

CoCoA 5.1.0 Manual

June 4, 2014

Contents

I	Alphabetical List of Commands	19
I-0	Special Characters	21
I-0.1	operators, shortcuts	21
I-1	A	23
I-1.1	abs	23
I-1.2	adjoint	23
I-1.3	AffHilbert	24
I-1.4	AffHilbertFn	24
I-1.5	AffHilbertSeries	24
I-1.6	AffPoincare	25
I-1.7	alias	25
I-1.8	aliases	26
I-1.9	AlmostQR	26
I-1.10	and	27
I-1.11	append	27
I-1.12	apply	28
I-1.13	ascii	28
I-1.14	AsINT	29
I-1.15	AsRAT	29
I-2	B	31
I-2.1	BaseRing	31
I-2.2	BBasis5	31
I-2.3	binomial	31
I-2.4	BinomialRepr, BinomialReprShift	32
I-2.5	block	33
I-2.6	BlockMat	33
I-2.7	BlockMat2x2	34
I-2.8	Bool01	34
I-2.9	break	35
I-2.10	BringIn	35
I-3	C	37

I-3.1	Call	37
I-3.2	CanonicalHom	37
I-3.3	CartesianProduct, CartesianProductList	38
I-3.4	Cast	38
I-3.5	ceil	38
I-3.6	CFApprox	39
I-3.7	CFApproximants	39
I-3.8	characteristic	39
I-3.9	CharPoly	40
I-3.10	CheckArgTypes	40
I-3.11	ciao	41
I-3.12	ClearDenom	41
I-3.13	close	41
I-3.14	CloseLog	42
I-3.15	CoCoA-4 mode	42
I-3.16	CocoaLimits	43
I-3.17	CocoaPackagePath	43
I-3.18	CoeffEmbeddingHom	43
I-3.19	coefficients	43
I-3.20	CoefficientsWRT	44
I-3.21	CoeffListWRT	45
I-3.22	CoeffOfTerm	46
I-3.23	CoeffRing	46
I-3.24	ColMat	46
I-3.25	colon	47
I-3.26	ColumnVectors	47
I-3.27	Comp	48
I-3.28	Comparison Operators	48
I-3.29	CompleteToOrd	48
I-3.30	compts	49
I-3.31	concat	50
I-3.32	ConcatAntiDiag	50
I-3.33	ConcatDiag	50
I-3.34	ConcatHor	51
I-3.35	ConcatHorList	51
I-3.36	ConcatLists	52
I-3.37	ConcatVer	52
I-3.38	ConcatVerList	53
I-3.39	content	53
I-3.40	ContentFreeFactor	54
I-3.41	ContentWRT	54
I-3.42	ContFrac	55
I-3.43	ContFracToRat	55

I-3.44	count	55
I-3.45	CpuTime	56
I-3.46	CRT	56
I-3.47	CurrentRing	56
I-3.48	CurrentTypes	57
I-3.49	cyclotomic	57
I-4	D	59
I-4.1	dashes	59
I-4.2	date	59
I-4.3	DecimalStr	59
I-4.4	define	60
I-4.5	DefiningIdeal	61
I-4.6	deg	61
I-4.7	den	62
I-4.8	DensePoly	63
I-4.9	Depth	63
I-4.10	deriv	64
I-4.11	DerivationAction	65
I-4.12	describe	65
I-4.13	det	65
I-4.14	DF	66
I-4.15	DiagMat	66
I-4.16	diff	67
I-4.17	dim	67
I-4.18	discriminant	68
I-4.19	distrib	68
I-4.20	div	68
I-4.21	DivAlg	69
I-5	E	71
I-5.1	E_	71
I-5.2	eigenvectors	71
I-5.3	elim	72
I-5.4	ElimMat	73
I-5.5	EqSet	73
I-5.6	Equality Test	73
I-5.7	EquiIsoDec	74
I-5.8	error	74
I-5.9	eval	75
I-5.10	EvalHilbertFn	75
I-5.11	Ext	76
I-6	F	79

I-6.1	factor	79
I-6.2	factorial	79
I-6.3	FactorMultiplicity	80
I-6.4	FGLM5	80
I-6.5	fields	81
I-6.6	first	81
I-6.7	FirstNonZero	81
I-6.8	FirstNonZeroPosn	82
I-6.9	flatten	82
I-6.10	FloatApprox	83
I-6.11	FloatStr	83
I-6.12	floor	84
I-6.13	for	84
I-6.14	foreach	85
I-6.15	format	86
I-6.16	FrbAlexanderDual	86
I-6.17	FrbAssociatedPrimes	87
I-6.18	FrbIrreducibleDecomposition	87
I-6.19	FrbMaximalStandardMonomials	88
I-6.20	FrbPrimaryDecomposition	88
I-6.21	func	88
I-6.22	Function	89
I-6.23	functions	89

I-7	G	91
I-7.1	GBasis	91
I-7.2	GBasisTimeout	91
I-7.3	GBM	92
I-7.4	ged	92
I-7.5	GCDFreeBasis	93
I-7.6	GenericPoints	93
I-7.7	GenRepr	94
I-7.8	gens	94
I-7.9	GensAsCols, GensAsRows	95
I-7.10	Get	96
I-7.11	GetCol	96
I-7.12	GetCols	97
I-7.13	GetEnv	97
I-7.14	GetErrMesg	97
I-7.15	GetRow	98
I-7.16	GetRows	98
I-7.17	Gin, Gin5	98
I-7.18	GradingDim	99

I-8	H	101
I-8.1	HColon	101
I-8.2	HGBM	101
I-8.3	hilbert	102
I-8.4	HilbertBasisKer	102
I-8.5	HilbertFn	103
I-8.6	HilbertPoly	103
I-8.7	HilbertSeries	104
I-8.8	HilbertSeriesMultiDeg	105
I-8.9	HilbertSeriesShifts	106
I-8.10	homog	106
I-8.11	HomogElimMat	107
I-8.12	HSaturation	107
I-8.13	HVector	108
I-9	I	109
I-9.1	ideal	109
I-9.2	IdealAndSeparatorsOfPoints	109
I-9.3	IdealAndSeparatorsOfProjectivePoints	110
I-9.4	IdealOfPoints	112
I-9.5	IdealOfProjectivePoints	113
I-9.6	IdentityMat	114
I-9.7	if	114
I-9.8	ILogBase	114
I-9.9	image	115
I-9.10	ImplicitPlot	115
I-9.11	ImplicitPlotOn	116
I-9.12	ImportByRef, ImportByValue	116
I-9.13	in	117
I-9.14	incr, decr	117
I-9.15	indent	118
I-9.16	indet	119
I-9.17	IndetIndex	119
I-9.18	IndetName	119
I-9.19	indets	120
I-9.20	IndetSubscripts	121
I-9.21	IndetSymbols	121
I-9.22	InducedHom	122
I-9.23	InitialIdeal	122
I-9.24	insert	123
I-9.25	Interpolate	123
I-9.26	interreduce, interreduced	124
I-9.27	intersection	124

I-9.28	IntersectList	125
I-9.29	inverse	125
I-9.30	InverseSystem	126
I-9.31	IO.SprintTrunc	126
I-9.32	iroot	126
I-9.33	IsAntiSymmetric	127
I-9.34	IsConstant	127
I-9.35	IsContained	127
I-9.36	IsDefined	128
I-9.37	IsDiagonal	128
I-9.38	IsDivisible	128
I-9.39	IsElem	129
I-9.40	IsEven, IsOdd	129
I-9.41	IsFactorClosed	130
I-9.42	IsField	130
I-9.43	IsFiniteField	130
I-9.44	IsHomog	131
I-9.45	IsIn	131
I-9.46	IsIndet	132
I-9.47	IsInRadical	132
I-9.48	IsInSubalgebra	133
I-9.49	IsLexSegment	133
I-9.50	IsNumber	133
I-9.51	IsPositiveGrading	134
I-9.52	IsPrime	134
I-9.53	IsProbPrime	134
I-9.54	IsPthPower	135
I-9.55	isqrt	135
I-9.56	IsQuotientRing	136
I-9.57	IsStable	136
I-9.58	IsStdGraded	136
I-9.59	IsStronglyStable	137
I-9.60	IsSubset	137
I-9.61	IsSymmetric	137
I-9.62	IsTerm	138
I-9.63	IsTermOrdering	138
I-9.64	IsTree5	139
I-9.65	IsTrueGCDDomain	139
I-9.66	IsZero	140
I-9.67	IsZeroCol, IsZeroRow	140
I-9.68	IsZeroDim	141
I-9.69	IsZeroDivisor	141
I-9.70	It	141

I-10	J		143
I-10.1	jacobian		143
I-10.2	JanetBasis		143
I-11	K		145
I-11.1	ker		145
I-12	L		147
I-12.1	last		147
I-12.2	Latex		147
I-12.3	LC		148
I-12.4	lcm		148
I-12.5	len		149
I-12.6	LexMat		149
I-12.7	LexSegmentIdeal		150
I-12.8	LF		150
I-12.9	LinearSimplify		151
I-12.10	LinKer		151
I-12.11	LinKerBasis		152
I-12.12	LinKerModP		152
I-12.13	LinSol		153
I-12.14	LinSolve		153
I-12.15	List Constructors		154
I-12.16	LM		154
I-12.17	log		155
I-12.18	LogCardinality		155
I-12.19	LogToTerm		155
I-12.20	LPosn		156
I-12.21	LPP		156
I-12.22	LT		157
I-13	M		159
I-13.1	MakeCheck		159
I-13.2	MakeMatByRows, MakeMatByCols		159
I-13.3	MakeSet		160
I-13.4	MantissaAndExponent10		160
I-13.5	MantissaAndExponent2		160
I-13.6	Manual		161
I-13.7	MapDown		162
I-13.8	matrix		162
I-13.9	Max		163
I-13.10	MayerVietorisTreeN1		163
I-13.11	Min		164
I-13.12	MinGens		164

I-13.13	MinGensGeneral	165
I-13.14	minimalize	165
I-13.15	minimalized	165
I-13.16	MinimalPresentation	166
I-13.17	minors	166
I-13.18	MinPoly	167
I-13.19	MinPowerInIdeal	167
I-13.20	mod	167
I-13.21	Mod2Rat	168
I-13.22	ModuleElem	168
I-13.23	ModuleOf	168
I-13.24	monic	169
I-13.25	monomials	169
I-13.26	MonsInIdeal	170
I-13.27	multiplicity	170
I-14	N	171
I-14.1	NewFractionField	171
I-14.2	NewFreeModule	171
I-14.3	NewId	172
I-14.4	NewLine	172
I-14.5	NewList	172
I-14.6	NewMat	173
I-14.7	NewMatFilled	173
I-14.8	NewPolyRing	174
I-14.9	NewQuotientRing	174
I-14.10	NewRingFp	175
I-14.11	NewRingTwinFloat	175
I-14.12	NextPrime	175
I-14.13	NextProbPrime	176
I-14.14	NF	176
I-14.15	NFsAreZero	177
I-14.16	NmzComputation	177
I-14.17	NmzHilbertBasis	178
I-14.18	NmzIntClosureMonIdeal	178
I-14.19	NmzIntClosureToricRing	178
I-14.20	NmzNormalToricRing	179
I-14.21	NonZero	179
I-14.22	not	179
I-14.23	NR	180
I-14.24	num	180
I-14.25	NumCols	181
I-14.26	NumCompts	181

I-14.27	NumIndets	181
I-14.28	NumPartitions	182
I-14.29	NumRows	182
I-14.30	NumTerms	182
I-15	O	185
I-15.1	one	185
I-15.2	OpenIFile	185
I-15.3	OpenIString	186
I-15.4	OpenLog	186
I-15.5	OpenOFile	187
I-15.6	OpenOString	188
I-15.7	OpenSocket	188
I-15.8	Option	189
I-15.9	or	189
I-15.10	OrdMat	189
I-16	P	191
I-16.1	Packages	191
I-16.2	panel	191
I-16.3	panels	191
I-16.4	partitions	192
I-16.5	permutations	192
I-16.6	PerpIdealOfForm	192
I-16.7	pfaffian	193
I-16.8	PkgName	193
I-16.9	PlotPoints	194
I-16.10	PlotPointsOn	194
I-16.11	poincare	195
I-16.12	PoincareMultiDeg	195
I-16.13	PoincareShifts	195
I-16.14	PolyAlgebraHom	195
I-16.15	PolyRingHom	196
I-16.16	PowerMod	196
I-16.17	PreprocessPts	197
I-16.18	PrimaryDecomposition	198
I-16.19	PrimaryPoincare	198
I-16.20	PrimitiveRoot	199
I-16.21	print	199
I-16.22	print on	200
I-16.23	PrintBettiDiagram	200
I-16.24	PrintBettiMatrix	201
I-16.25	println	201

I-16.26	PrintRes	202
I-16.27	product	202
I-16.28	protect	203
I-16.29	PthRoot	203
I-17	Q	205
I-17.1	QQ	205
I-17.2	quit	205
I-17.3	QuotientBasis	205
I-17.4	QZP	206
I-18	R	209
I-18.1	radical	209
I-18.2	RadicalOfUnmixed	209
I-18.3	random	210
I-18.4	randomize	210
I-18.5	randomized	211
I-18.6	rank	211
I-18.7	RatReconstructByContFrac, RatReconstructByLattice	212
I-18.8	RatReconstructWithBounds	212
I-18.9	RealRootRefine	213
I-18.10	RealRoots	213
I-18.11	RealRootsApprox	214
I-18.12	record	215
I-18.13	record field selector	215
I-18.14	ReducedGBasis	216
I-18.15	ref	216
I-18.16	RefineGCDFreeBasis	217
I-18.17	Reg, Reg5	217
I-18.18	RegularityIndex	218
I-18.19	RelNotes	218
I-18.20	ReloadMan	218
I-18.21	remove	219
I-18.22	repeat	219
I-18.23	res	220
I-18.24	Reset	221
I-18.25	ResetPanels	221
I-18.26	resultant	221
I-18.27	return	221
I-18.28	reverse, reversed	222
I-18.29	RevLexMat	222
I-18.30	RingElem	223
I-18.31	RingOf	224

I-18.32	RingQQ	224
I-18.33	RingQQt	225
I-18.34	RingSet	225
I-18.35	RingZZ	226
I-18.36	RMap	226
I-18.37	RootBound	226
I-18.38	round	227
I-18.39	RowMat	227
I-19	S	229
I-19.1	saturate	229
I-19.2	ScalarProduct	229
I-19.3	ScientificStr	230
I-19.4	seed	230
I-19.5	SeparatorsOfPoints	231
I-19.6	SeparatorsOfProjectivePoints	232
I-19.7	shape	233
I-19.8	sign	234
I-19.9	SimplestRatBetween	234
I-19.10	size	234
I-19.11	skip	235
I-19.12	SmoothFactor	235
I-19.13	sort	235
I-19.14	SortBy	236
I-19.15	sorted	236
I-19.16	SortedBy	237
I-19.17	source	238
I-19.18	SourceRegion	238
I-19.19	spaces	238
I-19.20	sprint	239
I-19.21	SqFreeFactor	239
I-19.22	StableBBasis5	240
I-19.23	StableIdeal	241
I-19.24	StarPrint, StarSprint	241
I-19.25	starting	242
I-19.26	StdDegLexMat	242
I-19.27	StdDegRevLexMat	243
I-19.28	StronglyStableIdeal	243
I-19.29	SubalgebraMap	243
I-19.30	SubalgebraRepr	244
I-19.31	submat	244
I-19.32	submodule	245
I-19.33	SubmoduleCols, SubmoduleRows	245

I-19.34	subsets	246
I-19.35	subst	246
I-19.36	sum	247
I-19.37	support	248
I-19.38	swap	248
I-19.39	sylvester	248
I-19.40	SymbolRange	249
I-19.41	syz	249
I-19.42	SyzOfGens	250
I-20	T	253
I-20.1	tag	253
I-20.2	tagged	253
I-20.3	tail	254
I-20.4	TensorMat	254
I-20.5	TgCone	254
I-20.6	TimeFrom	255
I-20.7	TimeOfDay	255
I-20.8	TmpNBM	256
I-20.9	TopLevel	256
I-20.10	TopLevelFunctions	257
I-20.11	toric	257
I-20.12	transposed	258
I-20.13	try	259
I-20.14	tuples	259
I-20.15	type	260
I-21	U	261
I-21.1	UnivariateIndetIndex	261
I-21.2	unprotect	261
I-21.3	Unset	262
I-21.4	untagged	262
I-21.5	updating CoCoA-4 code	262
I-21.6	use	263
I-22	V	265
I-22.1	valuation	265
I-22.2	VersionInfo	265
I-23	W	267
I-23.1	wdeg	267
I-23.2	WeightsMatrix	267
I-23.3	while	268
I-23.4	WithoutNth	268

	I-23.5	WLog	269
I-24	X		271
	I-24.1	XelMat	271
I-25	Z		273
	I-25.1	zero	273
	I-25.2	ZeroMat	273
	I-25.3	ZPQ	274
	I-25.4	ZZ	274
II	The CoCoA Programming Language		277
II-1	Introduction to CoCoA Programming		279
	II-1.1	An Overview of CoCoA Programming	279
II-2	Language Elements		281
	II-2.1	Character Set and Special Symbols	281
	II-2.2	Identifiers	281
	II-2.3	Reserved Names	282
	II-2.4	Comments	282
II-3	Operators		283
	II-3.1	CoCoA Operators	283
	II-3.2	Algebraic Operators	283
	II-3.3	Relational Operators	284
	II-3.4	Selection Operators	284
	II-3.5	Range Operator	284
II-4	Evaluation and Assignment		287
	II-4.1	Evaluation	287
	II-4.2	Assignment	287
II-5	Flow Control: Conditional Statements and Loops		289
	II-5.1	Commands and Functions for Branching	289
	II-5.2	Commands and Functions for Loops	289
II-6	Input/Output		291
	II-6.1	Introduction to IO	291
	II-6.2	Standard IO	291
	II-6.3	File IO	291
	II-6.4	String IO	292
	II-6.5	Commands and Functions for IO	293
	II-6.6	Tagged Printing	293
	II-6.7	Tagging an Object	293

II-6.8	Printing a Tagged Object	294
II-6.9	Describing a Tagged Object	294
II-6.10	Commands and Functions for Tags	295
II-7	CoCoA Packages	297
II-7.1	Introduction to Packages	297
II-7.2	First Example of a Package	297
II-7.3	Package Essentials	298
II-7.4	Global Aliases	298
II-7.5	Sharing Your Package	298
II-7.6	Commands and Functions for Packages	298
II-7.7	Supported Packages	299
II-7.8	Galois Package	299
II-7.9	Integer Programming	299
II-7.10	Algebra of Invariants	299
II-7.11	Special Varieties	299
II-7.12	Statistics	300
II-7.13	Geometrical Theorem-Proving	300
II-7.14	Typevectors	300
II-7.15	Conductor	300
II-7.16	Matrix Normal Form	301
II-7.17	CantStop	301
II-7.18	Control	301
II-8	Linked libraries	303
II-8.1	CoCoALib	303
II-8.2	GMP	303
II-8.3	Frobby	303
II-8.4	Normaliz	303
III	CoCoA datatypes	305
III-1	BOOL	307
III-1.1	Introduction to BOOL	307
III-1.2	Commands and Functions for BOOL	307
III-1.3	Commands and Functions returning BOOL	307
III-2	INT	309
III-2.1	Introduction to INT	309
III-2.2	Commands and Functions for INT	309
III-2.3	Commands and Functions returning INT	311
III-3	RAT	313
III-3.1	Introduction to RAT	313

III-3.2	Commands and Functions for RAT	313
III-3.3	Commands and Functions returning RAT	314
III-4	STRING	315
III-4.1	String Literals	315
III-4.2	String Operations	315
III-4.3	Commands and Functions for STRING	316
III-5	LIST	317
III-5.1	Introduction to Lists	317
III-5.2	Commands and Functions for LIST	317
III-6	RECORD	321
III-6.1	Introduction to RECORD	321
III-6.2	Commands and Functions for RECORD	322
III-6.3	Commands and Functions returning RECORD	322
III-7	FUNCTION	323
III-7.1	Introduction to FUNCTION	323
III-7.2	FUNCTIONs are first class objects	323
III-7.3	Commands and Functions for FUNCTION	323
III-7.4	Commands and Functions returning FUNCTION	324
III-8	TYPE	325
III-8.1	Commands and Functions for TYPE	325
III-8.2	Commands and Functions returning TYPE	325
III-9	RING	327
III-9.1	Introduction to Rings	327
III-9.2	Polynomial Rings	327
III-9.3	Coefficient Rings	328
III-9.4	Indeterminates	329
III-9.5	Orderings	329
III-9.6	Module Orderings	329
III-9.7	Accessing Other Rings	330
III-9.8	Quotient Rings	330
III-9.9	Commands and Functions for RING	330
III-9.10	Commands and Functions returning RING	332
III-10	RINGHOM	333
III-10.1	Commands and Functions for RINGHOM	333
III-10.2	Commands and Functions returning RINGHOM	333
III-11	RINGELEM	335
III-11.1	Introduction to RINGELEM	335
III-11.2	Evaluation of Polynomials	336

III-11.3	Commands and Functions for RINGELEM	336
III-11.4	Commands and Functions returning RINGELEM	338
III-12	IDEAL	341
III-12.1	Commands and Functions for IDEAL	341
III-12.2	Commands and Functions returning IDEAL	342
III-13	MAT	345
III-13.1	Introduction to MAT	345
III-13.2	Commands and Functions for MAT	345
III-13.3	Commands and Functions returning MAT	347
III-14	MODULE	349
III-14.1	Commands and Functions for MODULE	349
III-14.2	Commands and Functions returning MODULE	350
III-15	MODULEELEM	351
III-15.1	Introduction to MODULEELEM	351
III-15.2	Commands and Functions for MODULEELEM	351
III-15.3	Commands and Functions returning MODULEELEM	351

Part I

Alphabetical List of Commands

Chapter I-0

Special Characters

I-0.1 operators, shortcuts

syntax

```
A := B    A:variable, B:OBJECT
A = B     A,B:OBJECT
A <> B    A,B:OBJECT
[...]     LIST
[..|...]  LIST
L[N]      N: INT, L: LIST
A..B      A,B: INT or A,B:indeterminates
R.F       R:RECORD and F field name
R ::= ... R:variable
A >< B     A,B: LIST
M : N     M, N: MODULE or IDEAL
S[N]      N: INT, S: STRING
***E***   E:expression
?S        S: STRING
<< S      S: STRING
```

Description

“A := B;” compute “B” then assign the result to “A” “A = B” test whether “A” and “B” are equal “A <> B” test whether “A” and “B” are not equal “[...]” build a new list (see “List Constructors” (I-12.15 pg.154)) “[..|...]” build a new list (see “List Constructors” (I-12.15 pg.154)) “L[N]” access “N”-th entry of list “L” (indexes start from 1) “A..B” is the “Range Operator” (II-3.5 pg.284) “R.F” “record field selector” (I-18.13 pg.215) for field named “F” of record “R” “R ::= ” for the special ring syntax (“NewPolyRing” (I-14.8 pg.174))

“A >< B” equivalent to “CartesianProduct(A, B)”, “CartesianProductList([A,B])” “M : N” equivalent to “colon(M, N)” “R/I” equivalent to “NewQuotientRing(R,I)” “S[N]” access “N”-th char of string “L” (indexes start from 1) “***E***” interpret “E” in “CoCoA-4 mode” (I-3.15 pg.42) “\$<< S\$” OBSOLESCENT equivalent to “source(S)”

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

See Also: colon(I-3.25 pg.47), List Constructors(I-12.15 pg.154), CartesianProduct, CartesianProductList(I-3.3 pg.38), NewPolyRing(I-14.8 pg.174), NewQuotientRing(I-14.9 pg.174), record field selector(I-18.13 pg.215), Range Operator(II-3.5 pg.284), source(I-19.17 pg.238), Manual(I-13.6 pg.161), Character Set and Special Symbols(II-2.1 pg.281), CoCoA Operators(II-3.1 pg.283)

Chapter I-1

A

I-1.1 abs

syntax

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

Description

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

example

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

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

I-1.2 adjoint

syntax

```
adjoint(M: MAT): MAT
```

Description

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

example

```
/**/ Use R := QQ[t,x,y,z];
/**/ adjoint(mat([[x,y,z],[t,y,x],[x,x^2,x*y]]));
matrix([
  [-x^3 +x*y^2, -x*y^2 +x^2*z, x*y -y*z],
  [-t*x*y +x^2, x^2*y -x*z, -x^2 +t*z],
  [t*x^2 -x*y, -x^3 +x*y, -t*y +x*y]
])

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

```
[[1, -2],
 [2, 1]])
```

I-1.3 AffHilbert

syntax

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

Description

The first form of this function computes the affine Hilbert function for R . The second form computes the N -th value of the affine Hilbert function. The weights of the indeterminates of R must all be 1. For repeated evaluations of the Hilbert function, use “EvalHilbertFn” (I-5.10 pg.75) instead of “hilbert(R , N)” in order to speed up execution.

This function is the same as “AffHilbertFn” (I-1.4 pg.24).

The coefficient ring must be a field.

example

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

See Also: AffHilbertSeries(I-1.5 pg.24), EvalHilbertFn(I-5.10 pg.75)

I-1.4 AffHilbertFn

syntax

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

Description

Same as “AffHilbert” (I-1.3 pg.24).

See Also: EvalHilbertFn(I-5.10 pg.75), HilbertPoly(I-8.6 pg.103), HVector(I-8.13 pg.108), HilbertSeries(I-8.7 pg.104)

I-1.5 AffHilbertSeries

syntax

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

Description

This function computes the affine Hilbert-Poincare series of M . The grading must be a positive Z^1 -grading (i.e. “WeightsMatrix” (I-23.2 pg.267) 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 is exactly the same as “AffPoincare” (I-1.6 pg.25).

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: AffHilbert(I-1.3 pg.24), HilbertSeries(I-8.7 pg.104)

I-1.6 AffPoincare

syntax

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

Description

Same as “AffHilbertSeries” (I-1.5 pg.24).

See Also: AffHilbert(I-1.3 pg.24), AffHilbertSeries(I-1.5 pg.24), hilbert(I-8.3 pg.102), HilbertSeries(I-8.7 pg.104)

I-1.7 alias

syntax

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

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

Description

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

1. Global aliases. To avoid typing the full package name as a prefix to package functions, one may declare a short global alias during a CoCoA session. A list of the global aliases is produced by the function “aliases” (I-1.8 pg.26). For examples, see the chapter on packages in the manual, in particular the section, “Global Aliases” (II-7.4 pg.298). Online, enter “?global aliases”.

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

example

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

```

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

```

See Also: [aliases\(I-1.8 pg.26\)](#), [Introduction to Packages\(II-7.1 pg.297\)](#)

I-1.8 aliases

syntax

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

Description

This function prints a list of global aliases for packages. Aliases are formed with the function “[alias\(I-1.7 pg.25\)](#)”.

example

```

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

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

```

See Also: [alias\(I-1.7 pg.25\)](#), [Introduction to Packages\(II-7.1 pg.297\)](#)

I-1.9 AlmostQR

syntax

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

Description

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

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

example

```

/**/  M := matrix(QQ, [ [4, -2, 3], [3, 2, -2], [0, 0, 3] ]);
/**/  Dec := AlmostQR(M);
/**/  Dec;
record[Q := matrix(QQ,
  [[4, -42/25, 0],
   [3, 56/25, 0],
   [0, 0, 3]])
, R := matrix(QQ,

```

```

[[1, -2/25, 6/25],
 [0, 1, -17/14],
 [0, 0, 1]]
]

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

```

See Also: [Matrix Normal Form](#)([II-7.16](#) pg.301)

I-1.10 and

syntax

```

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

```

Description

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

example

```

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

```

See Also: [or](#)([I-15.9](#) pg.189), [not](#)([I-14.22](#) pg.179)

I-1.11 append

syntax

```

append(ref L: LIST, E: OBJECT)

```

Description

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

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

example

```

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

```

```
/**/ L;
[1, 2, 3, 4]
```

See Also: [ref\(I-18.15 pg.216\)](#), [concat\(I-3.31 pg.50\)](#), [ConcatLists\(I-3.36 pg.52\)](#), [insert\(I-9.24 pg.123\)](#), [remove\(I-18.21 pg.219\)](#)

I-1.12 apply

syntax

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

Description

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

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

example

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

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

See Also: [PolyAlgebraHom\(I-16.14 pg.195\)](#), [CanonicalHom\(I-3.2 pg.37\)](#)

I-1.13 ascii

syntax

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

Description

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

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

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

example

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

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

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

I-1.14 AsINT

syntax

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

Description

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

example

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

See Also: AsRAT([I-1.15](#) pg.29)

I-1.15 AsRAT

syntax

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

Description

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

example

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

See Also: AsINT([I-1.14](#) pg.29)

Chapter I-2

B

I-2.1 BaseRing

— syntax —

```
BaseRing(M: MAT: RING
BaseRing(M: MODULE: RING
BaseRing(RmodI: (Quotient)RING: RING
BaseRing(K: (Fraction Field)RING): RING
```

Description

Might be replaced completely by “RingOf” ([I-18.31 pg.224](#))

— example —

```
...work in progress...
```

See Also: [matrix\(I-13.8 pg.162\)](#), [NewFractionField\(I-14.1 pg.171\)](#), [NewQuotientRing\(I-14.9 pg.174\)](#), [NewFreeModule\(I-14.2 pg.171\)](#)

I-2.2 BBasis5

— syntax —

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

Description

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

This function is implemented in ApCoCoALib by Stefan Kaspar.

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

— example —

```
Use QQ[x, y], DegLex;
I := ideal([x^2, xy + y2]);
BBasis := BBasis5(I);
```

I-2.3 binomial

— syntax —

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

Description

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

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

example

```

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

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

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

```

See Also: BinomialRepr, BinomialReprShift(I-2.4 pg.32)

I-2.4 BinomialRepr, BinomialReprShift

syntax

```

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

```

where N and K are positive.

Description

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

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

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

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

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

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

example

```

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

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

```

See Also: binomial(I-2.3 pg.31)

I-2.5 *block*

syntax

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

where each C_i is a command.
```

Description

The “**block**” command executes the commands as if they where 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
-----
/**/ 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-2.6 *BlockMat*

syntax

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

Description

This function creates a block matrix from a LIST of rows of matrices.

The following restrictions on the sizes of the matrices apply: in each row of matrices “**NumRows(M)**” must be constant, and for all rows of matrices the total number of columns must be the same.

The function “**BlockMat2x2**” (I-2.7 pg.34) 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]])
```

See Also: [ConcatHor\(I-3.34 pg.51\)](#), [ConcatVer\(I-3.37 pg.52\)](#), [ConcatHorList\(I-3.35 pg.51\)](#), [ConcatVerList\(I-3.38 pg.53\)](#), [ConcatDiag\(I-3.33 pg.50\)](#), [ConcatAntiDiag\(I-3.32 pg.50\)](#), [BlockMat2x2\(I-2.7 pg.34\)](#)

I-2.7 BlockMat2x2

syntax

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

Description

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

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

The obvious restrictions on the sizes of the matrices apply:

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

The function “BlockMat” ([I-2.6 pg.33](#)) 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\(I-3.34 pg.51\)](#), [ConcatVer\(I-3.37 pg.52\)](#), [ConcatDiag\(I-3.33 pg.50\)](#), [ConcatAntiDiag\(I-3.32 pg.50\)](#), [BlockMat\(I-2.6 pg.33\)](#)

I-2.8 Bool01

syntax

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

Description

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

example

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

```
[[1, 0, 0, 0],
 [0, 1, 0, 0],
 [0, 0, 1, 0],
 [0, 0, 0, 1]])
```

I-2.9 *break*

— syntax —

```
break
```

Description

This command must be used inside a loop statement (“for”, “foreach”, “repeat”, or “while”). When executed, the current loop statement is terminated and control passes to the command following the loop statement. Thus, in the case of nested loops “break” does “not” break out of all loops back to the “top level” (see “Return”).

— example —

```
/**/ For I := 5 To 1 Step -1 Do
/**/   For J := 1 To 100 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: [return\(I-18.27 pg.221\)](#)

I-2.10 *BringIn*

— syntax —

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

Description

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

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

— Changing characteristic 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).

If the two polynomial rings differ only in characteristic then it is faster to use the functions “QZP” ([I-17.4 pg.206](#)), “ZPQ” ([I-25.3 pg.274](#)).

```

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

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

-- NOT YET IMPLEMENTED -- changing characteristic
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) = BringIn(F);
True
-----

```

See Also: PolyAlgebraHom(I-16.14 pg.195), apply(I-1.12 pg.28), image(I-9.9 pg.115), QZP(I-17.4 pg.206), ZPQ(I-25.3 pg.274)

Chapter I-3

C

I-3.1 Call

syntax

[OBSOLETE]

Description

OBSOLETE: in CoCoA-5 functions can be used directly.

See Also: FUNCTIONS are first class objects([III-7.2 pg.323](#))

I-3.2 CanonicalHom

syntax

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

Description

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

```
ZZ -> S      QQ -> S
R -> R/I      R -> 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
```

See Also: NewFractionField([I-14.1 pg.171](#)), NewQuotientRing([I-14.9 pg.174](#)), NewPolyRing([I-14.8 pg.174](#)), CanonicalHom([I-3.2 pg.37](#)), PolyAlgebraHom([I-16.14 pg.195](#)), PolyRingHom([I-16.15 pg.196](#))

I-3.3 CartesianProduct, CartesianProductList

syntax

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

where each Li is a LIST
```

Description

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

For the N-fold product of a list with itself, one may use “tuples” (I-20.14 pg.259).

example

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

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

See Also: CoCoA Operators(II-3.1 pg.283), operators, shortcuts(I-0.1 pg.21), tuples(I-20.14 pg.259)

I-3.4 Cast

syntax

[OBSOLETE]

Description

[OBSOLETE] To cast INT, RAT, STRING to a polynomial (and more in general to a RINGELEM) use “RingElem” (I-18.30 pg.223).

To cast RINGELEM to INT, RAT use “AsINT” (I-1.14 pg.29), “AsRAT” (I-1.15 pg.29).

To cast LIST to MAT use “matrix” (I-13.8 pg.162). To cast MAT to LIST use “GetRows” (I-7.16 pg.98), “GetCols” (I-7.12 pg.97).

See Also: AsINT(I-1.14 pg.29), AsRAT(I-1.15 pg.29), gens(I-7.8 pg.94), GensAsCols, GensAsRows(I-7.9 pg.95), GetCols(I-7.12 pg.97), GetRows(I-7.16 pg.98), ideal(I-9.1 pg.109), matrix(I-13.8 pg.162), ModuleElem(I-13.22 pg.168), RingElem(I-18.30 pg.223), SubmoduleCols, SubmoduleRows(I-19.33 pg.245)

I-3.5 ceil

syntax

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

Description

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

example

```
/**/ ceil(0.99);
1

/**/ ceil(0.01);
1

/**/ ceil(1);
1

/**/ ceil(-0.99);
0
```

See Also: floor([I-6.12](#) pg.84), round([I-18.38](#) pg.227), num([I-14.24](#) pg.180), den([I-4.7](#) pg.62)

I-3.6 CFAprox

syntax

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

Description

“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-3.7](#) pg.39), ContFrac([I-3.42](#) pg.55), SimplestRatBetween([I-19.9](#) pg.234)

I-3.7 CFAproximants

syntax

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

Description

“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-3.6](#) pg.39), ContFrac([I-3.42](#) pg.55)

I-3.8 characteristic

syntax

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

Description

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

example

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

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

See Also: IsFiniteField([I-9.43](#) pg.130), LogCardinality([I-12.18](#) pg.155)

I-3.9 CharPoly

syntax

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

Description

X is an indeterminate, and M is a square matrix whose entries are in the coefficient ring of X.

This function returns the characteristic polynomial of M in the indeterminate X. See also “MinPoly” ([I-13.18](#) pg.167).

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-13.18](#) pg.167)

I-3.10 CheckArgTypes

syntax

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

Description

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

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

example

```
-- the following returns nothing
CheckArgTypes("MyFunc", [INT, POLY, IDEAL], [10, 20, ideal(x)]);
-- the following returns an error for the 3rd argument
CheckArgTypes("MyFunc", [INT, POLY, IDEAL], [10, 20, 30]);
ERROR: MyFunc: arg 3 is INT but must be IDEAL
CONTEXT: Return(S)
-----
```

```

-- an example of use for type checking
Define Power(F, N)
  CheckArgTypes("Power", [POLY, INT], [F, N]);
  Return F^N;
EndDefine; -- Power
Power(x, 3);
x^3
-----
Power(2, 3);
8
-----
Power(2, x);
ERROR: Power: arg 2 is POLY but must be INT
CONTEXT: Return(S)
-----

```

I-3.11 *ciao*

syntax

```
ciao
```

Description

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

See Also: [quit\(I-17.2 pg.205\)](#)

I-3.12 *ClearDenom*

syntax

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

Description

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

example

```

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

```

I-3.13 *close*

syntax

```
close(D: DEVICE)
```

Description

This function closes the device D.

example

```
D := OpenOFile("my-test"); -- open file for output from CoCoA
Print "test" On D; -- write to my-file
Close(D); -- close the file
Close(DEV.STDIN); -- close the standard input device
-- Bye

(Close(DEV.OUT) suppresses all output to the CoCoA window.)
```

See Also: Introduction to IO([II-6.1](#) pg.291)

I-3.14 CloseLog

syntax

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

Description

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

This function “OpenLog” ([I-15.4](#) pg.186) 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.186)

I-3.15 CoCoA-4 mode

syntax

```
*** E ***

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

Description

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

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

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

example

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

See Also: updating CoCoA-4 code([I-21.5](#) pg.262), not([I-14.22](#) pg.179), and([I-1.10](#) pg.27), or([I-15.9](#) pg.189)

I-3.16 CocoaLimits

syntax

```
CocoaLimits(): RECORD
```

Description

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

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

example

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

I-3.17 CocoaPackagePath

syntax

```
CocoaPackagePath(): STRING
```

Description

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

example

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

I-3.18 CoeffEmbeddingHom

syntax

```
CoeffEmbeddingHom(P: RING: RINGELEM
```

Description

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

example

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

See Also: CanonicalHom(I-3.2 pg.37)

I-3.19 coefficients

syntax

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

Description

This function returns a list of coefficients of F in “`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 it returns the coefficients of the list of specified terms S ; their order is determined by the list S .

The old form (CoCoA-4) “`Coefficients(F,x)`” for the coefficients of F w.r.t an indeterminate x is now implemented as “`CoefficientsWRT`” (I-3.20 pg.44) and “`CoeffListWRT`” (I-3.21 pg.45).

example

```

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

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

/**/ L := CoefficientsWRT(F,[x,y,z]); indent(L); -- similar function
[
  record[PP := y^3, coeff := 5],
  record[PP := x^2*y, coeff := 3],
  record[PP := x*y^5, coeff := -1]
]
/**/ F = sum([X.coeff * X.PP | X In L]);
true

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

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

```

See Also: Coefficient Rings(III-9.3 pg.328), CoefficientsWRT(I-3.20 pg.44), CoeffListWRT(I-3.21 pg.45), LC(I-12.3 pg.148), monomials(I-13.25 pg.169), support(I-19.37 pg.248)

I-3.20 CoefficientsWRT

syntax

```

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

```

Description

This function returns the list of the coefficients and PPs of F seen as a polynomial in X , and indeterminate or a list of indeterminates. All entries in the returned list are `RingElem` in `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 := 1, coeff := y +2*z],
  record[PP := x, coeff := y +z],
  record[PP := x^3, coeff := 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 := 1, coeff := y],
  record[PP := z, coeff := 2],
  record[PP := x, coeff := y],
  record[PP := x*z, coeff := 1],
  record[PP := x^3*z, coeff := 1]
]

```

See Also: [Coefficient Rings](#)(III-9.3 pg.328), [CoeffOfTerm](#)(I-3.22 pg.46), [ContentWRT](#)(I-3.41 pg.54), [LC](#)(I-12.3 pg.148), [monomials](#)(I-13.25 pg.169), [support](#)(I-19.37 pg.248)

I-3.21 *CoeffListWRT*

syntax

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

Description

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

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

example

```

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

```

See Also: [coefficients](#)(I-3.19 pg.43), [CoefficientsWRT](#)(I-3.20 pg.44)

I-3.22 CoeffOfTerm

syntax

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

Description

This function returns the coefficient of the term “T” occurring in “F”. NB: 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\(I-3.19 pg.43\)](#), [LC\(I-12.3 pg.148\)](#), [log\(I-12.17 pg.155\)](#), [LogToTerm\(I-12.19 pg.155\)](#), [monomials\(I-13.25 pg.169\)](#), [support\(I-19.37 pg.248\)](#)

I-3.23 CoeffRing

syntax

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

Description

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

example

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

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

See Also: [characteristic\(I-3.8 pg.39\)](#), [coefficients\(I-3.19 pg.43\)](#), [CurrentRing\(I-3.47 pg.56\)](#), [indets\(I-9.19 pg.120\)](#)

I-3.24 ColMat

syntax

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

Description

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

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\(I-13.8 pg.162\)](#), [BlockMat\(I-2.6 pg.33\)](#), [DiagMat\(I-4.15 pg.66\)](#), [RowMat\(I-18.39 pg.227\)](#), [GensAsCols](#), [GensAsRows\(I-7.9 pg.95\)](#)

I-3.25 colon

syntax

```

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

```

Description

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

See also “HColon” ([I-8.1 pg.101](#)) for non-homogeneous input.

example

```

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

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

```

See Also: [saturate\(I-19.1 pg.229\)](#), [HSaturation\(I-8.12 pg.107\)](#), [HColon\(I-8.1 pg.101\)](#)

I-3.26 ColumnVectors

syntax

```
[OBSOLETE]
```

Description

Essentially replaced by “[GensAsCols](#), [GensAsRows](#)” ([I-7.9 pg.95](#)) and “[SubmoduleCols](#), [SubmoduleRows](#)” ([I-19.33 pg.245](#)) **See Also:** [GensAsCols](#), [GensAsRows\(I-7.9 pg.95\)](#), [SubmoduleCols](#), [SubmoduleRows\(I-19.33 pg.245\)](#)

I-3.27 Comp

syntax

[OBSOLETE]

Description

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

See Also: operators, shortcuts(I-0.1 pg.21), record field selector(I-18.13 pg.215)

I-3.28 Comparison Operators

syntax

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

Description

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

example

```
/**/ "abc" < "def"; -- lex ordering for strings
true
```

See Also: Equality Test(I-5.6 pg.73), operators, shortcuts(I-0.1 pg.21)

I-3.29 CompleteToOrd

syntax

```
CompleteToOrd(M: MAT): MAT
CompleteToOrd(M1, M2: MAT): MAT
```

Description

This function returns an ordering matrix (i.e. with non-zero determinant) completing the first matrix.

Given two matrices M1 and M2, it just concatenates of M1 and M2 and makes it square removing the redundant rows.

Given only one matrix M it completes M to an ordering matrix; if M is suitable the resulting matrix defines a term-ordering.

example

```
/**/ M := matrix([[1,2,3,4]]);
/**/ CompleteToOrd(M);
[[1, 2, 3, 4],
 [0, 0, 0, -1],
 [0, 0, -1, 0],
 [0, -1, 0, 0]]
```

```

/**/ CompleteToOrd(M, LexMat(4));
matrix(QQ,
  [[1, 2, 3, 4],
   [1, 0, 0, 0],
   [0, 1, 0, 0],
   [0, 0, 1, 0]])

/**/ CompleteToOrd(matrix([[1,2,0,0]]));
[[1, 2, 0, 0],
 [0, 0, 1, 1],
 [0, 0, 0, -1],
 [0, -1, 0, 0]]

/**/ CompleteToOrd(matrix([[1,2,0,0],[0,0,3,0]]));
matrix(QQ,
  [[1, 2, 0, 0],
   [0, 0, 3, 0],
   [0, 0, 0, 1],
   [0, -1, 0, 0]])

/**/ CompleteToOrd(matrix([[1,2,0,0],[0,0,3,0]]), RevLexMat(4));
matrix(QQ,
  [[1, 2, 0, 0],
   [0, 0, 3, 0],
   [0, 0, 0, -1],
   [0, -1, 0, 0]])
--> not a term-ordering

```

See Also: LexMat([I-12.6](#) pg.[149](#)), RevLexMat([I-18.29](#) pg.[222](#)), StdDegLexMat([I-19.26](#) pg.[242](#)), StdDegRevLexMat([I-19.27](#) pg.[243](#))

I-3.30 *compts*

— syntax —

```

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

```

Description

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

— example —

```

/**/ use R := QQ[x,y,z];
/**/ R3 := NewFreeModule(R,3);
/**/ V := ModuleElem(R3, ***[3x^2+4y, 2x-5z^3, 2x+2y]***); 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-14.26](#) pg.181)

I-3.31 concat

syntax

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

Description

This function returns the list obtained by concatenating the lists “L_1,...,L_n”.

NOTE: to concatenate strings just use “+”.

example

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

See Also: append([I-1.11](#) pg.27), ConcatLists([I-3.36](#) pg.52), String Operations([III-4.2](#) pg.315)

I-3.32 ConcatAntiDiag

syntax

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

Description

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

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

example

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

See Also: BlockMat([I-2.6](#) pg.33), ConcatDiag([I-3.33](#) pg.50), ConcatHor([I-3.34](#) pg.51), ConcatVer([I-3.37](#) pg.52)

I-3.33 ConcatDiag

syntax

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

Description

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

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

example

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

See Also: BlockMat(I-2.6 pg.33), ConcatAntiDiag(I-3.32 pg.50), ConcatHor(I-3.34 pg.51), ConcatVer(I-3.37 pg.52), DiagMat(I-4.15 pg.66)

I-3.34 ConcatHor

syntax

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

where A and B have the same number of rows

Description

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

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

example

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

See Also: BlockMat(I-2.6 pg.33), MakeMatByRows, MakeMatByCols(I-13.2 pg.159), ConcatAntiDiag(I-3.32 pg.50), ConcatDiag(I-3.33 pg.50), ConcatHorList(I-3.35 pg.51), ConcatVer(I-3.37 pg.52), RowMat(I-18.39 pg.227)

I-3.35 ConcatHorList

syntax

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

where the matrices in L have the same number of rows

Description

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

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

example

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

See Also: [BlockMat\(I-2.6 pg.33\)](#), [MakeMatByRows](#), [MakeMatByCols\(I-13.2 pg.159\)](#), [ConcatAntiDiag\(I-3.32 pg.50\)](#), [ConcatDiag\(I-3.33 pg.50\)](#), [ConcatHor\(I-3.34 pg.51\)](#), [ConcatVerList\(I-3.38 pg.53\)](#), [RowMat\(I-18.39 pg.227\)](#)

I-3.36 ConcatLists

syntax

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

Description

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

example

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

See Also: [append\(I-1.11 pg.27\)](#), [concat\(I-3.31 pg.50\)](#)

I-3.37 ConcatVer

syntax

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

where A and B have the same number of columns

Description

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

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

example

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

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

See Also: BlockMat(I-2.6 pg.33), ColMat(I-3.24 pg.46), MakeMatByRows, MakeMatByCols(I-13.2 pg.159), ConcatAntiDiag(I-3.32 pg.50), ConcatDiag(I-3.33 pg.50), ConcatHor(I-3.34 pg.51), ConcatVerList(I-3.38 pg.53)

I-3.38 ConcatVerList

syntax

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

where the matrices in L have the same number of columns

Description

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

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

example

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

See Also: BlockMat(I-2.6 pg.33), ColMat(I-3.24 pg.46), MakeMatByRows, MakeMatByCols(I-13.2 pg.159), ConcatAntiDiag(I-3.32 pg.50), ConcatDiag(I-3.33 pg.50), ConcatVer(I-3.37 pg.52), ConcatHorList(I-3.35 pg.51)

I-3.39 content

syntax

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

Description

This function returns the content of F (i.e. a gcd of its coefficients).

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

example

```
/**/ Use R ::= QQ[x,y,z];
/**/ F := *** 1234x^3z + 3456xyz^3 + 5678y^2z ***;
/**/ content(F);
2
/**/ RingOf(It);
QQ
```

See Also: [ContentWRT\(I-3.41 pg.54\)](#), [coefficients\(I-3.19 pg.43\)](#)

I-3.40 ContentFreeFactor

syntax

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

Description

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

example

```
/**/ Use R := QQ[x,y,z];
/**/ F := 2*(x+1)*(y+2)*(x+y);
/**/ indent(ContentFreeFactor(F));
record[
  RemainingFactor := 2,
  factors := [y +2, x +1, x +y],
  multiplicities := [1, 1, 1]
]
```

See Also: [ContentWRT\(I-3.41 pg.54\)](#), [factor\(I-6.1 pg.79\)](#), [SqFreeFactor\(I-19.21 pg.239\)](#)

I-3.41 ContentWRT

syntax

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

Description

This function returns the content of F (i.e. a gcd of its coefficients) seen as a polynomial in X, indet or list of indeterminates. The returned value is a `RingElem` in `RingOf(F)`.

example

```
/**/ Use R := 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\(I-3.20 pg.44\)](#), [content\(I-3.39 pg.53\)](#), [monomials\(I-13.25 pg.169\)](#)

I-3.42 ContFrac

syntax

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

Description

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

example

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

See Also: CFAprox([I-3.6 pg.39](#)), CFAproximants([I-3.7 pg.39](#)), ContFracToRat([I-3.43 pg.55](#))

I-3.43 ContFracToRat

syntax

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

Description

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

example

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

See Also: ContFrac([I-3.42 pg.55](#)), CFAprox([I-3.6 pg.39](#)), CFAproximants([I-3.7 pg.39](#))

I-3.44 count

syntax

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

Description

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

example

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

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

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

See Also: distrib([I-4.19 pg.68](#)), len([I-12.5 pg.149](#))

I-3.45 CpuTime

syntax

```
CpuTime(): RAT
```

Description

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

The most common usage is with “TimeFrom” ([I-20.6 pg.255](#)) as shown in the example.

example

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

You can use “DecimalStr” ([I-4.3 pg.59](#)) to see the value of “CpuTime” in a more easily comprehensible form.

See Also: TimeFrom([I-20.6 pg.255](#)), DecimalStr([I-4.3 pg.59](#))

I-3.46 CRT

syntax

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

Description

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

example

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

See Also: RatReconstructByContFrac, RatReconstructByLattice([I-18.7 pg.212](#))

I-3.47 CurrentRing

syntax

```
CurrentRing
```

Description

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

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

example

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

```

/**/ CurrentRing;
RingDistrMPolyClean(FFp(3), 1)

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

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

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

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

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

```

See Also: [RingOf\(I-18.31 pg.224\)](#), [TopLevel\(I-20.9 pg.256\)](#)

I-3.48 *CurrentTypes*

— syntax —

`CurrentTypes(): LIST of TYPE`

Description

This function lists all CoCoA data types.

— example —

```

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

```

I-3.49 *cyclotomic*

— syntax —

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

Description

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

— example —

```

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

```


Chapter I-4

D

I-4.1 dashes

syntax

```
dashes()
```

Description

This function returns a string of dashes:

example

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

I-4.2 date

syntax

```
date() : INT
```

Description

This function returns the date.

Note that from version 5.0.4 the result is an INT and the date is in the form YYYYMMDD. See also “TimeOfDay” ([I-20.7 pg.255](#)).

example

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

See Also: TimeOfDay([I-20.7 pg.255](#))

I-4.3 DecimalStr

syntax

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



```

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

```

3. VARIABLE NUMBER OF PARAMETERS. It is also possible to have a variable number of parameters using the syntax

```
Define F(...) C EndDefine;
```

In this case the special variable “ARGV” will contain the list of the arguments passed to the function.

example

```

/**/ Define MySum(...)
/**/   If len(ARGV) = 0 Then Return 12345;
/**/   Else
/**/     ans := 0;
/**/     Foreach N In ARGV Do ans := ans+N; EndForeach;
/**/   EndIf;
/**/   Return ans;
/**/ EndDefine;

/**/ MySum(1,2,3,4,5);
15
/**/ MySum();
12345

```

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

See Also: [return\(I-18.27 pg.221\)](#), [TopLevel\(I-20.9 pg.256\)](#), [ref\(I-18.15 pg.216\)](#)

I-4.5 DefiningIdeal

syntax

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

Description

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

example

```

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

```

See Also: [NewQuotientRing\(I-14.9 pg.174\)](#)

I-4.6 deg

syntax

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

Description

The first form of this function returns the “*standard degree*” of “F” (see “wdeg” (I-23.1 pg.267) 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-13.27 pg.170).

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"], CompleteToOrd(Ws), 1);
/**/ Use P;
/**/ deg(x*y^2+y);
3
/**/ wdeg(x*y^2+y);
[8]
/**/ deg(x*y^2+y, x);
1
/**/ deg(x*y^2+y, x);
2

```

See Also: wdeg(I-23.1 pg.267), NewPolyRing(I-14.8 pg.174), multiplicity(I-13.27 pg.170)

I-4.7 den

syntax

```

den(N: INT or RAT): INT
den(N: RINGELEM): RINGELEM

```

Description

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.

NB In CoCoA 4 the numerator and denominator could also be found using “.Num” and “.Den”, but this fragile syntax is no longer supported.

example

```

/**/ den(3);
1

/**/ P ::= QQ[x,y];
/**/ F := NewFractionField(P);
/**/ Use F;
/**/ den(x/(x+y));
x +y
/**/ RingOf(It);
RingDistrMPolyClean(QQ, 2)

```

See Also: num(I-14.24 pg.180)

I-4.8 DensePoly

syntax

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

Description

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

example

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

/**/ Ws := RowMat([2,3]);
/**/ P := NewPolyRing(QQ, ["x","y"], CompleteToOrd(Ws), 1);
/**/ Use P;
/**/ DensePoly(P,1); // NB standard degree!!
y +x
```

See Also: [randomize\(I-18.4 pg.210\)](#), [randomized\(I-18.5 pg.211\)](#)

I-4.9 Depth

syntax

```
Depth(I: IDEAL, M: TAGGED("Quotient")): INT
Depth(M: TAGGED("Quotient")): INT
```

Description

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

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 I is not specified, it assumes that I is the maximal ideal generated by the indeterminates, i.e. “`ideal(Indets())`”.

Note that if M is homogeneous and I is the maximal ideal, then it uses the Auslander-Buchsbaum formula “`depth_I(M) = N - pd(M)`” where N is the number of indeterminates and pd is the projective dimension, otherwise it returns “`min{N | Ext^N(R/I, M) <> 0}`” using the function “Ext” ([I-5.11 pg.76](#)).

example

```
Use R := QQ[x,y,z];
Depth(R/ideal(0)); -- the (x,y,z)-depth of the entire ring is 3
3
-----
I := ideal(x^5,y^3,z^2);
-- one can check that it is zerodimensional and CM this way
dim(R/I);
0
-----
Depth(R/I);
0
-----

N := Module([x^2,y], [x+z,0]);
Depth(I, R^2/N); --- a max reg sequence would be (z^2,y^3)
2
```

```

-----
Use R := 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(R^2/N);
3
-----
dim(R^2/N);
3
-----
--- yes!

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

```

See Also: [res\(I-18.23 pg.220\)](#), [Ext\(I-5.11 pg.76\)](#)

I-4.10 deriv

syntax

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

Description

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

example

```

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

/**/ Define Jac(F) --> The Jacobian matrix for a polynomial.
/**/ Return matrix([[deriv(F, X) | X In Indets(RingOf(F))]]);
/**/ EndDefine;

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

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

```

See Also: `jacobian`([I-10.1](#) pg.143)

I-4.11 DerivationAction

syntax

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

Description

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-9.30](#) pg.126), `PerpIdealOfForm`([I-16.6](#) pg.192)

I-4.12 describe

syntax

```
describe X: OBJECT
```

Description

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

example

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

/**/ describe $chebyshev;
The package $chebyshev exports the following names:
* ChebyshevPoly
* ChebyshevPoly2
```

I-4.13 det

syntax

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

Description

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

example

```
/**/ Use R ::= QQ[x];
/**/ M := mat(R, [[x,x^2], [x,x^3]]);
/**/ det(M);
```

```
x^4 -x^3

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

See Also: minors([I-13.17](#) pg.166)

I-4.14 DF

— syntax —

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

Description

Same as “LF” ([I-12.8](#) pg.150), but does not throw an error if the argument is zero or if the “GradingDim” ([I-7.18](#) pg.99) 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-12.8](#) pg.150)

I-4.15 DiagMat

— syntax —

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

Description

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

— example —

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

```

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

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

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

```

See Also: [BlockMat\(I-2.6 pg.33\)](#), [IsDiagonal\(I-9.37 pg.128\)](#), [ColMat\(I-3.24 pg.46\)](#), [RowMat\(I-18.39 pg.227\)](#)

I-4.16 *diff*

— [syntax](#) —

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

Description

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

— [example](#) —

```

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

```

See Also: [insert\(I-9.24 pg.123\)](#), [remove\(I-18.21 pg.219\)](#)

I-4.17 *dim*

— [syntax](#) —

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

Description

This function computes the dimension of R.

The coefficient ring must be a field.

— [example](#) —

```

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

```

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

I-4.18 discriminant

syntax

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

Description

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

The discriminant is defined as

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

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

example

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

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

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

See Also: resultant(I-18.26 pg.221)

I-4.19 distrib

syntax

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

Description

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

example

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

See Also: count(I-3.44 pg.55)

I-4.20 div

syntax

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

Description

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

NOTE: To perform the division algorithm on a polynomial use “NR” (I-14.23 pg.180) (normal remainder) to find the remainder, or “DivAlg” (I-4.21 pg.69) to get both quotients and remainder. To determine if a polynomial is in a given ideal or a vector is in a given module, use “NF” (I-14.14 pg.176) or “IsIn” (I-9.45 pg.131), and to find a representation in terms of the generators “GenRepr” (I-7.7 pg.94).

— example —

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

See Also: DivAlg(I-4.21 pg.69), GenRepr(I-7.7 pg.94), NF(I-14.14 pg.176), NR(I-14.23 pg.180), mod(I-13.20 pg.167)

I-4.21 DivAlg

— syntax —

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

Description

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

— example —

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

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

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

See Also: [div\(I-4.20 pg.68\)](#), [mod\(I-13.20 pg.167\)](#), [GenRepr\(I-7.7 pg.94\)](#), [NF\(I-14.14 pg.176\)](#), [NR\(I-14.23 pg.180\)](#)

Chapter I-5

E

I-5.1 E_

syntax

OBSOLETE

Description

Essentially replaced by “gens” ([I-7.8 pg.94](#)) of a FreeModule.

example

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

See Also: gens([I-7.8 pg.94](#)), GensAsCols, GensAsRows([I-7.9 pg.95](#))

I-5.2 eigenvectors

syntax

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

Description

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

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

example

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

```

    [-1]]]),
record[MinPoly := x^2 -15*x -18,
eigenspace := [[1, (1/8)*x +1/4, (1/4)*x -1/2]]]
]

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

```

I-5.3 elim

syntax

```

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

```

Description

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

As opposed to this function, there is also the “*modifier*”, “**elim**”, used when constructing a ring (see “Orderings” (III-9.5 pg.329)).

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: Orderings(III-9.5 pg.329)

I-5.4 ElimMat

syntax

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

Description

This function returns an “NxN” matrix for a term ordering eliminating the indeterminates with indices in “ElimInd”. If a weight matrix is given, then these weights are included after the first elimination row.

example

```
/**/ ElimMat(3, [2,3]);
matrix(QQ,
  [[0, 1, 1],
   [1, 1, 1],
   [0, 0, -1]])

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

See Also: [elim\(I-5.3 pg.72\)](#), [HomogElimMat\(I-8.11 pg.107\)](#)

I-5.5 EqSet

syntax

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

Description

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

example

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

See Also: [intersection\(I-9.27 pg.124\)](#), [IntersectList\(I-9.28 pg.125\)](#), [IsSubset\(I-9.60 pg.137\)](#)

I-5.6 Equality Test

syntax

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

Description

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

example

```

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

```

See Also: Comparison Operators([I-3.28](#) pg.48), operators, shortcuts([I-0.1](#) pg.21)

I-5.7 EquiIsoDec

syntax

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

Description

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

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

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

example

```

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

```

See Also: PrimaryDecomposition([I-16.18](#) pg.198), radical([I-18.1](#) pg.209), RadicalOfUnmixed([I-18.2](#) pg.209)

I-5.8 error

syntax

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

Description

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

example

```

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

```

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

/**/ T(7);
2
```

See Also: [try\(I-20.13 pg.259\)](#), [GetErrMsg\(I-7.14 pg.97\)](#)

I-5.9 eval

syntax

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

Description

This function substitutes “L[I]” for “[indet\(I\)](#)” in the expression E which must be of type POLY, MODULEELEM, LIST, or MAT, and defined in the current ring. For more general substitutions use “[subst](#)” ([I-19.35 pg.246](#)).

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

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\(III-11.2 pg.336\)](#), [image\(I-9.9 pg.115\)](#), [subst\(I-19.35 pg.246\)](#)

I-5.10 EvalHilbertFn

syntax

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

Description

This function evaluates the Hilbert function H at N . If H is the Hilbert function of a quotient R/I , then the value returned is the same as that returned by “`hilbert(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 := Hilbert(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: `hilbert`(I-8.3 pg.102), `HilbertFn`(I-8.5 pg.103), `HilbertPoly`(I-8.6 pg.103)

I-5.11 Ext

syntax

```

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

```

Description

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

-----
```

Since version 4.7.3 the output modules are presented minimally.

See Also: [res\(I-18.23 pg.220\)](#), [Depth\(I-4.9 pg.63\)](#), [MinimalPresentation\(I-13.16 pg.166\)](#)

Chapter I-6

F

I-6.1 factor

syntax

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

Description

This function factorizes a polynomial into irreducibles in its ring of definition. Multivariate factorization is not yet supported over finite fields. To factorize an integer use “SmoothFactor” ([I-19.12 pg.235](#)). (For information about the algorithm, consult John Abbott’s papers)

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

example

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

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

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

See Also: SmoothFactor([I-19.12 pg.235](#)), SqFreeFactor([I-19.21 pg.239](#)), ContentFreeFactor([I-3.40 pg.54](#))

I-6.2 factorial

syntax

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



```
Use QQ[x, y, z], Ord(M);
-- New basis (Lex)
BringIn(GBNew);
```

I-6.5 fields

syntax

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

Description

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

example

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

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

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

See Also: [record\(I-18.12 pg.215\)](#)

I-6.6 first

syntax

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

Description

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

example

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

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

See Also: [last\(I-12.1 pg.147\)](#)

I-6.7 FirstNonZero

syntax

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

Description

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

example

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

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

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

See Also: `FirstNonZeroPosn`([I-6.8](#) pg.82), `IsZero`([I-9.66](#) pg.140), `NonZero`([I-14.21](#) pg.179)

I-6.8 FirstNonZeroPosn

syntax

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

Description

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

example

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

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

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

See Also: `FirstNonZero`([I-6.7](#) pg.81), `IsZero`([I-9.66](#) pg.140), `NonZero`([I-14.21](#) pg.179)

I-6.9 flatten

syntax

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

Description

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

example

```

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

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

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

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

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

```

I-6.10 FloatApprox

syntax

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

Description

This function computes an approximation of the form $M * 2^E$ to a rational number “X” where the mantissa satisfies $2^{(B-1)} \leq |M| < 2^B - 1$ where B 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 “ILogBase” (I-9.8 pg.114)). 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-3.6 pg.39), FloatStr(I-6.11 pg.83), MantissaAndExponent2(I-13.5 pg.160)

I-6.11 FloatStr

syntax

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

Description

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

The aim is to produce an easily readable result.

example

```

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

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

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

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

```

See Also: `DecimalStr`([I-4.3](#) pg.59), `ScientificStr`([I-19.3](#) pg.230), `FloatApprox`([I-6.10](#) pg.83), `MantissaAndExponent10`([I-13.4](#) pg.160)

I-6.12 floor

syntax

```

floor(X: RAT): INT

```

Description

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

example

```

/**/ floor(0.99);
0

/**/ floor(1.01);
1

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

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

```

See Also: `ceil`([I-3.5](#) pg.38), `round`([I-18.38](#) pg.227), `num`([I-14.24](#) pg.180), `den`([I-4.7](#) pg.62)

I-6.13 for

syntax

```

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

```

where I is a dummy variable, N_1, N_2, and D are integer expressions,
and C is a sequence of commands.

Description

In the first form, the variable I is assigned the values “N_1, N_1+1, ..., N_2” in succession. After each assignment, the command sequence C is executed. The second form is the same, except that I is assigned

the values “N₁, N₁+D, N₁+2D”, etc. until the greatest value less than or equal to “N₂” is reached. If “N₂ < N₁”, then C is not executed.

NOTE: Large values for “N₁, N₂”, or D are not permitted; typically they should lie in the range about -10^9 to $+10^9$.

NOTE: Don’t forget the capitalization in the word “To”.

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
/**/     M := I;
/**/     For J := I+1 To len(L) Do
/**/       If L[J] < L[M] Then M := J; EndIf;
/**/     EndFor;
/**/     If M <> I Then
/**/       C := L[M];
/**/       L[M] := L[I];
/**/       L[I] := C;
/**/     EndIf;
/**/   EndFor;
/**/ EndDefine;

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

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

See Also: [foreach\(I-6.14 pg.85\)](#), [repeat\(I-18.22 pg.219\)](#), [while\(I-23.3 pg.268\)](#)

I-6.14 *foreach*

syntax

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

Description

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

example

```

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

/**/  Use R ::= QQ[x,y,z];
/**/  F := x^2*y + 3*y^2*z - z^3;
/**/  J := [deriv(F, X) | X In indets(R)];
/**/  J;
[2*x*y, x^2 +6*y*z, 3*y^2 -3*z^2]

/**/  Foreach X In J Do
/**/    PrintLn X^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\(I-6.13 pg.84\)](#), [repeat\(I-18.22 pg.219\)](#), [while\(I-23.3 pg.268\)](#)

I-6.15 format

syntax

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

Description

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

example

```

/**/  L := [5^n | n In 0..7];
/**/  Foreach F In L Do print Format(F,8); EndForeach;
      1      5      25      125      625      3125      15625      78125

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

```

See Also: [IO.SprintTrunc\(I-9.31 pg.126\)](#), [Latex\(I-12.2 pg.147\)](#), [sprint\(I-19.20 pg.239\)](#)

I-6.16 FrbAlexanderDual

syntax

```

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

```

Description

Using the “Frobby” ([II-8.3 pg.303](#)) library linked with CoCoALib.

example

```

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

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

```

See Also: Frobyy([II-8.3](#) pg.303), FrbPrimaryDecomposition([I-6.20](#) pg.88), PrimaryDecomposition([I-16.18](#) pg.198)

I-6.17 FrbAssociatedPrimes

syntax

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

Description

Using the “Frobyy” ([II-8.3](#) pg.303) C++ library by Bjarke Roune.

example

```

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

```

See Also: Frobyy([II-8.3](#) pg.303), FrbIrreducibleDecomposition([I-6.18](#) pg.87), FrbPrimaryDecomposition([I-6.20](#) pg.88)

I-6.18 FrbIrreducibleDecomposition

syntax

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

Description

Using the “Frobyy” ([II-8.3](#) pg.303) C++ library by Bjarke Roune.

example

```

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

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

```

See Also: Frobyy([II-8.3](#) pg.303), FrbAssociatedPrimes([I-6.17](#) pg.87), FrbIrreducibleDecomposition([I-6.18](#) pg.87)

I-6.19 FrbMaximalStandardMonomials

syntax

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

Description

Using the “Frobby” ([II-8.3 pg.303](#)) library linked with CoCoALib.

example

```
/**/ I := ideal(x^2, x*y, y^2, z^2);
/**/ FrbMaximalStandardMonomials(I);
ideal(y*z, x*z)
```

See Also: Frobby([II-8.3 pg.303](#))

I-6.20 FrbPrimaryDecomposition

syntax

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

Description

Using the “Frobby” ([II-8.3 pg.303](#)) 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-8.3 pg.303](#)), FrbAssociatedPrimes([I-6.17 pg.87](#)), FrbIrreducibleDecomposition([I-6.18 pg.87](#)), PrimaryDecomposition([I-16.18 pg.198](#))

I-6.21 func

syntax

```
Func ... EndFunc
  returns FUNCTION
```

Description

This syntactic structure defines a function without giving it a name; anonymous functions can be passed as parameters and assigned to variables. Note that “Func...EndFunc” can be used inside function definitions.

example

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

/**/ SortedBy(["zzz", "x", "yy"], Func(x,y) Return len(x)>len(y); EndFunc);
["zzz", "yy", "x"]
```

See Also: define([I-4.4 pg.60](#)), TopLevel([I-20.9 pg.256](#)), ImportByRef, ImportByValue([I-9.12 pg.116](#))

I-6.22 Function

— syntax —

[OBSOLETE]

Description

[OBSOLETE] In CoCoA-5 functions are "first class objects", and so may be passed like any other value – the operator "Function" serves no purpose. In CoCoA-4 it was possible to have a variable and a function with the same name; the operator "Function" was used to instruct CoCoA-4 to search for the function of the given name, e.g. to pass it as an argument to another function.

See Also: FUNCTIONS are first class objects([III-7.2](#) pg.323), SortBy([I-19.14](#) pg.236), SortedBy([I-19.16](#) pg.237)

I-6.23 functions

— syntax —

[OBSOLETE]

Description

This function is OBSOLETE; please use the command "describe" ([I-4.12](#) pg.65).

Chapter I-7

G

I-7.1 GBasis

syntax

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

Description

This function returns a list whose components form a Groebner basis for the ideal (or module) “I” with respect to the term-ordering of the polynomial ring of “I”.

If “I” is a variable then the result is stored in “I” for later use.

For the reduced Groebner basis, use the command “ReducedGBasis” ([I-18.14](#) pg.216).

The coefficient ring must be a field.

example

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

See Also: [GBasisTimeout\(I-7.2](#) pg.91)

I-7.2 GBasisTimeout

syntax

```
GBasisTimeout(I: IDEAL, SECONDS: INT): LIST
GBasisTimeout(M: MODULE, SECONDS: INT): LIST
```

Description

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

Same as “GBasis” ([I-7.1](#) pg.91), but it will stop and return an error if the computation is not completed.

For dealing with errors see “try” ([I-20.13](#) pg.259).

example

```
Use R ::= QQ[t,x,y,z];
I := ideal(t^3-x, t^4-y, t^5-z);
J := I^5; Time G := GBasisTimeout(J, 1);
ERROR: Time expired: use $gb.Complete to complete the computation
```

```
CONTEXT: Error(GBasisTimeout_Err)
-----
J := I^5; Time G := GBasisTimeout(J, 10);
Cpu time = 1.96, User time = 2
-----
```

See Also: [GBasis\(I-7.1 pg.91\)](#), [try\(I-20.13 pg.259\)](#)

I-7.3 GBM

syntax

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

Description

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

This function computes the intersection of ideals corresponding to zero-dimensional schemes: GBM is for affine schemes, and “HGBM” ([I-8.2 pg.101](#)) for projective schemes. The list L must be a list of ideals. The function “IntersectList” ([I-9.28 pg.125](#)) 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\(I-9.2 pg.109\)](#), [IdealAndSeparatorsOfProjectivePoints\(I-9.3 pg.110\)](#), [IdealOfPoints\(I-9.4 pg.112\)](#), [IdealOfProjectivePoints\(I-9.5 pg.113\)](#), [HGBM\(I-8.2 pg.101\)](#)

I-7.4 gcd

syntax

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

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

Description

This function returns the greatest common divisor of “F₁, . . . , F_n” or of the elements in the list L. For the calculation of the GCDs and LCMs of polynomials, the coefficient ring must be a field.

example

```

/**/ Use R ::= QQ[x,y];
/**/ F := x^2-y^2;
/**/ G := (x+y)^3;
/**/ gcd(F, G);
-x -y

/**/ gcd([3*4,3*8,6*16]);
12

```

See Also: [div\(I-4.20 pg.68\)](#), [mod\(I-13.20 pg.167\)](#), [lcm\(I-12.4 pg.148\)](#)

I-7.5 GCDFreeBasis

syntax

```
GCDFreeBasis(L: LIST of INT): LIST of INT
```

Description

This function returns a GCD free basis of a set of integers; you can think of this as the set of all numbers (except 1) obtainable by performing GCD and exact division operations.

Given a set $N = [N_1, \dots, N_k]$ we seek a basis $G = [G_1, \dots, G_s]$ such that each N_i is a product of powers of the G_j , and the G_j are pairwise coprime; the set G is called a GCD free basis for N . In general the set G is not uniquely defined.

example

```

/**/ GCDFreeBasis([factorial(20), factorial(10)]);
[46189, 4, 14175]

```

See Also: [gcd\(I-7.4 pg.92\)](#)

I-7.6 GenericPoints

syntax

```

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

```

Description

“GenericPoints” returns a list of NumPoints generic projective points with integer coordinates; it is not guaranteed that these points are distinct. RandomRange specifies the largest value any coordinate may take. If the second argument is omitted, the largest value possible is 100 (or P-1 where P is the characteristic of the coefficient ring).

example

```

/**/ Use R ::= QQ[x,y]; GenericPoints(R,7);
[[1, 0], [0, 1], [1, 1], [12, 59], [6, 63], [12, 80], [17, 63]]

/**/ GenericPoints(R,7,500);
[[1, 0], [0, 1], [1, 1], [220, 162], [206, 452], [98, 106], [403, 449]]

/**/ Use R ::= ZZ/(5)[x,y,z];
/**/ GenericPoints(R,7);
[[1, 0, 0], [0, 1, 0], [0, 0, 1], [1, 1, 1], [2, 1, 1], [2, 2, 4], [3, 1, 3]]

```

```

/**/ GenericPoints(R,7,500);
[[1, 0, 0], [0, 1, 0], [0, 0, 1], [1, 1, 1], [1, 4, 2], [1, 3, 2], [2, 3, 3]]

```

I-7.7 GenRepr

syntax

```

GenRepr(X: RINGELEM, I: IDEAL): LIST of RINGELEM
GenRepr(X: MODULEELEM, I: MODULE): LIST of RINGELEM

```

Description

This function returns a list giving a representation of X in terms of generators for I . Let the generators for I be “ $[G_1, \dots, G_t]$ ”. If X is in I , then “GenRepr” will return a list “ $[F_1, \dots, F_t]$ ” such that

$$X = F_1 G_1 + \dots + F_t G_t.$$

If X is not in I , then “GenRepr” returns the empty list, $[]$.

example

```

/**/ Use R ::= QQ[x,y];
/**/ I := ideal(x+y^2, x^2-x*y);
/**/ GenRepr(x^3-x^2*y-y^3-x*y, I);
[-y, x]
/**/ -y*gens(I)[1] + x*gens(I)[2];
x^3 -x^2*y -y^3 -x*y
/**/ GenRepr(x+y, I); -- x+y is not in I
[]

/**/ K := NewFractionField(NewPolyRing(QQ, ["a"]));
/**/ Use R ::= K[x,y];
/**/ L := [x+y^2, x^2-x*y];
/**/ GenRepr((a-2)*L[1] - (x-a)*L[2], ideal(L));
[a -2, -x +a]

/**/ R3 := NewFreeModule(R,3);
/**/ V1 := ModuleElem(R3, [x, y, y^2]);
/**/ V2 := ModuleElem(R3, [x-y, 0, x^2]);
/**/ V := x^2*V1 - y^2*V2;
/**/ M := submodule(R3, [V1, V2]);
--/**/ GenRepr(V, M); -- NOT YET IMPLEMENTED *****
--[x^2, -y^2]

```

See Also: DivAlg(I-4.21 pg.69), IsIn(I-9.45 pg.131), NF(I-14.14 pg.176), syz(I-19.41 pg.249), SyzOfGens(I-19.42 pg.250)

I-7.8 gens

syntax

```

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

```

Description

This function returns a list of polynomials which generate the ideal I or the module M. The list is not necessarily minimal.

example

```

/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(y^2-x^3, x*y);
/**/ gens(I);
[-x^3 +y^2, x*y]

/**/ gens(I^2);
[x^6 -2x^3*y^2 +y^4, -x^4*y +x*y^3, x^2*y^2]

/**/ R3 := NewFreeModule(R, 3);
/**/ e := gens(R3); // canonical basis
/**/ e[2];
[0, 1, 0]

/**/ M := SubmoduleRows(R3, mat([[x,y,z], [x-1,0,z]]));
/**/ gens(M);
[[x, y, z], [x -1, 0, z]]
/**/ shape(It);
[MODULEELEM, MODULEELEM]
/**/ GensAsRows(M);
matrix( /*RingDistrMPolyClean(QQ, 3)*/
  [[x, y, z],
   [x -1, 0, z]])

```

See Also: GensAsCols, GensAsRows([I-7.9 pg.95](#)), minimalize([I-13.14 pg.165](#)), minimalized([I-13.15 pg.165](#))

I-7.9 GensAsCols, GensAsRows

syntax

```

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

```

Description

These functions returns a matrix which generate the module M with the components as row (or columns) of a matrix.

The generators are not necessarily minimal.

example

```

/**/ Use R ::= QQ[x,y,z];
/**/ R3 := NewFreeModule(R, 3);
/**/ L := [[x,y,z], [x-1,0,z]];
/**/ M := SubmoduleRows(R3, mat(L));
/**/ gens(M);
[[x, y, z], [x -1, 0, z]]
/**/ shape(It);
[MODULEELEM, MODULEELEM]

/**/ GensAsRows(M);
matrix( /*RingDistrMPolyClean(QQ, 3)*/
  [[x, y, z],
   [x -1, 0, z]])

```

```

/**/ GensAsCols(M);
matrix( /*RingDistrMPolyClean(QQ, 3)*/
  [[x, x -1],
   [y, 0],
   [z, z]])

```

See Also: gens(I-7.8 pg.94), SubmoduleCols, SubmoduleRows(I-19.33 pg.245)

I-7.10 Get

syntax

```
Get(D: DEVICE, N: INT): LIST of INT
```

Description

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

This function reads N characters from D and returns the list of their ASCII codes.

example

```

D := OpenIFile("io.cpkg"); -- open the file "io.cpkg"
Get(D,10); -- get the first 10 characters
[45, 45, 32, 105, 111, 100, 101, 118, 46, 112]
-----
ascii(It); convert the ASCII code to characters
-- iodev.p
-----
ascii(Get(D,10)); -- get the next 10 characters and convert
kg : 0.1 :
-----
Close(D);

```

The instruction “Get(DEV.STDIN,3)”, for instance, will read 3 characters typed in by the user. Clever use of this function can be used to prompt a user for input to a function, although it is usually easier for functions to take input directly as arguments. NOTE: this function does not work properly under the GUI Interface.

See Also: Introduction to IO(II-6.1 pg.291), OpenIFile(I-15.2 pg.185), OpenOFile(I-15.5 pg.187), OpenIString(I-15.3 pg.186), OpenOString(I-15.6 pg.188)

I-7.11 GetCol

syntax

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

Description

This function makes a list containing the entries of the “K”-th column of “M”.

example

```

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

```

See Also: GetRow(I-7.15 pg.98), GetCols(I-7.12 pg.97)

I-7.12 GetCols

— syntax —

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

Description

This function produces a list of lists containing the columns of “M”.

— example —

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

See Also: GetCol([I-7.11](#) pg.96), GetRows([I-7.16](#) pg.98)

I-7.13 GetEnv

— syntax —

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

Description

This function returns the value of system shell variables

— example —

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

/**/ GetEnv("COCOARC");
/Users/bigatti/.cocoarc

/**/ GetEnv("COCOA_PACKAGES");
/Applications/CoCoA-4.7/packages
```

I-7.14 GetErrMsg

— syntax —

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

Description

This function returns the string containing the error message associated with an error.

— example —

```
/**/ ErrMsg := "";

Try
  F := 1/0;
UponError E Do
  ErrMsg := GetErrMsg(E);
EndTry; -- no error is thrown with Try .. UponError .. EndTry
```

```
/**/ ErrMsg;
Division by zero or by a zero-divisor
```

See Also: [try\(I-20.13 pg.259\)](#), [error\(I-5.8 pg.74\)](#)

I-7.15 GetRow

syntax

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

Description

This function makes a list containing the entries of the “K”-th row of “M”.

example

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

See Also: [GetRows\(I-7.16 pg.98\)](#)

I-7.16 GetRows

syntax

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

Description

This function produces a list of lists containing the rows of “M”.

example

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

See Also: [GetRow\(I-7.15 pg.98\)](#)

I-7.17 Gin, Gin5

syntax

```
Gin(I: IDEAL): IDEAL
Gin(I: IDEAL, Range: INT): IDEAL
Gin5(I: IDEAL): IDEAL
Gin5(I: IDEAL, Range: INT): IDEAL
```

Description

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

These functions return the [probabilistic] gin (generic initial ideal) of the ideal I. It is obtained by computing the leading term ideal of $g(I)$, where g is a random change of coordinates.

While “Gin” uses integer coefficients in $[-\text{Range}, \text{Range}]$, with default value $[-100, 100]$ (repeated until 4 consecutive random changes of coordinates give the same result), the function “Gin5” uses the special TwinFloat implementation to allow a much wider range of coefficients (and then performs the computation only twice). The latter is faster, but needs you to start the server!

example

```
Use R := QQ[x,y,z], DegRevLex;
Gin(ideal(y^2-xz, x^2z-yz^2));
ideal(x^2, xy^2, y^4)
-----
Use R := QQ[x,y,z], Lex;
Gin(ideal(y^2-xz, x^2z-yz^2), 10); -- coeffs in range [-10, 10]
ideal(x^2, xy^2, xyz^2, xz^4, y^6)
-----
Use R := QQ[x,y,z], DegRevLex;
Gin5(ideal(y^2-xz, x^2z-yz^2)); -- default coeff range [-10000, 10000]
ideal(x^2, xy^2, y^4)
-----
Use R := QQ[x,y,z], Lex;
Gin5(ideal(y^2-xz, x^2z-yz^2), 2); -- choose dangerously small range [-2,2]:
-- ==> answer might be wrong
```

I-7.18 GradingDim

syntax

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

Description

This function returns the grading dimension of a polynomial ring, i.e. how many of the rows of OrderMatrix are to be taken as specifying the grading.

example

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

See Also: [NewPolyRing\(I-14.8 pg.174\)](#), [WeightsMatrix\(I-23.2 pg.267\)](#)

Chapter I-8

H

I-8.1 HColon

syntax

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

Description

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

The function “colon” ([I-3.25 pg.47](#)) 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-3.25 pg.47](#)) only when the input is non-homogeneous, in which case, “HColon” may be faster.

example

```
Use R := QQ[x,y];
ideal(xy, x^2) : ideal(x);
ideal(y, x)
-----
colon(ideal(x^2, xy), ideal(x, x-y^2));
ideal(x)
-----
HColon(ideal(x^2, xy), ideal(x, x-y^2));
ideal(x)
-----
```

See Also: [HSaturation\(I-8.12 pg.107\)](#), [saturate\(I-19.1 pg.229\)](#), [HColon\(I-8.1 pg.101\)](#), [colon\(I-3.25 pg.47\)](#)

I-8.2 HGBM

syntax

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

Description

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

This function computes the intersection of ideals corresponding to zero-dimensional schemes: “GBM” ([I-7.3 pg.92](#)) is for affine schemes, and HGBM for projective schemes. The list L must be a list of ideals. The function “IntersectList” ([I-9.28 pg.125](#)) should be used for computing the intersection of a collection of general ideals.

The name GBM comes from the name of the algorithm used: Generalized Buchberger-Moeller. The prefix H comes from Homogeneous since ideals of projective schemes are necessarily homogeneous.

example

```
Use QQ[x[0..2]];
I1 := IdealOfProjectivePoints([[1,2,1], [0,1,0]]); -- simple projective scheme
I2 := IdealOfProjectivePoints([[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\(I-9.2 pg.109\)](#), [IdealAndSeparatorsOfProjectivePoints\(I-9.3 pg.110\)](#), [IdealOfPoints\(I-9.4 pg.112\)](#), [IdealOfProjectivePoints\(I-9.5 pg.113\)](#), [GBM\(I-7.3 pg.92\)](#)

I-8.3 hilbert

syntax

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

Description

(sorry Hilbert for the lower-case: here we follow the naming convention “*single name goes lower-case*”)

This function is the same as “HilbertFn” ([I-8.5 pg.103](#)).

See Also: [HilbertFn\(I-8.5 pg.103\)](#)

I-8.4 HilbertBasisKer

syntax

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

where M is a matrix over ZZ.

Description

This function returns a list whose components are lists (of non-negative integers) representing the Hilbert basis for the monoid of elements with non-negative coordinates in the kernel of M.

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([
  [0, 0, 0],
```

```
[0, 0, 0]
])
```

See Also: LinKerBasis([I-12.11](#) pg.152)

I-8.5 HilbertFn

syntax

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

Description

The first form of this function computes the Hilbert function for R. The second form computes the N-th value of the Hilbert function. The weights of the indeterminates of R must all be 1. If the input is not homogeneous, the Hilbert function of the corresponding leading term (initial) ideal or module is calculated. For repeated evaluations of the Hilbert function, use “EvalHilbertFn” ([I-5.10](#) pg.75) instead of “Hilbert(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-5.10](#) pg.75), HilbertPoly([I-8.6](#) pg.103), HVector([I-8.13](#) pg.108), HilbertSeries([I-8.7](#) pg.104)

I-8.6 HilbertPoly

syntax

```
HilbertPoly(R: RING or TAGGED("Quotient")): RINGELEM in the ring QQt.
```

Description

This function returns the Hilbert polynomial for R as a polynomial in the standard CoCoA ring $QQ_t (= QQ[t])$.

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 “[hilbert](#)” ([I-8.3](#) [pg.102](#)).

example

```

/**/ Use R ::= QQ[w,x,y,z];
/**/ I := ideal(z^2-x*y, x*z^2+w^3);
/**/ Hilbert(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-5.10](#) [pg.75](#)), [hilbert\(I-8.3](#) [pg.102](#)), [HVector\(I-8.13](#) [pg.108](#)), [HilbertSeries\(I-8.7](#) [pg.104](#))

I-8.7 HilbertSeries

syntax

```

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

```

Description

This function computes the Hilbert-Poincare series of M . The input, M , must be homogeneous (with respect to the first row of the weights matrix). In the standard case, i.e. the weights of all indeterminates are 1, the result is simplified so that the power appearing in the denominator is the dimension of M .

The function “[poincare](#)” ([I-16.11](#) [pg.195](#)) is exactly the same as “[HilbertSeries](#)” ([I-8.7](#) [pg.104](#)).

NOTES:

- (i) the coefficient ring must be a field.
- (ii) these functions produce tagged objects: they cannot safely be (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"],CompleteToOrd(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"],CompleteToOrd(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"], CompleteToOrd(Ws), 2);
/**/ Use P;
/**/ Poincare(P/ideal(t^2, y^3));
--- Non-simplified HilbertPoincare' Series ---
(1 - t[1]^2 - t[1]^9*t[2]^15 + t[1]^11*t[2]^15) /
((1-t[1])^1*(1-t[1]^2)*(1-t[1]^3*t[2]^5)*(1-t[1]^4*t[2]^8) )

```

See Also: [dim\(I-4.17 pg.67\)](#), [multiplicity\(I-13.27 pg.170\)](#), [HilbertFn\(I-8.5 pg.103\)](#), [HVector\(I-8.13 pg.108\)](#), [HilbertSeriesShifts\(I-8.9 pg.106\)](#), [HilbertSeriesMultiDeg\(I-8.8 pg.105\)](#), [WeightsMatrix\(I-23.2 pg.267\)](#)

I-8.8 HilbertSeriesMultiDeg

syntax

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

Description

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

This function computes the Hilbert-Poincare series of M. The input, M, must be homogeneous with respect to the grading defined by the second argument.

The function “PoincareMultiDeg” ([I-16.12 pg.195](#)) is exactly the same as “HilbertSeriesMultiDeg” ([I-8.8 pg.105](#)).

NOTES: CoCoA-4 had an intrinsic limitation on multigradings which did not allow zero-entries in the first row of the defining matrix. This function performs all the appropriate conversions for computing the HilbertSeries wrt any positive grading (“IsPositiveGrading” ([I-9.51 pg.134](#))).

example

```
Use R ::= QQ[x,y,z];
```

```
WM := mat([[1,0,0],[1,-1,0]]);
HilbertSeriesMultiDeg(R/ideal(Indets())^2, WM);
```

See Also: HilbertSeries(I-8.7 pg.104), IsPositiveGrading(I-9.51 pg.134), PoincareMultiDeg(I-16.12 pg.195)

I-8.9 HilbertSeriesShifts

syntax

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

Description

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

This function computes the Hilbert-Poincare series of a (single-graded) module M with shifts. The input, M, must be homogeneous (with respect the weights list). 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.

The function “PoincareShifts” (I-16.13 pg.195) is exactly the same as “HilbertSeriesShifts” (I-8.9 pg.106).

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 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];
M := Module([x,y^3], [x-z,0]);
HilbertSeriesShifts(M, [2,0]); -- HilbertPoincare series of a shifted module
(2x^3) / (1-x)^3
-----
PoincareShifts(P^2/M, [3,1]); -- HP series of a shifted quotient module
(x + x^2 + 2x^3) / (1-x)^2
-----
```

See Also: dim(I-4.17 pg.67), hilbert(I-8.3 pg.102), HVector(I-8.13 pg.108), multiplicity(I-13.27 pg.170), WeightsMatrix(I-23.2 pg.267)

I-8.10 homog

syntax

```
homog(V: RINGELEM, X: RINGELEM): RINGELEM
homog(V: MODULEELEM, X: RINGELEM): MODULEELEM
homog(L: LIST, X: RINGELEM): LIST
homog(I: IDEAL, X: RINGELEM): IDEAL
homog(M: MODULE, X: RINGELEM): MODULE
```

Description

This function returns the homogenization of the first arg with respect to the indeterminate “X”, which must have weight 1. The elements of the list “L” are homogenized separately. For an ideal/module the result is

the ideal/module containing the homogenizations of all elements (and not simply the homogenizations of the specific generators).

example

```

/**/ Use R ::= QQ[x,y,z,w];
/**/ homog(x^3-y, w);
x^3 -y*w^2

/**/ homog([x^3-y, x^4-z], w);
[x^3 -y*w^2, x^4 -z*w^3]

/**/ I := ideal(x^3-y, x^4-z);
/**/ homog(I, w);    -- don't just get the homogenizations of
                    -- the generators of I
ideal(x*y -z*w, x^2*z -y^2*w, x^3 -y*w^2, y^3 -x*z^2)

```

See Also: IsHomog(I-9.44 pg.131)

I-8.11 HomogElimMat

syntax

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

Description

This function returns a matrix for a term ordering eliminating the indeterminates with indices in “ElimInd” for homogeneous input wrt the weights in the matrix “W”. If you don’t understand what this means use “ElimMat” (I-5.4 pg.73) instead ;-)

example

```

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

```

See Also: elim(I-5.3 pg.72), ElimMat(I-5.4 pg.73)

I-8.12 HSaturation

syntax

```
HSaturation(I: IDEAL, J: IDEAL): IDEAL
```

Description

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

This functions returns the saturation of I with respect to J: the ideal of polynomials F such that $F \cdot G$ is in I for all G in J^d for some positive integer d.

It calculates the saturation using a Hilbert-driven algorithm. It differs from “saturate” (I-19.1 pg.229) 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\(I-3.25 pg.47\)](#), [HColon\(I-8.1 pg.101\)](#), [saturate\(I-19.1 pg.229\)](#)

I-8.13 HVector

syntax

```
HVector(R: RING or TAGGED("Quotient")): LIST
```

Description

This function returns the h-vector of M, i.e., the coefficients of the numerator of the simplified Poincare series for M. M can be a module or a quotient.

The weights of the indeterminates of the polynomial ring of M must all be 1, and the coefficient ring must be a field.

If the input is not homogeneous, the Hilbert function of the corresponding leading term (initial) ideal or module is calculated.

example

```

/**/ Use R := QQ[t,x,y,z];
/**/ HVector(R/ideal(x,y,z)^5);
[1, 3, 6, 10, 15]

/**/ Poincare(R/ideal(x,y,z)^5);
(1 + 3t + 6t^2 + 10t^3 + 15t^4) / (1-t)

```

See Also: [hilbert\(I-8.3 pg.102\)](#), [HilbertSeries\(I-8.7 pg.104\)](#)

Chapter I-9

I

I-9.1 ideal

— syntax —

```
ideal(P_1: RINGELEM,...,P_n: RINGELEM): IDEAL
ideal(L: LIST): IDEAL
ideal(R: RING, L: LIST): IDEAL
```

Description

The first form returns the ideal generated by “P₁,...P_n”. The second form returns the ideal generated by the polynomials in “L”. 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-9.2 IdealAndSeparatorsOfPoints

— syntax —

```
IdealAndSeparatorsOfPoints(Points: LIST): RECORD
```

where Points is a list of lists of coefficients representing a set of “{\it distinct}” points in affine space.

Description

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

This function computes the results of “[IdealOfPoints](#)” ([I-9.4 pg.112](#)) and “[SeparatorsOfPoints](#)” ([I-19.5 pg.231](#)) together at a cost lower than making the two separate calls. The result is a record with three fields:

```
points      -- the points given as argument
ideal       -- the result of IdealOfPoints
separators  -- the result of SeparatorsOfPoints
```

Thus, if the result is stored in a variable with identifier X, then: X.points will be the input list of points; X.ideal will be the ideal of the set of points, with generators forming the reduced Groebner basis for the ideal; X.separators will be a list of polynomials whose i-th element will take the value 1 on the i-th point and 0 on the others.

NOTE:

- * the current ring must have at least as many indeterminates as the dimension of the space in which the points lie;
- * the base field for the space in which the points lie is taken to be the coefficient ring, which should be a field;
- * in the polynomials returned, the first coordinate in the space is taken to correspond to the first indeterminate, the second to the second, and so on;
- * if the number of points is large, say 100 or more, the returned value can be very large. To avoid possible problems when printing such values as a single item we recommend printing out the elements one at a time as in this example:

```
X := IdealAndSeparatorsOfPoints(Pts);
Foreach Element In gens(X.ideal) Do
  PrintLn Element;
EndForeach;
```

For ideals and separators of points in projective space, see “[IdealAndSeparatorsOfProjectivePoints](#)” ([I-9.3 pg.110](#)).

example

```
Use R ::= QQ[x,y];
Points := [[1, 2], [3, 4], [5, 6]];
X := IdealAndSeparatorsOfPoints(Points);
X.points;
[[1, 2], [3, 4], [5, 6]]
-----
X.ideal;
ideal(x - y + 1, y^3 - 12y^2 + 44y - 48)
-----
X.separators;
[1/8y^2 - 5/4y + 3, -1/4y^2 + 2y - 3, 1/8y^2 - 3/4y + 1]
-----
```

See Also: [GBM\(I-7.3 pg.92\)](#), [HGBM\(I-8.2 pg.101\)](#), [GenericPoints\(I-7.6 pg.93\)](#), [IdealAndSeparatorsOfProjectivePoints\(I-9.3 pg.110\)](#), [IdealOfPoints\(I-9.4 pg.112\)](#), [IdealOfProjectivePoints\(I-9.5 pg.113\)](#), [Interpolate\(I-9.25 pg.123\)](#), [QuotientBasis\(I-17.3 pg.205\)](#), [SeparatorsOfPoints\(I-19.5 pg.231\)](#), [SeparatorsOfProjectivePoints\(I-19.6 pg.232\)](#)

I-9.3 IdealAndSeparatorsOfProjectivePoints

syntax

```
IdealAndSeparatorsOfProjectivePoints(Points: LIST): RECORD
```

where Points is a list of lists of coefficients representing a set of “{\it distinct}” points in projective space.

Description

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

This function computes the results of “IdealOfProjectivePoints” (I-9.5 pg.113) and “SeparatorsOfProjectivePoints” (I-19.6 pg.232) together at a cost lower than making the two separate calls. The result is a record with three fields:

```
points      -- the points given as argument
ideal       -- the result of IdealOfProjectivePoints
separators  -- the result of SeparatorsOfProjectivePoints
```

Thus, if the result is stored in a variable with identifier X, then: X.ideal will be the ideal of the set of points, with generators forming a reduced Groebner basis for the ideal; X.separators will be a list of homogeneous polynomials whose i-th element will be non-zero (actually 1, using the given representatives for the coordinates of the points) on the i-th point and 0 on the others.

NOTE:

- * the current ring must have at least one more indeterminate than the dimension of the projective space in which the points lie, i.e, at least as many indeterminates as the length of an element of the input, Points;
- * the base field for the space in which the points lie is taken to be the coefficient ring, which should be a field;
- * in the polynomials returned, the first coordinate in the space is taken to correspond to the first indeterminate, the second to the second, and so on;
- * if the number of points is large, say 100 or more, the returned value can be very large. To avoid possible problems when printing such values as a single item we recommend printing out the elements one at a time as in this example:

```
X := IdealAndSeparatorsOfProjectivePoints(Pts);
Foreach Element In gens(X.ideal) Do
  PrintLn Element;
EndForeach;
```

For ideals and separators of points in affine space, see “IdealAndSeparatorsOfPoints” (I-9.2 pg.109).

example

```
Use R ::= QQ[x,y,z];
Points := [[0,0,1],[1/2,1,1],[0,1,0]];
X := IdealAndSeparatorsOfProjectivePoints(Points);
X.points;
[[0, 0, 1], [1, 1, 1], [0, 1, 0]]
-----
X.ideal;
ideal(xz - 1/2yz, xy - 1/2yz, x^2 - 1/4yz, y^2z - yz^2)
-----
X.separators;
[-2x + z, x, -2x + 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
-----
Hilbert(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-8.2 pg.101), GBM(I-7.3 pg.92), GenericPoints(I-7.6 pg.93), IdealAndSeparatorsOfPoints(I-9.2 pg.109), IdealOfPoints(I-9.4 pg.112), IdealOfProjectivePoints(I-9.5 pg.113), Interpolate(I-9.25 pg.123), QuotientBasis(I-17.3 pg.205), SeparatorsOfPoints(I-19.5 pg.231), SeparatorsOfProjectivePoints(I-19.6 pg.232)

I-9.4 IdealOfPoints

syntax

```

IdealOfPoints(P: RING, Points: MAT): IDEAL

```

where Points is a list of lists of coefficients representing a set of “{\it distinct}” points in affine space.

Description

This function computes the reduced Groebner basis for the ideal of all polynomials which vanish at the given set of points. It returns the ideal generated by that Groebner basis.

NOTE:

- * the current ring must have at least as many indeterminates as the dimension of the space in which the points lie;
- * 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-9.5 pg.113).

example

```

/**/ Use P := QQ[x,y];
/**/ Points := mat([[1, 2], [3, 4], [5, 6]]);
/**/ I := IdealOfPoints(P, Points);
/**/ I;
ideal(x -y +1, y^3 -12*y^2 +44*y -48)

/**/ K := NewFractionField(NewPolyRing(QQ, ["a"]));
/**/ Use K;
/**/ Points := mat([[1,2,0], [3,4,a], [5,1,6]]);
/**/ Use P := K[x,y,z], Lex;
/**/ I := IdealOfPoints(P, Points);
/**/ indent(I);
ideal(
  z^3 +(-a -6)*z^2 +(6*a)*z,
  y +((-a -12)/(6*a^2 -36*a))*z^2 +((a^2 +72)/(6*a^2 -36*a))*z -2,
  x +((2*a -6)/(3*a^2 -18*a))*z^2 +((-2*a^2 +36)/(3*a^2 -18*a))*z -1
)

```

See Also: GBM(I-7.3 pg.92), HGBM(I-8.2 pg.101), GenericPoints(I-7.6 pg.93), IdealAndSeparatorsOfPoints(I-9.2 pg.109), IdealAndSeparatorsOfProjectivePoints(I-9.3 pg.110), IdealOfProjectivePoints(I-9.5 pg.113), Interpolate(I-9.25 pg.123), QuotientBasis(I-17.3 pg.205), SeparatorsOfPoints(I-19.5 pg.231), SeparatorsOfProjectivePoints(I-19.6 pg.232)

I-9.5 IdealOfProjectivePoints

— syntax —

```

IdealOfProjectivePoints(Points: LIST): IDEAL

```

where Points is a list of lists of coefficients representing a set of ‘‘{\it distinct}’’ points in projective space.

Description

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

This function computes the 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 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:

```

I := IdealOfProjectivePoints(Pts);
Foreach Element In gens(I) Do
  PrintLn Element;
EndForeach;

```

For ideals of points in affine space, see “IdealOfPoints” (I-9.4 pg.112).

— example —

```

Use R ::= QQ[x,y,z];
I := IdealOfProjectivePoints([[0,0,1],[1/2,1,1],[0,1,0]]);
I;
ideal(xz - 1/2yz, xy - 1/2yz, x^2 - 1/4yz, y^2z - yz^2)
-----
I.Gens; -- the reduced Groebner basis
[xz - 1/2yz, xy - 1/2yz, x^2 - 1/4yz, y^2z - yz^2]
-----

```

See Also: GBM(I-7.3 pg.92), HGBM(I-8.2 pg.101), GenericPoints(I-7.6 pg.93), IdealAndSeparatorsOfPoints(I-9.2 pg.109), IdealAndSeparatorsOfProjectivePoints(I-9.3 pg.110), IdealOfPoints(I-9.4 pg.112), Interpolate(I-9.25 pg.123), QuotientBasis(I-17.3 pg.205), SeparatorsOfPoints(I-19.5 pg.231), SeparatorsOfProjectivePoints(I-19.6 pg.232)

I-9.6 IdentityMat

syntax

```
IdentityMat(R: RING, N: INT): MAT
```

Description

This function returns the NxN identity matrix.

example

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

I-9.7 if

syntax

```
If B_1 Then C_1 EndIf
If B_1 Then C_1 Else D EndIf
If B_1 Then C_1 Elif B_2 Then C_2 Elif ... EndIf
If B_1 Then C_1 Elif B_2 Then C_2 Elif ... Else D EndIf
```

where the B_j are boolean expressions,
and the C_j and D are command sequences.

Description

If “B_n” is the first in the sequence of the “B_j” to evaluate to True, then “C_n” is executed. If none of the “B_j” evaluates to True, then “D” is executed if present otherwise nothing is done. The construct, “Elif B_j Then C_j” can be repeated any number of times.

NB: “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
```

I-9.8 ILogBase

syntax

```
ILogBase(X: RAT, Base: INT): INT
```

Description

This function computes the integer part (floor) of the logarithm of a rational number in a given base. The signs of X and Base are ignored.

example

```
/**/ ILogBase(128,2);
7

/**/ ILogBase(81.5,3);
4
```

I-9.9 image

syntax

```
Image(V: OBJECT, F:TAGGED("RMap")): OBJECT
```

Description

In CoCoA-5 homomorphisms are properly implemented as “RINGHOM”. “Image” was the CoCoA-4 function mimicking homomorphisms, in particular “PolyAlgebraHom” ([I-16.14 pg.195](#)).

example

```
/**/ Use D ::= QQ[x,y]; -- domain
/**/ f := x-y; -- a RINGELEM in D

/**/ Use C ::= QQ[a,b,c]; -- codomain
/**/ F := RMap(a, c^2-a*b); -- OBSOLESCE
/**/ Image(f, F); -- OBSOLESCE
a*b -c^2 +a

/**/ phi := PolyAlgebraHom(D, C, [a, c^2-a*b]); -- a RINGHOM
/**/ phi(f);
a*b -c^2 +a
```

See Also: PolyAlgebraHom([I-16.14 pg.195](#)), apply([I-1.12 pg.28](#)), BringIn([I-2.10 pg.35](#)), subst([I-19.35 pg.246](#))

I-9.10 ImplicitPlot

syntax

```
ImplicitPlot(F: POLY, Xrange: LIST, Yrange: LIST)
```

Description

This function evaluates the first argument, a bivariate polynomial, at a grid of points in the range given by the second and third arguments. The coordinates of the approximate zeroes are output to a file called “CoCoAPlot”. See “ImplicitPlotOn” ([I-9.11 pg.116](#)) for outputting to another file.

This result can be plotted using your preferred plotting program. For example, start “gnuplot” and then give it the command

```
plot "CoCoAPlot"
```

to see the plot.

example

```
/**/ Use R ::= QQ[x,y];
/**/ ImplicitPlot(x^2 + y^2 - 200^2, [-256,256], [-256,256]);
Plotting points...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
800 plotted points have been placed in the file CoCoAPlot
```

See Also: `ImplicitPlotOn`([I-9.11](#) pg.116), `PlotPoints`([I-16.9](#) pg.194)

I-9.11 ImplicitPlotOn

syntax

```
ImplicitPlotOn(F: POLY, Xrange: LIST, Yrange: LIST, PlotFileName: STRING)
```

Description

This function is the same as “`ImplicitPlot`” ([I-9.10](#) pg.115) with a fourth argument giving the name of the file to print on.

Note that the last argument is a `STRING`, the name of the file, and not a `DEVICE`, as for “`print on`” ([I-16.22](#) pg.200).

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], "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], "lines");

/**/ ImplicitPlotOn(H, [-16,16], [-16,16], "curve");
Plotting points...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
1962 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-9.10](#) pg.115), `PlotPointsOn`([I-16.10](#) pg.194)

I-9.12 ImportByRef, ImportByValue

syntax

```
ImportByRef X;
ImportByValue X;
  where ‘\verb&X&’ is the name of a variable in the containing scope.
```

Description

“****YOU PROBABLY SHOULDN'T USE THESE COMMANDS YET!****” It seems that they can be used only inside anonymous functions (see “**func**” (I-6.21 pg.88)).

These commands “import” an external variable by reference or value. “**ImportByValue**” creates a local variable with the given name, and its initial value is taken from the variable of the same name in the context the anonymous function is defined. “**ImportByRef**” creates a reference to the named variable in the context where the anonymous function is defined.

— 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-20.9 pg.256), func(I-6.21 pg.88)

I-9.13 in

— syntax —

```
[X in L | B: BOOL]
[E | X in L]
[E | X in L and B]
  where L: LIST, B: BOOL, E: expression
  returns LIST
```

Description

See “List Constructors” (I-12.15 pg.154) 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(I-12.15 pg.154), IsIn(I-9.45 pg.131)

I-9.14 incr, decr

— syntax —

```
incr(ref X: INT)
decr(ref X: INT)

where ‘‘\verb&X&’’ has an integer value.
```

Description

“**incr(ref X)**” adds 1 to the value of “X”. “**decr(ref X)**” subtracts 1 from the value of “X”.

These functions are useful when counting objects or adjusting pointers.

example

```

/**/ L := [(10^k-1)/9 | k in 1..99];
/**/ NPrimes := 0;
/**/ Foreach N in L Do
/**/   If IsPrime(N) Then incr(ref NPrimes); EndIf;
/**/ EndForeach;
/**/ PrintLn "The list L contains ", NPrimes, " primes.";
The list L contains 3 primes.

```

I-9.15 indent

syntax

```

indent(X: OBJECT)
indent(X: OBJECT, N: INT)

```

Description

This function prints the argument “X” splitting it into several lines: a “LIST” or “IDEAL” is printed one element per line, a “RECORD” one field per line.

The second optional argument is for setting the level of recursive indentation; it is useful for example when printing a list of records.

example

```

/**/ L := [1,2] >< [3,4];
/**/ L;
[[1, 3], [1, 4], [2, 3], [2, 4]]

/**/ indent(L);
[
  [1, 3],
  [1, 4],
  [2, 3],
  [2, 4]
]

/**/ indent(L,2);
[
  [
    1,
    3
  ],
  --( Output suppressed )--
  [
    2,
    4
  ]
]

/**/ indent(record[B:=1,A:=2]);
record[
  A := 2,
  B := 1
]

```

See Also: [format\(I-6.15 pg.86\)](#)

I-9.16 *indet*

— [syntax](#) —

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

Description

This function returns the N-th indeterminate of the current ring.

— [example](#) —

```
/**/ Use R ::= QQ[x,y,z];
/**/ indet(R, 2);
y
```

See Also: [IndetSubscripts\(I-9.20 pg.121\)](#), [IndetIndex\(I-9.17 pg.119\)](#), [IndetName\(I-9.18 pg.119\)](#), [indets\(I-9.19 pg.120\)](#), [NumIndets\(I-14.27 pg.181\)](#)

I-9.17 *IndetIndex*

— [syntax](#) —

```
IndetIndex(X: RINGELEM): INT
```

Description

This function returns the position in which the indeterminate is listed when the corresponding ring was created.

— [example](#) —

```
/**/ Use R ::= QQ[x,y,z];
/**/ IndetIndex(y);
2

/**/ Use R ::= QQ[x[1..2,1..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\(I-9.16 pg.119\)](#), [IndetSubscripts\(I-9.20 pg.121\)](#), [IndetName\(I-9.18 pg.119\)](#), [indets\(I-9.19 pg.120\)](#), [NumIndets\(I-14.27 pg.181\)](#), [UnivariateIndetIndex\(I-21.1 pg.261\)](#)

I-9.18 *IndetName*

— [syntax](#) —

```
IndetName(X: RINGELEM): STRING
```

Description

This function returns the name of the indeterminate X as a string (i.e. the letter without the indices).

example

```

/**/ Use R ::= QQ[x,y,z];
/**/ IndetName(indet(R, 2));
y

/**/ type(It);
STRING

/**/ Use R ::= QQ[a, x[1..3]];
/**/ IndetName(Indet(R, 2));
x

/**/ indent(IndetSymbols(R));
[
  record[head := "a", indices := []],
  record[head := "x", indices := [1]],
  record[head := "x", indices := [2]],
  record[head := "x", indices := [3]]
]
```

See Also: [indet\(I-9.16 pg.119\)](#), [IndetSubscripts\(I-9.20 pg.121\)](#), [IndetIndex\(I-9.17 pg.119\)](#), [IndetSymbols\(I-9.21 pg.121\)](#), [NumIndets\(I-14.27 pg.181\)](#)

I-9.19 indets

syntax

```

indets(R: RING): LIST
indets(R: RING, X: STRING): LIST
```

Description

With one argument (a polynomial ring), this function returns the list of indeterminates of that polynomial ring. With two arguments (the second a STRING), it returns the list of all indeterminates whose name is the given string. The indeterminates in the list appear in order of increasing index (see the function “[IndetIndex](#)”).

This function used to be called “[IndetsCalled](#)” up to version 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"); -- work 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");
[]
```

See Also: [indet\(I-9.16 pg.119\)](#), [IndetSubscripts\(I-9.20 pg.121\)](#), [IndetIndex\(I-9.17 pg.119\)](#), [IndetName\(I-9.18 pg.119\)](#), [NumIndets\(I-14.27 pg.181\)](#)

I-9.20 IndetSubscripts

— syntax —

```
IndetSubscripts(X: RINGELEM): LIST
```

Description

This function returns the subscripts of the name of the argument, an indeterminate (used to be called “IndetInd”).

Please note the difference with “IndetIndex” ([I-9.17 pg.119](#)).

— 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\(I-9.16 pg.119\)](#), [IndetIndex\(I-9.17 pg.119\)](#), [IndetName\(I-9.18 pg.119\)](#), [IndetSymbols\(I-9.21 pg.121\)](#), [indets\(I-9.19 pg.120\)](#), [NumIndets\(I-14.27 pg.181\)](#)

I-9.21 IndetSymbols

— syntax —

```
IndetSymbols(P: RING): RECORD
```

Description

This function returns the list of the symbols in a polynomial ring. A symbol is a record “with” head (as “IndetName” ([I-9.18 pg.119](#))) and “indices” (as “IndetSubscripts” ([I-9.20 pg.121](#)))

— 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\(I-9.16 pg.119\)](#), [IndetSubscripts\(I-9.20 pg.121\)](#), [IndetIndex\(I-9.17 pg.119\)](#), [IndetName\(I-9.18 pg.119\)](#), [NumIndets\(I-14.27 pg.181\)](#), [SymbolRange\(I-19.40 pg.249\)](#)

I-9.22 InducedHom

syntax

```
InducedHom(RmodI: RING, phi: RINGHOM): RINGHOM
```

Description

`InducedHom(R, phi)` – where `R` is a `QuotientRing`, gives the homomorphism R/I into S induced by `phi`: R into S (which must have the base ring of `RmodI` as its domain, and whose kernel must contain the defining ideal of `RmodI`)

`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
/**/ psi := InducedHom(RmodI, CanonicalHom(S,SmodJ)(phi));
/**/ Use RmodI;
/**/ psi(x);
(a)

```

See Also: [NewQuotientRing\(I-14.9 pg.174\)](#), [CanonicalHom\(I-3.2 pg.37\)](#), [PolyAlgebraHom\(I-16.14 pg.195\)](#), [PolyRingHom\(I-16.15 pg.196\)](#)

I-9.23 InitialIdeal

syntax

```
InitialIdeal(I: IDEAL, Inds: LIST): IDEAL
```

Description

The “*initial form*” of a polynomial F is the homogeneous component of F of the lowest degree (in contrast with the “*leading form*”, see “`LF`” ([I-12.8 pg.150](#)), “`DF`” ([I-4.14 pg.66](#))). 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 II, pg.463).

example

```
/**/ Use R ::= QQ[x,y,z];
/**/ InitialIdeal(ideal(x^3-y), [x,y,z]);
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(y^2 -x*z, z^2, -y*z)
```

See Also: [InitialIdeal\(I-9.23 pg.122\)](#), [PrimaryPoincare\(I-16.19 pg.198\)](#)

I-9.24 *insert*

syntax

```
insert(ref V: LIST, N: INT, E: OBJECT)
```

where V is a variable containing a list.

Description

This function inserts E into the list L as the N-th component.

example

```
/**/ L := ["a","b","d","e"];
/**/ insert(ref L,3,"c");
/**/ L;
["a", "b", "c", "d", "e"]
```

See Also: [append\(I-1.11 pg.27\)](#), [remove\(I-18.21 pg.219\)](#)

I-9.25 *Interpolate*

syntax

```
Interpolate(Points: LIST, Values: LIST): RINGELEM
```

where Points is a list of lists of coefficients representing a set of “{\it distinct}” points and Values is a list of the same size containing numbers from the coefficient ring.

Description

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

This function returns a multivariate polynomial which takes given values at a given set of points.

NOTE:

- * the current ring must have at least as many indeterminates as the dimension of the space in which the points lie;
- * the base field for the space in which the points lie is taken to be the coefficient ring, which should be a field;
- * in the polynomials returned, the first coordinate in the space is taken to correspond to the first indeterminate, the second to the second, and so on;
- * if the number of points is large, say 100 or more, the returned value can be very large. To avoid possible problems when printing such values as a single item we recommend printing out the elements one at a time as in this example:

example

```

X := Interpolate(Pts, Vals);
Foreach Element In X Do
  PrintLn Element;
EndForeach;

Use QQ[x,y];
Points := [[1/2, 2], [3/4, 4], [5, 6/11], [-1/2, -2]];
Values := [1/2, 1/3, 1/5, -1/2];
F := Interpolate(Points, Values);
F;
-46849/834000y^2 - 1547/52125x + 13418/52125y + 46849/208500
-----
[Eval(F, P) | P In Points] = Values;  -- check
True
-----

```

I-9.26 interreduce, interreduced

syntax

```

interreduce(ref L: LIST of RINGELEM)
interreduced(L: LIST of RINGELEM): LIST of RINGELEM

```

Description

These functions reduce each polynomial using the other polynomials as reduction rules. The process terminates when each is in normal form with respect to the others. The function “**interreduce**” takes a variable containing a list and overwrites that variable with the interreduced list. The second returns an interreduced list without affecting its arguments.

example

```

/**/ interreduced([x^3-x*y^2+y*z, x*y, z]);
[x*y, z, x^3]

/**/ L := [x^3-x*y^2+y*z, x*y, z];
/**/ interreduce(ref L);
/**/ L;
[x*y, z, x^3]

```

I-9.27 intersection

syntax

```

intersection(A: LIST, B: LIST): LIST
intersection(A: IDEAL, B: LIST): LIST

```

```
intersection(A: LIST, B: IDEAL): LIST
intersection(A: IDEAL, B: IDEAL): IDEAL
```

Description

This function returns the intersection of “A” and “B”.

The coefficient ring must be a field.

NOTE: To compute the intersection of ideals corresponding to zero-dimensional schemes, see the commands “GBM” (I-7.3 pg.92) and “HGBM” (I-8.2 pg.101).

example

```
/**/ Use R ::= QQ[x,y,z];
/**/ intersection(ideal(x,y,z), ideal(x*y));
ideal(x*y)

/**/ intersection(["a","b","c"], ["b","c","d"]);
["b", "c"]
-----
```

See Also: GBM(I-7.3 pg.92), HGBM(I-8.2 pg.101), IntersectList(I-9.28 pg.125)

I-9.28 IntersectList

syntax

```
IntersectionList(L: LIST of LIST): LIST
IntersectionList(L: LIST of IDEAL): IDEAL
IntersectionList(L: LIST of MODULE): MODULE
```

Description

This function returns the intersection of all elements in “L”. Generalizes “intersection” (I-9.27 pg.124).

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-9.27 pg.124), IdealOfProjectivePoints(I-9.5 pg.113), IdealOfPoints(I-9.4 pg.112), HGBM(I-8.2 pg.101), intersection(I-9.27 pg.124)

I-9.29 inverse

syntax

```
inverse(X: MAT): MAT
```

Description

This function computes the multiplicative inverse of its argument. It is included for use when writing inverse(X) comes more naturally than writing “X⁽⁻¹⁾”, though both notations are functionally equivalent.

example

```

/**/ inverse(mat(QQ, [[1,2], [3,4]]));
matrix([
  [-2, 1],
  [3/2, -1/2]
])

```

I-9.30 InverseSystem

syntax

```
InverseSystem(I: IDEAL, D: INT): LIST of RINGELEM
```

Description

Given an ideal of derivations “I”, and an integer “D”, this function computes the degree “D” part of the inverse system of “I”.

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

example

```

/**/ Use QQ[x,y,z];
/**/ InverseSystem(ideal(x^3+x*y*z), 3);
[z^3, y*z^2, x*z^2, y^2*z, x^2*z, y^3, x*y^2, x^2*y, x^3 - 6*x*y*z]

```

See Also: [DerivationAction\(I-4.11 pg.65\)](#), [PerpIdealOfForm\(I-16.6 pg.192\)](#)

I-9.31 IO.SprintTrunc

syntax

```
$io.SprintTrunc(E: OBJECT, N: INT): STRING
```

Description

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

This function works like “[sprint](#)” ([I-19.20 pg.239](#)), turning the value of the expression E into a string, but if the string has length greater than N-1, it is truncated and the string “...” is concatenated. This function is useful in formatting reports of results.

example

```

Use R := QQ[x,y];
I := ideal(x,y);
$io.SprintTrunc(I,4);
Idea...
-----

```

See Also: [format\(I-6.15 pg.86\)](#), [sprint\(I-19.20 pg.239\)](#)

I-9.32 iroot

syntax

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

Description

This function computes the R-th root of an integer. If the argument is not a perfect R-th power it returns the integer part of the root.

example

```
/**/ iroot(25, 2);
5
/**/ iroot(99, 3);
4
/**/ iroot(-1, 3);
-1
```

See Also: ILogBase([I-9.8](#) pg.114)

I-9.33 IsAntiSymmetric

syntax

```
IsAntiSymmetric(M: MAT): BOOL
```

Description

This function tests whether the square matrix “M” is anti-symmetric.

example

```
/**/ M := mat([[0, 1, 2], [-1, 0, 3], [-2, -3, 0]]);
/**/ IsAntiSymmetric(M);
true
```

See Also: IsSymmetric([I-9.61](#) pg.137)

I-9.34 IsConstant

syntax

```
IsConstant(X: RINGELEM): BOOL
```

Description

This function tests whether the value of a RINGELEM of a polynomial ring actually lies in the image of the coefficient ring. It is equivalent to checking that the degree in each indeterminate is 0.

example

```
/**/ QQx := QQ[x];
/**/ Use QQx[y,z];
/**/ IsConstant(y+1);
false
/**/ IsConstant(x+1);
true
```

See Also: indets([I-9.19](#) pg.120)

I-9.35 IsContained

syntax

```
IsContained(A: IDEAL, B: IDEAL): BOOL
IsContained(A: MODULE, B: MODULE): BOOL
```

Description

This function tests whether A is contained in B. Was “ \leq ” 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-9.45](#) pg.131), IsSubset([I-9.60](#) pg.137)

I-9.36 IsDefined

syntax

IsDefined(E)

where E is a CoCoA expression.

Description

This function returns true if E is defined, otherwise it returns false. Typically, it is used to check if a name has already been assigned.

To know if a field in a record has been assigned use “fields” ([I-6.5](#) pg.81).

example

```
/**/ IsDefined(MyVariable);
false

/**/ MyVariable := 3;
/**/ IsDefined(MyVariable);
true
```

See Also: fields([I-6.5](#) pg.81)

I-9.37 IsDiagonal

syntax

IsDiagonal(M: MAT): BOOL

Description

This function tests whether the square matrix M is diagonal.

example

```
/**/ M := mat([[0, 1, 2],[-1, 0, 3],[-2, -3, 0]]);
/**/ IsDiagonal(M);
false
```

See Also: IsSymmetric([I-9.61](#) pg.137), DiagMat([I-4.15](#) pg.66)

I-9.38 IsDivisible

syntax

IsDivisible(A: RINGELEM, B: RINGELEM): BOOL

Description

This function says whether “A” is divisible by “B”; it returns “true” if so, otherwise “false”.

example

```
/**/ Use QQ[x,y,z];
/**/ IsDivisible(x, x^2*(y-1));
false
/**/ IsDivisible(x^2*(y-1), x);
true
```

See Also: valuation(I-22.1 pg.265)

I-9.39 IsElem

syntax

```
IsElem(A: RINGELEM, B: IDEAL): BOOL
IsElem(A: MODULEELEM, B: MODULE): BOOL
```

Description

This function tests whether A is an element of B. Same as the command “IsIn” (I-9.45 pg.131), 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-9.45 pg.131)

I-9.40 IsEven, IsOdd

syntax

```
IsEven(N: INT): BOOL
IsOdd(N: INT): BOOL
```

Description

These functions test whether an integer is even or odd.

example

```
/**/ IsEven(3);
false
/**/ IsOdd(3);
true
```

See Also: IsZero(I-9.66 pg.140)

I-9.41 IsFactorClosed

syntax

```
IsFactorClosed(L: LIST of power products): BOOL
```

Description

A set of power products is factor closed iff it contains every factor of every one of its elements. This function checks whether the given set is factor closed (also known as "order-ideal"). It is an error if 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\(I-17.3 pg.205\)](#), [LT\(I-12.22 pg.157\)](#), [TmpNBM\(I-20.8 pg.256\)](#), [IsStronglyStable\(I-9.59 pg.137\)](#)

I-9.42 IsField

syntax

```
IsField(R: RING): BOOL
```

Description

This function tests whether a ring is a field.

example

```
/**/ IsField(ZZ);
false
/**/ IsField(QQ);
true
```

See Also: [IsFiniteField\(I-9.43 pg.130\)](#)

I-9.43 IsFiniteField

syntax

```
IsFiniteField(R: RING): BOOL
```

Description

This function tests whether a ring is a finite field.

example

```
/**/ IsFiniteField(ZZ);
false
/**/ IsFiniteField(QQ);
false
/**/ Fp:=ZZ/(7); IsFiniteField(Fp);
true
```

See Also: [IsField\(I-9.42 pg.130\)](#), [IsPthPower\(I-9.54 pg.135\)](#), [LogCardinality\(I-12.18 pg.155\)](#), [PthRoot\(I-16.29 pg.203\)](#)

I-9.44 IsHomog

syntax

```
IsHomog(F: RINGELEM|MODULEELEM): BOOL
IsHomog(L: LIST): BOOL
IsHomog(I: IDEAL or MODULE): BOOL
```

Description

The first form of this function returns True if F is homogeneous. The second form returns True if every element of L is homogeneous. Otherwise, they return False. The third form returns True if the ideal/module can be generated by homogeneous elements, and False if not. Homogeneity is with respect to the first row of the weights matrix.

NOTE: when the grading dimension is 0 everything is trivially true. For safety reasons (from version 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 !!! instead of "true"
```

See Also: [deg\(I-4.6 pg.61\)](#), [wdeg\(I-23.1 pg.267\)](#)

I-9.45 IsIn

syntax

```
X IsIn Y (return BOOL)
```

Description

The semantics of “IsIn” is explained in the following table:

```

-----
| OBJECT      IsIn  LIST    checks if the list contains the object.  |
| POLY        IsIn  IDEAL   checks for ideal membership.         |
| MODULEELEM IsIn  MODULE  checks for module membership.        |
| STRING      IsIn  STRING  checks if the first string is a substring |
|                                     of the second one.           |
-----

```

IsIn operator

I-9.46 IsIndet

syntax

```
IsIndet(X: RINGELEM): BOOL
```

Description

This function tests whether “X” is an indeterminate. If so, it returns “true”; otherwise it returns “false”. An error is signalled if “X” is not a “RINGELEM” or if “RingOf(X)” is not a polynomial ring.

example

```

/**/ Use QQ[x,y,z];
/**/ IsIndet(x);
true
/**/ IsIndet(x-1);
false

```

I-9.47 IsInRadical

syntax

```
IsInRadical(F: RINGELEM, I: IDEAL): BOOL
IsInRadical(J: IDEAL, I: IDEAL): BOOL
```

Description

This function tests whether the first argument, a polynomial or an ideal, is contained in the radical of the second argument, an ideal.

This function is much faster than asking “F IsIn Radical(I);”.

example

```

/**/ Use QQ[x,y,z];
/**/ I := ideal(x^6*y^4, z);
/**/ IsInRadical(x*y, I);
true

/**/ IsInRadical(ideal(x,y), I);
false

/**/ MinPowerInIdeal(x*y, I);
6

```

See Also: [MinPowerInIdeal\(I-13.19 pg.167\)](#), [radical\(I-18.1 pg.209\)](#)

I-9.48 IsInSubalgebra

syntax

```
SubalgebraRepr(F: RINGELEM, L: LIST): RINGELEM
SubalgebraRepr(F: RINGELEM, M: TAGGED("$alghom.Map")): RINGELEM
```

Description

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

This function tests whether a polynomial is in a subalgebra, i.e. can be written as a polynomial expression of the elements of a list.

example

```
Use QQ[s,t];
L := [s^3, s^2t, st^2, t^3];
IsInSubalgebra(s^6t^6, L);
True
-----
SAM := SubalgebraMap(L);
IsInSubalgebra(s^6t^6, SAM); -- more efficient for repeated checks
True
-----
```

See Also: SubalgebraMap([I-19.29](#) pg.243), SubalgebraRepr([I-19.30](#) pg.244), ker([I-11.1](#) pg.145)

I-9.49 IsLexSegment

syntax

```
IsLexSegment(I: IDEAL): BOOL
```

Description

This function tests whether the monomial ideal I is a lex-segment ideal.

example

```
/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(x*y^3, y^4, x^3, x^2*y, x^2*z);
/**/ IsLexSegment(I);
false
```

See Also: IsStable([I-9.57](#) pg.136), IsStronglyStable([I-9.59](#) pg.137), LexSegmentIdeal([I-12.7](#) pg.150)

I-9.50 IsNumber

syntax

```
[OBSOLETE]
```

Description

[OBSOLETE] See “IsInteger”, “IsRational”

I-9.51 IsPositiveGrading

— syntax —

```
IsPositiveGrading(M: MAT): BOOL
IsPositiveGrading(M: MAT, N: INT): BOOL
```

Description

This function determines whether a matrix defines a positive grading, i.e. foreach column the first nonnegative entry is positive.

— example —

```
/**/ IsPositiveGrading(LexMat(5));
true

/**/ IsPositiveGrading(LexMat(5),3); --considering only the first three rows
false

/**/ IsPositiveGrading(mat([[0,2,3], [1, -1, 0]]));
true

/**/ IsPositiveGrading(mat([[1,1], [0,-1], [-1, 0]]));
true
```

See Also: HilbertSeriesMultiDeg([I-8.8](#) pg.105)

I-9.52 IsPrime

— syntax —

```
IsPrime(N: INT): BOOL
```

Description

This function determines whether a positive integer is prime; if N is not positive, an error is signalled. This function may be extremely slow when N is a large prime; in practice it is usually better to call IsProbPrime.

For the curious: currently, the function first performs a probabilistic check (Miller-Rabin), if that passes, it then verifies primality (via Lucas test).

— example —

```
/**/ IsPrime(32003);
true

/**/ IsPrime(10^100);
false
```

See Also: IsProbPrime([I-9.53](#) pg.134), NextPrime([I-14.12](#) pg.175)

I-9.53 IsProbPrime

— syntax —

```
IsProbPrime(N: INT): BOOL
```

Description

This function returns True if its integer argument passes a fairly stringent primality test; otherwise it returns False. There is a very small chance of the function returning True even though the argument is composite; if it returns False, we are certain that the argument is composite. Some people call it a compositeness test.

example

```

/**/ IsProbPrime(2);
true

/**/ IsProbPrime(1111111111111111111);
true

/**/ [N in 1..1111 | IsProbPrime((10^N-1)/9)]; -- only five values are known
[2, 19, 23, 317, 1031] -- next might be 49081

```

See Also: IsPrime([I-9.52](#) pg.134), NextProbPrime([I-14.13](#) pg.176)

I-9.54 IsPthPower

syntax

```

IsPthPower(X: RINGELEM): BOOL

```

Description

This function determines whether a polynomial over a finite field (of char p) is a p-th power. If the coefficient ring is not a finite field then an error is signalled.

example

```

/**/ Use ZZ/(7)[x];
/**/ IsPthPower(x^7+3);
true
/**/ IsPthPower(x^6+3);
false

```

See Also: IsFiniteField([I-9.43](#) pg.130), PthRoot([I-16.29](#) pg.203)

I-9.55 isqrt

syntax

```

isqrt(N: INT): INT

```

Description

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

example

```

/**/ isqrt(16);
4

/**/ isqrt(99);
9

-- /**/ isqrt(-1); --> !!! ERROR !!!

```

```
ERROR: Value must be non-negative
isqrt(-1);
~~~~~
```

I-9.56 IsQuotientRing

syntax

```
IsQuotientRing(R: RING): BOOL
```

Description

This function tests whether a ring is a quotient ring; it returns “true” if the ring is a quotient ring.

example

```
/**/ Use R ::= QQ[x,y];
/**/ S := R/ideal(x);
/**/ IsQuotientRing(S);
true;
```

See Also: DefiningIdeal(I-4.5 pg.61)

I-9.57 IsStable

syntax

```
IsStable(I: IDEAL): BOOL
```

Description

This function tests whether the monomial ideal I is stable.

example

```
/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(x*y^3, y^4, x^3, x^2*y, x^2*z);
/**/ IsStable(I);
true
```

See Also: IsLexSegment(I-9.49 pg.133), IsStronglyStable(I-9.59 pg.137), LexSegmentIdeal(I-12.7 pg.150)

I-9.58 IsStdGraded

syntax

```
IsStdGraded(P: RING): BOOL
```

Description

This function tests whether “P” is standard graded, “i.e.” “GradingDim” is 1 and all indeterminates in “P” have degree 1.

example

```
/**/ P ::= QQ[x,y,z];
/**/ IsStdGraded(P);
true
/**/ P ::= QQ[x,y,z], lex;
```

```

/**/ IsStdGraded(P);
false
/**/ P := NewPolyRing(QQ, ["x","y"], mat([[2,3],[1,2]]), 1);
/**/ IsStdGraded(P);
false

```

See Also: [NewPolyRing\(I-14.8 pg.174\)](#), [wdeg\(I-23.1 pg.267\)](#)

I-9.59 IsStronglyStable

syntax

```
IsStronglyStable(I: IDEAL): BOOL
```

Description

This function tests whether the monomial ideal I is strongly stable (Borel-fixed in characteristic 0).

example

```

/**/ Use R := QQ[x,y,z];
/**/ I := ideal(x*y^3, y^4, x^3, x^2*y, x^2*z);
/**/ IsStronglyStable(I);
true

```

See Also: [IsLexSegment\(I-9.49 pg.133\)](#), [IsStable\(I-9.57 pg.136\)](#)

I-9.60 IsSubset

syntax

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

Description

This function returns “true” if “MakeSet(L)” is contained in “MakeSet(M)”; otherwise it returns “false”.

example

```

/**/ IsSubset([1,1,2],[1,2,3,"a"]);
true
/**/ IsSubset([1,2],["a","b"]);
false
/**/ IsSubset([], [1,2]);
true

```

See Also: [IsContained\(I-9.35 pg.127\)](#), [IsIn\(I-9.45 pg.131\)](#), [EqSet\(I-5.5 pg.73\)](#), [MakeSet\(I-13.3 pg.160\)](#), [subsets\(I-19.34 pg.246\)](#)

I-9.61 IsSymmetric

syntax

```
IsSymmetric(M: MAT): BOOL
```

Description

This function tests whether the square matrix “M” is symmetric.

example

```
/**/ M := mat([[1, 2, 3], [2, 4, 5], [3, 5, 6]]);
/**/ IsSymmetric(M);
true
```

See Also: IsAntiSymmetric(I-9.33 pg.127)

I-9.62 IsTerm

syntax

```
IsTerm(X: RINGELEM|MODULEELEM): BOOL
```

Description

The function determines whether X is a term. For a polynomial, a “*term*” is a power-product, i.e., 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, e.g., $(0, x * y^2 * z, 0)$.

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-9.63 IsTermOrdering

syntax

```
IsTermOrdering(M: MAT): BOOL
```

Description

This function determines whether a square matrix defines a term-ordering, i.e. if its determinant is non-zero and if foreach column the first nonnegative entry is positive.

example

```
/**/ IsTermOrdering(LexMat(5));
true

/**/ IsTermOrdering(StdDegRevLexMat(5));
```

```

true

/**/ IsTermOrdering(RevLexMat(5));
false

```

See Also: NewPolyRing(I-14.8 pg.174), OrdMat(I-15.10 pg.189)

I-9.64 IsTree5

syntax

```

IsTree5(L: LIST): [BOOL, LIST ]
IsTree5(L: LIST, "NOOPT"): [BOOL, LIST]
IsTree5(L: LIST, "OPT"): [BOOL, LIST]
IsTree5(L: LIST, "CS_NOOPT"): [BOOL, LIST]
IsTree5(L: LIST, "CS_OPT"): [BOOL, LIST]

```

Description

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

This function is implemented in CoCoALib.

This function tests whether the facet complex described by the list L of square free power products is a tree, plus a list which:

- is empty if L is a tree
- contains three elements of a cycle of L if L is not a tree.

Four options “NOOPT”, “OPT”, “CS_NOOPT”, “CS_OPT” are available as second argument, specifying different algorithms; the default is “CS_OPT”.

For a full description of the algorithms we refer to the paper by M. Caboara, S. Faridi, and P. Selinger, “Simplicial cycles and the computation of simplicial trees”, Journal of Symbolic Computation, vol.42/1-2, pp.77-88 (2006).

example

```

Use R := QQ[x,y,z,t];
D := [x*y, y*z, z*t, t*x];
IsTree5(D);
[False, [xy, xt, yt]]
-----
IsTree5([xy, yz, zt]);
[True, [ ]]
-----

```

I-9.65 IsTrueGCDDomain

syntax

```

IsTrueGCDDomain(R: RING): BOOL

```

Description

This function tests whether a ring is a (true) GCD domain but not a field. CoCoA can compute GCDs of elements of a true GCD domain.

example

```

/**/ IsTrueGCDDomain(ZZ);
true
/**/ IsTrueGCDDomain(QQ);
false

```

See Also: IsField(I-9.42 pg.130)

I-9.66 IsZero

syntax

```

IsZero(X: OBJECT): BOOL

```

Description

This function tests whether its argument is zero; the argument can be of almost any type for which “zero” makes sense.

example

```

/**/ IsZero(23);
false
/**/ IsZero(3-3);
true
/**/ Use R ::= QQ[x,y,z];
/**/ IsZero(x^2+3*y-1);
false
/**/ IsZero(ideal(x^2,x*y^3));
false
/**/ F := NewFreeModule(R, 3);
/**/ zero(F);
[0, 0, 0]
/**/ IsZero(zero(F));
true
/**/ IsZero(matrix([[0,0,0], [0,0,0]]));
true

```

See Also: IsEven, IsOdd(I-9.40 pg.129), zero(I-25.1 pg.273), ZeroMat(I-25.2 pg.273)

I-9.67 IsZeroCol, IsZeroRow

syntax

```

IsZeroCol(M: MAT, N: INT): BOOL
IsZeroRow(M: MAT, N: INT): BOOL

```

Description

This function tests whether all entries in the “N”-th column(row) of “M” are zero.

example

```

/**/ IsZeroRow(matrix([[1,0,0], [0,0,0]]), 1);
false
/**/ IsZeroCol(matrix([[1,0,0], [0,0,0]]), 2);
true

```

See Also: IsZero(I-9.66 pg.140), ZeroMat(I-25.2 pg.273)

I-9.68 IsZeroDim

syntax

```
IsZeroDim(I: IDEAL): BOOL
```

Description

This function tests whether its argument is zero-dimensional.

example

```
/**/ Use QQ[x,y,z];
/**/ IsZero(ideal(x));
false
/**/ IsZeroDim(ideal(x^3, y^4-x ,z-3));
true
/**/ IsZeroDim(ideal(x^2, x*y^3));
false
```

See Also: [dim\(I-4.17 pg.67\)](#)

I-9.69 IsZeroDivisor

syntax

```
IsZeroDivisor(X: RINGELEM): BOOL
```

Description

This function tests whether its argument is a zero-divisor.

example

```
/**/ Use P ::= QQ[x,y,z];
/**/ R := NewQuotientRing(P, ideal(x*y));
/**/ IsZeroDivisor(RingElem(R,x));
true
/**/ colon(ideal(zero(R)), ideal(RingElem(R,x)));
ideal((y))
```

See Also: [colon\(I-3.25 pg.47\)](#)

I-9.70 It

syntax

```
It
```

Description

“**It**” is a top-level SYSTEM VARIABLE containing the last result computed but not assigned. It is the CoCoA equivalent to GAP’s “**last**”.

When CoCoA evaluates a “*standalone expression*”, the result is assigned to the system variable named “**It**” (and then printed as if in a “**println**” ([I-16.25 pg.201](#)) 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\(I-16.21 pg.199\)](#), [println\(I-16.25 pg.201\)](#), [Evaluation and Assignment\(II-4 pg.287\)](#)

Chapter I-10

J

I-10.1 jacobian

syntax

```
jacobian(L: LIST of RINGELEM): MAT
```

Description

This function returns the Jacobian matrix of the polynomials in “L” with respect to all the indeterminates of the current ring.

example

```
/**/ Use R ::= QQ[x,y];
/**/ L := [x-y, x^2-y, x^3-y^2];
/**/ jacobian(L);
matrix( /*RingDistrMPolyClean(QQ, 2)*/
  [[1, -1],
   [2*x, -1],
   [3*x^2, -2*y]])
```

I-10.2 JanetBasis

syntax

```
JanetBasis(I: IDEAL): LIST of RINGELEM
```

Description

Thanks to Mario Albert.

This function returns the Janet basis of an ideal.

example

```
/**/ Use R ::= QQ[x,y,z];
/**/ L := [x-y, x^2-z+1, x^3-y^2];
/**/ JanetBasis(ideal(L));
[x -y, z^2 -3*z +2, y*z -y -z +1, y^2 -z +1]
```

See Also: GBasis([I-7.1](#) pg.91)

Chapter I-11

K

I-11.1 ker

[syntax](#)

```
Ker(M:TAGGED("$alghom.Map")): IDEAL
```

Description

*** NOT YET IMPLEMENTED ***

This function returns the kernel of a K-algebra homomorphism.

[example](#)

```
Use QQ[s,t];
L := [s^3, s^2t, st^2, t^3];
SAM := SubalgebraMap(L);
SAM;
record[
  DomainIndets := SubalgebraRing::[x[1], x[2], x[3], x[4]],
  Images := CurrentRingEnv::[s^3, s^2t, st^2, t^3],
  ElimIdeal := AlgHomRing :: ideal(
    -y[1]^3 + x[1],
    -y[1]^2y[2] + x[2],
    -y[1]y[2]^2 + x[3],
    -y[2]^3 + x[4])
]
-----
Ker(SAM);
SubalgebraRing :: ideal(x[3]^2 - x[2]x[4], x[2]x[3] - x[1]x[4], x[2]^2 - x[1]x[3])
-----
```

See Also: SubalgebraMap([I-19.29](#) pg.243), SubalgebraRepr([I-19.30](#) pg.244), IsInSubalgebra([I-9.48](#) pg.133)

Chapter I-12

L

I-12.1 last

syntax

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

Description

In the first form, the function returns the last element of L. In the second form, it returns the list of the last N elements of L.

The CoCoA equivalent to GAP “last” is the variable “It” ([I-9.70 pg.141](#)).

example

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

See Also: first([I-6.6 pg.81](#)), tail([I-20.3 pg.254](#)), It([I-9.70 pg.141](#))

I-12.2 Latex

syntax

```
Latex(X: OBJECT): STRING
```

Description

This function returns a string containing the argument formatted in LaTeX. From version 4.7.5 it returns a string, so it can be printed on a file. Can also be called as “LaTeX”.

example

```
/**/ Use R ::= QQ[x,y,z];  
/**/ F := x^3+2*y^2*z;  
/**/ Latex(F);  
x^{3} + 2y^{2}z  
  
/**/ 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));
(x^2,
 y +z)

/**/ P := NewFractionField(R);
/**/ Use P;
/**/ F := (x+y)/(1-z)^3;
/**/ Latex(F);
\frac{-x-y}{z^3-3z^2+3z-1}

```

See Also: [format\(I-6.15 pg.86\)](#), [sprint\(I-19.20 pg.239\)](#)

I-12.3 LC

— syntax —

```
LC(F: RINGELEM|MODULEELEM): RINGELEM
```

Description

This function returns the leading coefficient of F, as determined by the term-ordering of the ring to which F belongs.

— example —

```

/**/ Use R := QQ[x,y];
/**/ LC(x +3*x^2 -5*y^2);
3

/**/ F := NewFreeModule(R,3);
/**/ LC(ModuleElem(F, [0, 5*y+6*x^2, y^2]));
6

```

See Also: [coefficients\(I-3.19 pg.43\)](#), [CoeffOfTerm\(I-3.22 pg.46\)](#), [LT\(I-12.22 pg.157\)](#)

I-12.4 lcm

— syntax —

```

lcm(N: INT, M: INT): INT
lcm(L: LIST of INT): INT

lcm(F: RINGELEM, G: RINGELEM): RINGELEM
lcm(L: LIST of RINGELEM): RINGELEM

```

Description

This function returns the least common multiple of “F₁, . . . , F_n” or of the elements in the list L. For the calculation of the GCDs and LCMs of polynomials, the coefficient ring must be a field.

example

```

/**/ Use R ::= QQ[x,y];
/**/ F := x^2-y^2;
/**/ G := (x+y)^3;
/**/ lcm(F, G);
-x^4 -2*x^3*y +2*x*y^3 +y^4

/**/ IsDivisible(F*G, It);
true

/**/ lcm(F, G) * gcd(F,G) = F*G;
true

/**/ lcm([3*4,3*8,6*16]);
96

```

See Also: [div\(I-4.20 pg.68\)](#), [mod\(I-13.20 pg.167\)](#), [gcd\(I-7.4 pg.92\)](#)

I-12.5 *len*

syntax

```
len(E: STRING|LIST): INT
```

Description

This function returns the “*length*” of an object, as summarized in the table below:

type	length
STRING	number of bytes in the string
LIST	number of items in the list

The function ‘‘\verb&len&’’

example

```

/**/ len( [2,3,4] );
3

/**/ len( "string" );
6

```

Previously “*len*” could be applied to other types too; this is no longer supported. See “*NumCompts*” ([I-14.26 pg.181](#)) for module elements, “*NumRows*” ([I-14.29 pg.182](#)) for matrices, and “*NumTerms*” ([I-14.30 pg.182](#)) for polynomials.

See Also: [count\(I-3.44 pg.55\)](#), [NumCompts\(I-14.26 pg.181\)](#), [NumRows\(I-14.29 pg.182\)](#), [NumTerms\(I-14.30 pg.182\)](#)

I-12.6 *LexMat*

syntax

```
LexMat(N: INT): MAT
```

Description

This function return the matrix defining a standard term-ordering. These functions return matrices defining standard term-orderings.

example

```
LexMat(3);
matrix([
  [1, 0, 0],
  [0, 1, 0],
  [0, 0, 1]
])
```

See Also: Orderings([III-9.5](#) pg.329), StdDegLexMat([I-19.26](#) pg.242), StdDegRevLexMat([I-19.27](#) pg.243), RevLexMat([I-18.29](#) pg.222), XelMat([I-24.1](#) pg.271)

I-12.7 LexSegmentIdeal

syntax

```
LexSegmentIdeal(L: LIST of power-products): IDEAL
LexSegmentIdeal(I: IDEAL): IDEAL
```

Description

If the argument is a list of power-products L, this function returns the smallest lex-segment ideal containing the power-products in L.

If it is an ideal I, it returns the lex-segment ideal having the same Hilbert function as I.

example

```
/**/ Use R ::= QQ[x,y,z];
/**/ LexSegmentIdeal([y^3]);
ideal(y^3, x*z^2, x*y*z, x*y^2, x^2*z, x^2*y, x^3)
/**/ LexSegmentIdeal(ideal(y^3));
ideal(x^3)
```

See Also: IsLexSegment([I-9.49](#) pg.133), StableIdeal([I-19.23](#) pg.241), StronglyStableIdeal([I-19.28](#) pg.243)

I-12.8 LF

syntax

```
LF(I: IDEAL): IDEAL
LF(F: RINGELEM): RINGELEM
```

Description

For a polynomial “F” this function returns the leading form, i.e. the sum of all summands having highest degree. It throws an error if the argument is zero or if the “GradingDim” ([I-7.18](#) pg.99) of the polynomial ring is 0 (use “DF” ([I-4.14](#) pg.66) 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-7.18](#) pg.99) 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 !!! instead of x-1

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

```

See Also: DF(I-4.14 pg.66), IsHomog(I-9.44 pg.131), LC(I-12.3 pg.148), LM(I-12.16 pg.154), LPP(I-12.21 pg.156), LT(I-12.22 pg.157)

I-12.9 LinearSimplify

syntax

```
LinearSimplify(F: RINGELEM): RECORD
```

Description

This function returns a “RECORD[LinearChange, SimplePoly]” where “LinearChange” is a linear change of variable and “SimplePoly” is simple (in a heuristic sense). The composition “SimplePoly(LinearChange)” is equal the univariate polynomial “F”.

example

```

/**/ Use QQ[x];
/**/ LinearSimplify((123*x-456)^9-1);
record[LinearChange := 123*x - 456, SimplePoly := x^9 - 1]

/**/ LinearSimplify(x^9-1); -- the heuristic finds no useful simplification
record[LinearChange := x, SimplePoly := x^9 - 1]

```

I-12.10 LinKer

syntax

```
LinKer(M: MAT): MAT
```

Description

This function returns a matrix whose columns representing 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 (CoCoA-5.0.3) on matrices whose entries are in a field. The CoCoA-4 function returning a ZZ-basis for the kernel of M is not yet implemented.

The output as it was given by CoCoA-4 (the basis of the ker) is now given by “LinKerBasis” (I-12.11 pg.152). See also “HilbertBasisKer” (I-8.4 pg.102).

example

```

/**/ M := mat([[1,2,3,4],[5,6,7,8],[9,10,11,12]]);
/**/ LinKer(M);
matrix(QQ,
  [[-1, -2],

```

```

[2, 3],
[-1, 0],
[0, -1]])

/**/ M*It;
matrix(QQ,
[[0, 0],
[0, 0],
[0, 0]])

```

See Also: [LinKerModP\(I-12.12 pg.152\)](#), [LinSol\(I-12.13 pg.153\)](#), [HilbertBasisKer\(I-8.4 pg.102\)](#)

I-12.11 LinKerBasis

— syntax —

```
LinKerBasis(M: MAT): LIST
```

Description

This function returns a list whose components are lists representing 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 (CoCoA-5.0.3) on matrices whose entries are in a field.

The CoCoA-4 function returning a Z-basis for the kernel of M is not yet implemented.

— example —

```

/**/ 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: [LinKerModP\(I-12.12 pg.152\)](#), [LinSol\(I-12.13 pg.153\)](#)

I-12.12 LinKerModP

— syntax —

```
LinKerModP(M: MAT): LIST
```

where M is a matrix over QQ or ZZ.

Description

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

This function returns a list whose components are lists representing a basis for the kernel of M over the current field of coefficients.

example

```

M := mat([[1,2,3,4],[5,6,7,8],[9,10,11,12]]);
Use ZZ/(3)[x];
LinKerModP(M);
[[1, 1, 1, 0], [0, 2, 2, 2]]
-----
Use ZZ/(7)[x];
LinKerModP(M);
[[2, 3, 2, 0], [1, 3, 5, 5]]
-----
N := M*Transposed(mat(It));
[ [ mod(X,Characteristic()) | X In Row ] | Row In N ];
[[0, 0], [0, 0], [0, 0]]
-----

```

See Also: LinKer([I-12.10 pg.151](#)), LinSol([I-12.13 pg.153](#))

I-12.13 LinSol

syntax

```
[OBSOLETE] use LinSolve
```

Description

REPLACED BY “LinSolve” ([I-12.14 pg.153](#)) **See Also:** LinSolve([I-12.14 pg.153](#))

I-12.14 LinSolve

syntax

```
LinSolve(M: MAT over ZZ|QQ, RHS: MAT): MAT
```

Description

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 “LinKer” ([I-12.10 pg.151](#)).

NOTE: an easy way of converting a list into a column matrix (for the second argument) is to use the function “ColMat” ([I-3.24 pg.46](#)).

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: LinKer([I-12.10](#) pg.[151](#))

I-12.15 List Constructors

syntax

```
A..B
[A,B,C,...]
[X in L: LIST | B: BOOL]: LIST
[E:expression | X in L]: LIST
[E:expression | X in L: LIST and B: BOOL]: LIST
```

Description

These operators create new lists.

“A..B” creates the list of integers from “A” to “B”, both ends are included.

“[A,B,C,...]” makes a list containing “A”, “B”, “C” and so on, in that order.

“[X in L | B]” makes a list of those elements in “L” for which condition “B” is true.

“[E | X in L]” evaluates the expression “E” for each “X” in “L”, and collects the results in a new list.

“[E | X in L and B]” evaluates the expression “E” for each “X” in “L” which satisfies the condition “B”, and collects the results in a new list.

example

```
/**/ []; --> empty list
[]
/**/ 1..4;
[1, 2, 3, 4]
/**/ [3,1,4,2];
[3, 1, 4, 2]
/**/ [N in 1..10 | IsPrime(N)];
[2, 3, 5, 7]
/**/ [N^2 | N in 1..4];
[1, 4, 9, 16]
/**/ [N^2 | N in 1..10 and IsPrime(N)];
[4, 9, 25, 49]
```

See Also: NewList([I-14.5](#) pg.[172](#)), append([I-1.11](#) pg.[27](#)), concat([I-3.31](#) pg.[50](#)), CartesianProduct, CartesianProductList([I-3.3](#) pg.[38](#))

I-12.16 LM

syntax

```
LM(X: RINGELEM): RINGELEM
LM(X: MODULEELEM): MODULEELEM
```

Description

This function returns the leading monomial of “X”. The monomial includes the coefficient. To get the leading term of “P”, (which does not include the coefficient), use “LT” ([I-12.22](#) pg.[157](#)).

example

```

/**/ Use R ::= QQ[x,y];
/**/ LM(3*x^2*y + y);
3*x^2*y

```

See Also: LC([I-12.3](#) pg.148), LF([I-12.8](#) pg.150), LPP([I-12.21](#) pg.156), LT([I-12.22](#) pg.157)

I-12.17 *log*

syntax

```
log(F: RINGELEM): LIST
```

Description

This function returns the list of exponents of the leading term of “F”. The inverse function is “LogToTerm” ([I-12.19](#) pg.155).

example

```

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

```

See Also: ILogBase([I-9.8](#) pg.114), LT([I-12.22](#) pg.157), LogToTerm([I-12.19](#) pg.155)

I-12.18 *LogCardinality*

syntax

```
LogCardinality(Fp: RING): INT
```

Description

This function returns the extension degree of a finite field over its prime field, or equivalently the log (base p) of its cardinality.

example

```

/**/ Fp ::= ZZ/(7);
/**/ Use Fpx ::= Fp[x];
/**/ Fq := Fpx/ideal(x^2+1);
/**/ LogCardinality(Fq);
2

```

See Also: IsFiniteField([I-9.43](#) pg.130), characteristic([I-3.8](#) pg.39)

I-12.19 *LogToTerm*

syntax

```
LogToTerm(R: RING, L: LIST of INT): RINGELEM
```

Description

This function returns the power-product whose list of exponents is “L”. It is the inverse of “log” ([I-12.17](#) pg.155).

example

```

/**/ Use R ::= QQ[x,y,z];
/**/ LogToTerm(R, [2,3,5]);
x^2*y^3*z^5

/**/ log(It);
[2, 3, 5]

```

See Also: [log\(I-12.17 pg.155\)](#)

I-12.20 LPosn

syntax

```
LPosn(V: MODULEELEM): INT
```

Description

This function returns the position of the leading power-product of “V”.

This function used to be called “LPos” up to version 5.0.3.

example

```

/**/ Use R ::= QQ[x,y,z]; -- the default term-ordering is DegRevLex
/**/ R4 := NewFreeModule(R,4); -- the default module ordering is TOPos
/**/ LPosn(ModuleElem(R4, [0, x, y^2, x^2]));
4
/**/ LPP(ModuleElem(R4, [0, x, y^2, x^2]));
x^2
/**/ LT(ModuleElem(R4, [0, x, y^2, x^2]));
[0, 0, 0, x^2]

Use R ::= QQ[x,y], PosTo;
LT(Vector(x,y^2));
Vector(x, 0)
-----
LPP(Vector(x,y^2));
x
-----
LPosn(Vector(x,y^2));
1
-----

```

See Also: [LF\(I-12.8 pg.150\)](#), [LM\(I-12.16 pg.154\)](#), [LPP\(I-12.21 pg.156\)](#), [LT\(I-12.22 pg.157\)](#)

I-12.21 LPP

syntax

```
LPP(X: RINGELEM): RINGELEM
LPP(X: MODULEELEM): RINGELEM
```

Description

This function returns the leading power-product of “X”; it discards information about which component the power-product appears in.

example

```

/**/ Use R ::= QQ[x,y];
/**/ LPP(3*x^2*y+y); -- LPP is the same as LT for polynomials
x^2*y

-- Note the difference between LPP and LT for MODULEELEM.
/**/ R4 := NewFreeModule(R,4); -- the default module ordering is TOPos
/**/ LPP(ModuleElem(R4, [0, x, y^2, x^2]));
x^2
/**/ LT(ModuleElem(R4, [0, x, y^2, x^2]));
[0, 0, 0, x^2]
/**/ LPosn(ModuleElem(R4, [0, x, y^2, x^2]));
4

```

See Also: LC(I-12.3 pg.148), LF(I-12.8 pg.150), LM(I-12.16 pg.154), LPosn(I-12.20 pg.156), LT(I-12.22 pg.157)

I-12.22 LT

syntax

```

LT(I: RINGELEM):  RINGELEM
LT(I: IDEAL):     IDEAL
LT(I: MODULEELEM): MODULEELEM
LT(I: MODULE):    MODULE

```

Description

If E is a polynomial this function returns the leading term of the polynomial E with respect to the term-ordering of the polynomial ring of E . For the leading monomial, which includes the coefficient, use “LM” (I-12.16 pg.154).

example

```

/**/ Use R ::= QQ[x,y,z]; -- the default term-ordering is DegRevLex
/**/ LT(y^2-x*z);
y^2

/**/ Use R ::= QQ[x,y,z], Lex;
/**/ LT(y^2-x*z);
x*z

```

If “ E ” is a MODULEELEM, “LT(E)” gives the leading term of “ E ” with respect to the module term-ordering of “ E ”. For the leading monomial, which includes the coefficient, use “LM” (I-12.16 pg.154).

example

```

/**/ R3 := NewFreeModule(R,3);
/**/ LT(ModuleElem(R3, [0, x, y^2]));
[0, 0, y^2]

Use R ::= QQ[x,y], PosTo;
V := Vector(0, x, y^2);
LT(V); -- the leading term of V w.r.t. PosTo
Vector(0, x, 0)
-----

```

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-12.3](#) pg.[148](#)), LF([I-12.8](#) pg.[150](#)), LM([I-12.16](#) pg.[154](#)), LPP([I-12.21](#) pg.[156](#)), Module Orderings([III-9.6](#) pg.[329](#)), Orderings([III-9.5](#) pg.[329](#))

Chapter I-13

M

I-13.1 MakeCheck

syntax

```
MakeCheck()
```

Description

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

This function run a series of tests on the whole system. To get a reliable result you should run this on a “*just opened*” CoCoA because some printouts may mysteriously add some empty spaces which will result in an, apparent, failure of some tests.

example

```
MakeCheck();
```

I-13.2 MakeMatByRows, MakeMatByCols

syntax

```
MakeMatByRows(R: INT, C: INT, L: LIST): MAT  
MakeMatByCols(R: INT, C: INT, L: LIST): MAT
```

Description

These functions convert the list L into a matrix. The first argument is the number of rows and the second the number of columns.

example

```
/**/ MakeMatByRows(2, 10, 1..20);  
matrix([  
  [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([  
  [1, 3, 5, 7, 9, 11, 13, 15, 17, 19],  
  [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]  
)
```

See Also: BlockMat(I-2.6 pg.33), matrix(I-13.8 pg.162), NewMat(I-14.6 pg.173), ColMat(I-3.24 pg.46), RowMat(I-18.39 pg.227), DiagMat(I-4.15 pg.66), ConcatHor(I-3.34 pg.51), ConcatVer(I-3.37 pg.52)

I-13.3 MakeSet

syntax

```
MakeSet(L: LIST): LIST
```

Description

This function returns a list obtained by removing duplicates from “L”.

example

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

NOTE: to test two sets for equality use the function “EqSet” (I-5.5 pg.73) instead of a normal equality test (because the latter yields false if the elements are in a different order).

See Also: EqSet(I-5.5 pg.73), intersection(I-9.27 pg.124), IntersectList(I-9.28 pg.125), insert(I-9.24 pg.123), remove(I-18.21 pg.219)

I-13.4 MantissaAndExponent10

syntax

```
MantissaAndExponent10(X: INT|RAT|RINGELEM, Prec: INT): RECORD
```

Description

This function converts a rational number into a “RECORD” with components named “exponent”, “mantissa” and “NumDigits”.

If “X=0”, all fields of the record are set to zero.

For non-zero “X” the fields give the best representation of the form $M * 10^E$ where “M” has “Prec” decimal digits. The value of “NumDigits” is simply “Prec”. The value of “exponent” is “ILogBase(X,10)”, plus 1 if the mantissa “overflows”. The value of “mantissa” is an integer “M” satisfying $10^{(Prec-1)} \leq |M| < 10^{Prec}$.

example

```
/**/ MantissaAndExponent10(1/2,3);      -- 1/2 = 5.00*10^(-1)  
record[NumDigits := 3, exponent := -1, mantissa := 500]  
  
/**/ MantissaAndExponent10(0.99999, 4); -- 0.99999 rounds up to give 1.000  
record[NumDigits := 4, exponent := 0, mantissa := 1000]
```

See Also: DecimalStr(I-4.3 pg.59), FloatApprox(I-6.10 pg.83), FloatStr(I-6.11 pg.83), ILogBase(I-9.8 pg.114), MantissaAndExponent2(I-13.5 pg.160), ScientificStr(I-19.3 pg.230)

I-13.5 MantissaAndExponent2

syntax

```
MantissaAndExponent2(X: INT|RAT|RINGELEM, Prec: INT): RECORD
```

Description

This function converts a rational number into a “RECORD” with components named “exponent”, “mantissa” and “NumDigits”.

If “X=0”, all fields of the record are set to zero.

For non-zero “X” the fields give the best representation of the form $M * 2^E$ where “M” has “Prec” bits. The value of “NumDigits” is simply “Prec”. The value of “exponent” is “ILogBase(X,2)”, plus 1 if the mantissa “overflows”. The value of “mantissa” is an integer “M” satisfying $2^{(Prec-1)} \leq |M| < 2^{Prec} - 1$

— 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: FloatApprox(I-6.10 pg.83), ILogBase(I-9.8 pg.114), MantissaAndExponent10(I-13.4 pg.160)

I-13.6 Manual

— syntax —

```

? key
?? key
where “{\it key}” is a literal string (without quotes)
```

Description

The “?” operator is used to search the online help system for information matching a keyword.

The command has the form “?key” where “key” is a string without quotes. The “?” operator 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 “?” operator was introduced in CoCoA 4.2.

The online help comprises a descriptive page for each CoCoA command/function. Each page summarises the syntax, gives a short verbal description, and some examples of use. Often there are also references to other closely related command/functions.

The search system is fairly simple. The searching algorithm looks through the title and keywords of each manual page. If the given “key” appears as a (case-insensitive) substring of the title/keywords of just one manual page, or if it matches exactly then that page is displayed. In any case the titles of all other pages where a match was found are printed in “menu” style.

— example —

```

/**/ ?pol

All matches for "pol":
? CharPoly
? DensePoly

    --( Further output suppressed )--

/**/ ?hvec
===== [ HVector ] =====
----- syntax -----
-- HVector(R: RING or TAGGED("Quotient")): LIST
----- description -----
This function returns the h-vector of M, i.e., the...
```

```
--( Output suppressed )--
```

I-13.7 MapDown

syntax

[OBSOLETE]

Description

[OBSOLETE] See “AsINT” ([I-1.14](#) pg.29), “AsRAT” ([I-1.15](#) pg.29).

I-13.8 matrix

syntax

```
matrix(L: LIST): MAT
matrix(R: RING, L: LIST): MAT
matrix(R: RING, M: MAT): MAT
```

Description

This function returns a matrix in the ring “R”.

When the input is “L”, a “*rectangular*” LIST of LIST of RINGELEM all in “R” (or INT, or RAT). When the ring is not specified it “guesses” the right ring; if all elements are INT or RAT the resulting matrix is in QQ.

The third form is equivalent to “`apply(CanonicalHom(RingOf(M),R), M)`”.

example

```
/**/ Use R := QQ[x,y];
/**/ L := [[1,2],[3,4]];
/**/ mat(L);
matrix(QQ,
  [[1, 2],
   [3, 4]])
/**/ mat(R,L);
matrix( /*RingDistrMPolyClean(QQ, 2)*/
  [[1, 2],
   [3, 4]])
/**/ mat(ZZ,L);
matrix(ZZ,
  [[1, 2],
   [3, 4]])

/**/ RingOf(mat(R, [[1,2],[3,4]]));
RingDistrMPolyClean(QQ, 2)

/**/ M := IdentityMat(ZZ,2); matrix(QQ, M);
matrix(QQ,
  [[1, 0],
   [0, 1]])
```

See Also: NewMat([I-14.6](#) pg.173), ColMat([I-3.24](#) pg.46), RowMat([I-18.39](#) pg.227), DiagMat([I-4.15](#) pg.66), MakeMatByRows, MakeMatByCols([I-13.2](#) pg.159), ConcatHor([I-3.34](#) pg.51), ConcatVer([I-3.37](#) pg.52), BlockMat([I-2.6](#) pg.33), apply([I-1.12](#) pg.28), CanonicalHom([I-3.2](#) pg.37)

I-13.9 Max

syntax

```
Max(E_1: OBJECT,...,E_n: OBJECT): OBJECT
Min(E_1: OBJECT,...,E_n: OBJECT): OBJECT

Max(L: LIST): OBJECT
Min(L: LIST): OBJECT
```

Description

In the first form, this function returns a maximum of E_1, \dots, E_n . In the second form, it returns a maximum of the objects in the list L.

example

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

/**/ Max(1,2,3);
3

/**/ Min(1,2,3);
1

/**/ 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-13.11 pg.164), Relational Operators(II-3.3 pg.284)

I-13.10 MayerVietorisTreeN1

syntax

```
MayerVietorisTreeN1(I: IDEAL): INT
```

Description

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

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^2yz, xy^2z, xyz^2]
```

See Also: Frobbys([II-8.3](#) pg.303)

I-13.11 Min

syntax

```
Min(E_1: OBJECT,...,E_n: OBJECT): OBJECT
```

```
Min(L: LIST): OBJECT
```

Description

In the first form, this function returns a minimum of E_1, \dots, E_n . In the second form, it returns a minimum of the objects in the list L.

example

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

/**/ Min(1,2,3);
1
```

See Also: Max([I-13.9](#) pg.163), Relational Operators([II-3.3](#) pg.284)

I-13.12 MinGens

syntax

```
MinGens(M: IDEAL|MODULE): LIST
```

Description

If M is a homogeneous ideal or module, this function returns a list of minimal generators for M.

The coefficient ring must be a field.

For non-homogeneous input use “MinGensGeneral” ([I-13.13](#) pg.165).

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: minimize([I-13.14](#) pg.165), minimized([I-13.15](#) pg.165)

I-13.13 MinGensGeneral

syntax

```
MinGensGeneral(M: IDEAL|MODULE): LIST
```

Description

If “M” is a homogeneous ideal or module, use “MinGens” (I-13.12 pg.164).

Otherwise this function returns a list of generators for “M” minimal in the sense that none can be generated by the others. There is no optimal algorithm of theoretical guarantee when input is not homogeneous.

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);
/**/ MinGensGeneral(I);
[x -1, x^4 -4*x^3*y +6*x^2*y^2 -4*x*y^3 +y^4, y +z]
```

See Also: [minimalize\(I-13.14 pg.165\)](#), [minimalized\(I-13.15 pg.165\)](#)

I-13.14 minimalize

syntax

```
minimalize(ref X: IDEAL)
minimalize(ref X: MODULE)
```

Description

Similar to “minimalized” (I-13.15 pg.165), but modifies the argument “X” and returns NULL.

example

```
/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(x^2-y^2, z^4-y^4, x^2-z^2);
/**/ I;
ideal(x^2 -y^2, -y^4 +z^4, x^2 -z^2)
/**/ minimalize(ref I); -- returns NULL and modifies I
/**/ I;
ideal(x^2 -z^2, y^2 -z^2)
```

See Also: [MinGens\(I-13.12 pg.164\)](#), [minimalized\(I-13.15 pg.165\)](#)

I-13.15 minimalized

syntax

```
minimalized(E: IDEAL): IDEAL
minimalized(E: MODULE): MODULE
```

Description

It works only in the homogeneous case: it returns the ideal or module generated by a set of minimal generators of E (with minimal cardinality). The minimal set of generators is not necessarily a subset of the given generators.

In the inhomogeneous case use “MinGensGeneral” (I-13.13 pg.165).

The coefficient ring is assumed to be a field.

The similar function “`minimalize`” (I-13.14 pg.165) performs the same operation, but modifies the argument (“`ref`” (I-18.15 pg.216)) and returns NULL.

example

```
/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(x^2-y^2, z^4-y^4, x^2-z^2);
/**/ I;
ideal(x^2 -y^2, -y^4 +z^4, x^2 -z^2)
/**/ minimalized(I);
ideal(x^2 -z^2, y^2 -z^2)
/**/ I; -- not modified
ideal(x^2 -y^2, -y^4 +z^4, x^2 -z^2)
```

See Also: `MinGens`(I-13.12 pg.164), `minimalize`(I-13.14 pg.165)

I-13.16 MinimalPresentation

syntax

```
MinimalPresentation(Q:TAGGED):TAGGED
```

where Q is a quotient module of the type R^s/M

Description

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

Given a quotient module of the type R^s/M , or a zero module, this function computes an isomorphic quotient, R^t/N , minimally presented [using the algortihm in Kreuzer-Robbiano II].

example

```
Use R ::= QQ[x,y,z];
MinimalPresentation(R^3/Module([[x,1,1], [x,2,2]]));
R^2/Module([[x, 0]])
-----
```

I-13.17 minors

syntax

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

Description

This function returns the list of all determinants of $N \times N$ submatrices of M .

example

```
/**/ M := mat([[1,2,3], [-1,2,4]]);
/**/ minors(M, 2);
[4, 7, 2]
```

See Also: `det`(I-4.13 pg.65)

I-13.18 MinPoly

syntax

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

Description

This function returns the minimal polynomial of the matrix “M” in the indeterminate “X” (with “M” a square matrix whose entries lies in the coefficient ring of “X”). See also “CharPoly” (I-3.9 pg.40).

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-3.9 pg.40)

I-13.19 MinPowerInIdeal

syntax

```
MinPowerInIdeal(F: RINGELEM, I: IDEAL): INT
```

Description

This function returns the minimum power of F, the first argument, in the ideal I, the second argument. If F is not in the radical I then -1 is returned.

example

```
/**/ Use QQ[x,y,z];
/**/ I := ideal(x^6*y^4, z);
/**/ IsInRadical(x*y, I);
true

/**/ MinPowerInIdeal(x*y, I);
6
```

See Also: IsInRadical(I-9.47 pg.132), radical(I-18.1 pg.209)

I-13.20 mod

syntax

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

Description

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

NOTE: To perform the division algorithm on a polynomial, use “NR” (I-14.23 pg.180) (normal remainder) to find the remainder, or “DivAlg” (I-4.21 pg.69) to get both the quotients and the remainder.

example

```

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

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

```

See Also: [div\(I-4.20 pg.68\)](#), [DivAlg\(I-4.21 pg.69\)](#), [GenRepr\(I-7.7 pg.94\)](#), [NF\(I-14.14 pg.176\)](#), [NR\(I-14.23 pg.180\)](#)

I-13.21 Mod2Rat

syntax

```

[OBSOLETE]

```

Description

[OBSOLETE] See “RatReconstructWithBounds” ([I-18.8 pg.212](#)).

I-13.22 ModuleElem

syntax

```

ModuleElem(M: MODULE, L: LIST): MODULEELEM

```

Description

This function returns the MODULEELEM (called “Vector” in CoCoA-4) in the module “M” whose components are the components of the list L.

example

```

/**/  Use R ::= QQ[x];
/**/  R3 := NewFreeModule(R,3);
/**/  V := ModuleElem(R3, [1, x, x^2]);  V;
[1, x, x^2]
/**/  type(V);
MODULEELEM
/**/  zero(R3);
[0, 0, 0]

```

See Also: [SubmoduleCols](#), [SubmoduleRows\(I-19.33 pg.245\)](#)

I-13.23 ModuleOf

syntax

```

ModuleElem(M: MODULE, L: LIST of RINGELEM): MODULEELEM

```

Description

This function returns the module on which the object E is defined.

NB A module contains many information and two separate rings, even when defined with the same commands, are not “equal”. When a module is printed only a few informations are shown, so different modules might look the same.

example

```

/**/ Use R ::= QQ[x];
/**/ R3 := NewFreeModule(R,3);
/**/ V := ModuleElem(R3, [1, x, x^2]); V;
[1, x, x^2]
/**/ type(V);
MODULEELEM
/**/ ModuleOf(V) = R3;
true
/**/ ModuleOf(V);
FreeModule(RingDistrMPolyClean(QQ, 1), 3)

```

I-13.24 monic

syntax

```
monic(F: RINGELEM): RINGELEM
```

Description

This function returns “F” divided by its leading coefficient (see “LC” (I-12.3 pg.148)) or, if “F” is zero, it returns just zero.

example

```

/**/ Use R ::= QQ[x,y];
/**/ F := 4*x^5-y^2;
/**/ monic(F);
x^5 +(-1/4)*y^2

/**/ Use R ::= ZZ[x,y];
/**/ F := 4*x^5-y^2;
-- /**/ monic(F); --> !!! ERROR !!! can't invert coefficients over ZZ
ERROR: Inexact division
monic(L); -- can't invert coefficient ...
~~~~~

/**/ Use P ::= ZZ/(5)[x,y];
/**/ F := 2*x^2+4*y^3;
/**/ monic(F);
y^3 -2*x^2

```

See Also: LC(I-12.3 pg.148)

I-13.25 monomials

syntax

```
monomials(F: RINGELEM|MODULEELEM): LIST
```

Description

This function returns the list of monomials of F. The function “support” (I-19.37 pg.248) 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\(I-3.19 pg.43\)](#), [support\(I-19.37 pg.248\)](#)

I-13.26 MonsInIdeal

syntax

```
MonsInIdeal(I: IDEAL): IDEAL
```

Description

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

This function returns the ideal generated by all monomials in the original ideal I.

example

```

Use R ::= QQ[x,y,z];
I := ideal(xy^3+z^2, y^5-z^3, xz-y^2-x^3, x^4-xz^2+y^3);
MonsInIdeal(I);
ideal(z^3, yz^2, x^2z^2, x^5z, x^4yz, x^5y, x^2y^2z, x^7, x^4y^2,
      xy^3z, y^4z, xy^4, x^3y^3, y^5)
-----

```

I-13.27 multiplicity

syntax

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

Description

This function computes the multiplicity (or degree) of M, i.e., the leading coefficient of the Hilbert polynomial multiplied by the factorial of the degree of the Hilbert polynomial. M can be a module or a quotient.

example

```

/**/ Use R ::= QQ[t,x,y,z];
/**/ multiplicity(R/ideal(x,y,z)^5);
35

```

See Also: [hilbert\(I-8.3 pg.102\)](#), [HilbertSeries\(I-8.7 pg.104\)](#), [HVector\(I-8.13 pg.108\)](#), [poincare\(I-16.11 pg.195\)](#)

Chapter I-14

N

I-14.1 NewFractionField

syntax

```
NewFractionField(R: RING): RING
```

Description

NOTE: calling twice “NewFractionField” will produce two different rings, even with identical input: equality test is performed on the pointers.

example

```
/**/ K := NewFractionField(NewPolyRing(QQ, ["a","b"]));
/**/ Use K;
/**/ M := mat([[1,2,3,a],[5,6,7,a*b]]);
/**/ LinKerBasis(M);
[[-1, 2, -1, 0], [(a*b -3*a)/2, (-a*b +5*a)/4, 0, -1]]
```

See Also: NewQuotientRing(I-14.9 pg.174), den(I-4.7 pg.62), num(I-14.24 pg.180)

I-14.2 NewFreeModule

syntax

```
NewFreeModule(R: RING, N: INT): MODULE
NewFreeModule(R: RING, Shifts: MAT): MODULE
```

Description

This function returns a free module which can be used as any programming variable.

NOTE: as for rings, calling twice “NewFreeModule” will produce two different modules, even with identical input: equality test is performed on the pointers.

This function does accept shifts from version CoCoA-5.0.4.

example

```
/**/ F := NewFreeModule(R, 3);
/**/ zero(F);
[0, 0, 0]
/**/ type(zero(F)); -- is NOT a LIST
MODULEELEM
/**/ gens(F);
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
```

```

/**/ F := NewFreeModule(R, matrix([[1],[2],[3]])); -- shifts
/**/ [wdeg(e) | e in gens(F)];
[[1], [2], [3]]

```

See Also: [BaseRing\(I-2.1 pg.31\)](#), [RingOf\(I-18.31 pg.224\)](#)

I-14.3 NewId

syntax

[OBSOLETE]

Description

This command is “*OBSOLETE*”.

I-14.4 NewLine

syntax

NewLine(): STRING

Description

This function is “*OBSOLESCE*NT” 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\(III-4.1 pg.315\)](#), [println\(I-16.25 pg.201\)](#), [ascii\(I-1.13 pg.28\)](#)

I-14.5 NewList

syntax

NewList(N: INT): LIST
NewList(N: INT, E: OBJECT): LIST

Description

The first form returns a list of length “N” filled with 0 (“INT”). The second form returns a list of length “N”, filled with copies of “E”.

example

```

/**/ NewList(4,"a");
["a", "a", "a", "a"]

```

```
/**/ NewList(4);
[0, 0, 0, 0]
```

See Also: List Constructors([I-12.15](#) pg.154)

I-14.6 NewMat

syntax

```
NewMat(R: RING, M: INT, N: INT): MAT
```

Description

This function is kept for CoCoA-4 nostalgia: better use “ZeroMat” ([I-25.2](#) pg.273).

example

```
/**/ Use S ::= QQ[x,y,z];
/**/ NewMat(S,2,3);
matrix( /*RingDistrMPolyClean(QQ, 3)*/
  [[0, 0, 0],
   [0, 0, 0]])
/**/ ZeroMat(S,2,3);
matrix( /*RingDistrMPolyClean(QQ, 3)*/
  [[0, 0, 0],
   [0, 0, 0]])
```

See Also: matrix([I-13.8](#) pg.162), NewMatFilled([I-14.7](#) pg.173)

I-14.7 NewMatFilled

syntax

```
NewMatFilled(M: INT, N: INT, Val: INT|RAT|RINGELEM): MAT
```

Description

This function returns an “MxN” matrix, filled with “Val”. If “Val” is an integer or rational the ring of the matrix is defined in “RingQQ” ([I-18.32](#) pg.224).

example

```
/**/ Use S ::= QQ[x,y,z];
/**/ NewMatFilled(1,3,x);
matrix( /*RingDistrMPolyClean(QQ, 3)*/
  [[x, x, x]])

/**/ NewMatFilled(1,3, 0);
matrix(QQ,
  [[0, 0, 0]])
/**/ ZeroMat(QQ, 1, 3); --> same as NewMatFilled(1,3, 0)
matrix(QQ,
  [[0, 0, 0]])
```

See Also: NewMat([I-14.6](#) pg.173), matrix([I-13.8](#) pg.162)

I-14.8 NewPolyRing

syntax

```
NewPolyRing(CoeffRing: RING, IndetNames: LIST: RING
NewPolyRing(CoeffRing: RING, IndetNames: LIST, OrdMat: MAT, GradingDim: INT: RING
```

Description

This function returns a polynomial ring which can be used as any programming variable (assigned with “:=”).

The “:=” syntax starts the input method for a new polynomial ring, with the special interpretation of brackets and symbols (i.e. $R ::= \mathbb{Q}\mathbb{Q}[x]$ is not read as $X := LL[i]$). The pre-defined orderings for the “:=” syntax are “Lex” (no grading), “DegLex”, “DegRevLex” (standard grading). For more orderings use the “NewPolyRing” function call (see also “ElimMat” (I-5.4 pg.73)).

NB: calling “NewPolyRing” twice with the same arguments gives two “different rings”, therefore incompatible.

example

```
/**/ R ::= QQ[x,y,alpha]; -- is equivalent to
/**/ R := NewPolyRing(RingQQ(), ["x","y","alpha"]);

/**/ R ::= QQ[x,y], DegRevLex; -- is equivalent to
/**/ R := NewPolyRing(RingQQ(), ["x","y"], StdDegRevLexMat(2), 1);

/**/ OrdM := matrix([[2,3],[1,0]]);
/**/ P := NewPolyRing(QQ, ["x","y"], OrdM, 1);
/**/ GradingDim(P);
1
/**/ P2 := NewPolyRing(RingZZ(), IndetSymbols(P));
/**/ Indets(P2);
[x, y]
/**/ P3 := NewPolyRing(P2, SymbolRange("alpha", 0,2));
/**/ indets(P3);
[alpha[0], alpha[1], alpha[2]]
```

See Also: ElimMat(I-5.4 pg.73), IndetSymbols(I-9.21 pg.121), SymbolRange(I-19.40 pg.249), GradingDim(I-7.18 pg.99)

I-14.9 NewQuotientRing

syntax

```
NewQuotientRing(R: RING, I: IDEAL): RING
R/I
```

Description

example

```
/**/ Use Qi ::= QQ[i];
/**/ CC := Qi/ideal(i^2+1); -- sort of ;-)
/**/ Use CC[x];
/**/ (x+i)^2;
(2*i*x +x^2 -1)

/**/ R ::= QQ[x,y,z];
/**/ S := NewQuotientRing(R, ideal(indet(R,1)-3));
/**/ Use S;
```

See Also: [QuotientBasis\(I-17.3 pg.205\)](#), [NewFractionField\(I-14.1 pg.171\)](#)

syntax

Description

example

See Also: NewQuotientRing(I-14.9 pg.174)

syntax

Description

example

See Also: AsRAT(I-1.15 pg.29)

syntax

```
NextPrime(N: INT): INT
```



```
/**/ NF(x^2+x*y+x*z+y^2+y*z+z^2, I);
x^2 +x*y +y^2 +x +y +1
```

See Also: DivAlg(I-4.21 pg.69), GenRepr(I-7.7 pg.94), IsIn(I-9.45 pg.131), NFsAreZero(I-14.15 pg.177), NR(I-14.23 pg.180)

I-14.15 NFsAreZero

syntax

```
NFsAreZero(L: LIST of RINGELEM, M: IDEAL): BOOL
NFsAreZero(L: LIST of MODULEELEM, M: MODULE): BOOL
```

Description

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

This function returns True if each component of L has normal form 0 with respect to M, i.e., if each component is an element of M. Otherwise, it returns False. The coefficient ring is assumed to be a field.

example

```
Use S ::= QQ[t,x,y,z];
I := ideal(t^31-t^6-x, t^8-y, t^10-z);
F := y^5-z^4;
G := (t^8-y)(3F+t^10-z);
NFsAreZero([F, G], I); -- F and G are in I
True
-----
NFsAreZero([F,x, G], I); -- x is not in I
False
-----
```

See Also: IsIn(I-9.45 pg.131), NF(I-14.14 pg.176)

I-14.16 NmzComputation

syntax

```
NmzComputation(Cone: RECORD): RECORD
NmzComputation(Cone: RECORD, ToCompute: LIST): RECORD
```

Description

(while waiting for a proper description, have a guess from this example ;-)

example

```
/**/ Cone := record[ integral_closure := mat([[1,2],[2,1]]) ];
/**/ NC := NmzComputation(Cone,
/**/ ["HilbertBasis","SupportHyperplanes"]
/**/ );
/**/ indent(NC);
record[
HilbertBasis := [[2, 1], [1, 2], [1, 1]],
SupportHyperplanes := [[-1, 2], [2, -1]]
]
```

I-14.17 NmzHilbertBasis

syntax

```
NmzHilbertBasis(M: MAT): MAT
```

Description

Given a matrix M, this function returns a matrix whose rows represent the Hilbert 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])

-- Different result for...
/**/      HilbertBasisKer(M);
-- the Hilbert basis of M is the Hilbert basis of the monoid of
-- elements in the kernel of M, namely...
[]
-- ...no elements! (except the zero-element)
```

See Also: [HilbertBasisKer\(I-8.4 pg.102\)](#), [NmzComputation\(I-14.16 pg.177\)](#), [NmzNormalToricRing\(I-14.20 pg.179\)](#), [NmzIntClosureMonIdeal\(I-14.18 pg.178\)](#)

I-14.18 NmzIntClosureMonIdeal

syntax

```
NmzIntClosureMonRing(L: LIST of RINGELEM): LIST of RINGELEM
```

Description

Given a list L of power-products in a ring R, the function returns the generators the integral closure of the ideal generated by the elements in L.

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
```

See Also: [NmzComputation\(I-14.16 pg.177\)](#), [NmzHilbertBasis\(I-14.17 pg.178\)](#), [NmzNormalToricRing\(I-14.20 pg.179\)](#), [NmzIntClosureMonIdeal\(I-14.18 pg.178\)](#)

I-14.19 NmzIntClosureToricRing

syntax

```
NmzIntClosureToricRing(L: LIST of RINGELEM): LIST of RINGELEM
```

Description

Given a list L of power-products in a ring R, the function returns the generators of the integral closure of the algebra generated by the list.

example

```
/**/      Use R:=QQ[x,y,t];
/**/      NmzIntClosureToricRing([x^3,x^2*y,y^3]);
-- the integral closure of QQ[x^3, x^2*y, y^3] is...
[y,x]
-- ... QQ[y, x]
```

See Also: NmzComputation(I-14.16 pg.177), NmzHilbertBasis(I-14.17 pg.178), NmzNormalToricRing(I-14.20 pg.179), NmzIntClosureMonIdeal(I-14.18 pg.178)

I-14.20 NmzNormalToricRing

syntax

```
NmzNormalToricRing(L: LIST of RINGELEM): LIST of RINGELEM
```

Description

Given a list L of power-products in a ring R, the function returns the generators of the normalization of the algebra generated by the list.

example

```
/**/      Use R:=QQ[x,y,t];
/**/      NmzNormalToricRing([x^3,x^2*y,y^3]);
-- the normalization of QQ[x^3,x^2*y,y^3] is...
[y^3, x^2*y, x^3, x*y^2]
-- QQ[y^3, x^2*y, x^3, x*y^2]
```

See Also: NmzComputation(I-14.16 pg.177), NmzHilbertBasis(I-14.17 pg.178), NmzIntClosureToricRing(I-14.19 pg.178), NmzIntClosureMonIdeal(I-14.18 pg.178)

I-14.21 NonZero

syntax

```
NonZero(L: LIST|MODULEELEM): LIST
```

Description

This function returns the list obtained by removing the zeroes from L.

example

```
/**/      Use R := QQ[x,y,z];
/**/      NonZero([0,0,3, ideal(y),0]);
[3, ideal(y)]
```

See Also: FirstNonZero(I-6.7 pg.81), FirstNonZeroPosn(I-6.8 pg.82), IsZero(I-9.66 pg.140)

I-14.22 not

syntax

```
not(A: BOOL): BOOL
```

Description

This function negates a boolean: i.e. if “A” gives “true” then “not(A)” gives “false”, and vice versa.

Note that from CoCoA-5.1 “not” is a function, so its argument must be between brackets!

example

```
/**/ [n in 1..10 | not(IsPrime(n))];
[1,4,6,8,9]
```

See Also: and(I-1.10 pg.27), or(I-15.9 pg.189)

I-14.23 NR

syntax

```
NR(X: RINGELEM, L: LIST of RINGELEM): RINGELEM
NR(X: MODULEELEM, L: LIST of MODULEELEM): MODULEELEM
```

Description

This function returns the normal remainder of X with respect to L, i.e., it returns the remainder from the division algorithm. To get both the quotients and the remainder, use “DivAlg” (I-4.21 pg.69).

Note that if the list does not form a Groebner basis, the remainder may not be zero even if X is in the ideal or module generated by L (use “GenRepr” (I-7.7 pg.94) or “NF” (I-14.14 pg.176) instead).

Currently (v 5.0.3) the internal code for computing NF(F, I) and NR(F, GBasis(I)) is identical, but the second is slower just for the overhead in interpreting a possibly long list of polynomials.

example

```
/**/ Use R ::= QQ[x,y,z];
/**/ F := x^2*y + x*y^2 + y^2;
/**/ NR(F, [x*y-1, y^2-1]);
x + y + 1

// NOT YET IMPLEMENTED for MODULEELEM
```

See Also: DivAlg(I-4.21 pg.69), GenRepr(I-7.7 pg.94), NF(I-14.14 pg.176)

I-14.24 num

syntax

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

Description

This function returns the numerator of “N”.

The OBSOLETE fragile syntax in CoCoA 4 “N.Num” and “N.Den” is no longer supported.

example

```
/**/ num(3);
3

/**/ P ::= QQ[x,y];
/**/ F := NewFractionField(P);
/**/ Use F;
```

```
/**/ num(x/(x+y));
x
```

See Also: [den\(I-4.7 pg.62\)](#)

I-14.25 NumCols

— syntax —

```
NumCols(M: MAT): INT
```

Description

This function returns the number of columns in a matrix.

— example —

```
/**/ M := mat([[1,2,3], [4,5,6]]);
/**/ NumCols(M);
3
```

See Also: [matrix\(I-13.8 pg.162\)](#), [NumRows\(I-14.29 pg.182\)](#)

I-14.26 NumCompts

— syntax —

```
NumCompts(X: MODULEELEM|MODULE): INT
```

Description

If “X” is a “MODULEELEM”, it returns the number of components of “X”. If “X” is a “MODULE”, it returns the rank of the free module in which “X” is defined.

This function used to be called “NumComps” in CoCoA-4.

— example —

```
/**/ Use R ::= QQ[x,y];
/**/ R2 := NewFreeModule(R, 3);
/**/ M := SubmoduleRows(R2, matrix(R, mat([[x,0,y], [x^2+y^2,x^2,3]])));
/**/ NumCompts(M);
3
/**/ NumCompts(gens(M)[1]);
3
```

See Also: [len\(I-12.5 pg.149\)](#)

I-14.27 NumIndets

— syntax —

```
NumIndets(R: RING): INT
```

Description

This function returns the number of indeterminates of the current ring or of R.

example

```

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

/**/ NumIndets(S);
2

```

See Also: [indet\(I-9.16 pg.119\)](#), [IndetSubscripts\(I-9.20 pg.121\)](#), [IndetIndex\(I-9.17 pg.119\)](#), [IndetName\(I-9.18 pg.119\)](#), [indets\(I-9.19 pg.120\)](#)

I-14.28 NumPartitions

syntax

```
NumPartitions(N: INT): INT
```

Description

This function returns the number of partitions of a non-negative integer, i.e. the number of distinct ways of writing “N” as a sum of positive integers.

example

```

/**/ NumPartitions(2); -- 2 and 1+1
2
/**/ NumPartitions(5);
7

```

I-14.29 NumRows

syntax

```
NumRows(M: MAT): INT
```

Description

This function returns the number of rows in a matrix.

example

```

/**/ M := mat([[1,2,3], [4,5,6]]);
/**/ NumRows(M);
2

```

See Also: [matrix\(I-13.8 pg.162\)](#), [NumCols\(I-14.25 pg.181\)](#)

I-14.30 NumTerms

syntax

```
NumTerms(F: RINGELEM): INT
```

Description

This function returns the number of terms in a polynomial.

example

```
/**/ Use R := QQ[x,y,z];  
/**/ NumTerms((x+y+z)^5) = binomial(3+5-1, 5);  
true
```

See Also: [len\(I-12.5 pg.149\)](#)

Chapter I-15

O

I-15.1 one

— syntax —

```
one(R: RING): RINGELEM
```

Description

This function return the multiplicative identity of a ring. For when you want to force the integer “1” to be a “RINGELEM”.

— example —

```
/**/ P := ZZ/(101)[x,y,z];
/**/ N := 1; Print N, " of type ", type(N);
1 of type INT
/**/ N := one(P); Print N, " of type ", type(N);
1 of type RINGELEM
/**/ N := 300*1; Print N, " of type ", type(N);
300 of type INT
/**/ N := 300*one(P); Print N, " of type ", type(N);
-3 of type RINGELEM
```

See Also: [zero\(I-25.1 pg.273\)](#)

I-15.2 OpenIFile

— syntax —

```
OpenIFile(S: STRING): DEVICE
```

Description

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

This function opens the file with name S for input. Input from that file can then be read with “[Get](#)” ([I-7.10 pg.96](#)).

(Note: one would normally use “[source](#)” ([I-19.17 pg.238](#)) to read CoCoA commands from a file.)

— example —

```
D := OpenOFile("my-test"); -- open "my-test" for output from CoCoA
Print "hello world" On D;   -- print string into "mytest"
Close(D);
```

```

D := OpenIFile("my-test"); -- open "my-test" for input to CoCoA
Get(D,3); -- get the first three characters (in Ascii code)
[104, 101, 108]
-----
ascii(It); -- convert the ASCII code into characters
hel
-----
Close(D);

```

See Also: [close\(I-3.13 pg.41\)](#), [Introduction to IO\(II-6.1 pg.291\)](#), [OpenOFile\(I-15.5 pg.187\)](#), [OpenIString\(I-15.3 pg.186\)](#), [OpenOString\(I-15.6 pg.188\)](#), [OpenSocket\(I-15.7 pg.188\)](#), [source\(I-19.17 pg.238\)](#)

I-15.3 OpenIString

syntax

```

OpenIString(S: STRING, T: STRING): DEVICE
OpenOString(S: STRING): DEVICE

```

Description

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

This function open strings for input. The string S serves as the name of the device opened for input or output; one may use the empty string. “OpenIString” is used to read input from the string T with the help of “Get” ([I-7.10 pg.96](#)).

example

```

S := "hello world";
D := OpenIString("", S); -- open the string S for input to CoCoA
L := Get(D,7); -- read 7 characters from the string
L; -- ASCII code
[104, 101, 108, 108, 111, 32, 119]
-----
ascii(L); -- convert ASCII code to characters
hello w
-----
Close(D); -- close device D

```

See Also: [close\(I-3.13 pg.41\)](#), [Introduction to IO\(II-6.1 pg.291\)](#), [OpenOString\(I-15.6 pg.188\)](#), [OpenIFile\(I-15.2 pg.185\)](#), [OpenOFile\(I-15.5 pg.187\)](#), [source\(I-19.17 pg.238\)](#), [sprintf\(I-19.20 pg.239\)](#)

I-15.4 OpenLog

syntax

```

OpenLog(D: DEVICE)

```

Description

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

This function opens the output device D and starts to record the output from a CoCoA session on D. The “CloseLog” ([I-3.14 pg.42](#)) 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.187)) or an output string (see “[OpenOString](#)” ([I-15.6](#) pg.188)). Several output devices may be open at a time. If the panel option “Echo” is set to True, both the input and output of the CoCoA session are logged; otherwise, just the output is logged.

example

```
D := OpenOFile("MySession");
OpenLog(D);
1+1;
2
-----
G := 1;
Set Echo;
2+2;
2 + 2
4
-----
F := 2;
F := 2
CloseLog(D);
CloseLog(D)
UnSet Echo;
SET(Echo, False)

-- The contents of "MySession":
2
-----
2 + 2
4
-----
F := 2
CloseLog(D)
```

See Also: Introduction to IO([II-6.1](#) pg.291), [OpenIFile](#)([I-15.2](#) pg.185), [OpenOFile](#)([I-15.5](#) pg.187), [OpenIString](#)([I-15.3](#) pg.186), [OpenOString](#)([I-15.6](#) pg.188), [Unset](#)([I-21.3](#) pg.262)

I-15.5 *OpenOFile*

syntax

```
OpenOFile(S: STRING): DEVICE
OpenOFile(S: STRING,"w" or "W"): DEVICE
```

Description

This function opens the file with name S—creating it if it does not already exist—for output. If used with second argument “w” or “W” then it immediately erases the file S. The function “[print on](#)” ([I-16.22](#) pg.200) is then used for appending output to S.

example

```
D := OpenOFile("my-test"); -- open "my-test" for output from CoCoA
Print "hello world" On D; -- print string into "mytest"
Print " test" On D; -- append to the file "mytest"
Close(D); -- close the file
D := OpenOFile("my-test","w"); -- clear "my-test"
Print "goodbye" On D; -- "mytest" now consists only of the string "goodbye"
Close(D);
```

See Also: [close\(I-3.13 pg.41\)](#), [Introduction to IO\(II-6.1 pg.291\)](#), [OpenIFile\(I-15.2 pg.185\)](#), [OpenIString\(I-15.3 pg.186\)](#), [OpenOString\(I-15.6 pg.188\)](#), [source\(I-19.17 pg.238\)](#)

I-15.6 OpenOString

syntax

```
OpenOString(S: STRING): DEVICE
```

Description

This function opens strings for output. The string S serves as the name of the device opened for input or output; one may use the empty string. “OpenOString” is used to write to a string with the help of “print on”.

example

```
D := OpenOString(""); -- open a string for output from CoCoA
L := [1,2,3]; -- a list
Print L On D; -- print to D
D;
record[Name := "", Type := "OString", Protocol := "CoCoALanguage"]
-----
S := Cast(D, STRING); -- S is the string output to D
S; -- a string
[1, 2, 3]
-----
Print " more characters" On D; -- append to the existing output string
Cast(D, STRING);
[1, 2, 3] more characters
-----
```

See Also: [close\(I-3.13 pg.41\)](#), [Introduction to IO\(II-6.1 pg.291\)](#), [OpenIFile\(I-15.2 pg.185\)](#), [OpenOFile\(I-15.5 pg.187\)](#), [OpenIString\(I-15.3 pg.186\)](#), [source\(I-19.17 pg.238\)](#), [sprintf\(I-19.20 pg.239\)](#)

I-15.7 OpenSocket

syntax

```
OpenSocket(Machine: STRING, Port: STRING): DEVICE
```

Description

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

This function opens a client socket (I/O) connection. It requires the name of the machine with the server socket and the port number (expressed as a STRING).

CoCoA-4 communicates with the CoCoAServer via socket which, by default, runs on “localhost” on port “0xc0c0”. To change these settings redefine in your “userinit.coc” or “.cocoarc” the variables

```
MEMORY.CoCoAServerMachine := "localhost";
MEMORY.CoCoAServerPort := "0xc0c0";
```

example

```
D := OpenSocket("localhost", "10000");
Print 100^6 On D;
Source D;
Close(D);
```

See Also: Introduction to IO([II-6.1](#) pg.291), OpenIFile([I-15.2](#) pg.185), OpenOFile([I-15.5](#) pg.187), OpenIString([I-15.3](#) pg.186), OpenOString([I-15.6](#) pg.188), Unset([I-21.3](#) pg.262)

I-15.8 Option

syntax

[OBSOLETE]

Description

“OBSOLETE”

I-15.9 or

syntax

A or B (where A, B: BOOL, return BOOL)

Description

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

example

```
/**/ Define IsUnsuitable(X)
/**/   Return X < 0 or isqrt(X) >= 2^16;
/**/ EndDefine;
/**/ IsUnsuitable(-9);
true
/**/ IsUnsuitable(9);
false
```

See Also: and([I-1.10](#) pg.27), not([I-14.22](#) pg.179)

I-15.10 OrdMat

syntax

OrdMat(R: RING): MAT

Description

This function returns a matrix which describes the term-ordering of the ring “R”.

example

```
/**/ Use S ::= QQ[x,y,z];
/**/ M := mat([ [1,2,3], [3,4,5], [0,0,1]]);
/**/ P := NewPolyRing(CoeffRing(S), IndetSymbols(S), M, 2);
/**/ GradingDim(P);
2
/**/ OrdMat(P);
matrix(QQ,
  [[1, 2, 3],
```

```
[3, 4, 5],  
[0, 0, 1]])  
  
/**/ GradingDim(S);  
1  
/**/ OrdMat(S);  
matrix(QQ,  
  [[1, 1, 1],  
   [0, 0, -1],  
   [0, -1, 0]])
```

See Also: StdDegLexMat([I-19.26](#) pg.242), StdDegRevLexMat([I-19.27](#) pg.243), LexMat([I-12.6](#) pg.149), RevLexMat([I-18.29](#) pg.222), XelMat([I-24.1](#) pg.271), elim([I-5.3](#) pg.72), GradingDim([I-7.18](#) pg.99), Orderings([III-9.5](#) pg.329), NewPolyRing([I-14.8](#) pg.174)

Chapter I-16

P

I-16.1 Packages

`Packages()`: LIST syntax

Description

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

This function returns the names of the loaded packages as a list of strings. The string “\$user” refers to the user-defined functions defined in the current CoCoA session.

example
`Packages();`
`["$builtin", "$coclib", "$user", "$help", "$io", "$misc"]`

See Also: CoCoA Packages([II-7 pg.297](#)), Supported Packages([II-7.7 pg.299](#))

I-16.2 panel

syntax
[OBSOLETE]

Description

“OBSOLETE”

I-16.3 panels

syntax
[OBSOLETE]

Description

“OBSOLETE”

I-16.4 partitions

syntax

```
partitions(N: INT): LIST
```

Description

This function returns all integer partitions of N, positive integer

example

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

See Also: subsets([I-19.34 pg.246](#)), tuples([I-20.14 pg.259](#))

I-16.5 permutations

syntax

```
permutations(L: LIST): LIST
```

Description

This function computes all permutations of the entries of a list (set). If L has repeated elements it will return repeated elements.

example

```
/**/ permutations(3..5);
[[3, 4, 5], [3, 5, 4], [4, 3, 5], [4, 5, 3], [5, 3, 4], [5, 4, 3]]

/**/ permutations([2, 2, x]);
[[2, 2, x], [2, x, 2], [2, 2, x], [2, x, 2], [x, 2, 2], [x, 2, 2]]

/**/ MakeSet(permutations([2, 2, x]));
[[2, 2, x], [2, x, 2], [x, 2, 2]]
```

See Also: subsets([I-19.34 pg.246](#)), tuples([I-20.14 pg.259](#))

I-16.6 PerpIdealOfForm

syntax

```
PerpIdealOfForm(F: RINGELEM): IDEAL
```

where ‘‘\verb&F&’’ is homogeneous.

Description

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)

/**/ Hilbert(R/It);
H(0) = 1
H(1) = 3
H(2) = 3
H(3) = 1
H(t) = 0   for t >= 4
```

See Also: [InverseSystem\(I-9.30 pg.126\)](#), [DerivationAction\(I-4.11 pg.65\)](#)

I-16.7 pfaffian

— syntax —

```
pfaffian(M: MAT): RINGELEM
```

Description

This function returns the Pfaffian of M.

— example —

```
/**/ Use R := QQ[x,y];
/**/ pfaffian(mat([[0,y],[-y,0]]));
y
```

See Also: [det\(I-4.13 pg.65\)](#)

I-16.8 PkgName

— syntax —

```
PkgName(): STRING
S.PkgName(): STRING
```

where S is the identifier or alias for a package.

Description

This function returns the (long) name of a package. The first form returns “\$coclib” and the second returns the name of the package whose name or alias is S. This function is useful as a shorthand, when S is an alias, for the full name a package.

— example —

```
GB.PkgName();
$gb
-----
$gb.PkgName();
$gb
-----
PkgName();
$coclib
-----
```

I-16.9 PlotPoints

syntax

```
PlotPoints(L: LIST of points)
```

Description

This function outputs the coordinates of the points (with two components) to a file called "CoCoAPlot". See "PlotPointsOn" (I-16.10 pg.194) for outputting to another file.

This result can be plotted using your preferred plotting program. For example, start "gnuplot" and then give it the command

```
plot "CoCoAPlot"
```

to see the plot.

example

```
/**/ PlotPoints([ [X, X^2-X+14] | X In -10..10]);
Plotting points...100%
21 plotted points have been placed in the file CoCoAPlot
```

See Also: ImplicitPlot(I-9.10 pg.115), PlotPointsOn(I-16.10 pg.194)

I-16.10 PlotPointsOn

syntax

```
PlotPointsOn(L: LIST of points, S: STRING)
```

Description

This function is the same as "PlotPoints" (I-16.9 pg.194) with a second argument giving the name of the file to print on.

Note that the last argument is a STRING, the name of the file, and not a DEVICE, as for "print on" (I-16.22 pg.200).

example

```
/**/ PlotPointsOn([ [1/(X+1/2), X^2-X+14] | X In -10..10], "points");
Plotting points...100%
21 plotted points have been placed in the file points

/**/ ImplicitPlotOn(x^2*y - (59/4)*x^2 + 2*x - 1, [-3,3], [0,250], "curve");
Plotting points...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
735 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 "curve"
replot "points"
```

See Also: ImplicitPlot(I-9.10 pg.115), PlotPoints(I-16.9 pg.194)

I-16.11 poincare

syntax

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

Description

(sorry Poincare' for the lower-case: here we follow the naming convention “*single name goes lower-case*”)

Same as “HilbertSeries” (I-8.7 pg.104).

See Also: HilbertSeries(I-8.7 pg.104)

I-16.12 PoincareMultiDeg

syntax

```
PoincareMultiDeg(TAGGED("Quotient"), WM: MAT):TAGGED("$hp.PSeries")
```

Description

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

Same as “HilbertSeriesMultiDeg” (I-8.8 pg.105).

example

```
Use R ::= QQ[x,y,z];

WM := mat([[1,0,0],[1,-1,0]]);
PoincareMultiDeg(R/ideal(Indets())^2, WM);
```

See Also: HilbertSeries(I-8.7 pg.104), HilbertSeriesMultiDeg(I-8.8 pg.105), WeightsMatrix(I-23.2 pg.267)

I-16.13 PoincareShifts

syntax

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

Description

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

Same as “HilbertSeriesShifts” (I-8.9 pg.106).

See Also: dim(I-4.17 pg.67), hilbert(I-8.3 pg.102), HilbertSeries(I-8.7 pg.104), HilbertSeriesShifts(I-8.9 pg.106), HVector(I-8.13 pg.108), multiplicity(I-13.27 pg.170), poincare(I-16.11 pg.195), PoincareMultiDeg(I-16.12 pg.195), WeightsMatrix(I-23.2 pg.267)

I-16.14 PolyAlgebraHom

syntax

```
PolyAlgebraHom(Domain: RING, Codomain: RING, images: LIST): RINGHOM
```

Description

This function creates the homomorphism of (polynomial) algebras from “R” to “S” with the same ring of coefficients. This is uniquely defined by the images of the indeterminates of “R” which are specified by the entries of “images”.

This is a cleaner mathematical implementation of the (OBSOLESCENT) function “image” ([I-9.9 pg.115](#)) in CoCoA-4.

example

```

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

/**/ S := QQ[a];
/**/ phi := PolyAlgebraHom(R, S, [RingElem(S,"a"),1,0]);
/**/ phi(x^2-y);
a^2 -1

/**/ phi := PolyAlgebraHom(R, QQ, [2,1,0]); --> evaluate at [2,1,0]
/**/ phi(x^2-y);
3

```

See Also: [apply\(I-1.12 pg.28\)](#), [CanonicalHom\(I-3.2 pg.37\)](#)

I-16.15 PolyRingHom

syntax

```
PolyRingHom(R: RING, S: RING, CoeffHom: RINGHOM, images: LIST): RINGHOM
```

Description

This function create the homomorphism of (polynomial) algebras between R and S. This is uniquely defined by the images of the indeterminated of R and the homomorphism $\text{CoeffRing}(R)$ into S.

example

```

/**/ R := QQ[x,y];
/**/ S := QQ[a,b,c];
/**/ SmodJ := NewQuotientRing(S, ideal(RingElem(S,"a")^2-1));

/**/ Use SmodJ;
/**/ phi := PolyRingHom(R, SmodJ, CanonicalHom(QQ,SmodJ), [a,b]);
/**/ Use R;
/**/ phi(x);
(a)

```

See Also: [apply\(I-1.12 pg.28\)](#), [CanonicalHom\(I-3.2 pg.37\)](#)

I-16.16 PowerMod

syntax

```
PowerMod(A: INT, B: INT, M: INT): INT
```

Description

This function calculates efficiently an integer power modulo a given modulus. Thus “PowerMod(A, B, M)” is equal to “mod(A^B, M)”, but the former is computed faster. “B” must be non-negative.

example

```
/**/ PowerMod(12345,41041,41041); -- 41041 is a Carmichael number
12345

/**/ PowerMod(123456789,987654321,32003); -- cannot compute 123456789^987654321 directly
2332
```

I-16.17 PreprocessPts

syntax

```
PreprocessPts(Pts: MAT, Toler: MAT): RECORD
PreprocessPtsGrid(Pts: MAT, Toler: MAT): RECORD
PreprocessPtsAggr(Pts: MAT, Toler: MAT): RECORD
PreprocessPtsSubDiv(Pts: MAT, Toler: MAT): RECORD
```

Description

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 tolerances (one for each dimension).

The return value is a record containing two fields: “NewPoints” contains a matrix whose rows represent a list of “*well-separated*” points, and “weights” which contains the number of input points associated to each output point.

There are three underlying algorithms: “Grid” is fast but crude; “Subdiv” works best when the original points are densely packed (so the result will be a small list); finally “Aggr” is best suited to situations where the original points are less densely packed.

The function “PreprocessPts” automatically chooses between “Subdiv” and “Aggr” with the aim of minimising computation time. Note that the “Aggr” and “Subdiv” methods regard the tolerances as being slightly flexible.

For a full description of the algorithms we refer to the paper J.Abbott, C.Fassino, L.Torrente “*Thinning Out Redundant Empirical Data*” (Mathematics in Computer Science, 2007).

example

```
/**/ Pts := matrix([[-1,0],[0,0],[1,0],[99,1],[99,0],[99,-1]]);
/**/ Toler := RowMat([3,3]);
/**/ PreprocessPts(Pts, Toler);
record[NewPoints := matrix(QQ,
  [[99, 0],
   [0, 0]]), weights := [3, 3]]

/**/ PreprocessPts(Pts, RowMat([0.8,0.8]));
record[Points := matrix(QQ,
  [[-1/2, 0],
   [1, 0],
   [99, 1/2],
   [99, -1]], Weights := [2, 1, 2, 1]]

/**/ PreprocessPtsAggr(Pts, RowMat([0.9,0.9])); -- exhibits tolerance flex
```

```
record[Points := matrix(QQ,
  [[0, 0],
  [99, 0]], Weights := [3, 3]]
```

I-16.18 PrimaryDecomposition

syntax

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

Description

This function returns the primary decomposition of the ideal I . Currently it is implemented ONLY for squarefree monomial ideals using the Alexander dual technique. See “FrbPrimaryDecomposition” (I-6.20 pg.88) for monomial ideals.

example

```
/**/ Use R ::= QQ[x,y,z];
/**/ PrimaryDecomposition(***Ideal(xy, yz, zx)***);
[ideal(y, z), ideal(x, z), ideal(x, y)]
```

See Also: FrbPrimaryDecomposition(I-6.20 pg.88), EquiIsoDec(I-5.7 pg.74)

I-16.19 PrimaryPoincare

syntax

```
PrimaryPoincare(I: IDEAL, Q: Ideal): TAGGED("PSeries")
```

Description

Let P be a polynomial ring, M the maximal ideal generated by the indeterminates. This function computes the Hilbert-Poincare’ series of $(P/I)/((Q+I)/I)$, where $(Q+I)/I$ is a primary ideal for M/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 := PrimaryPoincare(I, Q); PS;

/**/ Use S ::= ZZ/(32003)[x,y,z,w];
/**/ I := ***Ideal(x^5 - yz, y^4 - xz^2, xy^3 - zw, x^2z - yw,
/**/ y^2z^2 - w^3, y^3z - x^2w^2, x^3w - z^2, xyw^2 - z^3,
/**/ x^3y^2 - w^2, xz^4 - y^2w^3, yz^5 - xw^5, y^3w^5 - z^7,
/**/ x^2w^7 - z^8, z^9 - yw^8)***;
/**/ Q := ideal(x, y, z);
/**/ PS := PrimaryPoincare(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 := PrimaryPoincare(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 := PrimaryPoincare(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 := PrimaryPoincare(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 := PrimaryPoincare(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\(I-9.23 pg.122\)](#), [TgCone\(I-20.5 pg.254\)](#)

I-16.20 PrimitiveRoot

— syntax —

```
PrimitiveRoot(P: INT): INT
```

Description

Find a primitive root modulo the prime “P”, i.e. a generator of the cyclic multiplicative group of non-zero integers mod “P”.

Currently, the function produces the least positive primitive root.

— example —

```

/**/ PrimitiveRoot(17551561);
97
/**/ PrimitiveRoot(4111);
12;

```

See Also: [IsPrime\(I-9.52 pg.134\)](#)

I-16.21 print

— syntax —

```
print E_1, ..., E_n
```

Description

This command displays the value of each of the expressions “E_i”. To insert a newline write “\n”.

The similar command “println” (I-16.25 pg.201) is equivalent to “print” with a final newline.

example

```
/**/ for I := 1 To 10 Do print I^2, " "; endfor;
1 4 9 16 25 36 49 64 81 100

/**/ print "a\nb";
a
b
```

See Also: print on(I-16.22 pg.200), println(I-16.25 pg.201), format(I-6.15 pg.86), Latex(I-12.2 pg.147), StarPrint, StarSprint(I-19.24 pg.241)

I-16.22 print on

syntax

```
print E: OBJECT on D: DEVICE
```

Description

This command prints the value of expression E to the device D. 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.187) or “OpenOString” (I-15.6 pg.188).

example

```
/**/ D := OpenOFile("my-test"); -- open "my-test" for output from CoCoA
/**/ Print "hello world" On D; -- print string into "mytest"
/**/ close(D); -- close the file
```

See “OpenOFile” (I-15.5 pg.187) for an example using output strings. For printing to the CoCoA window, just use “print E” which is short for “print E On DEV.OUT”.

See Also: Introduction to IO(II-6.1 pg.291), OpenIFile(I-15.2 pg.185), OpenOFile(I-15.5 pg.187), OpenIString(I-15.3 pg.186), OpenOString(I-15.6 pg.188), print(I-16.21 pg.199), println(I-16.25 pg.201)

I-16.23 PrintBettiDiagram

syntax

```
PrintBettiDiagram(M: IDEAL or (QUOTIENT)RING or MODULE or Resolution)
```

Description

This function prints the (“Macaulay-style”) Betti diagram for “M”.

example

```
/**/ Use R ::= QQ[t,x,y,z];
/**/ I := ideal(x^2-y*t, x*y-z*t, x*y);
/**/ RES := res(I);
/**/ PrintRes(RES);
0 --> R(-5)^2 --> R(-4)^4 --> R(-2)^3
/**/ PrintBettiDiagram(RES);
      0      1      2
```

```

-----
2:    3    -    -
3:    -    4    2
-----
Tot:   3    4    2

```

See Also: [PrintRes\(I-16.26 pg.202\)](#), [PrintBettiMatrix\(I-16.24 pg.201\)](#)

I-16.24 *PrintBettiMatrix*

syntax

```
PrintBettiMatrix(M: IDEAL|MODULE|Resolution)
```

Description

This function returns the Betti matrix for M.

example

```

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

```

See Also: [PrintRes\(I-16.26 pg.202\)](#), [PrintBettiDiagram\(I-16.23 pg.200\)](#)

I-16.25 *println*

syntax

```
println E_1,...,E_n
PrintLn E_1,...,E_n
```

Description

This command is equivalent to “**print**” ([I-16.21 pg.199](#)) with a final newline; in other words, it prints the values of its arguments, then moves the cursor to the next line.

example

```

/**/ for i := 1 to 3 do print i; endfor;
123

/**/ for i := 1 to 3 do println i; endfor;
1
2
3

```

See Also: [print\(I-16.21 pg.199\)](#), [print on\(I-16.22 pg.200\)](#)

I-16.26 PrintRes

syntax

```
PrintRes(M)
```

Description

This function prints the minimal free resolution of “M”. (see “res” (I-18.23 pg.220)).

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-16.23 pg.200), PrintBettiMatrix(I-16.24 pg.201), res(I-18.23 pg.220)

I-16.27 product

syntax

```
product(L: LIST): OBJECT
product(L: LIST, O: OBJECT): OBJECT
```

Description

This function returns the product of the objects in the list L and the optional last argument.

example

```
/**/ Use R ::= QQ[x,y];
/**/ product([3, x, y^2]);
3*x*y^2

/**/ product(1..40) = factorial(40);
true

/**/ product([]); -- gives 1 of type INT
1
/**/ product([], y);
y
/**/ product([3, x], y);
3*x*y
```

See Also: Algebraic Operators(II-3.2 pg.283), sum(I-19.36 pg.247)

I-16.28 *protect*

— syntax —

```
protect X;
protect X : reason;
  where reason: STRING
```

Description

This command protects the variable “X” from being assigned to. Attempting to assign to it will produce an error; if a “reason” (STRING) was given it is printed in the error message.

— example —

```
/**/ MaxSize := 99;
/**/ protect MaxSize : "size limit for fast computation";
-- /**/ MaxSize := 1000; --> !!! ERROR !!!
ERROR: Cannot set "MaxSize" (size limit for fast computation)

/**/ unprotect MaxSize; --> remove protection, X may be assigned to now
/**/ MaxSize := 1000; --> OK
```

See Also: `unprotect`([I-21.2](#) pg.261)

I-16.29 *PthRoot*

— syntax —

```
PthRoot(X: RINGELEM): RINGELEM
```

Description

This function returns the p-th root of a polynomial over a finite field. If no p-th root exists then an error is signalled. p is the characteristic of the field.

— example —

```
/**/ Use R ::= ZZ/(7)[x,y];
/**/ F := x^7-y^14+3;
/**/ PthRoot(F);
-y^2+x+3
```

See Also: `IsFiniteField`([I-9.43](#) pg.130), `IsPthPower`([I-9.54](#) pg.135)

Chapter I-17

Q

I-17.1 QQ

syntax

QQ

Description

This system variable is constant; its value is the field of rationals. Its name is protected so that it cannot be re-assigned to any other value.

example

```
/**/ Use QQ;  
  
/**/ type(5);  
INT  
/**/ type(RingElem(QQ, 5));  
RINGELEM
```

See Also: ZZ([I-25.4](#) pg.[274](#)), NewQuotientRing([I-14.9](#) pg.[174](#))

I-17.2 quit

syntax

quit

Description

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

See Also: ciao([I-3.11](#) pg.[41](#))

I-17.3 QuotientBasis

syntax

QuotientBasis(I: IDEAL): LIST

Description

This function determines a vector space basis (of power products) for the quotient space associated to a zero-dimensional ideal. That is, if R is a polynomial ring with field of coefficients k , and I is a zero-dimensional ideal in R then $\text{QuotientBasis}(I)$ is a set of power products forming a k -vector space basis of R/I .

The actual set of power products chosen depends on the term ordering in the ring R : the power products chosen are those not divisible by the leading term of any member of the reduced Groebner basis of I (and consequently they form a factor-closed set).

example

```

/**/ Points := [[Rand(-9,9) | N In 1..3] | S In 1..25];
/**/ Use P := QQ[x,y,z];
/**/ I := IdealOfPoints(P, mat(QQ, Points));
/**/ QuotientBasis(I);
[1, z, z^2, z^3, z^4, y, y*z, y*z^2, y*z^3, y^2, y^2*z, y^2*z^2, y^3, x,
x*z, x*z^2, x*z^3, x*y, x*y*z, x*y*z^2, x*y^2, x^2, x^2*z, x^2*y, x^3]

/**/ Use P := QQ[x,y,z], Lex;
/**/ I := IdealOfPoints(P, mat(QQ, Points));
/**/ QuotientBasis(I);      -- power products underneath the Lex reduced GBasis
[1, z, z^2, z^3, z^4, z^5, z^6, z^7, z^8, z^9, z^10, z^11, z^12, y, y*z,
y*z^2, y*z^3, y*z^4, y*z^5, y*z^6, y^2, y^2*z, y^2*z^2, y^2*z^3, y^3]
```

See Also: [IdealOfPoints\(I-9.4 pg.112\)](#), [IsFactorClosed\(I-9.41 pg.130\)](#)

I-17.4 QZP

syntax

```

QZP(F: RINGELEM): RINGELEM
QZP(F: LIST of POLY): LIST of POLY
QZP(I: IDEAL): IDEAL
```

Description

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

The functions “QZP” and “ZPQ” ([I-25.3 pg.274](#)) 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.274](#)) does the reverse, mapping a polynomial with finite field coefficients into one with rational (actually, integer) coefficients. The function “ZPQ” ([I-25.3 pg.274](#)) is not uniquely defined mathematically, and currently for each coefficient the least non-negative equivalent integer is chosen. Users should not rely on this choice, though any change will be documented.

example

```

Use R := QQ[x,y,z];
F := 1/2*x^3 + 34/567*x*y*z - 890; -- a poly with rational coefficients
Use S := ZZ/(101)[x,y,z];
QZP(F);                                -- compute its image with coeffs in ZZ/(101)
-50x^3 - 19xyz + 19
-----
G := It;
Use R;
ZPQ(G);                                -- now map that result back to QQ[x,y,z]
-- it is NOT the same as F...
```

```

51x^3 + 82xyz + 19
-----
H := It;
F - H;                -- ... but the difference is divisible by 101
-101/2x^3 - 46460/567xyz - 909
-----
Use S;
QZP(H) - G;           -- F and H have the same image in ZZ/(101)[x,y,z]
0
-----

```

See Also: [Accessing Other Rings\(III-9.7 pg.330\)](#), [BringIn\(I-2.10 pg.35\)](#), [image\(I-9.9 pg.115\)](#)

Chapter I-18

R

I-18.1 radical

syntax

```
radical(I: IDEAL): IDEAL
```

Description

This function computes the radical of I 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

```
/**/ Use R ::= QQ[x,y];
/**/ I := ideal(x,y)^3;
/**/ radical(I);
ideal(y, x)
```

See Also: [IsInRadical\(I-9.47 pg.132\)](#), [EquiIsoDec\(I-5.7 pg.74\)](#), [RadicalOfUnmixed\(I-18.2 pg.209\)](#)

I-18.2 RadicalOfUnmixed

syntax

```
RadicalOfUnmixed(I: IDEAL): IDEAL
```

Description

This function computes the radical of an unmixed ideal.

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

example

```
/**/ Use R ::= QQ[x,y];
/**/ I := ideal(x^2 - y^2 - 4*x + 4*y, x - 2);
/**/ RadicalOfUnmixed(I);
ideal(x^2 -y^2 -4*x +4*y, x -2, y -2)
/**/ interreduced(gens(It)); -- the result may not be in its simplest form
[y -2, x -2]
```

See Also: [EquiIsoDec\(I-5.7 pg.74\)](#), [radical\(I-18.1 pg.209\)](#)

I-18.3 random

syntax

```
random(X: INT, Y: INT): INT
```

Description

The function returns a random integer between X and Y, inclusive. The range $|X - Y|$ should be less than 2^{33} to assure a more random distribution.

NB: every time you restart CoCoA the sequence of random numbers will be the same (as happens in many programming languages). If you want better randomness, see “seed” ([I-19.4 pg.230](#)).

example

```
/**/ random(1,100);
6

/**/ random(-10^4,0);
-3263
```

See Also: [randomize\(I-18.4 pg.210\)](#), [randomized\(I-18.5 pg.211\)](#), [seed\(I-19.4 pg.230\)](#)

I-18.4 randomize

syntax

```
Randomize(V: RINGELEM): RINGELEM
```

Description

***** NOT YET IMPLEMENTED: use random *****

This function replaces the coefficients of terms of the polynomial contained in V with randomly generated coefficients. The result is stored in V, overwriting the original polynomial.

Note: It is possible that some coefficients will be replaced by zeroes, i.e., some terms from the original polynomial may disappear in the result.

The similar function “[randomized](#)” ([I-18.5 pg.211](#)) performs the same operation, but returns the randomized polynomial without modifying the argument.

NB: every time you restart CoCoA the sequence of random numbers will be the same (as in other programming languages). If you want total randomness read “seed” ([I-19.4 pg.230](#)).

example

```
Use R := QQ[x];
F := 1+x+x^2;
Randomized(F);
-2917104644x^2 + 3623608766x - 2302822308
-----
F;
x^2 + x + 1
-----
Randomize(F);
F;
-1010266662x^2 + 1923761602x - 4065654277
-----
```

See Also: [random\(I-18.3 pg.210\)](#), [randomized\(I-18.5 pg.211\)](#), [seed\(I-19.4 pg.230\)](#)

I-18.5 randomized

syntax

```
Randomized(F: RINGELEM): RINGELEM
Randomized(F: INT): INT
```

Description

***** NOT YET IMPLEMENTED: use random *****

This function with a polynomial argument returns a polynomial obtained by replacing the coefficients of F with randomly generated coefficients. The original polynomial, F , is unaffected. With an integer argument, it returns a random integer.

Note: It is possible that some coefficients will be replaced by zeroes, i.e., some terms from the original polynomial may disappear in the result.

The similar function “[randomize\(I-18.4 pg.210\)](#)” performs the same operation, but returns NULL and modifies the argument.

NB: every time you restart CoCoA the sequence of random numbers will be the same (as in other programming languages). If you want total randomness read “[seed\(I-19.4 pg.230\)](#)”.

example

```
Use R ::= QQ[x];
F := 1 + x + x^2;
Randomized(F);
-2917104644x^2 + 3623608766x - 2302822308
-----
F;
x^2 + x + 1
-----
Randomized(23);
-3997312402
-----
Use R ::= ZZ/(7)[x,y];
Randomized(x^2 + 3x - 5);
3x^2 + 2x - 2
-----
```

See Also: [random\(I-18.3 pg.210\)](#), [randomize\(I-18.4 pg.210\)](#), [seed\(I-19.4 pg.230\)](#)

I-18.6 rank

syntax

```
rank(M: MAT): INT
rank(M: MODULE): INT
```

Description

This function computes the rank of M . For a module M this is defined as the vector space dimension of the subspace generated by the generators of M over the quotient field of the base ring – contrast this with the function “[NumCompts\(I-14.26 pg.181\)](#)” which simply counts the number of components the module has.

example

```

/**/ Use R := QQ[x,y,z];
/**/ rank(IdentityMat(R, 4));
4

Rank(Module([x,y,z,0])); --***WORK IN PROGRESS***
1
-----
Rank(Module([[1,2,3],[2,4,6]])); --***WORK IN PROGRESS***
1
-----
Rank(Module([[1,2,3],[2,5,6]])); --***WORK IN PROGRESS***
2
-----

```

I-18.7 RatReconstructByContFrac, RatReconstructByLattice

syntax

```

RatReconstructByContFrac(X: INT, M: INT): RECORD
RatReconstructByContFrac(X: INT, M: INT, threshold: INT): RECORD
RatReconstructByLattice(X: INT, M: INT): RECORD
RatReconstructByLattice(X: INT, M: INT, threshold: INT): RECORD

```

Description

These functions attempt to reconstruct rational numbers from a modular image “ $X \bmod M$ ”. The algorithms are fault-tolerant: they will succeed provided that “ X ” is correct modulo a sufficiently large factor of “ M ”.

The result is a record: the boolean field “failed” is “true” if no “convincing” result was found; otherwise it is “false”, and a second field, called “ReconstructedRat”, contains the value reconstructed.

An optional third argument, “threshold”, determines what “convincing” means: a higher value gives a more reliable answer, but may need a larger modulus before the answer is found.

There are two different underlying heuristic algorithms: a faster one based on continued fractions, and a slower one based on 2-dimensional lattice reduction. See arXiv: “<http://arxiv.org/abs/1303.2965>”

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-18.8 pg.212), CRT(I-3.46 pg.56)

I-18.8 RatReconstructWithBounds

syntax

```

RatReconstructWithBounds(e: INT, P: INT, Q: INT, res: LIST of INT, mod: LIST of INT): RECORD

```

Description

This function attempts to reconstruct a rational number from a collection of residue-modulus pairs “(res[i],mod[i])”. The function also requires the input of three bounds: “e” is an upper bound on the number of bad moduli, and “P” and “Q” are upper bounds for (respectively the numerator and denominator of) the rational to be reconstructed.

The result is a record: the boolean field “failed” is “true” if no result exists; otherwise it is “false”, and a second field, called “ReconstructedRat”, contains the value reconstructed.

example

```
/**/ moduli := [11,13,15,17,19];
/**/ residues := [-2, -5, 0, 7, 4];
/**/ RatReconstructWithBounds(1,10,10,residues,moduli);
record[ReconstructedRat := 1/5, failed := false]

/**/ RatReconstructWithBounds(0,10,10,residues,moduli);
record[failed := true]
```

See Also: CRT(I-3.46 pg.56), RatReconstructByContFrac, RatReconstructByLattice(I-18.7 pg.212)

I-18.9 RealRootRefine

syntax

```
RealRootRefine(Root: RECORD, Precision: RAT): RECORD
```

Description

This function computes a refinement of a real root of a univariate polynomial over QQ to the desired precision (width of isolating interval). The starting root must be a record produced by “RealRoots” (I-18.10 pg.213).

example

```
/**/ 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-18.10 pg.213), RealRootsApprox(I-18.11 pg.214), RootBound(I-18.37 pg.226)

I-18.10 RealRoots

syntax

```
RealRoots(F: RINGELEM): LIST
RealRoots(F: RINGELEM, Precision: RAT): LIST
RealRoots(F: RINGELEM, Precision: RAT, Interval:[RAT, RAT]): LIST
```

Description

This function computes isolating intervals for the real roots of a univariate polynomial over 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. A third field (called CoeffList) has an obscure meaning.

An optional second argument specifies the maximum width an isolating interval may have. An optional third argument specifies a closed interval in which to search for roots.

The interval represented by a root record may be refined by using the function “**RealRootRefine**” (I-18.9 pg.213). The function “**RealRootsApprox**” (I-18.11 pg.214) may be more useful to you: it produces rational approximations to the real roots (but these cannot later be refined).

— example —

```

/**/ indent(RealRoots(x^2-2));
[
  record[CoeffList := [-1, 0, 2], inf := -4, sup := 0],
  record[CoeffList := [1, 0, -2], inf := 0, sup := 4]
]

/**/ RR := RealRoots((x^2-2)*(x-1), 10^(-5));
/**/ FloatStr(RR[1].inf); -- left end of interval
-1.414213562*10^0

/**/ FloatStr(RR[1].sup); -- right end of interval
-1.414213561*10^0

/**/ RR := RealRoots(x^2-2, 10^(-20), [0, 2]);

/**/ RR[1].inf; -- incomprehensible
60153992292001127886258443119406264231/42535295865117307932921825928971026432
/**/ FloatStr(RR[1].inf, 20); -- comprehensible
1.4142135623730950488*10^0

```

See Also: **RealRootRefine**(I-18.9 pg.213), **RealRootsApprox**(I-18.11 pg.214), **RootBound**(I-18.37 pg.226)

I-18.11 RealRootsApprox

— syntax —

```

RealRootsApprox(F: RINGELEM): LIST
RealRootsApprox(F: RINGELEM, Precision: RAT): LIST
RealRootsApprox(F: RINGELEM, Precision: RAT, Interval:[RAT, RAT]): LIST

```

Description

This function computes rational approximations to the real roots of a univariate polynomial (with rational coefficients).

An optional second argument specifies the maximum separation between the approximations produced and the corresponding exact root. An optional third argument specifies a closed interval in which to search for roots.

— example —

```

/**/ RealRootsApprox(x^2-2);
[-3037000499/2147483648, 3037000499/2147483648]

/**/ RR := RealRootsApprox(x^2-2, 10^(-15), [0, 2]);
/**/ RR;
[6521908912666391107/4611686018427387904]

/**/ FloatStr(RR[1], 15);
1.41421356237310*10^0

```

See Also: **RealRoots**(I-18.10 pg.213), **RootBound**(I-18.37 pg.226)

I-18.12 record

syntax

```
record[F_1 := OBJECT,..., F_n := OBJECT]
  where each F_i is a field name
  returns RECORD
```

Description

This constructor creates a record with fields called “F_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-18.13 pg.215), fields(I-6.5 pg.81)

I-18.13 record field selector

syntax

```
R.FieldName
R["FieldName"]
  where ‘\verb&R&’ is a RECORD
```

Description

A record is a data structure containing named entries. They are created using the command “**record**” (I-18.12 pg.215). Each entry may be selected using the “dot operator”, or equivalently a string index.

example

```
/**/ rec := record[name := "David", year := 1961];
/**/ rec.name;
David

/**/ rec.year := 1849; --> change value of a field
/**/ rec.surname := "Copperfield"; --> create a new field
/**/ rec["year"]; -- alternative syntax
1849

/**/ foreach F in fields(rec) do print rec[F]; endforeach;
DavidCopperfield1849
```

See Also: record(I-18.12 pg.215)

I-18.14 ReducedGBasis

syntax

```
ReducedGBasis(I: IDEAL): LIST of RINGELEM
ReducedGBasis(I: MODULE): LIST of MODULEELEM
```

Description

This function returns a list whose components form a reduced Groebner basis for the ideal (or module) “I” with respect to the term-ordering of the polynomial ring of “I”.

example

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

See Also: GBasis([I-7.1 pg.91](#))

I-18.15 ref

syntax

```
ref X
  where X is the identifier of a CoCoA variable.
```

Description

The keyword “**ref**” is used to pass a parameter “by reference” to a function which may modify its value (e.g. “**append**” ([I-1.11 pg.27](#))). 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.60](#))

I-18.16 RefineGCDFreeBasis

syntax

```
RefineGCDFreeBasis(B: LIST of INT, N: INT): LIST of INT
```

Description

This function computes a refined GCD free basis by adjoining a given integer to it. The value returned is [NewB, N2] where NewB is the refined basis and N2 is the part of N coprime to every element of B.

example

```
/**/ B := GCDFreeBasis([Fact(10), binomial(20,10)]); B;
[14175, 4, 46189]

/**/ RefineGCDFreeBasis(B, 15);
[[7, 3, 5, 4, 46189], 1]
```

See Also: GCDFreeBasis(I-7.5 pg.93)

I-18.17 Reg, Reg5

syntax

```
Reg(M: IDEAL or TAGGED("Quotient")): INT
```

Description

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

These functions computes the Castelnuovo-Mumford regularity of a module. The implementation of “Reg” is just implementation of the definition (i.e. computes the resolution). The implementation of “Reg5” implements Bermejo-Gimenez Algorithm and uses functions from the CoCoAServer (linked with the Frobbly library). This algorithm has a chance when the computation of the resolution is unfeasable.

Note: this is different from “RegularityIndex” (I-18.18 pg.218), the regularity of a Hilbert Function.

example

```
Use R ::= QQ[x,y,z];
I := ideal(x^3, y^2);
Res(I);
0 --> R(-5) --> R(-2)(+)R(-3)
-----
BettiDiagram(I);
      0      1
-----
2:      1      -
3:      1      -
4:      -      1
-----
Tot:      2      1
-----
Reg(I);
4
-----
Reg(R/I);
3
-----
Reg5(I);
```

4

See Also: [res\(I-18.23 pg.220\)](#), [PrintRes\(I-16.26 pg.202\)](#), [PrintBettiDiagram\(I-16.23 pg.200\)](#), [PrintBettiMatrix\(I-16.24 pg.201\)](#), [RegularityIndex\(I-18.18 pg.218\)](#)

I-18.18 RegularityIndex

syntax

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

Description

This function computes the regularity index of a Hilbert function. The input might be expressed as a Hilbert function or as the corresponding Hilbert series (computed with standard weights).

example

```
/**/ Use R ::= QQ[x,y,z];
/**/ Quot := R/ideal(x^3, y^2);
/**/ HilbertFn(Quot);
H(0) = 1
H(1) = 3
H(2) = 5
H(t) = 6 for t >= 3
/**/ RegularityIndex(HilbertFn(Quot));
3
/**/ RegularityIndex(HilbertSeries(Quot));
3
```

See Also: [hilbert\(I-8.3 pg.102\)](#), [HilbertFn\(I-8.5 pg.103\)](#), [HilbertSeries\(I-8.7 pg.104\)](#), [poincare\(I-16.11 pg.195\)](#)

I-18.19 RelNotes

syntax

```
RelNotes()
```

Description

This function prints the release notes of the version you are running.

example

```
RelNotes();
```

I-18.20 ReloadMan

syntax

```
ReloadMan()
```

Description

This function reloads the xml source of the manual “CoCoAHelp.xml” (in directory “CoCoAManual”) and recreates the internal manual index in a running CoCoA-5.

It is necessary when a substantial change has been made to “CoCoAHelp.xml” (such as a new entry been added) and the index needs updating. If the change is in the description of an existing entry (so the internal index is still valid) there is no need to reload the manual: the description is always searched in the current file.

example

```
/**/ ReloadMan();
```

I-18.21 *remove*

syntax

```
remove(ref L:LIST, N: INT)
```

Description

This function removes the “N”-th component from “L”; it changes the value of “L”. Use the function “WithoutNth” (I-23.4 pg.268) 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: insert(I-9.24 pg.123), WithoutNth(I-23.4 pg.268)

I-18.22 *repeat*

syntax

```
repeat C until B
repeat C endrepeat
(where C is a sequence of commands and B is BOOL)
```

Description

In the first form, the command sequence “C” is repeated until “B” evaluates to “false”. Unlike the “while” (I-23.3 pg.268) command, “C” is executed at least once. Note that there is no “endrepeat” following “B”.

example

```
/**/ Define GCD_Euclid(A, B)
/**/ Repeat
/**/ R := mod(A, B);
/**/ A := B;
/**/ B := R;
/**/ Until B = 0;
```

```

/**/      Return A;
/**/      EndDefine;

/**/  GCD_Euclid(6,15);
3

/**/  N := 0;
/**/  repeat
/**/    N := N+1;
/**/    PrintLn N;
/**/    If N > 5 Then Break; EndIf;
/**/  endrepeat;
1
2
3
4
5

```

See Also: for([I-6.13](#) pg.84), foreach([I-6.14](#) pg.85), while([I-23.3](#) pg.268)

I-18.23 res

syntax

```

res(M: IDEAL): LIST
res(M: MODULE): LIST

```

Description

This function returns the minimal free resolution of “M”. “res” only works in the homogeneous context, and the coefficient ring must be a field.

NOTE: the current implementation (CoCoA-5.1.0) is very naive so it might be very slow (better slow than nothing?).

example

```

/**/  Use R ::= QQ[x,y,z];
/**/  I := ideal(x, y, z^2);
/**/  PrintRes(R/I);
0 --> R(-4) --> R(-2)(+)R(-3)^2 --> R(-1)^2(+)R(-2) --> R
/**/  indent(Res(R/I),2);
[
  QuotientRing(RingDistrMPolyClean(QQ, 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\(I-16.23 pg.200\)](#), [PrintBettiMatrix\(I-16.24 pg.201\)](#), [PrintRes\(I-16.26 pg.202\)](#)

I-18.24 Reset

syntax

OBSOLETE

Description

“OBSOLETE”

I-18.25 ResetPanels

syntax

OBSOLETE

Description

“OBSOLETE”

I-18.26 resultant

syntax

resultant(F: RINGELEM, G: RINGELEM, X: RINGELEM): RINGELEM

Description

This function returns the resultant of the polynomials F and G with respect to the indeterminate X.

example

```
/**/ Use R ::= QQ[p,q,x];
/**/ F := x^3+p*x-q;   G := deriv(F, x);
/**/ resultant(F, G, x);
4*p^3 +27*q^2
```

See Also: [discriminant\(I-4.18 pg.68\)](#), [sylvester\(I-19.39 pg.248\)](#)

I-18.27 return

syntax

```
return
return E
```

Description

This command is used to exit from a procedure/function. The latter 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 returns a value (procedure). To exit from a loop see “**break**” ([I-2.9 pg.35](#)).

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]

---- mixing function/procedure returns is not allowed
-- /**/ Define AFailingExample(X)
-- /**/   If X=1 Then Return 123456;
-- /**/   Else Return; -- ..... --> !!! ERROR !!!
ERROR: Inside a function definition all Return statements must be
      either with or without an expression
      Else Return;
      ~~~~~

```

See Also: [break\(I-2.9 pg.35\)](#), [define\(I-4.4 pg.60\)](#)

I-18.28 reverse, reversed

syntax

```

reverse(ref L: LIST)
reversed(L: LIST): LIST

```

Description

The the function “**reverse**” reverses the order of the elements of the list in “L”; it changes the value of “L” and returns nothing. The function “**reversed**” returns the reversed list without changing “L”.

example

```

/**/ L := [1,2,3,4];
/**/ reverse(ref L);
/**/ L; -- L has been modified
[4, 3, 2, 1]

/**/ M := [1,2,3,4];
/**/ reversed(M); -- the reversed list is returned
[4, 3, 2, 1]

/**/ M; -- M has not been modified
[1, 2, 3, 4]

```

See Also: [sort\(I-19.13 pg.235\)](#), [sorted\(I-19.15 pg.236\)](#)

I-18.29 RevLexMat

syntax

```

RevLexMat(N: INT): MAT

```

Description

This function return the matrix defining a standard ordering (which is not a term-ordering!).

example

```
/**/ RevLexMat(3);
matrix([
  [0, 0, -1],
  [0, -1, 0],
  [-1, 0, 0]
])
```

See Also: OrdMat([I-15.10](#) pg.189), Orderings([III-9.5](#) pg.329), StdDegRevLexMat([I-19.27](#) pg.243), StdDegLexMat([I-19.26](#) pg.242), LexMat([I-12.6](#) pg.149), XelMat([I-24.1](#) pg.271)

I-18.30 RingElem

syntax

```
RingElem(R: RING, E: RINGELEM): RINGELEM
RingElem(R: RING, E: INT): RINGELEM
RingElem(R: RING, E: RAT): RINGELEM
RingElem(R: RING, E: STRING): RINGELEM
RingElem(R: RING, E:[STRING, INT, .., INT]): RINGELEM
```

Description

This function converts the expression E into a RINGELEM in R, if possible.

Can be used for mapping ring element between rings when a “CanonicalHom” ([I-3.2](#) pg.37) exists. For other homomorphisms search see “RINGHOM” ([III-10](#) pg.333).

example

```
/**/ Use P ::= QQ[x,y,z];
/**/ -- RINGELEM (via CanonicalHom)
/**/ F := 2*x-3;
-- /**/ F/LC(F); -- !!! ERROR !!! LC(F) in CoeffRing(P)
/**/ F/RingElem(P,LC(F));
x +1
-- /**/ 1/x; -- !!! ERROR !!! x in P is not invertible
/**/ K := NewFractionField(P);
/**/ 1/RingElem(K, x); -- x in K is invertible
1/x

/**/ Use P ::= ZZ/(5)[x,y,z];
/**/ -- INT and RAT
/**/ RingElem(P, 7);
2
/**/ RingElem(P, 3/2);
-1

/**/ -- STRING (indet name, symbol)
/**/ S ::= QQ[x,y,z[1..4,3..7]];
/**/ 7*RingElem(P, "x"); --> x as an element of P
2*x
/**/ 7*RingElem(S, "x"); --> x as an element of S
7*x
```

```
/**/ 7*RingElem(S, ["z",2,5]);
7*z[2,5]
```

See Also: [RingOf\(I-18.31 pg.224\)](#), [AsINT\(I-1.14 pg.29\)](#), [AsRAT\(I-1.15 pg.29\)](#), [IndetName\(I-9.18 pg.119\)](#), [IndetSubscripts\(I-9.20 pg.121\)](#), [CanonicalHom\(I-3.2 pg.37\)](#)

I-18.31 RingOf

syntax

```
RingOf(E: RINGELEM|IDEAL|MAT|MODULE): RING
```

Description

This function returns the ring on which the object E is defined.

NB A ring contains many information and two separate rings, even when defined with the same commands, are not "equal". When a ring is printed only a few informations are shown, so different rings might look the same.

example

```
/**/ Use R := QQ[x,y,z];
/**/ I := ideal(x,y);
/**/ RingOf(I);
RingDistrMPolyClean(QQ, 3)

/**/ 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
RingDistrMPolyClean(QQ, 3)
/**/ indets(RingOf(F));
[a, b, c]
```

See Also: [CurrentRing\(I-3.47 pg.56\)](#), [RingSet\(I-18.34 pg.225\)](#), [BaseRing\(I-2.1 pg.31\)](#)

I-18.32 RingQQ

syntax

```
RingQQ(): RING
```

Description

This function returns the ring of rationals. It is particularly useful when you want to use "QQ" (which is a pre-defined top-level variable) inside a function.

NB calling "RingQQ" twice gives the same identical ring, whereas calling "NewPolyRing" ([I-14.8 pg.174](#)) or "NewFractionField" ([I-14.1 pg.171](#)) return each time a new ring.

example

```
/**/ Two := RingElem(RingQQ(), 2); Two;
2
/**/ type(Two);
```

```
RINGELEM;
/**/ RingOf(Two) = RingQQ();
true
```

See Also: TopLevel(I-20.9 pg.256), RingQQt(I-18.33 pg.225), RingZZ(I-18.35 pg.226)

I-18.33 RingQQt

syntax

```
RingQQt(N: INT): RING
```

Description

This function returns a polynomial ring over “QQ” with indeterminates $t[1] \dots t[N]$. In particular “RingQQt(1)” is the polynomial ring in which Hilbert polynomials are defined.

NB calling “RingQQt(5)” twice gives the same identical ring, whereas calling “NewPolyRing(RingQQ(), SymbolRange("t", 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-20.9 pg.256), RingQQ(I-18.32 pg.224), RingZZ(I-18.35 pg.226)

I-18.34 RingSet

syntax

```
RingSet(E: LIST|RINGELEM|IDEAL|MAT|MODULE|MODULEELEM): LIST of RING and TYPE
```

Description

This function returns the list of the RINGs (or types, if not dependent from a RING) on which the object E is dependent. Similar to “RingOf” (I-18.31 pg.224), this function also works on lists and returns the set of ring environments of all entries. ...needless to say that it may be quite slow on big inputs!

example

```
/**/ Use R ::= QQ[x,y,z];
/**/ L1 := [x, y];
/**/ L2 := [x, y, 0, 5/4];
/**/ Z3 := NewRingFp(3);
/**/ Use S ::= Z3[a,b];
/**/ RingSet(L1);
[RingDistrMPolyClean(QQ, 3)]

/**/ RingSet(L2);
[RingDistrMPolyClean(QQ, 3), INT, RAT]
```

```
/**/ RingSet([L2, a+b]);
[RingDistrMPolyClean(QQ, 3), INT, RAT, RingDistrMPolyClean(FFp(3), 2)]
```

See Also: [CurrentRing\(I-3.47 pg.56\)](#), [RingOf\(I-18.31 pg.224\)](#)

I-18.35 RingZZ

syntax

```
RingZZ(): RING
```

Description

This function returns the ring of integers. It is particularly useful when you want to use “ZZ” (which is a pre-defined top-level variable) inside a function.

NB calling “RingZZ” twice gives the same identical ring, whereas calling “NewPolyRing” ([I-14.8 pg.174](#)) or “NewFractionField” ([I-14.1 pg.171](#)) return each time a new ring.

example

```
/**/ Two := RingElem(RingZZ(), 2); Two;
2
/**/ type(Two);
RINGELEM;
/**/ RingOf(Two) = RingZZ();
true
/**/ RingOf(Two) = RingQQ();
false
```

See Also: [TopLevel\(I-20.9 pg.256\)](#), [RingQQt\(I-18.33 pg.225\)](#), [RingQQ\(I-18.32 pg.224\)](#)

I-18.36 RMap

syntax

```
RMap(L: LIST): TAGGED("RMap")
```

Description

See “image” ([I-9.9 pg.115](#)).

I-18.37 RootBound

syntax

```
RootBound(F: RINGELEM): INT
```

Description

This function computes a bound on the absolute values of the complex roots of a univariate polynomial over QQ.

example

```
/**/ Use R ::= QQ[x,y,z];
/**/ RootBound(x^2-2);
4
```

See Also: [RealRootRefine\(I-18.9 pg.213\)](#), [RealRoots\(I-18.10 pg.213\)](#)

I-18.38 *round*

syntax

```
round(X: RAT): INT
```

Description

This function rounds a rational to the nearest integer; halves are rounded towards zero.

example

```
/**/ round(4.56);
5

/**/ round(-1/2);
0
```

See Also: [num\(I-14.24 pg.180\)](#), [den\(I-4.7 pg.62\)](#), [floor\(I-6.12 pg.84\)](#), [ceil\(I-3.5 pg.38\)](#)

I-18.39 *RowMat*

syntax

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

Description

This function return the matrix whose only row consists of the elements of the list L.

example

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

/**/ RowMat(QQ,[5,6,7]);
matrix([
  [5, 6, 7]
])
```

See Also: [BlockMat\(I-2.6 pg.33\)](#), [DiagMat\(I-4.15 pg.66\)](#), [ColMat\(I-3.24 pg.46\)](#)

Chapter I-19

S

I-19.1 saturate

— syntax —

```
saturate(I: IDEAL, J: IDEAL): IDEAL
```

Description

This function returns the saturation of I with respect to J: the ideal of polynomials F such that $F^d G$ is in I for all G in J^d for some positive integer d.

The coefficient ring must be a field.

— example —

```
/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(x-z, y-2*z);
/**/ J := ideal(x-2*z, y-z);
/**/ K := intersection(I, J); -- ideal of two points in the
                             -- projective plane
/**/ L := intersection(K, ideal(x,y,z)^3); -- add an irrelevant component
/**/ Hilbert(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\(I-3.25 pg.47\)](#), [HColon\(I-8.1 pg.101\)](#), [HSaturation\(I-8.12 pg.107\)](#)

I-19.2 ScalarProduct

— syntax —

```
ScalarProduct(L, M): OBJECT
  where each of L and M is of type MODULEELEM or LIST
```

Description

This function returns the sum of the product of the components of L and M; precisely $(\text{len}(L)=\text{len}(M))$:

$\text{ScalarProduct}(L, M) = \text{sum}([L[I]*M[I] \text{ --- } I \text{ In } 1..\text{len}(L)]).$

The function works whenever the product of the components of L and M are defined (see “Algebraic Operators” (II-3.2 pg.283)).

example

```

/**/  ScalarProduct([1,2,3], [5,0,-1]);
2

  Use R ::= QQ[x,y];
  ScalarProduct([ideal(x,y), ideal(x^2-xy)], [x^2,y]);
ideal(x^3, x^2y, x^2y - xy^2)
-----

```

See Also: Algebraic Operators(II-3.2 pg.283)

I-19.3 ScientificStr

syntax

```

ScientificStr(X: INT|RAT|RINGELEM): STRING
ScientificStr(X: INT|RAT|RINGELEM, Prec: INT): STRING

```

Description

This function converts a rational number “X” into a (decimal) floating-point string. The optional second argument “Prec” says how many decimal digits to include in the mantissa; the default value is 5. Note that an exponent is always included.

example

```

/**/  ScientificStr(2/3);      -- last printed digit is rounded
6.6667*10^(-1)

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

/**/  ScientificStr(1/81, 50); -- precision of mantissa specified by user
1.2345679012345679012345679012345679012345679012346*10^(-2)

/**/  ScientificStr(1/2);      -- trailing zeroes are not suppressed
5.0000*10^(-1)

```

See Also: DecimalStr(I-4.3 pg.59), FloatStr(I-6.11 pg.83), FloatApprox(I-6.10 pg.83), MantissaAndExponent10(I-13.4 pg.160)

I-19.4 seed

syntax

```

Seed(N: INT): INT

```

Description

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

This function seeds the random number generator, “random” (I-18.3 pg.210).

NB: every time you restart CoCoA the sequence of random numbers will be the same (as happens in many programming languages). If you want better randomness, see the example below.

example

```
Seed(5);
Rand();
1991603592
-----
Rand();
-1650270230
-----
Seed(5); -- with the same seed, "Rand" generates the same sequence
Rand();
1991603592
-----
Rand();
-1650270230
-----
-- Better randomness:
-- the following shows how to make a random seed based on the date.
D := Date();
D;
Mon Mar 02 14:43:44 1998
-----
Seed(Sum(Ascii(D)));
```

See Also: [random\(I-18.3 pg.210\)](#)

I-19.5 SeparatorsOfPoints

syntax

```
SeparatorsOfPoints(Points: LIST): LIST
```

where `Points` is a list of lists of coefficients representing a set of ‘‘{\it distinct}’’ points in affine space.

Description

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

This function computes separators for the points: that is, for each point a polynomial is determined whose value is 1 at that point and 0 at all the others. The separators yielded are reduced with respect to the reduced Groebner basis which would be found by “`IdealOfPoints`” ([I-9.4 pg.112](#)).

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 Element In S Do
  PrintLn Element;
EndForeach;

```

For separators of points in projective space, see “SeparatorsOfProjectivePoints” (I-19.6 pg.232).

example

```

Use R ::= QQ[x,y];
Points := [[1, 2], [3, 4], [5, 6]];
S := SeparatorsOfPoints(Points); -- compute the separators
S;
[1/8y^2 - 5/4y + 3, -1/4y^2 + 2y - 3, 1/8y^2 - 3/4y + 1]
-----
[[Eval(F, P) | P In Points] | F In S]; -- verify separators
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
-----

```

See Also: GenericPoints(I-7.6 pg.93), IdealAndSeparatorsOfPoints(I-9.2 pg.109), IdealAndSeparatorsOfProjectivePoints(I-9.3 pg.110), IdealOfPoints(I-9.4 pg.112), IdealOfProjectivePoints(I-9.5 pg.113), Interpolate(I-9.25 pg.123), SeparatorsOfProjectivePoints(I-19.6 pg.232)

I-19.6 SeparatorsOfProjectivePoints

syntax

```
SeparatorsOfProjectivePoints(Points: LIST): LIST
```

where Points is a list of lists of coefficients representing a set of ‘‘{\it distinct}’’ points in projective space.

Description

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

This function computes separators for the points: that is, for each point a homogeneous polynomial is determined whose value is non-zero at that point and zero at all the others. (Actually, choosing the values listed in Points as representatives for the homogeneous coordinates of the corresponding points in projective space, the non-zero value will be 1.) The separators yielded are reduced with respect to the reduced Groebner basis which would be found by “IdealOfProjectivePoints” (I-9.5 pg.113).

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 Element In S Do
  PrintLn Element;
EndForeach;

```

For separators of points in affine space, see “[SeparatorsOfPoints](#)” ([I-19.5 pg.231](#)).

example

```
Use R ::= QQ[x,y,z];
Points := [[0,0,1],[1/2,1,1],[0,1,0]];
S := SeparatorsOfProjectivePoints(Points);
S;
[-2x + z, 2x, -2x + y]
-----
[[Eval(F, P) | P In Points] | F In S];  -- verify separators
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
-----
```

See Also: [GenericPoints](#)([I-7.6 pg.93](#)), [IdealAndSeparatorsOfPoints](#)([I-9.2 pg.109](#)), [IdealAndSeparatorsOfProjectivePoints](#)([I-9.3 pg.110](#)), [IdealOfPoints](#)([I-9.4 pg.112](#)), [IdealOfProjectivePoints](#)([I-9.5 pg.113](#)), [Interpolate](#)([I-9.25 pg.123](#)), [SeparatorsOfPoints](#)([I-19.5 pg.231](#))

I-19.7 *shape*

syntax

```
shape(E: LIST): LIST (of TYPE)
shape(E: RECORD): RECORD (of TYPE)
shape(E:OTHER): TYPE
```

where OTHER stands for a type which is not LIST, MAT, or RECORD.

Description

This function returns the extended list of types involved in the expression E as outlined below:

`type(E) = LIST`

In this case, `Shape(E)` is the list whose *i*-th component is the type of the *i*-th component of E.

`type(E) = MAT`

In this case, `Shape(E)` is a matrix with (*i,j*)-th entry equal to the type of the (*i,j*)-th entry of E.

`type(E) = RECORD`

In this case, `Shape(E)` is a record whose fields are the types of the fields of E.

Otherwise, “`Shape(E)`” is the type of E.

example

```
/**/ Use R ::= QQ[x];
/**/ L := [1,[1,"a"], x^2-x];
/**/ shape(L);
[INT, [INT, STRING], POLY]

/**/ R := record[name := "test", contents := L];
/**/ shape(R);
record[contents := [INT, [INT, STRING], POLY], name := STRING]

/**/ It.name;
STRING
```

There are undocumented functions, “IsSubShape” and “IsSubShapeOfSome”, for determining if the “shape” of a CoCoA expression is a “subshape” of another. To see the code for these functions, enter

```
Describe Function("$misc.IsSubShape");
Describe Function("$misc.IsSubShapeOfSome");
```

I-19.8 sign

— syntax —

```
sign(X: INT|RAT): INT
```

Description

This function returns -1 if $X < 0$, 0 if $X = 0$, and 1 if $X > 0$. X must be INT or RAT.

— example —

```
/**/ sign(123);
1

/**/ sign(-5/2);
-1
```

I-19.9 SimplestRatBetween

— syntax —

```
SimplestRatBetween(A: RAT, B: RAT): RAT
```

Description

This function finds the simplest rational in the closed interval with end points “A” and “B”.

— example —

```
/**/ SimplestRatBetween(0.123, 0.456);
1/3

/**/ SimplestRatBetween(-3.14159, -2.71828);
-3
```

See Also: CFApprox(I-3.6 pg.39), FloatApprox(I-6.10 pg.83)

I-19.10 size

— syntax —

```
OBSOLETE
```

Description

This is OBSOLETE; consider using “len” (I-12.5 pg.149).

See Also: len(I-12.5 pg.149)

I-19.11 skip

syntax

```
skip
```

Description

This command does nothing. I suppose it might be used to make the structure of a user-defined function more clear. It is probably at least as useful as the function “Tao”.

example

```
/**/ skip;
```

I-19.12 SmoothFactor

syntax

```
SmoothFactor(N: INT, MaxP: INT): RECORD
```

Description

This function finds the small prime factors of an integer. It simply tries dividing by all primes up to the given bound “MaxP”. The result is a list of the prime factors found together with the unfactored part of N. Be careful about supplying large values for “MaxP” (e.g. greater than a million) as the function could take a very long time.

From version 5.0.4 the field are called “factors” and “multiplicities” instead of “Factors” and “Exponents” to comply with the naming conventions.

example

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

/**/ SmoothFactor(123456789,3700);
record[factors := [3, 3607], multiplicities := [2, 1], RemainingFactor := 3803]
```

See Also: [IsPrime\(I-9.52 pg.134\)](#), [IsProbPrime\(I-9.53 pg.134\)](#)

I-19.13 sort

syntax

```
sort(V: LIST)
```

where V is a variable containing a list.

Description

This function sorts the elements of the list in V with respect to the default comparisons related to their types; it overwrites V and returns NULL.

For more on the default comparisons, see “Relational Operators” ([II-3.3 pg.284](#)) in the chapter on operators. For more complicated sorting, see “SortBy” ([I-19.14 pg.236](#)), “SortedBy” ([I-19.16 pg.237](#)).

example

```

/**/ L := [3,2,1];
/**/ sort(ref L); -- this returns nothing and modifies L
/**/ L;
[1, 2, 3]

/**/ Use R ::= QQ[x,y,z];
/**/ L := [x,y,z];
/**/ sort(ref L); -- this returns nothing and modifies L
/**/ L[1];
z

/**/ sorted([y, x, x^2]); -- this returns the sorted list
[y, x, x^2]

```

See Also: Relational Operators([II-3.3](#) pg.284), sorted([I-19.15](#) pg.236), SortBy([I-19.14](#) pg.236), SortedBy([I-19.16](#) pg.237)

I-19.14 SortBy

syntax

```
SortBy(L: LIST, CmpFn: FUNCTION)
```

Description

This function sorts the elements of the list in L with respect to the comparisons made by CmpFn; it overwrites L and returns NULL.

The comparison function CmpFn takes two arguments and returns True if the first argument is less than the second, otherwise it returns False. The sorted list is in increasing order.

Note that to call SortBy(L,CmpFn) inside a function you will need to make the name CmpFn accessible using TopLevel CmpFn;

Note that if both CmpFn(A, B) and CmpFn(B, A) return true, then A and B are viewed as being equal.

example

```

/**/ Define ByLength(S, T) -- define the sorting function
/**/   Return len(S) > len(T);
/**/ EndDefine;

/**/ M := ["bird","mouse","cat"];
/**/ SortBy(ref M, ByLength);
/**/ M;
["mouse", "bird", "cat"]

```

See Also: func([I-6.21](#) pg.88), sort([I-19.13](#) pg.235), sorted([I-19.15](#) pg.236), SortedBy([I-19.16](#) pg.237), TopLevel([I-20.9](#) pg.256)

I-19.15 sorted

syntax

```
sorted(L: LIST): LIST
```

Description

This function returns the list of the sorted elements of L without affecting L, itself.

For more on the default comparisons, see “Relational Operators” (II-3.3 pg.284) in the chapter on operators. For more complicated sorting, see “SortBy” (I-19.14 pg.236), “SortedBy” (I-19.16 pg.237).

example

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

/**/ Use R ::= QQ[x,y,z];
/**/ L := [x,y,z];
/**/ sorted(L);
[z, y, x]

/**/ sorted([y, x, z, x^2]);
[z, y, x, x^2]

/**/ sorted([3, 1, 1, 2]);
[1, 1, 2, 3]

/**/ sorted(["b","c","a"]);
["a", "b", "c"]
```

See Also: Relational Operators(II-3.3 pg.284), SortBy(I-19.14 pg.236), SortedBy(I-19.16 pg.237), sort(I-19.13 pg.235)

I-19.16 SortedBy

syntax

```
SortedBy(L: LIST, F: FUNCTION): LIST
```

Description

This function returns the list of the sorted elements of L without affecting L, itself. As for “SortBy” (I-19.14 pg.236), the comparison function F takes two arguments and returns True if the first argument is less than the second, otherwise it returns False. The sorted list is in increasing order.

Note that if both $F(A, B)$ and $F(B, A)$ return True, then A and B are viewed as being equal.

example

```
/**/ Define ByLength(S, T)    -- define the sorting function
/**/   Return len(S) > len(T);
/**/ EndDefine;

/**/ M := ["bird","mouse","cat"];
/**/ SortedBy(M, ByLength);
["mouse", "bird", "cat"]

/**/ M; -- M is not changed
["bird", "mouse", "cat"]

/**/ sorted(M); -- the function "Sort" sorts using the default ordering:
                -- in this case, alphabetical order.
["cat", "bird", "mouse"]
```

```
/**/ SortBy(ref M, ByLength); -- sort M in place, changing M
/**/ M;
["mouse", "bird", "cat"]
```

See Also: [func\(I-6.21 pg.88\)](#), [sort\(I-19.13 pg.235\)](#), [sorted\(I-19.15 pg.236\)](#), [SortBy\(I-19.14 pg.236\)](#)

I-19.17 source

syntax

```
source S: STRING
```

Description

This command executes all CoCoA commands in the file or device named S. A typical use of “source” is to collect user-defined functions and variables in a text file, say, “MyFile.coc” and then execute:

```
source "MyFile.cocoa5";
```

or, equivalently, the obsolescent

```
<< "MyFile.cocoa5";
```

Functions and variables read in from a file in this way will erase functions and variables with identical names that may already exist. This can be avoided by using packages. Repeatedly used functions can be read into CoCoA at start-up by using “source” in the “userinit.coc” file.

See Also: [Introduction to IO\(II-6.1 pg.291\)](#), [Introduction to Packages\(II-7.1 pg.297\)](#)

I-19.18 SourceRegion

syntax

```
SourceRegion FromLine: INT,FromChar: INT To ToLine: INT,ToChar: INT In S: STRING
```

Description

This command executes all CoCoA commands in the specified region of the given file. It is not 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\(I-19.17 pg.238\)](#)

I-19.19 spaces

syntax

```
spaces(N: INT): STRING
```

Description

This function returns a string consisting of N spaces.

example

```
/**/ L := "a" + Spaces(5) + "b";
/**/ L;
a      b
```

See Also: [dashes\(I-4.1 pg.59\)](#)

I-19.20 *sprint*

syntax

```
sprint(E: OBJECT): STRING
```

Description

This function takes any CoCoA expression and converts its value to a string. One use is to check for extremely long output before printing in a CoCoA window.

example

```
/**/ Use R ::= QQ[x,y];
/**/ I := ideal(x,y);
/**/ J := sprint(I);
/**/ I;
ideal(x, y)

/**/ J;          -- The output for I and J looks the same, but ...
ideal(x, y)

/**/ type(I); -- I is an ideal, and
IDEAL

/**/ type(J); -- J is just the string "ideal(x, y)".
STRING

/**/ J[1]; -- the 1st character of J
i

/**/ J[2]; -- the 2nd character of J
d

/**/ len(J); -- J has 11 characters
11
```

See Also: [Introduction to IO\(II-6.1 pg.291\)](#), [IO.SprintTrunc\(I-9.31 pg.126\)](#), [print\(I-16.21 pg.199\)](#), [println\(I-16.25 pg.201\)](#)

I-19.21 *SqFreeFactor*

syntax

```
SqFreeFactor(F: RINGELEM): RECORD
```

Description

Compute a factorization (of a polynomial) into coprime squarefree factors. The factorization may sometimes be finer than necessary, i.e. two factors could have the same multiplicity.

example

```

/**/ Use R ::= QQ[x,y];
/**/ f := (x^2-1)^2*(y+2)^3;
/**/ indent(SqFreeFactor(f));
record[
  RemainingFactor := 1,
  factors := [x^2 -1, y +2],
  multiplicities := [2, 3]
]
```

See Also: `factor`(I-6.1 pg.79), `ContentFreeFactor`(I-3.40 pg.54)

I-19.22 StableBBasis5

syntax

```

StableBBasis5(Pts: LIST, Toler: LIST): RECORD
StableBBasis5(Pts: LIST, Toler: LIST, Gamma: RAT): RECORD
```

Description

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

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

```

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\(I-9.4 pg.112\)](#), [TmpNBM\(I-20.8 pg.256\)](#)

I-19.23 *StableIdeal*

syntax

```
StableIdeal(L: LIST of power-products): IDEAL
```

Description

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

This function returns the smallest stable ideal containing the power-products in L.

example

```

Use R ::= QQ[x,y,z];
L := [xz^4, y^3];
StableIdeal(L);
ideal(x^2z^3, xyz^3, xz^4, x^3, x^2y, xy^2, y^3)
-----

```

See Also: [IsStable\(I-9.57 pg.136\)](#), [LexSegmentIdeal\(I-12.7 pg.150\)](#), [StronglyStableIdeal\(I-19.28 pg.243\)](#)

I-19.24 *StarPrint, StarSprint*

syntax

```

StarPrint(F: RINGELEM)
StarPrintFold(F: RINGELEM, LineWidth: INT)
StarSprint(F: RINGELEM): STRING
StarSprintFold(F: RINGELEM, LineWidth: INT): STRING

```

Description

These functions print the polynomial F with asterisks added to denote multiplications. They may be useful when transferring polynomials or rational functions from CoCoA to other mathematical software (e.g. Gap, Maple, Macaulay, Singular,...). “**StarPrint**” inserts newline characters (only between terms) with the aim of avoiding lines longer than 70 characters; the second argument to “**StarPrintFold**” is for specifying a different width limit; a non positive value is treated as meaning no limit. The “**StarSprint**” functions print the value into a string.

example

```

Use R ::= QQ[x,y];
F := x^3+2xy-y^2;
StarPrint(F);
1*x^3 +2*x*y -1*y^2

```

```

-----
  StarPrintFold(F,1); -- this will print one term per line
1*x^3
+2*x*y
-1*y^2
-----

D := OpenOFile("example");
Print StarSprint(F) On D; -- this will print F into the file "example"
Close(D);
-----

```

See Also: Latex([I-12.2](#) pg.147)

I-19.25 starting

syntax

```
starting(S: STRING): LIST of RECORD
```

Description

This function returns a list of all CoCoA functions starting with the string “S”. In general, this list will include undocumented commands. For these, one may find some information using “Describe Function(“Fn_Name”)” or “Describe Function(“\$PackageName.Fn_Name”)”.

example

```

/**/ indent(starting("Su"));
[
  record[IsExported := true, name := "$BackwardCompatible.Subsets"],
  record[IsExported := true, name := "$BackwardCompatible.Subst"],
  record[IsExported := true, name := "$BackwardCompatible.Sum"],
  record[IsExported := true, name := "$BackwardCompatible.Support"]
]

```

I-19.26 StdDegLexMat

syntax

```
StdDegLexMat(N: INT): MAT
```

Description

This function return the matrix defining a standard term-ordering.

example

```

/**/ StdDegLexMat(3);
matrix(QQ,
  [[1, 1, 1],
   [1, 0, 0],
   [0, 1, 0]])

```

See Also: OrdMat([I-15.10](#) pg.189), Orderings([III-9.5](#) pg.329), StdDegRevLexMat([I-19.27](#) pg.243), LexMat([I-12.6](#) pg.149), RevLexMat([I-18.29](#) pg.222), XelMat([I-24.1](#) pg.271)

I-19.27 StdDegRevLexMat

syntax

```
StdDegRevLexMat(N: INT): MAT
```

Description

This function return the matrix defining a standard term-ordering.

example

```
/**/ StdDegRevLexMat(3);
matrix(QQ,
  [[1, 1, 1],
   [0, 0, -1],
   [0, -1, 0]])
```

See Also: OrdMat([I-15.10](#) pg.189), Orderings([III-9.5](#) pg.329), StdDegLexMat([I-19.26](#) pg.242), LexMat([I-12.6](#) pg.149), RevLexMat([I-18.29](#) pg.222), XelMat([I-24.1](#) pg.271)

I-19.28 StronglyStableIdeal

syntax

```
StronglyStableIdeal(L: LIST of power-products): IDEAL
```

Description

This function returns the smallest strongly stable ideal containing the power-products in L.

example

```
/**/ Use R ::= QQ[x,y,z];
/**/ L := [x*y^2*z];
/**/ StableIdeal(L);
ideal(x^4, x^3*y, x^2*y^2, x*y^3, x*y^2*z)

/**/ StronglyStableIdeal(L);
ideal(x^4, x^3*y, x^2*y^2, x*y^3, x^3*z, x^2*y*z, x*y^2*z)
```

See Also: IsStronglyStable([I-9.59](#) pg.137), LexSegmentIdeal([I-12.7](#) pg.150), StableIdeal([I-19.23](#) pg.241)

I-19.29 SubalgebraMap

syntax

```
IsInSubalgebra(F: RINGELEM, L: LIST)
IsInSubalgebra(F: RINGELEM, TAGGED("$alghom.Map"))
```

Description

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

This function returns the homomorphism representing a subalgebra.

example

```
Use QQ[s,t];
SAM := SubalgebraMap([s^3, s^2t, st^2, t^3]);
```

```

    Ker(SAM);
SubalgebraRing :: ideal(x[3]^2 - x[2]x[4], x[2]x[3] - x[1]x[4], x[2]^2 - x[1]x[3])
-----
    IsInSubalgebra(s^3, SAM);
True
-----
    SubalgebraRepr(s^3, SAM);
SubalgebraRing :: x[1]
-----
    IsInSubalgebra(s^5, [s^3, s^2t, st^2, t^3]);
False
-----
    SubalgebraRepr(s^5, SAM);
NULL
-----

```

See Also: [SubalgebraRepr\(I-19.30 pg.244\)](#), [IsInSubalgebra\(I-9.48 pg.133\)](#), [ker\(I-11.1 pg.145\)](#)

I-19.30 SubalgebraRepr

— syntax —

```

SubalgebraRepr(F: RINGELEM, L: LIST): RINGELEM
SubalgebraRepr(F: RINGELEM, M: TAGGED("$alghom.Map")): RINGELEM

```

Description

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

This function returns the representation of a polynomial as a subalgebra element in terms of the subalgebra generators.

— example —

```

Use QQ[s,t];
L := [s^3, s^2t, st^2, t^3];
SAM := SubalgebraMap(L);
IsInSubalgebra(s^6t^6, SAM);
True
-----
    SubalgebraRepr(s^6t^6, L);
SubalgebraRing :: x[1]^2x[4]^2
-----
    SubalgebraRepr(s^6t^6, SAM); -- more efficient for repeated computations
SubalgebraRing :: x[1]^2x[4]^2
-----
    Image(It, RMap(L));
s^6t^6
-----

```

See Also: [SubalgebraMap\(I-19.29 pg.243\)](#), [IsInSubalgebra\(I-9.48 pg.133\)](#), [ker\(I-11.1 pg.145\)](#)

I-19.31 submat

— syntax —

```

submat(M: MAT, R: LIST of INT, C: LIST of INT): MAT

```

Description

This function returns the submatrix of M formed by the rows listed in R and the columns listed in C . If M is a list, it is interpreted as a matrix in the natural way.

example

```

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

/**/ M := mat([[1,2,3],[4,5,6]]);
/**/ submat(M,[2],[1,3]);
matrix([
  [4, 6]
])

```

See Also: minors(I-13.17 pg.166)

I-19.32 submodule

syntax

```

submodule(L: LIST of MODULEELEM): MODULE
submodule(F: MODULE, L: LIST of MODULEELEM): MODULE

```

Description

The first form returns the ideal generated by “ L ”. The second is the same as the first but works also if “ $L = []$ ”.

This function is not friendly if you write the input by hand: we suggest “SubmoduleCols, SubmoduleRows” (I-19.33 pg.245) for creating a module from the rows or columns of a matrix.

NOTE: the second argument is a LIST of MODULEELEM, not a LIST of LISTS of RINGELEM.

example

```

/**/ Use R ::= QQ[x,y,z];
/**/ R3 := NewFreeModule(R, 3);
/**/ L := [ModuleElem(R3, [x,y,z]), ModuleElem(R3, [x-1,0,z])];
/**/ M := submodule(R3, L); -- equivalent to
/**/ M := submodule(L);      -- (L not empty)
/**/ gens(M);
[[x, y, z], [x -1, 0, z]]

```

See Also: SubmoduleCols, SubmoduleRows(I-19.33 pg.245), GensAsCols, GensAsRows(I-7.9 pg.95), gens(I-7.8 pg.94)

I-19.33 SubmoduleCols, SubmoduleRows

syntax

```

SubmoduleCols(F: MODULE, M: MATRIX): MODULE
SubmoduleRows(F: MODULE, M: MATRIX): MODULE

```

Description

The first (second) function returns the submodule of F generated by the module elements described by the columns (rows) in the matrix M (which might be empty).

Dimensions must be compatible.

— example —

```

/**/ 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: 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.95), submodule(I-19.32 pg.245), ModuleElem(I-13.22 pg.168)

I-19.34 subsets

— syntax —

```

subsets(S: LIST): LIST
subsets(S: LIST, N: INT): LIST

```

Description

This function computes all sublists (subsets) of a list (set). If N is specified, it computes all sublists of cardinality N .

— example —

```

/**/ subsets([1, 4, 7]);
[[ ], [7], [4], [4, 7], [1], [1, 7], [1, 4], [1, 4, 7]]

/**/ subsets([1, 4, 7], 2);
[[1, 4], [1, 7], [4, 7]]

/**/ subsets([2,3,3]); -- list with repeated entries
[[ ], [3], [3], [3, 3], [2], [2, 3], [2, 3], [2, 3, 3]]

/**/ subsets(MakeSet([2,3,3]));
[[ ], [3], [2], [2, 3]]

```

See Also: IsSubset(I-9.60 pg.137), partitions(I-16.4 pg.192), permutations(I-16.5 pg.192), MakeSet(I-13.3 pg.160), tuples(I-20.14 pg.259)

I-19.35 subst

— syntax —

```

subst(E: OBJECT, X, F): OBJECT
subst(E: OBJECT, [[X_1, F_1], ..., [X_r, F_r]]): OBJECT
  where each X or X_i is an indeterminate
  and each F or F_i is a RINGELEM

```

Description

The first form of this function substitutes “F_i” for “X_i” in the expression E. The second form is a shorthand for the first in the case of a single indeterminate. When substituting for the indeterminates in order, it is easier to use “eval” (I-5.9 pg.75).

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-5.9 pg.75), Evaluation of Polynomials(III-11.2 pg.336), image(I-9.9 pg.115), PolyAlgebraHom(I-16.14 pg.195), QZP(I-17.4 pg.206), RingElem(I-18.30 pg.223), ZPQ(I-25.3 pg.274)

I-19.36 *sum*

syntax

```

sum(L: LIST): OBJECT
sum(L: LIST, 0: OBJECT): OBJECT

```

Description

This function returns the sum of the objects in the list L.

example

```

/**/ Use R ::= QQ[x,y];
/**/ sum([3, x, y^2]);
y^2 +x +3

/**/ sum(1..40) = binomial(41,2);
true

/**/ sum(["c","oc","oa"]);
cocoa

/**/ sum([]); -- gives 0 of type INT
0
/**/ sum([], ""); -- gives empty STRING

/**/ sum([], x); -- gives type RINGELEM
x

```

See Also: Algebraic Operators(II-3.2 pg.283), product(I-16.27 pg.202)

I-19.37 support

syntax

```
support(F: RINGELEM): LIST
support(F: MODULEELEM): LIST
```

Description

This function returns the list of terms of F. To get a list of monomials, which includes coefficients, use “monomials” ([I-13.25 pg.169](#)).

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\(I-3.19 pg.43\)](#), [monomials\(I-13.25 pg.169\)](#)

I-19.38 swap

syntax

```
swap(ref A: OBJECT, ref B: OBJECT)
```

Description

This procedure swaps two values; it returns nothing!

example

```
/**/ A := 1;
/**/ B := 2;
/**/ swap(ref A, ref B);
/**/ PrintLn [A,B];
[2, 1]
```

See Also: [ref\(I-18.15 pg.216\)](#)

I-19.39 sylvester

syntax

```
sylvester(F: RINGELEM, G: RINGELEM, X: RINGELEM): MAT
```

Description

(sorry Sylvester for the lower-case: here we follow the naming convention “*single name goes lower-case*”)

This function returns the Sylvester matrix of the polynomials F and G with respect to the indeterminate X. This is the matrix used to calculate the resultant.

example

```

/**/ Use R ::= QQ[p,q,x];
/**/ F := x^3+p*x-q; G := deriv(F, x);
/**/ sylvester(F, G, x);
matrix([
  [1, 0, p, -q, 0],
  [0, 1, 0, p, -q],
  [3, 0, p, 0, 0],
  [0, 3, 0, p, 0],
  [0, 0, 3, 0, p]
])

/**/ det(sylvester(F, G, x)) = resultant(F, G, x);
true

```

See Also: resultant(I-18.26 pg.221)

I-19.40 SymbolRange

syntax

```
SymbolRange(H: STRING, LO: INT, HI: INT): LIST of RINGELEM
```

Description

This function returns the list of the symbols with a given head and a range of indices. A symbol is a record with head (as “IndetName” (I-9.18 pg.119)) and indices (as “IndetSubscripts” (I-9.20 pg.121))

example

```

/**/ indent(SymbolRange("x", 3, 5));
[
  record[head := "x", indices := [3]],
  record[head := "x", indices := [4]],
  record[head := "x", indices := [5]]
]
/**/ P := NewPolyRing(QQ, SymbolRange("x", 0,2));
/**/ indets(P);
[x[0], x[1], x[2]]

```

See Also: indet(I-9.16 pg.119), IndetSubscripts(I-9.20 pg.121), IndetIndex(I-9.17 pg.119), IndetName(I-9.18 pg.119), NumIndets(I-14.27 pg.181), SymbolRange(I-19.40 pg.249)

I-19.41 syz

syntax

```

Syz(L: LIST of RINGELEM): MODULE
Syz(M: IDEAL|MODULE, Index: INT): MODULE

```

Description

In the first two forms this function computes the syzygy module of a list of polynomials or module elements. “SyzOfGens(I)” is the same as “Syz(gens(I))”.

In the last form this function returns the specified syzygy module of the minimal free resolution of M which must be homogeneous. As a side effect, it computes the Groebner basis of M. (***** NOT YET IMPLEMENTED *****)

The coefficient ring must be a field.

example

```

/**/ Use R ::= QQ[x,y,z];
/**/ indent(Syz([x^2-y-1, y^3-z, x^2-y, y^3-z]));
SubmoduleRows(F, matrix(
  [y^3 -z, 0, 0, -x^2 +y +1],
  [0, 1, 0, -1],
  [x^2 -y, 0, -x^2 +y +1, 0],
  [0, 0, y^3 -z, -x^2 +y]
))
-----
/**/ I := ideal(x, x, y);
/**/ syz(gens(I));
submodule(FreeModule(..), [[1, -1, 0], [0, y, -x]])
/**/ SyzOfGens(I);
submodule(FreeModule(..), [[1, -1, 0], [0, y, -x]])

Syz(I, 1);      -- NOT YET IMPLEMENTED
Module([[x, -y]])
-----
I := ideal(x^2-yz, xy-z^2, xyz);  -- NOT YET IMPLEMENTED
Syz(I,0);
Module([x^2 - yz], [xy - z^2], [xyz])
-----
Syz(I,1);      -- NOT YET IMPLEMENTED
Module([-x^2 + yz, xy - z^2, 0], [xz^2, -yz^2, -y^2 + xz], [z^3, 0,
-xy + z^2], [0, z^3, -x^2 + yz])
-----
Syz(I,2);
Module([0, z, -x, y], [-z^2, -x, y, -z])
-----
Syz(I,3);
Module([[0]])
-----
Res(I);
0 --> R(-6)^2 --> R(-4)(+)R(-5)^3 --> R(-2)^2(+)R(-3)
-----

```

See Also: [res\(I-18.23 pg.220\)](#), [SyzOfGens\(I-19.42 pg.250\)](#)

I-19.42 SyzOfGens

syntax

```
SyzOfGens(M: IDEAL|MODULE): MODULE
```

Description

If M is an ideal or submodule, this function calculates the syzygy module for the given set of generators of M.

If M is a quotient of a ring by an ideal I or a quotient of a free module by a submodule N, then this function calculates the syzygy module for the given set of generators of I or N, respectively.

“SyzOfGens(I)” is the same as “Syz(gens(I))”.

The coefficient ring must be a field.

example

```

/**/ Use R ::= QQ[x,y,z];

```

```
/**/ I := ideal(x, y, x+y);
/**/ indent(SyzOfGens(I));
SubmoduleRows(F, matrix([
  [1, 1, -1],
  [0, x +y, -y]
]))

/**/ R3 := NewFreeModule(R, 3);
/**/ MGens := matrix(R, [[x,y,z], [x-y,0,z], [y^2,y^2,0]]);
/**/ indent(SyzOfGens(SubmoduleRows(R3, MGens)));
SubmoduleRows(F, matrix([
  [1, -1, -1]
]))
```

See Also: [syz\(I-19.41 pg.249\)](#)

Chapter I-20

T

I-20.1 tag

syntax

```
tag(E: OBJECT): STRING
```

Description

If E is a tagged object, this function returns the tag of E; otherwise, it returns the empty string.

example

```
/**/ L := tagged(3,"MyTag");
/**/ type(L);
TAGGED("$TopLevel.MyTag")

/**/ tag(L);
MyTag
```

See Also: Tagged Printing([II-6.6 pg.293](#)), tagged([I-20.2 pg.253](#)), untagged([I-21.4 pg.262](#))

I-20.2 tagged

syntax

```
tagged(E: OBJECT, S: STRING): TAGGED(S)
```

Description

This first function returns the object E, tagged with the string S. Tagging is used for pretty printing of objects. See the reference listed below.

example

```
/**/ L := [1,2,3];
/**/ M := tagged(L,"MyTag");
/**/ type(L);
LIST

/**/ type(M);
TAGGED("$TopLevel.MyTag")
```

```
/**/ type(untagged(M));
LIST
```

See Also: Tagged Printing([II-6.6](#) pg.293), tag([I-20.1](#) pg.253), untagged([I-21.4](#) pg.262)

I-20.3 tail

syntax

```
tail(L: LIST): LIST
```

Description

This function returns the list obtained from L by removing its first element. It cannot be applied to the empty list.

example

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

See Also: first([I-6.6](#) pg.81), last([I-12.1](#) pg.147)

I-20.4 TensorMat

syntax

```
TensorMat(M: MATRIX, N: MATRIX): MAT
```

Description

This function returns the tensor product of two matrices.

example

```
/**/ Use R ::= QQ[x,y,z,w];
/**/ TensorMat(mat(R, [[1,-1],[2,-2],[3,-3]]), mat(R, [[x,y],[z,w]]));
matrix([
  [x, y, -x, -y],
  [z, w, -z, -w],
  [2*x, 2*y, -2*x, -2*y],
  [2*z, 2*w, -2*z, -2*w],
  [3*x, 3*y, -3*x, -3*y],
  [3*z, 3*w, -3*z, -3*w]
])
```

I-20.5 TgCone

syntax

```
TgCone(I: IDEAL): IDEAL
```

Description

The “*initial form*” of a polynomial F is the homogeneous component of F of the lowest degree (in contrast with the “*leading form*”, see “LF” ([I-12.8](#) pg.150), “DF” ([I-4.14](#) pg.66)). 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 II, 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(y^2 -x*z, z^2, -y*z)
```

See Also: InitialIdeal(I-9.23 pg.122), PrimaryPoincare(I-16.19 pg.198)

I-20.6 TimeFrom

syntax

```
TimeFrom(StartPoint: RAT): STRING
```

Description

This function returns a string indicating the number of CPU seconds consumed since “StartPoint”; the value in “StartPoint” should be the value produced by the function “CpuTime” (I-3.45 pg.56) at the point where timing should commence.

example

```
/**/ t0 := CpuTime();
/**/ N := factorial(10000000);
/**/ PrintLn "Time to compute N: ",TimeFrom(t0);
Time to compute N: 7.538
```

I-20.7 TimeOfDay

syntax

```
TimeOfDay(): INT
```

Description

This function returns the current time as an INT in the form HHMMSS.

Note that from version 5.0.4 this information is no longer given by the function “date” (I-4.2 pg.59).

example

```
/**/ date();          -- 2013-05-30
20130530
/**/ TimeOfDay();    -- 09:08:13
90813
```

See Also: date(I-4.2 pg.59)

I-20.8 TmpNBM

syntax

```
TmpNBM(P: RING, Pts: MAT, Toler: MAT): RECORD[QuotientBasis: LIST, BBasis: LIST, AlmostVanishing: LIS
```

Description

This function checks that the current ring is suitable: see below for details.

This function returns a record containing a factor-closed set of power-products “**QuotientBasis**” and a list of “*almost vanishing*” polynomials. If the cardinality of the “**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.

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). 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.

For a full description of the algorithms we refer to the paper C.Fassino “*Almost Vanishing Polynomials for Sets of Limited Precision Points*” (arXiv:0807.3412).

example

```
/**/ P ::= QQ[x,y];
/**/ Eps := [0.1, 0.1];
/**/ Points := [[10, 0], [-10, 0], [0, 10], [0, -10], [7, 7], [-7, -7]];
/**/ indent(TmpNBM(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\(I-9.4 pg.112\)](#), [StableBBasis5\(I-19.22 pg.240\)](#)

I-20.9 TopLevel

syntax

```
TopLevel X;
  where ‘‘\verb&X&’’ is the name of a top level variable or function.
```

Description

This command makes a top-level variable accessible from inside a function. It is useful for making “QQ” and “ZZ” visible, and also if a top-level function is to be passed as a parameter (e.g. to the function “**SortBy**” ([I-19.14 pg.236](#))).

The command may be used with any top-level variable, but it is poor style to use it for purposes other than those mentioned above.

example

```
/**/ 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-6.21 pg.88), ImportByRef, ImportByValue(I-9.12 pg.116)

I-20.10 TopLevelFunctions

syntax

TopLevelFunctions(): LIST of FUNCTION

Description

This function returns the list of all functions available at top-level

example

```

/**/  indent(TopLevelFunctions());
[
  record[IsExported := true, Name := "$BackwardCompatible.Abs"],
  record[IsExported := true, Name := "$BackwardCompatible.Append"],
  record[IsExported := true, Name := "$BackwardCompatible.Ascii"],
  ...

```

I-20.11 toric

syntax

```

toric(I: IDEAL): IDEAL
toric(I: IDEAL, L: LIST of INDETS): IDEAL
toric(M: MAT|LIST of LIST): IDEAL

```

Description

These functions return the saturation of an ideal, I , generated by binomials. In the first two cases, I is the ideal generated by the binomials in L . To describe the ideal in the last case, let K be the integral elements in the kernel of M . For each k in K , we can write $k = k(+) - k(-)$ where the i -th component of $k(+)$ is the i -th component of k , if positive, otherwise zero. Then I is the ideal generated by the binomials " $x^{k(+)} - x^{k(-)}$ " as k ranges over K .

NOTE: successive calls to this last form of the function may produce different generators for the saturation.

The first and third functions return the saturation of I . For the second function, if the saturation of I with respect to the variables in X happens to equal the saturation of I , then the saturation of I is returned. Otherwise, an ideal "*containing*" the saturation with respect to the given variables is returned. The point is that if one knows, a priori, that the saturation of I can be obtained by saturating with respect to a subset of the variables, the second function may be used to save time.

For more details, see the article: A.M. Bigatti, R. La Scala, L. Robbiano, "*Computing Toric Ideals*," Preprint (1998). The article describes three different algorithms; the one implemented in CoCoA is "*EATP*". The first two examples below are motivated by B. Sturmfels, "*Groebner Bases and Convex Polytopes*," Chapter 6, p. 51. They count the number of homogeneous primitive partition identities of degrees 8 and 9.

example

```

/**/ Use QQ[x[1..8],y[1..8]];
/**/ HPPI8 := [x[1]^I*x[I+2]*y[2]^(I+1) -y[1]^I*y[I+2]*x[2]^(I+1) | I In 1..6];
/**/ BL := toric(ideal(HPPI8), [x[1],y[2]]);
/**/ len(gens(BL));
340

/**/ Use QQ[x[1..9],y[1..9]];
/**/ HPPI9 := [x[1]^I*x[I+2]*y[2]^(I+1) -y[1]^I*y[I+2]*x[2]^(I+1) | I In 1..7];
/**/ BL := toric(ideal(HPPI9), [x[1],y[2]]);
/**/ len(gens(BL));
798

/**/ Use R ::= QQ[x,y,z,w];
/**/ toric(ideal(x*z-y^2, x*w-y*z));
ideal(-y^2 +x*z, -y*z +x*w, z^2 -y*w)

/**/ toric(ideal(x*z-y^2, x*w-y*z), [y]);
ideal(-y^2 +x*z, -y*z +x*w, z^2 -y*w)

/**/ Use R ::= QQ[x,y,z];
/**/ toric([[1,3,2],[3,4,8]]);
ideal(-x^16 +y^2*z^5)

/**/ toric(mat([[1,3,2],[3,4,8]]));
ideal(-x^16 +y^2*z^5)

```

I-20.12 transposed

syntax

```
transposed(M: MAT): MAT
```

Description

This function returns the transpose of the matrix M.

example

```

/**/ M := mat([[1,2,3],[4,5,6]]);
/**/ M;
matrix([
  [1, 2, 3],
  [4, 5, 6]
])

/**/ transposed(M);
matrix([
  [1, 4],
  [2, 5],
  [3, 6]
])

```

I-20.13 try

syntax

```
Try C1 UponError E Do C2 EndTry
  where C1, C2 are sequences of commands and E is a variable identifier.
```

Description

Usually, when an error occurs during the execution of a command, the error is automatically propagated out of the nesting of the evaluation. This can be prevented with the use of “Try..UponError”.

If an error occurs during the execution of the commands C1, then it is captured by the command “UponError” and assigned to the variable E, and the commands C2 are executed; the string inside E may be retrieved using “GetErrMesg” (I-7.14 pg.97). If no error occurs then the variable E and the commands C2 are ignored.

example

```
-- /**/ deg(zero(R)); --> !!! ERROR !!!
-- ERROR: Non-zero RingElem required
-- deg(zero(R));
-- ~~~~~

/**/ Define MyDeg(F)
/**/   Try
/**/     D := Deg(F);
/**/     Return D;
/**/   UponError E Do
/**/     MyDegError := GetErrMesg(E);
/**/     If "Non-zero RingElem required" IsIn MyDegError Then
/**/       Return -123456;
/**/     Else
/**/       error(MyDegError);
/**/     EndIf;
/**/   EndTry;
/**/ EndDefine;

/**/ MyDeg(x);
1
/**/ MyDeg(zero(R));
-123456
-- /**/ MyDeg("a"); --> !!! ERROR !!!
ERROR: Expecting type RINGELEM, but found type STRING
      error(MyDegError);
      ~~~~~
```

See Also: error(I-5.8 pg.74), GetErrMesg(I-7.14 pg.97)

I-20.14 tuples

syntax

```
tuples(S: LIST, N: INT): LIST
```

Description

This function computes all N-tuples with entries in S. It is equivalent to “S >< S >< ... >< S” [N times].

example

```
/**/ tuples([1, 4, 7], 2);
[[1, 1], [1, 4], [1, 7], [4, 1], [4, 4], [4, 7], [7, 1], [7, 4], [7, 7]]
```

See Also: `CartesianProduct`, `CartesianProductList`([I-3.3](#) pg.38), `permutations`([I-16.5](#) pg.192), `subsets`([I-19.34](#) pg.246)

I-20.15 type

syntax

```
type(E: OBJECT): TYPE
```

Description

This function returns the data type of E.

example

```
/**/ L := [1,"a",2,"b",3,"c"];
/**/ [ X In L | type(X)=INT ];
[1, 2, 3]

/**/ type(type(INT)); -- Type returns a value of type TYPE
TYPE

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

See Also: `CurrentTypes`([I-3.48](#) pg.57)

Chapter I-21

U

I-21.1 UnivariateIndetIndex

syntax

```
UnivariateIndetIndex(F: RINGELEM): INT
```

Description

This function returns 0 if the polynomial F is not univariate otherwise it returns the indeterminate index of F.

NB: If F is a constant, it returns 1.

example

```
/**/ Use R := QQ[x,y,z];
/**/ UnivariateIndetIndex(3*x^4-2*x-1);
1

/**/ UnivariateIndetIndex(x-y-1);
0

/**/ UnivariateIndetIndex(one(R));
1
```

See Also: [indet\(I-9.16 pg.119\)](#), [IndetSubscripts\(I-9.20 pg.121\)](#), [IndetIndex\(I-9.17 pg.119\)](#), [IndetName\(I-9.18 pg.119\)](#), [indets\(I-9.19 pg.120\)](#), [NumIndets\(I-14.27 pg.181\)](#)

I-21.2 unprotect

syntax

```
unprotect X;
```

Description

This command undoes the effect of the “**protect**” ([I-16.28 pg.203](#)) 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\(I-16.28 pg.203\)](#)

I-21.3 Unset

syntax

```
UnSet 0
```

Description

“*OBSOLETE*”

I-21.4 untagged

syntax

```
untagged(E:TAGGED_OBJECT):UNTAGGED_OBJECT
```

Description

This function strips an object E of its tag, if any. “@E” is equivalent to “untagged(E)”.

Tags are used for pretty printing of objects. See the reference listed below.

example

```
/**/ L := [1,2,3];
/**/ M := tagged(L,"MyTag");
/**/ type(L);
LIST

/**/ type(M);
TAGGED("MyTag")

/**/ type(untagged(M));
LIST
```

See Also: [Tagged Printing\(II-6.6 pg.293\)](#), [tag\(I-20.1 pg.253\)](#), [tagged\(I-20.2 pg.253\)](#)

I-21.5 updating CoCoA-4 code

syntax

```
Not E
Var X
Using R Do...EndUsing
```

Description

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(...)”.

Several functions modify one of their arguments (e.g. “append” (I-1.11 pg.27), “sort” (I-19.13 pg.235)); CoCoA-5 wants these arguments to be identified with the new keyword “ref” (I-18.15 pg.216), and will issue a warning if you don’t do this.

Implicit multiplication has gone: either write “x*y” instead of “xy” for every product, or use “CoCoA-4 mode” (I-3.15 pg.42).

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-20.9 pg.256). 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-18.31 pg.224).

The function “LinKer” (I-12.10 pg.151) has been replaced by “LinKerBasis” (I-12.11 pg.152), and there is a new function called “LinKer” (I-12.10 pg.151) which produces a matrix. More generally, see also the CoCoA-4 “translation table” (where is it???)

example

```
/*C4*/ If Not X IsIn L Then ... EndIf;
/*C5*/ If not(X IsIn L) Then ... EndIf;

/*C4*/ F := 3xyzt;
/*C5*/ F := 3*x*y*z*t; OR F := ***3xyzt***;

/*C4*/ Define LastIndet() Return Last(Indets()); EndDefine;
/*C5*/ Define LastIndet() TopLevel CurrentRing; Return last(indets(CurrentRing)); EndDefine;
```

See Also: CoCoA-4 mode(I-3.15 pg.42), TopLevel(I-20.9 pg.256), CurrentRing(I-3.47 pg.56), RingOf(I-18.31 pg.224)

I-21.6 use

syntax

```
Use R
Use RingDefn
Use R ::= RingDefn
```

where R is a RING, and RingDefn is a ring definition.

Description

This command works only at top-level; it makes a ring active, i.e. it makes that ring the “current ring”. The command will also let you create a new ring, and make it active immediately “Use NewR ::= RingDefn;” where “RingDefn” is a ring definition; this is a shorthand for “NewR ::= RingDefn; Use NewR;”

This command cannot be called inside a function, and it is never necessary (if you write clean programs ;-) 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: Accessing Other Rings([III-9.7](#) pg.330), CurrentRing([I-3.47](#) pg.56), RingOf([I-18.31](#) pg.224)

Chapter I-22

V

I-22.1 valuation

syntax

```
valuation(p: INT, N: INT): INT
```

Description

This function determines the p-adic valuation of a non-zero integer: the largest integer k such that power(p,k) divides N. The first argument is the prime p.

example

```
/**/ valuation(5,10);  
1
```

See Also: [IsDivisible\(I-9.38 pg.128\)](#)

I-22.2 VersionInfo

syntax

```
VersionInfo(): RECORD
```

Description

This function returns a record with various information about CoCoA and CoCoALib (the mathematical core of CoCoA)

example

```
/**/ indent(VersionInfo());  
record[  
  CoCoALibVersion := "0.99533",  
  CoCoAVersion := "5.1.0",  
  CompilationDate := "May 14 2014 15:48:58",  
  ...
```

See Also: [CoCoALib\(II-8.1 pg.303\)](#)

Chapter I-23

W

I-23.1 wdeg

syntax

```
wdeg(F: RINGELEM): LIST
```

Description

This function returns the multi-weighted degree of F, as determined by the matrix weights of the polynomial ring of F. The function “deg” (I-4.6 pg.61) returns the standard degree.

NB: In CoCoA-4 “deg” (I-4.6 pg.61) 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.61), LF(I-12.8 pg.150)

I-23.2 WeightsMatrix

syntax

```
WeightsMatrix(R: RING): MAT
```

Description

This function returns the weights matrix for the current ring.

example

```

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

/**/ deg(x*y);
2
/**/ wdeg(x*y);
[5]
```

See Also: [deg\(I-4.6 pg.61\)](#), [wdeg\(I-23.1 pg.267\)](#)

I-23.3 while

syntax

```

While B Do C EndWhile

where B is a boolean expression and C is a sequence of commands.
```

Description

The command sequence C is repeated until B evaluates to False.

example

```

/**/ N := 0;
/**/ while N <= 5 do
/**/   PrintLn 2, "^", N, " = ", 2^N;
/**/   N := N+1;
/**/ EndWhile;
2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
2^4 = 16
2^5 = 32
```

See Also: [for\(I-6.13 pg.84\)](#), [foreach\(I-6.14 pg.85\)](#), [repeat\(I-18.22 pg.219\)](#)

I-23.4 WithoutNth

syntax

```

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

Description

This function returns the list obtained by removing the “N”-th component of the list “L”. The list “L” itself is not changed; compare with “[remove](#)” ([I-18.21 pg.219](#)).

example

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

See Also: [insert\(I-9.24 pg.123\)](#), [remove\(I-18.21 pg.219\)](#)

I-23.5 WLog

syntax

[OBSOLETE]

Description

[OBSOLETE] This function returns the weighted list of exponents of the leading term of F, as determined by the first row of the weights matrix.

See Also: [log\(I-12.17 pg.155\)](#)

Chapter I-24

X

I-24.1 XelMat

syntax

```
XelMat(N: INT): MAT
```

Description

This function return the matrix defining a standard term-ordering.

example

```
/**/ XelMat(3);  
matrix([  
  [0, 0, 1],  
  [0, 1, 0],  
  [1, 0, 0]  
])
```

See Also: OrdMat([I-15.10](#) pg.189), Orderings([III-9.5](#) pg.329), StdDegRevLexMat([I-19.27](#) pg.243), StdDegLexMat([I-19.26](#) pg.242), LexMat([I-12.6](#) pg.149), RevLexMat([I-18.29](#) pg.222)

Chapter I-25

Z

I-25.1 zero

syntax

```
zero(R: RING): RINGELEM
```

Description

This function return the additive identity of a ring. For when you want to force the integer “0” to be a “RINGELEM”.

example

```
/**/ P := ZZ/(101)[x,y,z];

/**/ N := 0; Print N, " of type ", type(N);
0 of type INT
/**/ N := zero(P); Print N, " of type ", type(N);
0 of type RINGELEM
/**/ N := 300*0; Print N, " of type ", type(N);
0 of type INT
/**/ N := 300*zero(P); Print N, " of type ", type(N);
0 of type RINGELEM

/**/ F := NewFreeModule(P, 3);
/**/ zero(F);
[0, 0, 0]
```

See Also: [one\(I-15.1 pg.185\)](#), [IsZero\(I-9.66 pg.140\)](#)

I-25.2 ZeroMat

syntax

```
ZeroMat(R: RING, NumRows: INT, NumCols: INT): MAT
```

Description

This function returns the “NumRows x NumCols” zero matrix with entries in “R”.

example

```
/**/ Use R := QQ[x,y,z];
/**/ ZeroMat(QQ, 1, 3); --> same as NewMatFilled(1,3, 0)
matrix(QQ,
```

```
[[0, 0, 0]]
/**/ ZeroMat(R, 1, 3); --> same as NewMatFilled(1,3, zero(R))
matrix( /*RingDistrMPolyClean(QQ, 3)*/
[[0, 0, 0]])
```

See Also: [matrix\(I-13.8 pg.162\)](#), [IdentityMat\(I-9.6 pg.114\)](#), [NewMatFilled\(I-14.7 pg.173\)](#)

I-25.3 ZPQ

syntax

```
ZPQ(F: RINGELEM): RINGELEM
ZPQ(F: LIST of RINGELEM): LIST of RINGELEM
ZPQ(I: IDEAL): IDEAL
```

Description

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

The function “ZPQ” maps a polynomial with finite field coefficients into one with rational (actually, integer) coefficients. It is not uniquely defined mathematically, and currently for each coefficient the least non-negative equivalent integer is chosen. Users should not rely on this choice, though any change will be documented.

See “QZP” ([I-17.4 pg.206](#)) for more details.

example

```
Use R := QQ[x,y,z];
F := 1/2*x^3 + 34/567*x*y*z - 890; -- a poly with rational coefficients
Use S := ZZ/(101)[x,y,z];
ZPQ(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\(I-2.10 pg.35\)](#), [image\(I-9.9 pg.115\)](#)

I-25.4 ZZ

syntax

```
ZZ
```

Description

This system variable is constant; its value is the ring of integers. Its name is protected so that it cannot be re-assigned to any other value.

example

```
/**/ P := ZZ/(101)[x,y,z];

/**/ Use ZZ;
```

```
/**/ type(5);  
INT  
/**/ type(RingElem(ZZ, 5));  
RINGELEM
```

See Also: [QQ\(I-17.1 pg.205\)](#)

Part II

The CoCoA Programming Language

Chapter II-1

Introduction to CoCoA Programming

II-1.1 An Overview of CoCoA Programming

The CoCoA system includes a full-fledged high level programming language, CoCoALanguage, complete with loops, branching, scoping of variables, and input/output control. The language is used whenever one issues commands during a CoCoA session. A sequence of commands may be stored in a text file and then read into a CoCoA session using the “**source**” ([I-19.17 pg.238](#)) command.

The most important construct in CoCoA programming is the user-defined function, created with “**define**” ([I-4.4 pg.60](#)). 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.

Chapter II-2

Language Elements

II-2.1 Character Set and Special Symbols

The CoCoA character set consists of the 26 lower case letters, the 26 upper case letters, the 10 digits and the special characters listed in the table below. Note that the special character “|” looks a bit different on some keyboards (its ASCII code is 124).

	blank	_	underscore	(left parenthesis	
	+	plus	=	equal)	right parenthesis
	-	minus	<	less than	[left bracket
	*	asterisk	<	greater than	[right bracket
	/	slash		vertical bar	'	single quote
	:	colon	.	period	" "	double quote
	^	caret	;	semicolon		
	,	comma	%	percent		

Special Characters

The character-groups listed in the table below are special symbols in CoCoA

	:=	assign	..	range	
	<<	input from	//	start line comment	
	<>	not equal	--	start line comment	
	><	Cartesian product	::=	ring definition	
	<=	less than or equal to	/*	start embedded comment	
	>=	greater than or equal to	*/	end embedded comment	

Special Character-groups

II-2.2 Identifiers

There are two types of identifiers or names.

- * Identifiers of ring indeterminates (see “NewPolyRing” ([I-14.8](#) pg.174))
- * Predefined or user-defined names (functions and CoCoALanguage variables).

See Also: Indeterminates([III-9.4](#) pg.329)

II-2.3 Reserved Names

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

The names in the following tables are reserved and cannot be used otherwise. The names in the first table are case insensitive (e.g. CLEAR, Clear and cLEaR are all reserved). The names in the second table are case sensitive.

...work in progress...

Alias	And	Block	Ciao	Define	
Describe	Do	Elif	Else	End	
EndBlock	EndTry		EndDefine	EndFor	
EndForeach	EndIf	EndPackage	EndRepeat	EndUsing	
EndWhile	Eof	False	For	Foreach	
Global	Help	If	In	IsIn	
NewLine	Not	On	Or	Package	
Print	PrintLn	Quit	Repeat	Record	
Return	Set	Skip	Source	Step	
Then	Time	To	True	Unset	
Until	Use	Using	Var	While	
QQ	ZZ				

Case insensitive reserved names

BOOL	DegLex	DegRevLex	DEVICE	ERROR	
FUNCTION	IDEAL	INT	LIST	Lex	
MAT	MODULE	NULL	Null	PANEL	
POLY	PosTo	RAT	RATFUN	RING	
STRING	TAGGED	ToPos	TYPE	MODULEELEM	
Xel	ZMOD				

Case sensitive reserved names

II-2.4 Comments

End-of-line comments in CoCoA start with either “--” or “//”; all text up to the end of the line is considered comment. CoCoA also allows embedded comments; these begin with the symbol “/*” and end with the symbol “*/”. CoCoA ignores the contents of a comment, and treats it as if it were just a space.

example

```
/**/ // This is an end-of-line comment
/**/ Print 1+1; -- a command followed by an end-of-comment
2
/**/ A := [1 /*x-coord*/, 2 /*y-coord*/ ]; --> embedded comments
```

Chapter II-3

Operators

II-3.1 CoCoA Operators

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.

The CoCoA operators are, from the highest to the lowest priority:

```
[ ] .      (selection operators)
^ %
+ -      (as unary operators)
* : /
+ -      (as binary operators)
..
= <> < <= > >=
IsIn
And
Or
```

Operations with equal priority are performed from left to right. When in doubt, parentheses may be used to enforce a particular order of evaluation.

See Also: operators, shortcuts(I-0.1 pg.21)

II-3.2 Algebraic Operators

The algebraic operators are:

+ - * / : ^

The following table shows which operations the system can perform between two objects of the same or of different types; the first column lists the type of the first operand and the first row lists the type of the second operand. So, for example, the symbol “:” in the box on the seventh row and fourth column means that it is possible to divide an ideal by a polynomial.

	INT	RAT	RINGELEM	MODULEELEM	IDEAL	MODULE	MAT	LIST
INT	+-*/^	+-*/	+-*/	*	*	*	*	*
RAT	+-*/^	+-*/	+-*/	*	*	*	*	*

RINGELEM	+*/^	+*/	+*/	*	*	*	*	*
MODULEELEM	*	*	*	+-				
IDEAL	*^	*	*		++:	*		
MODULE	*	*	*		*	+:		
MAT	*^	*	*				+-*	
LIST	*	*	*					+-

Algebraic operators

Remarks:

* Let F and G be two polynomials. If F is a multiple of G , then F/G is the polynomial obtained from the division of F by G , otherwise F/G is a rational function (common factors are simplified). The functions “div” (I-4.20 pg.68) and “mod” (I-13.20 pg.167) can be used to get the quotient and the remainder of a polynomial division.

* Let L_1 and L_2 be two lists of the same length. Then $L_1 + L_2$ is the list obtained by adding L_1 to L_2 componentwise.

* If I and J are both ideals or both modules, then $I : J$ is the ideal consisting of all polynomials f such that fg is in I for all g in J .

II-3.3 Relational Operators

See Also: Equality Test(I-5.6 pg.73), Comparison Operators(I-3.28 pg.48), IsIn(I-9.45 pg.131)

II-3.4 Selection Operators

The selection operators are

`[]` .

Let N be of type INT and let L be of type STRING, MODULEELEM, LIST, or MAT. Then the meaning of $L[N]$ depends on the type of L as explained in the following table:

Type of L	Meaning of $L[N]$
STRING	string consisting of the N -th character of L .
MODULEELEM	N -th component of L
LIST	N -th element of L
MAT	N -th element of L

Selection Operator

If N is an identifier and L is of type RECORD, then “ $L.N$ ” indicates the object contained in the field N of the record L (see “record” (I-18.12 pg.215)).

See Also: record(I-18.12 pg.215), List Constructors(I-12.15 pg.154)

II-3.5 Range Operator

If M and N are of type INT, then the expression: “ $M \dots N$ ” returns

- * the list “[M , $M+1$, \dots , N]” if $M \leq N$;
- * the empty list, “[]”, otherwise.

NOTE: Large values for M and N are not permitted; typically they should lie in the range about -10^9 to $+10^9$.

NOTE: see example for how to select a sub-range of a list

If x and y are indeterminates in a ring, then “x .. 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([II-3.1](#) pg.283)

Chapter II-4

Evaluation and Assignment

II-4.1 Evaluation

An expression is by itself a valid command. The effect of this command is that the expression is evaluated in the current ring and its value is displayed.

The evaluation of an expression in CoCoA is normally performed in a full recursive evaluation mode. Usually the result is the fully evaluated expression.

The result of the evaluation is automatically stored in the variable “It” ([I-9.70 pg.141](#)).

— 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-9.70 pg.141](#)).

If an error occurs during the evaluation of an expression, then the evaluation is interrupted and the user is notified about the error.

II-4.2 Assignment

An assignment command has the form

$$L := E$$

where L is a variable and E is an expression. The assignment command binds the result of the evaluation of the expression E to L in the working memory.

— example —

```
/**/ Use R ::= QQ[t,x,y,z];  
/**/ I := ideal(x,y);  
  
/**/ M := 5; N := 8;  
/**/ T := M+N;  
/**/ T;
```

```
13
/**/ T := T+1;  -- note that T occurs on the right, also
/**/ T;
14

/**/ L := [1,2,3];
/**/ L[2] := L[3];
/**/ L;
[1, 3, 3]

/**/ P := record[F := x*z];
/**/ P.Degree := Deg(P.F);
/**/ P;
record[Degree := 2, F := x*z]
```

Chapter II-5

Flow Control: Conditional Statements and Loops

II-5.1 Commands and Functions for Branching

The following are the CoCoA commands for constructing conditional statements:

`if` conditional statement

II-5.2 Commands and Functions for Loops

The following are the commands and functions for loops:

<code>break</code>	break out of a loop
<code>for</code>	loop command
<code>foreach</code>	loop command
<code>repeat</code>	loop command
<code>return</code>	exit from a function
<code>while</code>	loop command

Chapter II-6

Input/Output

II-6.1 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-19.17 pg.238) command—be executed as CoCoA commands.

II-6.2 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-16.21 pg.199) and “*println*” (I-16.25 pg.201) for more control over the format of the output.

The official devices that are being used here are “*DEV.STDIN*” and “*DEV.OUT*”. So for instance, the commands “*Get*” (I-7.10 pg.96) and “*print on*” (I-16.22 pg.200) can be used with the standard devices although they are really meant to be used with the other devices. “*Print E On DEV.OUT*” is synonymous with “*Print E*”. Also, one may use “*Get(DEV.STDIN,10)*”, for example, to get the next 10 characters typed in the CoCoA window. Thus, clever use of “*Get*” (I-7.10 pg.96) 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-6.3 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.187) then prints to the file using “*print on*” (I-16.22 pg.200).

To receive verbatim input from a file, one first opens the file with “*OpenIFile*” (I-15.2 pg.185), then gets characters from the file with “*Get*” (I-7.10 pg.96). Actually, “*Get*” (I-7.10 pg.96) gets a list of ASCII codes for the characters in the file. These can be converted to real characters using the function “*ascii*” (I-1.13 pg.28).

example

```
D := OpenOFile("my-file"); -- open text file with name "my-file",
                           -- creating it if necessary
```

```

Print "hello world" On D; -- append "hello world" to my-file
Close(D); -- close the file
D := OpenIFile("my-file"); -- open "my-file"
Get(D,10); -- get the first ten characters, in ASCII code
[104, 101, 108, 108, 111, 32, 119, 111, 114, 108]
-----
  ascii(It); -- convert the ASCII code
hello worl
-----
Close(D);

```

To read and execute a sequence of CoCoA commands from a text file, one uses the “**source**” (I-19.17 pg.238) 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\(I-1.13 pg.28\)](#), [close\(I-3.13 pg.41\)](#), [Get\(I-7.10 pg.96\)](#), [OpenIFile\(I-15.2 pg.185\)](#), [OpenOFile\(I-15.5 pg.187\)](#), [OpenLog\(I-15.4 pg.186\)](#), [CloseLog\(I-3.14 pg.42\)](#), [print on\(I-16.22 pg.200\)](#), [source\(I-19.17 pg.238\)](#)

II-6.4 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.188) to “*open*” the string, then “**print on**” (I-16.22 pg.200) to write to it. To read from a string, one may open the string for input with “**OpenIString**” (I-15.3 pg.186) then get characters from it with “**Get**” (I-7.10 pg.96).

example

```

S := "hello world";
D := OpenIString("", S); -- open the string S for input to CoCoA
      -- the first argument is just a name for the device
L := Get(D,7); -- read 7 characters from the string
L; -- ASCII code
[104, 101, 108, 108, 111, 32, 119]
-----
  ascii(L); -- convert ASCII code to characters
hello w
-----
Close(D); -- close device D
D := OpenOString(""); -- open a string for output from CoCoA
L := [1,2,3]; -- a list
Print L On D; -- print to D
D;
record[Name := "", Type := "OString", Protocol := "CoCoALanguage"]
-----
S := Cast(D, STRING); -- S is the string output printed on D
S; -- a string
[1, 2, 3]
Print " more characters" On D; -- append to the existing output string
Cast(D, STRING);
[1, 2, 3] more characters
-----

```

There are usually more direct ways to collect results in strings. For instance, if the output of a CoCoA command is not already of type STRING, one may convert it to a string using “**sprint**” (I-19.20 pg.239).

II-6.5 Commands and Functions for IO

The following are commands and functions for input/output:

<code>block</code>	group several commands into a single command
<code>close</code>	close a device
<code>CloseLog</code>	close a log of a CoCoA session
<code>format</code>	convert object to formatted string
<code>Get</code>	read characters from a device
<code>IO.SprintTrunc</code>	convert to a string and truncate
<code>Latex</code>	LaTeX formatting
<code>NewFreeModule</code>	create a new FreeModule
<code>NewLine</code>	[OBSOLESCENT] string containing a newline
<code>OpenIFile</code>	open input file
<code>OpenIString</code>	open input string
<code>OpenLog</code>	open a log of a CoCoA session
<code>OpenOFile</code>	open output file
<code>OpenOString</code>	open output string
<code>OpenSocket</code>	open a socket connection
<code>print</code>	print the value of an expression
<code>print on</code>	print to an output device
<code>println</code>	print the value of an expression
<code>source</code>	read commands from a file or device
<code>SourceRegion</code>	read commands from a region in a file
<code>sprint</code>	convert to a string
<code>tag</code>	returns the tag string of an object
<code>tagged</code>	tag an object for pretty printing
<code>untagged</code>	untag an object

II-6.6 Tagged Printing

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

Some CoCoA objects are intrinsically complicated, so printing them verbatim might be confusing. For this reason a mechanism has been implemented which enables automatic pretty printing through the use of “*tags*”. A user may “*tag*” any object with a string and then define how objects tagged with that string should be printed or described. Commands that do not have to do with printing ignore the tag.

II-6.7 Tagging an Object

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

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 type of the returned object is “`TAGGED(S)`” (but the type of *E* is unchanged). The function “`tag`” ([I-20.1 pg.253](#)) returns the tag string of an object, and the function “`untagged`” ([I-21.4 pg.262](#)) (or “`@`”) returns the object, stripped of its tag.

example

```
L := ["Dave","March 14, 1959",372];
M := Tagged(L, "MiscData"); -- L has been tagged with the string "MiscData"
type(L); -- L is a list
LIST
```

```

-----
  type(M); -- M is a tagged object
TAGGED("MiscData")
-----
  Tag(M); -- the tag string of M (it would be the empty string if M
          -- were not a tagged object).
MiscData
-----
  M; -- Until a special print function is defined, the printing of L
      -- and M is identical.
["Dave", "March 14, 1959", 372]
-----
  Untagged(M) = L; -- "untagged" removes the tag from M, recovering L.
True
-----

```

The next section explains how to define functions for pretty printing of tagged objects.

II-6.8 Printing a Tagged Object

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

Suppose the object E is tagged with the string S. When one tries to print E—say with “Print E” or just “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]; -- continuing with the previous example
  M := Tagged(L,"MiscData");
  M; -- M is printed as normal in the absence of a function "Print_MiscData"
["Dave", "March 14, 1959", 372]
-----
  Define Print_MiscData(X) -- Exactly one parameter is required.
    M := Untagged(X);
    Print M[1];
  EndDefine;
  Print M; -- Now, any object tagged with the string "MiscData" will be
           -- printed using Print_MiscData
Dave
-----
  M; -- Whenever printing of M is called for, "Print_MiscData" is executed.
Dave
-----

```

The line “M := Untagged(X)” is actually not necessary here, but in general one may get into an infinite loop trying to print X, a tagged object, from within the function that is being defined in order to print X, if that makes sense. Untagging X prevents this problem.

II-6.9 Describing a Tagged Object

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

If the object E is tagged with the string S, then when the user enters the command “Describe E”, CoCoA first looks for a user-defined function with name “Describe_S” and executes it; if not found, the output depends on the type of Untagged(E).

```

----- example -----
Use R ::= QQ[x,y,z];
I := ideal(x-y^2, x-z^3);
I := Tagged(I,"MyIdeals"); -- I is now tagged with "MyIdeals"
Describe I; -- the default description of an ideal
record[Type := IDEAL, Value := record[Gens := [-y^2 + x, -z^3 + x]]]
-----
Define Describe_MyIdeals(X)
  Y := Untagged(X);
  PrintLn "The generators are:";
  Foreach G In Y.Gens Do
    PrintLn"  ", G;
  EndForeach;
EndDefine;
Describe I; -- Any object tagged with "MyIdeals" is now described
            -- using "Describe_MyIdeals".
The generators are:
  -y^2 + x
  -z^3 + x
-----

```

II-6.10 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

Chapter II-7

CoCoA Packages

II-7.1 Introduction to Packages

User-defined functions may be saved in separate files and read into a CoCoA session using the “**source**” ([I-19.17 pg.238](#)) command. If one sources several such files or, especially, if a file is to be made available for general use, a possible problem arises from conflicting function names. If two functions with the same name are read into a CoCoA session, only the one last read survives. To avoid this, functions may be collected in “*packages*”.

A CoCoA package is essentially a list of functions labeled with prefix.

Writing a package in CoCoA-5 is slightly different from how it was done in CoCoA-4 it is easier!).

See Also: [define\(I-4.4 pg.60\)](#), [source\(I-19.17 pg.238\)](#)

II-7.2 First Example of a Package

The following is an example of a package. It could be typed into a window as-is during a CoCoA session, but we will assume that it is stored in a file in the CoCoA directory under the name “**one.cpkg5**”.

example

```
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:

example

```
-- read in the package:
Source "one.cpkg";
```

```

/* */ ToyTest(4); -- was exported
Not the number 1
/* */ IsNumberOne(4); -- wasn't exported
ERROR: Cannot find a variable named "IsNumberOne" in scope
IsNumberOne(4);
~~~~~

/* */ $contrib/toypackage.IsNumberOne(4);
false

```

II-7.3 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. There are no restrictions on the string PackageName, but keep in mind that it serves to distinguish functions in the package from those in all other CoCoA packages. A name of the form “contrib/subject” is typical.

All packages in the CoCoA directory “packages” are automatically loaded when starting CoCoA.

II-7.4 Global Aliases

A global alias for a package is formed by using the command “alias” (I-1.7 pg.25) during a CoCoA session. Note: global aliases cannot be used in function definitions. This is to force independence of context. Inside a function, one must use the complete package name.

See Also: alias(I-1.7 pg.25), aliases(I-1.8 pg.26)

II-7.5 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-7.6 Commands and Functions for Packages

The following are commands and functions for packages:

alias	define aliases for package names
aliases	list of global aliases
Packages	list of loaded packages
PkgName	returns the name of a package

II-7.7 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-7.8 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-7.9 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-7.10 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-7.11 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-7.12 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-7.13 Geometrical Theorem-Proving

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

```

TITLE      : thmproving.cpkg
DESCRIPTION : CoCoA package for geometrical theorem-proving in euclidean space
AUTHOR     : L. Bazzotti, G. Dalzotto

-- Enter
    $contrib/thmproving.Man();
    to get a complete description of the package including a suggested alias.

```

II-7.14 Typevectors

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

```

TITLE      : typevectors.cpkg
DESCRIPTION : CoCoA package for computing type-vectors associated to
              Hilbert functions of ideals of points
AUTHOR     : E.Carlini, M.Stewart

-- Enter
    $contrib/typevectors.Man();
    to get a complete description of the package including a suggested alias.

```

II-7.15 Conductor

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

```

TITLE      : conductor.cpkg

```

DESCRIPTION : CoCoA package for computing conductor sequence of points
 AUTHOR : L.Bazzotti

```
-- Enter
    $contrib/conductor.Man();
to get a complete description of the package including a suggested alias.
```

II-7.16 Matrix Normal Form

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

TITLE : matrixnormalform.cpkg
 DESCRIPTION : CoCoA package for computing normal forms of a matrix,
 Smith Normal Form (PID)
 AUTHOR : A.Bigatti, S.DeFrancisci

```
-- Enter
    $contrib/matrixnormalform.Man();
to get a complete description of the package including a suggested alias.
```

II-7.17 CantStop

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

TITLE : CantStop.cpkg
 DESCRIPTION : CoCoA package for playing Can't Stop and studying strategies
 AUTHOR : A.Bigatti

```
-- Enter
    $contrib/CantStop.Man();
to get a complete description of the package including a suggested alias.
```

II-7.18 Control

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

TITLE : control.cpkg
 DESCRIPTION : CoCoA package for Geometric Control Theory
 AUTHOR : M. Anderlucchi and M. Caboara

```
-- Enter
    $contrib/control.Man();
to get a complete description of the package including a suggested alias.
```


Chapter II-8

Linked libraries

II-8.1 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-8.2 GMP

GMP “<https://gmplib.org>”

All arbitrary precision integer/rational/floating-point datatypes and operations are based on GMP.

II-8.3 Frobbby

Frobbby “<http://www.broune.com/frobbby>”

All functions starting with “Frb” are implemented in Frobbby.

II-8.4 Normaliz

Normaliz “<http://www.home.uni-osnabrueck.de/wbruns/normaliz>”

All functions starting with “Nmz” are implemented in Normaliz.

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-9.7 pg.114](#)) and “while” ([I-23.3 pg.268](#)), etc., inside CoCoA programs.

The relational operators

= <> < <= > >=

return boolean constants (see “Relational Operators” ([II-3.3 pg.284](#))).

The boolean operators are “and” ([I-1.10 pg.27](#)), “or” ([I-15.9 pg.189](#)), “IsIn” ([I-9.45 pg.131](#)). From version CoCoA-5.0.9 “not” ([I-14.22 pg.179](#)) is a function (instead of an operator).

See Also: Relational Operators([II-3.3 pg.284](#))

III-1.2 Commands and Functions for BOOL

and	boolean ”and” operator
Bool01	Convert a boolean to an integer
in	list element selector in list constructor
List Constructors	build a new list
not	boolean ”not” operator
or	boolean ”or” operators
TmpNBM	Numerical Border Basis of ideal of points

III-1.3 Commands and Functions returning BOOL

and	boolean ”and” operator
Comparison Operators	less than, greater than, ...
EqSet	checks if the set of elements in two lists are equal
Equality Test	test whether two values are equal or not
IsAntiSymmetric	checks if a matrix is anti-symmetric
IsConstant	checks if a ringelem is in the coefficient ring
IsContained	checks if A is Contained in B

<code>IsDiagonal</code>	checks if a matrix is diagonal
<code>IsDivisible</code>	checks if A is divisible by B
<code>IsElem</code>	checks if A is an element of B
<code>IsEven, IsOdd</code>	test whether an integer is even or odd
<code>IsFactorClosed</code>	test whether a list of PPs is factor closed
<code>IsField</code>	test whether a ring is a field
<code>IsFiniteField</code>	test whether a ring is a finite field
<code>IsHomog</code>	test whether given polynomials are homogeneous
<code>IsIn</code>	check if one object is contained in another
<code>IsIndet</code>	checks argument is an indeterminate
<code>IsInRadical</code>	check if a polynomial (or ideal) is in a radical
<code>IsLexSegment</code>	checks if an ideal is lex-segment
<code>IsPositiveGrading</code>	check if a matrix defines a positive grading
<code>IsPrime</code>	prime integer test
<code>IsProbPrime</code>	checks if an integer is a probable prime
<code>IsPthPower</code>	p-th power test
<code>IsQuotientRing</code>	test whether a ring is a quotient ring
<code>IsStable</code>	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>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>IsZeroDim</code>	test whether an ideal is zero-dimensional
<code>IsZeroDivisor</code>	test whether a RINGELEM is a zero-divisor
<code>NFsAreZero</code>	test if normal forms are zero
<code>not</code>	boolean "not" operator
<code>or</code>	boolean "or" operators

Chapter III-2

INT

III-2.1 Introduction to INT

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

There are three types of numbers recognized by CoCoA: integers (type “INT”), rationals (type “RAT”), and modular integers (type “ZMOD”). 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;
```

The last line in the above example does not return 3; it is interpreted as a comment.

III-2.2 Commands and Functions for INT

abs	absolute value of a number
AffHilbert	the affine Hilbert function
AffHilbertFn	the affine Hilbert function
ascii	convert between characters and ascii code
AsINT	convert into an INT
AsRAT	convert into a RAT
binomial	binomial coefficient
BinomialRepr, BinomialReprShift	binomial representation of integers
ContFracToRat	convert continued fraction to rational
CRT	Chinese Remainder Theorem
cyclotomic	n-th cyclotomic polynomial
date	the date
DecimalStr	convert rational number to decimal string
den	denominator
DensePoly	the sum of all power-products of a given degree
div	quotient for integers
ElimMat	matrix for elimination ordering
EvalHilbertFn	evaluate the Hilbert function
Ext	presentation Ext modules as quotients of free modules

factorial	factorial function
FactorMultiplicity	multiplicity of a factor of an integer
first	the first N elements of a list
flatten	flatten a list
FloatApprox	approx. of rational number of the form $M * 2^E$
FloatStr	convert rational number to a decimal string
format	convert object to formatted string
GBasisTimeout	compute a Groebner basis with a timeout
gcd	greatest common divisor
GCDFreeBasis	determine (minimal) GCD free basis of a set of integers
GenericPoints	random projective points
Get	read characters from a device
GetCol	convert a column of a matrix into a list
GetRow	convert a row of a matrix into a list
Gin, Gin5	generic initial ideal
hilbert	the Hilbert-Poincare' function
HilbertFn	the Hilbert function
IdentityMat	the identity matrix
ILogBase	integer part of the logarithm
incr, decr	increment/decrement a counter
indent	prints in a more readable way
indet	individual indeterminates
insert	insert an object in a list
InverseSystem	Inverse system of an ideal of derivations
IO.SprintTrunc	convert to a string and truncate
iroot	integer part of r-th root of an integer
IsEven, IsOdd	test whether an integer is even or odd
IsPositiveGrading	check if a matrix defines a positive grading
IsPrime	prime integer test
IsProbPrime	checks if an integer is a probable prime
isqrt	(truncated) square root of an integer
IsZero	test whether an object is zero
IsZeroCol, IsZeroRow	test whether a column(row) is zero
last	the last N elements of a list
lcm	least common multiple
LexMat	matrices for std. term-orderings
LogToTerm	returns a monomial (power-product) with given exponents
MakeMatByRows, MakeMatByCols	convert a list into a matrix
MantissaAndExponent10	convert rational number to a float
MantissaAndExponent2	convert rational number to a binary float
Max	a maximum element of a sequence or list
Min	a minimum element of a sequence or list
minors	list of minor determinants of a matrix
mod	remainder for integers
NewFreeModule	create a new FreeModule
NewList	create a new list
NewMat	Zero matrix
NewMatFilled	matrix filled with value
NewPolyRing	create a new PolyRing
NewRingFp	create a new finite field
NewRingTwinFloat	create a new twin-float ring
NextPrime	find the next largest prime number
NextProbPrime	find the next largest probable prime number
num	numerator
NumPartitions	number of partitions of an integer
operators, shortcuts	Special characters equivalent to commands
partitions	partitions of an integer

PowerMod	compute a modular power efficiently
PrimitiveRoot	find a primitive root modulo a prime
product	the product of the elements of a list
random	random integer
randomized	randomize the coefficients of a given polynomial
RatReconstructByContFrac, RatReconstructByLattice	rational reconstruction from modular image
RatReconstructWithBounds	deterministic rational reconstruction from modular image
RefineGCDFreeBasis	refine an integer GCD free basis
remove	remove an object in a list
RevLexMat	matrices for std. term-orderings
RingElem	convert an expression into a RINGELEM
RingQQt	pre-defined polynomial rings
ScientificStr	convert integer/rational to a floating-point string
seed	seed for “random”
sign	the sign of a number
SmoothFactor	find small prime factors of an integer
SourceRegion	read commands from a region in a file
spaces	return a string of spaces
StarPrint, StarSprint	print polynomial with *’s for multiplications
StdDegLexMat	matrices for std. term-orderings
StdDegRevLexMat	matrices for std. term-orderings
submat	submatrix
subsets	returns all sublists of a list
sum	the sum of the elements of a list
SymbolRange	range of symbols for the indeterminates of a PolyRing
syz	syzygy modules
tuples	N-tuples
valuation	p-adic valuation
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
AffHilbert	the affine Hilbert function
AffHilbertFn	the affine Hilbert function
AsINT	convert into an INT
binomial	binomial coefficient
BinomialRepr, BinomialReprShift	binomial representation of integers
Bool01	Convert a boolean to an integer
ceil	round rational up to integer
characteristic	the characteristic of a ring
ContFrac	continued fraction quotients
count	count the objects in a list
deg	the standard degree of a polynomial or moduleelem
den	denominator
Depth	Depth of a module
dim	the dimension of a ring or quotient object
div	quotient for integers
EvalHilbertFn	evaluate the Hilbert function
factorial	factorial function

FactorMultiplicity	multiplicity of a factor of an integer
floor	round rational down to integer
gcd	greatest common divisor
GradingDim	Number of components in weighted degree
hilbert	the Hilbert-Poincare' function
HilbertFn	the Hilbert function
ILogBase	integer part of the logarithm
IndetIndex	index of an indeterminate
iroot	integer part of r-th root of an integer
isqrt	(truncated) square root of an integer
lcm	least common multiple
len	the length of an object
LogCardinality	extension degree of a finite field
LPosn	the position of the leading power-product in a ModuleElem
MayerVietorisTreeN1	N-1st Betti multidegrees of monomial ideals using Mayer-Vietoris trees
MinPowerInIdeal	the minimum power of a polynomial in an ideal
mod	remainder for integers
multiplicity	the multiplicity (degree) of a ring or quotient object
NextPrime	find the next largest prime number
NextProbPrime	find the next largest probable prime number
num	numerator
NumCols	number of columns in a matrix
NumCompts	the number of components
NumIndets	number of indeterminates
NumPartitions	number of partitions of an integer
NumRows	number of rows in a matrix
NumTerms	number of terms in a polynomial
PowerMod	compute a modular power efficiently
PrimitiveRoot	find a primitive root modulo a prime
random	random integer
randomized	randomize the coefficients of a given polynomial
rank	rank of a matrix or module
Reg, Reg5	Castelnuovo-Mumford regularity of a module
RegularityIndex	regularity index of a Hilbert function or series
RootBound	bound on roots of a polynomial over QQ
round	round to integer
seed	seed for "random"
sign	the sign of a number
TimeOfDay	the current time
UnivariateIndetIndex	the index of the indeterminate of a univariate polynomial
valuation	p-adic valuation

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
```

III-3.2 Commands and Functions for RAT

<code>abs</code>	absolute value of a number
<code>AsINT</code>	convert into an INT
<code>AsRAT</code>	convert into a RAT
<code>ceil</code>	round rational up to integer
<code>CFApprox</code>	continued fraction approximation
<code>CFApproximants</code>	continued fraction approximants
<code>ContFrac</code>	continued fraction quotients
<code>DecimalStr</code>	convert rational number to decimal string
<code>FloatApprox</code>	approx. of rational number of the form $M * 2^E$
<code>FloatStr</code>	convert rational number to a decimal string
<code>floor</code>	round rational down to integer
<code>ILogBase</code>	integer part of the logarithm
<code>IsZero</code>	test whether an object is zero
<code>MantissaAndExponent10</code>	convert rational number to a float
<code>MantissaAndExponent2</code>	convert rational number to a binary float
<code>Max</code>	a maximum element of a sequence or list
<code>Min</code>	a minimum element of a sequence or list
<code>NewMatFilled</code>	matrix filled with value
<code>num</code>	numerator
<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>	computes the real roots of a univariate polynomial
<code>RealRootsApprox</code>	computes approximations to the real roots of a univariate polynomial
<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>sign</code>	the sign of a number
<code>SimplestRatBetween</code>	find simplest rational in a closed interval
<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>FloatApprox</code>	approx. of rational number of the form $M * 2^E$
<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: CoCoA still accepts an “*obsolescent*” syntax for string literals (between single-quotes); do not use this!

See Also: String Operations([III-4.2](#) pg.315), sprint([I-19.20](#) pg.239)

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-9.45 pg.131](#)) can be used to test if one string is a substring of another.

example

```
/**/  msg := "Banana";  
/**/  "ana" IsIn msg;  
true  
/**/  "Ana" IsIn msg;  --> substring must be an exact match  
false
```

See Also: String Literals([III-4.1 pg.315](#)), ascii([I-1.13 pg.28](#)), concat([I-3.31 pg.50](#)), IsIn([I-9.45 pg.131](#)), len([I-12.5 pg.149](#))

III-4.3 Commands and Functions for STRING

Chapter III-5

LIST

III-5.1 Introduction to Lists

A CoCoA list is a sequence of CoCoA objects between square brackets. In particular, a list may contain other lists. The empty list is “[]”. If L is a list and N is an integer, then $L[N]$ is the N -th component of L . If L contains sublists, then “ $L[N_1, N_2, \dots, N_s]$ ” is shorthand for “ $L[N_1][N_2] \dots [N_s]$ ” (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 are
[3, "ab"] -- 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)
```

III-5.2 Commands and Functions for LIST

append	append an object to a list
apply	apply homomorphism
ascii	convert between characters and ascii code
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 polynomial wrt an indet or a list of indets
ColMat	single column matrix
concat	concatenate lists
ConcatHorList	create a simple block matrix
ConcatLists	concatenate a list of lists
ConcatVerList	create a simple block matrix
ContentWRT	content of a polynomial wrt and indet or a list of indets
ContFracToRat	convert continued fraction to rational
count	count the objects in a list
DiagMat	matrix with given diagonal
diff	returns the difference between two lists
distrib	the distribution of objects in a list
DivAlg	division algorithm
elim	eliminate variables
ElimMat	matrix for elimination ordering
EqSet	checks if the set of elements in two lists are equal
eval	substitute numbers or polynomials for indeterminates
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
GBM	intersection of ideals for zero-dimensional schemes
gcd	greatest common divisor
GCDFreeBasis	determine (minimal) GCD free basis of a set of integers
HGBM	intersection of ideals for zero-dimensional schemes
HilbertSeriesShifts	the Hilbert-Poincare series
homog	homogenize with respect to an indeterminate
HomogElimMat	matrix for elimination ordering
ideal	ideal generated by list
IdealAndSeparatorsOfPoints	ideal and separators for affine points
IdealAndSeparatorsOfProjectivePoints	ideal and separators for points
IdealOfProjectivePoints	ideal of a set of projective points
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
InitialIdeal	Initial ideal
insert	insert an object in a list
Interpolate	interpolating polynomial
interreduce, interreduced	interreduce a list of polynomials
intersection	intersect lists, ideals, or modules
IntersectList	intersect lists, ideals, or modules
IsFactorClosed	test whether a list of PPs is factor closed
IsHomog	test whether given polynomials are homogeneous
IsIn	check if one object is contained in another
IsInSubalgebra	check if one polynomial is in a subalgebra
IsSubset	checks if the elements of one list are a subset of another
IsTree5	checks if a facet complex is a tree
jacobian	the Jacobian of a list of polynomials
last	the last N elements of a list
lcm	least common multiple
len	the length of an object
LexSegmentIdeal	lex-segment ideal containing L, or with the same Hilbert fn as I
List Constructors	build a new list
LogToTerm	returns a monomial (power-product) with given exponents
MakeMatByRows, MakeMatByCols	convert a list into a matrix
MakeSet	remove duplicates from a list

<code>matrix</code>	convert a list into a matrix
<code>Max</code>	a maximum element of a sequence or list
<code>Min</code>	a minimum element of a sequence or list
<code>ModuleElem</code>	create a module element
<code>ModuleOf</code>	the module environment of the object
<code>NewPolyRing</code>	create a new PolyRing
<code>NFsAreZero</code>	test if normal forms are zero
<code>NmzComputation</code>	flexible access to Normaliz
<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>operators, shortcuts</code>	Special characters equivalent to commands
<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>PoincareShifts</code>	the Hilbert-Poincare series
<code>PolyAlgebraHom</code>	homomorphism of polynomial algebras
<code>PolyRingHom</code>	homomorphism of polynomial rings
<code>product</code>	the product of the elements of a list
<code>QZP</code>	change field for polynomials and ideals
<code>RatReconstructWithBounds</code>	deterministic rational reconstruction from modular image
<code>RefineGCDFreeBasis</code>	refine an integer GCD free basis
<code>remove</code>	remove an object in a list
<code>reverse, reversed</code>	reverse a list
<code>RingSet</code>	list of the rings of an object
<code>RMap</code>	define ring homomorphism for function image
<code>RowMat</code>	single row matrix
<code>ScalarProduct</code>	scalar product
<code>SeparatorsOfPoints</code>	separators for affine points
<code>SeparatorsOfProjectivePoints</code>	separators for projective points
<code>shape</code>	extended list of types involved in an expression
<code>sort</code>	sort a list
<code>SortBy</code>	sort a list
<code>sorted</code>	sort a list
<code>SortedBy</code>	sort a list
<code>StableBBasis5</code>	Stable Border Basis of ideal of points
<code>StableIdeal</code>	stable ideal containing L
<code>StronglyStableIdeal</code>	strongly stable ideal containing L
<code>SubalgebraMap</code>	algebra homomorphism representing a subalgebra
<code>SubalgebraRepr</code>	representation of a polynomial as a subalgebra element
<code>submat</code>	submatrix
<code>submodule</code>	submodule generated by list
<code>subsets</code>	returns all sublists of a list
<code>sum</code>	the sum of the elements of a list
<code>syz</code>	syzygy modules
<code>tail</code>	remove the first element of a list
<code>TmpNBM</code>	Numerical Border Basis of ideal of points
<code>toric</code>	saturate toric ideals
<code>tuples</code>	N-tuples
<code>WithoutNth</code>	removes the N-th component from a list
<code>ZPQ</code>	change field for polynomials and ideals

Chapter III-6

RECORD

III-6.1 Introduction to RECORD

A record is a data type in CoCoA representing a list of bindings of the form “*name to object*”.

example

```
/**/ Use R ::= QQ[x,y,z];
/**/ P := record[ I := ideal(x,y^2-z), F := x^2 + y, Misc := [1,3,4]];
/**/ P.I;
ideal(x, y^2 -z)
/**/ P["I"];
ideal(x, y^2 -z)

/**/ P.Misc;
[1, 3, 4]
/**/ P.Misc[2];
3

/**/ P.Date := "1/1/98";
/**/ indent(P);
record[
  Date := "1/1/98",
  F := x^2 +y,
  I := ideal(x, y^2 -z),
  Misc := [1, 3, 4]
]

/**/ P["Misc",3]; -- equivalent to P.Misc[3]
4
```

Each entry in a record is called a “*field*”. Note that records are “*open*” in the sense that their fields can be extended, as shown in the previous example. At present, there is no function for deleting fields from a record, one must rewrite the record, selecting the fields to retain:

example

```
/**/ P := record[A := 2, B := 3, C := 5, D := 7];
/**/ Q := record[];

Foreach F In Fields(P) Do
  If F <> "C" Then Q[F] := P[F]; EndIf;
EndForeach;

/**/ P := Q;
```

```

/**/ P;
record[A := 2, B := 3, D := 7]

```

III-6.2 Commands and Functions for RECORD

<code>fields</code>	list the fields of a record
<code>NmzComputation</code>	flexible access to Normaliz
<code>RealRootRefine</code>	refine a real root of a univariate polynomial
<code>record field selector</code>	select a field of a record
<code>shape</code>	extended list of types involved in an expression

III-6.3 Commands and Functions returning RECORD

<code>AlmostQR</code>	QR decomposition of a matrix
<code>CocoaLimits</code>	limits on exponents and ring characteristics
<code>ContentFreeFactor</code>	factorization of multivariate polynomial into content-free factors
<code>CRT</code>	Chinese Remainder Theorem
<code>DivAlg</code>	division algorithm
<code>eigenvectors</code>	eigenvalues and eigenvectors of a matrix
<code>factor</code>	factor a polynomial
<code>IdealAndSeparatorsOfPoints</code>	ideal and separators for affine points
<code>IdealAndSeparatorsOfProjectivePoints</code>	ideal and separators for points
<code>IndetSymbols</code>	the names of the indeterminates in a PolyRing
<code>LinearSimplify</code>	simplifying linear substitution for a univariate polynomial over a field
<code>MantissaAndExponent10</code>	convert rational number to a float
<code>MantissaAndExponent2</code>	convert rational number to a binary float
<code>NmzComputation</code>	flexible access to Normaliz
<code>PreprocessPts</code>	Reduce redundancy in a set of approximate points
<code>RatReconstructByContFrac, RatReconstructByLattice</code>	rational reconstruction from modular image
<code>RatReconstructWithBounds</code>	deterministic rational reconstruction from modular image
<code>RealRootRefine</code>	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>SmoothFactor</code>	find small prime factors of an integer
<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>TmpNBM</code>	Numerical Border Basis of ideal of points
<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-19.17 pg.238). 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.60) (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

ref	passing function parameters by reference
return	exit from a function
SortBy	sort a list
SortedBy	sort a list

`TopLevel` make a top level variable accessible

III-7.4 Commands and Functions returning **FUNCTION**

<code>define</code>	define a function
<code>func</code>	Anonymous function
<code>TopLevelFunctions</code>	returns the functions available at top-level

Chapter III-8

TYPE

III-8.1 Commands and Functions for TYPE

`describe` information about an object

III-8.2 Commands and Functions returning TYPE

<code>CurrentTypes</code>	lists all data types
<code>shape</code>	extended list of types involved in an expression
<code>type</code>	the data type of an expression

Chapter III-9

RING

III-9.1 Introduction to Rings

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

Polynomial rings play a central role in CoCoA. Indeed, every object in CoCoA is defined over a base ring which is a polynomial ring. The user can define many rings, but at any time a “*current ring*” is active within the system. Most commands use the current ring as the base ring.

Once a ring has been defined, the system can handle the following mathematical objects defined over that ring:

- * numbers (integers, rationals, modular integers);
- * polynomials;
- * vectors of polynomials;
- * ideals;
- * modules (submodules of a free module);
- * lists of objects;
- * matrices of objects.

Variables containing ring-dependent objects such as polynomials, ideals, and modules are “*labeled*” by their ring. Variables containing objects such as integers which are not dependent on a particular ring are not labeled.

IMPORTANT NOTE: Starting with CoCoA 3.5, variables are no longer local to specific rings, i.e., all variables are accessible from all rings.

The next sections explain how to create a ring.

III-9.2 Polynomial Rings

CoCoA starts with the default (polynomial) ring “ $R = \mathbb{Q}\langle x, y, z \rangle$ ”. There is a simplified syntax for defining a polynomial ring; it must be preceded by the command “*use*” ([I-21.6 pg.263](#)) or by the symbol “ $::=$ ”

```
R ::= C[X:INDETS];           use C[X:INDETS];
R ::= C[X:INDETS], 0;         use C[X:INDETS], 0;
```

where “ R ” is the identifier of a CoCoALanguage variable, C is a RING is an expression defining a coefficient ring (\mathbb{Z} , \mathbb{Q} or $\mathbb{Z}/(N)$), X is an expression that defines the indeterminates, and the “ 0 ” is a pre-defined ordering (“*lex*”, “*deglex*”, “*degrevlex*”).

The modifiers are used to modify the default settings of the base ring. The modifiers are of three classes: term-orderings, weights, term orderings for modules. These classes are discussed in separate sections below. It is possible to have no modifiers. The default values are: DegRevLex for the term-ordering, 1 for the weight of each indeterminate, and ToPos for the module term-ordering.

After the ring is defined using the above syntax, it can be made to be the current ring with the command “use” (I-21.6 pg.263).

```

----- example -----
Use R ::= QQ[x,y,z]; -- define and use the ring R
S ::= ZZ/(103)[x,y], Lex; -- define the ring S with Lex term-ordering
CurrentRing(); -- the current ring is still R
QQ[x,y,z]
-----
Use S; -- now the ring S is the current ring
ZZ/(103)[x,y]
-----
Using R Do X := z^2-3 EndUsing; -- define a variable in R (not the current
                                -- ring)
Y := R::x^2+y^2+z^2; -- another way of defining a variable in R while
                      -- S is the current ring
T ::= QQ[x[1..4],y[1..5,1..5]], Elim(y[1..5,1]), ToPos; -- a more
                                                         -- complicated ring

```

III-9.3 Coefficient Rings

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

As mentioned above, the coefficient ring for a CoCoA polynomial ring may be any ring.

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. When integers modulo N are used, the system checks whether N is a prime number and, if it is not, a warning message is given. However non-prime integers are accepted. Hence it is possible to do some work with polynomials whose coefficients are not in a field, but it is up to the user to ensure that no illegal operation will be attempted. To find the upper limit for the characteristic in the third case, see the field “MaxChar” returned by the function “CocoaLimits” (I-3.16 pg.43); $N \leq 32767$ is typical. (Note: 32003 is prime)

IMPORTANT NOTE: Presently the implementation of the Buchberger algorithm for computing Groebner bases operates correctly only if polynomials have coefficients in a field. So the high level operations on ideals and modules (and the polynomial GCD) involving Groebner bases computations work only if polynomials have coefficients in a field. Otherwise it is very likely that the user will run into trouble.

“CoeffRing”: When creating a new ring, the word “CoeffRing” may be used to refer to the current coefficient ring. Examples below illustrate its use.

```

----- example -----
Use R ::= QQ[x,y]; -- coefficient ring is Q
S ::= ZZ/(5)[t]; -- coefficient ring is the field with 5 elements
T ::= ZZ[u,v]; -- A warning is issued if the coefficient ring
               -- is not a field.
-- WARNING: Coeffs are not in a field
-- GBasis-related computations could fail to terminate or be wrong
-----
Use R ::= ZZ/(2)[x,y,z]; CurrentRing(); -- these examples show the usage
                                         -- of "CoeffRing"
ZZ/(2)[x,y,z]
-----
Use S ::= CoeffRing[a,b]; CurrentRing();

```

ZZ/(2)[a,b]

III-9.4 Indeterminates

An “*indeterminate*” is represented by an identifier followed by one or more 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 Orderings

Polynomials are always sorted with respect to the ordering of their base ring. All the operations involving polynomials utilize and preserve this ordering. The user can define custom orderings or choose a predefined term-ordering (see “NewPolyRing” (I-14.8 pg.174))

The predefined term-orderings are:

- * standard-degree reverse lexicographic: “DegRevLex” (default)
- * standard-degree lexicographic: “DegLex”
- * pure lexicographic: “Lex” (no grading)
- * pure xel: “Xe1” (NOT YET IMPLEMENTED)

If the indeterminates are given in the order “x₁, ..., x_n”, then “x₁ > ... > x_n” with respect to Lex, and “x₁ < ... < x_n” with respect to Xel.

See Also: OrdMat(I-15.10 pg.189), elim(I-5.3 pg.72)

III-9.6 Module Orderings

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

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, T₁, T₂” in T(R), with T not equal to 1, and for all i, j in 1,...,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:

$$T_1 * e_i > T_2 * e_j \iff T_1 > T_2 \text{ in } R$$

$$\text{or, if } T_1 = T_2, i < j$$

* the ordering called “PosTo” defined by:

$$T_1 * e_i > T_2 * e_j \iff i < j$$

$$\text{or, if } i = j, T_1 > T_2 \text{ in } R.$$

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 Accessing Other Rings

There are a variety of ways of interacting with a ring outside of the current ring. If a variable contains an object which does not depend on a user-defined ring—for example an integer—that object can be immediately accessed and used within any ring. Built-in CoCoA functions should be smart enough to take into account the rings in which their value was defined, for example “GBasis” (I-7.1 pg.91), “LT” (I-12.22 pg.157), “wdeg” (I-23.1 pg.267),...

If you want to move an object from one ring to another, think mathematically and use a “RINGHOM” and “apply” (I-1.12 pg.28). However there are handy (slower) shortcuts like “image” (I-9.9 pg.115) and “BringIn” (I-2.10 pg.35), or the constructors “matrix” (I-13.8 pg.162) and “RingElem” (I-18.30 pg.223).

“QZP”, “ZPQ” NOT YET IMPLEMENTED.

See Also: Commands and Functions for RINGHOM (III-10.1 pg.333), Commands and Functions returning RINGHOM (III-10.2 pg.333), apply (I-1.12 pg.28), CanonicalHom (I-3.2 pg.37), PolyAlgebraHom (I-16.14 pg.195), PolyRingHom (I-16.15 pg.196), matrix (I-13.8 pg.162), RingElem (I-18.30 pg.223), image (I-9.9 pg.115), BringIn (I-2.10 pg.35), QZP (I-17.4 pg.206), ZPQ (I-25.3 pg.274), use (I-21.6 pg.263)

III-9.8 Quotient Rings

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

If “R” is a ring identifier and I is an ideal defined in “R”, then “R/I” represents the corresponding quotient ring. It has type “TAGGED(“Quotient”)”.

example

```
Use R := QQ[x,y];
I := ideal(y-x^2);
Q := R/I;
Hilbert(Q); -- the Hilbert function for Q
H(0) = 1
H(x) = 2 for x >= 1
-----
```

III-9.9 Commands and Functions for RING

AffHilbert	the affine Hilbert function
AffHilbertFn	the affine Hilbert function
BaseRing	the base ring of an object
CanonicalHom	canonical homomorphism
characteristic	the characteristic of a ring
CoeffEmbeddingHom	returns the coefficient embedding homomorphism of a polynomial ring
CoeffRing	the ring of coefficients of a polynomial ring
ColMat	single column matrix
DefiningIdeal	defining ideal of a quotient ring
DensePoly	the sum of all power-products of a given degree
DiagMat	matrix with given diagonal
dim	the dimension of a ring or quotient object
GenericPoints	random projective points
hilbert	the Hilbert-Poincare' function
HilbertFn	the Hilbert function
HilbertPoly	the Hilbert polynomial
HilbertSeries	the Hilbert-Poincare series
HVector	the h-vector of a module or quotient object
ideal	ideal generated by list
IdealOfPoints	ideal of a set of affine points
IdentityMat	the identity matrix
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
IsField	test whether a ring is a field
IsFiniteField	test whether a ring is a finite field
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
LogCardinality	extension degree of a finite field
LogToTerm	returns a monomial (power-product) with given exponents
matrix	convert a list into a matrix
multiplicity	the multiplicity (degree) of a ring or quotient object
NewFractionField	create a new fraction field
NewFreeModule	create a new FreeModule
NewMat	Zero matrix
NewPolyRing	create a new PolyRing
NewQuotientRing	create a new quotient ring
NumIndets	number of indeterminates
one	one of a ring
OrdMat	matrix defining a term-ordering
poincare	the Hilbert-Poincare series
PolyAlgebraHom	homomorphism of polynomial algebras
PolyRingHom	homomorphism of polynomial rings
RegularityIndex	regularity index of a Hilbert function or series
RingElem	convert an expression into a RINGELEM
RowMat	single row matrix
TmpNBM	Numerical Border Basis of ideal of points
WeightsMatrix	matrix of generalized weights for indeterminates
zero	zero of a ring
ZeroMat	matrix filled with 0

III-9.10 Commands and Functions returning RING

BaseRing	the base ring of an object
CoeffRing	the ring of coefficients of a polynomial ring
NewFractionField	create a new fraction field
NewPolyRing	create a new PolyRing
NewQuotientRing	create a new quotient ring
NewRingFp	create a new finite field
NewRingTwinFloat	create a new twin-float ring
RingOf	the ring of the object
RingQQ	the ring of rationals
RingQQt	pre-defined polynomial rings
RingSet	list of the rings of an object
RingZZ	the ring of integers

Chapter III-10

RINGHOM

III-10.1 Commands and Functions for RINGHOM

<code>apply</code>	apply homomorphism
<code>InducedHom</code>	homomorphism induced by a homomorphism
<code>PolyRingHom</code>	homomorphism of polynomial rings

III-10.2 Commands and Functions returning RINGHOM

<code>CanonicalHom</code>	canonical homomorphism
<code>InducedHom</code>	homomorphism induced by a homomorphism
<code>PolyAlgebraHom</code>	homomorphism of polynomial algebras
<code>PolyRingHom</code>	homomorphism of polynomial rings

Chapter III-11

RINGELEM

III-11.1 Introduction to RINGELEM

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

An object of type POLY in CoCoA represents a polynomial. To fix terminology: a polynomial is a sum of terms; each term is the product of a coefficient and power-product, a power-product being 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 R ::= QQ[x,y,z];
F := 3xyz + xy^2;
F;
xy^2 + 3xyz
-----
Use R ::= 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+xy;
G := x;
F/G;
x + y
-----
F/(x+z);
(x^2 + xy)/(x + z)
-----
F^2;
x^4 + 2x^3y + x^2y^2
-----
F^(-1);
```

```
1/(x^2 + xy)
-----
```

III-11.2 Evaluation of Polynomials

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

Polynomials may be evaluated using the function “`subst`” (I-19.35 pg.246). More generally, “`subst`” (I-19.35 pg.246) allows one to substitute polynomials from the current ring for the indeterminates of a given polynomial. If substitutions are to be made for each indeterminate, in order, it is easier to use “`eval`” (I-5.9 pg.75). For more general substitutions, see “`image`” (I-9.9 pg.115).

```

----- example -----
Use R ::= QQ[x,y,z];
F := x+y+z;
Eval(F,[2,1]); -- let x=2 and y=1 in F
z + 3
-----
Subst(F,[[x,2],[y,1]]); -- the same thing using Subst
z + 3
-----
Subst(F,y,1); -- the syntax is easier when substituting for a single
               -- indeterminate
x + z + 1
-----
Subst(F, [[y,x-y], [z,2]]); -- substitute x-y for y and 2 for z
2x - y + 2
-----
```

See Also: `eval`(I-5.9 pg.75), `image`(I-9.9 pg.115), `subst`(I-19.35 pg.246)

III-11.3 Commands and Functions for RINGELEM

<code>abs</code>	absolute value of a number
<code>apply</code>	apply homomorphism
<code>AsINT</code>	convert into an INT
<code>AsRAT</code>	convert into a RAT
<code>binomial</code>	binomial coefficient
<code>CharPoly</code>	characteristic polynomial of a matrix
<code>ClearDenom</code>	clear common denominator of a polynomial with rational coeffs
<code>coefficients</code>	list of coefficients of a polynomial
<code>CoefficientsWRT</code>	list of coeffs and PPs of a polynomial wrt an indet or a list of indets
<code>CoeffListWRT</code>	list of coefficients of a polynomial wrt and indet
<code>CoeffOfTerm</code>	coefficient of a term of a polynomial
<code>content</code>	content of a polynomial
<code>ContentFreeFactor</code>	factorization of multivariate polynomial into content-free factors
<code>ContentWRT</code>	content of a polynomial wrt and indet or a list of indets
<code>cyclotomic</code>	n-th cyclotomic polynomial
<code>DecimalStr</code>	convert rational number to decimal string
<code>deg</code>	the standard degree of a polynomial or moduleelem
<code>den</code>	denominator
<code>deriv</code>	the derivative of a polynomial or rational function
<code>DerivationAction</code>	Action of a derivation
<code>DF</code>	the degree form of a polynomial

<code>discriminant</code>	the discriminant of a polynomial
<code>DivAlg</code>	division algorithm
<code>eigenvectors</code>	eigenvalues and eigenvectors of a matrix
<code>elim</code>	eliminate variables
<code>eval</code>	substitute numbers or polynomials for indeterminates
<code>factor</code>	factor a polynomial
<code>FloatApprox</code>	approx. of rational number of the form $M * 2^E$
<code>FloatStr</code>	convert rational number to a decimal string
<code>FrbAlexanderDual</code>	Alexander Dual of monomial ideals
<code>gcd</code>	greatest common divisor
<code>GenRepr</code>	representation in terms of generators
<code>homog</code>	homogenize with respect to an indeterminate
<code>ideal</code>	ideal generated by list
<code>IndetIndex</code>	index of an indeterminate
<code>IndetName</code>	the name of an indeterminate
<code>IndetSubscripts</code>	the index of an indeterminate
<code>interreduce, interreduced</code>	interreduce a list of polynomials
<code>IsConstant</code>	checks if a ringelem is in the coefficient ring
<code>IsDivisible</code>	checks if A is divisible by B
<code>IsElem</code>	checks if A is an element of B
<code>IsHomog</code>	test whether given polynomials are homogeneous
<code>IsIn</code>	check if one object is contained in another
<code>IsIndet</code>	checks argument is an indeterminate
<code>IsInRadical</code>	check if a polynomial (or ideal) is in a radical
<code>IsInSubalgebra</code>	check if one polynomial is in a subalgebra
<code>IsPthPower</code>	p-th power test
<code>IsTerm</code>	checks if the argument is a term
<code>IsZero</code>	test whether an object is zero
<code>IsZeroDivisor</code>	test whether a RINGELEM is a zero-divisor
<code>jacobian</code>	the Jacobian of a list of polynomials
<code>LC</code>	the leading coefficient of a polynomial or ModuleElem
<code>lcm</code>	least common multiple
<code>LF</code>	the leading form of a polynomial or an ideal
<code>LinearSimplify</code>	simplifying linear substitution for a univariate polynomial over QQ
<code>LM</code>	the leading monomial of a polynomial or ModuleElem
<code>log</code>	the list of exponents of the leading term of a polynomial
<code>LPP</code>	the leading power-product of a polynomial or ModuleElem
<code>LT</code>	the leading term of an object
<code>MantissaAndExponent10</code>	convert rational number to a float
<code>MantissaAndExponent2</code>	convert rational number to a binary float
<code>MinPoly</code>	minimal polynomial of a matrix
<code>MinPowerInIdeal</code>	the minimum power of a polynomial is an ideal
<code>ModuleOf</code>	the module environment of the object
<code>monic</code>	divide polynomials by their leading coefficients
<code>monomials</code>	the list of monomials of a polynomial
<code>NewMatFilled</code>	matrix filled with value
<code>NF</code>	normal form
<code>NFsAreZero</code>	test if normal forms are zero
<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>NR</code>	normal reduction
<code>num</code>	numerator
<code>NumTerms</code>	number of terms in a polynomial
<code>PerpIdealOfForm</code>	Ideal of derivations annihilating a form
<code>product</code>	the product of the elements of a list
<code>PthRoot</code>	Compute p-th root

QZP	change field for polynomials and ideals
randomize	randomize the coefficients of a given polynomial
randomized	randomize the coefficients of a given polynomial
RealRoots	computes the real roots of a univariate polynomial
RealRootsApprox	computes approximations to the real roots of a univariate polynomial
resultant	the resultant of two polynomials
RingElem	convert an expression into a RINGELEM
RingOf	the ring of the object
RingSet	list of the rings of an object
RootBound	bound on roots of a polynomial over QQ
ScientificStr	convert integer/rational to a floating-point string
SqFreeFactor	compute a squarefree factorization
StarPrint, StarSprint	print polynomial with *'s for multiplications
SubalgebraMap	algebra homomorphism representing a subalgebra
SubalgebraRepr	representation of a polynomial as a subalgebra element
subst	substitute values for indeterminates
sum	the sum of the elements of a list
support	the list of terms of a polynomial or moduleelem
sylvester	the Sylvester matrix of two polynomials
syzy	syzygy modules
UnivariateIndetIndex	the index of the indeterminate of a univariate polynomial
wdeg	multi-degree of an polynomial
ZPQ	change field for polynomials and ideals

III-11.4 Commands and Functions returning RINGELEM

abs	absolute value of a number
apply	apply homomorphism
binomial	binomial coefficient
CharPoly	characteristic polynomial of a matrix
ClearDenom	clear common denominator of a polynomial with rational coeffs
CoeffEmbeddingHom	returns the coefficient embedding homomorphism of a polynomial ring
CoeffListWRT	list of coefficients of a polynomial wrt and indet
CoeffOfTerm	coefficient of a term of a polynomial
content	content of a polynomial
ContentWRT	content of a polynomial wrt and indet or a list of indets
cyclotomic	n-th cyclotomic polynomial
den	denominator
DensePoly	the sum of all power-products of a given degree
deriv	the derivative of a polynomial or rational function
det	the determinant of a matrix
DF	the degree form of a polynomial
discriminant	the discriminant of a polynomial
FirstNonZero	the first non-zero entry in a MODULEELEM
FirstNonZeroPosn	the first non-zero entry in a MODULEELEM
gcd	greatest common divisor
HilbertPoly	the Hilbert polynomial
homog	homogenize with respect to an indeterminate
indet	individual indeterminates
Interpolate	interpolating polynomial
interreduce, interreduced	interreduce a list of polynomials
InverseSystem	Inverse system of an ideal of derivations

<code>IsInSubalgebra</code>	check if one polynomial is in a subalgebra
<code>JanetBasis</code>	the Janet basis of an ideal
<code>LC</code>	the leading coefficient of a polynomial or <code>ModuleElem</code>
<code>lcm</code>	least common multiple
<code>LF</code>	the leading form of a polynomial or an ideal
<code>LM</code>	the leading monomial of a polynomial or <code>ModuleElem</code>
<code>LogToTerm</code>	returns a monomial (power-product) with given exponents
<code>LPP</code>	the leading power-product of a polynomial or <code>ModuleElem</code>
<code>LT</code>	the leading term of an object
<code>MinPoly</code>	minimal polynomial of a matrix
<code>monic</code>	divide polynomials by their leading coefficients
<code>NF</code>	normal form
<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>NR</code>	normal reduction
<code>num</code>	numerator
<code>one</code>	one of a ring
<code>pfaffian</code>	the Pfaffian of a skew-symmetric matrix
<code>PthRoot</code>	Compute p-th root
<code>QZP</code>	change field for polynomials and ideals
<code>randomize</code>	randomize the coefficients of a given polynomial
<code>randomized</code>	randomize the coefficients of a given polynomial
<code>ReducedGBasis</code>	compute reduced Groebner basis
<code>resultant</code>	the resultant of two polynomials
<code>RingElem</code>	convert an expression into a <i>RINGELEM</i>
<code>SubalgebraRepr</code>	representation of a polynomial as a subalgebra element
<code>SymbolRange</code>	range of symbols for the indeterminates of a <code>PolyRing</code>
<code>zero</code>	zero of a ring
<code>ZPQ</code>	change field for polynomials and ideals

Chapter III-12

IDEAL

III-12.1 Commands and Functions for IDEAL

BBasis5	Border Basis of zero dimensional ideal
colon	ideal or module quotient
Depth	Depth of a module
elim	eliminate variables
EquiIsoDec	equidimensional isoradical decomposition
FrbAlexanderDual	Alexander Dual of monomial ideals
FrbAssociatedPrimes	Associated primes of monomial ideals
FrbIrreducibleDecomposition	Irreducible decomposition of monomial ideals
FrbMaximalStandardMonomials	Maximal standard monomials of monomial ideals
FrbPrimaryDecomposition	Primary decomposition of monomial ideals
GBasis	calculate a Groebner basis
GBasisTimeout	compute a Groebner basis with a timeout
GenRepr	representation in terms of generators
gens	list of generators of an ideal
Gin, Gin5	generic initial ideal
HColon	ideal or module quotient
HilbertFn	the Hilbert function
HilbertSeries	the Hilbert-Poincare series
homog	homogenize with respect to an indeterminate
HSaturation	saturation of ideals
InitialIdeal	Initial ideal
intersection	intersect lists, ideals, or modules
IntersectList	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
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 Hilbert fn 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
MinGensGeneral	list of minimal generators
minimalize	remove redundant generators
minimalized	ideal, submodule with minimal generators
MinPowerInIdeal	the minimum power of a polynomial is an ideal
MonsInIdeal	ideal generated by the monomials in an ideal
NewQuotientRing	create a new quotient ring
NF	normal form
NFsAreZero	test if normal forms are zero
poincare	the Hilbert-Poincare series
PrimaryDecomposition	primary decomposition of an ideal
PrimaryPoincare	primary
PrintBettiDiagram	the diagram of the graded Betti numbers
PrintBettiMatrix	print the matrix of the graded Betti numbers
product	the product of the elements of a list
QuotientBasis	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	compute reduced Groebner basis
Reg, Reg5	Castelnuovo-Mumford regularity of a module
res	free resolution
RingOf	the ring of the object
RingSet	list of the rings of an object
saturate	saturation of ideals
sum	the sum of the elements of a list
syz	syzygy modules
SyzOfGens	syzygy module for a given set of generators
TgCone	tangent cone
toric	saturate toric ideals
ZPQ	change field for polynomials and ideals

III-12.2 Commands and Functions returning IDEAL

colon	ideal or module quotient
DefiningIdeal	defining ideal of a quotient ring
elim	eliminate variables
EquiIsoDec	equidimensional isoradical decomposition
GBM	intersection of ideals for zero-dimensional schemes
Gin, Gin5	generic initial ideal
HColon	ideal or module quotient
HGBM	intersection of ideals for zero-dimensional schemes
homog	homogenize with respect to an indeterminate
HSaturation	saturation of ideals
ideal	ideal generated by list
IdealOfPoints	ideal of a set of affine points
IdealOfProjectivePoints	ideal of a set of projective points
InitialIdeal	Initial ideal
intersection	intersect lists, ideals, or modules
IntersectList	intersect lists, ideals, or modules
ker	Kernel of a K-algebra homomorphism

<code>LexSegmentIdeal</code>	lex-segment ideal containing L, or with the same Hilbert fn as I
<code>LF</code>	the leading form of a polynomial or an ideal
<code>LT</code>	the leading term of an object
<code>minimalized</code>	ideal, submodule with minimal generators
<code>MonsInIdeal</code>	ideal generated by the monomials in an ideal
<code>PerpIdealOfForm</code>	Ideal of derivations annihilating a form
<code>PrimaryDecomposition</code>	primary decomposition of an ideal
<code>QZP</code>	change field for polynomials and ideals
<code>radical</code>	radical of an ideal
<code>RadicalOfUnmixed</code>	radical of an unmixed ideal
<code>saturate</code>	saturation of ideals
<code>StableIdeal</code>	stable ideal containing L
<code>StronglyStableIdeal</code>	strongly stable ideal containing L
<code>TgCone</code>	tangent cone
<code>toric</code>	saturate toric ideals
<code>ZPQ</code>	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-13.8 pg.162)). 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( /*RingDistrMPolyClean(QQ, 2)*/
  [[7, 10],
   [15, 22]])

/**/ x * N;
matrix( /*RingDistrMPolyClean(QQ, 2)*/
  [[x, 2*x],
   [3*x, 4*x]])

/**/ N + matrix([[x,x], [y,y]]);
matrix( /*RingDistrMPolyClean(QQ, 2)*/
  [[x +1, x +2],
   [y +3, y +4]])
```

III-13.2 Commands and Functions for MAT

<code>adjoint</code>	adjoint matrix
<code>AlmostQR</code>	QR decomposition of a matrix
<code>apply</code>	apply homomorphism
<code>BaseRing</code>	the base ring of an object
<code>BlockMat</code>	create a block matrix

BlockMat2x2	create a block matrix with 4 matrices
CharPoly	characteristic polynomial of a matrix
CompleteToOrd	completes a matrix to an ordering 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
eigenvalues	eigenvalues and eigenvectors of a matrix
ElimMat	matrix for elimination ordering
eval	substitute numbers or polynomials for indeterminates
FGLM5	perform a FGLM Groebner Basis conversion
GetCol	convert a column of a matrix into a list
GetCols	convert a matrix into a list of lists
GetRow	convert a row of a matrix into a list
GetRows	convert a matrix into a list of lists
HilbertBasisKer	Hilbert basis for a monoid
HilbertSeriesMultiDeg	the Hilbert-Poincare series wrt a multigrading
HomogElimMat	matrix for elimination ordering
IdealOfPoints	ideal of a set of affine points
inverse	multiplicative inverse
IsAntiSymmetric	checks if a matrix is anti-symmetric
IsDiagonal	checks if a matrix is diagonal
IsPositiveGrading	check if a matrix defines a positive grading
IsSymmetric	checks if a matrix is symmetric
IsTermOrdering	check if a matrix defines a term-ordering
IsZero	test whether an object is zero
IsZeroCol, IsZeroRow	test whether a column(row) is zero
LinKer	find the kernel of a matrix
LinKerBasis	find the kernel of a matrix
LinKerModP	find the kernel of a matrix
LinSolve	find a solution to a linear system
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
NmzHilbertBasis	Hilbert Basis of a monoid
NumCols	number of columns in a matrix
NumRows	number of rows in a matrix
pfaffian	the Pfaffian of a skew-symmetric matrix
PoincareMultiDeg	the Hilbert-Poincare series wrt a multigrading
PreprocessPts	Reduce redundancy in a set of approximate points
product	the product of the elements of a list
rank	rank of a matrix or module
RingOf	the ring of the object
RingSet	list of the rings of an object
submat	submatrix
sum	the sum of the elements of a list
TmpNBM	Numerical Border Basis of ideal of points
toric	saturate toric ideals
transposed	the transposition of a matrix

III-13.3 Commands and Functions returning MAT

<code>adjoint</code>	adjoint matrix
<code>apply</code>	apply homomorphism
<code>BlockMat</code>	create a block matrix
<code>BlockMat2x2</code>	create a block matrix with 4 matrices
<code>ColMat</code>	single column matrix
<code>CompleteToOrd</code>	completes a matrix to an ordering matrix
<code>ConcatAntiDiag</code>	create a simple block matrix
<code>ConcatDiag</code>	create a simple block matrix
<code>ConcatHor</code>	create a simple block matrix
<code>ConcatHorList</code>	create a simple block matrix
<code>ConcatVer</code>	create a simple block matrix
<code>ConcatVerList</code>	create a simple block matrix
<code>DiagMat</code>	matrix with given diagonal
<code>ElimMat</code>	matrix for elimination ordering
<code>GensAsCols, GensAsRows</code>	matrix of generators of a module
<code>HomogElimMat</code>	matrix for elimination ordering
<code>IdentityMat</code>	the identity matrix
<code>inverse</code>	multiplicative inverse
<code>jacobian</code>	the Jacobian of a list of polynomials
<code>LexMat</code>	matrices for std. term-orderings
<code>LinKer</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>matrix</code>	convert a list into a matrix
<code>NewMat</code>	Zero matrix
<code>NewMatFilled</code>	matrix filled with value
<code>NmzHilbertBasis</code>	Hilbert Basis of a monoid
<code>OrdMat</code>	matrix defining a term-ordering
<code>RevLexMat</code>	matrices for std. term-orderings
<code>RowMat</code>	single row matrix
<code>StdDegLexMat</code>	matrices for std. term-orderings
<code>StdDegRevLexMat</code>	matrices for std. term-orderings
<code>submat</code>	submatrix
<code>sylvester</code>	the Sylvester matrix of two polynomials
<code>TensorMat</code>	returns the tensor product of two matrices
<code>transposed</code>	the transposition of a matrix
<code>WeightsMatrix</code>	matrix of generalized weights for indeterminates
<code>XelMat</code>	matrices for std. term-orderings
<code>ZeroMat</code>	matrix filled with 0

Chapter III-14

MODULE

III-14.1 Commands and Functions for MODULE

BaseRing	the base ring of an object
colon	ideal or module quotient
elim	eliminate variables
GBasis	calculate a Groebner basis
GBasisTimeout	compute a Groebner basis with a timeout
GenRepr	representation in terms of generators
gens	list of generators of an ideal
GensAsCols, GensAsRows	matrix of generators of a module
HilbertSeries	the Hilbert-Poincare series
HilbertSeriesShifts	the Hilbert-Poincare series
homog	homogenize with respect to an indeterminate
IntersectList	intersect lists, ideals, or modules
IsContained	checks if A is Contained in B
IsElem	checks if A is an element of B
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
MinGensGeneral	list of minimal generators
minimalize	remove redundant generators
minimalized	ideal, submodule with minimal generators
ModuleElem	create a module element
ModuleOf	the module environment of the object
NF	normal form
NFsAreZero	test if normal forms are zero
NumCompts	the number of components
operators, shortcuts	Special characters equivalent to commands
PoincareShifts	the Hilbert-Poincare series
PrintBettiMatrix	print the matrix of the graded Betti numbers
rank	rank of a matrix or module
ReducedGBasis	compute reduced Groebner basis
res	free resolution
RingOf	the ring of the object
RingSet	list of the rings of an object
submodule	submodule generated by list
SubmoduleCols, SubmoduleRows	convert a matrix into a module
syz	syzygy modules
SyzOfGens	syzygy module for a given set of generators

III-14.2 Commands and Functions returning MODULE

<code>elim</code>	eliminate variables
<code>homog</code>	homogenize with respect to an indeterminate
<code>IntersectList</code>	intersect lists, ideals, or modules
<code>LT</code>	the leading term of an object
<code>minimalized</code>	ideal, submodule with minimal generators
<code>NewFreeModule</code>	create a new FreeModule
<code>submodule</code>	submodule generated by list
<code>SubmoduleCols, SubmoduleRows</code>	convert a matrix into a module
<code>syz</code>	syzygy modules
<code>SyzOfGens</code>	syzygy module for a given set of generators

Chapter III-15

MODULEELEM

III-15.1 Introduction to MODULEELEM

An object of type MODULEELEM in CoCoA represents a module element; in CoCoA this usually means an element of the free module “ P^r ”, where “ P ” is a polynomial ring. For “ v ” and “ w ” MODULEELEM in the same MODULE, and “ f ” RINGELEM in its base ring, the following are also MODULEELEM:

$+v$, $-v$, $f*v$, $v*f$, $v+w$, $v-w$

See “ModuleElem” ([I-13.22](#) pg.168).

III-15.2 Commands and Functions for MODULEELEM

III-15.3 Commands and Functions returning MODULEELEM

homog	homogenize with respect to an indeterminate
LM	the leading monomial of a polynomial or ModuleElem
LT	the leading term of an object
ModuleElem	create a module element
ModuleOf	the module environment of the object
NF	normal form
NR	normal reduction
ReducedGBasis	compute reduced Groebner basis