appending output to “S”; you must also “**close**” ([I-3.19](#) pg.[63](#)) the file after printing has finished.

— 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"); -- re-open "my-test" and clear it
/**/ print "goodbye" on file;       -- "mytest" now contains only the string "goodbye"
/**/ close(file);
/**/ file := OpenIFile("my-test");  -- open "my-test" for input
/**/ GetLine(file);
goodbye

```

See Also: close([I-3.19](#) pg.[63](#)), Introduction to IO([II-8.1](#) pg.[371](#)), OpenIFile([I-15.2](#) pg.[241](#)), OpenIString([I-15.3](#) pg.[242](#)), OpenOString([I-15.6](#) pg.[243](#)), source([I-19.32](#) pg.[310](#))

I-15.6 OpenOString

— syntax —

```

OpenOString(): OSTREAM

```

This function opens strings for output. “OpenOString” is used to write to a string with the help of “**print on**” ([I-16.41](#) pg.[261](#)). use “**close**” ([I-3.19](#) pg.[63](#)) 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\(I-3.19 pg.63\)](#), [Introduction to IO\(II-8.1 pg.371\)](#), [OpenIFile\(I-15.2 pg.241\)](#), [OpenOFile\(I-15.5 pg.243\)](#), [OpenIString\(I-15.3 pg.242\)](#), [source\(I-19.32 pg.310\)](#), [sprintf\(I-19.35 pg.311\)](#)

I-15.7 OpenSocket

syntax

```
OpenSocket(HostName: STRING, PortNum: INT): RECORD
```

This function opens a client socket (I/O) connection to port “PortNum” on the computer “HostName”; to make a connection to the same computer CoCoA is running on specify the host name “localhost”.

The result is a “RECORD” where the field “send” is an “out-stream”, and the field “recv” is an “in-stream”. The expectation is that a *remote procedure call* is sent on “send”, and the result is read from “recv” using the function “GetLine” ([I-7.16 pg.127](#)).

The first arg “HostName” is a string containing the name of the computer which is waiting for a connection; the second arg “PortNum” is the *port number* at which the server is waiting.

An error will be signalled if the computer “HostName” is not waiting for a connection to port “PortNum”. The obsolete CoCoAServer waits for a connection on the port number 49344 (which is “c0c0” in hexadecimal).

example

```
/**/ --> assuming an echo server is waiting on port 50000...
/**/ IOPair := OpenSocket("localhost", 50000);
/**/ PrintLn 123+456 On IOPair.send;
/**/ str := GetLine(IOPair.recv);
/**/ str;
579
```

See Also: [Introduction to IO\(II-8.1 pg.371\)](#), [OpenIFile\(I-15.2 pg.241\)](#), [OpenOFile\(I-15.5 pg.243\)](#), [OpenIString\(I-15.3 pg.242\)](#), [OpenOString\(I-15.6 pg.243\)](#)

I-15.8 Option [OBSOLETE]

[OBSOLETE]

I-15.9 or

syntax

```
A or B
where A, B: BOOL; return BOOL
```

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\(I-1.13 pg.35\)](#), [not\(I-14.33 pg.235\)](#)

I-15.10 Order Comparison Operators

syntax

$A < B$	less than
$A > B$	greater than
$A \leq B$	less than or equal
$A \geq B$	greater than or equal
return BOOL	

These ordering operators perform the corresponding comparison between “A” and “B”. They will signal an error if “A” and “B” are not comparable (*e.g.* of different types, or if the type does not have a natural ordering). Polynomials which are just power-products are ordered using the power-product ordering.

example

```

/**/ 1 > 1/2;
true
/**/ "abc" < "def"; -- lex ordering for strings
true
/**/ use QQ[x,y];
/**/ x < y; -- x and y are viewed as power-products
false

```

See Also: Equality Operator([I-5.10 pg.98](#)), operators, shortcuts([I-0.1 pg.29](#))

I-15.11 OrdMat

syntax

```
OrdMat(R: RING): MAT
```

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: NewPolyRing([I-14.9 pg.226](#)), MakeTermOrdMat([I-13.6 pg.206](#)), GradingDim([I-7.32 pg.130](#)), StdDegLexMat([I-19.46 pg.316](#)), StdDegRevLexMat([I-19.47 pg.316](#)), LexMat([I-12.8 pg.196](#)), RevLexMat([I-18.45 pg.287](#)), XelMat([I-24.1 pg.343](#)), elim([I-5.5 pg.96](#)), Term Orderings([III-9.5 pg.422](#))

I-15.12 OrlikTeraoIdeal

syntax

`OrlikTeraoIdeal(A: LIST): IDEAL`

This function returns the Orlik-Terao ideal of the list A of hyperplanes of an arrangement.

example

```
/**/ use QQ[x,y];
/**/ A := [x, y, x-y];
/**/ OrlikTeraoIdeal(A);
ideal(y[1]*y[2] +y[1]*y[3] -y[2]*y[3])
```

See Also: [SolomonTeraoIdeal\(I-19.27 pg.308\)](#), [ArtinianOrlikTeraoIdeal\(I-1.48 pg.46\)](#)

Chapter I-16

P

I-16.1 package

— syntax —

```
package PkgName: STRING;
```

This is for more advanced use of CoCoA-5, when you have several functions for closely related operations which you wish to store together in a file.

We recommend putting all functions belonging together in a package into a single file (whose name should indicate the purpose of the package). Inside the file, at the start write “**package**” followed by the package name (usu. the same as as the file name, but with a dollar-sign prepended); at the end of the file put “**EndPackage;**”.

Use the “**export**” (I-5.19 pg.102) command to indicate which functions are intended for public use (rather than being auxiliary functions).

See also the entry “First Example of a Package” (II-9.2 pg.375).

— example —

```
/**/ package $demo;  
/**/  export MyFunc;  
/**/  define MyFunc(N) return N+1; enddefine;  
/**/ EndPackage;
```

See Also: Introduction to Packages(II-9.1 pg.375), First Example of a Package(II-9.2 pg.375), export(I-5.19 pg.102)

I-16.2 PackageOf

— syntax —

```
PackageOf(S: STRING): STRING
```

If “S” contains the name of an identifier defined and exported from a package, it returns the name of that package; otherwise it returns a string saying it is not a package-exported name.

Consult also the documentation for the function “**starting**” (I-19.44 pg.315).

— example —

```
/**/ PackageOf("starting");  
$coclib  
/**/ PackageOf("deg");  
<not a package-exported name>  
/**/ starting("star");  
[record[IsExported := true, name := "$coclib.starting"]]
```

See Also: [starting\(I-19.44 pg.315\)](#), [print\(I-16.40 pg.260\)](#), [describe\(I-4.13 pg.89\)](#), [Introduction to Packages\(II-9.1 pg.375\)](#), [Supported Packages\(II-9.7 pg.377\)](#)

I-16.3 packages

syntax

```
packages(): LIST of STRING
```

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: [Introduction to Packages\(II-9.1 pg.375\)](#), [Supported Packages\(II-9.7 pg.377\)](#)

I-16.4 panel [OBSOLETE]

[OBSOLETE]

I-16.5 panels [OBSOLETE]

[OBSOLETE]

I-16.6 partitions

syntax

```
partitions(N: INT): LIST
```

These function returns all integer partitions of N, positive integer

example

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

See Also: [subsets\(I-19.58 pg.320\)](#), [tuples\(I-20.15 pg.332\)](#)

I-16.7 permutations

syntax

```
permutations(L: LIST): LIST
```

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\(I-19.58 pg.320\)](#), [tuples\(I-20.15 pg.332\)](#)

I-16.8 *PerpIdealOfForm*

syntax

```
PerpIdealOfForm(F: RINGELEM): IDEAL
```

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\(I-9.37 pg.161\)](#), [DerivationAction\(I-4.12 pg.89\)](#)

I-16.9 *pfaffian*

syntax

```
pfaffian(M: MAT): RINGELEM
```

This function returns the Pfaffian of M.

example

```
/**/ use R ::= QQ[x,y];
/**/ pfaffian(mat([[0,y],[-y,0]]));
y
```

See Also: [det\(I-4.14 pg.89\)](#)

I-16.10 *PkgName*

syntax

```
PkgName(): STRING
S.PkgName(): STRING
```

where S is the identifier or alias for a package.

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 of a package.

example

```

GB.PkgName();
$gb
-----
$gb.PkgName();
$gb
-----
PkgName();
$coclib
-----

```

I-16.11 PlayCantStop

syntax

```
PlayCantStop(Player1: STRING, Player2: STRING, ...)
```

This is an interactive game! (see “GetLine” (I-7.16 pg.127)).

Rules at “<https://boardgamegeek.com/boardgame/41/cant-stop>” In brief, this is actually in the **mountain** scenario: you have 3 sherpas which are climbing a mountain. After each 4-die roll, if you can, you move your sherpas up. Then you decide if you stop and establish your base camps for following turns, or if you chance your luck, and roll the 4 dice again and again. If you cannot move, your sherpas fall off the mountain, and that ends your turn.

The game can be played with human players and/or computer players. The computer players are: “ComputerTest”: stupid strategy; “ComputerAnna1”: close to 7 and 50 “ComputerAnna”: more human-like choices ;-).

example

```

-- PlayCantStop("Dave", "Diane", "Joy", "Cole");
-- 4 humans
-- PlayCantStop("ComputerAnna", "ComputerTest", "Anna");
-- 2 computer2 + 1 human
-- PlayCantStop("ComputerAnna", "ComputerTest", "ComputerAnna1");
-- 3 computers ;-)
```

See Also: SleepFor(I-19.24 pg.307), GetLine(I-7.16 pg.127)

I-16.12 PlotPoints

syntax

```
PlotPoints(L: LIST of points)
```

This function outputs the coordinates of the points (with two components) to a file called “CoCoAPlot”. See “PlotPointsOn” (I-16.13 pg.251) 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 ‘\verb&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.151), PlotPointsOn(I-16.13 pg.251)

I-16.13 PlotPointsOn

syntax

```
PlotPointsOn(L: LIST of points, S: STRING)
```

This function is the same as “PlotPoints” (I-16.12 pg.250) with a second argument giving the name of the file to print on.

NOTE: the last argument is a “STRING”, the name of the file, and not an “OSTREAM”, as for “print on” (I-16.41 pg.261).

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.151), PlotPoints(I-16.12 pg.250)

I-16.14 poincare [OBSOLESCENT]

Renamed to “HilbertSeries” (I-8.11 pg.139).

I-16.15 PoincareMultiDeg [OBSOLETE]

Renamed to “HilbertSeriesMultiDeg” (I-8.12 pg.140).

I-16.16 PoincareShifts [OBSOLETE]

Renamed to “HilbertSeriesShifts” (I-8.13 pg.140).

I-16.17 PolyAlgebraHom

syntax

```
PolyAlgebraHom(Domain: RING, Codomain: RING, images: LIST): RINGHOM
PolyAlgebraHom(Domain: RING, Codomain: RING, images: STRING): RINGHOM
```

These functions create 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”. If the rings of coefficients are not the same, consider using “PolyRingHom” (I-16.18 pg.252).

This is a cleaner mathematical implementation of the function “image [OBSOLESCENT]” (I-9.12 pg.149) 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, "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: Introduction to RINGHOM([III-10.1](#) pg.427), CanonicalHom([I-3.4](#) pg.58), PolyRingHom([I-16.18](#) pg.252)

I-16.18 PolyRingHom

syntax

```

PolyRingHom(R: RING, S: RING, CoeffHom: RINGHOM, images: LIST): RINGHOM
PolyRingHom(R: RING, S: RING, CoeffHom: RINGHOM, images: STRING): RINGHOM

```

These functions create 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”. If “CoeffHom” is trivial, consider using “PolyAlgebraHom” ([I-16.17](#) pg.251) instead.

example

```

/**/ R := QQ[x,y];
/**/ S := QQ[a,b,c];
/**/ SmodJ := NewQuotientRing(S, "a^2-1");

/**/ phi := PolyRingHom(R, SmodJ, CanonicalHom(QQ,SmodJ), "a,b");
/**/ use R;
/**/ phi(x); --> round brackets in output indicate class in SmodJ
(a)

```

See Also: Introduction to RINGHOM([III-10.1](#) pg.427), CanonicalHom([I-3.4](#) pg.58), PolyAlgebraHom([I-16.17](#) pg.251)

I-16.19 PosetCharPoly

syntax

```

PosetCharPoly(relP: LIST): RINGELEM

```

This function returns the characteristic polynomial (in the variable “t”) from the list “relP” of the strict relations in a graded poset.

example

```

// POSET:
//      3   4
//      \ /
//      2

```

```
//      |
//      1
/**/ relP := [[1, 2], [2, 3], [2, 4]];
/**/ PosetCharPoly(relP);
t^2 -t
```

See Also: PosetPoincarePoly(I-16.24 pg.254), ArrCharPoly(I-1.26 pg.39)

I-16.20 PosetDual

— syntax —

```
PosetDual(relP: LIST): LIST
```

This function returns the dual from the list “relP” of the strict relations in a poset.

— example —

```
// POSET:
//      3  4
//      \ /
//      2
//      |
//      1
/**/ relP := [[1, 2], [2, 3], [2, 4]];
/**/ PosetDual(relP);
[[2, 1], [3, 2], [4, 2]]
```

See Also: PosetJoin(I-16.21 pg.253), PosetMeet(I-16.22 pg.253)

I-16.21 PosetJoin

— syntax —

```
PosetJoin(relP: LIST, N: INT, M: INT): LIST
```

This function returns the join (*i.e.* the least upper bound) between the two elements “N”, “M” in the poset “P” from the list “relP” of its strict relations.

— example —

```
// POSET:
//      3  4
//      \ /
//      2
//      |
//      1
/**/ relP := [[1, 2], [2, 3], [2, 4]];
/**/ PosetJoin(relP, 3, 4);
[] --> if it does not exist!

/**/ PosetJoin(relP, 2, 4);
[4]
```

See Also: PosetDual(I-16.20 pg.253), PosetMeet(I-16.22 pg.253)

I-16.22 PosetMeet

— syntax —

```
PosetMeet(relP: LIST, N: INT, M: INT): LIST
```

This function returns the meet (*i.e.* the greatest lower bound) between the two elements “N”, “M” in the poset “P” from the list “relP” of its strict relations.

example

```
// POSET:
//      3   4
//      \ / \
//      2   5
//      |
//      1
/**/ relP := [[1, 2], [2, 3], [2, 4], [5,4]];
/**/ PosetMeet(relP, 3, 4);
[2]

/**/ PosetMeet(relP, 2, 5);
[] --> if it does not exist!
```

See Also: PosetDual(I-16.20 pg.253), PosetJoin(I-16.21 pg.253)

I-16.23 PosetNRank

syntax

```
PosetNRank(relP: LIST, N: INT): INT
```

This function returns the rank of the node “N” from the list “relP” of the strict relations in a graded poset.

example

```
// POSET:
//      3   4
//      \ /
//      2
//      |
//      1
/**/ relP := [[1, 2], [2, 3], [2, 4]];
/**/ PosetNRank(relP, 2);
1
```

See Also: PosetRank(I-16.25 pg.255), MaxChains(I-13.14 pg.210)

I-16.24 PosetPoincarePoly

syntax

```
PosetPoincarePoly(relP: LIST): RINGELEM
```

This function returns the Poincare polynomial (in the variable “t”) from the list “relP” of the strict relations in a graded poset.

example

```
// POSET:
//      3   4
//      \ /
//      2
//      |
//      1
/**/ relP := [[1, 2], [2, 3], [2, 4]];
/**/ PosetPoincarePoly(relP);
t +1
```

See Also: PosetCharPoly([I-16.19](#) pg.252), ArrPoincarePoly([I-1.35](#) pg.42)

I-16.25 PosetRank

— syntax —

```
PosetRank(relP: LIST): INT
```

This function returns the rank of a poset from the list “relP” of its strict relations.

— example —

```
// POSET:
//      3   4
//      \ /
//      2
//      |
//      1
/**/ relP := [[1, 2], [2, 3], [2, 4]];
/**/ PosetRank(relP);
2
```

See Also: PosetNRank([I-16.23](#) pg.254), MaxChains([I-13.14](#) pg.210)

I-16.26 power

— syntax —

```
power(X: INT, E: INT): INT
power(X: RAT, E: INT): RAT
power(X: RINGELEM, E: INT): RINGELEM
power(X: MAT, E: INT): MAT
```

This function calculates powers; it is the same as the infix **hat** operator.

— example —

```
/**/ power(2,3);
8
/**/ use QQ[x];
/**/ power(x+1,2);
x^2 +2*x +1
```

See Also: PowerMod([I-16.27](#) pg.255), operators, shortcuts([I-0.1](#) pg.29)

I-16.27 PowerMod

— syntax —

```
PowerMod(A: INT, B: INT, M: INT): INT
```

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. If “A” and “M” are coprime then “B” may be negative.

— example —

```
/**/ PowerMod(12345,41041,41041); -- 41041 is a Carmichael number
12345

/**/ PowerMod(123456789,987654321,32003); -- cannot compute 123456789^987654321 directly
2332
```

I-16.28 PreImage [OBSOLESCENT]

Changed into “preimage0” (I-16.29 pg.256).

See Also: preimage0(I-16.29 pg.256)

I-16.29 preimage0

syntax

```
preimage0(phi: RINGHOM, f: RINGELEM): RECORD
```

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.191), IsSurjective(I-9.101 pg.182)

I-16.30 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
```

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.

This function returns the **primitive part** of a polynomial with rational coefficients: scaled by a rational so that the coefficients are integer and coprime.

— **example** —

```
/**/ use P := QQ[x,y,z];
/**/ f := 2*x+4/7;
/**/ prim(f);
7*x +2
```

See Also: `content`([I-3.51](#) pg.75), `CommonDenom`([I-3.36](#) pg.70), `ClearDenom`([I-3.18](#) pg.63), `monic`([I-13.35](#) pg.217)

I-16.33 PrimaryDecomposition

— **syntax** —

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

This function returns the primary decomposition of the ideal I. Currently it responds ONLY for zero-dimensional ideals (Abbott, Bigatti, Palezzato, Robbiano **Computing and Using Minimal Polynomials** “<https://arxiv.org/abs/1702.07262>” – implemented by E.Palezzato and A.Bigatti), or squarefree monomial ideals (using the Alexander dual technique). See “`FrbPrimaryDecomposition`” ([I-6.30](#) pg.117) 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)]

/**/ PD := PrimaryDecomposition(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)]
```

See Also: `PrimaryDecompositionGTZ0`([I-16.35](#) pg.258), `FrbPrimaryDecomposition`([I-6.30](#) pg.117), `Equi-IsoDec`([I-5.11](#) pg.99)

I-16.34 PrimaryDecomposition0 [OBSOLETE]

Automatically called by “`PrimaryDecomposition`” ([I-16.33](#) pg.258) when appropriate.

See Also: `PrimaryDecomposition`([I-16.33](#) pg.258)

I-16.35 PrimaryDecompositionGTZ0

— **syntax** —

```
PrimaryDecompositionGTZ0(I: IDEAL): LIST of IDEAL
```

This function, an alternative to “`PrimaryDecomposition`” ([I-16.33](#) pg.258), returns the primary decomposition of the 0-dimensional ideal “I” using the GTZ algorithm.

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-z^2, z^2));
/**/ indent(PD);
```

See Also: PrimaryDecomposition(I-16.33 pg.258)

I-16.36 PrimaryHilbertSeries

syntax

```
PrimaryHilbertSeries(I: IDEAL, Q: IDEAL): TAGGED("PSeries")
```

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 := QQ[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\(I-9.29 pg.157\)](#), [TgCone\(I-20.5 pg.328\)](#)

I-16.37 PrimaryPoincare [OBSOLESCE]

Renamed to “PrimaryHilbertSeries” ([I-16.36 pg.259](#)).

See Also: [PrimaryHilbertSeries\(I-16.36 pg.259\)](#)

I-16.38 PrimitiveRoot

syntax

```
PrimitiveRoot(P: INT): INT
```

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\(I-9.85 pg.177\)](#)

I-16.39 primorial

syntax

```
primorial(N: INT): INT
```

This function returns the **primorial** of “N”, the product of all positive primes less than or equal to “N”.

example

```

/**/ primorial(5);
30

/**/ primorial(21);
9699690

```

See Also: [factorial\(I-6.3 pg.106\)](#)

I-16.40 print

syntax

```
print E_1, ..., E_n
```

This command displays the value of each of the expressions “E_i”. To insert a newline write “\n”.

The similar command “println” ([I-16.45 pg.263](#)) is equivalent to “print” followed by a newline.

See “print on” ([I-16.41 pg.261](#)) for printing on file.

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-16.41](#) pg.261), `println`([I-16.45](#) pg.263), `format`([I-6.25](#) pg.115), `indent`, `IndentStr`([I-9.20](#) pg.153), `latex`([I-12.3](#) pg.193), `sprint`([I-19.35](#) pg.311), All CoCoA commands([II-2.2](#) pg.357)

I-16.41 `print on`

syntax

```
print E: OBJECT on OUT: OSTREAM
```

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.243) or “`OpenOString`” ([I-15.6](#) pg.243).

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.243) for an example using output strings. For printing to the CoCoA window, just use “`println E`”.

See Also: Introduction to IO([II-8.1](#) pg.371), `OpenIFile`([I-15.2](#) pg.241), `OpenOFile`([I-15.5](#) pg.243), `OpenIString`([I-15.3](#) pg.242), `OpenOString`([I-15.6](#) pg.243), `print`([I-16.40](#) pg.260), `println`([I-16.45](#) pg.263), All CoCoA commands([II-2.2](#) pg.357)

I-16.42 `PrintBettiDiagram`

syntax

```
PrintBettiDiagram(X: IDEAL or (quotient)RING or MODULE)
PrintBettiDiagram(X: LIST(res) or RECORD(diagram))
```

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\(I-2.3 pg.49\)](#), [BettiMatrix\(I-2.4 pg.50\)](#), [PrintRes\(I-16.46 pg.263\)](#), [PrintBettiMatrix\(I-16.43 pg.262\)](#)

I-16.43 PrintBettiMatrix

syntax

```
PrintBettiMatrix(M: IDEAL|MODULE|Resolution)
```

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\(I-16.46 pg.263\)](#), [PrintBettiDiagram\(I-16.42 pg.261\)](#), [PrintBettiNumbers\(I-16.44 pg.262\)](#)

I-16.44 PrintBettiNumbers

syntax

```
PrintBettiNumbers(M: IDEAL|MODULE|Resolution)
```

This function prints the Betti numbers for “M”.

example

```
/**/ M := MakeTermOrdMat(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-16.46](#) pg.263), `PrintBettiDiagram`([I-16.42](#) pg.261), `PrintBettiMatrix`([I-16.43](#) pg.262), `BettiNumbers`([I-2.5](#) pg.51)

I-16.45 `println`

syntax

```
println E_1,...,E_n
PrintLn E_1,...,E_n
```

This command is equivalent to “`print`” ([I-16.40](#) pg.260) 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 = ", i; endfor;
1
2
3
```

See Also: `format`([I-6.25](#) pg.115), `indent`, `IndentStr`([I-9.20](#) pg.153), `latex`([I-12.3](#) pg.193), `print`([I-16.40](#) pg.260), `print on`([I-16.41](#) pg.261), All CoCoA commands([II-2.2](#) pg.357)

I-16.46 `PrintRes`

syntax

```
PrintRes(M)
```

This function prints the minimal free resolution of “`M`”. (see “`res`” ([I-18.38](#) pg.285)).

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-16.42](#) pg.261), `PrintBettiMatrix`([I-16.43](#) pg.262), `res`([I-18.38](#) pg.285)

I-16.47 PrintSectionalMatrix

syntax

```
PrintSectionalMatrix(I: IDEAL): MAT
PrintSectionalMatrix(P/I: RING): MAT
```

This function prints the sectional matrix of “I” or “P/I”. See “SectionalMatrix” (I-19.6 pg.300) 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-19.6 pg.300)

I-16.48 product

syntax

```
product(L: LIST): OBJECT
product(L: LIST, InitVal: OBJECT): OBJECT
```

This function returns the product of the objects in the list “L” (together with rightmost factor “InitVal”, if specified). If the list “L” may be empty, you must specify also “InitVal”.

example

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

/**/ product(1..40) = factorial(40);
true

/**/ product([], y);
y
/**/ product([3, x], y);
3*x*y
```

See Also: Algebraic Operators(II-4.2 pg.361), sum(I-19.61 pg.321)

I-16.49 *protect*

syntax

```
protect X;
protect X : reason;
  where reason: STRING
```

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: Cannot set "MaxSize"

/**/ unprotect MaxSize; --> remove protection, MaxSize may be assigned to now
/**/ MaxSize := 1000; --> OK
```

See Also: [unprotect\(I-21.3 pg.336\)](#)

I-16.50 *PthRoot*

syntax

```
PthRoot(X: RINGELEM): RINGELEM
```

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\(I-9.56 pg.167\)](#), [IsPthPower\(I-9.88 pg.178\)](#)

Chapter I-17

Q

I-17.1 QQ

syntax

```
QQ: RING
```

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.

NOTE: this is a (protected) **variable**, so in “define/undefine” use “RingQQ” (I-18.52 pg.291) instead (or import it with “TopLevel” (I-20.10 pg.329)).

example

```
/**/ use QQ;

/**/ type(5);
INT
/**/ type(RingElem(QQ, 5));
RINGELEM

/**/ QQ = RingQQ();
true
```

See Also: ZZ(I-25.4 pg.346), NewQuotientRing(I-14.11 pg.227), RingQQ(I-18.52 pg.291), TopLevel(I-20.10 pg.329)

I-17.2 QQEmbeddingHom

syntax

```
QQEmbeddingHom(R: RING): RINGHOM
```

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\(I-3.4 pg.58\)](#)

I-17.3 quit

syntax

```
quit
```

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

See Also: [ciao\(I-3.17 pg.63\)](#), [exit\(I-5.17 pg.101\)](#)

I-17.4 QuotientBasis

syntax

```
QuotientBasis(I: IDEAL): LIST
```

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.268](#)) 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\(I-9.7 pg.147\)](#), [IsFactorClosed\(I-9.54 pg.166\)](#)

I-17.5 QuotientBasisSorted

syntax

```
QuotientBasisSorted(I: IDEAL): LIST
```

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.268](#)), 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\(I-17.4 pg.268\)](#)

I-17.6 QuotientingHom

syntax

```
QuotientingHom(P: RING): RINGHOM
```

This function returns the projection homomorphism of a ring “R” into a quotient ring “R/I”.

It is equivalent to calling “[CanonicalHom\(BaseRing\(RModI\), RModI\)](#)”.

example

```

/**/ use P := QQ[i];
/**/ f := i^3;
/**/ K := P/ideal(i^2+1);
/**/ phi := QuotientingHom(K);  -- phi: P -> K
/**/ RingOf(phi(f));
/**/ phi(f); --> round brackets indicate it is the class of -i in K = P/I
(-i)

```

See Also: [CanonicalHom\(I-3.4 pg.58\)](#)

I-17.7 QZP

syntax

```

QZP(F: RINGELEM): RINGELEM
QZP(F: LIST of POLY): LIST of POLY
QZP(I: IDEAL): IDEAL

```

***** NOT YET IMPLEMENTED ***** See example below

The functions “QZP” and “ZPQ” ([I-25.3 pg.346](#)) 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.346](#)) does the reverse, mapping a polynomial with finite field coefficients into one with rational (actually, integer) coefficients. The function “ZPQ” ([I-25.3 pg.346](#)) 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];
/**/ -- this is the clean way to do it!
/**/ phi := PolyRingHom(R, S, QQEmbeddingHom(S), indets(S));
/**/ phi(F);
-50*x^3 -19*x*y*z +19
***** NOT YET IMPLEMENTED *****
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.[427](#)), BringIn([I-2.13](#) pg.[55](#))

Chapter I-18

R

I-18.1 radical

syntax

```
radical(N: INT): INT
radical(X: RINGELEM): RINGELEM
radical(I: IDEAL): IDEAL
```

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\(I-9.64 pg.170\)](#), [EquiIsoDec\(I-5.11 pg.99\)](#), [RadicalOfUnmixed\(I-18.2 pg.271\)](#), [SqFreeFactor\(I-19.37 pg.312\)](#)

I-18.2 RadicalOfUnmixed

syntax

```
RadicalOfUnmixed(I: IDEAL): IDEAL
```

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\(I-5.11 pg.99\)](#), [radical\(I-18.1 pg.271\)](#)

I-18.3 random

syntax

```
random(X: INT, Y: INT): INT
```

This function returns a random integer between “X” and “Y”, inclusive. The distribution is uniform.

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-18.39 pg.286](#)).

example

```

/**/ random(1,100);
6

/**/ random(-10^4,0);
-3263

```

See Also: [RandomSubset\(I-18.11 pg.274\)](#), [RandomSubsetIndices\(I-18.12 pg.274\)](#), [RandomTuple\(I-18.13 pg.275\)](#), [RandomTupleIndices\(I-18.14 pg.275\)](#), [RandomNBitPrime\(I-18.7 pg.273\)](#), [RandomSmallPrime\(I-18.9 pg.273\)](#), [reseed\(I-18.39 pg.286\)](#)

I-18.4 randomize [OBSOLETE]

OBSOLETE: you can do “f := sum([random(-99,99)*t | t in support(f)])” instead, or use “RandomLinearForm” ([I-18.6 pg.272](#)).

See Also: [random\(I-18.3 pg.272\)](#), [RandomLinearForm\(I-18.6 pg.272\)](#)

I-18.5 randomized [OBSOLETE]

OBSOLETE: you can do “f := sum([random(-99,99)*t | t in support(f)])” instead, or use “RandomLinearForm” ([I-18.6 pg.272](#)).

See Also: [random\(I-18.3 pg.272\)](#), [RandomLinearForm\(I-18.6 pg.272\)](#)

I-18.6 RandomLinearForm

syntax

```

RandomLinearForm(R: RING, n: INT): RINGELEM
RandomLinearForm(R: RING): RINGELEM

```

The first function returns a non-zero random linear form in the polynomial ring “R”, with integer coefficients in the range “-n..n”.

The second function is for polynomial rings “R” with finite characteristic “p”, it returns a random linear form with integer coefficients in the range “0..p-1”.

NOTE: If the ring has weighted degrees, the weights are ignored: consider applying “HomogCompt” ([I-8.15 pg.141](#)) to the linear form produced.

example

```

/**/ R := ZZ/(101)[a,b,c,d,e,f];
/**/ RandomLinearForm(R, 1);
-a -c +d -e -f
/**/ RandomLinearForm(R);
49*a +37*b +36*c +12*d -37*e +34*f

```

See Also: [random\(I-18.3 pg.272\)](#), [HomogCompt\(I-8.15 pg.141\)](#)

I-18.7 RandomNBitPrime

syntax

```
RandomNBitPrime(N: INT): INT
```

This function returns a random prime in the range “ $2^{(N-1)}$ ” and “ 2^N ”. The distribution is uniform. “N” must be between 10 and “31”.

example

```

/**/ RandomNBitPrime(10);
991

```

See Also: [random\(I-18.3 pg.272\)](#), [IsPrime\(I-9.85 pg.177\)](#), [RandomSmallPrime\(I-18.9 pg.273\)](#)

I-18.8 RandomPermutation

syntax

```
RandomPermutation(N: INT): LIST of INT
```

This function returns a random permutation of the integers “1..N”.

example

```

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

```

See Also: [random\(I-18.3 pg.272\)](#), [RandomSubsetIndices\(I-18.12 pg.274\)](#), [RandomTupleIndices\(I-18.14 pg.275\)](#)

I-18.9 RandomSmallPrime

syntax

```
RandomSmallPrime(N: INT): INT
```

This function returns a random prime in the range 5 to “N” (incl.). The distribution is uniform. “N” must be between 5 and “ $2^{31}-1$ ”.

example

```

/**/ RandomSmallPrime(99);
43

```

See Also: [random\(I-18.3 pg.272\)](#), [IsPrime\(I-9.85 pg.177\)](#), [RandomNBitPrime\(I-18.7 pg.273\)](#)

I-18.10 RandomSparseNonSing01Mat

syntax

```
RandomSparseNonSing01Mat(R: RING, N: INT): MAT
```

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.272](#)), `RandomUnimodularMat`([I-18.15 pg.275](#))

I-18.11 RandomSubset

syntax

```
RandomSubset(L: LIST): LIST
RandomSubset(L: LIST, K: INT): LIST
```

This function returns a random subset of “L”; in the second form it ensures that the cardinality of the subset is “K”.

The 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-19.58 pg.320](#))).

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.272](#)), `subsets`([I-19.58 pg.320](#)), `RandomSubsetIndices`([I-18.12 pg.274](#)), `RandomTuple`([I-18.13 pg.275](#)), `RandomTupleIndices`([I-18.14 pg.275](#))

I-18.12 RandomSubsetIndices

syntax

```
RandomSubsetIndices(N: INT): LIST
RandomSubsetIndices(N: INT, K: INT): LIST
```

These functions return a random subset of “1..N”; the second form ensures that the cardinality is “K”. See also “`RandomSubset`” ([I-18.11 pg.274](#)).

NOTE: the resulting list is sorted.

example

```
/**/ RandomSubsetIndices(10);
[1, 3, 4, 5, 8, 10]
/**/ RandomSubsetIndices(10, 3);
[2, 3, 6]
```

See Also: [random\(I-18.3 pg.272\)](#), [subsets\(I-19.58 pg.320\)](#), [RandomPermutation\(I-18.8 pg.273\)](#), [RandomSubset\(I-18.11 pg.274\)](#), [RandomTuple\(I-18.13 pg.275\)](#), [RandomTupleIndices\(I-18.14 pg.275\)](#)

I-18.13 RandomTuple

syntax

```
RandomTuple(L: LIST, K: INT): LIST
```

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.332](#))).

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\(I-18.3 pg.272\)](#), [subsets\(I-19.58 pg.320\)](#), [RandomSubset\(I-18.11 pg.274\)](#), [RandomSubsetIndices\(I-18.12 pg.274\)](#), [RandomTupleIndices\(I-18.14 pg.275\)](#)

I-18.14 RandomTupleIndices

syntax

```
RandomTupleIndices(N: INT, K: INT): LIST
```

The function returns a random tuple of “1..N” of cardinality “K”. See also “[RandomTuple](#)” ([I-18.13 pg.275](#)).

example

```
/**/ RandomTupleIndices(32003, 10);
[4987, 13034, 10044, 7148, 11122, 1144, 21264, 5379, 2934, 7015]
```

See Also: [random\(I-18.3 pg.272\)](#), [subsets\(I-19.58 pg.320\)](#), [RandomSubset\(I-18.11 pg.274\)](#), [RandomSubsetIndices\(I-18.12 pg.274\)](#), [RandomTuple\(I-18.13 pg.275\)](#)

I-18.15 RandomUnimodularMat

syntax

```
RandomUnimodularMat(R: RING, N: INT): MAT
RandomUnimodularMat(R: RING, N: INT, Niters: INT): MAT
```

The function returns a random “N”-by-“N” matrix with integer entries, and determinant +1 or -1. The matrix is over the ring “R”.

The optional 3rd argument controls the size of entries in the matrix produced: a larger value implies larger entries, in general. It actually 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]])
/**/ det(It);
1

```

See Also: [random\(I-18.3 pg.272\)](#)

I-18.16 rank [OBSOLESCE]

See “rk” ([I-18.57 pg.292](#))

I-18.17 RationalSolve

syntax

```

RationalSolve(L: LIST of RINGELEM): RECORD

```

This function computes all rational solutions (aka. points) of a 0-dimensional polynomial system “L”; approximate real solutions can be computed using “[ApproxSolve](#)” ([I-1.17 pg.36](#)). Projective solutions of a homogeneous system can be obtained using “[RationalSolveHomog](#)” ([I-18.18 pg.276](#)).

Result is a “record” saying which indeterminates are **active**, and list of the solution points.

example

```

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

/**/ L := [x^2+y^2-1, x*y-1]; -- indet z not used
/**/ RationalSolve(L);
record[AffinePts := [], indets := [x, y]]

```

See Also: [ApproxSolve\(I-1.17 pg.36\)](#), [LinSolve\(I-12.17 pg.199\)](#), [RationalSolveHomog\(I-18.18 pg.276\)](#)

I-18.18 RationalSolveHomog

syntax

```

RationalSolveHomog(L: LIST of RINGELEM): RECORD

```

This function computes all rational solutions (aka. projective points) of a 1-dimensional homogeneous polynomial system “L”. See also “[RationalSolve](#)” ([I-18.17 pg.276](#)) for solving 0-dimensional systems.

Result is a “record” saying which indeterminates are **active**, and list of the solution points.

example

```

/**/ use QQ[x,y,z];
/**/ L := [x^3-y^2*x, x-2*z];
/**/ RationalSolveHomog(L);
record[ProjectivePts := [[0, 1, 0], [1, -1, 1/2], [1, 1, 1/2]], indets := [x, y, z]]

```

See Also: [RationalSolve\(I-18.17 pg.276\)](#), [LinSolve\(I-12.17 pg.199\)](#)

I-18.19 RatReconstructByContFrac

— syntax —

```
RatReconstructByContFrac(X: INT, M: INT): RECORD
RatReconstructByContFrac(X: INT, M: INT, LogToler: INT): RECORD
```

These functions attempt to reconstruct rational numbers from a modular image “ $X \bmod M$ ”. The result is a record: the boolean field “failed” is “true” if no **convincing** result was found; otherwise that field is “false”, and a second field, called “ReconstructedRat”, contains the value reconstructed.

The algorithms are **fault-tolerant**: they will succeed provided that “ X ” is correct modulo a sufficiently large factor of “ M ”.

An optional third argument 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 (“RatReconstructByLattice” (I-18.20 pg.277)). 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 := 3333333333;
/**/ 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-18.22 pg.278), RatReconstructByLattice(I-18.20 pg.277), CRT(I-3.62 pg.79)

I-18.20 RatReconstructByLattice

— syntax —

```
RatReconstructByLattice(X: INT, M: INT): RECORD
RatReconstructByLattice(X: INT, M: INT, threshold: INT): RECORD
```

Same as “RatReconstructByContFrac” (I-18.19 pg.277) but with a different underlying heuristic algorithms, based on 2-dimensional lattice reduction.

— example —

```
/**/ X := 3333333333;
/**/ M := 10^10;
/**/ RatReconstructByLattice(X,M);
record[ReconstructedRat := -1/3, failed := false]

/**/ X := 3141592654;
/**/ M := 10^10;
/**/ RatReconstructByLattice(X,M);
record[failed := true]
```

See Also: RatReconstructWithBounds(I-18.22 pg.278), RatReconstructByContFrac(I-18.19 pg.277), CRT(I-3.62 pg.79)

I-18.21 RatReconstructPoly

syntax

```
RatReconstructPoly(f1: RINGELEM, M1: INT): RINGELEM
```

This function attempts to reconstruct the rational coefficients of a polynomial “f” from a modular image “f mod 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(10923689802589*x^2 +8192767351939*y, 32771069407757);
(10/3)*x^2 +(-1/4)*y

-- input comes from CTR computation:
/**/ RingElem(NewPolyRing(NewZZmod(32003),"x,y"), "(10/3)*x^2 +(-1/4)*y");
10671*x^2 -8001*y
/**/ RingElem(NewPolyRing(NewZZmod(31991),"x,y"), "(10/3)*x^2 +(-1/4)*y");
10667*x^2 -7998*y
/**/ RingElem(NewPolyRing(NewZZmod(32009),"x,y"), "(10/3)*x^2 +(-1/4)*y");
10673*x^2 +8002*y

/**/ CRTPoly(10671*x^2 -8001*y, 32003, 10667*x^2 -7998*y, 31991);
record[modulus := 1023807973, residue := -341269321*x^2 +255951993*y]
/**/ CRTPoly(It.residue, It.modulus, 10673*x^2 +8002*y, 32009);
record[modulus := 32771069407757,
      residue := 10923689802589*x^2 +8192767351939*y]
/**/ RatReconstructPoly(It.residue, It.modulus);
(10/3)*x^2 +(-1/4)*y
```

See Also: CRTPoly(I-3.63 pg.80), RatReconstructByContFrac(I-18.19 pg.277), RatReconstructByLattice(I-18.20 pg.277)

I-18.22 RatReconstructWithBounds

syntax

```
RatReconstructWithBounds(e: INT, P: INT, Q: INT, res: LIST of INT, mod: LIST of INT): RECORD
```

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-3.62 pg.79), RatReconstructByContFrac(I-18.19 pg.277), RatReconstructByLattice(I-18.20 pg.277)

I-18.23 ReadExpr [OBSOLETE]

[OBSOLETE] from version 5.2.0 “RingElem” (I-18.47 pg.288) does the same, and more.

I-18.24 RealRootRefine

syntax

```
RealRootRefine(Root: RECORD, Precision: RAT): RECORD
```

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-18.25 pg.279).

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-18.25 pg.279), RealRootsApprox(I-18.26 pg.280), RootBound(I-18.59 pg.293)

I-18.25 RealRoots

syntax

```
RealRoots(F: RINGELEM): LIST
RealRoots(F: RINGELEM, Precision: RAT): LIST
RealRoots(F: RINGELEM, Precision: RAT, Interval:[RAT, RAT]): LIST
```

This function computes isolating intervals for the real roots of a non-zero univariate polynomial over “QQ”. 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. There is a third field (called “CoeffList”) not intended for **public use**.

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-18.24 pg.279).

The function “RealRootsApprox” (I-18.26 pg.280) may be easier to use: 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,10); -- left end of interval
-1.414213562*10^0

/**/ FloatStr(RR[1].sup,10); -- right end of interval
```

```

-1.414213561*10^0

/**/ RR := RealRoots(x^2-2, 10^(-20), [0, 2]);
/**/ FloatStr(RR[1].inf, 20); -- print rational RR[1].inf in a comprehensible way
1.4142135623730950488*10^0
/**/ RR[1].inf; -- actual raional is rather incomprehensible
60153992292001127886258443119406264231/42535295865117307932921825928971026432

```

See Also: NumRealRoots(I-14.43 pg.238), RealRootRefine(I-18.24 pg.279), RealRootsApprox(I-18.26 pg.280), RootBound(I-18.59 pg.293)

I-18.26 RealRootsApprox

— syntax —

```

RealRootsApprox(F: RINGELEM): LIST
RealRootsApprox(F: RINGELEM, Precision: RAT): LIST
RealRootsApprox(F: RINGELEM, Precision: RAT, Interval:[RAT, RAT]): LIST

```

This function computes rational approximations to the real roots of a univariate polyomial (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.

NOTE: the value of “F” at an approximate root may not be small; see also “ApproxSolve” (I-1.17 pg.36) which uses a heuristic to produce approximate roots such that the evaluation is also **small**.

— example —

```

/**/ use QQ[x];
/**/ RealRootsApprox(x^2-2);
[-3037000499/2147483648, 3037000499/2147483648]

/**/ RR := RealRootsApprox(x^2-2, 10^(-15), [0, 2]);
/**/ FloatStr(RR[1], 15); --> print as a decimal, easier to read
1.41421356237310*10^0

```

See Also: ApproxSolve(I-1.17 pg.36), RealRoots(I-18.25 pg.279), RootBound(I-18.59 pg.293)

I-18.27 record

— syntax —

```

record[F_1 := OBJECT,..., F_n := OBJECT]
  where each F_i is a field name
  returns RECORD

```

This constructor creates a record with fields called “F_1”, ..., “F_n”. 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-18.28 pg.281), fields(I-6.8 pg.108)

I-18.28 record field selector

syntax

```
R.FieldName
R["FieldName"]
  where R is a RECORD
```

A record is a data structure containing named entries. They are created using the command “**record**” (I-18.27 pg.280). Each entry may be selected using the **dot operator**, or equivalently via 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-18.27 pg.280), fields(I-6.8 pg.108)

I-18.29 ReducedGBasis

syntax

```
ReducedGBasis(I: IDEAL): LIST of RINGELEM
ReducedGBasis(I: MODULE): LIST of MODULEELEM
```

This function returns a list whose elements 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.121)

I-18.30 ref

syntax

```
ref X
  where X is the identifier of a CoCoA variable.
```

The keyword “**ref**” is used to pass a parameter **by reference** to a function which may modify its value (*e.g.* “**append**” (I-1.14 pg.35)). 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\(I-4.4 pg.84\)](#)

I-18.31 RefineGCDFreeBasis [OBSOLETE]

OBSOLETE. Just build a new list and call “CoprimeFactorBasis” (I-3.58 pg.78).

See Also: [CoprimeFactorBasis\(I-3.58 pg.78\)](#)

I-18.32 reg

syntax

```

reg(I: IDEAL): INT
reg(R: (Quotient)RING): INT

```

These functions compute 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-18.33 pg.283), 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\(I-18.38 pg.285\)](#), [PrintRes\(I-16.46 pg.263\)](#), [PrintBettiDiagram\(I-16.42 pg.261\)](#), [PrintBettiMatrix\(I-16.43 pg.262\)](#), [RegularityIndex\(I-18.33 pg.283\)](#)

I-18.33 RegularityIndex

syntax

```
RegularityIndex(R: (Poly or Quotient)RING): INT
```

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\(I-8.8 pg.137\)](#), [HilbertSeries\(I-8.11 pg.139\)](#)

I-18.34 RelNotes

syntax

```
RelNotes()
RelNotes(N: INT)
```

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\(I-22.3 pg.340\)](#)

I-18.35 ReloadMan

syntax

```
ReloadMan()
ReloadMan(ListOfFiles: LIST)
```

This function reloads the xml source of the manual “CoCoAHelp.xml” (in directory “CoCoAManual”) and the optional list of manual extensions: it recreates the internal manual **index** in a running CoCoA-5.

Some details: 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 because the descriptions are always read from disk.

NOTE: This function is useful for those who develop a CoCoA package and want to test/add/offer its manual in proper CoCoA style. Such manual extensions will be included in “CoCoAHelp.xml” if the package is generously offered for distribution with CoCoA :-)

example

```
/**/ ReloadMan(); // reloads the official manual only
/**/ // for loading manual extensions, official manual+ file1 + file2:
/**/ ReloadMan(["path1/file1.xml", "path2/file2.xml"]);
```

I-18.36 remove

syntax

```
remove(ref L: LIST, N: INT)
```

This function removes the “N”-th component from “L”; it changes the value of “L”. Use the function “WithoutNth” (I-23.4 pg.342) 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: diff(I-4.18 pg.91), WithoutNth(I-23.4 pg.342)

I-18.37 repeat

syntax

```
repeat C until B
repeat C endrepeat
(where C is a sequence of commands and B is BOOL)
```

In the first form, the command sequence “C” is repeated until “B” evaluates to “false”. Unlike the “while” (I-23.3 pg.342) command, “C” is executed at least once.

NOTE: there is no “endrepeat” following the condition “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-6.23](#) pg.113), foreach([I-6.24](#) pg.114), All CoCoA commands([II-2.2](#) pg.357), while([I-23.3](#) pg.342)

I-18.38 res

— syntax —

```

res(M: IDEAL): LIST
res(M: MODULE): LIST

```

This function returns the minimal free resolution of homogeneous “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?).

NOTE: for the resultant of two polynomials use “resultant” ([I-18.42](#) pg.286).

— 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-16.42](#) pg.261), `PrintBettiMatrix`([I-16.43](#) pg.262), `PrintRes`([I-16.46](#) pg.263), `resultant`([I-18.42](#) pg.286)

I-18.39 reseed

syntax

```
reseed(N: INT): VOID
```

CoCoA offers a pseudo-random number generator (see “`random`” ([I-18.3](#) pg.272)); 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.272)

I-18.40 Reset [OBSOLETE]

[OBSOLETE]

I-18.41 ResetPanels [OBSOLETE]

[OBSOLETE]

I-18.42 resultant

syntax

```
resultant(F: RINGELEM, G: RINGELEM): RINGELEM
resultant(F: RINGELEM, G: RINGELEM, X: RINGELEM): RINGELEM
```

These functions compute the resultant of the polynomials “`F`” and “`G`” with respect to the indeterminate “`X`”; in the first form “`F`” and “`G`” must be univariate, and “`X`” is taken to be their common indeterminate.

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-4.20](#) pg.91), `SylvesterMat`([I-19.67](#) pg.323)

I-18.43 *return*

— syntax —

```
return
return E
```

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.54](#)).

— 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\(I-2.12 pg.54\)](#), [define\(I-4.4 pg.84\)](#), All CoCoA commands([II-2.2 pg.357](#))

I-18.44 *reverse, reversed*

— syntax —

```
reverse(ref L: LIST)
reversed(L: LIST): LIST
```

The function “**reverse**” reverses the order of the elements of the list “*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\(I-19.28 pg.308\)](#), [sorted\(I-19.30 pg.309\)](#)

I-18.45 *RevLexMat*

— syntax —

```
RevLexMat(N: INT): MAT
```

This function returns 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\(I-15.11 pg.245\)](#), [Term Orderings\(III-9.5 pg.422\)](#), [StdDegRevLexMat\(I-19.47 pg.316\)](#), [StdDegLexMat\(I-19.46 pg.316\)](#), [LexMat\(I-12.8 pg.196\)](#), [XelMat\(I-24.1 pg.343\)](#)

I-18.46 rgin

syntax

```
rgin(I: IDEAL): IDEAL
```

These functions return the [probabilistic] rgin (generic initial ideal wrt StdDegRevLex) of the ideal “I”. See “gin” ([I-7.30 pg.129](#)) 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\(I-7.30 pg.129\)](#), [LT\(I-12.24 pg.202\)](#)

I-18.47 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

```

This function converts the expression “E” into a RINGELEM in “R”, if possible. These functions are useful for operating with different rings.

If “E” is a “STRING” it evaluates “E” in “R”, (with no need for “use R”). The expression “E” may contain arithmetic operations and parentheses (but no programming variables or function calls). Use “RingElemList, RingElems” ([I-18.48 pg.289](#)) to read many (comma-separated) ringelems into a list.

If “E” is a RINGELEM it is equivalent to applying the “CanonicalHom” ([I-3.4 pg.58](#)) from “RingOf(E)” to “R” (for other homomorphisms see “RINGHOM” ([III-10 pg.427](#))).

example

```

/**/ P := ZZ/(5)[x,y]; S := QQ[x,y,z[1..4]]; K := NewFractionField(S);
/**/ QR := NewQuotientRing(S, "x^2-3, y^2-5");

/**/ -- STRING
/**/ RingElem(P, "7*x"); --> 7*x as an element of P
2*x
/**/ RingElem(S, "7*x"); --> 7*x as an element of S

```



```

7*x
/**/ RingElem(S, "((7/3)*z[2] - 1)^2"); -- expr without function calls
(49/9)*z[2]^2 + (-14/3)*z[2] + 1
/**/ RingElem(K, "(x^2-x*y)/(x*y-y^2)");
x/y
/**/ RingElem(QR, "(x+y)^3");
(18*x +14*y)

/**/ -- RINGELEM (via CanonicalHom)
/**/ use S;
/**/ f := 2*x-3;
-- /**/ 1/f; --> !!! ERROR !!! as expected: f in P is not invertible
/**/ 1/RingElem(K, f); -- f in K is invertible
1/(2*x -3)

/**/ use P;
/**/ -- INT and RAT
/**/ RingElem(P, 7);
2
/**/ RingElem(P, 3/2);
-1

/**/ -- LIST for indexed indets
/**/ i := 2; RingElem(S, ["z",i]);
z[2]

```

See Also: [RingElemList](#), [RingElems](#)([I-18.48](#) pg.289), [RingOf](#)([I-18.51](#) pg.290), [AsINT](#)([I-1.50](#) pg.46), [AsRAT](#)([I-1.51](#) pg.47), [IndetName](#)([I-9.23](#) pg.154), [IndetSubscripts](#)([I-9.26](#) pg.156), [CanonicalHom](#)([I-3.4](#) pg.58), [sprint](#)([I-19.35](#) pg.311)

I-18.48 *RingElemList, RingElems*

syntax

```

RingElems(R: RING, S: STRING): LIST of RINGELEM
RingElemList(R: RING, S: STRING): LIST of RINGELEM

```

These functions convert the string “S” into a LIST of RINGELEM in “R”; the only difference is that “RingElemList” expects the list of values in the string to be inside square brackets. The successive values in “S” must be comma separated.

The expressions in “S” may contain arithmetic operations and parentheses (but no programming variables nor function calls). These functions are useful for operating with different rings (without needing to call the “use” ([I-21.6](#) pg.336) command).

“RingElems” from version 5.3.0; “RingElemList” from version 5.4.0.

example

```

/**/ P ::= ZZ/(5)[x,y];   S ::= QQ[x,y,z[1..4]];

/**/ RingElems(P, "x^2, (x-3*y)^5"); --> elements of P
[x^2, x^5 +2*y^5]
/**/ RingElemList(S, "[x^2, (x-3*y)^5]"); --> elements of S
[x^2, x^5 -15*x^4*y +90*x^3*y^2 -270*x^2*y^3 +405*x*y^4 -243*y^5]

-- even for making ideals in new rings:
/**/ I := ideal(RingElems(NewPolyRing(QQ, "i"), "i^2 +1")); I;
ideal(i^2 +1)

```

See Also: `RingOf`([I-18.51](#) pg.290), `RingElem`([I-18.47](#) pg.288), `NewQuotientRing`([I-14.11](#) pg.227), `PolyAlgebraHom`([I-16.17](#) pg.251), `sprint`([I-19.35](#) pg.311)

I-18.49 RingEnv [OBSOLETE]

See “`RingOf`” ([I-18.51](#) pg.290).

See Also: `RingOf`([I-18.51](#) pg.290)

I-18.50 RingID

syntax

```
RingID(R: RING): INT
```

This function is likely useful only for debugging. It returns the identification number of the ring “R”.

NOTE: that CoCoA considers two rings to be equal if and only if they have the same ID.

example

```
/**/ R ::= QQ[x,y,z];
/**/ S ::= QQ[x,y,z]; // creates a separate copy
/**/ R = S;
false
/**/ RingID(R);
7
/**/ RingID(S);
8
```

See Also: `print`([I-16.40](#) pg.260), `println`([I-16.45](#) pg.263), `Evaluation and Assignment`([II-5](#) pg.365)

I-18.51 RingOf

syntax

```
RingOf(E: RINGELEM|IDEAL|MAT|MODULE): RING
```

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: [CoeffRing\(I-3.32 pg.68\)](#), [CurrentRing\(I-3.64 pg.80\)](#), [RingsOf\(I-18.55 pg.292\)](#), [BaseRing\(I-2.1 pg.49\)](#)

I-18.52 RingQQ

syntax

```
RingQQ(): RING
```

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-14.9 pg.226](#)) and “NewFractionField” ([I-14.1 pg.223](#)) 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\(I-20.10 pg.329\)](#), [QQ\(I-17.1 pg.267\)](#), [RingQQt\(I-18.53 pg.291\)](#), [RingZZ\(I-18.56 pg.292\)](#)

I-18.53 RingQQt

syntax

```
RingQQt(N: INT): RING
```

This function returns a polynomial ring over “QQ” with indeterminates $t[1]...t[N]$. In particular “RingQQt(1)” is the polynomial ring in which Hilbert polynomials are defined.

NOTE: calling “RingQQt(2)” twice gives the same identical ring, whereas calling “NewPolyRing(QQ, "t[1],t[2]");” returns each time a new ring (therefore incompatible).

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\(I-20.10 pg.329\)](#), [RingQQ\(I-18.52 pg.291\)](#), [RingZZ\(I-18.56 pg.292\)](#)

I-18.54 RingSet [OBSOLETE]

Renamed to “RingsOf” ([I-18.55 pg.292](#))

See Also: [RingsOf\(I-18.55 pg.292\)](#)

I-18.55 RingsOf

syntax

`RingsOf(E: LIST|RINGELEM|IDEAL|MAT|MODULE|MODULEELEM): LIST of RING and TYPE`

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-18.51 pg.290), 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 := NewZZmod(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-3.64 pg.80), `RingOf`(I-18.51 pg.290)

I-18.56 RingZZ

syntax

`RingZZ(): RING`

This function returns the ring of integers. It is useful when you want to use “ZZ” (I-25.4 pg.346) inside “define/enddefine”.

NOTE: calling “RingZZ” twice gives the same identical ring, whereas calling “NewPolyRing” (I-14.9 pg.226) or “NewFractionField” (I-14.1 pg.223) 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.329), `RingQQt`(I-18.53 pg.291), `RingQQ`(I-18.52 pg.291), `ZZ`(I-25.4 pg.346)

I-18.57 rk

syntax

`rk(M: MAT): INT`
`rk(M: MODULE): INT`

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-14.39 pg.237) 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-18.58 RMap [OBSOLESCE

syntax

```
[OBSOLESCE] RMap(L: LIST): TAGGED("RMap")
```

[OBSOLESCE] related with “image [OBSOLESCE]” (I-9.12 pg.149). In CoCoA-5 such homomorphisms are properly implemented as “PolyAlgebraHom” (I-16.17 pg.251).

I-18.59 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

```

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 specifies a trade-off between speed and tightness of the bound (more precisely: it says how many iterations of Graeffe transformation to apply); higher 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-7.34 pg.131), RootBoundTransform(I-18.60 pg.293), LinearSimplify(I-12.11 pg.197), RealRootRefine(I-18.24 pg.279), RealRoots(I-18.25 pg.279)

I-18.60 RootBoundTransform

syntax

```
RootBoundTransform(F: RINGELEM): RINGELEM
```

The function “**RootBoundTransform**” computes a simple transform of a univariate polynomial (of degree at least 1) with rational coefficients. The transform is “ $a[d]*x^d - a[d-1]*x^{(d-1)} - \dots - a[0]$ ” where “ $a[k]$ ” denotes the absolute value of the coefficient of “ x^k ” in “ F ”.

example

```
/**/ use P := QQ[x];
/**/ RootBoundTransform(-3*x^3+x^2-2);
3*x^3 -x^2 -2
```

See Also: [graeffe\(I-7.34 pg.131\)](#), [RootBound\(I-18.59 pg.293\)](#)

I-18.61 round

syntax

```
round(X: RAT): INT
```

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\(I-14.35 pg.236\)](#), [den\(I-4.7 pg.86\)](#), [floor\(I-6.17 pg.111\)](#), [ceil\(I-3.9 pg.60\)](#), [div\(I-4.22 pg.92\)](#)

I-18.62 RowMat

syntax

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

This function returns 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\(I-2.9 pg.53\)](#), [DiagMat\(I-4.16 pg.90\)](#), [ColMat\(I-3.33 pg.69\)](#)

I-18.63 rref

syntax

```
rref(M: MAT): MAT
```

This function returns the reduced row echelon form of “M”; the columns are considered in left-to-right order. An error is signalled if the matrix is not over a field.

example

```
/**/ M := mat([[1,2,3],[2,3,4]]);  
/**/ rref(M);  
matrix(QQ,  
  [[1, 0, -1],  
   [0, 1, 2]])
```

See Also: [rk\(I-18.57 pg.292\)](#)

Chapter I-19

S

I-19.1 SAGBI, SAGBIHomog

syntax

```
sagbi(L: LIST): LIST
SAGBI(L: LIST): LIST
SAGBIHomog(L: LIST): LIST
SAGBIHomog(L: LIST, TruncDeg: INT): LIST
```

These functions return a SAGBI (Subalgebra Analogue of GBases for Ideals) of the algebra generated by “L”.
NOTE: a SAGBI might be infinite, so (interrupt the computation and) use truncation if the computation takes too long.

Verbosity range 40-90.

example

```
/**/ use P := QQ[x,y,z];
/**/ G := [x^2, x*y -z^2, y^2];
/**/ sagbi(G);
[y^2, x*y -z^2, x^2, x*y*z^2 +(-1/2)*z^4]

/**/ SetVerbosityLevel(50);
/**/ SAGBIHomog(G, 3); -- Truncation: up to degree 3
(... verbosity ...)
[y^2, x*y -z^2, x^2]
```

See Also: SubalgebraMinGens(I-19.52 pg.318), Introduction to verbosity and interrupt(II-7.1 pg.369)

I-19.2 SatSAGBI

syntax

```
SatSAGBI(L: LIST, x: RINGELEM): LIST
SatSAGBI(L: LIST, x: RINGELEM, TruncDeg: INT): LIST
```

These functions return a SAGBI (Subalgebra Analogue of GBases for Ideals) of the algebra generated by “L” saturated by the indeterminate “x”. The ordering must be of “x”-DegRev-type.

If the input is bi-homogenous, and “g” is an indeterminate with first weight 0, then the second form computes the truncated SatSAGBI.

For all details, see paper A.M.Bigatti, L.Robbiano, JSC (2022) **Saturations of subalgebras, SAGBI bases, and U-invariants.**

Verbosity range 40-90.

example

```

/**/ use P := QQ[x,y,z], DegRevLex;
/**/ g_1 := x^2;
/**/ g_2 := x*y -z^2;
/**/ g_3 := x^5*y -5*x^3*y^3 -x^4*z^2 +15*x^2*y^2*z^2 -15*x*y*z^4 +5*z^6;
/**/ G := [g_1,g_2,g_3];
/**/ SAGBI(G);
[x*y -z^2, x^2]
/**/ SatSAGBI(G,z);
[z, x*y, x^2]
/**/ -- SatSAGBI(G,x); --> !!! ERROR !!! as expected: must be x-DegRev-type
/**/ -- SatSAGBI(G,z,1); --> !!! ERROR !!! as expected: must be bi-homogenous

/**/ O := MakeTermOrdMat(mat([[0,1,2,3],[1,1,1,1], [-1, 0, 0, 0]]));
/**/ P := NewPolyRing(QQ, "a_0,a_1,a_2,a_3", 0, 2);
/**/ Use P;
/**/ G := [a_0, (-1/2)*a_1^2 +a_0*a_2, (1/3)*a_1^3 -a_0*a_1*a_2 +a_0^2*a_3];
/**/ IsHomog(G);
true
/**/ indent(SAGBI(G));
[
  a_0,
  a_1^2 -2*a_0*a_2,
  a_1^3 -3*a_0*a_1*a_2 +3*a_0^2*a_3,
  a_0^2*a_1^2*a_2^2 -2*a_0^2*a_1^3*a_3 +(-8/3)*a_0^3*a_2^3 +6*a_0^3*a_1*a_2*a_3 -3*a_0^4*a_3^2
]
/**/ indent(SatSAGBI(G,a_0));
[
  a_0,
  a_1^2 -2*a_0*a_2,
  a_1^3 -3*a_0*a_1*a_2 +3*a_0^2*a_3,
  a_1^2*a_2^2 -2*a_1^3*a_3 +(-8/3)*a_0*a_2^3 +6*a_0*a_1*a_2*a_3 -3*a_0^2*a_3^2
]
/**/ indent(SatSAGBI(G, a_0, 2)); -- truncated
[
  a_0,
  a_1^2 -2*a_0*a_2,
  a_1^3 -3*a_0*a_1*a_2 +3*a_0^2*a_3
]

```

See Also: SAGBI, SAGBIHomog(I-19.1 pg.297), Introduction to verbosity and interrupt(II-7.1 pg.369)

I-19.3 saturate

syntax

```
saturate(I: IDEAL, J: IDEAL): IDEAL
```

This function returns the saturation of “I” with respect to “J”: the ideal of polynomials f such that $f * g$ is in “I” for all g in J^k for some positive integer k .

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\(I-3.34 pg.69\)](#), [HColon\(I-8.3 pg.136\)](#), [HSaturation\(I-8.17 pg.142\)](#)

I-19.4 ScalarProduct

syntax

```
ScalarProduct(L: LIST, M: LIST): INT, RAT, or RINGELEM
```

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-4.2 pg.361](#))).

example

```

/**/ ScalarProduct([1,2,3], [5,0,-1]);
2

/**/ use QQ[x,y,z];
/**/ ScalarProduct([x,y,z], [5,0,-1]);
5*x -z

```

See Also: [Algebraic Operators\(II-4.2 pg.361\)](#)

I-19.5 ScientificStr

syntax

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

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. The exponent is always included (even if it is zero).

example

```

/**/ ScientificStr(2/3);      -- last printed digit is rounded
6.6667*10^(-1)

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

/**/ ScientificStr(1/81, 35); -- specify number of digits
1.2345679012345679012345679012345679*10^(-2)

/**/ ScientificStr(1/2);      -- trailing zeroes are not suppressed
5.0000*10^(-1)

```

See Also: `DecimalStr`([I-4.3](#) pg.83), `FloatStr`([I-6.16](#) pg.111), `FloatApprox`([I-6.15](#) pg.110), `MantissaAndExponent10`([I-13.7](#) pg.207)

I-19.6 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
```

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” (homogenous ideal) or “PmodI”.

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-19.7 seed [OBSOLETE]

Replaced by “`reseed`” ([I-18.39](#) pg.286).

I-19.8 SeparatorsOfPoints

syntax

```
SeparatorsOfPoints(Points: LIST): LIST
```

where Points is a list of lists of coefficients representing a set of `\textbf{distinct}` points in affine space.

***** 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.147](#)).

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-19.9 pg.301](#)).

example

```
***** NOT YET IMPLEMENTED *****
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.123](#)), [IdealAndSeparatorsOfPoints](#)([I-9.3 pg.143](#)), [IdealAndSeparatorsOfProjectivePoints](#)([I-9.4 pg.144](#)), [IdealOfPoints](#)([I-9.7 pg.147](#)), [IdealOfProjectivePoints](#)([I-9.8 pg.147](#)), [Interpolate](#)([I-9.31 pg.158](#)), [SeparatorsOfProjectivePoints](#)([I-19.9 pg.301](#))

I-19.9 SeparatorsOfProjectivePoints

syntax

```
SeparatorsOfProjectivePoints(Points: LIST): LIST
```

where `Points` is a list of lists of coefficients representing a set of `\textbf{distinct}` points in projective space.

***** 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.147](#)).

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-19.8 pg.300](#)).

example

```
***** NOT YET IMPLEMENTED *****
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\(I-7.6 pg.123\)](#), [IdealAndSeparatorsOfPoints\(I-9.3 pg.143\)](#), [IdealAndSeparatorsOfProjectivePoints\(I-9.4 pg.144\)](#), [IdealOfPoints\(I-9.7 pg.147\)](#), [IdealOfProjectivePoints\(I-9.8 pg.147\)](#), [Interpolate\(I-9.31 pg.158\)](#), [SeparatorsOfPoints\(I-19.8 pg.300\)](#)

I-19.10 SetCol

syntax

```
SetCol(ref M: MAT, j: INT, L: LIST)
```

This procedure sets the elements in “L” as entries of the “j”-th column in the matrix “M”; it returns nothing!

example

```
/**/ M := IdentityMat(QQ, 5);
/**/ SetCol(ref M, 1, [2,3,4,5,6]);
/**/ M;
matrix(QQ,
  [[2, 0, 0, 0, 0],
   [3, 1, 0, 0, 0],
   [4, 0, 1, 0, 0],
   [5, 0, 0, 1, 0],
   [6, 0, 0, 0, 1]])
```

See Also: [ref\(I-18.30 pg.281\)](#), [GetCol\(I-7.12 pg.126\)](#), [SetRow\(I-19.12 pg.303\)](#), [SwapCols\(I-19.64 pg.322\)](#)

I-19.11 SetEntry

syntax

```
SetEntry(ref M: MAT, i: INT, j: INT, val: RAT)
SetEntry(ref M: MAT, i: INT, j: INT, val: RINGELEM)
```

This procedure sets the value “val” as i,j-entry in the matrix “M”; it returns nothing! It is equivalent to “M[i,j] := val;”, but is offered as the CoCoA-5 counterpart of the syntax needed in C++ for CoCoALib.

example

```
/**/ M := IdentityMat(QQ, 5);
/**/ SetEntry(ref M, 1, 4, 7/2);
/**/ M;
matrix(QQ,
  [[1, 0, 0, 7/2, 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-18.30](#) pg.281), GetCol([I-7.12](#) pg.126), SetRow([I-19.12](#) pg.303), SwapCols([I-19.64](#) pg.322)

I-19.12 SetRow

syntax

```
SetRow(ref M: MAT, i: INT, L: LIST)
```

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, [2,3,4,5,6]);
/**/ M;
matrix(QQ,
  [[2, 3, 4, 5, 6],
   [0, 1, 0, 0, 0],
   [0, 0, 1, 0, 0],
   [0, 0, 0, 1, 0],
   [0, 0, 0, 0, 1]])
```

See Also: ref([I-18.30](#) pg.281), GetRow([I-7.17](#) pg.127), SetCol([I-19.10](#) pg.302), SwapRows([I-19.65](#) pg.323)

I-19.13 SetStackSize

syntax

```
SetStackSize(NewSize: INT)
```

Secret and dangerous.

I-19.14 SetVerbosityLevel

syntax

```
SetVerbosityLevel(N: INT)
```

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.339](#)) 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\(I-22.2 pg.339\)](#)

I-19.15 shape

syntax

```
shape(E: LIST): LIST (of TYPE)
shape(E: RECORD): RECORD (of TYPE)
shape(E: OTHER): TYPE
```

“shape” behaves like “type” ([I-20.20 pg.334](#)) except for LIST and RECORD, where “shape” recurses into the structure.

LIST: gives [shape(obj) | obj in E]

RECORD: gives record[field1 := shape(E.field1), ...]
retaining the original field names

example

```
/**/ use R ::= QQ[x];
/**/ L := [1,[1,"a"], x^2-x];
/**/ shape(L);
[INT, [INT, STRING], RINGELEM]

/**/ R := record[name := "test", contents := L];
/**/ shape(R);
record[contents := [INT, [INT, STRING], RINGELEM], name := STRING]
```

See Also: [type\(I-20.20 pg.334\)](#)

I-19.16 sign

syntax

```
sign(X: INT|RAT): INT
```

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
```

See Also: [abs\(I-1.1 pg.31\)](#)

I-19.17 SimplestBinaryRatBetween

syntax

```
SimplestBinaryRatBetween(A: RAT, B: RAT): RAT
```

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-19.18 pg.305](#)).

example

```
/**/  SimplestBinaryRatBetween(0.123, 0.456);
1/4

/**/  SimplestBinaryRatBetween(-3.14159, -2.71828);
-3

/**/  SimplestBinaryRatBetween(5,10); // contrast with SimplestRatBetween!
8
```

See Also: [CFAprox\(I-3.10 pg.60\)](#), [FloatApprox\(I-6.15 pg.110\)](#), [SimplestRatBetween\(I-19.18 pg.305\)](#)

I-19.18 SimplestRatBetween

syntax

```
SimplestRatBetween(A: RAT, B: RAT): RAT
```

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\(I-3.10 pg.60\)](#), [FloatApprox\(I-6.15 pg.110\)](#), [SimplestBinaryRatBetween\(I-19.17 pg.305\)](#)

I-19.19 SimplexInfo

syntax

```
SimplexInfo(A: LIST): RECORD
```

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-6.36 pg.118), SimplicialHomology(I-19.20 pg.306)

I-19.20 SimplicialHomology

syntax

```

SimplicialHomology(A: LIST): RECORD
SimplicialHomology(A: LIST, B: LIST): RECORD

```

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-19.21 singular value decomposition

Oh! Don't use Singular! CoCoA is nicer ;-)

Singular value decomposition is not yet implemented.

I-19.22 size [OBSOLETE]

[OBSOLETE] see “len” (I-12.7 pg.195).

See Also: len(I-12.7 pg.195)

I-19.23 skip

— syntax —

```
skip
```

This command does nothing. I suppose it might be used to make the structure of a user-defined function more clear.

— example —

```
/**/ skip;
```

I-19.24 SleepFor

— syntax —

```
SleepFor(zzz: INT or RAT)
```

This command makes CoCoA sleep for (at least) “zzz” seconds. It is an error if the duration is negative or too long (current limit is about 65000 seconds).

The command may be useful when developing an interactive program.

— example —

```
/**/ SleepFor(1.23);
```

See Also: PlayCantStop(I-16.11 pg.250)

I-19.25 SmallestNonDivisor

— syntax —

```
SmallestNonDivisor(N: INT): INT
```

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-9.85 pg.177), IsProbPrime(I-9.87 pg.178), FactorINT(I-6.4 pg.106)

I-19.26 SmoothFactor [OBSOLESCENT]

syntax

```
SmoothFactor(N: INT, MaxP: INT): RECORD
```

Renamed: see “FactorINT” (I-6.4 pg.106) in particular “TrialDiv” version.

I-19.27 SolomonTeraoIdeal

syntax

```
SolomonTeraoIdeal(A: LIST, f: RINGELEM): IDEAL
```

This function returns the Solomon-Terao ideal of the list A of hyperplanes of an arrangement respect to a polynomial f.

example

```
/**/ use QQ[x,y];
/**/ A := [x, y, x-y];
/**/ f := x^2+y^2;
/**/ SolomonTeraoIdeal(A, f);
ideal(2*x^2 +2*y^2, 2*x*y^2 -2*y^3)
```

See Also: OrlikTeraoIdeal(I-15.12 pg.246), ArtinianOrlikTeraoIdeal(I-1.48 pg.46)

I-19.28 sort

syntax

```
sort(ref L: LIST)
```

This procedure sorts the elements of the list in “L” with respect to the default comparisons related to their types; it overwrites “L” and returns nothing.

For more on the default comparisons, see “Relational Operators” (II-4.3 pg.362) in the chapter on operators. For sorting with respect to your own ordering, see “SortBy” (I-19.29 pg.308), “SortedBy” (I-19.31 pg.310).

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-4.3 pg.362), sorted(I-19.30 pg.309), SortBy(I-19.29 pg.308), SortedBy(I-19.31 pg.310)

I-19.29 SortBy

syntax

```
SortBy(ref L: LIST, LessThanFunc: FUNCTION)
```

This procedure sorts the elements of the list in “L” into increasing order with respect to the comparisons made by “LessThanFunc”; it overwrites “L” and returns nothing.

The comparison function “LessThanFunc” takes two arguments and must return “true” if the first argument is less than the second, otherwise it must return “false”.

If you want to call “SortBy(ref L, LessThanFunc)” inside a function, and you have defined “LessThanFunc” at top level (rather than as an anonymous function), then you need to ensure that the name “LessThanFunc” is visible using “TopLevel LessThanFunc;”

NOTE: if both “LessThanFunc(A, B)” and “LessThanFunc(B, A)” return “true”, then “A” and “B” are viewed as being equivalent.

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-6.33 pg.118), MinBy(I-13.17 pg.211), MaxBy(I-13.13 pg.210), sort(I-19.28 pg.308), sorted(I-19.30 pg.309), SortedBy(I-19.31 pg.310), TopLevel(I-20.10 pg.329)

I-19.30 sorted

syntax

```
sorted(L: LIST): LIST
```

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-4.3 pg.362) in the chapter on operators. For sorting with respect to your own ordering, see “SortBy” (I-19.29 pg.308), “SortedBy” (I-19.31 pg.310).

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-4.3 pg.362), SortBy(I-19.29 pg.308), SortedBy(I-19.31 pg.310), sort(I-19.28 pg.308)

I-19.31 SortedBy

syntax

```
SortedBy(L: LIST, LessThanFn: FUNCTION): LIST
```

This function returns the list of the sorted elements of “L” without affecting “L”, itself. As for “SortBy” (I-19.29 pg.308), the comparison function “LessThanFn” 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: if both “LessThanFn(A, B)” and “LessThanFn(B, A)” return “true”, then “A” and “B” are viewed as being equivalent.

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-6.33 pg.118), MinBy(I-13.17 pg.211), MaxBy(I-13.13 pg.210), sort(I-19.28 pg.308), sorted(I-19.30 pg.309), SortBy(I-19.29 pg.308)

I-19.32 source

syntax

```
source S: STRING
```

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-8.1 pg.371), Introduction to Packages(II-9.1 pg.375)

I-19.33 SourceRegion

syntax

```
SourceRegion FromLine: INT,FromChar: INT To ToLine: INT,ToChar: INT In S: STRING
```

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\(I-19.32 pg.310\)](#)

I-19.34 spaces

syntax

```
spaces(N: INT): STRING
```

This function returns a string consisting of N spaces.

example

```
/**/ L := "a" + Spaces(5) + "b";
/**/ L;
a      b
```

See Also: [dashes\(I-4.1 pg.83\)](#)

I-19.35 sprint

syntax

```
sprint(E: OBJECT): STRING
```

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; see also “**fold**” ([I-6.21 pg.113](#)).

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
```

See Also: [fold\(I-6.21 pg.113\)](#), [SprintTrunc\(I-19.36 pg.311\)](#), [print\(I-16.40 pg.260\)](#), [println\(I-16.45 pg.263\)](#), [Introduction to IO\(II-8.1 pg.371\)](#)

I-19.36 SprintTrunc

syntax

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

This function works like “**sprint**” (I-19.35 pg.311), 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(987*x+123*y)^4;
/**/ SprintTrunc(I, 20);
ideal(949005240561*x...
/**/ SprintTrunc(gens(I)[1], 20);
949005240561*x^4 +47...
```

See Also: `format`(I-6.25 pg.115), `sprint`(I-19.35 pg.311)

I-19.37 SqFreeFactor

— syntax —

```
SqFreeFactor(F: RINGELEM): RECORD
```

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]
]
/**/ use ZZ/(3)[x,y];
/**/ SqFreeFactor((x^3+y)*(x+y^3)).factors;
[x^3 +y, y^3 +x]
```

See Also: `radical`(I-18.1 pg.271), `factor`(I-6.1 pg.105), `ContentFreeFactor`(I-3.52 pg.75)

I-19.38 StableBBasis5

— syntax —

```
StableBBasis5(Pts: LIST, Toler: LIST): RECORD
StableBBasis5(Pts: LIST, Toler: LIST, Gamma: RAT): RECORD
```

***** NOT YET IMPLEMENTED ***** See “ApproxPointsNBM” (I-1.16 pg.36)

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” ≥ 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 —

```

***** NOT YET IMPLEMENTED ***** See ApproxPointsNBM
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\(I-9.7 pg.147\)](#), [ApproxPointsNBM\(I-1.16 pg.36\)](#)

I-19.39 *StableIdeal*

— syntax —

```
StableIdeal(L: LIST of power-products): IDEAL
```

This function returns the smallest stable ideal containing the power-products in “L” (see also “[StronglyStableIdeal](#)” ([I-19.48 pg.316](#))).

— 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\(I-9.97 pg.181\)](#), [LexSegmentIdeal\(I-12.9 pg.196\)](#), [StronglyStableIdeal\(I-19.48 pg.316\)](#)

I-19.40 *StagedTrees*

— syntax —

```
StagedTrees(L: LIST of power-products): IDEAL
```

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\(I-9.97 pg.181\)](#), [LexSegmentIdeal\(I-12.9 pg.196\)](#), [StronglyStableIdeal\(I-19.48 pg.316\)](#)

I-19.41 StandardInput

syntax

```
StandardInput(): ISTREAM
```

This function returns an “ISTREAM” for reading interactive input: see “[GetLine\(I-7.16 pg.127\)](#)”.

Warning: this function may not work well if you are running CoCoA inside a GUI.

example

```

// UserInput := StandardInput();
// str := GetLine(UserInput);
// println "The input was: ", str;

```

See Also: [GetLine\(I-7.16 pg.127\)](#), [OpenIFile\(I-15.2 pg.241\)](#), [OpenIString\(I-15.3 pg.242\)](#)

I-19.42 StandardOutput

syntax

```
StandardOutput(): OSTREAM
```

This function returns an “OSTREAM” for printing on the standard output (usually this is the computer screen).

This function may not work well if you are running CoCoA inside a GUI.

example

```
/**/ UserOutput := StandardOutput();
/**/ println "Hello" on UserOutput; -- same as println "Hello";
```

See Also: `print on` (I-16.41 pg.261), `OpenOfFile` (I-15.5 pg.243), `OpenOString` (I-15.6 pg.243), `StandardInput` (I-19.41 pg.314)

I-19.43 *StarRoot*

syntax

```
StarRoot(N: INT): INT
```

This function returns the smallest root of “N”; that is, it produces the smallest “R” such that “ $N = R^k$ ” for some “k”. An error is reported if “ $N < 0$ ”. Note that this is not the same as the radical or squarefree part of “N”.

example

```
/**/ StarRoot(8000);
20
```

See Also: `FactorINT` (I-6.4 pg.106), `FloorSqrt` (I-6.20 pg.112), `FloorRoot` (I-6.19 pg.112)

I-19.44 *starting*

syntax

```
starting(S: STRING): LIST of RECORD
```

This function returns a list of all CoCoA functions starting with the string “S”. This list may also include some undocumented functions; be wary of using these (there is probably a reason they are not documented!)

For each matching name there is a “RECORD” comprising two fields: “IsExported” and “name”. The field “IsExported” is “false” for user-defined and for built-in functions (implemented in C++), and “true” for functions exported from a CoCoA-5 package. The field “name” contains the full name of the function; for exported functions, this is of the form “\$PkgName.FnName”, where the definition of “FnName” can be found in the CoCoA-5 package “PkgName.cpkg”.

example

```
/**/ indent(starting("Subs"));
[
  record[IsExported := true, name := "$BackwardCompatible.Subsets"],
  record[IsExported := true, name := "$BackwardCompatible.Subst"]
]
```

See Also: `PackageOf` (I-16.2 pg.247), `print` (I-16.40 pg.260), `describe` (I-4.13 pg.89)

I-19.45 *StdBasis*

syntax

```
StdBasis(I: IDEAL): LIST
```

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.328) for more details).

The implementation is based on method of Lazard (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\(I-20.5 pg.328\)](#), [PrimaryHilbertSeries\(I-16.36 pg.259\)](#)

I-19.46 StdDegLexMat

syntax

```
StdDegLexMat(N: INT): MAT
```

This function returns the matrix defining a standard term-ordering.

example

```

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

```

See Also: [OrdMat\(I-15.11 pg.245\)](#), [Term Orderings\(III-9.5 pg.422\)](#), [StdDegRevLexMat\(I-19.47 pg.316\)](#), [LexMat\(I-12.8 pg.196\)](#), [RevLexMat\(I-18.45 pg.287\)](#), [XelMat\(I-24.1 pg.343\)](#)

I-19.47 StdDegRevLexMat

syntax

```
StdDegRevLexMat(N: INT): MAT
```

This function returns the matrix defining a standard term-ordering.

example

```

/**/ StdDegRevLexMat(3);
matrix(ZZ,
[[1, 1, 1],
[0, 0, -1],
[0, -1, 0]])

```

See Also: [OrdMat\(I-15.11 pg.245\)](#), [Term Orderings\(III-9.5 pg.422\)](#), [StdDegLexMat\(I-19.46 pg.316\)](#), [LexMat\(I-12.8 pg.196\)](#), [RevLexMat\(I-18.45 pg.287\)](#), [XelMat\(I-24.1 pg.343\)](#)

I-19.48 StronglyStableIdeal

syntax

```
StronglyStableIdeal(L: LIST of power-products): IDEAL
```

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-9.99 pg.181), LexSegmentIdeal(I-12.9 pg.196), StableIdeal(I-19.39 pg.313)

I-19.49 SturmSeq

syntax

```
SturmSeq(f: RINGELEM): LIST of RINGELEM
```

This function returns a scaled Sturm sequence of the univariate polynomial “f” which must have rational coefficients.

example

```

/**/ use R := QQ[x];
/**/ f := 3*x^3-5*x^2+7*x-11;
/**/ SturmSeq(f);
[3*x^3 -5*x^2 +7*x -11, 9*x^2 -10*x +7, -38*x +131, -1]

```

See Also: RealRoots(I-18.25 pg.279), NumRealRoots(I-14.43 pg.238)

I-19.50 SubalgebraHom

syntax

```
SubalgebraHom(R: RING, L: LIST): RINGHOM
```

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-16.29 pg.256), ker(I-11.1 pg.191)

I-19.51 SubalgebraMap [OBSOLETE]

[OBSOLETE] See “SubalgebraHom” (I-19.50 pg.317).

See Also: SubalgebraHom(I-19.50 pg.317)

I-19.52 SubalgebraMinGens

— syntax —

```
SubalgebraMinGens(L: LIST): LIST
```

If “L” is a list of polynomials, this function returns a list of minimal generators for the algebra “K[L]” (not necessarily a subset of “L”).

Verbosity range 40-90.

— example —

```
/**/ use P:=QQ[x,y,z];
/**/ f := 3*x^2;
/**/ g := x*y -z^2;
/**/ h := f^2*g -5*g^3;
/**/ SubalgebraMinGens([f, g, h]);
[x*y -z^2, x^2]
```

See Also: MinGens(I-13.19 pg.212), SAGBI, SAGBIHomog(I-19.1 pg.297)

I-19.53 SubalgebraRepr [OBSOLESCENT]

[OBSOLESCENT] See “preimage0” (I-16.29 pg.256).

See Also: SubalgebraHom(I-19.50 pg.317), preimage0(I-16.29 pg.256), ker(I-11.1 pg.191)

I-19.54 submat

— syntax —

```
submat(M: MAT, R: LIST of INT, C: LIST of INT): MAT
```

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.

There are also two handy shortcuts: “FirstCols, FirstRows” (I-6.10 pg.108).

— 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]])

/**/ FirstCols(M,2);
matrix(QQ,
  [[1, 2],
   [4, 5]])
```

See Also: minors(I-13.24 pg.213), FirstCols, FirstRows(I-6.10 pg.108)

I-19.55 submodule

syntax

```
submodule(L: LIST of MODULEELEM): MODULE
submodule(F: MODULE, L: LIST of MODULEELEM): MODULE
```

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-19.56 pg.319) for creating a module from the rows or columns of a matrix.

NOTE: the generators are 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-13.32 pg.216), SubmoduleCols, SubmoduleRows(I-19.56 pg.319), GensAsCols, GensAsRows(I-7.9 pg.125), gens(I-7.8 pg.124)

I-19.56 SubmoduleCols, SubmoduleRows

syntax

```
SubmoduleCols(F: MODULE, M: MATRIX): MODULE
SubmoduleRows(F: MODULE, M: MATRIX): MODULE
```

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.125), submodule(I-19.55 pg.319), ModuleElem(I-13.31 pg.216)

I-19.57 SubmoduleOfMinGens

syntax

```
SubmoduleOfMinGens(M: MODULE): MODULE
```

It works only in the homogeneous case: for the inhomogeneous case see “[MinSubsetOfGens](#)” ([I-13.28](#) pg.215). 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.146).

— 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-13.19](#) pg.212), [MinSubsetOfGens](#)([I-13.28](#) pg.215)

I-19.58 subsets

— syntax —

```
subsets(S: LIST): LIST
subsets(S: LIST, N: INT): LIST
```

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-9.100](#) pg.182), [partitions](#)([I-16.6](#) pg.248), [permutations](#)([I-16.7](#) pg.248), [MakeSet](#)([I-13.3](#) pg.206), [tuples](#)([I-20.15](#) pg.332)

I-19.59 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
```

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-5.14](#) pg.100).

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-5.14](#) pg.100), Evaluation of Polynomials([III-11.2](#) pg.430), PolyAlgebraHom([I-16.17](#) pg.251), QZP([I-17.7](#) pg.269), RingElem([I-18.47](#) pg.288), ZPQ([I-25.3](#) pg.346)

I-19.60 substring

syntax

```
substring(S: STRING, StartPosn: INT, Length: INT): STRING
```

This function extracts a substring (with given start posn and length) from a string. It is an error if the start posn is not inside the string, and also if the length is negative.

example

```

/**/ substring("abcdef", 2, 3);
bcd

```

See Also: concat([I-3.41](#) pg.71)

I-19.61 sum

syntax

```

sum(L: LIST): OBJECT
sum(L: LIST, InitVal: OBJECT): OBJECT

```

This function returns the sum of the objects in the list “L” (together with “InitVal”, if specified). 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 empty STRING

```

```
/**/ sum([], zero(R)); -- gives type RINGELEM
0
```

See Also: Algebraic Operators([II-4.2](#) pg.361), product([I-16.48](#) pg.264)

I-19.62 support

— syntax —

```
support(F: RINGELEM): LIST
support(F: MODULEELEM): LIST
```

This function returns the list of terms of F. To get a list of monomials, which includes coefficients, use “monomials” ([I-13.36](#) pg.218).

— 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-3.27](#) pg.66), monomials([I-13.36](#) pg.218)

I-19.63 swap

— syntax —

```
swap(ref A: OBJECT, ref B: OBJECT)
```

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-18.30](#) pg.281)

I-19.64 SwapCols

— syntax —

```
SwapCols(ref M: MAT, i: INT, j: INT)
```

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\(I-18.30 pg.281\)](#), [swap\(I-19.63 pg.322\)](#), [GetCol\(I-7.12 pg.126\)](#), [SetCol\(I-19.10 pg.302\)](#)

I-19.65 SwapRows

— syntax —

```
SwapRows(ref M: MAT, i: INT, j: INT)
```

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\(I-18.30 pg.281\)](#), [swap\(I-19.63 pg.322\)](#), [GetRow\(I-7.17 pg.127\)](#), [SetRow\(I-19.12 pg.303\)](#)

I-19.66 SwinnertonDyerPoly

— syntax —

```
SwinnertonDyerPoly(x: RINGELEM, RootList: LIST of INT): RINGELEM
```

This function computes the Swinnerton-Dyer polynomial whose roots are the sum of all square-roots of the elements in “RootList”. The result is irreducible if “RootList” contains distinct prime numbers.

— example —

```
/**/ use QQ[x];
/**/ SwinnertonDyerPoly(x, [2,3]); --> roots are +/- sqrt(2) +/- sqrt(3)
x^4 -10*x^2 +1
/**/ SwinnertonDyerPoly(x, [2,2,2]); -- you may also repeat the roots
x^8 -24*x^6 +120*x^4 -224*x^2 +144
```

See Also: [factor\(I-6.1 pg.105\)](#), [RealRoots\(I-18.25 pg.279\)](#), [resultant\(I-18.42 pg.286\)](#)

I-19.67 SylvesterMat

— syntax —

```
SylvesterMat(F: RINGELEM, G: RINGELEM, X: RINGELEM): MAT
```

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);
/**/ SylvesterMat(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(SylvesterMat(F, G, x)) = resultant(F, G, x);
true

```

See Also: [resultant\(I-18.42 pg.286\)](#)

I-19.68 SymbolRange

syntax

```

SymbolRange(H: STRING, lo: INT, hi: INT): LIST of RECORD
SymbolRange(H: STRING, lo: LIST of INT, hi: LIST of INT): LIST of RECORD

```

This function is designed to work well with “NewPolyRing” ([I-14.9 pg.226](#)). It returns the list of the symbols with head given by the string “H” and the range of indices. A symbol is represented as a record with “head” (as produced by “IndetName” ([I-9.23 pg.154](#))) and “indices” (as produced by “IndetSubscripts” ([I-9.26 pg.156](#))).

example

```

/**/ P := NewPolyRing(QQ, SymbolRange("x", 0,4));
/**/ indets(P);
[x[0], x[1], x[2], x[3], x[4]]

/**/ P2 := NewPolyRing(QQ, SymbolRange("mu", [3,1], [5,2]));
/**/ indets(P2);
[mu[3,1], mu[3,2], mu[4,1], mu[4,2], mu[5,1], mu[5,2]]

```

See Also: [NewPolyRing\(I-14.9 pg.226\)](#), [indet\(I-9.21 pg.153\)](#), [IndetSubscripts\(I-9.26 pg.156\)](#), [IndetIndex\(I-9.22 pg.154\)](#), [IndetName\(I-9.23 pg.154\)](#), [NumIndets\(I-14.41 pg.237\)](#)

I-19.69 SymmetricPolys

syntax

```

SymmetricPolys(P: RING): LIST of RINGELEM

```

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-19.70 SystemCommand

syntax

```

SystemCommand(CMD: STRING): INT

```

This function will work only if the CoCoA interpreter was started with the “**enable system command**” flag set; by default the flag is not set since system commands can be used maliciously (*e.g.* to delete files).

This function expects a string containing a **system command** (*e.g.* a shell command on Linux platforms). The string “CMD” is passed to the underlying operating system command line interpreter for execution. The return value of “SystemCommand” depends on the platform (*i.e.* the C++ run-time environment), so should not be relied upon to provide any specific information: *e.g.* on some Linux systems the value returned is the exit code, on others it is 256 times the process exit code.

example

```
/**/ ExitCode := SystemCommand("echo abc"); --> value of ExitCode is unclear
abc
```

I-19.71 syz

syntax

```
syz(F: freeMODULE, L: LIST of RINGELEM): MODULE
syz(L: LIST of RINGELEM): MODULE
syz(M: IDEAL|MODULE, Index: INT): MODULE
```

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]
))
-----
/**/ L := [x^2, 0, 0, y];
/**/ I := ideal(L); -- 0 gens are removed
/**/ gens(I); --> [x^2, y]
/**/ syz(gens(I)); -- same as SyzOfGens(I);
submodule(FreeModule(..), [[y, -x^2]])

-- /**/ syz(L); --> !!! ERROR !!! as expected: 0 entries
-- for dealing with 0 entries we need to specify the FreeModule
/**/ F := NewFreeModule(R, 4);
/**/ indent(syz(F, L));
SubmoduleRows(F, matrix([
  [0, 1, 0, 0],
  [0, 0, 1, 0],
  [y, 0, 0, -x^2]
]))

/**/ FwShifts := NewFreeModuleForSyz([x^2, x^100, 1, y]);
/**/ S := syz(FwShifts, L); indent(S); --> prints "F" for free module
SubmoduleRows(F, matrix([
  [0, 1, 0, 0],
```

```

    [0, 0, 1, 0],
    [y, 0, 0, -x^2]
]))
/**/ [ wdeg(v) | v in gens(S) ];
[[100], [0], [3]]
ModuleOf(S); --> FreeModule(RingWithID(168, "QQ[x,y,z]"), 4)
-----
/**/ I := ideal(x^2-y*z, x*y-z^2, x*y*z);
/**/ indent(res(I));
-----
-- syz(I,i) for module in minimal resolution: NOT YET IMPLEMENTED

```

See Also: [res\(I-18.38 pg.285\)](#), [SyzOfGens\(I-19.72 pg.326\)](#)

I-19.72 SyzOfGens

syntax

```
SyzOfGens(M: IDEAL|MODULE): MODULE
```

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([
    [y, -y, -1]
]))

```

See Also: [syz\(I-19.71 pg.325\)](#)

Chapter I-20

T

I-20.1 tag

syntax

```
tag(E: OBJECT): STRING
```

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.447](#)), tagged([I-20.2 pg.327](#)), untagged([I-21.5 pg.336](#))

I-20.2 tagged

syntax

```
tagged(E: OBJECT, S: STRING): TAGGED(S)
```

This 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.447](#)), tag([I-20.1 pg.327](#)), untagged([I-21.5 pg.336](#))

I-20.3 tail

syntax

```
tail(L: LIST): LIST
```

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-6.9 pg.108), last(I-12.2 pg.193)

I-20.4 TensorMat [OBSOLESCENT]

Renamed to “KroneckerProd” (I-11.2 pg.191).

I-20.5 TgCone

syntax

```
TgCone(I: IDEAL): IDEAL
```

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-12.10 pg.196), “DF” (I-4.15 pg.90)).

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-9.29 pg.157), PrimaryHilbertSeries(I-16.36 pg.259)

I-20.6 ThmProve [PROTOTYPE]

syntax

```
ThmProve(Hypothesis: IDEAL of RINGELEM, Thesis: IDEAL of RINGELEM): RECORD
```

This is just a prototype: exact semantics, and interface are likely to change!

example

```
/**/ ThmMan();  
/**/ ThmExample_FeetAndMidpoint();  
/**/ ThmExample_Pappus();  
/**/ ThmExample_DegenerateParallelogram();
```


I-20.7 TimeFrom

syntax

```
TimeFrom(StartPoint: RAT): STRING
```

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-3.61 pg.79) 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-20.8 TimeOfDay

syntax

```
TimeOfDay(): INT
```

This function returns the current time as an INT in the form $HHMMSS = HH * 10000 + MM * 100 + SS$ using a 24-hour clock.

NOTE: from version 5.0.4 this information is no longer given by the function “date” (I-4.2 pg.83).

example

```
/**/ TimeOfDay(); -- 09:08:13
90813
/**/ date();      -- 2013-05-30
20130530
```

See Also: date(I-4.2 pg.83)

I-20.9 TmpNBM [OBSOLETE]

Renamed to “ApproxPointsNBM” (I-1.16 pg.36).

I-20.10 TopLevel

syntax

```
TopLevel X;
  where ‘\verb&X&’ is the name of a top-level variable or function.
```

This command makes a top-level variable accessible from inside a function. For instance, it is useful for making the rings “QQ” (I-17.1 pg.267) and “ZZ” (I-25.4 pg.346) accessible, and also if a top-level function is to be passed as a parameter (e.g. to the function “SortBy” (I-19.29 pg.308)). It is poor style to use “TopLevel” for purposes other than these.

A fully qualified package variable can be used directly in a function; the “TopLevel” command is **not needed in this case** (and gives an error if a fully qualified package name is specified).

See also the commands “ImportByRef, ImportByValue” (I-9.17 pg.152) if you want to access a non-top-level variable inside an anonymous function.

example

```
/**/ define BeautifulRing(N)
/**/   TopLevel QQ;
```

```

/**/  R ::= QQ[b,e,a,u,t,y];
/**/  return R;
/**/  enddefine;

/**/  define CompareLen(X,Y) return len(X) < len(Y); EndDefine;
/**/
/**/  define LongestName(ListOfNameAndValue)
/**/    TopLevel CompareLen; --> to pass it as parameter 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\(I-6.33 pg.118\)](#), [ImportByRef](#), [ImportByValue\(I-9.17 pg.152\)](#)

I-20.11 TopLevelFunctions

syntax

TopLevelFunctions(): LIST of FUNCTION

This function returns the list of all functions available at top-level

example

```

/**/  indent(TopLevelFunctions());
[
  record[IsExported := false, name := "$ApproxSolve.ApplyShapeLemma"],
  record[IsExported := true, name := "$ApproxSolve.ApproxSolve"],
  ...
  record[IsExported := false, name := "zero"]
]

```

I-20.12 toric

syntax

```

toric(I: IDEAL): IDEAL
toric(I: IDEAL, L: LIST of INDETS): IDEAL
toric(M: MAT|LIST of LIST): IDEAL
toric(M: LIST of PP): IDEAL

```

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 **EATI**. 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)

/**/ toric(mat([[1,3,2],[3,-4,8]]));
ideal(-x^32 +y^2*z^13)

```

I-20.13 *transposed*

syntax

```
transposed(M: MAT): MAT
```

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.
```

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-7.15 pg.126). 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-5.12 pg.99), GetErrMsg(I-7.15 pg.126), All CoCoA commands(II-2.2 pg.357)

I-20.15 tuples

syntax

```
tuples(S: LIST, N: INT): LIST
```

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-3.6 pg.59), permutations(I-16.7 pg.248), subsets(I-19.58 pg.320)

I-20.16 TVecFromHF

syntax

```
TVecFromHF(H: TAGGED): LIST
TVecFromHF(H: LIST): LIST
```

This function returns a LIST representing the type vector corresponding to the Hilbert function “H”.

This is part of the CoCoA package “TypeVectors” originally by E.Carlini, M.Stewart for computing with **type vectors** as described in A.Geramita, T.Harima, Y.Shin **An alternative to the Hilbert function for the ideal of a finite set of points in P^n** Illinois J.Math. vol.45 (2001), no. 1, pages 1–23.

example

```

/**/ Use P ::= QQ[x,y,z,t];
/**/ HF := HilbertFn(P/Ideal(x, y^3, z^2-t*x)); HF;
/**/ TV := TVecFromHF(HF); TV;
[[[2], [4]]]
/**/ TV := TVecFromHF([[1, 3, 5], 6]); TV;
[[[2], [4]]]

```

See Also: TVecToHF(I-20.19 pg.333), TVecPrintRes(I-20.18 pg.333), TVecPoints(I-20.17 pg.333)

I-20.17 TVecPoints

syntax

TVecToHF(TV: LIST): TAGGED

This function return the points associated to the type vector “TV”.

See “TVecFromHF” (I-20.16 pg.332).

example

```

/**/ Use P ::= QQ[x,y,z,t];
/**/ HF := HilbertFn(P/Ideal(x, y^3, z^2-t*x)); HF;
/**/ TV := TVecFromHF(HF); TV;
/**/ Pts := TVecPoints(TV); indent(Pts);
[
  [0, 0, 0, 1],
  [0, 0, 1, 1],
  [0, 0, 2, 1],
  [0, 0, 3, 1],
  [0, 1, 0, 1],
  [0, 1, 1, 1]
]

```

See Also: TVecFromHF(I-20.16 pg.332), TVecPrintRes(I-20.18 pg.333), TVecToHF(I-20.19 pg.333)

I-20.18 TVecPrintRes

syntax

TVecPrintRes(TV: LIST)

This function prints the resolution associated to the type vector “TV”, *i.e.* of the ideal of a K-configuration described by “TV”.

See “TVecFromHF” (I-20.16 pg.332).

example

```

/**/ Use P ::= QQ[x,y,z,t];
/**/ HF := HilbertFn(P/Ideal(x, y^3, z^2-t*x)); HF;
/**/ TV := TVecFromHF(HF); TV;
/**/ TVecPrintRes(TV);
0 --> R(-5)(+)R(-6) --> R(-3)(+)R^2(-4)(+)R^2(-5) --> R(-1)(+)R(-2)(+)R(-3)(+)R(-4) --> R

```

See Also: TVecFromHF(I-20.16 pg.332), TVecPoints(I-20.17 pg.333), TVecToHF(I-20.19 pg.333)

I-20.19 TVecToHF

syntax

TVecToHF(TV: LIST): TAGGED

This function returns the Hilbert function associated to the type vector “TV”.

See “TVecFromHF” ([I-20.16 pg.332](#))

example

```
/**/ Use P ::= QQ[x,y,z,t];
/**/ T := TVecFromHF([[1, 3, 5], 6]); T;
[[[2], [4]]]
/**/ TVecToHF([[2], [4]]);
H(0) = 1
H(1) = 3
H(2) = 5
H(t) = 6 for t >= 3
```

See Also: TVecFromHF([I-20.16 pg.332](#)), TVecPrintRes([I-20.18 pg.333](#)), TVecPoints([I-20.17 pg.333](#))

I-20.20 type

syntax

```
type(E: OBJECT): TYPE
```

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: shape([I-19.15 pg.304](#)), CurrentTypes([I-3.65 pg.80](#))

Chapter I-21

U

I-21.1 UnivariateIndetIndex

syntax

```
UnivariateIndetIndex(F: RINGELEM): INT
```

This function returns 0 if the polynomial “F” is not univariate otherwise it returns the indeterminate index of “F”. If “F” is constant, the function throws an error.

example

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

/**/ UnivariateIndetIndex(x-y-1);
0
```

See Also: [indet\(I-9.21 pg.153\)](#), [IndetSubscripts\(I-9.26 pg.156\)](#), [IndetIndex\(I-9.22 pg.154\)](#), [IndetName\(I-9.23 pg.154\)](#), [indets\(I-9.24 pg.155\)](#), [NumIndets\(I-14.41 pg.237\)](#)

I-21.2 UniversalGBasis

syntax

```
UniversalGBasis(I: IDEAL): LIST of RINGELEM
```

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\(I-7.36 pg.132\)](#)

I-21.3 unprotect

syntax

```
unprotect X;
```

This command undoes the effect of the “**protect**” (I-16.49 pg.265) 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\(I-16.49 pg.265\)](#)

I-21.4 Unset [OBSOLETE]

[OBSOLETE]

I-21.5 untagged

syntax

```
untagged(E:TAGGED_OBJECT): OBJECT
```

This function strips an object “E” of its tag, if any.

Tags are used for pretty printing of objects: see the reference “Printing a Tagged Object” (III-16.2 pg.447).

NOTE: in CoCoA-4 the obsolete syntax “@E” should be replaced by “**untagged(E)**”.

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\(III-16.2 pg.447\)](#), [tag\(I-20.1 pg.327\)](#), [tagged\(I-20.2 pg.327\)](#)

I-21.6 use

syntax

```
use R
use RingDefn
use R ::= RingDefn
```

where R is a RING, and RingDefn is a ring definition.

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-18.47 pg.288) 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.427), CurrentRing(I-3.64 pg.80), RingOf(I-18.51 pg.290), RingElem(I-18.47 pg.288)

Chapter I-22

V

I-22.1 valuation [OBSOLETE]

Renamed to “FactorMultiplicity” (I-6.5 pg.107).

See Also: FactorMultiplicity(I-6.5 pg.107)

I-22.2 VerbosityLevel

syntax

```
VerbosityLevel(): INT
```

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;

/**/ use P ::= QQ[x,y,z];
/**/ 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-19.14 pg.303)

I-22.3 VersionInfo

— syntax —

```
VersionInfo(): RECORD
```

This function returns a record with various information about CoCoA and CoCoALib (the mathematical core of CoCoA)

— example —

```
/**/ indent(VersionInfo(),2);
record[
  CoCoALibVersion := "0.99***",
  CoCoAVersion := "5.*.*",
  CompilationDate := ....,
  ...
  ExternalLibs := [...]
```

See Also: CoCoALib([II-10.1](#) pg.381), RelNotes([I-18.34](#) pg.283)

Chapter I-23

W

I-23.1 wdeg

— syntax —

```
wdeg(F: RINGELEM): LIST
```

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.86) returns the standard degree.

NOTE: In CoCoA-4 “deg” (I-4.6 pg.86) 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 TOPos
/**/ 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.86), LF(I-12.10 pg.196)

I-23.2 WeightsMatrix [OBSOLESCENT]

Renamed to “GradingMat” (I-7.33 pg.131).

See Also: deg(I-4.6 pg.86), wdeg(I-23.1 pg.341)

I-23.3 while

syntax

```
While B Do C EndWhile
```

where B is a boolean expression and C is a sequence of commands.

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-6.23 pg.113](#)), foreach([I-6.24 pg.114](#)), repeat([I-18.37 pg.284](#)), All CoCoA commands([II-2.2 pg.357](#))

I-23.4 WithoutNth

syntax

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

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-18.36 pg.284](#)).

example

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

See Also: remove([I-18.36 pg.284](#))

I-23.5 WLog [OBSOLETE]

[OBSOLETE]

See Also: exponents([I-5.18 pg.101](#))

Chapter I-24

X

I-24.1 **XelMat**

— syntax —

```
XelMat(N: INT): MAT
```

This function returns the matrix defining a standard term-ordering.

— example —

```
/**/ XelMat(3);  
matrix(ZZ,  
  [[0, 0, 1],  
   [0, 1, 0],  
   [1, 0, 0]])
```

See Also: [OrdMat\(I-15.11 pg.245\)](#), [Term Orderings\(III-9.5 pg.422\)](#), [StdDegRevLexMat\(I-19.47 pg.316\)](#), [StdDegLexMat\(I-19.46 pg.316\)](#), [LexMat\(I-12.8 pg.196\)](#), [RevLexMat\(I-18.45 pg.287\)](#)

Chapter I-25

Z

I-25.1 zero

— syntax —

```
zero(R: RING): RINGELEM
```

This function returns 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\(I-15.1 pg.241\)](#), [IsZero\(I-9.107 pg.184\)](#)

I-25.2 ZeroMat

— syntax —

```
ZeroMat(R: RING, NumRows: INT, NumCols: INT): MAT
```

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\(I-13.11 pg.209\)](#), [IdentityMat\(I-9.9 pg.148\)](#), [NewMatFilled\(I-14.8 pg.225\)](#)

I-25.3 ZPQ

syntax

```
ZPQ(F: RINGELEM): RINGELEM
ZPQ(F: LIST of RINGELEM): LIST of RINGELEM
ZPQ(I: IDEAL): IDEAL
```

***** 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.269](#)) for more details.

example

```
***** NOT YET IMPLEMENTED *****
  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\(I-2.13 pg.55\)](#)

I-25.4 ZZ

syntax

```
ZZ: RING
```

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\(I-17.1 pg.267\)](#), [Quotient Rings\(III-9.7 pg.424\)](#)

Part II

CoCoA Tutorials and Programming Language

Chapter II-1

CoCoA Tutorials

II-1.1 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`” (I-3.17 pg.63). 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-5 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-5, you can try “`*/ciao;`” instead; note the extra two characters **star** and **slash** at the start. You may need to type this twice.

II-1.2 Tutorial: manual

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).

II-1.3 Tutorial: Emacs UI (basic)

The developers of CoCoA use the (GNU) emacs UI to CoCoA-5, so it is extensively tested. Other UIs may become available in the future.

Emacs is an extensible text editor with numerous functions. One of these extensions helps one edit files of CoCoA commands, and also run an interactive CoCoA-5 session. Many of the special functions for CoCoA-5 are available from a “drop-down” menu (under the title CoCoA-5).

We find the following approach convenient. Inside Emacs use the "Open File" function to open (or create) a file whose name **must** end in ".cocoa5". Type in your commands and function definitions into this file; then use "Source file into CoCoA-5" or "Source region into CoCoA-5". An advantage of this approach is that if you make a mistake, you can easily edit the file, and then send the corrected commands to CoCoA-5.

The functions can also be activated via suitable key-sequences; this is usually quicker than using the menus, but entails memorizing the most useful "key sequences". A key sequence usually looks cryptic: for example in the key sequence "C-M-"; the substring "M-" means to press and release the "Esc" key, while the substring "C-" means to press and hold the "Ctrl" key, finally you should press briefly the "\" (backslash) key.

Some more help for Emacs can be found online at the following link: "<http://cocoa.dima.unige.it/download/screenshot>".

See Also: Basic Tutorial for CoCoA-5 (II-1.1 pg.349)

II-1.4 Tutorial: variables, assignment

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"
```

II-1.5 Tutorial: arithmetic operators

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
```

II-1.6 Tutorial: printing

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 functions “`indent`, `IndentStr`” (I-9.20 pg.153) are useful for printing out long lists of values: they print 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
]
```

II-1.7 Tutorial: lists

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
```

II-1.8 Tutorial: polynomial rings, use command

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.336](#)) 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; note 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: polynomials([II-1.9 pg.352](#))

II-1.9 Tutorial: polynomials

If you are new to CoCoA-5, we recommend that you read first the “Tutorial: polynomial rings, use command” ([II-1.8 pg.352](#)).

A complete list of the functions and commands which operate on or return a polynomial can be obtained by looking up “RINGELEM” in the on-line manual.

In addition to the usual arithmetic operations, CoCoA-5 offers several functions for looking “inside” a polynomial. We give some examples here. Each non-zero polynomial has a leading monomial (see “`LM`” ([I-12.18 pg.200](#))), a leading coefficient (see “`LC`” ([I-12.5 pg.194](#))), and a leading term (see “`LT`” ([I-12.24 pg.202](#)) and “`LPP`” ([I-12.22 pg.201](#))).

One may also extract all monomials (see “`monomials`” ([I-13.36 pg.218](#))), all coefficients (see “`coefficients`” ([I-3.27 pg.66](#))), and all power-products (see “`support`” ([I-19.62 pg.322](#))). In each case the lists have the same order as in the polynomial itself.

There are more advanced functions for exploring the structure of a polynomial: see for instance “`CoefficientsWRT`” ([I-3.28 pg.67](#)), “`CoeffListWRT`” ([I-3.29 pg.67](#)), and “`CoeffOfTerm`” ([I-3.31 pg.68](#)).

example

```
/**/ use P := QQ[x,y]; // polys in x,y with coefficients in QQ
/**/ f := (2*x^2+y)^3; f;
8*x^6 +12*x^4*y +6*x^2*y^2 +y^3
/**/ [LM(f), LC(f), LT(f)];
[8*x^6, 8, x^6]
```



```

/**/ monomials(f);
[8*x^6, 12*x^4*y, 6*x^2*y^2, y^3]
/**/ coefficients(f);
[8, 12, 6, 1]
/**/ support(f);
[x^6, x^4*y, x^2*y^2, y^3]
/**/ CoeffListWRT(f,x);
[y^3, 0, 6*y^2, 0, 12*y, 0, 8]

```

II-1.10 Tutorial: defining new functions

To define a new function in CoCoA use the command “define” ([I-4.4 pg.84](#)).

When defining a new function you must pick a new name for the function, and state what arguments/parameters it expects. The parameters and variables used inside the function are different from any variables outside it (even if the names look the same).

Strictly there is a difference between “functions” and “procedures”: the former always return values, while the latter never return values. In CoCoA there is hardly any distinction: the only difference is the use of the “return” ([I-18.43 pg.287](#)) command inside the fn-proc.

example

```

/**/ -- Define new fn "SimpleFn" with just 1 parameter:
/**/ define SimpleFn(N) return N^2+1; enddefine;
/**/ SimpleFn(3);
10

/**/ define OneStep(N)
/**/   if IsEven(N) then return N/2;
/**/   else return 3*N+1;
/**/   endif;
/**/ enddefine;
/**/ OneStep(5);
16

/**/ define MaxAbs(X, Y) --> 2 parameters
/**/   return max(abs(X), abs(Y));
/**/ enddefine;
/**/ MaxAbs(2, -3);
3

/**/ -- Next defn is a "procedure" (returns no value)
/**/ define CheckIsPositive(X)
/**/   if X <= 0 then println "NOT POSITIVE"; endif
/**/ enddefine

```

II-1.11 Tutorial: defining new functions (advanced)

Please read “Tutorial: defining new functions” ([II-1.10 pg.353](#)) before reading this.

Inside a function it is possible to access variables which were defined outside it. The command “TopLevel” ([I-20.10 pg.329](#)) will make a global variable accessible from inside the function, and parameters may be passed “by reference”.

example

```

/**/ define negate(ref X) // pass by reference
/**/   X := -X;           // changes value of variable OUTSIDE the fn

```

```

/**/ enddefine;           // does not "return" a value, so is a procedure!
/**/ A := 1;
/**/ negate(ref A); // use "ref" also to call the fn
/**/ println A;      // now see that value has changed
-1

/**/ define QQx()
/**/   TopLevel QQ;    // make global variable QQ accessible
/**/   return NewPolyRing(QQ, "x");
/**/ enddefine;
/**/ QQx();
RingWithID(3, "QQ[x]")

```

See Also: Tutorial: defining new functions([II-1.10](#) pg.353)

II-1.12 Tutorial: homomorphisms

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. If the codomain is also a polynomial ring with the same ring of coefficients then use "PolyAlgebraHom" ([I-16.17](#) pg.251), otherwise use "PolyRingHom" ([I-16.18](#) pg.252) giving also the homomorphism saying how the coefficient ring is mapped.

Some "special" homomorphisms can be created easily via dedicated functions. If there is a canonical homomorphism between the rings then this may be specified using "CanonicalHom" ([I-3.4](#) pg.58). For the homomorphism which embed the coefficient ring into a polynomial ring use "CoeffEmbeddingHom" ([I-3.25](#) pg.65).

example

```

/**/ P1 := QQ[x,y]; // polys in x,y with coefficients in QQ
/**/ P2 := QQ[a,b]; // polys in a,b with coefficients in QQ
/**/ 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

```

II-1.13 Tutorial: programming and debugging

This tutorial is for those who have little experience in programming and debugging.

Despite your best efforts any non-trivial program may easily contain "bugs". When this happens one must somehow locate the fault, and then find a way to rectify it.

The CoCoA interpreter does not offer a special debugger.

A good guideline is to split your program into smaller pieces, each of which has a clearly defined purpose which can be described succinctly. Then these pieces can be tested, and fixed, independently. Understanding how to split the program becomes easier with more experience.

Try to find a simple input where your program misbehaves: this is not always easy, and may require quite some time! When you have a manageable troublesome input, you should insert some print commands in your code saying where execution has reached, and the values of some variables. For instance:

```

println "Start of function Blah: input A = ", A, " and B = ", B;
....
println "After doing XYZ: poly has degree ", deg(f);

```

The mistake might in the input, and the program is correct: double-check that the input really is what you intended!

It may also happen that the problem lies in CoCoA. Check carefully the CoCoA manual to see whether the function actually does what you think it does! If you are quite sure the problem lies inside CoCoA then see “Tutorial: feedback and reporting bugs” ([II-1.14](#) pg.355)

See Also: Tutorial: feedback and reporting bugs([II-1.14](#) pg.355)

II-1.14 Tutorial: feedback and reporting bugs

We try to make CoCoA natural and easy to use, and have everything properly described in the manual. However, there is always room for improvement, and we are happy to receive constructive criticism.

We try hard to make CoCoA reliable, but some bugs surely slip through our testing. So if you find an example where CoCoA goes wrong, let us know. If you can, try to find a ”small” example which triggers the problem. Send us all the commands we must execute to reproduce the bug, and the correct output you expected. Please, also tell us which version of CoCoA you are using (see “**VersionInfo**” ([I-22.3](#) pg.340)), and which operating system your computer uses (*i.e.* Linux, Mac or Microsoft Windows).

To give feedback or report a bug send email to the address “cocoa@dimma.unige.it” and we will reply soon. We also usually put information about bugs and new features into the ”redmine” issue-tracking system, at “<https://cocoa.dima.unige.it/redmine/>”.

See Also: Tutorial: manual([II-1.2](#) pg.349)

Chapter II-2

Introduction to CoCoA Programming

II-2.1 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-19.32 pg.310) command.

The most important construct in CoCoA programming is the user-defined function, created with “**define**” (I-4.4 pg.84). 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-2.2 pg.357).

II-2.2 All CoCoA commands

This is a complete list of all CoCoA commands:

break	break out of a loop command
ciao	quit CoCoA
continue	continue directly with next loop iteration
define	define a function
describe	information about an object
exit	quit CoCoA
for	loop command
foreach	loop command
if	conditional statement
print	print the value of an expression
print on	print to an output stream
println	print the value of an expression
protect	protect a variable from being overwritten
quit	quit CoCoA
repeat	loop command
return	exit from a function
source	read commands from a file or device
SourceRegion	read commands from a region in a file
try	try command sequence, catch any errors
unprotect	remove protection from a variable
use	command for making a ring active
while	loop command

Chapter II-3

Language Elements

II-3.1 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-3.2 Identifiers

There are two types of identifiers or names.

* Identifiers of ring indeterminates (see “NewPolyRing” (I-14.9 pg.226))

* Predefined or user-defined names (functions and CoCoALanguage variables).

See Also: Indeterminates(III-9.4 pg.422)

II-3.3 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-3.4 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

```

Writing multi-line comment inside “/*” and “*/” is **strongly** discouraged (even though allowed).

Chapter II-4

Operators

II-4.1 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.29)

II-4.2 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-4.22 pg.92) and “mod” (I-13.29 pg.215) 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-4.3 Relational Operators

See Also: Equality Operator(I-5.10 pg.98), Order Comparison Operators(I-15.10 pg.245), IsIn(I-9.59 pg.168)

II-4.4 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-18.27 pg.280)).

See Also: record(I-18.27 pg.280), List Constructors(III-5.2 pg.406)

II-4.5 Range Operator

If “ M ” and “ N ” are of type INT, then the expression: “ $M \dots N$ ” returns

* the list “ $[M, M+1, \dots, N]$ ” 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 “ $N \dots M$ ” 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: introduction([II-4.1](#) pg.[361](#)), List Constructors([III-5.2](#) pg.[406](#)), LIST([III-5](#) pg.[405](#))

Chapter II-5

Evaluation and Assignment

II-5.1 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-9.113 pg.186](#)).

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-9.113 pg.186](#)).

If an error occurs during the evaluation of an expression, then the evaluation is interrupted and the user is notified about the error.

II-5.2 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

```
/**/ M := 5; N := 8;  
/**/ T := M+N;  
/**/ println T;  
13  
/**/ T := T+1; -- evaluate RHS first, then assign value to T  
/**/ println T;
```

14

```
/**/ L := [1,2,3];  
/**/ L[2] := 0; -- assign to an entry in a list  
/**/ println L;  
[1, 0, 3]  
  
/**/ use QQ[x,y,z];  
/**/ P := record[F := x*z];  
/**/ P.Degree := deg(P.F); -- assign to a field in a record  
/**/ P;  
record[Degree := 2, F := x*z]
```

Chapter II-6

Flow Control: Conditional Statements and Loops

II-6.1 Commands and Functions for Branching

The following are the CoCoA commands for constructing conditional statements:

`if` conditional statement

II-6.2 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-7

Verbosity and interrupt

II-7.1 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.339) is higher than some value (see their specific manual for the values, anyway not less than 10). See also “`SetVerbosityLevel`” ([I-19.14](#) pg.303).

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-7.2 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>GBasisByHomog</code>	calculate a Groebner basis by homogenization
<code>gin</code>	generic initial ideal
<code>GroebnerFanIdeals</code>	all reduced Groebner bases of an ideal
<code>ImplicitHypersurface</code>	implicitization of hypersurface
<code>interreduce</code>	interreduce a list of polynomials
<code>interreduced</code>	interreduce a list of polynomials
<code>IsMaximal</code>	maximality test
<code>IsRadical</code>	check if an IDEAL is radical
<code>MinPolyQuot</code>	minimal polynomial in quotient ring
<code>PrimaryDecomposition</code>	primary decomposition of an ideal
<code>radical</code>	radical of an ideal
<code>rgin</code>	generic initial ideal wrt StdDegRevLex
<code>SAGBI, SAGBIHomog</code>	SAGBI bases for subalgebra
<code>SatSAGBI</code>	SAGBI bases for subalgebra
<code>SetVerbosityLevel</code>	set the verbosity level
<code>SubalgebraMinGens</code>	list of minimal generators as subalgebra
<code>UniversalGBasis</code>	universal Groebner basis of the input ideal
<code>VerbosityLevel</code>	verbosity level

II-7.3 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
<code>GBasisByHomog</code>	calculate a Groebner basis by homogenization
<code>ImplicitHypersurface</code>	implicitization of hypersurface
<code>MinPolyQuot</code>	minimal polynomial in quotient ring

Chapter II-8

Input/Output

II-8.1 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-19.32 pg.310) command—be executed as CoCoA commands.

II-8.2 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-16.40 pg.260) and “**println**” (I-16.45 pg.263) 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 “**GetLine**” (I-7.16 pg.127) and “**print on**” (I-16.41 pg.261) 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 “**GetLine**” (I-7.16 pg.127) 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-8.3 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.243) then prints to the file using “**print on**” (I-16.41 pg.261).

To receive verbatim input from a file, one first opens the file with “**OpenIFile**” (I-15.2 pg.241), then reads lines from the file with “**GetLine**” (I-7.16 pg.127).

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"
/**/ GetLine(D);
hello world
/**/ Close(D);

```

To read and execute a sequence of CoCoA commands from a text file, one uses the “**source**” (I-19.32 pg.310) 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\(I-1.49 pg.46\)](#), [close\(I-3.19 pg.63\)](#), [GetLine\(I-7.16 pg.127\)](#), [OpenIFile\(I-15.2 pg.241\)](#), [OpenOFile\(I-15.5 pg.243\)](#), [OpenLog\(I-15.4 pg.242\)](#), [CloseLog\(I-3.20 pg.64\)](#), [print on\(I-16.41 pg.261\)](#), [source\(I-19.32 pg.310\)](#)

II-8.4 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.243) to **open** the string, then “**print on**” (I-16.41 pg.261) to write to it. To read from a string, one may open the string for input with “**OpenIString**” (I-15.3 pg.242) then read substrings from it with “**GetLine**” (I-7.16 pg.127).

example

```

/**/ S := "hello world!\ngoodbye world!";
/**/ D := OpenIString(S); -- open the string S for input to CoCoA
/**/ line := GetLine(D); -- read 1st line from the string
/**/ line;
hello world!
/**/ Close(D); -- close device D

```

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-19.35 pg.311).

II-8.5 Commands and Functions for IO

The following are commands and functions for input/output:

block	group several commands into a single command
close	close an output stream
CloseLog	close a log of a CoCoA session
format	convert object to formatted string
GetLine	read a line of input from an in-stream
latex	LaTeX formatting
NewFreeModuleForSyz	create a new FreeModule with shifts for syz
NewLine [OBSOLESCE]	[OBSOLESCE] string containing a newline
OpenIFile	open input file
OpenIString	open input string
OpenLog	open a log of a CoCoA session
OpenOFile	open output file
OpenOString	open output string
OpenSocket	open a socket connection
print	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>SprintTrunc</code>	convert to a string and truncate
<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-9

CoCoA Packages

II-9.1 Introduction to Packages

User-defined functions may be saved in separate files and read into a CoCoA session using the “**source**” (I-19.32 pg.310) 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 easier than in CoCoA-4.

See Also: First Example of a Package(II-9.2 pg.375), define(I-4.4 pg.84), source(I-19.32 pg.310)

II-9.2 First Example of a Package

The following is an example of a package. It could be typed directly into a CoCoA session, but more normally it is saved into a file to be read into CoCoA using the “**source**” (I-19.32 pg.310) command. 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.cpkg5";
```

```

/**/ ToyTest(4); -- was exported
Not the number 1
-- /**/ IsNumberOne(4); --> !!! ERROR !!! as expected: fn was not exported

/**/ $contrib/toypackage.IsNumberOne(4);
false

```

II-9.3 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-9.4 Global Aliases

A global alias for a package is formed by using the command “`alias`” ([I-1.8 pg.33](#)) 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-1.8 pg.33](#)), `aliases`([I-1.9 pg.34](#))

II-9.5 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-9.6 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>package</code>	keyword marking content of a CoCoA package
<code>PackageOf</code>	package of an identifier
<code>packages</code>	list of loaded packages
<code>PkgName</code>	returns the name of a package

II-9.7 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-9.8 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-9.9 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-9.10 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-9.11 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-9.12 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-9.13 Geometrical Theorem-Proving [PROTOTYPE]

This is just a prototype: exact semantics, and interface are likely to change!

```

TITLE      : thmproving.cpkg
DESCRIPTION : CoCoA package for geometrical theorem-proving in euclidean space
AUTHOR     : L. Bazzotti, G. Dalzotto

ThmMan();

```

II-9.14 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-9.15 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-9.16 Control

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

```
TITLE      : control.cpkg
DESCRIPTION : CoCoA package for Geometric Control Theory
AUTHOR     : M. Anderlucci and M. Caboara
```

```
-- Enter
    $contrib/control.Man();
    to get a complete description of the package including a suggested alias.
```


Chapter II-10

Linked libraries

II-10.1 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-10.2 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-10.3 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 is still work in progress.

II-10.4 Frobbby

Frobbby - Computations With Monomial Ideals “<http://www.broune.com/frobbby>”

All functions starting with “Frb” are implemented in Frobbby.

II-10.5 MathSAT

MathSAT - Satisfiability modulo theories (SMT) solver “<http://mathsat.fbk.eu/>”

All functions starting with “MSAT” are implemented in MathSAT.

II-10.6 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-11

Migrating from CoCoA-4 and keeping up-to-date

II-11.1 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-1.14 pg.35), “sort” (I-19.28 pg.308)); CoCoA-5 wants these arguments to be identified with the new keyword “ref” (I-18.30 pg.281), and will issue a warning if you do not 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-3.21 pg.64).

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.329).

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-18.51 pg.290).

example

```
/*C5*/ I := ideal(x,y^2); NumIndets(RingOf(I));
```

The function “LinKer” (I-12.12 pg.197) has been replaced by “LinKerBasis” (I-12.13 pg.198), and there is a new function called “LinKer” (I-12.12 pg.197) which produces a matrix.

More generally, see also the CoCoA-4 “translation table” in the CoCoAManual directory or at the URL

<http://cocoa.dima.unige.it/cocoalib/doc/CoCoATranslationTable.html>

See Also: CoCoA-4 mode(I-3.21 pg.64), TopLevel(I-20.10 pg.329), CurrentRing(I-3.64 pg.80), RingOf(I-18.51 pg.290)

II-11.2 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-6.33 pg.118).

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-14.33 pg.235), func(I-6.33 pg.118)

II-11.3 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.

AllReducedGroebnerBases [OBSOLETE]	[OBSOLETE] Renamed to GroebnerFanIdeals
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
CompleteToOrd [OBSOLETE]	[OBSOLETE] renamed to MakeTermOrdMat
E_ [OBSOLETE]	[OBSOLETE] vectors of the canonical basis
Function [OBSOLETE]	[OBSOLETE]
functions [OBSOLETE]	[OBSOLETE] replaced by describe
Get [OBSOLETE]	[OBSOLETE] replaced by GetLine
ID [OBSOLETE]	[OBSOLETE] renamed to RingID
ILogBase [OBSOLETE]	[OBSOLETE] renamed to FloorLogBase
IsInSubalgebra [OBSOLETE]	[OBSOLETE] check if one polynomial is in a subalgebra
IsNumber [OBSOLETE]	[OBSOLETE] checks if the argument is a number
isqrt [OBSOLETE]	[OBSOLETE] renamed to FloorSqrt
LinKerModP [OBSOLETE]	[OBSOLETE] find the kernel of a matrix mod p
LinSol [OBSOLETE]	[OBSOLETE] renamed to LinSolve
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] Renamed to HilbertSeriesMultiDeg
PoincareShifts [OBSOLETE]	[OBSOLETE] Renamed to HilbertSeriesShifts
PrimaryDecomposition0 [OBSOLETE]	[OBSOLETE]
randomize [OBSOLETE]	[OBSOLETE] randomize the coefficients of a given polynomial
randomized [OBSOLETE]	[OBSOLETE] randomize the coefficients of a given polynomial
RefineGCDFreeBasis [OBSOLETE]	refine an integer GCD free basis
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 to RingsOf
seed [OBSOLETE]	[OBSOLETE] replaced by reseed
size [OBSOLETE]	[OBSOLETE]
SubalgebraMap [OBSOLETE]	[OBSOLETE] algebra homomorphism representing a subalgebra
TmpNBM [OBSOLETE]	[OBSOLETE] renamed to ApproxPointsNBM
Unset [OBSOLETE]	[OBSOLETE] set and unset panel options
valuation [OBSOLETE]	[OBSOLETE]
WLog [OBSOLETE]	[OBSOLETE]

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 [OBSCIENT]	[OBSCIENT] renamed AffHilbertFn
AffPoincare [OBSCIENT]	[OBSCIENT] Renamed to AffHilbertSeries
apply [OBSCIENT]	apply [OBSCIENT]
ArrDerMod [OBSCIENT]	[OBSCIENT] renamed to ArrDerModule
FactorAlgExt [OBSCIENT]	[OBSCIENT] factorization over algebraic extensions
hilbert [OBSCIENT]	[OBSCIENT] renamed to HilbertFn
HomogElimMat [OBSCIENT]	[OBSCIENT] renamed to ElimHomogMat
image [OBSCIENT]	[OBSCIENT] apply ring homomorphism
insert [OBSCIENT]	[OBSCIENT] insert an object in a list
log [OBSCIENT]	[OBSCIENT] renamed to exponents
MakeTermOrd [OBSCIENT]	[OBSCIENT] renamed to MakeTermOrdMat
MinGensGeneral [OBSCIENT]	[OBSCIENT] renamed to MinSubsetOfGens
minimalize [OBSCIENT]	[OBSCIENT]
minimalized [OBSCIENT]	[OBSCIENT] Renamed: IdealOfMinGens and SubmoduleOfMinGens
MultiArrDerMod [OBSCIENT]	[OBSCIENT] renamed to MultiArrDerModule
NewLine [OBSCIENT]	[OBSCIENT] string containing a newline
NewRingFp [OBSCIENT]	create a new finite field
poincare [OBSCIENT]	[OBSCIENT] Renamed to HilbertSeries
PreImage [OBSCIENT]	[OBSCIENT]
PrimaryPoincare [OBSCIENT]	[OBSCIENT] renamed to PrimaryHilbertSeries
rank [OBSCIENT]	[OBSCIENT] rank
ReadExpr [OBSCIENT]	[OBSCIENT] renamed RingElem
RMap [OBSCIENT]	[OBSCIENT] define ring homomorphism for function image
SmoothFactor [OBSCIENT]	[OBSCIENT] renamed: see FactorINT TrialDiv variant
SubalgebraRepr [OBSCIENT]	[OBSCIENT] representation of a polynomial as a subalgebra element
TensorMat [OBSCIENT]	[OBSCIENT] renamed to KroneckerProd
WeightsMatrix [OBSCIENT]	[OBSCIENT] 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.148) and “**while**” ([I-23.3](#) pg.342), etc., inside CoCoA programs.

The relational operators

`= <> < <= > >=`

return boolean constants (see “Relational Operators” ([II-4.3](#) pg.362)).

The boolean operators are “**and**” ([I-1.13](#) pg.35), “**or**” ([I-15.9](#) pg.244), “**IsIn**” ([I-9.59](#) pg.168). From version CoCoA-5.0.9 “**not**” ([I-14.33](#) pg.235) is a function (instead of an operator).

See Also: Relational Operators([II-4.3](#) pg.362), Commands and Functions for BOOL([III-1.2](#) pg.389), Commands and Functions returning BOOL([III-1.3](#) pg.389)

III-1.2 Commands and Functions for BOOL

<code>and</code>	boolean ”and” operator
<code>assert</code>	check an assertion
<code>Bool01</code>	Convert a boolean to an integer
<code>in</code>	list element selector in list constructor
<code>IsPalindromic</code>	test whether a univariate polynomial is palindromic
<code>IsPolyRing</code>	test whether a ring is a polynomial ring
<code>not</code>	boolean ”not” operator
<code>operators, shortcuts</code>	Special characters equivalent to commands
<code>or</code>	boolean ”or” operator

III-1.3 Commands and Functions returning BOOL

<code>and</code>	boolean ”and” operator
<code>AreGensMonomial</code>	checks if given gens are monomial
<code>AreGensSqFreeMonomial</code>	checks if given gens are squarefree monomial
<code>EqSet</code>	checks if the set of elements in two lists are equal

Equality Operator	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
IsArrCentral	checks if the arrangement is central
IsArrFree	checks if the arrangement is free
IsAtEOF	checks if input stream has reached end of input
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 two elements are coprime
IsDiagonal	checks if a matrix is diagonal
IsDivisible	checks if A is divisible by B
IsElem	checks if A is an element of B
IsEmpty	checks if a list is empty
IsEven, IsOdd	test whether an integer is even or odd
IsEvenPoly, IsOddPoly	test whether a polynomial is even or odd as a function
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	check if argument is an indeterminate
IsIndetPosPower	check if argument is a power of 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
IsLattice	checks if the poset is a lattice
IsLexSegment	checks if an ideal is lex-segment
IsLRSDegenerate	checks the given polynomial for LRS-degeneracy
IsLRSDegenerateOrder	checks the given polynomial for “n”-LRS-degeneracy
IsMaximal	maximality test
IsMinusOne	test whether an object is -1
IsMultiArrFree	checks if the multiarrangement is free
IsOne	test whether an object is one
IsPalindromic	test whether a univariate polynomial is palindromic
IsPolyRing	test whether a ring is a polynomial ring
IsPosetGraded	checks if the poset is graded
IsPositiveGrading	check if a matrix defines a positive grading
IsPowerOf2	check if an integer is a power of 2
IsPrimary	primary test
IsPrime	prime integer test
IsPrimitivePoly	test if polynomial over finite field is primitive
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
IsSigmaGoodPrime	check if INT is good prime for IDEAL
IsSqFree	check if an INT or RINGELEM is square-free
IsSquare	check if an INT is a square
IsStable	checks if an ideal is stable

<code>IsStdGraded</code>	checks if the grading is standard
<code>IsStronglyStable</code>	checks if an ideal is strongly stable
<code>IsSubset</code>	checks if the elements of one list are a subset of another
<code>IsSurjective</code>	check if a RINGHOM is surjective
<code>IsSymmetric</code>	checks if a matrix is symmetric
<code>IsTerm</code>	checks if the argument is a term
<code>IsTermOrdering</code>	check if a matrix defines a term-ordering
<code>IsTrueGCDDomain</code>	test whether a ring is a true GCD domain
<code>IsZero</code>	test whether an object is zero
<code>IsZeroCol, IsZeroRow</code>	test whether a column(row) is zero
<code>IsZeroDet</code>	test whether determinant is zero
<code>IsZeroDim</code>	test whether an ideal is zero-dimensional
<code>IsZeroDivisor</code>	test whether a RINGELEM is a zero-divisor
<code>IsZZ</code>	test whether a ring is the ring of integers
<code>not</code>	boolean "not" operator
<code>or</code>	boolean "or" operator
<code>Order Comparison Operators</code>	less than, greater than, ...

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 “NewZZmod” (I-14.15 pg.228)). 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.393), Commands and Functions returning INT(III-2.3 pg.396)

III-2.2 Commands and Functions for INT

abs	absolute value of a number
AdjacentMinors	list of adjacent minors of a matrix
AffHilbertFn	the affine Hilbert function
AllFactors	All positive factors of a (small) positive integer
ArrBoolean	boolean arrangement
ArrBraid	braid arrangement
ArrCatalanA	Catalan arrangement of type A
ArrCatalanB	Catalan arrangement of type B
ArrCatalanD	Catalan arrangement of type D
ArrShiA	Shi arrangement of type A
ArrShiB	Shi arrangement of type B
ArrShiCatalanA	Shi-Catalan arrangement of type A with multiplicities
ArrShiCatalanB	Shi-Catalan arrangement of type B with given multiplicities
ArrShiCatalanD	Shi-Catalan arrangement of type D with given multiplicities
ArrShiD	Shi arrangement of type D
ArrTypeB	reflection arrangement of type B
ArrTypeD	reflection arrangement of type D
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
CatalanNumber	Catalan number
ChebyshevPoly	Orthogonal Polynomials: Chebyshev, Hermite, Laguerre
ContFracToRat	convert continued fraction to rational
CoprimeFactor	determine factor of N coprime to given base
CoprimeFactorBasis	determine coprime factor base for a set of integers or ring elements
CRT	Chinese Remainder Theorem
CRTPoly	Chinese Remainder Theorem on polynomial coefficients
cyclotomic	n-th cyclotomic polynomial
DecimalStr	convert rational number to decimal string
den	denominator
DensePoly	the sum of all power-products of a given degree
DicksonPoly	Dickson polynomial
div	quotient for integers
ElimMat	matrix for elimination ordering
EulerTotient	Euler Totient function for positive integers
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
FactorINT	find prime factors of an integer
FactorMultiplicity	multiplicity of a factor of an integer
fibonacci	n-th fibonacci number
first	the first N elements of a list
FirstCols, FirstRows	submatrix of the first N cols or rows
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
FloorRoot	integer part of r-th root of an integer
FloorSqrt	(truncated) square root of an integer
fold	inset newlines into a long string
FoldToListInput	convert string so it looks like a list of short strings
format	convert object to formatted string
GBasis timeout	compute a Groebner basis with a timeout
gcd	greatest common divisor
GenericPoints	random projective points
GetCol	convert a column of a matrix into a list
GetRow	convert a row of a matrix into a list
graeffe	graeffe transformation (squares the roots)
HermitePoly	Orthogonal Polynomials: Chebyshev, Hermite, Laguerre
HilbertFn	the Hilbert function
HilbertMat	create a Hilbert matrix over QQ
HomogCompt	homogeneous part of given degree
IdentityMat	the identity matrix
incr, decr	increment/decrement a counter
indent, IndentStr	prints in a more readable way
indet	individual indeterminates
insert [OBSOLESCEMENT]	[OBSOLESCEMENT] insert an object in a list
InverseSystem	Inverse system of an ideal of derivations
InvTotient	Inverse totient function: preimages of Euler totient
IsCoprime	checks if two elements are coprime
IsDivisible	checks if A is divisible by B
IsEven, IsOdd	test whether an integer is even or odd
IsInteger	check if a RINGELEM is integer

IsLRSDegenerateOrder	checks the given polynomial for “n”-LRS-degeneracy
IsMinusOne	test whether an object is -1
IsOne	test whether an object is one
IsPowerOf2	check if an integer is a power of 2
IsPrime	prime integer test
IsProbPrime	checks if an integer is a probable prime
IsSigmaGoodPrime	check if INT is good prime for IDEAL
IsSqFree	check if an INT or RINGELEM is square-free
IsSquare	check if an INT is a square
IsZero	test whether an object is zero
IsZeroCol, IsZeroRow	test whether a column(row) is zero
KroneckerSymbol	Kronecker symbol of R mod M
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
MakeTermOrdMat	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
MoebiusFn	Moebius function mu on the whole numbers
NewFreeModule	create a new FreeModule
NewList	create a new list
NewMat	Zero matrix
NewMatFilled	matrix filled with value
NewPolyRing	create a new PolyRing
NewRingTwinFloat	create a new twin-float ring
NewZZmod	create a new finite ring (integers mod N)
NextPrime, NextProbPrime	find the next prime number
NmzSetVerbosityLevel	Set the verbosity state for Normaliz
num	numerator
NumPartitions	number of partitions of an integer
OpenSocket	open a socket connection
operators, shortcuts	Special characters equivalent to commands
partitions	partitions of an integer
PosetJoin	join between two elements of a poset from the relations of the poset
PosetMeet	meet between two elements of a poset from the relations of the poset
PosetNRank	rank of the node N from the relations of the graded poset
power	compute a power
PowerMod	compute a modular power efficiently
PrevPrime, PrevProbPrime	find the previous prime number
PrimitiveRoot	find a primitive root modulo a prime
primorial	primorial function
product	the product of the elements of a list
radical	radical of an ideal
random	random integer
RandomLinearForm	random linear form in polynomial ring
RandomNBitPrime	Random prime with N bits
RandomPermutation	random permutation (of indices)
RandomSmallPrime	Random prime between 5 and N
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	rational reconstruction from modular image
RatReconstructByLattice	rational reconstruction from modular image
RatReconstructPoly	Rational reconstruction of polynomial coefficients
RatReconstructWithBounds	deterministic rational reconstruction from modular image
RelNotes	print the release notes
remove	remove an object in a list
reseed	reseed the pseudo-random number generator
RevLexMat	matrix for rev lex term-ordering
RingElem	convert an expression into a RINGELEM
RingQQt	pre-defined polynomial rings
RootBound	bound on roots of a polynomial over QQ
SAGBI, SAGBIHomog	SAGBI bases for subalgebra
SatSAGBI	SAGBI bases for subalgebra
ScientificStr	convert integer/rational to a floating-point string
SectionalMatrix	sectional matrix
SetCol	set a list as a column into a matrix
SetEntry	set an entry 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
SleepFor	Make program sleep for a specified time
SmallestNonDivisor	find smallest prime which does not divide an integer
SmoothFactor [OBSOLESCE]	[OBSOLESCE] renamed: see FactorINT TrialDiv variant
SourceRegion	read commands from a region in a file
spaces	return a string of spaces
SprintTrunc	convert to a string and truncate
StarRoot	compute smallest root of an integer
StdDegLexMat	matrix for std deg lex term-ordering
StdDegRevLexMat	matrix for std deg rev lex term-ordering
submat	submatrix
subsets	returns all sublists of a list
substring	substring of a string
sum	the sum of the elements of a list
SwapCols	swap two columns in a matrix
SwapRows	swap two rows in a matrix
SwinnertonDyerPoly	compute Swinnerton-Dyer polynomial with given roots
SymbolRange	range of symbols for the indeterminates of a PolyRing
syz	syzygy module
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

AllFactors	All positive factors of a (small) positive integer
AsINT	convert into an INT
binomial	binomial coefficient
BinomialRepr, BinomialReprShift	binomial representation of integers
Bool01	Convert a boolean to an integer
CatalanNumber	Catalan number
ceil	round rational up to integer
characteristic	the characteristic of a ring
ContFrac	continued fraction quotients
CoprimeFactor	determine factor of N coprime to given base
count	count the objects in a list
CyclotomicFactorIndexes	find indexes of cyclotomic factors of a polynomial
CyclotomicIndex, CyclotomicTest	computes the index of a given cyclotomic polynomial
date	the date
deg	the standard degree of a polynomial or moduleelem
den	denominator
depth	Depth of a module
dim	the dimension of a (quotient) ring
div	quotient for integers
EulerTotient	Euler Totient function for positive 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
FloorRoot	integer part of r-th root of an integer
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
InvTotient	Inverse totient function: preimages of Euler totient
KroneckerSymbol	Kronecker symbol of R mod M
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
LRSDegeneracyOrder	the LRS degeneracy order of a polynomial
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
moebius	Moebius function of a poset
MoebiusFn	Moebius function mu on the whole numbers
multiplicity	the multiplicity (degree) of a ring
NextPrime, NextProbPrime	find the next prime number
NmzVerbosityLevel	Get the verbosity level for Normaliz
num	numerator
NumBChambers	number of bounded chambers of an arrangement of hyperplanes
NumChambers	number of chambers of an arrangement of hyperplanes
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
NumRealRoots	number of real roots of a polynomial
NumRows	number of rows in a matrix
NumTerms	number of terms in a polynomial
PosetNRank	rank of the node N from the relations of the graded poset
PosetRank	rank of the poset from its relations
power	compute a power
PowerMod	compute a modular power efficiently
PrevPrime, PrevProbPrime	find the previous prime number
PrimitiveRoot	find a primitive root modulo a prime
primorial	primorial function
radical	radical of an ideal
random	random integer
RandomNBitPrime	Random prime with N bits
RandomSmallPrime	Random prime between 5 and N
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
ScalarProduct	scalar product
sign	the sign of a number
SmallestNonDivisor	find smallest prime which does not divide an integer
StarRoot	compute smallest root of an integer
SystemCommand	run a system command
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.399](#)), Commands and Functions returning RAT([III-3.3 pg.400](#))

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

<code>NewMatFilled</code>	matrix filled with value
<code>num</code>	numerator
<code>power</code>	compute a power
<code>product</code>	the product of the elements of a list
<code>RealRootRefine</code>	refine a real root of a univariate polynomial
<code>RealRoots</code>	real roots of a univariate polynomial
<code>RealRootsApprox</code>	approximations to the real roots of a univariate poly
<code>RingElem</code>	convert an expression into a RINGELEM
<code>round</code>	round to integer
<code>ScientificStr</code>	convert integer/rational to a floating-point string
<code>SetEntry</code>	set an entry into a matrix
<code>sign</code>	the sign of a number
<code>SimplestBinaryRatBetween</code>	find simplest binary rational in a closed interval
<code>SimplestRatBetween</code>	find simplest rational in a closed interval
<code>SleepFor</code>	Make program sleep for a specified time
<code>StableBBasis5</code>	Stable Border Basis of ideal of points
<code>sum</code>	the sum of the elements of a list
<code>TimeFrom</code>	time elapsed since a given moment

III-3.3 Commands and Functions returning RAT

<code>abs</code>	absolute value of a number
<code>AsRAT</code>	convert into a RAT
<code>CFApprox</code>	continued fraction approximation
<code>CFApproximants</code>	continued fraction approximants
<code>ContFracToRat</code>	convert continued fraction to rational
<code>CpuTime</code>	Counts cpu time
<code>ElapsedTime</code>	Counts elapsed time
<code>FloatApprox</code>	approx. of rational number of the form $M * 2^E$
<code>power</code>	compute a power
<code>RootBound</code>	bound on roots of a polynomial over QQ
<code>ScalarProduct</code>	scalar product
<code>SimplestBinaryRatBetween</code>	find simplest binary rational in a closed interval
<code>SimplestRatBetween</code>	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: for backward compatibility CoCoA still accepts an **obsolescent** syntax for string literals (between single-quotes); do not use this!

See Also: String Operations(III-4.2 pg.401), sprint(I-19.35 pg.311), Commands and Functions for STRING(III-4.3 pg.402), Commands and Functions returning STRING(III-4.4 pg.403)

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-9.59](#) pg.168) 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.401), ascii([I-1.49](#) pg.46), ConcatStrings([I-3.47](#) pg.74), IsIn([I-9.59](#) pg.168), len([I-12.7](#) pg.195), substring([I-19.60](#) pg.321)

III-4.3 Commands and Functions for STRING

ascii	convert between characters and ascii code
ConcatStrings	concatenate strings
error	throw an error message
fold	inset newlines into a long string
FoldToListInput	convert string so it looks like a list of short strings
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
NewPolyRingWeights	create a new PolyRing with weights
NewQuotientRing	create a new quotient ring
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
package	keyword marking content of a CoCoA package
PackageOf	package of an identifier
PlayCantStop	First game in CoCoA
PlotPointsOn	outputs the coordinates of the points to a file
PolyAlgebraHom	homomorphism of polynomial algebras
PolyRingHom	homomorphism of polynomial rings
protect	protect a variable from being overwritten
RingElem	convert an expression into a RINGELEM
RingElemList, RingElems	convert expressions into a LIST of 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
substring	substring of a string
SymbolRange	range of symbols for the indeterminates of a PolyRing
SystemCommand	run a system command
tagged	tag an object for pretty printing

III-4.4 Commands and Functions returning *STRING*

<code>ascii</code>	convert between characters and ascii code
<code>CocoaPackagePath</code>	returns the path to the CoCoA packages
<code>ConcatStrings</code>	concatenate strings
<code>DecimalStr</code>	convert rational number to decimal string
<code>ExternalLibs</code>	Linked external libraries
<code>FloatStr</code>	convert rational number to a decimal string
<code>fold</code>	inset newlines into a long string
<code>FoldToListInput</code>	convert string so it looks like a list of short strings
<code>format</code>	convert object to formatted string
<code>GetEnv</code>	access shell variables
<code>GetErrMsg</code>	returns the message associated with an error
<code>GetLine</code>	read a line of input from an in-stream
<code>indent, IndentStr</code>	prints in a more readable way
<code>IndetName</code>	the name of an indeterminate
<code>latex</code>	LaTeX formatting
<code>operators, shortcuts</code>	Special characters equivalent to commands
<code>PackageOf</code>	package of an identifier
<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>SprintTrunc</code>	convert to a string and truncate
<code>substring</code>	substring of a string
<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.406](#)). 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_1, N_2, ..., N_s]” as shorthand for “L[N_1][N_2] ... [N_s]” (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\(III-12.7 pg.195\)](#), [List Constructors\(III-5.2 pg.406\)](#), [Commands and Functions for LIST\(III-5.3 pg.406\)](#), [Commands and Functions returning LIST\(III-5.4 pg.410\)](#)

III-5.2 List Constructors

These operators create new lists.

```
A..B
[A,B,C,...]
[X in L: LIST | Condn: BOOL]: LIST
[E:expression | X in L]: LIST
[E:expression | X in L: LIST and Condn: BOOL]: LIST
```

“A..B” creates the list of (consecutive) integers from “A” to “B”; both ends are included.

“[A,B,C,...]” makes a list containing the values “A”, “B”, “C” and so on, in that order. In particular, “[]” creates the empty list.

“[X in L | Condn]” makes a list of those elements in “L” for which condition “Condn” is true; the entries in “L” are considered in order.

“[E | X in L]” evaluates the expression “E” for each “X” in “L”, and collects the results in a new list; the entries in “L” are considered in order.

“[E | X in L and Condn]” evaluates the expression “E” for each “X” in “L” which satisfies the condition “Condn”, and collects the results in a new list; the entries in “L” are considered in order.

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-14.6 pg.225](#)), append([I-1.14 pg.35](#)), concat([I-3.41 pg.71](#)), Range Operator([II-4.5 pg.362](#)), CartesianProduct, CartesianProductList([I-3.6 pg.59](#))

III-5.3 Commands and Functions for LIST

append	append an object to a list
ApproxSolve	Approximate real solutions for polynomial system
ArrBettiNumbers	Betti numbers of an arrangement of hyperplanes
ArrCharPoly	characteristic polynomial of an arrangement of hyperplanes
ArrCone	cone of an arrangement of hyperplanes
ArrDeletion	deletes a hyperplane from a list of hyperplanes
ArrFlats	list of flats of an arrangement of hyperplanes
ArrGraphical	graphical arrangement
ArrLattice	lattice of an arrangement of hyperplanes
ArrPoincarePoly	Poincare polynomial of an arrangement of hyperplanes
ArrRestriction	arrangement of hyperplanes A restricted to a hyperplane
ArrShiCatalanA	Shi-Catalan arrangement of type A with multiplicities
ArrShiCatalanB	Shi-Catalan arrangement of type B with given multiplicities
ArrShiCatalanD	Shi-Catalan arrangement of type D with given multiplicities
ArrSignedGraphical	signed graphical arrangement

ArrToMultiArr	multiarrangement from an arrangement and a list of multiplicities
ArrTuttePoly	Tutte polynomial of an arrangement of hyperplanes
ArtinianOrlikTeraoIdeal	Artinian Orlik-Terao ideal of an arrangement of hyperplanes
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
CommonDenom	Common denominator of a polynomial with rational coefficients
concat	concatenate lists
ConcatHorList	create a simple block matrix
ConcatLists	concatenate a list of lists
ConcatStrings	concatenate strings
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
CoprimeFactorBasis	determine coprime factor base for a set of integers or ring elements
count	count the objects in a list
covers	a poset description from the list of the strict relations
DiagMat	matrix with given diagonal
diff	returns the difference between two lists
distrib	the distribution of objects in a list
DivAlg	division algorithm
elim	eliminate variables
ElimHomogMat	matrix for elimination ordering
ElimMat	matrix for elimination ordering
EqSet	checks if the set of elements in two lists are equal
eval	substitute numbers or polynomials for indeterminates
EvalQuasiPoly	Evaluate a quasi-polynomial at an integer
Ext	presentation Ext modules as quotients of free modules
FGLM5	perform a FGLM Groebner Basis conversion
first	the first N elements of a list
flatten	flatten a list
foreach	loop command
FrobeniusMat	matrix of the Frobenius Map
FVector	f-vector of a top simplices list
GBM	intersection of ideals for zero-dimensional schemes
gcd	greatest common divisor
HGBM	intersection of ideals for zero-dimensional schemes
HilbertSeriesShifts	the Hilbert-Poincare series
homog	homogenize wrt an indeterminate
ideal	ideal generated by list
IdealAndSeparatorsOfPoints	ideal and separators for affine points
IdealAndSeparatorsOfProjectivePoints	ideal and separators for points
implicit	implicitization
ImplicitHypersurface	implicitization of hypersurface
ImplicitPlot	outputs the zero locus of a bivariate polynomial to a file
ImplicitPlotOn	outputs the zero locus of a bivariate polynomial to a file
in	list element selector in list constructor
IndetsProd	(product of) indeterminates actually in a polynomial
InitialIdeal	Initial ideal
insert [OBSOLESCE]	[OBSOLESCE] insert an object in a list
Interpolate	interpolating polynomial
interreduce	interreduce a list of polynomials
interreduced	interreduce a list of polynomials

<code>intersection</code>	intersect lists, ideals, or modules
<code>IntersectionList</code>	intersect lists, ideals, or modules
<code>IsArrCentral</code>	checks if the arrangement is central
<code>IsEmpty</code>	checks if a list is empty
<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>IsLattice</code>	checks if the poset is a lattice
<code>IsMultiArrFree</code>	checks if the multiarrangement is free
<code>IsPosetGraded</code>	checks if the poset is graded
<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>JacobianMat</code>	the Jacobian matrix 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>MaxChains</code>	maximal chains of the poset (from its relations)
<code>min</code>	a minimum element of a sequence or list
<code>MinBy</code>	a minimum element of a list
<code>MinGBoverZZ [PROTOTYPE]</code>	[PROTOTYPE] minimal Groebner basis of polys over ZZ
<code>ModuleElem</code>	create a module element
<code>moebius</code>	Moebius function of a poset
<code>MultiArrDerModule</code>	set of generators of the module of logarithmic derivations of a multiarrangement
<code>MultiArrExponents</code>	exponents of a free multiarrangement of hyperplanes
<code>MultiArrRestrictionZiegler</code>	Ziegler multirestriction of the arrangement of hyperplanes wrt a hyperplane
<code>MultiArrToArr</code>	underlying arrangement from a multiarrangement
<code>MultiplicationMat</code>	multiplication matrix of a ringelem
<code>NewFreeModuleForSyz</code>	create a new FreeModule with shifts for syz
<code>NewPolyRing</code>	create a new PolyRing
<code>NewPolyRingWeights</code>	create a new PolyRing with weights
<code>NmzComputation</code>	flexible access to Normaliz
<code>NmzEhrhartRing</code>	Ehrhart ring
<code>NmzIntClosureMonIdeal</code>	integral closure of a monomial ideal
<code>NmzIntClosureToricRing</code>	integral closure of a toric ring
<code>NmzNormalToricRing</code>	normalization of a toric ring
<code>NonZero</code>	remove zeroes from a list
<code>NR</code>	normal reduction
<code>NumBChambers</code>	number of bounded chambers of an arrangement of hyperplanes
<code>NumChambers</code>	number of chambers of an arrangement of hyperplanes
<code>operators, shortcuts</code>	Special characters equivalent to commands
<code>OrlikTeraoIdeal</code>	Orlik-Terao ideal of an arrangement of hyperplanes
<code>permutations</code>	returns all permutations of the entries of a list
<code>PlotPoints</code>	outputs the coordinates of the points to a file
<code>PlotPointsOn</code>	outputs the coordinates of the points to a file
<code>PolyAlgebraHom</code>	homomorphism of polynomial algebras
<code>PolyRingHom</code>	homomorphism of polynomial rings
<code>PosetCharPoly</code>	characteristic polynomial of a poset from the relations of the poset
<code>PosetDual</code>	dual of a poset from the relations of the poset
<code>PosetJoin</code>	join between two elements of a poset from the relations of the poset

PosetMeet	meet between two elements of a poset from the relations of the poset
PosetNRank	rank of the node N from the relations of the graded poset
PosetPoincarePoly	Poincare polynomial of a poset from the relations of the poset
PosetRank	rank of the poset from its relations
PrintBettiDiagram	print 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
RationalSolve	Rational solutions for 0-dim polynomial system
RationalSolveHomog	Rational solutions for 0-dim polynomial system
RatReconstructWithBounds	deterministic rational reconstruction from modular image
ReloadMan	Reload CoCoAManual/CoCoAHelp.xml
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
SAGBI, SAGBIHomog	SAGBI bases for subalgebra
SatSAGBI	SAGBI bases for subalgebra
ScalarProduct	scalar product
SeparatorsOfPoints	separators for affine points
SeparatorsOfProjectivePoints	separators for projective points
SetCol	set a list as a column 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	simplicial homology of a top simplices list
SolomonTeraoIdeal	Solomon-Terao ideal of an arrangement of hyperplanes wrt a poly
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
SubalgebraMinGens	list of minimal generators as subalgebra
submat	submatrix
submodule	submodule generated by list
subsets	returns all sublists of a list
sum	the sum of the elements of a list
SwinnertonDyerPoly	compute Swinnerton-Dyer polynomial with given roots
SymbolRange	range of symbols for the indeterminates of a PolyRing
syz	syzygy module
tail	remove the first element of a list
toric	saturate toric ideals
tuples	N-tuples
TVecFromHF	Type vector from Hilbert Function
TVecPoints	points associated to type vector
TVecPrintRes	resolution associated to type vector
TVecToHF	Hilbert Function of a type vector
WithoutNth	removes the N-th component from a list
ZPQ	change field for polynomials and ideals

III-5.4 Commands and Functions returning LIST

AdjacentMinors	list of adjacent minors of a matrix
ArrBettiNumbers	Betti numbers of an arrangement of hyperplanes
ArrBoolean	boolean arrangement
ArrBraid	braid arrangement
ArrCatalanA	Catalan arrangement of type A
ArrCatalanB	Catalan arrangement of type B
ArrCatalanD	Catalan arrangement of type D
ArrCone	cone of an arrangement of hyperplanes
ArrDeletion	deletes a hyperplane from a list of hyperplanes
ArrExponents	exponents of a free arrangement of hyperplanes
ArrFlats	list of flats of an arrangement of hyperplanes
ArrGraphical	graphical arrangement
ArrLattice	lattice of an arrangement of hyperplanes
ArrRestriction	arrangement of hyperplanes A restricted to a hyperplane
ArrShiA	Shi arrangement of type A
ArrShiB	Shi arrangement of type B
ArrShiCatalanA	Shi-Catalan arrangement of type A with multiplicities
ArrShiCatalanB	Shi-Catalan arrangement of type B with given multiplicities
ArrShiCatalanD	Shi-Catalan arrangement of type D with given multiplicities
ArrShiD	Shi arrangement of type D
ArrSignedGraphical	signed graphical arrangement
ArrToMultiArr	multiarrangement from an arrangement and a list of multiplicities
ArrTypeB	reflection arrangement of type B
ArrTypeD	reflection arrangement of type D
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
CanonicalBasis	canonical basis of a free module
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 an indet
CoeffListWRTSupport	list of coefficients of a polynomial wrt a power-product basis
compts	list of components of a ModuleElem
concat	concatenate lists
ConcatLists	concatenate a list of lists
ContFrac	continued fraction quotients
CoprimeFactorBasis	determine coprime factor base for a set of integers or ring elements
covers	a poset description from the list of the strict relations
CurrentTypes	lists all data types
CyclotomicFactorIndexes	find indexes of cyclotomic factors of a polynomial
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

<code>FrbAssociatedPrimes</code>	Associated primes of monomial ideals
<code>FrbIrreducibleDecomposition</code>	Irreducible decomposition of monomial ideals
<code>FrbMaximalStandardMonomials</code>	Maximal standard monomials of monomial ideals
<code>FrbPrimaryDecomposition</code>	Primary decomposition of monomial ideals
<code>GBasis</code>	calculate a Groebner basis
<code>GBasis timeout</code>	compute a Groebner basis with a timeout
<code>GBasisByHomog</code>	calculate a Groebner basis by homogenization
<code>GenericPoints</code>	random projective points
<code>GenRepr</code>	representation in terms of generators
<code>gens</code>	list of generators of an ideal
<code>GensJacobian</code>	set of generators of the Jacobian ideal of a polynomial
<code>GetCol</code>	convert a column of a matrix into a list
<code>GetCols</code>	convert a matrix into a list of lists
<code>GetRow</code>	convert a row of a matrix into a list
<code>GetRows</code>	convert a matrix into a list of lists
<code>GraverBasis</code>	Graver basis
<code>GroebnerFanIdeals</code>	all reduced Groebner bases of an ideal
<code>GroebnerFanReducedGBases</code>	Groebner fan reduced GBases
<code>HilbertBasisKer</code>	Hilbert basis for a monoid
<code>homog</code>	homogenize wrt an indeterminate
<code>HVector</code>	the h-vector of a module or quotient object
<code>in</code>	list element selector in list constructor
<code>indets</code>	list of indeterminates in a PolyRing
<code>IndetSubscripts</code>	the indices in the name of an indeterminate
<code>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>InvTotient</code>	Inverse totient function: preimages of Euler totient
<code>JanetBasis</code>	the Janet basis of an ideal
<code>LinKerBasis</code>	find the kernel of a matrix
<code>LRSDegeneracyOrder</code>	the LRS degeneracy order of a polynomial
<code>MakeSet</code>	remove duplicates from a list
<code>MaxChains</code>	maximal chains of the poset (from its relations)
<code>MinGBoverZZ [PROTOTYPE]</code>	[PROTOTYPE] minimal Groebner basis of polys over ZZ
<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>moebius</code>	Moebius function of a poset
<code>monomials</code>	the list of monomials of a polynomial
<code>MultiArrExponents</code>	exponents of a free multiarrangement of hyperplanes
<code>MultiArrRestrictionZiegler</code>	Ziegler multirestriction of the arrangement of hyperplanes wrt a hyperplane
<code>MultiArrToArr</code>	underling arrangement from a multiarrangement
<code>NewList</code>	create a new list
<code>NmzDiagInvariants</code>	ring of invariants of a diagonalizable group action
<code>NmzEhrhartRing</code>	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>operators, shortcuts</code>	Special characters equivalent to commands
<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

PosetDual	dual of a poset from the relations of the poset
PosetJoin	join between two elements of a poset from the relations of the poset
PosetMeet	meet between two elements of a poset from the relations of the poset
PrimaryDecomposition	primary decomposition of an ideal
PrimaryDecompositionGTZ0	primary decomposition of a 0-dimensional ideal
QuotientBasis	vector space basis for zero-dimensional quotient rings
QuotientBasisSorted	vector space basis for zero-dimensional quotient rings
QZP	change field for polynomials and ideals
RandomPermutation	random permutation (of indices)
RandomSubset	random subset
RandomSubsetIndices	indices for random subset
RandomTuple	random tuple
RandomTupleIndices	indices for random tuples
RealRoots	real roots of a univariate polynomial
RealRootsApprox	approximations to the real roots of a univariate poly
ReducedGBasis	reduced Groebner basis
res	free resolution
reverse, reversed	reverse a list
RingElemList, RingElems	convert expressions into a LIST of RINGELEM
RingsOf	list of the rings of an object
SAGBI, SAGBIHomog	SAGBI bases for subalgebra
SatSAGBI	SAGBI bases for subalgebra
SeparatorsOfPoints	separators for affine points
SeparatorsOfProjectivePoints	separators for projective points
shape	extended list of types involved in an expression
sorted	sort a list
SortedBy	sort a list
starting	list functions starting with a given string
StdBasis	Standard basis
SturmSeq	Sturm sequence of a univariate polynomial
SubalgebraMinGens	list of minimal generators as subalgebra
subsets	returns all sublists of a list
support	the list of terms of a polynomial or moduleelem
SymbolRange	range of symbols for the indeterminates of a PolyRing
SymmetricPolys	list of symmetric polynomials
tail	remove the first element of a list
TopLevelFunctions	returns the functions available at top-level
tuples	N-tuples
TVecFromHF	Type vector from Hilbert Function
UniversalGBasis	universal Groebner basis of the input ideal
wdeg	multi-degree of a polynomial
WithoutNth	removes the N-th component from a list
ZPQ	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.414), Commands and Functions returning RECORD(III-6.3 pg.414)

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
operators, shortcuts	Special characters equivalent to commands
PrintBettiDiagram	print 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
ApproxSolve	Approximate real solutions for polynomial system
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
CyclotomicFactorIndexes	find indexes of cyclotomic factors of a polynomial
DivAlg	division algorithm
eigenvectors	eigenvalues and eigenvectors of a matrix
factor	factor a polynomial
FactorINT	find prime factors of an integer
FVector	f-vector of a top simplices list
HadamardBoundSq	Hadamard bound for determinant
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
OpenSocket	open a socket connection
preimage0	preimage of a RINGELEM
PreprocessPts	Reduce redundancy in a set of approximate points
RationalSolve	Rational solutions for 0-dim polynomial system
RationalSolveHomog	Rational solutions for 0-dim polynomial system
RatReconstructByContFrac	rational reconstruction from modular image
RatReconstructByLattice	rational reconstruction from modular image
RatReconstructWithBounds	deterministic rational reconstruction from modular image
RealRootRefine	refine a real root of a univariate polynomial

<code>record</code>	create a record
<code>shape</code>	extended list of types involved in an expression
<code>SimplexInfo</code>	Stanley-Reisner ideal, AlexanderDual complex, ideal of top simplices
<code>SimplicialHomology</code>	simplicial homology of a top simplices list
<code>SmoothFactor</code> [OBSOLESCENT]	[OBSOLESCENT] renamed: see FactorINT TrialDiv variant
<code>SqFreeFactor</code>	compute a squarefree factorization
<code>StableBBasis5</code>	Stable Border Basis of ideal of points
<code>starting</code>	list functions starting with a given string
<code>SymbolRange</code>	range of symbols for the indeterminates of a PolyRing
<code>ThmProve</code> [PROTOTYPE]	[PROTOTYPE] ThmProve
<code>VersionInfo</code>	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-19.32 pg.310). 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.84) (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 inside a function

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 single **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.421), Commands and Functions for RING(III-9.8 pg.424), Commands and Functions returning RING(III-9.9 pg.426)

III-9.2 Polynomial Rings

CoCoA starts with the default (polynomial) ring “ $R ::= \mathbb{Q}\mathbb{Q}[x,y,z]$ ”. Polynomial rings are created with the function “**NewPolyRing**” (I-14.9 pg.226), but there is a special simplified syntax working in most cases: it must be preceded by the command “**use**” (I-21.6 pg.336) or by the symbol “ $::=$ ” (or both)

```
R ::= C[X:INDETS];          use C[X:INDETS];
R ::= C[X:INDETS], Ord;      use C[X:INDETS], Ord;
```

“**R**” is the identifier of a CoCoALanguage variable, “**C**” is a RING, “**X**” is an expression that defines the indeterminates, “**Ord**” 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 chosen as the current ring with the command “**use**” (I-21.6 pg.336).

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 current ring is S
/**/ CurrentRing;
RingWithID(23, "RingWithID(22)[x,y]")

```

See Also: [NewPolyRing\(I-14.9 pg.226\)](#), [CurrentRing\(I-3.64 pg.80\)](#)

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.346](#)) and “QQ” ([I-17.1 pg.267](#)) are not directly visible. Use “RingZZ()” or “RingQQ()” instead (or import them with “TopLevel” ([I-20.10 pg.329](#))).

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, "i^2+1");
/**/ U ::= K[u,v];     U;

```

See Also: [CoeffRing\(I-3.32 pg.68\)](#)

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-14.9 pg.226](#)); this is useful for specifying an **elimination order** via a matrix created by “ElimMat” ([I-5.7 pg.97](#)).

The predefined term-orderings are:

- * standard-degree reverse lexicographic: “**DegRevLex**” (default)
- * standard-degree lexicographic: “**DegLex**”
- * pure lexicographic: “**Lex**” (no grading)
- * pure xel: “**Xel**” (no grading)

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-14.9 pg.226), OrdMat(I-15.11 pg.245), elim(I-5.5 pg.96)

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\(I-14.11 pg.227\)](#)

III-9.8 Commands and Functions for RING

AffHilbertFn	the affine Hilbert function
AffHilbertSeries	the affine Hilbert-Poincare series
ApproxPointsNBM	Numerical Border Basis of ideal of points
ArrBoolean	boolean arrangement
ArrBraid	braid arrangement
ArrCatalanA	Catalan arrangement of type A
ArrCatalanB	Catalan arrangement of type B
ArrCatalanD	Catalan arrangement of type D
ArrGraphical	graphical arrangement
ArrShiA	Shi arrangement of type A
ArrShiB	Shi arrangement of type B
ArrShiCatalanA	Shi-Catalan arrangement of type A with multiplicities
ArrShiCatalanB	Shi-Catalan arrangement of type B with given multiplicities
ArrShiCatalanD	Shi-Catalan arrangement of type D with given multiplicities
ArrShiD	Shi arrangement of type D
ArrSignedGraphical	signed graphical arrangement
ArrTypeB	reflection arrangement of type B
ArrTypeD	reflection arrangement of type D
BaseRing	the base ring of a ring
BettiDiagram	the diagram of the graded Betti numbers
BringIn	bring in objects from another ring
CanonicalHom	canonical homomorphism
ChainCanonicalHom	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 (quotient) ring

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
NewFractionField	create a new fraction field
NewFreeModule	create a new FreeModule
NewMat	Zero matrix
NewPolyRing	create a new PolyRing
NewPolyRingWeights	create a new PolyRing with weights
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
PolyAlgebraHom	homomorphism of polynomial algebras
PolyRingHom	homomorphism of polynomial rings
PrintBettiDiagram	print 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
RandomLinearForm	random linear form in polynomial ring
RandomSparseNonSing01Mat	random sparse non-singular (0,1) matrix
RandomUnimodularMat	random unimodular matrix
reg	Castelnuovo-Mumford regularity of a module
RegularityIndex	regularity index of a Hilbert function or series
RingElem	convert an expression into a RINGELEM
RingElemList, RingElems	convert expressions into a LIST of RINGELEM

RingID	identification for ring
RowMat	single row matrix
SectionalMatrix	sectional matrix
SymmetricPolys	list of symmetric polynomials
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
CurrentRing	the current ring
domain	domain of a homomorphism
NewFractionField	create a new fraction field
NewPolyRing	create a new PolyRing
NewPolyRingWeights	create a new PolyRing with weights
NewQuotientRing	create a new quotient ring
NewRingTwinFloat	create a new twin-float ring
NewWeylAlgebra	create a new Weyl Algebra
NewZZmod	create a new finite ring (integers mod N)
operators, shortcuts	Special characters equivalent to commands
QQ	the ring of rationals
RingOf	the ring of the object
RingQQ	the ring of rationals
RingQQt	pre-defined polynomial rings
RingsOf	list of the rings of an object
RingZZ	the ring of integers
ZZ	the ring of integers

Chapter III-10

RINGHOM

III-10.1 Introduction to RINGHOM

A value of type INT or RAT can be automatically mapped into any ring. The way to **move** a RINGELEM value from one ring to another is to apply a ring homomorphism; the homomorphism can also transform the value as it is mapped. **Think mathematically!** ;-)

Common ways to create a ring homomorphism include “CanonicalHom” (I-3.4 pg.58), “PolyRingHom” (I-16.18 pg.252) and “PolyAlgebraHom” (I-16.17 pg.251).

The syntax mimics that of a function call (as in mathematical formulas). Given a RINGHOM “phi” just type “phi(x)” if “x” is a RINGELEM, “phi(x)” if “x” is a LIST or MAT; applies “phi” to each entry. An ideal cannot be mapped directly, you must instead map the generators individually: *e.g.* “ideal(phi(gens(I)))”.

There are also a few handy shortcuts which automatically determine and apply a homomorphism. The functions “matrix” (I-13.11 pg.209) and “RingElem” (I-18.47 pg.288) will 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.55) (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.121), “LT” (I-12.24 pg.202), “wdeg” (I-23.1 pg.341),...

See Also: Commands and Functions for RINGHOM(III-10.3 pg.428), Commands and Functions returning RINGHOM(III-10.4 pg.428), CanonicalHom(I-3.4 pg.58), PolyAlgebraHom(I-16.17 pg.251), PolyRingHom(I-16.18 pg.252), matrix(I-13.11 pg.209), RingElem(I-18.47 pg.288), BringIn(I-2.13 pg.55)

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-3.4 pg.58), InducedHom(I-9.28 pg.157)

III-10.3 Commands and Functions for RINGHOM

<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>ChainCanonicalHom</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

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-16.17 pg.251). However there are some handy shortcuts: “subst” (I-19.59 pg.320) and “eval” (I-5.14 pg.100).

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-16.17 pg.251), eval(I-5.14 pg.100), subst(I-19.59 pg.320)

III-11.3 Commands and Functions for RINGELEM

abs	absolute value of a number
ApproxSolve	Approximate real solutions for polynomial system
ArrCone	cone of an arrangement of hyperplanes
ArrDeletion	deletes a hyperplane from a list of hyperplanes
ArrDerModule	set of generators of the module of logarithmic derivations of an arrangement of hy
ArrExponents	exponents of a free arrangement of hyperplanes
ArrRestriction	arrangement of hyperplanes A restricted to a hyperplane
AsINT	convert into an INT
AsRAT	convert into a RAT
binomial	binomial coefficient
CanonicalRepr	representative of a class in a quotient ring
CharPoly	characteristic polynomial of a matrix
ChebyshevPoly	Orthogonal Polynomials: Chebyshev, Hermite, Laguerre
ClearDenom	clear common denominator of a polynomial with rational coeffs
CoeffHeight	the maximum of the absolute values of the coefficients of a polynomial
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 an indet
CoeffListWRTSupport	list of coefficients of a polynomial wrt a power-product basis
CoeffOfTerm	coefficient of a term of a polynomial
CommonDenom	Common denominator of a polynomial with rational coefficients
ComputeElimFirst	ComputeElimFirst
ConstantCoeff	constant coefficient of a polynomial
content	content of a polynomial
ContentFreeFactor	factorization of multivariate polynomial into content-free factors
ContentWRT	content of a polynomial wrt and indet or a list of indets
CoprimeFactorBasis	determine coprime factor base for a set of integers or ring elements
CRTPoly	Chinese Remainder Theorem on polynomial coefficients
cyclotomic	n-th cyclotomic polynomial
CyclotomicFactorIndexes	find indexes of cyclotomic factors of a polynomial
CyclotomicIndex, CyclotomicTest	computes the index of a given cyclotomic polynomial
DecimalStr	convert rational number to decimal string
deg	the standard degree of a polynomial or moduleelem
den	denominator
deriv	the derivative of a polynomial or rational function
DerivationAction	Action of a derivation
DF	the degree form of a polynomial

DicksonPoly	Dickson polynomial
discriminant	the discriminant of a polynomial
DivAlg	division algorithm
eigenfactors	eigenfactors of a matrix
eigenvectors	eigenvalues and eigenvectors of a matrix
elim	eliminate variables
eval	substitute numbers or polynomials for indeterminates
EvalQuasiPoly	Evaluate a quasi-polynomial at an integer
exponents	the list of exponents of the leading term of a polynomial
factor	factor a polynomial
FixedDivisor	compute (integer) fixed divisor for polynomial
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
GensJacobian	set of generators of the Jacobian ideal of a polynomial
GinJacobian	generic initial ideal of the Jacobian ideal of a polynomial
graeffe	graeffe transformation (squares the roots)
HermitePoly	Orthogonal Polynomials: Chebyshev, Hermite, Laguerre
homog	homogenize wrt an indeterminate
HomogCompt	homogeneous part of given degree
ideal	ideal generated by list
IndetIndex	index of an indeterminate
IndetName	the name of an indeterminate
IndetsProd	(product of) indeterminates actually in a polynomial
IndetSubscripts	the indices in the name of an indeterminate
interreduce	interreduce a list of polynomials
interreduced	interreduce a list of polynomials
IsArrFree	checks if the arrangement is free
IsConstant	checks if a ringelem is in the coefficient ring
IsCoprime	checks if two elements are coprime
IsDivisible	checks if A is divisible by B
IsElem	checks if A is an element of B
IsEvenPoly, IsOddPoly	test whether a polynomial is even or odd as a function
IsHomog	test whether given polynomials are homogeneous
IsIn	check if one object is contained in another
IsIndet	check if argument is an indeterminate
IsIndetPosPower	check if argument is a power of an indeterminate
IsInImage	check if a RINGELEM is in image of RINGHOM
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
IsLRSDegenerate	checks the given polynomial for LRS-degeneracy
IsLRSDegenerateOrder	checks the given polynomial for “n”-LRS-degeneracy
IsMinusOne	test whether an object is -1
IsOne	test whether an object is one
IsPalindromic	test whether a univariate polynomial is palindromic
IsPrimitivePoly	test if polynomial over finite field is primitive
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
JacobianMat	the Jacobian matrix of a list of polynomials

LaguerrePoly	Orthogonal Polynomials: Chebyshev, Hermite, Laguerre
LC	the leading coefficient of a polynomial or ModuleElem
lcm	least common multiple
LF	the leading form of a polynomial or an ideal
LinearSimplify	simplifying linear substitution for a univariate poly over QQ
LM	the leading monomial of a polynomial or ModuleElem
LPP	the leading power-product of a polynomial or ModuleElem
LRSDegeneracyOrder	the LRS degeneracy order of a polynomial
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
MinGBoverZZ [PROTOTYPE]	[PROTOTYPE] minimal Groebner basis of polys over ZZ
MinPoly	minimal polynomial of a matrix
MinPolyQuot	minimal polynomial in quotient ring
MinPowerInIdeal	the minimum power of a polynomial in an ideal
monic	divide polynomials by their leading coefficients
monomials	the list of monomials of a polynomial
MultiArrRestrictionZiegler	Ziegler multirestriction of the arrangement of hyperplanes wrt a hyperplane
MultiplicationMat	multiplication matrix of a ringelem
NewFreeModuleForSyz	create a new FreeModule with shifts for syz
NewMatFilled	matrix filled with value
NF	normal form
NmzEhrhartRing	Ehrhart ring
NmzIntClosureMonIdeal	integral closure of a monomial ideal
NmzIntClosureToricRing	integral closure of a toric ring
NmzNormalToricRing	normalization of a toric ring
NR	normal reduction
num	numerator
NumRealRoots	number of real roots of a polynomial
NumTerms	number of terms in a polynomial
operators, shortcuts	Special characters equivalent to commands
PerpIdealOfForm	Ideal of derivations annihilating a form
power	compute a power
preimage0	preimage of a RINGELEM
prim	primitive part of a polynomial
product	the product of the elements of a list
PthRoot	Compute p-th root
QZP	change field for polynomials and ideals
radical	radical of an ideal
RationalSolve	Rational solutions for 0-dim polynomial system
RationalSolveHomog	Rational solutions for 0-dim polynomial system
RatReconstructPoly	Rational reconstruction of polynomial coefficients
RealRoots	real roots of a univariate polynomial
RealRootsApprox	approximations to the real roots of a univariate poly
resultant	the resultant of two polynomials
RingElem	convert an expression into a RINGELEM
RingOf	the ring of the object
RingsOf	list of the rings of an object
RootBound	bound on roots of a polynomial over QQ
RootBoundTransform	transform of a polynomial, helpful for checking RootBound
SatSAGBI	SAGBI bases for subalgebra
ScientificStr	convert integer/rational to a floating-point string
SetEntry	set an entry into a matrix
SolomonTeraoIdeal	Solomon-Terao ideal of an arrangement of hyperplanes wrt a poly
SqFreeFactor	compute a squarefree factorization
SturmSeq	Sturm sequence of a univariate polynomial

<code>subst</code>	substitute values for indeterminates
<code>sum</code>	the sum of the elements of a list
<code>support</code>	the list of terms of a polynomial or moduleelem
<code>SwinnertonDyerPoly</code>	compute Swinnerton-Dyer polynomial with given roots
<code>SylvesterMat</code>	the Sylvester matrix of two polynomials
<code>syz</code>	syzygy module
<code>ThmProve [PROTOTYPE]</code>	[PROTOTYPE] ThmProve
<code>UnivariateIndetIndex</code>	the index of the indeterminate of a univariate polynomial
<code>wdeg</code>	multi-degree of a polynomial
<code>ZPQ</code>	change field for polynomials and ideals

III-11.4 Commands and Functions returning RINGELEM

<code>abs</code>	absolute value of a number
<code>ArrCharPoly</code>	characteristic polynomial of an arrangement of hyperplanes
<code>ArrPoincarePoly</code>	Poincare polynomial of an arrangement of hyperplanes
<code>ArrTuttePoly</code>	Tutte polynomial of an arrangement of hyperplanes
<code>binomial</code>	binomial coefficient
<code>CanonicalRepr</code>	representative of a class in a quotient ring
<code>CharPoly</code>	characteristic polynomial of a matrix
<code>ChebyshevPoly</code>	Orthogonal Polynomials: Chebyshev, Hermite, Laguerre
<code>ClearDenom</code>	clear common denominator of a polynomial with rational coeffs
<code>CoeffHeight</code>	the maximum of the absolute values of the coefficients of a polynomial
<code>CoeffListWRT</code>	list of coefficients of a polynomial wrt an indet
<code>CoeffListWRTSupport</code>	list of coefficients of a polynomial wrt a power-product basis
<code>CoeffOfTerm</code>	coefficient of a term of a polynomial
<code>CommonDenom</code>	Common denominator of a polynomial with rational coefficients
<code>ComputeElimFirst</code>	ComputeElimFirst
<code>ConstantCoeff</code>	constant coefficient of a polynomial
<code>content</code>	content of a polynomial
<code>ContentWRT</code>	content of a polynomial wrt and indet or a list of indets
<code>cyclotomic</code>	n-th cyclotomic polynomial
<code>den</code>	denominator
<code>DensePoly</code>	the sum of all power-products of a given degree
<code>DenSigma</code>	den of ideal, wrt to ordering sigma
<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>DicksonPoly</code>	Dickson 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 position of the first non-zero entry in a MODULEELEM
<code>FixedDivisor</code>	compute (integer) fixed divisor for polynomial
<code>FrobeniusNormSq</code>	Frobenius norm of a matrix
<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, Laguerre
<code>HilbertPoly</code>	the Hilbert polynomial
<code>homog</code>	homogenize wrt an indeterminate

HomogCompt	homogeneous part of given degree
ImplicitHypersurface	implicitization of hypersurface
indet	individual indeterminates
IndetsProd	(product of) indeterminates actually in a polynomial
Interpolate	interpolating polynomial
interreduced	interreduce a list of polynomials
InverseSystem	Inverse system of an ideal of derivations
JanetBasis	the Janet basis of an ideal
LaguerrePoly	Orthogonal Polynomials: Chebyshev, Hermite, Laguerre
LC	the leading coefficient of a polynomial or ModuleElem
lcm	least common multiple
LF	the leading form of a polynomial or an ideal
LinKerBasis	find the kernel of a matrix
LM	the leading monomial of a polynomial or ModuleElem
LPP	the leading power-product of a polynomial or ModuleElem
LT	the leading term of an object
MakeTerm	returns a monomial (power-product) with given exponents
MinPoly	minimal polynomial of a matrix
MinPolyQuot	minimal polynomial in quotient ring
monic	divide polynomials by their leading coefficients
NF	normal form
NmzDiagInvariants	ring of invariants of a diagonalizable group action
NmzEhrhartRing	Ehrhart ring
NmzFiniteDiagInvariants	ring of invariants of a finite group action
NmzIntClosureMonIdeal	integral closure of a monomial ideal
NmzIntClosureToricRing	integral closure of a toric ring
NmzIntersectionValRings	intersection of ring of valuations
NmzNormalToricRing	normalization of a toric ring
NmzTorusInvariants	ring of invariants of torus action
NR	normal reduction
num	numerator
one	one of a ring
operators, shortcuts	Special characters equivalent to commands
pfaffian	the Pfaffian of a skew-symmetric matrix
PosetCharPoly	characteristic polynomial of a poset from the relations of the poset
PosetPoincarePoly	Poincare polynomial of a poset from the relations of the poset
power	compute a power
prim	primitive part of a polynomial
PthRoot	Compute p-th root
QZP	change field for polynomials and ideals
radical	radical of an ideal
RandomLinearForm	random linear form in polynomial ring
RatReconstructPoly	Rational reconstruction of polynomial coefficients
ReducedGBasis	reduced Groebner basis
resultant	the resultant of two polynomials
RingElem	convert an expression into a RINGELEM
RingElemList, RingElems	convert expressions into a LIST of RINGELEM
RootBoundTransform	transform of a polynomial, helpful for checking RootBound
ScalarProduct	scalar product
SwinnertonDyerPoly	compute Swinnerton-Dyer polynomial with given roots
SymmetricPolys	list of symmetric polynomials
UniversalGBasis	universal Groebner basis of the input ideal
zero	zero of a ring
ZPQ	change field for polynomials and ideals

Chapter III-12

IDEAL

III-12.1 Commands and Functions for IDEAL

<code>AreGensMonomial</code>	checks if given gens are monomial
<code>AreGensSqFreeMonomial</code>	checks if given gens are squarefree monomial
<code>BBasis5</code>	Border Basis of zero dimensional ideal
<code>BettiDiagram</code>	the diagram of the graded Betti numbers
<code>BettiMatrix</code>	the matrix of the graded Betti numbers
<code>BettiNumbers</code>	(Multi-)graded Betti numbers
<code>CallOnGroebnerFanIdeals</code>	apply a function to Groebner fan ideals
<code>colon</code>	ideal or module quotient
<code>ComputeElimFirst</code>	<code>ComputeElimFirst</code>
<code>DenSigma</code>	den of ideal, wrt to ordering sigma
<code>depth</code>	Depth of a module
<code>elim</code>	eliminate variables
<code>EquiIsoDec</code>	equidimensional isoradical decomposition
<code>FrbAlexanderDual</code>	Alexander Dual of monomial ideals
<code>FrbAssociatedPrimes</code>	Associated primes of monomial ideals
<code>FrbIrreducibleDecomposition</code>	Irreducible decomposition of monomial ideals
<code>FrbMaximalStandardMonomials</code>	Maximal standard monomials of monomial ideals
<code>FrbPrimaryDecomposition</code>	Primary decomposition of monomial ideals
<code>FrobeniusMat</code>	matrix of the Frobenius Map
<code>GBasis</code>	calculate a Groebner basis
<code>GBasis timeout</code>	compute a Groebner basis with a timeout
<code>GBasisByHomog</code>	calculate a Groebner basis by homogenization
<code>GenRepr</code>	representation in terms of generators
<code>gens</code>	list of generators of an ideal
<code>gin</code>	generic initial ideal
<code>GroebnerFanIdeals</code>	all reduced Groebner bases of an ideal
<code>GroebnerFanReducedGBases</code>	Groebner fan reduced GBases
<code>HasGBasis</code>	checks if the argument has a pre-computed GBasis
<code>HColon</code>	ideal or module quotient
<code>HilbertFn</code>	the Hilbert function
<code>HilbertSeries</code>	the Hilbert-Poincare series
<code>homog</code>	homogenize wrt an indeterminate
<code>HSaturation</code>	saturation of ideals
<code>IdealOfGBasis</code>	ideal generated by GBasis
<code>IdealOfMinGens</code>	ideal generated by minimal generators
<code>InitialIdeal</code>	Initial ideal
<code>intersection</code>	intersect lists, ideals, or modules
<code>IntersectionList</code>	intersect lists, ideals, or modules

InverseSystem	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
IsSigmaGoodPrime	check if INT is good prime for IDEAL
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 HilbertFn as I
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 Mayer-Vietoris trees
MinGens	list of minimal generators
MinPolyQuot	minimal polynomial in quotient ring
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	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
PrimaryDecomposition	primary decomposition of an ideal
PrimaryDecompositionGTZO	primary decomposition of a 0-dimensional ideal
PrimaryHilbertSeries	primary
PrintBettiDiagram	print 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 rings
QuotientBasisSorted	vector space basis for zero-dimensional quotient rings
QZP	change field for polynomials and ideals
radical	radical of an ideal
RadicalOfUnmixed	radical of an unmixed ideal
ReducedGBasis	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 module
SyzOfGens	syzygy module for a given set of generators
TgCone	tangent cone

ThmProve [PROTOTYPE]	[PROTOTYPE] ThmProve
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

ArtinianOrlikTeraoIdeal	Artinian Orlik-Terao ideal of an arrangement of hyperplanes
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
GinJacobian	generic initial ideal of the Jacobian ideal of a polynomial
HColon	ideal or module quotient
HGBM	intersection of ideals for zero-dimensional schemes
homog	homogenize wrt 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
operators, shortcuts	Special characters equivalent to commands
OrlikTeraoIdeal	Orlik-Terao ideal of an arrangement of hyperplanes
PerpIdealOfForm	Ideal of derivations annihilating a form
PrimaryDecomposition	primary decomposition of an 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
SolomonTeraoIdeal	Solomon-Terao ideal of an arrangement of hyperplanes wrt a poly
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-13.11 pg.209)). 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

adj	classical adjoint matrix (also known as adjugate)
AdjacentMinors	list of adjacent minors of a matrix
AlmostQR	QR decomposition of a matrix
ApproxPointsNBM	Numerical Border Basis of ideal of points
BlockMat	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
FirstCols, FirstRows	submatrix of the first N cols or rows
FrobeniusNormSq	Frobenius norm of a matrix
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
HadamardBoundSq	Hadamard bound for determinant
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
IsZeroDet	test whether determinant is zero
LinKer	find the kernel of a matrix
LinKerBasis	find the kernel of a matrix
LinKerZZ	find the kernel of a matrix
LinSolve	find a solution to a linear system
MakeTermOrdMat	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

NewPolyRingWeights	create a new PolyRing with weights
NmzDiagInvariants	ring of invariants of a diagonalizable group action
NmzFiniteDiagInvariants	ring of invariants of a finite group action
NmzHilbertBasis	Hilbert Basis of a monoid
NmzHilbertBasisKer	Hilbert basis for 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
power	compute a power
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
rref	reduced row echelon form of a matrix
SetCol	set a list as a column into a matrix
SetEntry	set an entry 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)
ArrDerModule	set of generators of the module of logarithmic derivations of an arrangement of hyp
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
FirstCols, FirstRows	submatrix of the first N cols or rows
FrobeniusMat	matrix of the Frobenius Map
GensAsCols, GensAsRows	matrix of generators of a module
GFanGeneratorsOfLinealitySpace	...
GFanGeneratorsOfSpan	...
GFanGetFacets	...
GFanGetImpliedEquations	...
GFanGetUniquePoint	...

<code>GFanRelativeInteriorPoint</code>	relative interior point of a cone
<code>GradingMat</code>	matrix of generalized weights for indeterminates
<code>HilbertMat</code>	create a Hilbert matrix over QQ
<code>IdentityMat</code>	the identity matrix
<code>inverse</code>	multiplicative inverse of matrix
<code>JacobianMat</code>	the Jacobian matrix of a list of polynomials
<code>KroneckerProd</code>	returns the Kronecker product of two matrices
<code>LawrenceMat</code>	Lawrence lifting of a matrix
<code>LexMat</code>	matrices for std. term-orderings
<code>LinKer</code>	find the kernel of a matrix
<code>LinKerZZ</code>	find the kernel of a matrix
<code>LinSolve</code>	find a solution to a linear system
<code>MakeMatByRows, MakeMatByCols</code>	convert a list into a matrix
<code>MakeTermOrdMat</code>	Make a term order matrix from a given matrix
<code>matrix</code>	convert a list into a matrix
<code>MultiArrDerModule</code>	set of generators of the module of logarithmic derivations of a multiarrangement of
<code>MultiplicationMat</code>	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>NmzHilbertBasisKer</code>	Hilbert basis for a monoid
<code>OrdMat</code>	matrix defining a term-ordering
<code>power</code>	compute a power
<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>	matrix for rev lex term-ordering
<code>RowMat</code>	single row matrix
<code>rref</code>	reduced row echelon form of a matrix
<code>StdDegLexMat</code>	matrix for std deg lex term-ordering
<code>StdDegRevLexMat</code>	matrix for std deg rev lex term-ordering
<code>submat</code>	submatrix
<code>SylvesterMat</code>	the Sylvester matrix of two polynomials
<code>transposed</code>	the transposition of a matrix
<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
CanonicalBasis	canonical basis of a free module
colon	ideal or module quotient
elim	eliminate variables
GBasis	calculate a Groebner basis
GBasis timeout	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 wrt an indeterminate
HVector	the h-vector of a module or quotient object
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
MinSubsetOfGens	list of minimal generators
ModuleElem	create a module element
ModuleOf	the module environment of the object
multiplicity	the multiplicity (degree) of a ring
NF	normal form
NumCompts	the number of components
PrintBettiDiagram	print the diagram of the graded Betti numbers
PrintBettiMatrix	print the matrix of the graded Betti numbers
PrintBettiNumbers	print the (multi-)graded Betti numbers
ReducedGBasis	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

<code>SubmoduleCols, SubmoduleRows</code>	convert a matrix into a module
<code>SubmoduleOfMinGens</code>	submodule generated by minimal generators
<code>syz</code>	syzygy module
<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 wrt 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>NewFreeModuleForSyz</code>	create a new FreeModule with shifts for syz
<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 module
<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-13.31](#) pg.216).

See Also: Commands and Functions for MODULEELEM([III-15.2](#) pg.445), Commands and Functions returning MODULEELEM([III-15.3](#) pg.446)

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 position of the first non-zero entry in a MODULEELEM
GenRepr	representation in terms of generators
homog	homogenize wrt 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
ModuleOf	the module environment of the 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

<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>CanonicalBasis</code>	canonical basis of a free module
<code>homog</code>	homogenize wrt 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>	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.327](#)) returns “S”, the tag string of an object, and the function “untagged” ([I-21.5 pg.336](#)) 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 am 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