CoCoA 4.3 Manual

May 14, 2004

Contents

Part I

Chapter I-1

Preamble

I-1.1 Version

```
-------------------------------------------------------
---           ___/       ___/         \             ---
--           /     _ \  /     _ \    , \             --
--           \    |   | \    |   |  ___ \            --
---          ____, __/  ____, __/ _/    _\          ---
-------------------------------------------------------
--   Online Help : CoCoA 4.3                         --
--        Author : David Perkinson                   --
--         email : davidp@reed.edu                   --
--          addr : Reed College, Portland, OR, USA   --
--          date : 2003/12/23                        --
--------------------------------------------\-----------
```

I-1.2 Preface

CoCoA is a system for doing "*Computations in Commutative Algebra.*" It is one of the projects of an active research team in Computer Algebra at the University of Genova, Italy, and whose current members are: Lorenzo Robbiano (team manager), John Abbott, Anna Bigatti, Massimo Caboara, Martin Kreuzer, David Perkinson, and occasionally other researchers and students. Notable contributions from outside the team have been received from Volker Augustin and Arndt Wills. Much of the original code for CoCoA was written by Gianfranco Niesi and Antonio Capani.

I-1.3 System Distribution

CoCoA is public domain software, available at

"`http://cocoa.dima.unige.it`"

where there is also a great deal of information about the system. Besides the main site in Genoa, Italy, there is a mirror of the official distribution:

USA, West Coast: "`http://www.reed.edu/cocoa`"

CoCoA is distributed freely under the following condition: any research activity which uses CoCoA should cite the system in the following form:

```
CoCoATeam
CoCoA, a system for doing Computations in Commutative Algebra,
Available at http://cocoa.dima.unige.it
```

(see also "`http://cocoa.dima.unige.it/citing.html`")

The system can be freely redistributed to other users. New users should notify the CoCoA team at the email address below so they can be included in a users list. All members of this list will be kept up to date about the progress of the system.

The system is distributed "*as is*". The authors make no guarantee about the fitness of the system for any particular application. They will not be liable for any direct, indirect, special, incidental or consequential damages in connection with the use of the system or of the manual.

Bug reports, questions, and suggestions should be sent to the following email address:

cocoa (at) dima.unige.it

Comments on the online help system would be greatly appreciated. They may be sent to the above address or to davidp (at) reed.edu.

I-1.4 System Requirements

CoCoA 4.3 runs on most common platforms. Please visit the CoCoA web site to see whether it is available for the platform you wish to use. If the version you seek is absent let us know, and we will try to rectify the situation.

I-1.5 Copyright and Trademarks

The product names mentioned in this manual are trademarks or registered trademarks of their manufacturers.

I-1.6 Acknowledgments

The CoCoA project is partly supported by:

* Department of Mathematics, University of Genova * Department of Computer Science, University of Genova * Consiglio Nazionale delle Ricerche * Ministero dell'Universita‘ e della Ricerca Scientifica e Tecnologica

The author of the online help package would like to thank Tony Geramita, Lorenzo Robbiano and the CoCoA team. Thanks to Antonio Capani and Gianfranco Niesi for patiently answering my questions and for allowing me to use their "*CoCoA User's Manual, version 3.0b Draft,*" from which the online help package borrows, sometimes verbatim.

Part II

Chapter II-1

The CoCoA System

II-1.1 An Overview of the System

CoCoA is a computer algebra system for doing "*Computations in Commutative Algebra*". Since its first version CoCoA has been designed to offer maximum ease of use and flexibility to the mathematician with little or no knowledge of computers. It is able to perform simple and sophisticated operations on multivariate polynomial rings and on various data connected with them (ideals, modules, matrices, rational functions).

The system is capable of performing basic operations such as

```
  * sums, products, powers, derivatives, gcd, lcm of polynomials;
  * sums, products, powers, derivatives  of rational functions;
  * sums, products, powers of ideals;
  * sums  of modules;
  * sums, products, powers, determinants, adjoints of matrices;
  * operations between heterogeneous values, like the product
    between an ideal and a polynomial, etc.
```

Besides these, more advanced operations are available. For example:

```
  * Groebner bases of ideals and modules;
  * syzygies of ideals and modules;
  * minimal free resolutions of ideals and modules;
  * intersection and division of ideals and modules;
  * inclusion and equality test for ideals and modules;
  * elimination of indeterminates;
  * homogenization of ideals;
  * Poincare series and Hilbert functions;
  * factorization of polynomials;
  * saturation of toric ideals.
```

Every computation is performed within a "*current*" or "*active*" ring, but the user can easily define and switch between many rings in a single CoCoA session.

CoCoA includes an extensive Pascal-like programming language, CoCoAL, that allows the user to customize the system and extend the embedded library.

II-1.2 System Structure

CoCoA is written in C language and is available on several platforms. The core of the system is an implementation of Buchberger's algorithm for computing Groebner bases of ideals and modules over a polynomial ring whose coefficient ring is a field, and a variation of it for computing syzygies. The algorithm has been optimized in several directions and is used as a building block for many operations. Most users can, however, completely ignore the theory of Groebner bases and even their existence: CoCoA will do all the necessary "*Groebner stuff*" in the background. However, for optimum use of the system some knowledge of the theory is helpful.

The system interacts with the external world using a Low Level Protocol (LLP) that is independent of any machine architecture. High level inputs from the user are translated into LLP-requests to the CoCoA kernel. The LLP-answers are then translated back into high level, user-readable outputs.

II-1.3 Contributions

Important parts of the system have been developed by:

```
John Abbott: factorization and linear algebra
Anna Bigatti: Hilbert-Poincare series, toric ideals
Massimo Caboara: advanced ideal and module operations
Martin Kreuzer: testing and debugging
David Perkinson: online help
Volker Augustin: graphical user interface
Arndt Wills: graphical help system
```

II-1.4 CoCoA and Macaulay

Most of the standard scripts from the computer algebra system Macaulay (Classic) have been translated into CoCoAL. For information, see

“`http://www.matha.mathematik.uni-dortmund.de/~kreuzer/projects.html`”

II-1.5 Pointers to the Literature

The following are articles which may be of interest to CoCoA users. Many of the algorithms discussed in these articles have been implemented in CoCoA.

J. Abbott, “*Univariate factorization over the integers,*” Preprint (1998).

A.M. Bigatti, “*Computations of Hilbert-Poincare Series,*” J. Pure Appl. Algebra, 119/3, 237–253 (1997).

A.M. Bigatti, A. Capani, G. Niesi, L. Robbiano, “*Hilbert-Poincare Series and Elimination Problems,*” Preprint (1998).

A.M. Bigatti, R. La Scala, L. Robbiano, “*Computing Toric Ideals,*” Preprint (1998).

A.M. Bigatti, L. Robbiano, “*Borel Sets and Sectional Matrices,*” Annals of Combinatorics, 1, 197–213, (1997)

M. Caboara, P. Conti, and C. Traverso, “*Yet Another Ideal Decomposition Algorithm,*” AAECC-12, Springer LNCS 1255, 39–54, (1997).

M. Caboara, G. De Dominicis, L. Robbiano, “*Multigraded Hilbert Functions and Buchberger Algorithm,*” In Proc. ISSAC‘96, 72–78 (1996), Y.N. Lakshman, editor, New York. ACM Press.

A. Capani, G. De Dominicis, “*Web Algebra,*” In Proc. of WebNet 96. Association for the Advancement of Computing in Education (AACE) Charlottesville, USA, (1996).

A. Capani, G. De Dominicis, G. Niesi, L. Robbiano, “*Computing Minimal Finite Free Resolutions,*” J. Pure Appl. Algebra, 117/118, 105–117, (1997).

A. Capani, G. Niesi, “*The CoCoA 3 Framework for a Family of Buchberger-like Algorithms,*” In Groebner Bases and Applications (Proc. of the Conf. 33 Years of Groebner Bases) , London Math. Soc. Lecture Notes Series, Vol. 251, B. Buchberger and F. Winkler eds., Cambridge University Press, p. 338–350, (1998).

A. Capani, G. Niesi, “*CoCoA 3.0 User's Manual,*” (1995).

A. Capani, G. Niesi, L. Robbiano, “*Some Features of CoCoA 3,*” Comput. Sci. J. of Moldova 4, 296–314, (1996).

A. Giovini, T. Mora, G. Niesi, L. Robbiano, C. Traverso, “*‘One sugar cube, please’ or selection strategies in the Buchberger algorithm,*” In Proc. ISSAC‘91, 49–54, Stephen M. Watt, editor, New York, ACM Press, (1991).

A. Giovini and G. Niesi, “*CoCoA: a user-friendly system for commutative algebra,*” In Design and Implementation of Symbolic Computation Systems – International Symposium DISCO‘90, Lecture Notes in Comput. Sci., 429, 20–29, Berlin, Springer Verlag, (1990).

B. Sturmfels, “*Groebner Bases and Convex Polytopes*”, AMS University Lecture Series, Vol. 8 (1995).

SOME BOOKS AND ARTICLES MENTIONING CoCoA: W. W. Adams, P. Loustaunau, “*An Introduction to Groebner Bases,*” Graduate Studies in Mathematics, AMS, Providence, R.I. (1994).

D. Cox, J. Little, D. O'shea, "*Ideals, Varieties, and Algorithms,*" Springer-Verlag, New York (1992).

M. Kreuzer, L. Robbiano, "*Computational Commutative Algebra 1*" Springer-Verlag, (2000).

L. Robbiano, "*Groebner Bases and Statistics,*" in "*Groebner Bases and Applications,*" (Proceedings of the Conference: 33 Years of Groebner Bases), LMS Lecture Note Series, Vol. 251, B. Buchberger and F. Winkler eds., Cambridge University Press, p. 79–204 (1998).

Chapter II-2

Tutorial

II-2.1 A Tutorial Introduction to CoCoA

This is an introduction to CoCoA, mainly through examples. It is just a small part of the online CoCoA manual.

If you are using an HTML version of the manual just keep clicking! The following instructions are for those who call the manual from a CoCoA shell.

To learn more about finding information in the manual, enter "`?`" (without the quotes). For now, the only essential command to know is "`H.Browse()`". Entering "`H.Browse();`" or "`H.Browse(1);`" will display the next section of the manual. "`H.Browse(0);`" will redisplay the current section, and "`H.Browse(-1);`" will display the previous section. Start browsing the tutorial now by entering "`H.Browse();`".

II-2.2 Setting Up CoCoA for the Tutorial

If you are using an HTML version of the manual skip this section.

Some of the information from the online manual (and results of CoCoA calculations, in general) may scroll off of the screen. It is best to run CoCoA from a system that has scrollable windows with enough room to hold the output from each CoCoA command.

Under X-windows, you might try running CoCoA from an xterm started with the command "`xterm -sb -sl 512`" (scroll bar enabled, saving 512 lines). In addition, you may want to increase the vertical size of your window, e.g., "`xterm -sb -sl 512 -geometry 80x40`" under X-windows. Better yet: run CoCoA from a shell within Emacs.

IF YOU DO NOT HAVE SCROLLABLE WINDOWS: enter the command "`H.SetMore();`" to receive output 20 lines at a time. As long as there is output waiting to be printed, you will be prompted to enter "`More();`" to get the next 20 lines. You may type "`H.SetMore(N)`" in order to get N lines at a time instead of 20.

Enter "`H.Browse();`" to continue. (The "`Browse`" command will be assumed from now on.)

II-2.3 Entering Commands

While reading the tutorial it is highly recommended that you have a copy of CoCoA running in a separate window in order to play with the examples presented.

If you are using the CoCoA GUI (Graphical User Interface) then you should follow the dedicated section "*User Interface Guide*".

To execute a CoCoA command, type it into the window, ending it with a semicolon, then press the "*return*" key or "*enter*" key (depending on your system). A command can run over several lines; CoCoA will wait for a semicolon before processing the command. Also, several commands may be written on a single line (each ending with a semicolon).

See the next section for examples.

IMPORTANT NOTES for Macintosh OS 9 (up to version 4.0):

1. Only the "*enter*" key (on the far right of the keyboard is the one to use) will function to enter CoCoA commands. Experiment.

2. To enter multiple lines, one needs to highlight the lines using the mouse before hitting the "*enter*" key. Otherwise, the enter key just feeds the line containing the cursor to CoCoA.

II-2.4 Examples of Entering Commands

Here are some examples of entering simple commands in CoCoA. (Macintosh and Windows users please remember the special instructions made in the previous section.)

example

```
-------------------------------
1+1;  -- a simple command
2
-------------------------------
1  +   -- the same command spread over several lines

1;
2
-------------------------------
1+1; 2+2; 3+3;  -- several commands on a single line
2
-------------------------------
4
-------------------------------
6
-------------------------------
```

NOTE: The output above appears just as it does in my CoCoA window. The examples in the online manual will often be annotated. When CoCoA encounters *double dashes*, as above, it regards the rest of the line as a comment.

II-2.5 More on Entering Commands

One may save a sequence of commands in a file and read them into a CoCoA session with the "`Source`" command (or equivalently, with "`<<`"). For instance, if a sequence of CoCoA commands is saved in the file "`MySession`", one may enter

```
<< MySession;
```

to run the commands (give the full pathname, or store your file in the cocoa directory). User-defined functions are often stored in files and sourced in this way.

II-2.6 After the Tutorial

Browsing through the examples in this tutorial may enable you to solve your particular problems. If not, or to learn more about CoCoA in general, enter "`?`" to learn more about CoCoA's online help. For example, you will see that "`H.Command("")`" gives a long annotated list of CoCoA commands, and "`?keyword`" will look for information about "*keyword*" in the manual.

The GUI Help System (html) and the printable/browsable version (pdf) contain exactly the same information as the online help. You may choose your preferred format or exploit the different searching methods suiting best your need.

With the following section, the tutorial proper begins. To see a table of contents for the tutorial, enter "`H.Toc(1,2)`". The command "`H.Browse()`" will then continue with the tutorial, or you may give the title of a tutorial section to "`?`" to skip to that section.

II-2.7 Arithmetic

Here are some first examples; they illustrate CoCoA evaluating arithmetic expressions.

example
```
(2+3)(1+1);  -- multiplication, as usual
10
-------------------------------
2^10;
1024
-------------------------------
2+2/3;
8/3
-------------------------------
1.5+2.3;  -- decimals are converted to fractions
19/5
-------------------------------
Mod(27,5);
2
-------------------------------
2*3;
6
-------------------------------
Fact(4);
24
-------------------------------
```

For multiplication, one may use “*”, parentheses, or just a space.

II-2.8 Variables

Results from CoCoA calculations can be stored in variables. Variables, like CoCoA functions, must begin with a capital letter or an error will result.

example
```
A := 3;
2A;
6
-------------------------------
b := 7;
ERROR: parse error in line 12 of device stdin
-------------------------------
7
-------------------------------
B := 7;
A^2 + B;
16
-------------------------------
```

II-2.9 The Variable “*It*”

When CoCoA evaluates an expression, the result is usually assigned to the special CoCoA variable named “`It`”.

example
```
1+1;
2
-------------------------------
It;
```

```
2
-------------------------------
It+1;
3
-------------------------------
It;
3
-------------------------------
X := 17;  -- "It" is not changed if there is no output
It;
3
-------------------------------
X+It;
20
-------------------------------
```

II-2.10 Making Lists

The following example illustrates the use of lists in CoCoA.

example

```
L := [2,3,"a string",[5,7],3,3];  -- L is now a list
L[3];  -- here is the 3rd component of L
a string
-------------------------------
L[4];  -- the 4th component of L is a list, itself
[5, 7]
-------------------------------
L[4][2];  -- the 2nd component of the 4th component of L
7
-------------------------------
L[4,2];  -- same as above
7
-------------------------------
Append(L,"new");
L;
[2, 3, "a string", [5, 7], 3, 3, "new"]
-------------------------------
-- insert 8 as the 4th component of L, shifting the other
-- entries to the right:
Insert(L,4,8);
L;
[2, 3, "a string", 8, [5, 7], 3, 3, "new"]
-------------------------------
Remove(L,4);  -- remove it again
L;
[2, 3, "a string", [5, 7], 3, 3, "new"]
-------------------------------
Len(L);  -- the number of components of L
7
-------------------------------
Set(L);  -- same as L but with repeats removed
[2, 3, "a string", [5, 7], "new"]
-------------------------------
1..5;  -- a range of values
[1, 2, 3, 4, 5]
-------------------------------
```

```
[ X^2 | X In 1..5];  -- a useful way to make lists
[1, 4, 9, 16, 25]
-------------------------------
[1,2] >< [3,4] >< [5];  -- Cartesian product: use a greater-than
                        -- sign ">" and a less-than sign "<" to make
                        -- the operator "><".
[[1, 3, 5], [1, 4, 5], [2, 3, 5], [2, 4, 5]]
-------------------------------
```

II-2.11 Setting Up a Ring

A CoCoA session automatically starts with the default ring, R = Q[x,y,z]. The command "`Use`" is used to change rings. The following example shows how to create the ring Z/(5)[a,b,c] (the coefficient ring is the integers mod 5). Once the ring has been declared, one may start to play with polynomials, ideals, modules, and other constructions in that ring. In the ring declaration, the indeterminates are optionally separated by commas, and note the use of two colons.

Some details on handling several rings is provided below in the section of the Tutorial entitled "Using More Than One Ring" (II-2.14 pg.29) and "Ring Mapping Example" (II-2.22 pg.35).

example

```
Use S ::= Z/(5)[a,b,c];
F := a-b;
I := Ideal(F^2,c);
I;
Ideal(a^2 - 2ab + b^2, c)
-------------------------------
J := Ideal(a-b);
I + J;
Ideal(a^2 - 2ab + b^2, c, a - b)
-------------------------------
Minimalized(It);  -- find a minimal set of generators for I+J
Ideal(a - b, c)
-------------------------------
```

II-2.12 A Groebner Basis Example

A Groebner basis of an ideal I is calculated with the command "`GBasis(I)`", as illustrated in the following example.

Let r be a root of the equation $x^7 - x - 1$ over the rationals. The minimal polynomial of $(4r-1)/r^3$ can be found by computing the reduced Groebner basis of the ideal $(x^7 - x - 1, x^3y - 4x + 1)$ with respect to the lexicographic term-ordering with $x > y$.

example

```
Use R ::= Q[x,y], Lex;
Set Indentation;  -- to improve the appearance of the output
G := GBasis(Ideal(x^7-x-1,x^3y-4x+1));
G;

[ 1602818757152090759440/34524608236181199361x - 4457540/5875764481y^7
  - 47746460716124220/34524608236181199361y^6 +
  890175715271333840/34524608236181199361y^5 -
  3541992534667352220/34524608236181199361y^4 -
  55943894513139464160/34524608236181199361y^3 -
  56473654361333280980/34524608236181199361y^2 -
  27971979712025453040/34524608236181199361y -
  400704689288022689860/34524608236181199361, 1/16384y^7 - 5/16384y^6
  + 147/16384y^4 + 5/128y^3 - 31/16384y^2 + 17/128y - 20479/16384]
```

```
-------------------------------
Len(G);
2
-------------------------------
F := 16384*G[2];  -- clear denominators
F;
y^7 - 5y^6 + 147y^4 + 640y^3 - 31y^2 + 2176y - 20479
-------------------------------
```

The Groebner basis is reported as a list with two elements. The second gives a univariate polynomial which is the minimal polynomial for r.

Note that the statement declaring the ring includes the modifier, "`Lex`". Without this modifier, the default term-ordering, DegRevLex, is used. The command "`Set Indentation`" forces each polynomial of the Groebner basis to be printed on a new line.

II-2.13 Eliminating Variables

The Cartesian equations of the space curve parametrized by

$t - - > (t^{31} + t^6, t^8, t^{10})$

can be found by eliminating the indeterminate t in the ideal $(t^{31} + t^6 - x, t^8 - y, t^{10} - z)$.

example

```
Use R ::= Q[t,x,y,z];
Set Indentation;
Elim(t,Ideal(t^31+t^6-x, t^8-y, t^10-z));
Ideal(
  y^5 - z^4,
  -y^4z^5 + y^4 - 2xy^2z + x^2z^2,
  -z^8 - 2xy^3 + x^2yz + z^3,
  2xy^4z^4 + yz^7 + 3x^2y^2 - 2x^3z - yz^2,
  -y^2z^6 - 1/2xz^7 + 1/2x^3y + y^2z - 3/2xz^2,
  -1/3x^2y^4z^3 - y^3z^5 - 2/3xyz^6 + 1/3x^4 + y^3 - 4/3xyz)
-------------------------------
```

With the command "`Elim`", CoCoA automatically switches to a term-ordering suitable for eliminating the variable t, then changes back to the declared term-ordering (in this case the default term-ordering, DegRevLex).

One may see the entire Groebner basis for our ideal with respect to the elimination term-ordering for t as follows:

example

```
Use R ::= Q[t,x,y,z], Elim(t);
Set Indentation;
GBasis(Ideal(t^31+t^6-x, t^8-y, t^10-z));
[
  -t^2y + z,
  y^5 - z^4,
  -t^6 - tz^3 + x,
  -tz^4 - y^2 + xz,
  -ty^2 + txz - y^4z,
  -y^4z^5 + y^4 - 2xy^2z + x^2z^2,
  -z^8 - 2xy^3 + x^2yz + z^3,
  2xy^4z^4 + yz^7 + 3x^2y^2 - 2x^3z - yz^2,
  tx^2y - tz^2 - y^2z^3 - xz^4,
  2txyz^3 - z^7 - x^2y + z^2,
  -y^2z^6 - 1/2xz^7 + 1/2x^3y + y^2z - 3/2xz^2,
  t^2x - tx^2z^2 + xy^4z^2 + yz^5 - y,
  t^2z + 2txz^3 - y^4z^3 - x^2,
  -3tx^2z^3 + 2xy^4z^3 + yz^6 + x^3 - yz,
  -1/3x^2y^4z^3 - y^3z^5 - 2/3xyz^6 + 1/3x^4 + y^3 - 4/3xyz,
  1/3tx^3 - 1/3tyz - 1/3x^2y^4 - 1/3y^3z^2 - 1/3xyz^3]
-------------------------------
```

II-2.14 Using More Than One Ring

In CoCoA, every calculation takes place in a "*current*" or "*active*" ring. Ring-dependent objects defined by the user such as polynomials, ideals, and modules are automatically labeled by the current ring. Objects that do not depend essentially on the ring, e.g., lists or matrices of integers, do not get labeled.

CoCoA automatically starts with the ring R = Q[x,y,z]. The following example illustrates setting up a ring with the construction "`::=`" and changing rings with the command "`Use`". One may temporarily change rings with the command "`Using`". The example assumes that you do not already have a ring with identifier "`S`".

example

```
Use R ::= Q[x,y];  -- declare and use a ring R
F := (x+y)^3;
F;
x^3 + 3x^2y + 3xy^2 + y^3
-------------------------------
M := [1,"test",2];
S ::= Q[x,y,z,a,b];   -- declare a ring S with indeterminates x,y,z,a,b
Use S;             -- switch to the ring S
F;  -- F is labeled by ring R
R :: x^3 + 3x^2y + 3xy^2 + y^3
-------------------------------
M;  -- this list is not labeled R since its elements are not
    -- ring dependent (e.g., "1" is considered a separate integer, not
    -- part of the ring R)
[1, "test", 2]
-------------------------------
F := Ideal(a^2+b^2);  -- change the definition of F
Use R;  -- switch back to R
F;   -- the old F no longer exists
S :: Ideal(a^2 + b^2)
-------------------------------
GBasis(F);  -- built in functions automatically recognize the ring
[S :: a^2 + b^2]
-------------------------------
```

II-2.15 Substitutions

To substitute a list of numbers or polynomials for the indeterminates (in the order specified by the definition of the ring), one may use the function "`Eval`". To substitute out of order, use the function "`Subst`".

example

```
Use R ::= Q[x,y,z];
F := x^2+y^2+z^2;
Eval(F,[1]);  -- substitute x=1
y^2 + z^2 + 1
-------------------------------
Eval(F,[1,2,3]);  -- substitute x=1, y=2, z=3
14
-------------------------------
Subst(F,y,2);  -- substitute y=2
x^2 + z^2 + 4
-------------------------------
Eval(F,[x,2,z]); -- same as above
```

```
x^2 + z^2 + 4
-------------------------------
Subst(F,[[y,y^2],[z,z^2]]);  -- substitute y^2 for y, z^2 for z
y^4 + z^4 + x^2
-------------------------------
Eval(Ideal(F),[x^2,z]); -- substitute x^2 for x, z for y
Ideal(x^4 + 2z^2)
-------------------------------
```

II-2.16 First Functions

CoCoA's gamut of functions can be easily extended with user-defined functions. Longer functions are usually cut-and-pasted from a text editor into a CoCoA session. If the functions are to be used repeatedly, they can be saved in a separate text file and read into a CoCoA session with the "`Source`" command (or "`<<`"). The usual way to define a function is with the syntax:

`Define < FunctionName >(< argument list >) < Commands > EndDefine;`

NOTE: Variables defined within a function are usually local to that function and disappear after the function returns. Normally, the only variables accessible within a function are the function's arguments and local variables. (For the exceptions, see the section of the manual entitled "Global Memory" (III-8.3 pg.64).)

example

```
Define Square(X)   -- a simple function
  Return X^2;
EndDefine;
Square(3);
9
-------------------------------
Define IsPrime(X)  -- a more complicated function
  If Type(X) <> INT Then Return Error("Expected INT") EndIf;
  I := 2;
  While I^2 <= X Do
    If Mod(X,I) = 0 Then Return False EndIf;
    I := I+1;
  EndWhile;
  Return TRUE;
EndDefine; -- end of function definition
IsPrime(4);
FALSE
-------------------------------
Define Test(A,B)  -- a function with two arguments
  Sum := A+B;
  Prod := A*B;
  PrintLn("The sum of ",A," and ",B," is ",Sum,".");
  Print("The product of ",A," and ",B," is ",Prod,".");
EndDefine;
Test(3,5);
The sum of 3 and 5 is 8.
The product of 3 and 5 is 15.
-------------------------------
```

II-2.17 More First Functions

A user-defined function can have any number of parameters of any type, even a variable number of parameters. Note that even a function with no parameters must be called with parentheses.

example

```
Define Test1()
  PrintLn("This is a function with no parameters.");
  For I := 1 To 10 Do
    Print(I^2, " ");
  EndFor;
EndDefine;
Test1();
This is a function with no parameters.
1 4 9 16 25 36 49 64 81 100
-------------------------------
Define Test2(...)  -- a variable number of parameters
  If Len(ARGV) = 0 Then -- parameters are stored in the list ARGV
    Return "Wrong number of parameters";
  Elsif Len(ARGV) = 1 Then
    Print("There is 1 parameter: ",ARGV[1]);
  Else
    Print("There are ", Len(ARGV), " parameters: ");
    Foreach P In ARGV Do
      Print(P, " ");
    EndForeach;
  EndIf;
EndDefine;
Test2(1, 2, "string",3);
There are 4 parameters: 1 2 string 3
-------------------------------
```

II-2.18 Rings Inside User-Defined Functions

As mentioned earlier, user-defined functions cannot reference (non-global) variables except those defined within the function or passed as arguments. However, functions can refer to rings via their identifiers and use them as one would outside of a function.

When a function is called, it assumes the current ring and performs operations in that ring. One may define new rings which will exist after the function returns, but one may not change the current ring with the command "`Use`". However, one may *temporarily* use a ring with the command "`Using`".

To make functions more portable, it may be useful to refer to the current ring not by its name but by using the command "`CurrentRing`".

Example I.

Test uses the existing rings, R, S, and creates a new ring T. While a (non-global) *variable* defined in a function will automatically disappear, a ring (and its name) will not.

example

```
Use R ::= Q[x,y,z];
S ::= Q[a,b];
Define Test()
  PrintLn (x+y)^2;
  PrintLn S :: (a+b)^3;
  T ::= Z/(5)[t];
  I := T :: Ideal(t^2);
  Print I;
EndDefine;
Test();
x^2 + 2xy + y^2
S :: a^3 + 3a^2b + 3ab^2 + b^3
T :: Ideal(t^2)
-------------------------------
I;  -- the variable I was local to the function
```

```
ERROR: Undefined variable I
CONTEXT: I
-------------------------------
T;  -- The function created the ring T.  (Note: T is not a variable.)
Z/(5)[t]
-------------------------------
```

Example II
The use of “`CurrentRing`” within a function.

example

```
Define Poincare2(I)
  Return Poincare(CurrentRing()/I);
EndDefine;
Use R ::= Q[x,y];
Poincare2(Ideal(x^2,y^2));
(1 + 2x + x^2)
-------------------------------
```

Example III
Creating a ring with a user-supplied name. For more information, see “`Var`”.

example

```
Define Create(Var(R));
  Var(R) ::= Q[a,b];
EndDefine;
Create("K");
K;
Q[a,b]
-------------------------------
Create("myring");
Var("myring");
Q[a,b]
-------------------------------
Use Var("myring");  -- make myring current
```

Example IV
A more complicated example, creating rings whose names are automatically generated. See “`NewId`” and “`Var`” for more information.

example

```
Define CreateRing(I)
  NewRingName := NewId();
  Var(NewRingName) ::= Q[x[1..I]],Lex;
  Return NewRingName;
EndDefine;

Use R ::= Q[x,y],DegRevLex;
Use S ::= Q[x,y,z],Lex;
N := 5;
For I := 1 To N Do
  RingName := CreateRing(I); -- RingName is a string
  Using Var(RingName) Do
    PrintLn Indets();
  EndUsing;
  -- Destroy Var(RingName); -- uncomment if you want to destroy the tmp
  -- ring
EndFor;
```

```
[x[1]]
[x[1], x[2]]
[x[1], x[2], x[3]]
[x[1], x[2], x[3], x[4]]
[x[1], x[2], x[3], x[4], x[5]]

-------------------------------

RingEnvs();
["Q", "Qt", "R", "S", "V#1", "V#3", "V#5", "V#7", "V#9", "Z"]
-------------------------------
```

II-2.19 Rational Normal Curve

In this example, we compute the ideal of the rational normal curve of degree N in P^N then compute its Poincare series for a range of values of N.

example

```
Define Minor2(M, I, J)
  Return M[1,I] M[2,J] - M[2,I] M[1,J];
EndDefine;

Define Rational_Normal_Curve_Ideal(N)
  -- first define the 2xN matrix whose 2x2 minors generate the ideal
  M := NewMat(2,N);
  For C := 0 To N-1 Do
    M[1,C+1] := x[C];
    M[2,C+1] := x[C+1];
  EndFor;
  -- then construct the generators of the ideal
  L := [];
  For C1 := 1 To N-1 Do
    For C2 := C1+1 To N Do
      P := M[1,C1] M[2,C2] - M[2,C1] M[1,C2];
      -- determinant for columns C1,C2
      Append(L,P)
    EndFor;
  EndFor;
  Return Ideal(L);
EndDefine;

For N := 3 To 5 Do
  S ::= Q[x[0..N]],Lex;
  PrintLn NewLine, "degree ", N;
  Using S Do  -- switch, temporarily, to ring S
    I := Rational_Normal_Curve_Ideal(N);
    Print("Poincare series: "), Poincare(S/I);
  EndUsing;
  PrintLn;
EndFor; -- for statement

degree 3
Poincare series: (1 + 2x[0]) / (1-x[0])^2

degree 4
Poincare series: (1 + 3x[0]) / (1-x[0])^2
```

```
degree 5
Poincare series: (1 + 4x[0]) / (1-x[0])^2

-------------------------------
```

II-2.20 Generic Minors

The following example computes the relations among the 2x2 minors of a generic 2xN matrix for a range of values of N. Note the use of indeterminates with multiple indices.

example

```
Define Minor2(M, I, J)
  Return M[1,I] M[2,J] - M[2,I] M[1,J];
EndDefine;

Define Det_SubAlgebra(N)
  M := Mat([[x[I,J] | J In 1..N] | I In 1..2]);
  Cols := (1..N) >< (1..N);
  L := [ y[C[1],C[2]] - Minor2(M, C[1], C[2]) | C In Cols And C[1]<C[2] ];
  Return Ideal(L);
EndDefine;

Define Det_SubAlgebra_Print(N)  -- calculate and print relations
  J := Det_SubAlgebra(N);
  PrintLn NewLine, "N = ", N;
  PrintLn "Sub-algebra equations:";
  PrintLn Gens(Elim(x,J))
EndDefine;

Set Indentation;
For N := 3 To 5 Do
  S ::= Z/(32003)[y[1..(N-1),2..N],x[1..2,1..N]];
  Using S Do
    Det_SubAlgebra_Print(N);
  EndUsing
EndFor;

N = 3
Sub-algebra equations:
[
  0]

N = 4
Sub-algebra equations:
[
  2y[1,4]y[2,3] - 2y[1,3]y[2,4] + 2y[1,2]y[3,4]]

N = 5
Sub-algebra equations:
[
  2y[2,5]y[3,4] - 2y[2,4]y[3,5] + 2y[2,3]y[4,5],
  2y[1,5]y[3,4] - 2y[1,4]y[3,5] + 2y[1,3]y[4,5],
  2y[1,5]y[2,4] - 2y[1,4]y[2,5] + 2y[1,2]y[4,5],
  2y[1,5]y[2,3] - 2y[1,3]y[2,5] + 2y[1,2]y[3,5],
  2y[1,4]y[2,3] - 2y[1,3]y[2,4] + 2y[1,2]y[3,4]]
```

```
-------------------------------
```

II-2.21 Leading Term (Initial) Ideals, Generic Polynomials

The following example produces the leading term (initial) ideal of the ideal generated by three "*generic*" polynomials of degree 2 with respect to the lexicographic term-ordering.

example

```
Use R ::= Z/(32003)[x[1..4]],Lex;
F := DensePoly(2);  -- sum of all power-products of degree 2
L := [ Randomized(F) | I In 1..3 ]; -- randomize coefficients
LT(Ideal(L));
Ideal(x[1]^2, x[1]x[2], x[1]x[3], x[2]^3, x[1]x[4]^2, x[2]^2x[3],
x[2]^2x[4]^2, x[2]x[3]^3, x[2]x[3]^2x[4]^2, x[2]x[3]x[4]^4,
x[2]x[4]^6, x[3]^8)
-------------------------------
```

II-2.22 Ring Mapping Example

If R is the current ring and E is an object in another ring, then the function "`Image`" may be used to map E into R by substituting polynomials from R for the indeterminates in E. (Also: see the command "`BringIn`" for a shortcut in certain cases.)

example

```
Use S ::= Q[a,b,c];
I := Ideal(a^2+b^2,ab-c^2);
Use R ::= Q[x,y];  -- the current ring is R
F := RMap(x+y,x-y,y^2); -- define a map F:S --> R sending a to x+y,
                        -- b to x-y, and c to y^2
Image(I,F); -- the image of I under F
Ideal(2x^2 + 2y^2, -y^4 + x^2 - y^2)
-------------------------------
```

II-2.23 Output to a File

The following example illustrates one way of saving CoCoA output to a file.

example

```
Use R ::= Q[t,x,y,z];
I := Ideal(t^2-x,t^5-y,t^7-z);
G := GBasis(I);
G;
[t^2 - x, -tx^2 + y, -x^3 + ty, -xy + z, -ty^2 + x^2z, txz - y^2, y^3 - tz^2]
-------------------------------
D := OpenOFile("MyFile"); -- open "MyFile" for output from CoCoA
Print G On D;  -- G is appended to the file "MyFile"
Close(D);
```

Text can be read from files using "`OpenIFile`" and "`Get`", and commands can be executed from files using "`Source`" (or "`<<`"). One may also keep a log of a CoCoA session (see "`OpenLog`")

II-2.24 Finite Point Sets: Buchberger-Moeller

CoCoA includes an implementation of the Buchberger-Moeller algorithm for efficient computations dealing with finite sets of points. These functions include:

* GBM, HGBM – intersection of ideals for zero-dimensional schemes
* IdealAndSeparatorsOfPoints – ideal and separators for affine points
* IdealAndSeparatorsOfProjectivePoints – ideal and separators for points
* IdealOfPoints – ideal of a set of affine points
* IdealOfProjectivePoints – ideal of a set of projective points
* Interpolate – interpolating polynomial
* SeparatorsOfPoints – separators for affine points
* SeparatorsOfProjectivePoints – separators for projective points

Details about these functions may be found individually in the online manual. Briefly, the functions above may be used to find the ideal of a set of points in affine or projective space along with a reduced Groebner basis and separators. Separators are polynomials that take the value 1 on one of the points and 0 on the remainder. The Buchberger-Moeller algorithm works *much* faster than the straightforward technique involving intersecting ideals for the individual points.

example

```
Use R ::= Q[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, 0]
-------------------------------
Res(R/X.Ideal);  -- the resolution of the ideal
0 --> R^10(-6) --> R^24(-5) --> R^15(-4) --> R
-------------------------------
```

II-2.25 Syzygies and Resolution Example

The following example, among other things, computes the resolution of ideals of sets of points.

example

```
Use R ::= Q[x,y,z];
X1 := [[0,0,1],[1,0,1],[2,0,1],[2,1,1]]; -- 4 points in the projective
                                          -- plane
X2 := [[0,0,1],[1,0,1],[0,1,1],[1,1,1]]; -- 4 more points
I1 := IdealOfProjectivePoints(X1);
I2 := IdealOfProjectivePoints(X2);
Hilbert(R/I1);  -- the Hilbert function of X1
H(0) = 1
H(1) = 3
H(x) = 4   for t >= 2
-------------------------------
Hilbert(R/I2) = Hilbert(R/I1);  -- The Hilbert functions for X1 and X2
                                -- are the same
TRUE
-------------------------------
Res(R/I1);                      -- but the resolutions ...
0 --> R(-3)(+)R(-4) --> R^2(-2)(+)R(-3) --> R
-------------------------------
```

```
Res(R/I2);                      -- are different.
0 --> R(-4) --> R^2(-2) --> R
-------------------------------
Describe Res(R/I1);  -- more information about the resolution for X1
Mat[
  [xy - 2yz, y^2 - yz, x^3 - 3x^2z + 2xz^2]
]
Mat[
  [y - z, x^2 - xz],
  [-x + 2z, 0],
  [0, -y]
]
-------------------------------
Syz(I1,1);  -- the first syzygy module for X1
Module([y - z, -x + 2z, 0], [x^2 - xz, 0, -y])
-------------------------------
```

II-2.26 Factoring Polynomials

CoCoA can factor a polynomial over its coefficient ring.

example

```
Use R ::= Q[x,y];
F := x^12 - 37x^11 + 608x^10 - 5852x^9 + 36642x^8 - 156786x^7 + 468752x^6
    - 984128x^5 + 1437157x^4 - 1422337x^3 + 905880x^2 - 333900x + 54000;
Factor(F);
[[x - 2, 1], [x - 4, 1], [x - 6, 1], [x - 3, 2], [x - 5, 3], [x - 1, 4]]
-------------------------------
F := (x+y)^2*(x^2y+y^2x+3);
F;
x^4y + 3x^3y^2 + 3x^2y^3 + xy^4 + 3x^2 + 6xy + 3y^2
-------------------------------
Factor(F);  -- multivariate factorization
[[x^2y + xy^2 + 3, 1], [x + y, 2]]
-------------------------------
Use Z/(37)[x];
Factor(x^6-1);
[[x - 1, 1], [x + 1, 1], [x + 10, 1], [x + 11, 1], [x - 11, 1], [x - 10, 1]]
-------------------------------
```

Part III

Chapter III-1

Introduction to CoCoA Programming

III-1.1 An Overview of CoCoA Programming

The CoCoA system includes a full-fledged high level programming language, CoCoAL, 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`" command (or "`<<`").

The most important construct in CoCoA programming is the user-defined function, created with "`Define`". 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 III-2

Language Elements

III-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 |
| <=   less than or equal to        /*   start comment     |
| >=   greater than or equal to     */   end comment       |
| ><   Cartesian product            ::   ring casting      |
| ::=  ring definition              ...  dots              |
 ----------------------------------------------------------
                 Special Character-groups
```

III-2.2 Identifiers

There are two types of identifiers or names.

* Identifiers of ring indeterminates. They must begin with lower case letters.

* Predefined or user-defined names (functions and CoCoAL variables). They must begin with upper case letters.

III-2.3 Names of Indeterminates

Each indeterminate of a polynomial ring may have one of the following forms:

* a single lower case letter; * a single lower case letter, indexed by one or more integer expressions, separated by commas, and enclosed by square brackets, e.g. x[1,3+5,2].

III-2.4 Reserved Names

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.

```
    ----------------------------------------------------
   | Alias      And      Block      Catch     Ciao     |
   | Clear      Cond     Define     Delete    Describe |
   | Destroy    Do       Elif       Else      End      |
   | EndBlock   EndCatch EndCond    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    |
    ----------------------------------------------------
          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    VECTOR |
   | Xel       ZMOD                              |
    --------------------------------------------
           Case sensitive reserved names
```

III-2.5 Comments

A comment begins with the symbol “/*” and ends with the symbol “*/”. Comments may contain any number of characters and are always treated as white space. Comments may be nested and may span several lines.

In addition, text starting with “//” or “--” up to the end of a line is also considered comment.

example

```
// This is a line comment
Print 1+1; -- a command followed by a comment
2
-------------------------------
/* example of /* nested */ comment */
```

III-2.6 Data Types

Each CoCoA object has a type. The possible types are:

```
BOOL     : A boolean.  The boolean constants are TRUE and FALSE.
DEVICE   : For input/output.
ERROR    : Objects of this type are used in error-handling.
FUNCTION : A CoCoA-defined function.
```

```
IDEAL    : An ideal.
INT      : An arbitrary precision integer.
LIST     : A list.
MAT      : A matrix.
MODULE   : A submodule of a free module.
NULL     : The null constant is Null.
POLY     : A polynomial.
RAT      : An arbitrary precision rational number.
RATFUN   : A rational function.
RECORD   : A record is a set of bindings, name --> object.
RING     : A base ring, a polynomial ring, or a quotient ring.
STRING   : A string.
TAGGED   : Used to help in printing complicated objects.
TYPE     : Returned by the function ``\verb&Type&'', for example.
VECTOR   : A vector.
ZMOD     : An integer modulo another integer, e.g. 3 % 5.
```

The types are partially ordered by inclusion of the sets that they represent, as follows:

```
MAT < LIST
VECTOR < LIST
INT < RAT < POLY < RATFUN
ZMOD < POLY < RATFUN  (if compatible).
```

In the manual, OBJECT is used to refer to an arbitrary CoCoA type. It is not a type itself.

III-2.7 Commands and Functions for Data Types

The following are commands and functions for data types:

`Cast`	type conversion
`Max, Min`	a maximum or minimum element of a sequence or list
`Shape`	extended list of types involved in an expression
`Type`	the data type of an expression
`TypeOfCoeffs`	type of the coefficients of the current ring
`Types`	lists all data types

For details look up each item by name. Online, try “`?ItemName`” or “`H.Syntax("ItemName")`”.

Chapter III-3

Operators

III-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 multiplication symbol "`*`" can always be omitted. The expression "`F(E)`" is intrinsically ambiguous; it can be the variable F multiplied by the parenthesized expression E, or the application of the function F to the argument E. CoCoA always interprets this expression in the latter way. In the former case the user must separate F from the left parenthesis with a blank or an "`*`".

The CoCoA operators are, from the highest to the lowest priority:

```
[]  .   (selection operators)
^  %
+  -    (as unary operators)
*  :  /
+  -    (as binary operators)
..
=  <>  <  <=  >  >=
IsIn
Not
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.

Furthermore there is the n-ary operator "`><`" (made by using a greater than sign "`>`" and a less than sign "`<`") for making Cartesian products of lists (see "¿¡" (VI-1.2 pg.141), "*Cartesian Product*") and the operator "`::`" for defining rings (see "New Rings" (IV-8.2 pg.95) and "`Use`").

III-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	ZMOD	POLY	RATFUN	VECTOR	IDEAL	MODULE	MAT	LIST
INT	+-*/^	+-*/	*	+-*/	+-*/	*	*	*	*	*
RAT	+-*/^	+-*/		+-*/	+-*/	*	*	*	*	*
ZMOD	*^		+-*/	+-*/	+-*/	*	*	*	*	*
POLY	+-*/^	+-*/	+-*/	+-*/	+-*/	*	*	*	*	*
RATFUN	+-*/^	+-*/	+-*/	+-*/	+-*/				*	*
VECTOR	*	*	*	*		+-				
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`" and "`Mod`" 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.

III-3.3 Relational Operators

The relational operators are:

```
=    <>    <    >    <=    >=    IsIn
```

The operator "`IsIn`" is quite flexible: see its manual for an explanation. The other relational operators can be applied to two objects of the same type. The admissible types for "`=`" and "`<>`" are:

```
NULL, BOOL, STRING, TYPE, INT, RAT, ZMOD, POLY, RATFUN,
VECTOR, IDEAL, MODULE, MAT, LIST.
```

The admissible types for the other relational operators are:

```
STRING, TYPE, INT, RAT, IDEAL, MODULE.
```

The meaning of "`<`" for string is the lexicographic comparison. For ideals and modules "`A < B`" and "`A <= B`" both mean that A is (not necessarily strictly) contained in B.

NOTE: It is sometimes hard to judge the type of an expression from the appearance of CoCoA output, leading to confusing results from the relational operators. Here is an simple example:

example

```
L:="3";
L;
3
-------------------------------
L=3;
FALSE
-------------------------------
Type(L);
STRING
-------------------------------
Type(3);
INT
-------------------------------
```

Tagged expressions are especially prone to causing confusion of this sort.

III-3.4 Boolean Operators

The boolean operators are:

```
Not  And  Or
```

see “Introduction to Booleans” (IV-1.1 pg.79).

III-3.5 Selection Operators

The selection operators are

```
[]  .
```

Let N be of type INT and let L be of type STRING, VECTOR, 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.
VECTOR	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 “Introduction to Records” (IV-5.1 pg.89) and “Records” (IV-5 pg.89)).

III-3.6 Range Operator

If M and N are of type INT, then the expression:

```
M .. N
```

returns

```
* the list [M, M+1, ... ,N] if M <= N;
* the empty list, [], otherwise.
```

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 ::= Q[x,y,z,a,b,c,d];
z..c;
[z, a, b, c]
-------------------------------
```

Chapter III-4

Evaluation and Assignment

III-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`".

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`".

If an error occurs during the evaluation of an expression, then the evaluation is interrupted and the user is notified about the error.

III-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 (see the chapter entitled "Memory Management" (III-8 pg.63)). If E is dependent upon a ring, then L is labeled with that ring. The label is listed when L is evaluated in another ring. Then command "`RingEnv(L)`" will return the label for L.

example

```
Use R ::= Q[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 = xz];
P.Degree := Deg(P.F);
P;
Record[Degree = 2, F = xz]
-------------------------------
Use S ::= Q[a,b];
I;  -- I is labeled by R since it depends on R
R :: Ideal(x, y)
-------------------------------
T;  -- T is not labeled by R
14
-------------------------------
J := R:: Ideal(x^2-y);  -- J contains an object dependent on R
J; -- since the ring S is active, J is labeled by R
R :: Ideal(x^2 - y)
-------------------------------
Use R;
J;
Ideal(x^2 - y)
-------------------------------
```

For information about interacting with rings outside of the current ring, see “Accessing Other Rings” (IV-8.11 pg.100) in the chapter entitled “Rings” (IV-8 pg.95).

To assign values to global variables, see “Introduction to Memory” (III-8.1 pg.63) or “Global Memory” (III-8.3 pg.64).

Chapter III-5

User-Defined Functions

III-5.1 Introduction to User-Defined Functions

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`" (or "`<<`"). 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,*" as described in a later chapter.

To learn about user functions, look up "`Define`" (online, enter "`?define`") for the complete syntax and for examples. The "Tutorial" (II-2 pg.23) also contains several examples of functions.

III-5.2 Commands and Functions for User-Defined Functions

User-defined functions can contain just about any CoCoA command and may refer to other user-defined functions. The following are some commands that pertain particularly to functions:

`Call`	apply a function to given arguments
`Define`	define a function
`Function`	return a function
`Functions`	list the functions of a package
`NewId`	create a new identifier
`Return`	exit from a structured command
`Var`	function calls by reference, other complex referencing

For details look up each item by name. Online, try "`?ItemName`" or "`H.Syntax("ItemName")`".

Chapter III-6

Flow Control: Conditional Statements and Loops

III-6.1 Commands and Functions for Branching

The following are the CoCoA commands for constructing conditional statements:

`Cond`	conditional expression
`If`	conditional statement

For details look up each item by name. Online, try "`?ItemName`" or "`H.Syntax("ItemName")`".

III-6.2 Commands and Functions for Loops

The following are the commands and functions for loops:

`Break`	break out of a loop
`For`	loop command
`Foreach`	loop command
`Repeat`	loop command
`Return`	exit from a structured command
`While`	loop command

For details look up each item by name. Online, try "`?ItemName`" or "`H.Syntax("ItemName")`".

Chapter III-7

Input/Output

III-7.1 Introduction to IO

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`" command—be executed as CoCoA commands.

III-7.2 Standard IO

Standard IO is what takes places normally when one interacts with CoCoA via the CoCoA window. 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`" and "`PrintLn`" 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`" and "`Print On`" 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`" 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.

III-7.3 File IO

To print CoCoA output to a file, one first opens the file with "`OpenOFile`" then prints to the file using "`Print On`".

To receive verbatim input from a file, one first opens the file with "`OpenIFile`", then gets characters from the file with "`Get`". Actually, "`Get`" gets a list of ascii codes for the characters in the file. These can be converted to real characters using the function "`Ascii`".

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`” command, or equivalently “`<<`”. For instance, if the file “`MyFile.coc`” contains a list of CoCoA commands, then

```
  <<‘‘\verb&MyFile.coc&’’;
```

reads and executes the commands.

III-7.4 String IO

To print CoCoA output to a string, on may use “`OpenOString`” to “*open*” the string, then “`Print On`” to write to it. To read from a string, one may open the string for input with “`OpenIString`” then get characters from it with “`Get`”.

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 = "CoCoAL"]
-------------------------------
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`”.

III-7.5 Commands and Functions for IO

The following are commands and functions for input/output:

`Block`	group several commands into a single command
`Close`	close a device
`Format`	convert object to formatted string
`Get`	read characters from a device
`IO.SprintTrunc`	convert to a string and truncate
`Latex`	LaTeX formatting
`More`	print a string, N lines at a time
`OpenIFile, OpenOFile`	open input or output file
`OpenIString, OpenOString`	open input or output string
`OpenLog, CloseLog`	open or close a log of a CoCoA session
`Print On`	print to an output device
`Print, PrintLn`	print the value of an expression
`Source, <<`	read commands from a file or device
`Sprint`	convert to a string
`Tag`	returns the tag string of an object
`Tagged, Untagged, @`	tag or untag an object for pretty printing

For details look up each item by name. Online, try “`?ItemName`” or “`H.Syntax("ItemName")`”.

III-7.6 Tagged Printing

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.

III-7.7 Tagging an Object

If E is any CoCoA object and S a string, then the function “`Tagged(E,S)`” returns the object E tagged with the string S. The type of the returned object is “`TAGGED(S)`” (but the type of E is unchanged). The function “`Tag`” returns the tag string of an object, and the function “`Untagged`” (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
        -- where 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.

III-7.8 Printing a Tagged Object

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.

III-7.9 Describing a Tagged Object

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 ::= Q[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

-------------------------------
```

III-7.10 Another Example Using Tags

Here is one more example using tags. Note that CoCoA commands that do not have to do with printing ignore tags.

example

```
N := Tagged(4,"Dots");
N;
4
-------------------------------
Define Print_Dots(X)
  For I := 1 To X Do
    Print "."
  EndFor
EndDefine;
N;
....
-------------------------------
N+N;  -- As long as printing is not involved, N is treated as if
      -- it has no tag.  In this case, the sum of two tagged objects
      -- returns an integer, not another tagged object.
8
-------------------------------
M := Tagged(12,"Dots");
M;
............
-------------------------------
```

III-7.11 Commands and Functions for Tags

The following are commands and functions involving tags:

`Tag`	returns the tag string of an object
`Tagged, Untagged, @`	tag or untag an object for pretty printing

For details look up each item by name. Online, try “`?ItemName`” or “`H.Syntax("ItemName")`”.

Chapter III-8

Memory Management

III-8.1 Introduction to Memory

CoCoA has three types of memory: "*working*", "*global*", and "*ring-bound*" memory. Unlike previous versions of CoCoA, starting with CoCoA 3.5, variables defined during a session are by default assigned to a *working memory*, accessible from all rings (but not from user-defined functions). There are no longer variables that are local to a particular ring. However, as in previous versions of CoCoA, one may define variables in the *global memory* by using the prefix "`MEMORY`". The word "*global*" now refers to the fact that these variables are accessible not only to all rings but also to user-defined functions. A special class of global variables can be stored in what is called the *ring-bound memory*. These variables are formed with the prefix "`MEMORY.ENV.R`" where "`R`" is a ring identifier; they are "*bound*" to the ring, which means that they are automatically destroyed when their corresponding rings cease to exist. Otherwise, variables in the ring-bound memory behave exactly as all other global variables. Most users will never need the ring-bound memory.

These three types of memory are discussed separately, below.

III-8.2 Working Memory

The *working memory* consists of all variables except those defined with the prefix "`MEMORY`", e.g. "`MEMORY.X`". All variables in the working memory are accessible from all rings, but they are not accessible from within a user-define function (see examples in the next section). The function "`Memory`" displays the contents of the working memory. More information is provided by "`Describe Memory()`".

Ring-dependent variables such as those containing polynomials, ideals, or modules, are labeled by their corresponding rings. If the ring of a ring-dependent variable in the working memory is destroyed, the variable will continue to exist, but labeled by a ring automatically generated by CoCoA. Once all variables dependent on this new ring cease to exist, so does the ring.

example

```
Use R ::= Q[x,y,z];
Memory();  -- the working memory is empty
[ ]
-------------------------------
I:= Ideal(xy-z^3,x^2-yz);
X := 3;
M := Mat([[1,2],[3,4]]);
Memory();
["I", "It", "M", "X"]
-------------------------------
Describe Memory();
------------[Memory]-----------
I = Ideal(-z^3 + xy, x^2 - yz)
It = ["I", "It", "M", "X"]
M = Mat[
  [1, 2],
  [3, 4]
]
X = 3
-------------------------------
Use S ::= Z/(3)[t];  -- switch to a different ring
X := t^2+t+1;  -- the identifier X is used again
Y := 7;
Describe Memory();  -- note that I is labeled by its ring
------------[Memory]-----------
I = R :: Ideal(-z^3 + xy, x^2 - yz)
It = ["I", "It", "M", "X"]
M = Mat[
  [1, 2],
  [3, 4]
]
X = t^2 + t + 1
Y = 7
-------------------------------
GBasis(I);  -- The Groebner basis for the ideal in R can be calculated
            -- even though the current ring is S.
[R :: x^2 - yz, R :: -z^3 + xy]
-------------------------------
M^2;
Mat[
  [7, 10],
  [15, 22]
]
-------------------------------
Use R ::= Q[s,t];  -- redefine the ring R
I;  -- Note that I is labeled by a new ring, automatically produced by
    -- CoCoA.  This ring will automatically cease to exist when there
    -- are no longer variables dependent upon it, as shown below.
R#17 :: Ideal(-z^3 + xy, x^2 - yz)
-------------------------------
RingEnvs();
["Q", "Qt", "R", "R#17", "S", "Z"]
-------------------------------
I:=3; -- I is overwritten with an integer, and since it is the only
      -- variable dependent on R#17, the ring R#17 ceases to exist.
RingEnvs();  -- Since the only variable that was dependent upon the
             -- temporary ring "R#17" was overwritten, that ring is
             -- destroyed.
["Q", "Qt", "R", "S", "Z"]
-------------------------------
```

III-8.3 Global Memory

Starting with CoCoA 3.5, a "*global*" variable is one that is accessible from within a user-defined function. A global variable is formed by using the prefix "`MEMORY`". The special prefixes "`DEV`", "`ENV`", "`ERR`", and "`PKG`" are shorthand for "`MEMORY.DEV`", "`MEMORY.ENV`", etc. Any global variable prefixed by "`MEMORY.ENV.R`" where "`R`" is the identifier of a ring, becomes part of the ring-bound memory discussed in the next section. A list of the global variables which are not ring-bound is provided by the function "`GlobalMemory`".

example

```
Use R ::= Q[x,y,z];
X := 5;  -- a variable called "X" in the working memory
MEMORY.X := 7; -- a global variable
```

```
X;
5
-------------------------------
MEMORY.X;
7
-------------------------------
Memory();  -- the working memory
["It", "X"]
-------------------------------
GlobalMemory(); -- the global memory
["DEV", "ENV", "ERR", "PKG", "X"]
-------------------------------
Define Test()
  PrintLn(MEMORY.X);
  MEMORY.X := "a new value";
  PrintLn(X);
EndDefine;
-- MEMORY.X is accessible from within a function
-- X is not accessible within a function (thus we get an error)
Test();
7


-------------------------------
ERROR: Undefined variable X
CONTEXT: PrintLn(X)
-------------------------------
MEMORY.X;  -- the contents of the global memory can be changed from
           -- within a function
a new value
-------------------------------
Fields(MEMORY.ENV);  -- a list of all defined rings
["Q", "Qt", "R", "Z"]
-------------------------------
```

III-8.4 Ring-Bound Memory

A variable prefixed by "`MEMORY.ENV.R`" where "`R`" is the identifier of a ring, becomes bound to the ring R. This means that when R ceases to exist, so does the variable (unlike variables that are part of the working memory, labeled by R: see "Working Memory" (III-8.2 pg.63), above). The collection of these variables comprises the *ring-bound memory.* The variables bound to a ring R can be listed with the command "`Memory(R)`". Note that since ring-bound variables are, in particular, prefixed by "`MEMORY`", they are also part of the global memory discussed in the previous section.

Most users will never need ring-bound variables. Their main use is within functions which need to define and use rings temporarily, destroying them (along with their variables) before returning.

The prefix "`ENV`" is shorthand for "`MEMORY.ENV`".

example

```
Use R ::= Q[x,y,z];
X := Ideal(x,y);  -- a variable in the current memory
ENV.R.Y := "bound to R";  -- a variable in the memory bound to R
Use S ::= Q[a,b];
Z := 6;
Memory();  -- the working memory
["X", "Z"]
-------------------------------
Memory(R);  -- the memory bound to R
```

```
["Y"]
-------------------------------
Destroy R;
Memory();
["It", "X", "Z"]
-------------------------------
X;  -- Since X is not in the ring-bound memory of R,
    -- it is not destroyed.  It was *dependent* on the ring R,
    -- so the base ring of R has been given the new name R#5.
R#5 :: Ideal(x, y)
-------------------------------
ENV.R.Y;  -- this variable was destroyed along with R
ERROR: Unknown record field R
CONTEXT: ENV.R.Y
-------------------------------
RingEnvs();
["Q", "Qt", "R#5", "S", "Z"]
-------------------------------
```

III-8.5 Commands and Functions for Memory

The following are commands and functions for memory:

`Clear`	clear the working memory or a ring-bound memory
`Defined`	check if an expression has been defined
`Delete`	delete variables from the working memory
`Destroy`	delete rings
`GlobalMemory`	contents of global memory
`Memory`	contents of local memory or ring-bound memory
`RingEnvs`	names of all defined rings
`Size`	the amount of memory used by an object

For details look up each item by name. Online, try “`?ItemName`” or “`H.Syntax("ItemName")`”.

Chapter III-9

CoCoA Packages

III-9.1 Introduction to Packages

User-defined functions may be saved in separate files and read into a CoCoA session using the “`Source`” 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 (made using the “`Define`” command), labeled with a long prefix. A function from a package is referred to by the package prefix plus the function name. The user may type the full prefix, but the usual method is to create a short alias for the prefix. Details are provided below, starting with a short example.

III-9.2 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.cpkg`”.

example

```
Package $contrib/toypackage

Define IsOne(N)
  If N = 1 Then Return TRUE Else Return FALSE EndIf;
EndDefine;

Define Test(N)
  If $.IsOne(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

```
<<"one.cpkg";  -- read in the package
Test(1);  -- error here because the function "Test" is not defined

-------------------------------
ERROR: Unknown operator Test
CONTEXT: Test(1)
-------------------------------
```

```
$contrib/toypackage.Test(1); -- this is the name of the function
                             -- we are looking for
The number 1.
-------------------------------
Alias Toy := $contrib/toypackage;  -- use an alias to save typing
Toy.Test(3);
Not the number 1.
-------------------------------
Toy.IsOne(3);
FALSE
-------------------------------
```

Once the package is read, the user can choose a "*substitute prefix*" using the "`Alias`" command and in that way avoid conflicts between functions in various packages and save on typing.

Note one other thing: the function "`IsOne`" is used in the definition of "`Test`". In that case, it is referred to as "`$.IsOne`". Otherwise, CoCoA would look for a global function, outside of the package, called "`IsOne`". Forgetting this kind of reference is a common source of errors when constructing a package.

III-9.3 Package Essentials

A package begins with

```
Package $PackageName
```

and ends with

```
EndPackage;
```

PackageName is a string that will be used to identify the package. The dollar sign is required. 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.

In between the "`Package`" declaration and its "`EndPackage`" one may: (1) declare Aliases (see below), (2) define functions, and (3) make comments (please).

If a function F in the package appears in the definition of another function within the package, it must be referred to as "`$.F`" (or "`$PackageName.F`", or using a local alias, see below).

Typically, the user will read in the package using the "`Source`" command (or "`<<`"). After that, to save on typing, the user will choose a global alias with which to refer to the package using the syntax:

```
Alias ShortName := $PackageName;
```

where ShortName is any convenient string, hopefully not conflicting with other global aliases. (A list of the global aliases is returned by the function "`Aliases`".)

A package function, F, is then called using the name "`ShortName.F`".

III-9.4 Package Sourcing and Autoloading

As mentioned above, packages are usually saved in files and then read into a CoCoA session using the command "`Source`" (or "`<<`").

(I) Full path name, ordinary file sourcing.

```
package name: $mypackage
   file name: this/is/my/stuff.cpkg
```

Suppose the name of your package is "`$mypackage`" and is kept in the file with full pathname "`this/is/my/stuff.cpkg`". Then the package can be loaded into the session as usual with either of the commands:

```
Source("this/is/my/stuff.cpkg");
<<"this/is/my/stuff.cpkg";
```

Functions can then be called from the package using the package name, “`$mypackage`”, as a prefix (or using aliases).

(II) The standard package path, “`$`”-shortcut.

```
package name: $mypackage
   file name: packages/mypackages/stuff.cpkg (relative to cocoa directory)
```

A package is in the “*standard package path*” if its file is kept in the “*packages*” directory inside the cocoa directory. Suppose your package has name “`$mypackage`” and is kept in the file with pathname (relative to the cocoa directory) “`packages/mypackages/stuff.pkg`”. Then the package can be read by passing this pathname to “`Source`”, as above, but there are the following shortcuts:

```
Source("$mypackages/stuff");
<<"$mypackages/stuff";
<<$mypackages/stuff;   -- quotes are optional in this case
```

In other words, the prefix “`$`” is taking the place of “`packages/`” and the suffix “`.cpkg`” is (and must be) left off. Functions can then be called as in the previous case.

(III) Autoloading.

```
package name: $mypackages/stuff
   file name: packages/mypackages/stuff.cpkg (relative to cocoa directory)
```

Now suppose that the package is in the standard package path, as above, in the file with pathname (relative to the cocoa directory) “`packages/mypackages/stuff.cpkg`”. However, now assume that the name of the package is “`$mypackages/stuff`”, i.e., that it matches the name of its file (without “`packages/`” and “`.cpkg`”). Then, if any function from the package is called, say “`$mypackages/stuff.MyFunction`”, the package will automatically be loaded. Of course, one may also source the package using either method I or II, from above.

* Initialize * NOTE: As explained in the section entitled “Package Initialization” (III-9.8 pg.71), below, no matter which method is used to source a package, any function in the package named “`Initialize`” will automatically be executed when the package is loaded.

III-9.5 Global Aliases

A global alias for a package is formed by using the command “`Alias`” during a CoCoA session. (Local aliases are formed with the same command, but are declared inside a package. They are for use only within the package.) The syntax for “`Alias`” is

Alias binding, ..., binding;

where a “*binding*” has the form

```
  identifier := $PackageName
```

The function “`Aliases`” prints a list of the global aliases.

example

```
Aliases();

H      = $cocoa/help
IO     = $cocoa/io
GB     = $cocoa/gb
HP     = $cocoa/hp
HL     = $cocoa/hilop
List   = $cocoa/list
Mat    = $cocoa/mat
Latex  = $cocoa/latex
LaTeX  = $cocoa/latex
Toric  = $cocoa/toric
Coclib = $cocoa/coclib
TT     = $abc
-------------------------------
```

```
Alias  My := $my_package,
      Old := $my_package/old_version;
Aliases();

HP      = $cocoa/hp
BinRepr = $cocoa/binrepr
SpPoly  = $cocoa/sppoly
HL      = $cocoa/hilop
H       = $cocoa/help
My      = $my_package
Old     = $my_package/old_version
-------------------------------
```

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. For example, "`$cocoa/gb.Step(M)M`" is a valid statement inside a function, but not "`GB.Step(M)`".

III-9.6 Local Aliases

A local alias is an alias declared inside a package, for use only within the package. A local alias can have the same identifier as a global alias. Only local aliases are recognized within packages.

There are two uses for local aliases. First, recall that if the definition of function in a package uses another function, F, also defined in the package, then F must be referred to using the package name as a prefix or, for short, "`$.F`". In this way, "`$`", is an automatically a local alias for the package itself. One may choose another alias, say "`DD`", and write "`DD.F`", instead. A second use for a local alias is to refer to a separate package. In that way, one may refer to functions from that package inside the current package without typing out the full package name.

Keep in mind that these aliases are used only to save typing. Examples appear below.

III-9.7 More Examples of Packages

Here is a simple package for printing lists.

example

```
Package $contrib/list

Define About()
  Return "
    Author: Antonio
    Version: 1.0
    Date: 18 July 1997
  "
EndDefine;

Define PrintList(L)
  Foreach X In L Do
    PrintLn X
  EndForeach
EndDefine;

EndPackage;
```

Here is another package that takes a pair of objects and turns the pair into a list. Note the local alias used to reference the previous package.

example

```
Package $contrib/pair

```

```
Alias L := $contrib/list; -- Local alias for another package.
                          -- This alias does not affect global
                          -- aliases.
Define Make(A,B)
  Return [A,B];
EndDefine;

Define First(P)
  Return P[1];
EndDefine;

Define Second(P)
  Return P[2];
EndDefine;

Define PrintPairOfLists(P)
  PrintLn "First list:";
  L.PrintList($.First(P));  -- The local alias, L, is used here,
  PrintLn "Second list:";
  L.PrintList($.Second(P))  -- and here.  ``\verb&$&'' refers to a function
EndDefine;                        -- defined in the current package.

EndPackage;
```

USING THE PACKAGES. After reading in the packages using "`Source`" or "`<<`", one may proceed as follows to use them:

example

```
Alias P := $contrib/pair;
X := P.Make([x^2,x],[x,y,z]);
P.PrintPairOfLists(X);
First list:
x^2
x
Second list:
x
y
z
-------------------------------
```

Note: suppose a package with identifier "`$contrib/newlist`" prints lists in another format. To switch to this format in "`$contrib/pair`", one need only change the alias for L from "`$contrib/list`" to "`$contrib/newlist`".

III-9.8 Package Initialization

Packages are often stored in files and read into a CoCoA session using the "`Source`" command. When a package is loaded, any function whose name is "`Initialize`" will automatically be executed. For example, this feature may be used to print a message or initialized global variables when the package is read into a session.

(Note: following this rule, if the first time you access package PKG is to make an explicit call to the function PKG.Initialize() then the function will be called twice!)

example

```
Package $example
  Define Initialize()
    Print "CALLED INITIALIZE";
    MEMORY.MyVar := 17;
  EndDefine;
```

```
EndPackage;
CALLED INITIALIZE
-------------------------------
MEMORY.MyVar;
17
-------------------------------
```

III-9.9 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
* list functions that are meant to be seen by the user
* give the syntax for these functions. (What are the arguments? What values are returned?)
* describe what each function does
* provide examples of the use of each function.

III-9.10 Commands and Functions for Packages

The following are commands and functions for packages:

`Alias`	define aliases for package names
`Alias In`	temporarily override global aliases
`Aliases`	list of global aliases
`Functions`	list the functions of a package
`Packages`	list of loaded packages
`PkgName`	returns the name of a package

For details look up each item by name. Online, try "`?ItemName`" or "`H.Syntax("ItemName")`".

III-9.11 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.

The supported packages are:

```
algmorph.cpkg    -- K-algebra homomorphisms
conductor.cpkg   -- conductor sequence of points
ext.cpkg         -- ext modules, depth, and presentations
galois.cpkg      -- computing in a cyclic extension
intprog.cpkg     -- integer programming
invariants.cpkg  -- generators of an algebra of invariants
matrixnormalform.cpkg  -- Smith normal form of a matrix
primary.cpkg     -- primary ideals
specvar.cpkg     -- special varieties
stat.cpkg        -- statistics, design of experiment
thmproving.cpkg  -- geometrical theorem proving
typevectors.cpkg -- typevectors for ideals of points
```

All of these packages are included in /packages/contrib of the distribution of CoCoA. The packages are likely to be updated more often than CoCoA, itself, and new packages may appear; so it may be worth checking at the CoCoA distribution sites, e.g., "`http://cocoa.dima.unige.it/`".

HOW TO USE A SUPPORTED PACKAGE (1) save the package in /packages/contrib/, if necessary; (2) to get the syntax, description, and examples of the main functions and a suggested alias for the package, type “`$contrib/"package_name".Man();`” (3) to know the author and version number, type “`$contrib/"package_name".About();`”

or just XX.Man(); XX.About() where XX is a defined alias (type “`Aliases();`” to get the list)

NOTE: The packages will load automatically when one of their functions is called (see “Package Sourcing and Autoloading” (III-9.4 pg.68)) for more information.

See below for more details about specific supported packages.

III-9.12 K-Algebra Homomorphisms

Supported CoCoA Package

```
TITLE       : algmorph.cpkg
DESCRIPTION : CoCoA package for computing  K-algebra homomorphisms
              and subalgebras
AUTHOR      : A. Bigatti

LOADING INSTRUCTIONS
-- Enter
       $contrib/algmorph.Man();
   to get a complete description of the package including a suggested alias.
-- Enter
       $contrib/algmorph.About();
   to find the version number.  You may want to check the CoCoA
   homepage for the latest version.
```

III-9.13 Ext Modules

Supported CoCoA Package

```
TITLE       : ext.cpkg
DESCRIPTION : CoCoA package for computing Ext modules, depth, and presentation
AUTHOR      : A. Damiano

LOADING INSTRUCTIONS
-- Enter
       $contrib/ext.Man();
   to get a complete description of the package including a suggested alias.
-- Enter
       $contrib/ext.About();
   to find the version number.  You may want to check the CoCoA
   homepage for the latest version.
```

III-9.14 Galois Package

Supported CoCoA Package

```
TITLE       : galois.cpkg
DESCRIPTION : CoCoA package for computing in a cyclic algebraic
              extension
AUTHOR      : A. Bigatti, D.La Macchia, F.Rossi

LOADING INSTRUCTIONS
-- Enter
       $contrib/galois.Man();
   to get a complete description of the package including a suggested alias.
-- Enter
```

```
       $contrib/galois.About();
   to find the version number.  You may want to check the CoCoA
   homepage for the latest version.
```

III-9.15 Integer Programming

Supported CoCoA Package

```
TITLE       : intprog.cpkg
DESCRIPTION : CoCoA package for applying toric ideals to integer
              programming
AUTHOR      : A. Bigatti

LOADING INSTRUCTIONS
-- Enter
       $contrib/intprog.Man();
   to get a complete description of the package including a suggested alias.
-- Enter
       $contrib/intprog.About();
   to find the version number.  You may want to check the CoCoA
   homepage for the latest version.
```

III-9.16 Algebra of Invariants

Supported CoCoA Package

```
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

LOADING INSTRUCTIONS
-- Enter
       $contrib/invariants.Man();
   to get a complete description of the package including a suggested alias.
-- Enter
       $contrib/invariants.About();
   to find the version number.  You may want to check the CoCoA
   homepage for the latest version.
```

III-9.17 Primary Ideals

Supported CoCoA Package

```
TITLE       : primary.cpkg
DESCRIPTION : CoCoA package for applying toric ideals to integer
              programming
AUTHORS     : A. Bigatti, L. Robbiano

LOADING INSTRUCTIONS
-- Enter
       $contrib/primary.Man();
   to get a complete description of the package including a suggested alias.
-- Enter
       $contrib/primary.About();
   to find the version number.  You may want to check the CoCoA
   homepage for the latest version.
```

III-9.18 Special Varieties

Supported CoCoA Package

```
TITLE       : specvar.cpkg
DESCRIPTION : CoCoA package for computing the Hilbert-Poincare
              series of special varieties (Segre, Veronese, Rees).
AUTHORS     : A. Bigatti, L. Robbiano

LOADING INSTRUCTIONS
-- Enter
       $contrib/specvar.Man();
   to get a complete description of the package including a suggested alias.
-- Enter
       $contrib/specvar.About();
   to find the version number.  You may want to check the CoCoA
   homepage for the latest version.
```

III-9.19 Statistics

Supported CoCoA Package

```
TITLE       : stat.cpkg
DESCRIPTION : package for design of experiments in statistics
AUTHOR      : M. Caboara

LOADING INSTRUCTIONS
-- Enter
       $contrib/stat.Man();
   to get a complete description of the package including a suggested alias.
-- Enter
       $contrib/stat.About();
   to find the version number.  You may want to check the CoCoA
   homepage for the latest version.
```

III-9.20 Geometrical Theorem-Proving

Supported CoCoA Package

```
TITLE       : thmproving.cpkg
DESCRIPTION : CoCoA package for geometrical theorem-proving in euclidean space
AUTHOR      : L. Bazzotti, G. Dalzotto

LOADING INSTRUCTIONS
-- Enter
       $contrib/thmproving.Man();
   to get a complete description of the package including a suggested alias.
-- Enter
       $contrib/thmproving.About();
   to find the version number.  You may want to check the CoCoA
   homepage for the latest version.
```

III-9.21 Typevectors

Supported CoCoA Package

```
TITLE       : typevectors.cpkg
DESCRIPTION : CoCoA package for computing typevectors associated to
              Hilbert functions of ideals of points
AUTHOR      : E.Carlini, M.Stewart

LOADING INSTRUCTIONS
-- Enter
       $contrib/typevectors.Man();
   to get a complete description of the package including a suggested alias.
-- Enter
       $contrib/typevectors.About();
   to find the version number.  You may want to check the CoCoA
   homepage for the latest version.
```

III-9.22 Conductor

Supported CoCoA Package

```
TITLE       : conductor.cpkg
DESCRIPTION : CoCoA package for computing conductor sequence of points
AUTHOR      : L.Bazzotti

LOADING INSTRUCTIONS
-- Enter
       $contrib/conductor.Man();
   to get a complete description of the package including a suggested alias.
-- Enter
       $contrib/conductor.About();
   to find the version number.  You may want to check the CoCoA
   homepage for the latest version.
```

III-9.23 Matrix Normal Form

Supported CoCoA Package

```
TITLE       : matrixnormalform.cpkg
DESCRIPTION : CoCoA package for computing normal forms of a matrix
AUTHOR      : S.DeFrancisci

LOADING INSTRUCTIONS
-- Enter
       $contrib/matrixnormalform.Man();
   to get a complete description of the package including a suggested alias.
-- Enter
       $contrib/matrixnormalform.About();
   to find the version number.  You may want to check the CoCoA
   homepage for the latest version.
```

Part IV

Chapter IV-1

Booleans

IV-1.1 Introduction to Booleans

The two boolean constants are "`TRUE`" and "`FALSE`". They are mainly used with the commands "`If`" and "`While`", etc., inside CoCoA programs. The relational operators

```
= <> < <= > >=
```

return boolean constants (see "Relational Operators" (III-3.3 pg.48)). The boolean operators are

```
Not And Or IsIn
```

and return boolean constants and are described below.

IV-1.2 Commands and Functions for Booleans

The following are commands and functions for booleans:

`EqSet`	checks if the set of elements in two lists are equal
`IsEven, IsOdd`	test whether an integer is even or odd
`IsHomog`	test whether given polynomials are homogeneous
`IsIn`	check if one object is contained in another
`IsNumber`	checks if the argument is a number
`IsPosTo, IsToPos`	checks the module term-ordering of a ring
`IsSubset`	checks if the elements of one list are a subset of another
`IsZero`	test whether an object is zero
`Not, And, Or`	boolean operators

For details look up each item by name. Online, try "`?ItemName`" or "`H.Syntax("ItemName")`".

Chapter IV-2

Numbers

IV-2.1 Introduction to Numbers

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.

IV-2.2 Rationals

Rational numbers can be entered as fractions or as terminating decimals. CoCoA always converts a rational number into a fraction in lowest terms.

example

```
3/5;
3/5
-------------------------------
3.8;
19/5
-------------------------------
N := 4/8;
N;
1/2
-------------------------------
```

IV-2.3 Numerators and Denominators for Rational Numbers

If F is a variable holding a fraction, then “`F.Num`” and “`F.Den`” are the numerator and denominator, respectively. The functions “`Num`” and “`Den`”, respectively, return the same.

example

```
F := 3/5;
F.Num;
3
```

```
-------------------------------
F.Den;
5
-------------------------------
Den(F);
5
-------------------------------
1.75;
7/4
-------------------------------
It.Num;
7
-------------------------------
```

IV-2.4 Modular Integers

Let A and B be integers. The expression “`A%B`” has type “`ZMOD`” and represents the class of A modulo B. The integer B should be greater than 0 and less then $32767 = 2^{15} - 1$.

When a modular integer is evaluated by CoCoA, it is reduced to a canonical form “`A%B`” with $-B/2 < A \leq B/2$.

Two modular integers of the form “`A%C`” and “`B%C`” are said to be “*compatible*”, and the usual arithmetical operations are applicable.

example

```
3%7;
3 % 7
-------------------------------
4%7;
-3 % 7
-------------------------------
2%5 + 4%5;
1 % 5
-------------------------------
Type(3%11);
ZMOD
-------------------------------
3%11 = 14%11;
TRUE
-------------------------------
3%11 = 3;
FALSE
-------------------------------
```

Use the functions “`Div`” and “`Mod`” for quotients and remainders.

IV-2.5 Commands and Functions for Numbers

The following are commands and functions for numbers:

INTEGERS

`Abs`	absolute value of a number
`Bin`	binomial coefficient
`BinExp, EvalBinExp`	binomial expansion
`Div, Mod`	quotient and remainder for integers
`EvalHilbertFn`	evaluate the Hilbert function
`Fact`	factorial function
`Fraction`	returns the quotient of its arguments
`GCD, LCM`	greatest common divisor, least common multiple
`GenericPoints`	random projective points
`ILogBase`	integer part of the logarithm
`Inverse`	multiplicative inverse
`Iroot`	integer part of r-th root of an integer
`IsEven, IsOdd`	test whether an integer is even or odd
`IsNumber`	checks if the argument is a number
`IsPrime`	prime integer test
`Isqrt`	computes the (truncated) square root of an integer
`IsZero`	test whether an object is zero
`Len`	the length of an object
`Max, Min`	a maximum or minimum element of a sequence or list
`Mod2Rat`	reconstructing rationals from modular integers
`NextPrime`	find the next largest prime number
`Num, Den`	numerator, denominator
`Partitions`	partitions of an integer
`Product, Sum`	the product or sum of the elements of a list
`Rand`	random integer
`Seed`	seed for "`Rand`"
`Size`	the amount of memory used by an object

RATIONALS

`Abs`	absolute value of a number
`CFApprox, CFApproximants, ContFrac`	continued fractions
`DecimalStr`	convert rational number to decimal string
`FloatApprox`	approx. of rational number of the form $M * 10^E$
`FloatStr, MantissaAndExponent`	convert rational number to a float string
`Fraction`	returns the quotient of its arguments
`ILogBase`	integer part of the logarithm
`Inverse`	multiplicative inverse
`IsNumber`	checks if the argument is a number
`IsZero`	test whether an object is zero
`Max, Min`	a maximum or minimum element of a sequence or list
`Num, Den`	numerator, denominator
`Product, Sum`	the product or sum of the elements of a list

MODULAR INTEGERS

`Fraction`	returns the quotient of its arguments
`Inverse`	multiplicative inverse
`IsNumber`	checks if the argument is a number
`IsZero`	test whether an object is zero
`Mod2Rat`	reconstructing rationals from modular integers
`Product, Sum`	the product or sum of the elements of a list

For details look up each item by name. Online, try "`?ItemName`" or "`H.Syntax("ItemName")`".

Chapter IV-3

Strings

IV-3.1 Introduction to Strings

A string constant consists of a sequence of characters between double quotes (“`" "`”). For backward compatibility also single quotes (“`’ ’`”) are allowed, but this syntax is now deprecated.

example

```
Print "This is a string.";
This is a string.
-------------------------------
’So is this.’  -- deprecated
So is this.
-------------------------------
```

One may not, however, start a string with a single quote and end with a double quote, or vice versa.

IV-3.2 Concatenation

Strings may be concatenated using “`+`” or using the list function “`Concat`”.

example

```
L := "hello ";
M := "world";
L + M;
hello world
-------------------------------
Concat(L,M);
hello world
-------------------------------
```

IV-3.3 Substrings

If L is a string and N is an integer, then L[N] is the N-th character of L.

example

```
L := "hello world";
L[2];
e
-------------------------------
```

The operator “`IsIn`” can be used to test if one string is a substring of another.

example
```
L := "hello world";
"hello" IsIn L;     -- one may also write IsIn("hello",L)
TRUE
-------------------------------
```

IV-3.4 Quotes Within Strings

Strings are delimited using single quotes or double quotes (but not mixed). One may directly use quotes inside a string if they are not of the same type as the delimiters. To get quotes inside a string which *are* of the same type as the delimiters, the quotes must be doubled.

example
```
Print "This string ""contains"" quotes of ’various’ styles.";
This string "contains" quotes of ’various’ styles.
-------------------------------
Print ’This string also "contains" quotes of ’’various’’ styles.’; --deprecated
This string also "contains" quotes of ’various’ styles.
-------------------------------
```

Imagine the difficulties in writing this section of the online manual within a CoCoA package. ;>

IV-3.5 Commands and Functions for Strings

The following are commands and functions for strings:

`Ascii`	convert between characters and ascii code
`Comp`	the N-th component of a list
`DecimalStr`	convert rational number to decimal string
`FloatStr, MantissaAndExponent`	convert rational number to a float string
`Format`	convert object to formatted string
`IO.SprintTrunc`	convert to a string and truncate
`IsIn`	check if one object is contained in another
`Latex`	LaTeX formatting
`Max, Min`	a maximum or minimum element of a sequence or list
`More`	print a string, N lines at a time
`NewId`	create a new identifier
`OpenIString, OpenOString`	open input or output string
`Spaces`	return a string of spaces
`Sprint`	convert to a string
`Var`	function calls by reference, other complex referencing

For details look up each item by name. Online, try “`?ItemName`” or “`H.Syntax("ItemName")`”.

Chapter IV-4

Lists

IV-4.1 Introduction to Lists

A CoCoA list is a sequence of CoCoA objects between 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,...,N_s]`” is shorthand for “`L[N_1][N_2]...[N_s]`” (see the example below). Lists are often used to build structured objects of type “`MAT`”, “`VECTOR`”, “`IDEAL`”, and “`MODULE`”.

example

```
Use R ::= Q[t,x,y,z];
L := [34x+y^2,"a string",[],[TRUE,FALSE]]; -- a list
L[1];  -- the list's 1st component
y^2 + 34x
-------------------------------
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,ty^2-z^3];
I := Ideal(L);
I;
Ideal(x^2 - y, ty^2 - z^3)
-------------------------------
```

IV-4.2 Commands and Functions for Lists

CoCoA provides a variety of commands for manipulating lists. Note in particular the command “`In`” which is useful for building lists.

The following are commands and functions for lists:

`..`	range operator
`><`	Cartesian product
`Append`	append an object to an existing list
`BlockMatrix`	create a block matrix
`BringIn`	bring in objects from another ring
`Comp`	the N-th component of a list
`Concat, ConcatLists`	concatenate lists or lists of lists, respectively
`Count`	count the objects in a list
`Diff`	returns the difference between two lists
`Distrib`	the distribution of objects in a list
`EqSet`	checks if the set of elements in two lists are equal
`First`	the first N elements of a list
`Flatten`	flatten a list
`GBM, HGBM`	intersection of ideals for zero-dimensional schemes
`GenericPoints`	random projective points
`Head`	the first element of a list
`HIntersection, HIntersectionList`	intersection of ideals
`IdealAndSeparatorsOfPoints`	ideal and separators for affine points
`IdealAndSeparatorsOfProjectivePoints`	ideal and separators for points
`IdealOfPoints`	ideal of a set of affine points
`IdealOfProjectivePoints`	ideal of a set of projective points
`In`	create a list satisfying given conditions
`Insert, Remove`	insert or remove an object in a list
`Interpolate`	interpolating polynomial
`Interreduce, Interreduced`	interreduce a list of polynomials or vectors
`Intersection, IntersectionList`	intersect lists, ideals, or modules
`IsIn`	check if one object is contained in another
`IsSubset`	checks if the elements of one list are a subset of another
`Last`	the last N elements of a list
`Len`	the length of an object
`List`	convert an expression into a list
`Mat`	convert an expression into a matrix
`Max, Min`	a maximum or minimum element of a sequence or list
`Monic`	divide polynomials by their leading coefficients
`NewList`	create a new list
`NonZero`	remove zeroes from a list
`Product, Sum`	the product or sum of the elements of a list
`Reverse, Reversed`	reverse a list
`ScalarProduct`	scalar product
`SeparatorsOfPoints`	separators for affine points
`SeparatorsOfProjectivePoints`	separators for projective points
`Set`	remove duplicates from a list
`Size`	the amount of memory used by an object
`Sort, Sorted`	sort a list
`SortBy, SortedBy`	sort a list
`Submat`	submatrix
`Subsets`	returns all sublists of a list
`Syz`	syzygy modules
`Tail`	remove the first element of a list
`Toric`	saturate toric ideals
`Toric.CheckInput`	check input to "`Toric`"
`Tuples`	N-tuples
`WithoutNth`	removes the N-th component from a list

For details look up each item by name. Online, try "`?ItemName`" or "`H.Syntax("ItemName")`".

Chapter IV-5

Records

IV-5.1 Introduction to Records

A record is a data type in CoCoA representing a list of bindings of the form "*name* –> *object*".

example
```
Use R ::= Q[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.F;
x^2 + y
-------------------------------
P.Misc;
[1, 3, 4]
-------------------------------
P.Misc[2];
3
-------------------------------
P.Date := "1/1/98";
P;
Record[Date = "1/1/98", F = x^2 + y, I = Ideal(x, y^2 - z), Misc = [1, 3, 4]]
-------------------------------
P["I"];  -- equivalent to P.I
Ideal(x, y^2 - z)
-------------------------------
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.Var(F) := P.Var(F) EndIf; -- "Q.F" would not work here
EndForeach;
P := Q;
Delete Q;  -- get rid of the variable Q
```

```
P;
 Record[A = 2, B = 3, D = 7]
-------------------------------
```

IV-5.2 Commands and Functions for Records

The following are commands and functions for records:

`Comp`	the N-th component of a list
`DivAlg`	division algorithm
`Fields`	list the fields of a record
`IdealAndSeparatorsOfPoints`	ideal and separators for affine points
`IdealAndSeparatorsOfProjectivePoints`	ideal and separators for points
`Record`	create a record

For details look up each item by name. Online, try “`?ItemName`” or “`H.Syntax("ItemName")`”.

Chapter IV-6

Vectors

IV-6.1 Introduction to Vectors

An object of type VECTOR in CoCoA represents a vector of polynomials. If V is a vector and F is a polynomial, then the following are also vectors:

```
+V, -V, F*V, V*F.
```

If V and W are vectors with the same number of components, then one may add and subtract V and W componentwise:

```
V+W, V-W.
```

example

```
Use R ::= Q[x,y];
V := Vector(x+1,y,xy^2);
V;
Vector(x + 1, y, xy^2)
-------------------------------
-V;
Vector(-x - 1, -y, -xy^2)
-------------------------------
x*V;
Vector(x^2 + x, xy, x^2y^2)
-------------------------------
W := Vector(x,y,x^2y^2-y);
x*V-W;
Vector(x^2, xy - y, y)
-------------------------------
```

IV-6.2 Commands and Functions for Vectors

The following are commands and functions for vectors:

`BringIn`	bring in objects from another ring
`CoeffOfTerm`	coefficient of a term of a polynomial or vector
`Colon, :, HColon`	ideal or module quotient
`ColumnVectors`	the list of column vectors of a matrix
`Comp`	the N-th component of a list
`Comps`	list of components of a vector
`Deg`	the degree of a polynomial or vector
`DivAlg`	division algorithm
`E`	canonical vector
`FirstNonZero, FirstNonZeroPos`	the first non-zero entry in a vector
`GenRepr`	representation in terms of generators
`IsIn`	check if one object is contained in another
`IsTerm`	checks if the argument is a term
`IsZero`	test whether an object is zero
`LC`	the leading coefficient of a polynomial or vector
`Len`	the length of an object
`List`	convert an expression into a list
`LM`	the leading monomial of a polynomial or vector
`LPos`	the position of the leading power-product in a vector
`LPP`	the leading power-product of a polynomial or vector
`LT`	the leading term of an object
`Mat`	convert an expression into a matrix
`Monomials`	the list of monomials of a polynomial or vector
`NewVector`	create a new vector
`NF`	normal form
`NFsAreZero`	test if normal forms are zero
`NonZero`	remove zeroes from a list
`NR`	normal reduction
`NumComps`	the number of components of a vector
`Product, Sum`	the product or sum of the elements of a list
`ScalarProduct`	scalar product
`Size`	the amount of memory used by an object
`Support`	the list of terms of a polynomial or vector
`Vector`	create a vector

For details look up each item by name. Online, try “`?ItemName`” or “`H.Syntax("ItemName")`”.

Chapter IV-7

Matrices

IV-7.1 Introduction to Matrices

An m x n matrix is represented in CoCoA by the list of its rows

`Mat(R_1,...,R_m)`

where each "`R_i`" is of type "`LIST`" and has length n. A matrix has type "`MAT`". The (A,B)-th entry of a matrix M is given by "`M[A][B]`" or "`M[A,B]`".

example

```
Use R ::= Q[x,y,z];
M := Mat([[x,y,xy^2],[y,z^2,2+x]]);
M;
Mat[
  [x, y, xy^2],
  [y, z^2, x + 2]
]
-------------------------------
M[1][3];
xy^2
-------------------------------
M[1,3];
xy^2
-------------------------------
```

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 or rational function, with the obvious restrictions on the dimensions of the matrices involved.

example

```
Use R ::= Q[x,y];
N := Mat([[1,2],[3,4]]);
N^2;
Mat[
  [7, 10],
  [15, 22]
]
-------------------------------
x/y * N;
Mat[
  [x/y, 2x/y],
  [3x/y, 4x/y]
]
-------------------------------
N + Mat([[x,x],[y,y]]);
Mat[
  [x + 1, x + 2],
  [y + 3, y + 4]
]
-------------------------------
```

IV-7.2 Commands and Functions for Matrices

The following are commands and functions for matrices:

`Adjoint`	adjoint matrix
`BlockMatrix`	create a block matrix
`BringIn`	bring in objects from another ring
`ColumnVectors`	the list of column vectors of a matrix
`DegLexMat, DegRevLexMat, LexMat, XelMat`	matrices for std. term-orderings
`Det`	the determinant of a matrix
`HilbertBasis`	Hilbert basis for a monoid
`Identity`	the identity matrix
`Inverse`	multiplicative inverse
`IsZero`	test whether an object is zero
`Jacobian`	the Jacobian of a list of polynomials
`Len`	the length of an object
`LinKer`	find the kernel of a matrix
`LinSol`	find a solution to a linear system
`List`	convert an expression into a list
`Mat`	convert an expression into a matrix
`Minors`	list of minor determinants of a matrix
`NewMat`	create a new matrix
`Pfaffian`	the Pfaffian of a skew-symmetric matrix
`Product, Sum`	the product or sum of the elements of a list
`Size`	the amount of memory used by an object
`Submat`	submatrix
`TensorMat`	returns the tensor product of two matrices
`Toric`	saturate toric ideals
`Toric.CheckInput`	check input to “`Toric`”
`Transposed`	the transposition of a matrix

For details look up each item by name. Online, try “`?ItemName`” or “`H.Syntax("ItemName")`”.

Chapter IV-8

Rings

IV-8.1 Introduction to Rings

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;
 * rational functions;
 * 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 explains how to create a ring.

IV-8.2 New Rings

CoCoA starts with the default ring R = Q[t,x,y,z]. The command for building a new ring is either:

```
          I ::= C
```

or

```
          I ::= C[X:INDETS], M_1, ... , M_n
```

where I is the identifier of a CoCoAL variable, C: RING is an expression defining a coefficient ring (Z, Q or Z/(N)), X is an expression that defines the indeterminates, and the "`M_i`" are "*modifiers*". Each of these components is discussed in separate sections, below.

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`" or it can be accessed temporarily from within the current ring with the command "`Using`". See these two commands for more information. There are also several examples of ring-building in the tutorial.

example

```
Use R ::= Q[x,y,z];  -- define and use the ring R
S ::= Z/(103)[x,y], Lex; -- define the ring S with Lex term-ordering
CurrentRing();  -- the current ring is still R
Q[x,y,z]
-------------------------------
Use S; -- now the ring S is the current ring
Z/(103)[x,y]
-------------------------------
Using R Do X:=z^2-3 End;  -- 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 ::= Q[x[1..4],y[1..5,1..5]], Elim(y[1..5,1]), ToPos; -- a more
                                            -- complicated ring
```

IV-8.3 Coefficient Rings

As mentioned above, the coefficient ring for a CoCoA polynomial ring may be:

1. `Z: (arbitrarily large) integer numbers;`
2. `Q: (arbitrarily large) rational numbers;`
3. `Z/(N): (see ‘‘CocoaLimits’’ (\ref{CocoaLimits} pg.\pageref{CocoaLimits})) integers modulo N,`

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`”; $N \leq 32767$ 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 ::= Q[x,y];  -- coefficient ring is Q
S ::= Z/(5)[t]; -- coefficient ring is the field with 5 elements
T ::= Z[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 ::= Z/(2)[x,y,z]; CurrentRing();  -- these examples show the usage
                                        -- of "CoeffRing"
Z/(2)[x,y,z]
-------------------------------
Use S ::= CoeffRing[a,b]; CurrentRing();
Z/(2)[a,b]
-------------------------------
```

IV-8.4 Indeterminates

An “*indeterminate*” is represented by a name consisting of either a single lower case letter or a lower case letter followed by one or more indices. For example, “`x`”, “`x[1]`”, “`x[1,2,3]`” are legal (and different) indeterminates, as is “`x[2I,2I+1]`” if I is an integer variable.

When creating a ring the indeterminates are listed, optionally separated by commas: lack of separating commas is now deprecated and might be unacceptable in future versions. Indeterminates with indices are formed with the syntax: “`x[R_1,...,R_n]`”, where “`x`” stands for any lower case letter and each “`R_i`” has the form “`A..B`” for integers “`A <= B`”.

example

```
Use R ::= Q[xyz];  -- deprecated

Use R ::= Q[x[1..2,4..8],y[1..3],u,v];
Indets();
[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]
-------------------------------
```

IV-8.5 Weights Modifier

In forming a ring, one of the possible modifiers that may be added has one of the forms: (i) “`Weights(W_1,...,W_n)`” where “`W_i`” is a positive integer specifying the weight of the i-th indeterminate (the number of weights listed must be equal to the number of indeterminates) or (ii) “`Weights(M)`” where M is a matrix with as many columns as there are indeterminates. In the latter case, the i-th indeterminate has the multi-degree given by the i-th column of M. The first row of the matrix M must have all positive entries.

If the weights are not specified the default value is 1 for all indeterminates.

example

```
Use S ::= Q[a,b,c], Weights(1,2,3);
Deg(b);
2
-------------------------------
L := [1,2,3];
Use S ::= Q[a,b,c], Weights(L);
Deg(b);
2
-------------------------------
W := Mat([[1,2,3],[4,5,6]]);
Use S ::= Q[a,b,c], Weights(W);
Deg(b);
2
-------------------------------
MDeg(b);  -- the multi-degree of b
[2, 5]
-------------------------------
Deg(b^3+a^2c);
6
-------------------------------
MDeg(b^3+a^2c);
[6, 15]
-------------------------------
WeightsMatrix();
Mat[
  [1, 2, 3],
  [4, 5, 6]
]
-------------------------------
WeightsList(); -- returns the first row of the Weights Matrix
[1, 2, 3]
-------------------------------
```

IV-8.6 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. Some commands temporarily change the term-ordering in order to make a computation. The various possibilities are discussed more fully, below.

The command "`Ord`" can be used both as a modifier to set the ordering of the ring and as a way to view the matrix defining the current term-ordering.

example

```
Use R ::= Q[x,y,z], Lex;  -- lexicographic term-ordering (predefined)
Use S ::= Q[u,v], Ord([[1,1],[2,-3]]);  -- custom term-ordering
Ord(R);  -- the matrix defining the term-ordering for R
Mat[
  [1, 0, 0],
  [0, 1, 0],
  [0, 0, 1]
]
-------------------------------
Ord(S);  -- the matrix defining the term-ordering for S
Mat[
  [1, 1],
  [2, -3]
]
-------------------------------
```

IV-8.7 Predefined Term-Orderings

The predefined term-orderings are:

```
 * degree reverse lexicographic: DegRevLex  (the default ordering)
 * degree lexicographic: DegLex
 * pure lexicographic: Lex
 * pure xel: Xel
 * elimination term-ordering: Elim(X:INDETS)
```

The first two term-orderings use the weights of the indeterminates for computing the degree of a monomial. If the indeterminates are given in the order "`x_1, ..., x_n`", then "`x_1 > ... > x_n`" with respect to Lex, but "`x_1 < ... < x_n`" with respect to Xel.

In the last ordering, X specifies the variables that are to be eliminated. It may be a single indeterminate or a range of indeterminates. However, X may not be an arbitrary list of indeterminates; for that, see the command "`Elim`" (as opposed to the modifier "`Elim`" being discussed here). A range of indeterminates can be specified using the syntax "`<first-indet>..<last-indet>`". Another shortcut: if there are indexed variables of the form, say, "`x[i,j]`", then "`Elim(x)`" specifies a term-ordering for eliminating all of the "`x[i,j]`".

example

```
Use R ::= Q[x,y,z], Lex;
x+y+z;
x + y + z
-------------------------------
Use R ::= Q[x,y,z], Xel;
x+y+z;
z + y + x
-------------------------------
Use R ::= Q[t,x,y,z], Elim(t);
I := Ideal(t-x,t-y^2,t^2-xz^3);
GBasis(I);
[t - x, -y^2 + x, xz^3 - x^2]
-------------------------------
```

```
Use R ::= Q[x[1..5],y,z], Elim(x); -- term-ordering for eliminating all
                                   -- of the x[i,j]'s
Ord();
Mat[
  [1, 1, 1, 1, 1, 0, 0],
  [0, 0, 0, 0, 0, 1, 1],
  [0, 0, 0, 0, 0, 0, -1],
  [0, 0, 0, 0, -1, 0, 0],
  [0, 0, 0, -1, 0, 0, 0],
  [0, 0, -1, 0, 0, 0, 0],
  [0, -1, 0, 0, 0, 0, 0]
]
-------------------------------
```

IV-8.8 Temporary Term-Orderings

For computations which temporarily require a different term-ordering (for example, to eliminate variables or to homogenize ideals), the system automatically changes the term-ordering to a more suitable one, performs the computation, and then restores the initial term-ordering and gives its output with respect to this one. In this way the user never has to deal with temporary changes.

example

```
Use R ::= Q[t,x,y,z];
I := Ideal(t-x,y-z,t-z);
Elim(y,I);
Ideal(t - z, -x + z)
-------------------------------
```

IV-8.9 Custom Term-Orderings

For special purposes, the user can enter a custom ordering. We recall that each ordering “`>`” on the set of the terms of a polynomial ring in n variables corresponds to a (not uniquely determined) array “`(u_1,...,u_s)`” of vectors of the real vector space R^n. More precisely if $a = (a_1, ..., a_n)$ and $b = (b_1, ..., b_n)$ are the n-tuples of the exponents of two terms t and t’, then

$t > t' <=> (a.u_1, ..., a.u_s) >_{lex} (b.u_1, ..., b.u_s)$

where $>_{lex}$ is the ordering on R^s given by: $(c_1, ..., c_s) >_{lex} (d_1, ..., d_s)$ if and only if the first (leftmost) non-zero coordinate of $(c_1 - d_1, ..., c_s - d_s)$ is positive.

CoCoA accepts orderings defined by means of n x n matrices of integers. This is not a real restriction if one is interested, for instance, in finding all possible Groebner bases of a given ideal.

Moreover $>$ is a term-ordering if and only if the matrix whose rows are the vectors $(u_1, ..., u_s)$ has maximal rank and is such that the first non-zero element in each column is positive.

To compute a Groebner basis a term-ordering is needed.

example

```
-- The following CoCoA command defines S to be a polynomial ring and
-- orders the terms of S using the term-ordering corresponding to the
-- vectors (1,1,0,0),(0,-1,0,0),(0,0,1,1),(0,0,0,-1):
Use S ::= Q[x,y,z,t], Ord( Mat([[1, 1, 0, 0],
                                [0,-1, 0, 0],
                                [0, 0, 1, 1],
                                [0, 0, 0,-1]]) );
```

IV-8.10 Module Orderings

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 $[e_i | i = 1, ..., r]$ 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, $T(R) \times [1, ..., r]$.

A “*module term-ordering*” is defined as a total ordering > on T(M) such that for all “`T, T_1, T_2`” in T(R), with T not equal to 1, and for all i, j in 1,...,r,

```
(1)  T * T_1 * e_i > T_1 * e_i
(2)  T_1 * e_i > T_2 * e_j  =>  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 <=>  T_1 > T_2 in R
                           or, if  T_1 = T_2 , i < j
```

* the ordering called “`PosTo`” defined by:

```
T_1 * e_i > T_2 * e_j <=> i < j
                          or, if i = j, T_1 > T_2 in R .
```

The leading term of the vector (x, y^2) with respect to two different module term-orderings:

example

```
Use R ::= Q[x,y],ToPos;
LT(Vector(x,y^2));
Vector(0, y^2)
-------------------------------
Use R ::= Q[x,y],PosTo;
LT(Vector(x,y^2));
Vector(x, 0)
-------------------------------
```

IV-8.11 Accessing Other Rings

There are a variety of ways of interacting with a ring outside of the current ring. First of all, unlike CoCoA 3.4, starting with CoCoA 3.5, variables are usually assigned to a “*working memory*” accessible from all rings. (The only exceptions are variables prefixed by “`MEMORY`”. See the chapter entitled “Memory Management” (III-8 pg.63) for further information.) 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. If a variable contains a ring-dependent object such as a polynomial, an ideal, or a module, the variable becomes labeled by the ring in which it was defined. Built-in CoCoA functions should be smart enough to take into account the rings on which their arguments depend (if you find an exception, please send a message to “`cocoa at dima.unige.it`”).

To access rings outside of the current ring, one may of course use the command “`Use`” to change the current ring. Some other ways of interacting with outside rings:

(1) The “`::`” construction. This construction can be used to define variables or perform operations in rings outside of the current ring.

example

```
Use R ::= Q[x,y,z];
I := Ideal(x,y,z)^3;
I;
Ideal(x^3, x^2y, x^2z, xy^2, xyz, xz^2, y^3, y^2z, yz^2, z^3)
-------------------------------
Use S ::= Z/(5)[a,b];
I;  -- I is labeled by its ring, R
R :: Ideal(x^3, x^2y, x^2z, xy^2, xyz, xz^2, y^3, y^2z, yz^2, z^3)
-------------------------------
RingEnv(I);  -- the name of the ring on which I is dependent
R
```

```
-------------------------------
R:: Poincare(R/I);  -- To be sure, one may prefix any operation
                    -- on I by "R::"  although this should not
                    -- be necessary
(1 + 3a + 6a^2)
-------------------------------
R:: (x+y)^2;  -- S is still the active ring, but we can perform
              -- operations in R
R :: x^2 + 2xy + y^2
-------------------------------
J := R :: Ideal(x^2-y);  -- while S is active, one may define an
                         -- object dependent on R.  This variable
                         -- becomes part of the working memory.
J;
R :: Ideal(x^2 - y)
-------------------------------
Use R;
J;  -- the label is not used if R is active
Ideal(x^2 - y)
-------------------------------
```

(2) “`Using`”. From within the current ring one may temporarily perform commands in an another ring using the command “`Using`”. A brief example appears below. For more information, see the online help entry for “`Using`”.

example

```
Use R ::= Q[x,y];
S ::= Z/(5)[a,b]; -- the current ring is still R
Using S Do
  X := (a+b)^5;  -- assign a value to a variable in another ring
EndUsing;
X;
S :: a^5 + b^5
-------------------------------
Use S;
X;
a^5 + b^5
-------------------------------
```

(3) “`Image`”. To map objects from one ring to another, one may use the command “`Image`”. An introduction to this command appears in the following section and more details can be found in the online help entry, “`Image`”.

(4) “`QZP`”, “`ZPQ`”. The commands “`QZP`” and “`ZPQ`” can sometimes be used to quickly map a polynomial or ideal from an outside ring into the current ring. See the online help entry, “`QZP, ZPQ`”, for details.

(5) “`BringIn`”. This is the easiest function, but may be slow, to map objects from one ring to another.

IV-8.12 Ring Mappings: the Image Function

The function “`Image`” implements a ring homomorphism. Suppose S is the current ring and R is another ring. If X is an object in R, the function “`Image`” may be used to substitute polynomials in S for the indeterminates in X. An example is given below and complete details are given in the online help entry for “`Image`”.

To make substitutions within a single ring, one would usually use “`Eval`” or “`Subst`” rather than “`Image`”. To map a polynomial or ideal from an outside ring into the current ring, the functions “`QZP`” and “`ZPQ`” are sometimes useful. To map a polynomial or rational function (or a list, matrix, or vector of these) from R to S without changing indeterminates, use the function “`BringIn`”. (“BringIn” (VI-1.16 pg.150) is only applicable if the indeterminates of the object to be mapped are a subset of those in S.)

example

```
Use R ::= Q[a,b,c];
X := a+b-3c;
Use S ::= Q[x,y];
F := RMap(x^2,2,y^2);  -- syntax for defining a map: the n-th
             -- indeterminate in the domain will be mapped to
             -- the n-th element listed in RMap.
X; -- X lives in the ring R
R :: a + b - 3c
-------------------------------
Image(X,F); -- the image of E under the map F
x^2 - 3y^2 + 2
-------------------------------
Image(R:: (a+b)^2,F);
x^4 + 4x^2 + 4
-------------------------------
```

IV-8.13 Quotient Rings

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 ::= Q[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
-------------------------------
```

IV-8.14 Commands and Functions for Rings

The following are commands and functions controlling rings:

`Characteristic`	the characteristic of a ring
`Clear`	clear the working memory or a ring-bound memory
`CurrentRing`	the current ring
`Deg`	the degree of a polynomial or vector
`Delete`	delete variables from the working memory
`Destroy`	delete rings
`Dim`	the dimension of a ring or quotient object
`Hilbert`	the Hilbert function
`HilbertPoly`	the Hilbert polynomial
`HVector`	the h-vector of a ring or quotient object
`Image`	ring homomorphism
`Indet`	individual indeterminates
`IndetInd`	the index of an indeterminate
`IndetIndex`	index of an indeterminate
`IndetName`	the name of an indeterminate
`Indets`	list of current indeterminates
`IsPosTo, IsToPos`	checks the module term-ordering of a ring
`MDeg`	multi-degree of an polynomial
`Multiplicity`	the multiplicity (degree) of a ring or quotient object
`NumIndets`	number of indeterminates
`Ord`	matrix defining a term-ordering
`Poincare, HilbertSeries`	the Poincare series
`QZP, ZPQ`	change field for polynomials and ideals
`Ring`	returns the ring with a given name
`RingEnv`	name of the current ring
`RingEnvs`	names of all defined rings
`TypeOfCoeffs`	type of the coefficients of the current ring
`Use`	command for making a ring active
`Using`	perform commands in non-active ring
`WeightsList`	first row of the weights matrix
`WeightsMatrix`	matrix of generalized weights for indeterminates

For details look up each item by name. Online, try “`?ItemName`” or “`H.Syntax("ItemName")`”.

Chapter IV-9

Polynomials

IV-9.1 Introduction to Polynomials

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

```
The following are CoCoA polynomials:

Use R ::= Q[x,y,z];
F := 3xyz + xy^2;
F;
xy^2 + 3xyz
-------------------------------
Use R ::= Q[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*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 ::= Q[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)
-------------------------------
```

IV-9.2 Evaluation of Polynomials

Polynomials may be evaluated using the function "`Subst`". More generally, "`Subst`" 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`". For more general substitutions, see "Image" (VI-1.130 pg.206).

example

```
Use R ::= Q[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 ``\verb&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
-------------------------------
```

IV-9.3 Commands and Functions for Polynomials

The following are commands and functions for polynomials:

`Bin`	binomial coefficient
`BringIn`	bring in objects from another ring
`Coefficients`	list of coefficients of a polynomial or vector
`CoeffOfTerm`	coefficient of a term of a polynomial or vector
`Colon, :, HColon`	ideal or module quotient
`Deg`	the degree of a polynomial or vector
`DensePoly`	the sum of all power-products of a given degree
`Der`	the derivative of a rational function
`Discriminant`	the discriminant of a polynomial
`DivAlg`	division algorithm
`Eval`	substitute numbers or polynomials for indeterminates
`Factor`	factor a polynomial
`Fraction`	returns the quotient of its arguments
`GCD, LCM`	greatest common divisor, least common multiple
`GenRepr`	representation in terms of generators
`Homogenized`	homogenize with respect to an indeterminate
`Interpolate`	interpolating polynomial
`Inverse`	multiplicative inverse
`IsHomog`	test whether given polynomials are homogeneous
`IsIn`	check if one object is contained in another
`IsTerm`	checks if the argument is a term
`IsZero`	test whether an object is zero
`Jacobian`	the Jacobian of a list of polynomials
`LC`	the leading coefficient of a polynomial or vector
`Len`	the length of an object
`LM`	the leading monomial of a polynomial or vector
`Log`	the list of exponents of the leading term of a polynomial
`LogToTerm`	returns a monomial (power-product) with given exponents
`LPP`	the leading power-product of a polynomial or vector
`LT`	the leading term of an object
`MapDown`	convert a constant polynomial to a number
`MDeg`	multi-degree of an polynomial
`Monic`	divide polynomials by their leading coefficients
`Monomials`	the list of monomials of a polynomial or vector
`NF`	normal form
`NFsAreZero`	test if normal forms are zero
`NR`	normal reduction
`Num, Den`	numerator, denominator
`Poly`	convert an expression into a polynomial
`Product, Sum`	the product or sum of the elements of a list
`QZP, ZPQ`	change field for polynomials and ideals
`Randomize, Randomized`	randomize the coefficients of a given polynomial
`RealRootRefine`	refine a root of a univariate polynomial over Q
`RealRoots`	computes a root of a univariate polynomial over Q
`Resultant`	the resultant of two polynomials
`RootBound`	bound on roots of a polynomial over Q
`Size`	the amount of memory used by an object
`StarPrint`	print polynomial with *'s for multiplications
`Subst`	substitute values for indeterminates
`Support`	the list of terms of a polynomial or vector
`Sylvester`	the Sylvester matrix of two polynomials
`WLog`	weighted list of exponents

For details look up each item by name. Online, try "`?ItemName`" or "`H.Syntax("ItemName")`".

Chapter IV-10

Rational Functions

IV-10.1 Introduction to Rational Functions

An object of type RATFUN in CoCoA represents a rational function, i.e., a quotient of polynomials. Each rational function is represented as P/Q where P and Q are polynomials (of type POLY) and $deg(Q) > 0$. Common factors of the numerator and denominator are automatically simplified. At present, rational functions in CoCoA are only available over a field.

example

```
Use R ::= Q[x,y];
F := x/(x+y);  -- a rational function
F*(x+y);
x
-------------------------------
(x^2-y^2)/(x+y);  -- the result here is a polynomial
x - y
-------------------------------
```

The following algebraic operations on rational functions are supported:

`F^N, +F, -F, F*G, F/G, F+G, F-G,`

where F, G are rational functions and N is an integer.

For details look up each item by name. Online, try “`?ItemName`” or “`H.Syntax("ItemName")`”.

IV-10.2 Numerators and Denominators for Rational Functions

If F is a variable holding a rational function, then F.Num and F.Den are the numerator and denominator, respectively. The functions “`Num`” and “`Den`”, respectively, return the same.

example

```
F := x/(x+y);
F.Num;
x
-------------------------------
F.Den;
x+y
-------------------------------
Den(F);
x + y
-------------------------------
```

IV-10.3 Commands and Functions for Rational Functions

The following are commands and functions for rational functions:

`BringIn`	bring in objects from another ring
`Der`	the derivative of a rational function
`Fraction`	returns the quotient of its arguments
`Inverse`	multiplicative inverse
`IsZero`	test whether an object is zero
`Len`	the length of an object
`Num, Den`	numerator, denominator
`Product, Sum`	the product or sum of the elements of a list
`Size`	the amount of memory used by an object
`Subst`	substitute values for indeterminates

For details look up each item by name. Online, try “`?ItemName`” or “`H.Syntax("ItemName")`”.

Chapter IV-11

Ideals

IV-11.1 Introduction to Ideals

An object of type IDEAL in CoCoA represents an ideal. An ideal is formed using the command "`Ideal(P_1,...,P_n)`" where the "`P_i`" are generators for the ideal.

example

```
Use R ::= Q[x,y];
I := Ideal(x,y^2,2+xy^2);
```

The following algebraic operations yield ideals:

```
I^N, I+J, E*F, P:Q
```

where: I and J are ideals; N is a non-negative integer; E,F are either both ideals or one is an ideal and the other is a polynomial; and the pair (P,Q) has the form (IDEAL,POLY), (IDEAL,IDEAL), (MODULE,VECTOR), (MODULE,MODULE).

example

```
Use R ::= Q[x,y];
I := Ideal(x,y^2,2+xy^2);
I^2;
Ideal(x^2, xy^2, x^2y^2 + 2x, y^4, xy^4 + 2y^2, x^2y^4 + 4xy^2 + 4)
-------------------------------
J := Ideal(y);
I+J;
Ideal(x, y^2, xy^2 + 2, y)
-------------------------------
I*J;
Ideal(xy, y^3, xy^3 + 2y)
-------------------------------
```

IV-11.2 Commands and Functions for Ideals

The following are commands and functions for ideals:

`Colon, :, HColon`	ideal or module quotient
`Elim`	eliminate variables
`EquiIsoDec`	equidimensional isoradical decomposition
`GB.GetBettiMatrix`	returns the Betti matrix computed so far
`GB.GetNthSyz`	returns the part of the Nth syzygy module computed so far
`GB.GetNthSyzShifts`	shifts of the Nth syzygy module computed so far
`GB.GetRes`	returns the resolution computed so far
`GB.GetResLen`	returns the length of the resolution computed so far
`GB.ResReport`	status of an interactive resolution calculation
`GBasis`	calculate a Groebner basis
`GBM, HGBM`	intersection of ideals for zero-dimensional schemes
`GenRepr`	representation in terms of generators
`Gens`	list of generators of an ideal
`Gin`	generic initial ideal
`HIntersection, HIntersectionList`	intersection of ideals
`Homogenized`	homogenize with respect to an indeterminate
`Ideal`	convert an expression into an ideal
`IdealAndSeparatorsOfPoints`	ideal and separators for affine points
`IdealAndSeparatorsOfProjectivePoints`	ideal and separators for points
`IdealOfPoints`	ideal of a set of affine points
`IdealOfProjectivePoints`	ideal of a set of projective points
`Interreduce, Interreduced`	interreduce a list of polynomials or vectors
`Intersection, IntersectionList`	intersect lists, ideals, or modules
`IsIn`	check if one object is contained in another
`IsStable, IsStronglyStable, IsLexSegment`	checks if an ideal is stable (resp. strongly stable or a lex-segment)
`IsZero`	test whether an object is zero
`Len`	the length of an object
`LT`	the leading term of an object
`Max, Min`	a maximum or minimum element of a sequence or list
`MinGens`	list minimal generators
`Minimalize, Minimalized`	remove redundant generators
`MinSyzMinGens`	minimal generators of syzygies of minimal generators
`MonsInIdeal`	ideal generated by the monomials in an ideal
`NF`	normal form
`NFsAreZero`	test if normal forms are zero
`PrimaryDecomposition`	primary decomposition of an ideal
`Product, Sum`	the product or sum of the elements of a list
`QuotientBasis`	vector space basis for zero-dimensional quotient rings
`QZP, ZPQ`	change field for polynomials and ideals
`Radical`	radical of an ideal
`RadicalOfUnmixed`	radical of an unmixed ideal
`ReducedGBasis`	compute a reduced Groebner basis
`Res`	free resolution
`Saturation, HSaturation`	saturation of ideals
`Size`	the amount of memory used by an object
`Syz`	syzygy modules
`SyzMinGens`	syzygy module for a set of minimal generators
`SyzOfGens`	syzygy module for a given set of generators
`Toric`	saturate toric ideals
`Toric.CheckInput`	check input to “`Toric`”

For details look up each item by name. Online, try “`?ItemName`” or “`H.Syntax("ItemName")`”.

Chapter IV-12

Modules

IV-12.1 Introduction to Modules

An object of type MODULE in CoCoA represents a submodule of a free module. A module is represented by its generators as:

```
Module(V_1,...,V_n)
```

Each “`V_i`” has the form “`[P_1,...P_r]`” or “`Vector(P_1,...P_r)`”, where r is the rank of the free module containing the given module and each “`P_j`” is of type POLY.

As with ideals, information about a module can be accessed using the same syntax as for records.

CoCoA supports quotient modules and modules, as described in the next section. Shifts have been disabled in CoCoA 4.

example

```
Use S ::= Q[x,y];
M := Module([x,y^2,2+x^2y],[x,0,y]);  -- define the submodule of S^3
                          -- generated by (x,y^2,2+x^2y) and (x,0,y)
GBasis(M);
[Vector(x, 0, y), Vector(x, y^2, x^2y + 2)]
-------------------------------
Describe M;
Record[Type = MODULE, Value = Record[Gens = [[x, y^2, x^2y + 2], [x,
0, y]], MRC = 1, GBasis = [[x, 0, y], [x, y^2, x^2y + 2]]]]
-------------------------------
M.GBasis;
[Vector(x, 0, y), Vector(x, y^2, x^2y + 2)]
-------------------------------
M.Gens[1];
Vector(x, y^2, x^2y + 2)
-------------------------------
M.NumComps;  -- M is a submodule of a free module of rank 3
3
-------------------------------
```

IV-12.2 Quotient Modules

If M is a CoCoA module which is a submodule of the free module R^k for some ring R, then R^k/M represents a quotient module. It has type TAGGED(”Quotient”).

example

```
Use R ::= Q[x,y];
M:= Module([x-y,x^2-y^2,x^3+xy^2],[y,x^2,x^2y]);
Q := R^3/M;
```

IV-12.3 Shifts

```
===========================================
   THIS FACILITY IS DISABLED IN COCOA 4

(See the new function ‘‘\verb&GB.GetNthSyzShifts&’’;
 read below for an explanation of shifts.)
===========================================
```

One creates a shifted module in CoCoA using the “`Shifts`” modifier:

```
Module(Shifts([P_1,...,P_r]),V_1,...,V_n)
```

where the “`P_i`”’s are integers or monomials in the current ring and, as usual, the “`V_i`”’s are lists of polynomials, each with length r. This object represents the submodule, generated by “`V_1,...,V_n`”, of the free module of rank r which is the direct sum of “`R(P_1),...,R(P_r)`”. Here, “`R(P_i)`” is the ring R with shifted degrees. To explain these shifts, recall that in a CoCoA ring, the weights of the indeterminates are given by a weights matrix, say W. The (multi)weight of the i-th indeterminate is given by the i-th column of W. By default, the weights matrix is a single row of 1s. If “`P_i`” is an integer, then the homogeneous part of “`R(P_i)`” of degree d is the homogeneous part of R of degree “`d+P_i`”. If “`P_i`” is a monomial, then the homogeneous part of “`R(P_i)`” in multidegree d is the homogeneous part of R in multidegree “`d+deg_W(P_i)`”.

example

```
Use R ::= Q[x,y,z];
M := Module([x,y,z],[x^2,y,0]);
LT(M.Gens[1]);
Vector(x, 0, 0)
-------------------------------
Deg(M.Gens[1]);
1
-------------------------------
Ss := Shifts([-3,-5,-2]);
M := Module(Ss,[x,y,z],[x^2,y,0]);
M;
Module(Shifts([-3, -5, -2]), [x, y, z], [x^2, y, 0])
-------------------------------
LT(M.Gens[1]);  -- the leading term changes in the shifted module
Vector(0, y, 0)
-------------------------------
Deg(M.Gens[1]);
6
-------------------------------
Use S ::= Q[x,y,z],Weights(Mat([[1,2,3],[4,5,6]]));
M := Module(Shifts([xy, xz]), [x, y], [x,z]);
LT(M.Gens[1]);
Vector(0, y)
-------------------------------
MDeg(M.Gens[1]);  -- multidegree of y in R(xz), i.e. of (xz)y in R
[6, 15]
-------------------------------
```

IV-12.4 Commands and Functions for Modules

The following are commands and functions for modules:

`Colon, :, HColon`	ideal or module quotient
`E`	canonical vector
`Elim`	eliminate variables
`GB.GetBettiMatrix`	returns the Betti matrix computed so far
`GB.GetNthSyz`	returns the part of the Nth syzygy module computed so far
`GB.GetNthSyzShifts`	shifts of the Nth syzygy module computed so far
`GB.GetRes`	returns the resolution computed so far
`GB.GetResLen`	returns the length of the resolution computed so far
`GB.ResReport`	status of an interactive resolution calculation
`GBasis`	calculate a Groebner basis
`GenRepr`	representation in terms of generators
`Gens`	list of generators of an ideal
`Interreduce, Interreduced`	interreduce a list of polynomials or vectors
`Intersection, IntersectionList`	intersect lists, ideals, or modules
`IsIn`	check if one object is contained in another
`IsZero`	test whether an object is zero
`Len`	the length of an object
`LT`	the leading term of an object
`Max, Min`	a maximum or minimum element of a sequence or list
`MinGens`	list minimal generators
`Minimalize, Minimalized`	remove redundant generators
`MinSyzMinGens`	minimal generators of syzygies of minimal generators
`Module`	convert an expression into a module
`NF`	normal form
`NFsAreZero`	test if normal forms are zero
`Rank`	rank of a module
`ReducedGBasis`	compute a reduced Groebner basis
`Res`	free resolution
`Size`	the amount of memory used by an object
`Syz`	syzygy modules
`SyzMinGens`	syzygy module for a set of minimal generators
`SyzOfGens`	syzygy module for a given set of generators

For details look up each item by name. Online, try “`?ItemName`” or “`H.Syntax("ItemName")`”.

Chapter IV-13

Groebner Bases and Related Computations

IV-13.1 Introduction to Groebner Bases in CoCoA

The heart of the CoCoA system is a implementation of Buchberger's algorithm for computing Groebner bases for ideals and modules over polynomial rings with coefficients in a field. CoCoA's Groebner basis engine can be used to compute Groebner bases, syzygies, free resolutions, Hilbert functions and Poincare series, and to eliminate variables and find minimal sets of generators. Considerable control over the computations is provided through CoCoA's "The Interactive Groebner Framework" (IV-13.3 pg.118).

Groebner bases can be calculated over Q, but large calculations depending on Groebner bases will take much less time over finite fields. A common tactic is to work mod large primes to get an idea of behavior expected over Q.

It would eventually be nice to have descriptions within this online help system of the specific algorithms used by CoCoA. For now, see "Pointers to the Literature" (II-1.5 pg.20) for references.

IV-13.2 Commands and Functions for Groebner-Type Computations

The following are the commands and functions for computations based on Groebner bases. In addition to these, there are many commands that provide finer control over the computations (see the next section: "The Interactive Groebner Framework" (IV-13.3 pg.118)).

`Colon, :, HColon`	ideal or module quotient
`Dim`	the dimension of a ring or quotient object
`Elim`	eliminate variables
`EquiIsoDec`	equidimensional isoradical decomposition
`GB.Complete`	Complete an interactive Groebner-type calculation
`GB.GetBettiMatrix`	returns the Betti matrix computed so far
`GB.GetNthSyz`	returns the part of the Nth syzygy module computed so far
`GB.GetNthSyzShifts`	shifts of the Nth syzygy module computed so far
`GB.GetRes`	returns the resolution computed so far
`GB.GetResLen`	returns the length of the resolution computed so far
`GB.ResReport`	status of an interactive resolution calculation
`GB.StartGBasis`	start interactive Groebner basis computation
`GB.StartMinGens`	start interactive minimal generator calculation
`GB.StartMinSyzMinGens`	start interactive calc., min. syzs. of min. gens.
`GB.StartRes`	start interactive resolution computation
`GB.startSyz`	start interactive syzygy computation
`GB.startSyzMinGens`	start interactive calc. of syzygies of min. gens.
`GB.Stats`	status of an interactive Groebner-type calculation
`GB.Step, GB.Steps`	take steps in an interactive Groebner-type calculation
`GBasis`	calculate a Groebner basis
`GBM, HGBM`	intersection of ideals for zero-dimensional schemes
`GCD, LCM`	greatest common divisor, least common multiple
`Hilbert`	the Hilbert function
`HilbertPoly`	the Hilbert polynomial
`HIntersection, HIntersectionList`	intersection of ideals
`Homogenized`	homogenize with respect to an indeterminate
`HVector`	the h-vector of a ring or quotient object
`IdealAndSeparatorsOfPoints`	ideal and separators for affine points
`IdealAndSeparatorsOfProjectivePoints`	ideal and separators for points
`IdealOfPoints`	ideal of a set of affine points
`IdealOfProjectivePoints`	ideal of a set of projective points
`Intersection, IntersectionList`	intersect lists, ideals, or modules
`MinGens`	list minimal generators
`MinSyzMinGens`	minimal generators of syzygies of minimal generators
`Multiplicity`	the multiplicity (degree) of a ring or quotient object
`NF`	normal form
`NFsAreZero`	test if normal forms are zero
`Poincare, HilbertSeries`	the Poincare series
`Radical`	radical of an ideal
`RadicalOfUnmixed`	radical of an unmixed ideal
`ReducedGBasis`	compute a reduced Groebner basis
`Res`	free resolution
`Saturation, HSaturation`	saturation of ideals
`Syz`	syzygy modules
`SyzMinGens`	syzygy module for a set of minimal generators
`SyzOfGens`	syzygy module for a given set of generators

For details look up each item by name. Online, try “`?ItemName`” or “`H.Syntax("ItemName")`”.

IV-13.3 The Interactive Groebner Framework

For the following computations:

```
* Groebner bases
* minimal generators
* syzygies
* free resolutions
* elimination of variables
```

CoCoA provides the following features:

```
* step-by-step computation
* monitoring of the execution (verbose mode)
* various types of truncation (degree, resolution length, or regularity)
* customization of algorithms (through the GROEBNER panel and P-Series).
```

It works like this: instead of using one of the normal Groebner basis-type commands (listed in the previous section), start the computation with one of the commands,

```
* GB.Start_GBasis -- start interactive Groebner basis computation
* GB.Start_MinGens -- start interactive minimal generator calculation
* GB.Start_Res -- start interactive resolution computation
* GB.Start_Syz -- start interactive syzygy computation
```

After starting the computation, the following commands are available:

```
* GB.Complete -- Complete an interactive Groebner-type calculation
* GB.GetBettiMatrix -- returns the Betti matrix computed so far
* GB.GetNthSyz -- returns the part of the Nth syzygy module computed so far
* GB.GetNthSyzShifts -- shifts of the Nth syzygy module computed so far
* GB.GetRes -- returns the resolution computed so far
* GB.GetResLen -- returns the length of the resolution computed so far
* GB.ResReport -- status of an interactive resolution calculation
* GB.Stats -- status of an interactive Groebner-type calculation
* GB.Step, GB.Steps -- take steps in an interactive Groebner-type calculation
* ReducedGBasis -- compute a reduced Groebner basis
```

Almost all of these functions report more information if you set the Verbose flag in the GROEBNER panel by typing

```
Set Verbose;
```

(to unset, enter “`Unset Verbose`”). For more possibilities, see “Options in the GROEBNER Panel” (V-1.11 pg.129).

Use of the Interactive Groebner Framework is illustrated in the examples below.

IV-13.4 Example: Interactive Groebner Basis Computation

example

```
Use R ::= Q[t,x,y,z];
I := Ideal(t^3-x,t^4-y,t^5-z);
GB.Start_GBasis(I);  -- start the interactive framework
I.GBasis;  -- the Groebner basis is initially empty
Null
-------------------------------
GB.Step(I); -- a single step of the computation
I.GBasis;
I.GBasis;
[t^3 - x]
-------------------------------
GB.Steps(I,4); -- 4 more steps
I.GBasis;
[t^3 - x, -tx + y, t^2y - x^2]
-------------------------------
GB.Complete(I); -- complete the computation
I.GBasis;
[t^3 - x, -tx + y, -ty + z, -y^2 + xz, -x^2 + tz, t^2z - xy]
```

```
-------------------------------
ReducedGBasis(I);
[t^3 - x, tx - y, ty - z, y^2 - xz, x^2 - tz, t^2z - xy]
-------------------------------

Note that Groebner bases calculated in the interactive framework may
not be reduced, as illustrated in the final step of the example.
```

IV-13.5 Example: Verbose Mode

The following example illustrates the use of "`Verbose`" mode. For more information, see "Verbose" (V-1.15 pg.130).

example

```
Set Verbose;
Use R ::= Q[t,x,y,z];
I := Ideal(t^3-x,t^4-y,t^5-z);
G := GBasis(I);
..................18
------------------------------------
 IPs  IVs Gens GBases MinGens MinDeg
------------------------------------
   0    0    3      6       0     -1
------------------------------------
Betti numbers:
18 steps of computation

-------------------------------
Describe I;  -- more information, (it would help to ‘‘\verb&Set Indentation&’’)
Record[Type = IDEAL, Value = Record[Gens = [t^3 - x, t^4 - y, t^5 -
z], GBasis = [t^3 - x, -tx + y, -ty + z, -y^2 + xz, -x^2 + tz, t^2z -
xy], IVs = [ ], Rules = [t^3 - x, t^4 - y, t^5 - z, -tx + y, t^2y -
x^2, x^3 - ty^2, -ty + z, -y^2 + xz, -x^2 + tz, t^2z - xy],
Discrepancy = -1, KFL = [1, 7, 14, 4, 16, 9, 10, 20, 22, 23, 28, 18,
0, 39, 42]]]
-------------------------------
```

IV-13.6 Example: Interactive Resolution Computation

In this example we compute the minimal free resolution of the ideal I generated by the 2 by 2 minors of a catalecticant matrix, A, using the interactive environment of the system. We define the ideal I, and start the computation of its minimal free resolution using the Hilbert-driven algorithm described in

A. Capani, G. De Dominicis, G. Niesi, L. Robbiano, "*Computing Minimal Finite Free Resolutions*", J. Pure Appl. Algebra, Vol. 117–118, Pages 105–117, 1997.

example

```
Use R ::= Z/(32003)[z[0..3,0..3,0..3]]; -- set up the ring
A := Mat([                              -- define the ideal
  [z[3,0,0], z[2,1,0], z[2,0,1]],
  [z[2,1,0], z[1,2,0], z[1,1,1]],
  [z[2,0,1], z[1,1,1], z[1,0,2]],
  [z[1,2,0], z[0,3,0], z[0,2,1]],
  [z[1,1,1], z[0,2,1], z[0,1,2]],
  [z[1,0,2], z[0,1,2], z[0,0,3]]
]);
I := Ideal(Minors(2,A));
GB.Start_Res(I);                        -- start interactive framework
GB.Steps(I,1000);                      -- first 1000 steps
GB.GetRes(I);
0 --> R^176(-5) --> R^189(-4) --> R^105(-3) --> R^27(-2)
-------------------------------
GB.ResReport(I);
--------------------------------------------------------------
Minimal Pairs,              :   650
  Groebner Pairs            :    14
  Minimal (Type S)          :   636
      H-Killed (Type S0)    :     9
--------------------------------------------------------------
-------------------------------
GB.Complete(I);                       -- complete the calculation
GB.GetRes(I);
0 --> R(-9) --> R^27(-7) --> R^105(-6) --> R^189(-5) -->
   R^189(-4) --> R^105(-3) --> R^27(-2)
-------------------------------
GB.ResReport(I);
--------------------------------------------------------------
Minimal Pairs,              :   730
  Groebner Pairs            :    25
  Minimal (Type S)          :   705
    Minimal (Type Smin)     :   616
    Minimal (Type S0)       :    89
      H-Killed (Type S0)    :    78
      Hard (Type S0)        :    11
--------------------------------------------------------------
-------------------------------
```

IV-13.7 Example: Truncations

The user may assign one or more of three different truncation conditions to a module: DegTrunc, ResTrunc and RegTrunc; in this case the execution will stop when a bound is reached (see the examples, below).

DEGREE TRUNCATION:

example

```
Use R ::= Z/(32003)[a,b,c,d,e];
I := Ideal(a+b+c+d, ab+bc+cd+da, abc+bcd+cda, abcd-e^4);
I.DegTrunc := 3;
GB.Start_GBasis(I);
GB.Complete(I);
LT(I.GBasis);
[a, b^2, bc^2]
-------------------------------
I.DegTrunc := 6;
GB.Complete(I);
LT(I.GBasis);
[a, b^2, bc^2, bcd^2, c^2d^2, cd^4, be^4, d^2e^4]
-------------------------------
```

RESOLUTION TRUNCATION:

example

```
Use R ::= Z/(32003)[x[1..10]];
I := Ideal(Indets());
I.ResTrunc := 4;
GB.Start_Res(I);
GB.Complete(I);
GB.GetRes(I);
0 --> R^252(-5) --> R^210(-4) --> R^120(-3) --> R^45(-2) --> R^10(-1)
-------------------------------
```

REGULARITY TRUNCATION: We know that the Castelnuovo regularity of I in the following example is 6.

example

```
Set Verbose;
Use R_Gen ::= Z/(5)[x,y,z,t];
M := 3; N := 4;
D := DensePoly(2);
P := Mat([ [ Randomized(D) | J In 1..N ] | I In 1.. M]);
I := Ideal(Minors(2,P));
GB.Start_Res(I);
GB.Complete(I);
-- text suppressed --
Betti numbers: 17 48 48 18
318 steps of computation

I := Ideal(Minors(2,P));
GB.Start_Res(I);
I.RegTrunc := 6; -- here we store the Castelnuovo Regularity
GB.Complete(I);
...
Betti numbers: 17 48 48 18
281 steps of computation

GB.GetBettiMatrix(I);
-------------------
   0    0    0    0
   0    0    0    0
   0    0    0    0
   0    0    0   18
   0    0   16    0
   0    0   32    0
   0   48    0    0
  17    0    0    0
-------------------
```

IV-13.8 Hilbert-Driven Computations

CoCoA Groebner basis algorithms will use knowledge of a Poincare series to improve efficiency.

example

```
Use R ::= Q[w,x,y,z];
PS := HP.PSeries(1 + 2w + 3w^2 + 4w^3 + 5w^4 + 6w^5 + 5w^6 +
 4w^7 + 3w^8 + 2w^9 + w^10,2);
PS;
(1 + 2w + 3w^2 + 4w^3 + 5w^4 + 6w^5 + 5w^6 + 4w^7 + 3w^8 + 2w^9 + w^10) / (1-w)^2
-------------------------------
I := Ideal((xy-zw)^3,(xz-yw)^3);
I.PSeries := PS; -- this is how to let CoCoA know about the Poincare series
G := GBasis(I);
```

Part V

Chapter V-1

CoCoA Panels

V-1.1 Introduction to Panels

The user can customize some features of CoCoA by setting (or unsetting) some boolean options (ON/OFF) which are grouped in three panels. The function “`Panels`” returns the list of panel names, and the settings in a particular panel can be seen using the function “`Panel`” as illustrated in the example below.

example

```
Panels();
["GENERAL", "GROEBNER"]
-------------------------------
Panel(GENERAL);

Echo............... : FALSE
Timer.............. : FALSE
Trace.............. : FALSE
Indentation........ : FALSE
TraceSources....... : FALSE
SuppressWarnings... : FALSE
ComputationStack... : FALSE
-------------------------------
```

V-1.2 Setting Options

Each option name is unique, not just among the options in a particular panel, but among the options from all panels. The command “`Set`” followed by an option name (without parentheses or quotes) sets the corresponding option to TRUE. Similarly, “`Unset`” sets the option to FALSE. In addition, one may use the “`Set`” command to set an option either true or false using the syntax: “`Set option-name := TRUE`” or “`Set option-name := TRUE`”.

example

```
Use R ::= Q[x,y,z];
L := [(x+y)^N | N In 1..3];
Set Indentation;  -- print each component on a separate line
L;
L;
[
  x + y,
  x^2 + 2xy + y^2,
  x^3 + 3x^2y + 3xy^2 + y^3]
-------------------------------
Unset Indentation;
```

The function “`Option`” takes as parameter an option name and returns the (boolean) value of the option. It is particularly useful within user-defined functions as illustrated in the example below:

example
```
Define Print_UnIndented(X)
  Opt := Option(Indentation);
  Unset Indentation;
  Print(X);
  Set Indentation := Opt;
EndDefine;

R := Record[X="test",L=[1,2,3]];
Set Indentation;
Print_UnIndented(R);
Record[L = [1, 2, 3], X = "test"]
-------------------------------
UnSet Indentation;
Print_UnIndented(R);
Record[L = [1, 2, 3], X = "test"]
-------------------------------
```

V-1.3 Options in the GENERAL Panel

The options in the GENERAL panel and their default settings are as follows:

example
```
Panel(GENERAL);

Echo............... : FALSE
Timer.............. : FALSE
Trace.............. : FALSE
Indentation........ : FALSE
TraceSources....... : FALSE
SuppressWarnings... : FALSE
ComputationStack... : FALSE
-------------------------------

They are discussed separately, below.
```

V-1.4 Echo

If the Echo option is on, then the system echoes every command at the top level. When logging a CoCoA session, one would set Echo to TRUE in order to log both the input as well as the output in a CoCoA session (see “`OpenLog`”).

example
```
1+1;
2
-------------------------------
Set Echo;
1+1;
1 + 1
2
-------------------------------
L := [1,2,3];
L := [1, 2, 3]
Unset Echo;
SET(Echo, FALSE)
L := [4,5,6];
-------------------------------
```

V-1.5 Timer

If the Timer option is on, then the system displays the execution time of each command submitted at top-level.

example

```
Use R ::= Z/(32003)[t,x,y,z],Lex;
N:=31;
I := Ideal(t^N+t^6+t-x, t^5-t-y, t^9-t-z);
Set Timer;
Null
-------------------------------
T := GBasis(I);
Cpu time = 2.88, User time = 30
-------------------------------

To time a single command, use ``\verb&Time&''.  For example, above, we could
have written ``\verb&Time T := GBasis(I)&'' instead of setting the timer.
```

V-1.6 Trace

If the Trace option is on, then the system echoes every command at every level. This is useful for debugging programs. The following (toy) user-defined function returns the same error message for N = 3 and N = 6. Turning the Trace option on makes the sources of the trouble clear.

example

```
Define T(N)
  M := 1/(N-3);
  If N = 6 Then N := 3 EndIf;
  M := 1/(N-3);
  Return M;
EndDefine;
T(3);

-------------------------------
ERROR: Division by zero
CONTEXT: 1 / (N - 3)
-------------------------------
T(6);

-------------------------------
ERROR: Division by zero
CONTEXT: 1 / (N - 3)
-------------------------------
Set Trace;
T(3);
T(3)
M := 1 / (N - 3); IF N = 6 THEN N := 3 END; M := 1 / (N - 3); Return(M);
M := 1 / (N - 3)

-------------------------------
ERROR: Division by zero
CONTEXT: 1 / (N - 3)
-------------------------------
T(6);
T(6)
M := 1 / (N - 3); IF N = 6 THEN N := 3 END; M := 1 / (N - 3); Return(M);
M := 1 / (N - 3)
IF N = 6 THEN N := 3 END
```

```
N := 3
M := 1 / (N - 3)

-------------------------------
ERROR: Division by zero
CONTEXT: 1 / (N - 3)
-------------------------------
```

V-1.7 Indentation

If the Indentation option is on, then the system performs some indentation on outputs. For example, each entry of a matrix will be printed on a new line.

example

```
Use R ::= Q[xyz]
T := GBasis(Ideal(x^2-xy+z^3,xz-y^4,x^3+y^3+xyz+z^5));
T;
[z^3 + x^2 - xy, -y^4 + xz, x^2z^2 - xyz^2 - x^3 - y^3 - xyz, x^4 -
2x^3y + x^2y^2 + x^3z + y^3z + xyz^2]
-------------------------------
Set Indentation;
T;
[
  z^3 + x^2 - xy,
  -y^4 + xz,
  x^2z^2 - xyz^2 - x^3 - y^3 - xyz,
  x^4 - 2x^3y + x^2y^2 + x^3z + y^3z + xyz^2]
-------------------------------
```

V-1.8 TraceSources

If the TraceSources option is on, then the name of every file read with the “`Source`” command will be echoed.

V-1.9 SuppressWarnings

If the SuppressWarnings option is on, then the system suppress warning statements.

example

```
Use Z/(4);
-- WARNING: Coeffs are not in a field
-- GBasis-related computations could fail to terminate or be wrong
-------------------------------

-------------------------------
Set SuppressWarnings;
Use Z/(4);
```

V-1.10 ComputationStack

If the ComputationStack option is on, a special variable named “`ComputationStack`” contains a list tracing errors that occur during the execution of CoCoA commands. This option is useful for debugging programs.

example

```
Define Test(X)
  If X>=0 Then PrintLn(1/X) EndIf;
EndDefine;
```

```
Set ComputationStack;
Test(0);



-------------------------------
ERROR: Division by zero
CONTEXT: 1 / X
-------------------------------
S := ComputationStack;  -- to save typing later
S[1];  -- the command that produced the error
PrintLn(1 / X)
-------------------------------
S[2];  -- S[1] was part of an If-statement
IF X >= 0 THEN PrintLn(1 / X) END
-------------------------------
S[3];  -- the command issued by the user
IF X >= 0 THEN PrintLn(1 / X) END;
-------------------------------
```

V-1.11 Options in the GROEBNER Panel

The options in the GROEBNER panel and their default settings are as follows:

example

```
Panel(GROEBNER);

Sugar........... : TRUE
FullRed......... : TRUE
SingleStepRed... : FALSE
Verbose......... : FALSE
-------------------------------
```

V-1.12 Sugar

If the Sugar option is on, as it is by default, then the critical pairs are processed by using the “*sugar*” strategy: the pairs are processed in an order which is as close as possible to the order which would have been chosen if the polynomials had been homogeneous. For details, see the article:

A. Giovini, T. Mora, G. Niesi, L. Robbiano, C. Traverso, “*‘One sugar cube, please’ or selection strategies in the Buchberger algorithm,*” In Proc. ISSAC‘91, 49–54 (1991), Stephen M. Watt, editor, New York, ACM Press.

V-1.13 FullRed

If FullRed is set to TRUE, then when a normal form is required in any Groebner-type computation, CoCoA will reduce all monomials in a polynomial; if FullRed is FALSE, only the leading terms will be reduced. The default is to have FullRed set to TRUE.

example

```
UnSet FullRed;
Use R ::= Q[x,y];
Interreduced([xy^3+y^2+x,x]);
[x, y^2 + x]
-------------------------------
Set FullRed;
Interreduced([xy^3+y^2+x,x]);
```

```
[x, y^2]
-------------------------------
```

V-1.14 SingleStepRed

Sorry: documentation not yet available.

V-1.15 Verbose

Sorry: documentation not yet available.

V-1.16 Commands and Functions for Panels

The following are commands and functions for panels:

`Option`	status of a panel option
`Panel`	print status of a panel's options
`Panels`	list of CoCoA panels
`Reset`	reset panels and random number seed to defaults
`ResetPanels`	reset panels to their default values
`Set, Unset`	set and unset panel options

For details look up each item by name. Online, try "`?ItemName`" or "`H.Syntax("ItemName")`".

Chapter V-2

CoCoA's Help System

V-2.1 Online Help

CoCoA's online help is roughly divided into two parts: a manual and a list of commands. The manual includes a tutorial which can be started by entering

?tutorial

If you are a new user of CoCoA, the tutorial is a good place to start. To see the titles of the sections of the online manual, enter "`H.Toc()`"; to see a list of commands, enter "`H.Commands("")`".

Each section of the manual and each command is uniquely identified by a set of keywords. The set of keywords always includes the title of the section or the title of the command. The online help command "`?`" takes a string from the user and searches for a match among the keywords. For instance, "`?gbasis`" will display information about the function "`GBasis`". Note that when using "`?`", the keyword does not appear in quotes. Also, a semicolon is not required.

The command "`??`" will return a list of entries in the manual that contain a given keyword. For instance, "`??gbasis`" returns

```
See:
  GB.Start_GBasis
  GBasis
  ReducedGBasis
-------------------------------
```

Each command is associated with a list of topics. (This applies only to commands, not to sections of the manual.) The online help function "`H.Commands`" takes a string from the user and searches for all matches among these topics. For each match, the title of the command and a brief description is displayed. For instance, "`H.Commands("poly")`" will find all commands having to do with polynomials. Information about a specific command can then be retrieved with "`?`". A list of topics is provided by "`H.Commands()`", with no argument.

IMPORTANT NOTE: Searches are case insensitive and your keyword need only be a substring to make a match.

For a thorough description of the search function "`?`", type "`? ?`" with a space bewteen the two questions marks.

Tips on using online help and summary of the online help functions appear below. Enter "`H.Browse();`" to see the next section.

V-2.2 Quick Tips for Using Online Help

Here are some tips for using the online help system:

1. Searches are case insensitive and your search string need only be a substring of a keyword to make a match. Thus, for instance, to find the section of the manual entitled "Commands and Functions for Polynomials" (IV-9.3 pg.106), it is enough to type: "`?for poly`".

2. In general, it is best to start with short keywords in order to maximize the number of matches.

3. If you cannot find a match using "`?`", try using "`H.Command`" to search by type. For example, suppose you are looking for a command that will give the remainder of the division of one polynomial into another. Trying

“`?remainder`” produces no matches. You know that the command you are looking for operates on polynomials, so try “`H.Command("poly")`”. It produces a list of a little over 30 commands, among which will be listed:
* DivAlg – division algorithm.

4. The command “`??keyword`”, with two question marks, will list all matches even if “*keyword*” exactly matches a keyword for the online help system.

V-2.3 Commands and Functions for Online Help

The following are commands and functions for CoCoA online help:

`?, Man`	search online help system
`Describe`	information about an expression
`Functions`	list the functions of a package
`H.Browse`	browse the online help system
`H.Commands`	list commands according to type
`H.Man`	search online help system (see “?, Man” (VI-1.3 pg.142))
`H.OutCommands`	print command descriptions to a file
`H.OutManual`	print the online manual to a file
`H.SetMore, H.UnSetMore`	more-device for online help
`H.Syntax`	display the syntax of a command
`H.Toc`	display the Table Of Contents of the online manual
`H.Tutorial`	run the CoCoA tutorial
`Help`	extend online help for user-defined functions
`Starting`	list functions starting with a given string

For details look up each item by name. Online, try “`?ItemName`” or “`H.Syntax("ItemName")`”. A good place to start is with the command, “`?`”, itself. To see more information about “`?`”, enter “`? ?`” (with a space between the two question marks)

V-2.4 Other Help

1. A user may provide help and sometimes (rarely) get help for a user-defined function using the “`Help`” feature of the “`Define`” command.

example

```
Help();
Type Help < Op > to get help on operator < Op >
-------------------------------
Help("GBasis");  -- note the typical response, one of the main
                 -- motivations for the author of the online manual.
No help available for GBasis
-------------------------------
```

2. The command, “`Describe`”, can be used to find more information about functions.

example

```
Describe Function("Insert");
DEFINE Insert(L,I,O)
  $cocoa/list.Insert(L,I,O)
END
-------------------------------
Describe Function("‘‘\verb&$cocoa/list.Insert&’’");
DEFINE Insert(L,I,O)
  IF NOT(Type(L) = LIST) THEN
    Return(Error(ERR.BAD_PARAMS, ": expected LIST"))
  ELSIF I = Len(L) + 1 THEN
    Append(L,O)
  ELSIF I > Len(L) THEN
```

```
    Return(Error(ERR.INDEX_TOO_BIG,I))
  ELSIF I <= 0 THEN
    Return(Error(ERR.INDEX_NEG,I))
  ELSE
    L := Concat([L[J]|J IN 1..(I - 1)],[O],[L[J]|J IN I..Len(L)]);
  END;
END
-------------------------------
```

3. The function, “`Functions`”, may be used to list all functions defined in a package. Note: “`Functions("$cocoa/user")`” lists all current user-defined functions.

example

```
Functions("$cocoa/mat");
[About(), Man(), Identity(N), Transposed(M), Submat(M,Rows,Cols),
Jacobian(S), Resultant(F,G,X), DirectSum(M1,M2), BlockMatrix(LL),
ColumnVectors(M), Complement(M,I,J), Bip(M,J), Pfaffian(M),
Sylvester(F,G,X), ExtendRows(M,N), PushRows(M,N), ConcatRows(L),
PkgName()]
-------------------------------
```

The list of packages is given by “`Packages()`”.

4. The function “`Starting(S)`” where S is a string returns a list of all functions starting with the string S.

example

```
Starting("Su");
["SubstPoly", "SubSet", "Submat", "Sum", "Subst", "Support"]
-------------------------------
```

Chapter V-3

Fine Tuning At Start-up

V-3.1 User Initialization

At the beginning of a CoCoA session, CoCoA reads in a file called "`init.coc`". This file performs certain initialization routines, reads in standard packages, and sets up global aliases for the packages. It also reads in a file called "`userinit.coc`". It is in this latter file that users should put their own commands to be run when CoCoA starts.

For example, suppose a user wants a file called "`MyFile.coc`"— containing function definitions, variable assignments, etc.—to automatically be read into the CoCoA system on start-up. It suffices to add the following line to "`userinit.coc`":

```
<<"MyFile.coc";
```

To load the package with identifier "`$contrib/mypackage`", contained in a file "`MyPackage.cpkg`" and use MP as an alias, it suffices to add the following lines to "`userinit.coc`" (you may want to look at the section of the manual entitled "Package Sourcing and Autoloading" (III-9.4 pg.68) for more details):

```
<<"MyPackage.cpkg";
Alias MP := $contrib/mypackage;
```

Chapter V-4

CoCoA Interfaces

V-4.1 CoCoA on a Macintosh

The CoCoA user interface on the Macintosh OS 9 is based on Mel Park's PlainText (v.1.6) which handles very large text files (larger than 32K). It uses standard Macintosh editing techniques, so Macintosh users should be familiar with its basic operations.

Double-clicking on the CoCoA icon or on the icon of a CoCoA document will start up the system. The system draws the menu bar, opens a text editing window and loads the CoCoA packages and then possibly user's packages (via the "`userinit.coc`" file).

After the system is started, it is ready to receive and execute commands. To execute a CoCoA command, type it into the window, ending it with a semicolon, then press the "*enter*" key. If the command occupies more than one line then highlight the whole command using the mouse and then press the "*enter*" key.

At this point the following part of the text of the active window is taken as being the "*current command*":

* if there is no selection (the cursor is blinking somewhere), then the row containing the cursor is taken as current command;

* if there is a non-null selection range, then the whole selection is taken as current command (in this way the system can process multiline commands).

The editor uses all the standard Macintosh editing techniques as well as some special ones:

* Double-clicking on a word select the entire word. * Triple-clicking anywhere in a line selects the whole line. * Double-clicking on or just before a parenthesis, a bracket, or a brace, i.e. one of following symbols "`(`" , "`)`" , "`[`" , "`]`" , "`{`" , "`}`" causes all the text between that symbol and its matching symbol to become selected.

IMPORTANT NOTE. Devices of type FILE are not yet available with the Macintosh interface.

V-4.2 CoCoA under Unix

Probably the best way to run CoCoA under Unix is through the editor, emacs, in shell-mode. In that way, one may easily edit or repeat commands. From within emacs, issue the command "`META-x shell`". You will be presented with a Unix prompt within emacs. Change to the CoCoA directory, and start CoCoA. For more information about shell-mode, issue the emacs command "`Control-h m`" after the shell is started. (You may want to use the emacs command: "`META-x set-variable comint-scroll-show-maximum-output`" to set the variable comint-scroll-show-maximum-output to the value 1.)

If running CoCoA in an xterm, it may be best to first start the xterm with the command "`xterm -sb -sl 512`" (scroll bar enabled, saving 512 lines). In addition, you may want to increase the vertical size of your window, e.g., "`xterm -sb -sl 512 -geometry 80x40`". In that way, output that scrolls off of the screen is captured and easily reviewed.

At any rate, complicated CoCoA command sequences or any sequences that you may want to repeat should be saved in a text editor. The commands can then be executed by copying and pasting into a CoCoA window or using the "`Source`" command. In addition, you may want to keep a log of your CoCoA session using the command "`OpenLog`".

V-4.3 CoCoA under Windows/DOS

Sorry this section is still under construction. Probably the best advice I could give is to repartition your disk and install Linux. ;¿

Part VI

Chapter VI-1

CoCoA commands

VI-1.1 ..

syntax

```
M .. N

where M and N are of type INT or M and N are indeterminates of the
current ring.
```

Description

If M and N are of type INT, then the expression: "`M .. N`" returns

* the list "`[M, M+1, ... ,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 $+10^9$.

If x and y are indeterminates in a ring, then

x .. y

gives the indeterminates lying 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 ::= Q[x,y,z,a,b,c,d];
z..c;
[z, a, b, c]
-------------------------------
```

See Also: CoCoA Operators (III-3.1 pg.47)

VI-1.2 ><

syntax

```
L_1 >< ... >< L_n

where each L_i is a list.
```

Description

This operator (made using a greater-than sign "`>`" and a less-than sign "`<`") returns the list whose elements form the Cartesian product of $L_1, ..., L_n$. For the N-fold product of a list with itself, one may use "`Tuples`".

example

```
L := [1,2,3];
L >< ["a","b"] >< [5];
```

```
[[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 (III-3.1 pg.47), Tuples (VI-1.270 pg.278)

VI-1.3 ?, Man

syntax

```
?key:NULL
??key:NULL
Man(S:STRING):NULL
Man(S:STRING,N:INT):NULL

where key is a literal string and N = 0 or 1.
```

Description

The "`?`" function is used to search the online help system for information matching a keyword. In the forms "`?key`" and "`??key`", the "*key*" is a literal string. This means that "*key*" should not appear enclosed in quotes. For instance, type "`?gbasis`" rather than "`?"gbasis"`". The command "`?`" is case insensitive and does not notice blank space before or after "*key*". Also, the semicolon usually required at the end of a line of CoCoA input is optional.

The online help contains a manual and a list of commands. Each section of the manual and each command has an associated list of keywords. To explain how the search system works, we will use the following terminology: say the key S "*matches*" a keyword K if S is a substring of K, not counting capitalization. The key S is an "*exact match*" if it is identical to K, not counting capitalization. The command "`?S`" searches for keywords in the online help system matching the S. If only one match occurs or an exact match occurs, the corresponding information is displayed. Otherwise, if more than one match occurs (and no exact match), matching keywords are listed. It is often advisable to make search words small at first to get as many matches as possible.

example

```
?po

See:
  Commands and Functions for Polynomials
  DensePoly
  EquiIsoDec
  Evaluation of Polynomials
  Factoring Polynomials

            --> Output suppressed  <--

?for poly

============ Commands and Functions for Polynomials =============

The following are commands and functions for polynomials:

            --> Output suppressed  <--
```

Intelligent choice of the string S can save a lot of typing. For example, there are many sections in the manual whose titles begin: "*Commands and Functions for*"; in the example above, we matched the corresponding section for polynomials by choosing the search string "*for poly*".

The command "`??S`" displays all keywords in the online help system that match S (an exact match is not required, and *all* keywords are listed even if there is an exact match).

example

```
??gbasis

See:
  GB.Start_GBasis
  GBasis
  ReducedGBasis
-------------------------------
```

Typing “`?gbasis`”, with a single question mark produces only the manual entry for “`GBasis`”.

The function “`Man`” is equivalent to “`?`” except it requires an actual string, e.g., “`Man("gbasis")`” rather than “`Man(gbasis)`”. “`Man`” with the optional second argument set to the number 0 is equivalent to “`??`”. Since “`Man`” requires more typing than “`?`”, there should never be a need to use it. (The command “`?`” was introduced in CoCoA 4.2 and is intended to replace “`Man`”.)

Note: The set of keywords associated with any section of the manual always includes the title of the manual, so it might help to first take a look at the table of contents, using “`H.Toc`”. (The titles of Parts, which are numbered by “`H.Toc`” do *not* appear as keywords: only titles of chapters and sections.) Similarly, the set of keywords for a command always includes the command’s name. The complete list of documented commands can be printed by entering “`H.Commands("")`”.

See Also: H.Commands (VI-1.105 pg.194), H.Syntax (VI-1.110 pg.196)

VI-1.4 Abs

syntax

```
Abs(N:INT):INT
Abs(N:RAT):RAT
```

Description

This function returns the absolute value of N.

example

```
Abs(-3);
3
-------------------------------
Abs(-2/3);
2/3
-------------------------------
```

VI-1.5 Adjoint

syntax

```
Adjoint(M:MAT):MAT

where M is a square matrix.
```

Description

This function returns the adjoint matrix of M.

example

```
Adjoint(Mat([[x,y,z],[t,y,x],[x,x^2,xy]]));
Mat[
  [-x^3 + xy^2, -xy^2 + x^2z, xy - yz],
  [-txy + x^2, x^2y - xz, -x^2 + tz],
  [tx^2 - xy, -x^3 + xy, -ty + xy]
]
-------------------------------
```

```
Adjoint(Mat([[1%5,2%5],[3%5,1%5]]));
Mat[
  [1 % 5, -2 % 5],
  [2 % 5, 1 % 5]
]
-------------------------------
```

VI-1.6 Alias

syntax

```
Alias B_1,..., B_r

where each B_i is a 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. Aliases for packages that are routinely loaded, may be added to "`userinit.coc`" (see "User Initialization" (V-3.1 pg.135)).

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`". For examples, see the chapter on packages in the manual, in particular the section, "Global Aliases" (III-9.5 pg.69). 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();

H      = $cocoa/help
IO     = $cocoa/io
GB     = $cocoa/gb
HP     = $cocoa/hp
HL     = $cocoa/hilop
List   = $cocoa/list

Mat    = $cocoa/mat
Latex  = $cocoa/latex
LaTeX  = $cocoa/latex
Toric  = $cocoa/toric
Coclib = $cocoa/coclib
-------------------------------
```

See Also: Alias In (VI-1.7 pg.144), Aliases (VI-1.8 pg.145), Global Aliases (III-9.5 pg.69), Introduction to Packages (III-9.1 pg.67), Local Aliases (III-9.6 pg.70)

VI-1.7 Alias In

syntax

```
Alias B_1,...,B_r In C EndAlias
```

```
where each B_i is a binding of the form: Identifier := $PackageName,
and C is a command sequence.
```

Description

This command allows one to use the aliases defined by the “`B_i`”’s in the command sequence C without affecting the global aliases.

example

```
Aliases(); -- the global aliases

H      = $cocoa/help
IO     = $cocoa/io
GB     = $cocoa/gb
HP     = $cocoa/hp
HL     = $cocoa/hilop
List   = $cocoa/list
Mat    = $cocoa/mat
Latex  = $cocoa/latex
LaTeX  = $cocoa/latex
Toric  = $cocoa/toric
Coclib = $cocoa/coclib
-------------------------------
Alias HP := $cocoa/help In HP.Man("Alias In") EndAlias;
============ Alias In =============
SYNTAX
Alias B_1,...,B_r In C EndAlias

where each B_i is a ‘‘{\it binding}’’ of the form: Identifier := $PackageName,
and C is a command sequence.
   ---> Output suppressed <---
HP.Examples();  -- the global alias HP is unaffected

    ----   STANDARD   ----
    Use R ::= Q[txyz];
    Poincare(R);
   ---> Output suppressed <---
```

See Also: Alias (VI-1.6 pg.144), Introduction to Packages (III-9.1 pg.67)

VI-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`”.

example

```
Aliases();

H      = $cocoa/help
IO     = $cocoa/io
GB     = $cocoa/gb
HP     = $cocoa/hp
HL     = $cocoa/hilop
List   = $cocoa/list
```

```
Mat    = $cocoa/mat
Latex  = $cocoa/latex
LaTeX  = $cocoa/latex
Toric  = $cocoa/toric
Coclib = $cocoa/coclib
-------------------------------
Alias TT := $abc;
Aliases();

H      = $cocoa/help
IO     = $cocoa/io
GB     = $cocoa/gb
HP     = $cocoa/hp
HL     = $cocoa/hilop
List   = $cocoa/list
Mat    = $cocoa/mat
Latex  = $cocoa/latex
LaTeX  = $cocoa/latex
Toric  = $cocoa/toric
Coclib = $cocoa/coclib
TT     = $abc
-------------------------------
```

See Also: Alias (VI-1.6 pg.144), Introduction to Packages (III-9.1 pg.67)

VI-1.9 Append

syntax

```
Append(V:LIST, E:OBJECT):NULL

where V is a variable containing a list.
```

Description

This function appends the object E to the list V.

example

```
Use R ::= Q[t,x,y,z];
L := [1,2,3];
Append(L,4);
L;
[1, 2, 3, 4]
-------------------------------
```

See Also: Concat, ConcatLists (VI-1.33 pg.159), Insert, Remove (VI-1.137 pg.210)

VI-1.10 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
-------------------------------
```

VI-1.11 Bin

syntax

```
Bin(N:INT or POLY,K:INT):INT
```

Description

This function computes the binomial coefficient, "*N choose K*" according to the formula
(N)(N-1)(N-2)...(N-K+1)/ K!

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

example

```
Bin(4,2);
6
-------------------------------
Bin(-4,3);
-20
-------------------------------
Bin(x^2+2y,3);
1/6x^6 + x^4y - 1/2x^4 + 2x^2y^2 - 2x^2y + 4/3y^3 + 1/3x^2 - 2y^2 + 2/3y
-------------------------------
It = (x^2+2y)(x^2+2y-1)(x^2+2y-2)/6;
TRUE
-------------------------------
```

See Also: BinExp, EvalBinExp (VI-1.12 pg.147)

VI-1.12 BinExp, EvalBinExp

syntax

```
BinExp(N:INT,K:INT):TAGGED($cocoa/binrepr.BinExp)
BinExp(N:INT,K:INT,Up:INT,Down:INT):INT
EvalBinExp(B:TAGGED($cocoa/binrepr.BinExp),Up:INT,Down:INT):INT

where N and K are positive integers, and Up and Down are integers.
```

Description

The first function computes the K-binomial expansion of N, i.e., the unique expression

```
N = Bin(N(K),K) + Bin(N(K-1),K-1) + ... + Bin(N(I),I)
```

where $N(K) > ... > N(I) >= 1$, for some I. The value returned is tagged for pretty printing.

The second function computes the sum of the binomial coefficients appearing in the K-binomial expansion of N after replacing each summand Bin(N(J),J) by Bin(N(J)+Up,J+Down). It is useful in generalizations of Macaulay's theorem characterizing Hilbert functions.

The third function computes the same integer as the second except it accepts BinExp(N,K) as an argument rather than N and K.

example

```
BE := BinExp(13,4);
BE;
Bin(5,4) + Bin(4,3) + Bin(3,2) + Bin(1,1)
-------------------------------
EvalBinExp(BE,1,1);
16
-------------------------------
BinExp(13,4,1,1);
16
-------------------------------
EvalBinExp(BE,0,0);  -- the integer value of BE
13
-------------------------------
```

See Also: Bin (VI-1.11 pg.147)

VI-1.13 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
-------------------------------
```

VI-1.14 BlockMatrix

syntax

```
BlockMatrix(L:LIST):MAT

where L is a list representing a block matrix.
```

Description

This function creates a block matrix. Each entry of the input list L has the form “`[M_1,...,M_k]`” where each “`M_i`” is either: (i) a matrix (or list cast-able to a matrix) or (ii) the number 0, representing a zero matrix of arbitrary size. The entry represents a row of a block matrix. For instance, if A, B, C, and D are matrices, then BlockMatrix([A,B,0],[C,0,D]] will return a matrix of the form

```
| A B 0 |
| C 0 D |.
```

The obvious restrictions on the sizes of the matrices apply. In the above example, we would need the number of rows in A and B to be the same. Similarly for C and D. The number of columns in A and C would need to be the same.

example

```
A := [[1,2,3],[4,5,6]];
B := [[1,2],[3,4]];
C := [[1,1,1],[2,2,2],[3,3,3]];
D := [[4,4],[5,5],[6,6]];
BlockMatrix([[A,B,0],[C,0,D]]);
Mat[
  [1, 2, 3, 1, 2, 0, 0],
  [4, 5, 6, 3, 4, 0, 0],
  [1, 1, 1, 0, 0, 4, 4],
  [2, 2, 2, 0, 0, 5, 5],
  [3, 3, 3, 0, 0, 6, 6]
]
-------------------------------
```

VI-1.15 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 (VI-1.232 pg.258)

VI-1.16 BringIn

syntax

```
BringIn(E:OBJECT):OBJECT

where E
 is a polynomial, a rational function, or a list/matrix/vector of
these.
```

Description

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

example

```
RR ::= Q[x[1..4],z,y];
SS ::= Z/(101)[z,y,x[1..2]];
Use RR;
F := (x[1]-y-z)^2;
F;
x[1]^2 - 2x[1]z + z^2 - 2x[1]y + 2zy + y^2
-------------------------------
Use SS;
B := BringIn(F);
B;
z^2 + 2zy + y^2 - 2zx[1] - 2yx[1] + x[1]^2
-------------------------------

Use R::=Q[x,y,z];
F := 1/2*x^3 + 34/567*x*y*z - 890;   -- a poly with rational coefficients
Use S::=Z/(101)[x,y,z];
QZP(F) = BringIn(F);
TRUE
-------------------------------
```

See Also: Image (VI-1.130 pg.206)

VI-1.17 Call

syntax

```
Call(F:FUNCTION,X_1,...,X_n):OBJECT

where X_1,...,X_n are the arguments for the function F.
```

Description

This function applies the function F to the arguments “`X_1,...X_n`”.

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 Call(LessThan,X,Y) Then Return Y Else Return X EndIf
EndDefine;

Let's use MyMax by giving two different orderings.

Define LT_Standard(X,Y)
  Return X < Y;
EndDefine;
Define LT_First(X,Y)
  Return TRUE;
End;

MyMax(Function("LT_Standard"),3,5);
5
-------------------------------
MyMax(Function("LT_First"),5,3);
3
-------------------------------
MyMax(Function("LT_First"),3,5);
5
-------------------------------
MyMax(Function("LT_First"),5,3);
3
-------------------------------
```

See Also: Function (VI-1.77 pg.182)

VI-1.18 Cast

syntax

```
Cast(E:OBJECT,T:TYPE):TYPE
```

Description

This function returns the value of the expression E after converting it to type T. If S and T are types with S ¡ T, then casting from S to T is usually possible.

example

```
L := [[1,2],[3,4]];
Type(L);
LIST
-------------------------------
Cast(L,MAT);
Mat[
  [1, 2],
  [3, 4]
]
-------------------------------
L;  -- L is unchanged; it is still a list.
[[1, 2], [3, 4]]
-------------------------------
Use Z/(5)[t];
```

```
A := 8;
A;  -- A has type INT
8
-------------------------------
Cast(A,POLY);  -- cast as a polynomial, A = -2 since the coefficient
               -- ring is Z/5Z
-2
-------------------------------
```

See Also: Data Types (III-2.6 pg.44), Shape (VI-1.245 pg.265), Type (VI-1.271 pg.278), Types (VI-1.273 pg.279)

VI-1.19 Catch

syntax

```
Catch C EndCatch;
Catch C In E EndCatch;

where C is a sequence 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 "`Catch`".

If an error occurs during the execution of C, then it is captured by the command "`Catch`" and (in the second form) assigned to the variable E. If no error occurs, then E will contain the value "`Null`". Note the use of the function "`GetErrMesg`" in the example below.

IMPORTANT NOTE: There is a bug in "`Catch`". Any "`Return`" command used inside "`Catch`" must return some value. If not, the "`Return`" command will just return from the Catch-EndCatch statement; it will not return from the function within which the statement is embedded. There is an example below.

example

```
Define Test(N)
  Catch
    PrintLn(1/N);
  In E EndCatch;
  If Type(E) = ERROR Then Print("An error occurred: ", GetErrMesg(E)) EndIf;
EndDefine;
Test(3);
1/3

-------------------------------
Test(0);

An error occurred: Division by zero
-------------------------------

         --Illustration of the BUG --
Define Test2()
  Catch
    Print("Hello ");
    Return;  -- incorrect: no value is returned
  EndCatch;
  PrintLn("world.");
EndDefine;
Test2();
Hello world.
```

```
-------------------------------
Define Test3()
  Catch
    Print("Hello ");
    Return 3;  -- correct a value is returned
  EndCatch;
  PrintLn("world.");
EndDefine;
Test3();
Hello 3
-------------------------------
```

See Also: Error (VI-1.62 pg.174), GetErrMesg (VI-1.101 pg.193)

VI-1.20 CFApprox, CFApproximants, ContFrac

syntax

```
CFApprox(X:RAT, Prec:RAT): RAT
CFApproximants(X:RAT): LIST of INT and RAT
ContFrac(X:RAT):LIST of INT
```

Description

CFApprox finds the "*simplest*" rational approximation within a maximum specified relative error. CFApproximants returns a list of all continued fraction approximants to a specified rational ContFrac returns a list of the continued fraction "*denominators*" for a given rational number.

example

```
CFApprox(1.414213, 10^(-2));
17/12
-------------------------------
CFApproximants(1.414213);
[1, 3/2, 7/5, 17/12, 41/29, 99/70, 239/169, 577/408, 816/577, 1393/985,
 6388/4517, 7781/5502, 14169/10019, 21950/15521, 36119/25540, 58069/41061,
 152257/107662, 210326/148723, 1414213/1000000]
-------------------------------
ContFrac(1.414213);
[1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 4, 1, 1, 1, 1, 1, 2, 1, 6]
-------------------------------
```

VI-1.21 Characteristic

syntax

```
Characteristic():INT
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 ::= Z/(3)[t];
S ::= Q[x,y];
Characteristic(); -- characteristic of the current ring, R
3
-------------------------------
Characteristic(S);
```

```
0
-------------------------------
```

VI-1.22 Ciao

syntax

```
Ciao
```

Description

This command is used to quit CoCoA. Note, it is issued as follows:

Ciao;

without parentheses.

See Also: Quit (VI-1.215 pg.249)

VI-1.23 Clear

syntax

```
Clear
Clear R_1,...,R_n

where the R_i are identifiers for rings.
```

Description

The first form clears the working memory, i.e, all non-global variables. In the second form, the command clears the global variables bound to the rings "`R_1,...,R_n`", i.e., the "*ring-bound*" memory for these rings. For more information on memory in CoCoA, see the chapter entitled "Memory Management" (III-8 pg.63).

The contents of the working memory are listed by the command "`Memory()`", and the global variables bound to the ring R are listed by the command "`Memory(R)`".

example

```
Use R ::= Q[x,y,z];
I := Ideal(x,y);  -- I is added to the working memory
MEMORY.X := 3;  -- a global variable
ENV.R.X := Ideal(x);  -- a global variable bound to the ring R
                    -- note that "ENV" is equivalent to "MEMORY.ENV"
Use S ::= Q[a,b];
ENV.S.Y := Ideal(a^2);  -- global variable bound to S
J := Ideal(a,b);  -- J is added to the working memory
Z := 4;  -- Z is added to the working memory
Memory();  -- the contents of the working memory
["I", "J", "UserInitFile", "Z"]
-------------------------------
Memory(R);  -- the global variables bound to R
["X"]
-------------------------------
Memory(S);  -- the global variables bound to S
["Y"]
-------------------------------
Clear;  -- clear the working memory
Memory();
[ ]
-------------------------------
Clear R;  -- clear the global variables bound to R
Memory(R);
[ ]
```

```
-------------------------------
Memory(S);
["Y"]
-------------------------------
ENV.S.Y;  -- this variable was never cleared
Ideal(a^2)
-------------------------------
```

See Also: Delete (VI-1.45 pg.166), Destroy (VI-1.49 pg.168), Memory (VI-1.173 pg.229), Memory Management (III-8 pg.63)

VI-1.24 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 (III-7.1 pg.57)

VI-1.25 CocoaLimits

syntax

```
CocoaLimits():RECORD
```

Description

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]
-------------------------------
```

VI-1.26 CocoaPackagePath

syntax

```
CocoaPackagePath():STRING
```

Description

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

example

```
CocoaPackagePath();
/usr/local/cocoa-4.3/packages
-------------------------------
```

VI-1.27 Coefficients

syntax

```
Coefficients(F:POLY or VECTOR):LIST
Coefficients(F:POLY,X:INDET):LIST
Coefficients(F:POLY,S:LIST):LIST
```

Description

This function returns the coefficients of F. In the first form, a list of the (non-zero) coefficients is returned; the order being decreasing on the terms in F as determined by the term-ordering of the ring to which F belongs.

In the second form, the function views F as a polynomial in X, and returns a list of coefficients which are polynomials in the remaining variables; their order is decreasing in powers of X, and a zero value is given for those powers of X absent from F.

In the third form, the coefficients of the specified terms are returned; their order is determined by the list S.

example

```
Use R ::= Q[x,y,z];
F := 3x^2y+5y^3-xy^5;
Coefficients(F);
[-1, 3, 5]
-------------------------------
ScalarProduct(Coefficients(F),Support(F)) = F;
TRUE
-------------------------------
V:=Vector(3x^2+y,x-5z^3);
Coefficients(V);
[-5, 3, 1, 1]
-------------------------------
ScalarProduct(Coefficients(V),Support(V))=V;
TRUE
-------------------------------
Coefficients(x^3z+xy+xz+y+2z,x);
[z, 0, y + z, y + 2z]
-------------------------------
F := (1+2*x+3*y^4+5*z^6)^7;
Skeleton := [1,x^3,y^12,z^19,x^2*y^8*z^12];
Coefficients(F, Skeleton);
[1, 280, 945, 0, 567000]
-------------------------------
```

See Also: Coefficient Rings (IV-8.3 pg.96), LC (VI-1.158 pg.220), Monomials (VI-1.181 pg.233), Support (VI-1.258 pg.272)

VI-1.28 CoeffOfTerm

syntax

```
CoeffOfTerm(T:POLY,F:POLY):C
CoeffOfTerm(T:VECTOR,F:VECTOR):C

where T is a term (no coefficient) and C is one of INT, RAT, or
ZMOD.
```

Description

This function returns the coefficient of the term T occurring in F.

example

```
Use R ::= Q[x,y,z];
F := 5xy^2-3z^3;
```

```
CoeffOfTerm(xy^2,F);
5
-------------------------------
CoeffOfTerm(x^3,F);
0
-------------------------------
CoeffOfTerm(z^3,F);
-3
-------------------------------
CoeffOfTerm(Vector(0,x^3,0),Vector(x+3,6xy-5x^3,x-z^2));
-5
-------------------------------
```

See Also: Coefficients (VI-1.27 pg.156), LC (VI-1.158 pg.220), Log (VI-1.164 pg.223), LogToTerm (VI-1.165 pg.224), Monomials (VI-1.181 pg.233), Support (VI-1.258 pg.272)

VI-1.29 Colon, :, HColon

syntax

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

Description

These functions return the quotient of M by N: the ideal of polynomials F in R such that FG is in M for all G in N. The command "`M : N`" is a shortcut for "`Colon(M,N)`".

The function "`HColon`" calculates the ideal quotient using a Hilbert-driven algorithm. It differs from "`Colon`" only when the input is inhomogeneous, in which case, "`HColon`" may be faster.

example

```
Use R ::= Q[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: Saturation, HSaturation (VI-1.238 pg.261)

VI-1.30 ColumnVectors

syntax

```
ColumnVectors(M:LIST or MAT):LIST of VECTOR

where if M is a list, is must be cast-able as a matrix.
```

Description

This function returns the list of column vectors of the matrix M.

example

```
Use R ::= Q[x,y];
M := Mat([[1,1],[x,y],[x^2,y^2]]);
M;
Mat[
  [1, 1],
  [x, y],
  [x^2, y^2]
]
-------------------------------
ColumnVectors(M);
[Vector(1, x, x^2), Vector(1, y, y^2)]
-------------------------------
```

VI-1.31 Comp

syntax

```
Comp(E:LIST, RECORD, STRING, or VECTOR,X_1:INT,...,X_k:INT):OBJECT
```

Description

This function returns "`E[X_1,...,X_k]`" except in the case where there are no additional arguments "`X_1,...,X_k`", in which case E, itself, is returned (in other words "`Comp(E)`" returns E). Since the square bracket notation works only for variables and indeterminates, this function must be used in all other situations (e.g. directly indexing into, or selecting from, the result of a function call).

example

```
Use R ::= Q[x,y,z];
L := [4,5,[6,7],8];
Comp(L,1);
4
-------------------------------
Comp(L,3);
[6, 7]
-------------------------------
Comp(L,3,2);
7
-------------------------------
F(X):=[X,X^2];  -- the following usage of "Comp" is useful for
                -- programming
F(2);
[2, 4]
-------------------------------
Comp(F(2),2);
4
-------------------------------
Struct := Record[L := [x,y,z], S := "string"];
Struct["L",3];       -- "Comp" works for records also
z
-------------------------------
Comp(Struct,"L",3);
z
-------------------------------
Comp("this is a string",3);  -- use of "Comp" with strings
i
-------------------------------
```

VI-1.32 Comps

syntax

```
Comps(V:VECTOR):LIST
```

Description

This function returns the list of components of V. It is the same as Cast(V,LIST).

example

```
Use R ::= Q[x,y];
Comps(Vector(x,x+y,x+y^2));
[x, x + y, y^2 + x]
-------------------------------
```

See Also: Comp (VI-1.31 pg.158), NumComps (VI-1.196 pg.239)

VI-1.33 Concat, ConcatLists

syntax

```
Concat(L_1:LIST,...,L_n:LIST):LIST
ConcatLists([L_1:LIST,...,L_n:LIST]):LIST
```

Description

The first function returns the list obtained by concatenating lists “`L_1,...,L_n`”.

example

```
Concat([1,2,3],[4,5],[],[6]);
[1, 2, 3, 4, 5, 6]
```

The second function takes a list whose components are lists and returns the concatenation of these components.

example

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

See Also: Concatenation (IV-3.2 pg.85)

VI-1.34 Cond

syntax

```
Cond B_1 Then E_1 EndCond
Cond B_1 Then E_1 Elsif B_2 Then E_2 Elsif ... EndCond
Cond B_1 Then E_1 Elsif B_2 Then E_2 Elsif ... Else E_r EndCond
Cond(B_1,E_1,B_2,E_2,...,E_r)

where the B_i's are boolean expressions and the E_i's are expressions.
```

Description

If B_n is the first in the sequence of B_i's to evaluate to TRUE, then E_n is returned. If none of the B_i's evaluates to TRUE, then Null is returned. The construct, “`Elsif B Then E`” can be repeated any number of times. Note: be careful not to type “`Elseif`” by mistake (it has an extraneous “`e`”).

The difference between “`Cond`” and “`If`” is that “`Cond`” is an expression which may be assigned to a variable; each of the E_i's is an expression, not a general sequence of commands (as their analogues in “`If`” might be).

example
```
Define Sign(A)
 Return Cond A>0 Then 1 Elsif A=0 Then 0 Else -1 EndCond;
EndDefine;
Sign(3);
1
-------------------------------
Define PrintSign(A)
  Return Cond(A>0,"positive",A=0,"zero","negative");
End;
PrintSign(3);
positive
-------------------------------
```

See Also: If (VI-1.128 pg.205)

VI-1.35 Contrib

syntax
```
Contrib():NULL
```

Description

This function returns a list of contributors to the main CoCoA system. In addition to these contributions, there are many other contributions which are not part of the standard distribution. For pointers to these, see the CoCoA homepage at “`http://cocoa.dima.unige.it`” or one of its mirrors.

VI-1.36 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 (VI-1.54 pg.170), Len (VI-1.159 pg.220)

VI-1.37 CurrentRing

syntax
```
CurrentRing()
```

Description

This function returns the current ring. The related command, "`RingEnv`" returns the name of the current ring.

example
```
Use R ::= Q[x,y];
Use S ::= Z/(3)[t];
CurrentRing();
Z/(3)[t]
-------------------------------
Use R;
CurrentRing();
Q[x,y]
-------------------------------
```

See Also: Ring (VI-1.234 pg.259), RingEnv (VI-1.235 pg.259), RingEnvs (VI-1.236 pg.260)

VI-1.38 Dashes

syntax
```
Dashes()
```

Description

This function returns a string of dashes:

example
```
Dashes(); 1+1;
Dashes(); 1+1;
-------------------------------
-------------------------------
2
-------------------------------
```

See Also: Equals (VI-1.60 pg.173)

VI-1.39 Date

syntax
```
Date()
```

Description

This function returns the date.

example
```
Date();
Fri Jan 30 20:47:18 1998
-------------------------------
```

VI-1.40 DecimalStr

syntax
```
DecimalStr(X:RAT):STRING
DecimalStr(X:RAT, NumDigits:INT):STRING
```

Description

This function computes a decimal approximation with NumDigits decimal digits of a rational number (default value is 3) and converts it into a string for printing. If the returned string presents less than NumDigits decimal digits, then the string represents the exact value of the input.

example

```
DecimalStr(1/3);
0.333
-------------------------------
DecimalStr(1/3, 60);
0.333333333333333333333333333333333333333333333333333333333333
-------------------------------
DecimalStr(121/10);
12.1
-------------------------------
```

See Also: FloatStr, MantissaAndExponent (VI-1.72 pg.178)

VI-1.41 Define

syntax

```
Define F(X_1,...,X_n) Help S:STRING; C EndDefine
F(X_1,...,X_n) := E
Define F(...) Help S:STRING; C EndDefine

where F is an identifier, C is a sequence of commands, the X_i's are
formal parameters and E is an expression.  The third form, which
literally includes the string ... is used for a variable number of
parameters.  The optional Help S, where S is a string, may be added
to provide help for the user.
```

Description

1. INTRODUCTION. This command adds the user-defined function F to the library. The function F can be called in the following way:

`F(E_1,...,E_n)`

where the "`E_i`"'s are expressions. The result of the evaluation of each expression "`E_i`" is assigned to the respective formal parameter "`X_i`", and the command sequence C is executed. If, during the execution of C, a statement "`Return E`" is executed, then the result of the evaluation of E is the return-value of the function F. If no "`Return`" command is executed, or "`Return`" is executed without argument, then the return-value is "`Null`".

example

```
Define Square(X)
  Return X^2;
EndDefine;

Square(5);
25
-------------------------------
```

2. SCOPE. Every variable defined or modified by the command sequence C is considered local to the function unless the variable is global or relative to a "`Var`" parameter. For the use of global variables, see "Global Memory" (III-8.3 pg.64) or the example below. See "`Var`" to learn about calling a function "*by reference*", i.e. so that the function can change the value of an existing variable.

example

```
Define Example_1(L)
  L := L + 5;
  Return L;
EndDefine;
L := 0;
Example_1(L);
5
-------------------------------
L;  -- L is unchanged despite the function call.
0
-------------------------------
Define Example_2(L)  -- Example using a global variable.
  MEMORY.X := L + 3;
EndDefine;
Example_2(10);
MEMORY.X;
13
-------------------------------
```

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

```
Define F(...) Help S:STRING; C EndDefine;
```

In this case the special variable ARGV will contain the list of the arguments passed to the function. (The statement, "`Help S;`" is optional.)

example

```
Define Sum(...)
  If Len(ARGV) = 0 Then Return Null;  -- empty sum
  Else
    Sum := 0;
    Foreach N In ARGV Do Sum := Sum+N EndForeach;
  EndIf;
  Return Sum;
EndDefine;
Sum(1,2,3,4,5);
15
-------------------------------
Sum();
Null
-------------------------------
```

4. SHORTCUT. The form "`F(X_1,...,X_n) := E`" is discouraged and may be discontinued in later versions of CoCoA. If is shorthand for "`Define F(X_1,...X_n) Return E EndDefine;`"

example

```
F(X) := X^2;
F(5);
25
-------------------------------
```

5. HELP. Inside a user-defined function, one may add the command:

```
Help S;
```

where S is a string. Then, when a user enters "`Help("F")`" where F is the identifier for the function, the string, S, is printed.

example
```
Define Test(N)
  Help "Usage: Test(N:INT):INT";
  Return N;
EndDefine;
Help "Test";
Usage: Test(N:INT):INT
-------------------------------
```

6. DEFINING RINGS INSIDE FUNCTIONS. For information on this topic, please see the section of the tutorial entitled, “Rings Inside User-Defined Functions” (II-2.18 pg.31)

See Also: An Overview of CoCoA Programming (III-1.1 pg.41), Introduction to User-Defined Functions (III-5.1 pg.53), Memory Management (III-8 pg.63), Return (VI-1.232 pg.258), Rings Inside User-Defined Functions (II-2.18 pg.31), Var (VI-1.276 pg.280)

VI-1.42 Defined

syntax
```
Defined(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.

example
```
Defined(MyVariable);
FALSE
-------------------------------
MyVariable := 3;
Defined(MyVariable);
TRUE
-------------------------------
```

VI-1.43 Deg

syntax
```
Deg(F:POLY or VECTOR):INT
Deg(F:POLY or VECTOR,X:INDET):INT
```

Description

The first form of this function returns the (weighted) degree of F. The second form returns the (un-weighted) degree of the indeterminate X in F. In either case, if F is a vector, the maximum of the degrees of its components is returned. (For the degree of a ring or quotient object, see “`Multiplicity`”.)

example
```
Use R ::= Q[x,y];
Deg(xy^2+y);
3
-------------------------------
Deg(xy^2+y,x);
1
-------------------------------
Use R ::= Q[x,y], Weights(2,3);
Deg(xy^2+y);
```

```
8
-------------------------------
Deg(xy^2+y,x);
1
-------------------------------
Deg(Vector(x^2,xy^3+y,x^2-y^5));
5
-------------------------------
Deg(Vector(x^2,xy^3+y,x^2-y^5),x);
2
-------------------------------
```

See Also: MDeg (VI-1.172 pg.228), Weights Modifier (IV-8.5 pg.97)

VI-1.44 DegLexMat, DegRevLexMat, LexMat, XelMat

syntax

```
DegLexMat(N:INTEGER):MAT
DegRevLexMat(N:INTEGER):MAT
LexMat(N:INTEGER):MAT
XelMat(N:INTEGER):MAT
```

Description

These functions return matrices defining standard term-orderings.

example

```
DegLexMat(3);
Mat[
  [1, 1, 1],
  [1, 0, 0],
  [0, 1, 0]
]
-------------------------------
DegRevLexMat(3);
Mat[
  [1, 1, 1],
  [0, 0, -1],
  [0, -1, 0]
]
-------------------------------
LexMat(3);
Mat[
  [1, 0, 0],
  [0, 1, 0],
  [0, 0, 1]
]
-------------------------------
XelMat(3);
Mat[
  [0, 0, 1],
  [0, 1, 0],
  [1, 0, 0]
]
-------------------------------
```

See Also: Ord (VI-1.202 pg.242), Orderings (IV-8.6 pg.98)

VI-1.45 Delete

syntax

```
Delete V_1, ..., V_n

where each V_i is the identifier of a variable in the working
memory.
```

Description

This function removes variables from the working memory. It will not delete global variables. For more information about memory in CoCoA, see the chapter entitled “Memory Management” (III-8 pg.63). The command “`Memory()`” lists the contents of the working memory.

example

```
Use R ::= Q[x,y,z];
X := Ideal(x,y);
Y := 3;
Use S ::= Q[a,b];
Z := a^2+b^2;
Memory();  -- the contents of the working memory
["X", "Y", "Z"]
-------------------------------
Delete X;
Memory();  -- X has been deleted from the working memory
["It", "Y", "Z"]
-------------------------------
```

See Also: Clear (VI-1.23 pg.154), Destroy (VI-1.49 pg.168), Memory (VI-1.173 pg.229), Memory Management (III-8 pg.63)

VI-1.46 DensePoly

syntax

```
DensePoly(N:INT):POLY
```

Description

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

example

```
Use R ::= Q[x,y];
DensePoly(3);
x^3 + x^2y + xy^2 + y^3
-------------------------------
Use R::=Q[x,y],Weights(2,3); --  <--- NOTE
DensePoly(1);
0
-------------------------------
DensePoly(6);
x^3 + y^2
-------------------------------
```

See Also: Randomize, Randomized (VI-1.221 pg.252)

VI-1.47 Der

syntax

```
Der(F,X:INDET):POLY

where F is a polynomial or a rational function.
```

Description

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

example

```
Use R ::= Q[x,y];
Der(xy^2,x);
y^2
-------------------------------
Define Jac(F)  --> The Jacobian matrix for a polynomial.
  Return Mat([[Der(F,X) | X In Indets()]]);
EndDefine;
Jac(xy^2);
Mat[
  [y^2, 2xy]
]
-------------------------------
Der(x/(x+y),x);
y/(x^2 + 2xy + y^2)
-------------------------------
```

See Also: Jacobian (VI-1.155 pg.218)

VI-1.48 Describe

syntax

```
Describe(E:OBJECT)
Describe E:OBJECT
```

Description

This command gives information about the expression E.

example

```
Use R ::= Z/(32003)[t,x,y];
I := Ideal(t^3-x,t^4-y);
G := SyzOfGens(I);
Print I;
Ideal(t^3 - x, t^4 - y)
-------------------------------
Describe I;
Record[Type = IDEAL, Value = Record[Gens = [t^3 - x, t^4 - y],
SyzOfGens = Module([-t^4 + y, t^3 - x], [t^4x - xy, -t^3x + x^2])]]
-------------------------------
Describe Function("$cocoa/mat.Transposed");
Define Transposed(M)
  If NOT(Type(M) = MAT) Then
    Error("Transposed: argument must be a matrix");
  EndIf;
  Return(Mat[
    J,
    1..Len(M[1]),
    TRUE,
    [M[I][J]|I In 1..Len(M)]
  ]);
EndDefine;
-------------------------------
```

See Also: Other Help (V-2.4 pg.132)

VI-1.49 Destroy

syntax

```
Destroy R_1, ... , R_n

where each R_i is the identifier of a ring.
```

Description

This command clears all global variables bound to the listed rings. Moreover, if R is a ring in the list and there are no variables in the current memory dependent upon R, then the ring identified by R is deleted; otherwise R is renamed with a name of the form “`R#N`” where N is an integer. This renamed ring is automatically removed as soon as the last variable dependent upon it is deleted.

The command will not work if one of the listed rings is the current ring.

For more information about memory in CoCoA, see the chapter entitled “Memory Management” (III-8 pg.63).

example

```
Use R ::= Q[x,y,z];
X := 3;
I := Ideal(x,y);  -- dependent on R
ENV.R.Y := 5;  -- in global memory bound to R
Use S ::= Q[a,b];
Destroy R;
RingEnvs();  -- R#1 created to hold because of the ideal I
["Q", "Qt", "R#1", "S", "Z"]
-------------------------------
Memory();  -- ENV.R.Y was destroyed along with R
["I", "It", "X"]
-------------------------------
I;  -- I was not destroyed
R#1 :: Ideal(x, y)
-------------------------------
I := 3;  -- overwrite I; it is no longer dependent on a CoCoA ring
Describe Memory();
------------[Memory]-----------
I = 3
It = R#1 :: Ideal(x, y)
X = 3
-------------------------------
RingEnvs();  -- subtle point here: the variable "It" is still dependent
             -- on R#1
["Q", "Qt", "R#1", "S", "Z"]
-------------------------------
RingEnvs();  -- However, the previous command caused It to become a
             -- string; hence, R#1 disappears.
["Q", "Qt", "S", "Z"]
-------------------------------
```

See Also: Clear (VI-1.23 pg.154), Delete (VI-1.45 pg.166), Memory Management (III-8 pg.63)

VI-1.50 Det

syntax

```
Det(M:MAT)

the resulting type depends on the entries of the matrix.
```

Description

This function returns the determinant of the matrix M. The resulting type depends on the types of the entries of the matrix.

example

```
Use R ::= Q[x];
M := Mat([[x,x^2],[x,x^3]]);
Det(M);
x^4 - x^3
-------------------------------
Det(Mat([[1,2],[0,5]]));
5
-------------------------------
```

See Also: Minors (VI-1.176 pg.230)

VI-1.51 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, Remove (VI-1.137 pg.210)

VI-1.52 Dim

syntax

```
Dim(R:RING or TAGGED(Quotient)):INT
```

Description

This function computes the dimension of R. The weights of the indeterminates of the current ring must all be 1.

The coefficient ring must be a field.

example

```
Use R ::= Q[x,y,z];
Dim(R);
3
-------------------------------
Dim(R/Ideal(y^2-x,xz-y^3));
1
-------------------------------
```

VI-1.53 Discriminant

syntax

```
Discriminant(F:POLY):POLY
Discriminant(F:POLY, X:INDET):POLY
```

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 to be the resultant of F and its derivative with respect to X.

example

```
Use R ::= Q[x,y];
Discriminant(x^2+3y^2, x);
12y^2
-------------------------------
Discriminant(x^2+3y^2, y);
36x^2
-------------------------------
Discriminant((x+1)^20+2);
54975581388800000000000000000000
-------------------------------
```

See Also: Resultant (VI-1.231 pg.257)

VI-1.54 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 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 (VI-1.36 pg.160)

VI-1.55 Div, Mod

syntax

```
Div(N:INT,D:INT):INT
Mod(N:INT,D:INT):INT
```

Description

If N = Q*D + R, and $0 \leq R < |D|$, then "`Div(N,D)`" returns Q and "`Mod(N,D)`" returns R.

NOTE: To perform the division algorithm on a polynomial or vector, use "`NR`" (normal remainder) to find the remainder, or "`DivAlg`" to get both the quotients and the remainder. To determine if a polynomial is in a given ideal or a vector is in a given module, use "`NF`" or "`IsIn`", and to find a representation in terms of the generators use "`GenRepr`".

example

```
Div(10,3);
3
-------------------------------
Mod(10,3);
1
-------------------------------
```

See Also: DivAlg (VI-1.56 pg.171), GenRepr (VI-1.98 pg.191), NF (VI-1.190 pg.237), NR (VI-1.194 pg.238)

VI-1.56 DivAlg

syntax

```
DivAlg(X:POLY,L:LIST of POLY):RECORD
DivAlg(X:VECTOR,L:LIST of VECTOR):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::= Q[x,y,z];
F := x^2y+xy^2+y^2;
L := [xy-1,y^2-1];
DivAlg(F,[xy-1,y^2-1]);
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
-------------------------------
V := Vector(x^2+y^2+z^2,xyz);
L := [Vector(x,y),Vector(y,z),Vector(z,x)];
DivAlg(V,L);
Record[Quotients = [0, -z^2, yz], Remainder = Vector(x^2 + y^2 + z^2, z^3)]
-------------------------------
```

See Also: Div, Mod (VI-1.55 pg.170), GenRepr (VI-1.98 pg.191), NF (VI-1.190 pg.237), NR (VI-1.194 pg.238)

VI-1.57 E_

syntax

```
E_(K:INT,N:INT or MODULE):VECTOR
```

Description

If N is an integer, this function returns the K-th canonical vector of the free module of rank N over the current ring. If N is a module, it returns the K-th canonical vector of N.

example

```
Use R ::= Q[x,y];
E_(4,7);
Vector(0, 0, 0, 1, 0, 0, 0)
-------------------------------
M := Module([x^2,0,y^2],[x^3,x+y,y^3]);
E_(2,M);
Vector(0, 1, 0)
-------------------------------
```

VI-1.58 Elim

syntax

```
Elim(X:INDETS,M:IDEAL):IDEAL
Elim(X:INDETS,M:MODULE):MODULE

where X is an indeterminate or a list of indeterminates.
```

Description

This function returns the ideal or module obtained by eliminating the indeterminates X 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`” and “Predefined Term-Orderings” (IV-8.7 pg.98)).

example

```
Use R ::= Q[t,x,y,z];
Set Indentation;
Elim(t,Ideal(t^15+t^6+t-x,t^5-y,t^3-z));
Ideal(
  -z^5 + y^3,
  -y^4 - yz^2 + xy - z^2,
  -xy^3z - y^2z^3 - xz^3 + x^2z - y^2 - y,
  -y^2z^4 - x^2y^3 - xy^2z^2 - yz^4 - x^2z^2 + x^3 - y^2z - 2yz - z,
  -y^3z^3 + xz^3 - y^3 - y^2)
-------------------------------
Use R ::= Q[t,s,x,y,z,w];
t..x;
[t, s, x]
-------------------------------
Elim(t..x,Ideal(t-x^2zw,x^2-t,y^2t-w)); -- Note the use of t..x.
Ideal(-zw^2 + w)
-------------------------------
Use R ::= Q[t[1..2],x[1..4]];
I := Ideal(x[1]-t[1]^4,x[2]-t[1]^2t[2],x[3]-t[1]t[2]^3,x[4]-t[2]^4);
t;
[t[1], t[2]]
-------------------------------
Elim(t,I);                         -- Note the use t.
Ideal(x[3]^4 - x[1]x[4]^3, x[2]^4 - x[1]^2x[4])
-------------------------------
```

See Also: Orderings (IV-8.6 pg.98), Predefined Term-Orderings (IV-8.7 pg.98)

VI-1.59 EqSet

syntax

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

Description

This function returns TRUE if Set(L) equals Set(M), otherwise it returns FALSE.

example

```
L := [1,2,2];
M := [2,1];
EqSet(L,M);
TRUE
-------------------------------
```

See Also: Intersection, IntersectionList (VI-1.140 pg.212), IsSubset (VI-1.152 pg.217)

VI-1.60 Equals

syntax

```
Equals()
```

Description

This function returns a string of equal signs:

example

```
Equals();
===============================
-------------------------------
```

See Also: Dashes (VI-1.38 pg.161)

VI-1.61 EquiIsoDec

syntax

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

Description

This function computes an equidimensional isoradical decomposition of I, i.e. a list of unmixed ideals $I_1, ..., I_k$ such that the radical of I is the intersection of the radicals of $I_1, ..., I_k$. 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::=Q[x,y,z];
I := Intersection(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 (VI-1.211 pg.247), Radical (VI-1.218 pg.250), RadicalOfUnmixed (VI-1.219 pg.251)

VI-1.62 Error

syntax

```
Error(S:STRING):ERROR
```

Description

This function returns an error labeled with the string S.

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: Argument must be an integer.
CONTEXT: Error("Argument must be an integer.")
-------------------------------
T(7);
2
-------------------------------
```

See Also: Catch (VI-1.19 pg.152), GetErrMesg (VI-1.101 pg.193)

VI-1.63 Eval

syntax

```
Eval(E:OBJECT,L:LIST):OBJECT
```

Description

This function substitutes the N-th element of L for the N-th indeterminate of the current ring for all N less than or equal to the minimum of the number of indeterminates of the current ring and the number of components of L.

example

```
Use Q[xy];
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]);
[2y, 0]
-------------------------------
```

See Also: Evaluation of Polynomials (IV-9.2 pg.106), Image (VI-1.130 pg.206), Subst (VI-1.257 pg.271), Substitutions (II-2.15 pg.29)

VI-1.64 EvalHilbertFn

syntax

```
EvalHilbertFn(H:TAGGED($cocoa/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 ::= Q[w,x,y,z];
I := Ideal(z^2-xy,xz^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 (VI-1.116 pg.198), HilbertPoly (VI-1.118 pg.199)

VI-1.65 Fact

syntax

```
Fact(N:INT):INT

where N is a non-negative integer.
```

Description

This function returns N factorial.

example

```
Fact(5);
120
-------------------------------
Fact(100);
9332621544394415268169923885626670049071596826438162146859
2963895217599993229915608941463976156518286253697920827223
758251185210916864000000000000000000000000
-------------------------------
```

See Also: Bin (VI-1.11 pg.147)

VI-1.66 Factor

syntax

```
Factor(F:POLY):LIST
```

Description

This function factors a polynomial in its ring of definition. Multivariate factorization is not yet supported over finite fields. (For information about the algorithm, consult "Pointers to the Literature" (II-1.5 pg.20).)

The function returns a list of the form "`[[F_1,N_1],...,[F_r,N_r]]`" where "`F_1^N_1 ... F_r^N_r = F`" and the "`F_i`" are irreducible in the current ring.

example

```
Use R ::= Q[x,y];
F := x^12 - 37x^11 + 608x^10 - 5852x^9 + 36642x^8 - 156786x^7 + 468752x^6
    - 984128x^5 + 1437157x^4 - 1422337x^3 + 905880x^2 - 333900x + 54000;
Factor(F);
[[x - 2, 1], [x - 4, 1], [x - 6, 1], [x - 3, 2], [x - 5, 3], [x - 1, 4]]
-------------------------------
G := Product([W[1]^W[2] | W In It]);  -- check solution
F = G;
TRUE
-------------------------------
Factor((8x^2+16x+8)/27); -- the "content" appears as a factor of degree 0;
                         -- it is not factorized into prime factors.
[[x + 1, 2], [8/27, 1]]
-------------------------------
F := (x+y)^2*(x^2y+y^2x+3);
F;
x^4y + 3x^3y^2 + 3x^2y^3 + xy^4 + 3x^2 + 6xy + 3y^2
-------------------------------
Factor(F);  -- multivariate factorization
[[x^2y + xy^2 + 3, 1], [x + y, 2]]
-------------------------------
Use Z/(37)[x];
Factor(x^6-1);
[[x - 1, 1], [x + 1, 1], [x + 10, 1], [x + 11, 1], [x - 11, 1], [x - 10, 1]]
-------------------------------
Factor(2x^2-4); -- over a finite field the factors are made monic;
                -- leading coeff appears as "content" if it is not 1.
[[x^2 - 2, 1], [2, 1]]
-------------------------------
```

VI-1.67 Fields

syntax

```
Fields(P:RECORD):LIST
```

Description

This function returns a list of all of the fields of the record P.

example

```
P := Record[ Name = "David", Number = 3728852, Data = ["X","Y"] ];
Fields(P);
["Data", "Name", "Number"]
-------------------------------
P.Data;
```

```
["X", "Y"]
-------------------------------
```

See Also: Introduction to Records (IV-5.1 pg.89), Record (VI-1.225 pg.254)

VI-1.68 First

syntax
```
First(L:LIST):OBJECT
First(L:LIST,N:INT):OBJECT
```

Description

In the first form, the function is the same as the function "`Head`"; it returns the first element of the list L. In the second form, it returns the list of the first N elements of L.

example
```
L := [1,2,3,4,5];
First(L);
1
-------------------------------
First(L,3);
[1, 2, 3]
-------------------------------
```

See Also: Head (VI-1.114 pg.197), Last (VI-1.156 pg.219)

VI-1.69 FirstNonZero, FirstNonZeroPos

syntax
```
FirstNonZero(V:VECTOR):POLY
FirstNonZeroPos(V:VECTOR):POLY
```

Description

The first function returns the first non-zero entry of V. The second function returns the index of the first non-zero entry of V. If either function is handed a zero vector then an error is signalled.

example
```
Use R ::= Q[x,y,z];
V := Vector(0,0,x^2+yz,0,z^2);
FirstNonZero(V);
x^2 + yz
-------------------------------
FirstNonZeroPos(V);
3
-------------------------------
V[FirstNonZeroPos(V)];
x^2 + yz
-------------------------------
```

See Also: NonZero (VI-1.192 pg.238)

VI-1.70 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 := 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, Flatten(L,N) = Flatten(Flatten(L,N-1),1). For N large, depending on L, Flatten(L,N) gives the same result as Flatten(L).

example

```
Flatten([1,["a","b",[2,3,4],"c","d"],5,6]);
[1, "a", "b", 2, 3, 4, "c", "d", 5, 6]
-------------------------------
L := [1,2, [3,4], [5, [6,7,[8,9]]]];
Flatten(L,1);
[1, 2, 3, 4, 5, [6, 7, [8, 9]]]
-------------------------------
Flatten(It,1);
[1, 2, 3, 4, 5, 6, 7, [8, 9]]
-------------------------------
Flatten(L,2);  -- same as in the previous line
[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]
-------------------------------
```

VI-1.71 FloatApprox

syntax

```
FloatApprox(X:RAT, RelErr:RAT):RAT
```

Description

This function computes an approximation of the form $M * 10^E$ to a rational number X and to within a maximum specified relative error. RelErr says indirectly how many decimal digits to include in the mantissa. Compare with "`MantissaAndExponent`".

example

```
FloatApprox(1/3, 10^(-2));
333/1000
-------------------------------
FloatApprox(1000000/3, 10^(-2));
333000
-------------------------------
FloatApprox(1/3, 10^(-9));
3333333333/10000000000
-------------------------------
```

See Also: DecimalStr (VI-1.40 pg.161), FloatStr, MantissaAndExponent (VI-1.72 pg.178)

VI-1.72 FloatStr, MantissaAndExponent

syntax

```
FloatStr(X:RAT):STRING
FloatStr(X:RAT, Prec:INT):STRING
MantissaAndExponent(X:RAT, Prec:INT):RECORD
```

Description

The first two functions convert 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 10. Note that an exponent is always included; the only exception being the number zero which is converted to the string “0”.

The third function converts a rational number into a Record with components named Mantissa and Exponent. The value of the Exponent field is the unique integer E such that $1 \le X * 10^E \le 10$, and the value of Mantissa is the nearest integer to $(X * 10^E) * 10^(Prec - 1)$. As an exception the case of X=0 always produces zero values for both components of the record.

example

```
FloatStr(2/3);         -- last printed digit is rounded
6.666666667*10^(-1)
-------------------------------
FloatStr(7^510);       -- no arbitrary limit on exponent range
1.000000938*10^431
-------------------------------
FloatStr(1/81, 50);    -- precision of mantissa specified by user
1.2345679012345679012345679012345679012345679012346*10^(-2)
-------------------------------
FloatStr(1/2);         -- trailing zeroes are not suppressed
5.000000000*10^(-1)
-------------------------------
MantissaAndExponent(1/2,3);           --  1/2 = 5.00*10^(-1)
Record[Exponent = -1, Mantissa = 500]
-------------------------------
MantissaAndExponent(0.9999, 3);       --  0.9999 rounds up to give 1.00
Record[Exponent = 0, Mantissa = 100]
-------------------------------
```

See Also: DecimalStr (VI-1.40 pg.161), FloatApprox (VI-1.71 pg.178)

VI-1.73 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 “`N_1, N_1+D, N_1+2D`”, etc. until the greatest value less than or equal to “`N_2`” is reached. If “`N_2 < N_1`”, then C is not executed.

Note: Large values for “`N_1, N_2`”, 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 Sort(Var(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];
Sort(M);
M;
[1, 2, 3, 4, 5]
-------------------------------
```

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

See Also: Foreach (VI-1.74 pg.180), Repeat (VI-1.227 pg.255), While (VI-1.280 pg.283)

VI-1.74 Foreach

syntax

```
Foreach X In L Do C EndForeach

where X is a dummy variable, L is a list, and C is a sequence of commands.
```

Description

The dummy variable X is assigned the value of each component of L in turn. After each assignment the command sequence C is executed. Note: don’t forget to capitalize “`In`”.

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 ::= Q[x,y,z];
F := x^2y + 3y^2z - z^3;
J := [Der(F,X) | X In Indets()];  -- the Jacobian for F
J;
[2xy, x^2 + 6yz, 3y^2 - 3z^2]
-------------------------------
Foreach X In J Do -- square each component of the Jacobian
  PrintLn(X^2);
EndForeach;
```

```
4x^2y^2
x^4 + 12x^2yz + 36y^2z^2
9y^4 - 18y^2z^2 + 9z^4

-------------------------------
```

See Also: For (VI-1.73 pg.179), Repeat (VI-1.227 pg.255), While (VI-1.280 pg.283)

VI-1.75 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 := [1,2,3];
M := Format(L,20);
M;
           [1, 2, 3]
-------------------------------
Type(L);
LIST
-------------------------------
Type(M);
STRING
-------------------------------
Format(L,2);  -- "Format" does not truncate
[1, 2, 3]
-------------------------------
```

See Also: IO.SprintTrunc (VI-1.142 pg.213), Latex (VI-1.157 pg.219), Sprint (VI-1.252 pg.269)

VI-1.76 Fraction

syntax

```
Fraction(E:OBJECT,F:OBJECT)
```

Description

This function returns E/F provided the quotient is defined (see “`Algebraic Operators`”).

example

```
Use R ::= Q[x,y];
Fraction(2,3);
2/3
-------------------------------
Fraction(2,4);
1/2
-------------------------------
Fraction(x,x+y);
x/(x + y)
-------------------------------
Fraction(5%11,6%11);
10 % 11
-------------------------------
```

See Also: Algebraic Operators (III-3.2 pg.47)

VI-1.77 Function

syntax

```
Function(S:STRING):FUNCTION
Function(P:STRING,S:STRING):FUNCTION
```

Description

This function returns the function—user-defined or built in—identified by the string S. In the second form, one first provides the name of the package, then the name of the function. (An alternative is the syntax "`Function(P.S)`".

One may use "`Function`" to assign a function to a variable which can then be executed via the function "`Call`":

example

```
F := Function("Deg");
F;
Deg(...)
-------------------------------
Type(F);
FUNCTION
-------------------------------
Call(F,x+y^2);
2
-------------------------------
-- the Call-statement here is equivalent to:
Deg(x+y^2);
2
-------------------------------
Function("Insert");
Insert(L,I,O)
-------------------------------
Function("$cocoa/list","Insert"); -- or "Function("$cocoa/list.Insert")"
Insert(L,I,O)
-------------------------------
```

See Also: Call (VI-1.17 pg.150), Functions (VI-1.78 pg.182)

VI-1.78 Functions

syntax

```
Functions(S:STRING):LIST of FUNCTION
```

Description

This function returns a list of functions defined in the package identified by S. (The function "`Packages`" lists the packages currently loaded into memory.)

example

```
Functions("$cocoa/binrepr");
[About(), Man(), Initialize(), PolyBinRepr_xi(P), PolyBinRepr_xii(P),
BinExp(...), EvalBinExp(BE,Up,Down), Aux_BinExp(H,N), Tagged(X,T),
Print_Bin(B), Print_BinExp(BE), Print_BinRepr(BR), PkgName()]
-------------------------------
L:= It;
Describe L[5];
DEFINE BinExp(...)
```

```
  IF Shape(ARGV) = [
    INT,
    INT] THEN
    Return($cocoa/binrepr.Aux_BinExp(ARGV[1],ARGV[2]))
  ELSIF Shape(ARGV) = [
    INT,
    INT,
    INT,
    INT] THEN
    Return(EvalBinExp($cocoa/binrepr.Aux_BinExp(ARGV[1],ARGV[2]),ARGV[3],ARGV[4]))
  ELSE
    Error(ERR.BAD_PARAMS,"(BinExp arguments must be 2 or 4 INT)")
  END;
END
-------------------------------
```

VI-1.79 GB.Complete

syntax

```
GB.Complete(M:IDEAL or MODULE):NULL
```

Description

This function completes a calculation started in the Interactive Groebner Framework. For explanations and examples, see "The Interactive Groebner Framework" (IV-13.3 pg.118).

See Also: The Interactive Groebner Framework (IV-13.3 pg.118)

VI-1.80 GB.GetBettiMatrix

syntax

```
GB.GetBettiMatrix(M:IDEAL or MODULE):TAGGED($cocoa/io.Matrix)
```

Description

This function, if used after executing "`Res(M)`", prints the Betti matrix for M. Within the Interactive Groebner Framework, in which resolutions may be computed one step at a time, the function returns the Betti matrix for the part of the resolution computed so far. See "`GB.GetRes`" for an example.

example

```
Use R ::= Q[t,x,y,z];
I := Ideal(x^2-yt,xy-zt,xy);
Res(I);
0 --> R^2(-5) --> R^4(-4) --> R^3(-2)
-------------------------------
GB.GetBettiMatrix(I);
--------------


--------------
   0    0    0
   0    0    3
   0    0    0
   0    4    0
   2    0    0
--------------


-------------------------------
```

See Also: GB.GetRes (VI-1.83 pg.185), GB.GetResLen (VI-1.84 pg.186), Res (VI-1.228 pg.255), The Interactive Groebner Framework (IV-13.3 pg.118)

VI-1.81 GB.GetNthSyz

syntax

```
GB.GetNthSyz(M:IDEAL or MODULE,N:INT):MODULE
```

Description

This function, if used after executing "`Res(M)`", returns the Nth syzygy module for M. Within the Interactive Groebner Framework, in which resolutions may be computed one step at a time, the function returns the part of the Nth syzygy module computed so far. In contrast, the function "`Syz`" always determines the complete syzygy module even from within the Interactive Groebner Framework.

example

```
Use R ::= Q[t,x,y,z];
I := Ideal(x^2-yt,xy-zt,xy);
GB.Start_Res(I);
GB.Step(I);
GB.GetNthSyz(I,1); GB.GetNthSyz(I,2);
Module([0])
-------------------------------
Module([0])
-------------------------------
GB.Step(I);
GB.GetNthSyz(I,1); GB.GetNthSyz(I,2);
Module([0, 0])
-------------------------------
Module([0])
-------------------------------
GB.Steps(I,5);
GB.GetNthSyz(I,1); GB.GetNthSyz(I,2);
Module([-xz, -y^2, yz])
-------------------------------
Module([0])
-------------------------------
GB.Complete(I);
GB.GetNthSyz(I,1); GB.GetNthSyz(I,2);
Module([-xz, -y^2, yz], [tz, xy, 0], [0, -x^2 + ty, -tz], [-x^2 + ty, 0, xy])
-------------------------------
Module([-x, -y, 0, z], [-t, -x, -y, 0])
-------------------------------
```

See Also: GB.GetNthSyzShifts (VI-1.82 pg.184), GB.GetRes (VI-1.83 pg.185), Res (VI-1.228 pg.255), Syz (VI-1.260 pg.273), The Interactive Groebner Framework (IV-13.3 pg.118)

VI-1.82 GB.GetNthSyzShifts

syntax

```
GB.GetNthSyzShifts(M:IDEAL or MODULE,N:INT):TAGGED(shifts)
```

Description

This function, if used after executing "`Res(M)`", returns the shifts for the Nth syzygy module for M. Within the Interactive Groebner Framework, in which resolutions may be computed one step at a time, the function returns shifts of the part of the Nth syzygy module computed so far.

example

```
Use R ::= Q[t,x,y,z];
I := Ideal(x^2-yt,xy-zt,xy);
GB.Start_Res(I);
GB.Steps(I,6);
GB.GetNthSyzShifts(I,2);
Shifts([x^2yz])
-------------------------------
GB.Complete(I);
GB.GetNthSyzShifts(I,2);
Shifts([x^2yz, txyz, tx^2z, x^3y])
-------------------------------
J := Ideal(t,x)^3;
Res(J);
0 --> R^3(-4) --> R^4(-3)
-------------------------------
GB.GetNthSyzShifts(J,1);
Shifts([x^3, tx^2, t^2x, t^3])
-------------------------------
GB.GetNthSyzShifts(J,2);
Shifts([tx^3, t^2x^2, t^3x])
-------------------------------
SS := It;
SS[1];
tx^3
-------------------------------
```

See Also: GB.GetNthSyz (VI-1.81 pg.184), GB.GetRes (VI-1.83 pg.185), Res (VI-1.228 pg.255), Shifts (IV-12.3 pg.114), The Interactive Groebner Framework (IV-13.3 pg.118)

VI-1.83 GB.GetRes

syntax

```
GB.GetRes(M:IDEAL or MODULE):TAGGED($cocoa/gb.Res)
```

Description

This function returns the part of the resolution of M computed so far. It does not compute the resolution of M as does "`Res`", rather, "`GB.GetRes`" is intended primarily to be used within the Interactive Groebner Framework. Thus, for example, "`GB.GetRes`" may be used to examine the resolution as it is computed, one step at a time.

example

```
Use R ::= Q[txyz];
I := Ideal(x^2-yt,xy-zt,xy);
GB.Start_Res(I); -- start Interactive Groebner Framework
GB.Step(I); -- take one step in calculation of resolution
GB.GetRes(I);  -- the resolution so far
0 --> R(-2)
-------------------------------
GB.Step(I);  -- one more step
GB.GetResLen(I);  -- the computed resolution still has length 1
1
-------------------------------
GB.GetBettiMatrix(I);  -- the Betti Matrix so far
----

----
   0
   2
----

-------------------------------
GB.GetRes(I);
0 --> R^2(-2)
-------------------------------
GB.Steps(I,5); -- five more steps
GB.GetRes(I);
0 --> R(-4) --> R^3(-2)
-------------------------------
GB.Complete(I); -- complete the calculation
GB.GetResLen(I);
3
-------------------------------
GB.GetBettiMatrix(I);
--------------

--------------
   0    0    0
   0    0    3
   0    0    0
   0    4    0
   2    0    0
--------------

-------------------------------
GB.GetRes(I);
0 --> R^2(-5) --> R^4(-4) --> R^3(-2)
-------------------------------
```

See Also: GB.GetBettiMatrix (VI-1.80 pg.183), GB.GetResLen (VI-1.84 pg.186), Res (VI-1.228 pg.255), The Interactive Groebner Framework (IV-13.3 pg.118)

VI-1.84 GB.GetResLen

syntax
```
GB.GetResLen(M:IDEAL or MODULE):INT
```

Description

This function, if used after executing "`Res(M)`", prints the length of the resolution for M. Within the Interactive Groebner Framework, in which resolutions may be computed one step at a time, the function returns the length of the part of the resolution computed so far. See "`GB.GetRes`" for an example.

example
```
Use R ::= Q[t,x,y,z];
I := Ideal(x^2-yt,xy-zt,xy);
Res(I);
0 --> R^2(-5) --> R^4(-4) --> R^3(-2)
-------------------------------
GB.GetResLen(I);
3
-------------------------------
```

See Also: GB.GetBettiMatrix (VI-1.80 pg.183), GB.GetRes (VI-1.83 pg.185), Res (VI-1.228 pg.255), The Interactive Groebner Framework (IV-13.3 pg.118)

VI-1.85 GB.ResReport

syntax

```
GB.ResReport(M:IDEAL or MODULE):NULL
```

Description

This function reports statistics about the current status of a resolution computation begun in the Interactive Groebner Framework. For explanations and examples, see "The Interactive Groebner Framework" (IV-13.3 pg.118).

See Also: GB.Stats (VI-1.92 pg.188), The Interactive Groebner Framework (IV-13.3 pg.118)

VI-1.86 GB.Start_GBasis

syntax

```
GB.Start_GBasis(M:IDEAL or MODULE):NULL
```

Description

This command starts the Interactive Groebner Framework for calculating a Groebner basis for M. For explanations and examples, see "The Interactive Groebner Framework" (IV-13.3 pg.118).

See Also: The Interactive Groebner Framework (IV-13.3 pg.118)

VI-1.87 GB.Start_MinGens

syntax

```
GB.Start_MinGens(M:IDEAL or MODULE):NULL
```

Description

This command starts the Interactive Groebner Framework for calculating minimal generators for M. For explanations and examples, see "The Interactive Groebner Framework" (IV-13.3 pg.118).

See Also: The Interactive Groebner Framework (IV-13.3 pg.118)

VI-1.88 GB.Start_MinSyzMinGens

syntax

```
GB.Start_MinSyzMinGens:  COMMAND ELIMINATED
```

Description

The "`GB.Start_MinSyzMinGens`" command has been removed.

See Also: The Interactive Groebner Framework (IV-13.3 pg.118)

VI-1.89 GB.Start_Res

syntax

```
GB.Start_Res(M:IDEAL or MODULE):NULL
```

Description

This command starts the Interactive Groebner Framework for calculating a resolution for M. For explanations and examples, see "The Interactive Groebner Framework" (IV-13.3 pg.118).

See Also: The Interactive Groebner Framework (IV-13.3 pg.118)

VI-1.90 GB.start_Syz

syntax

```
GB.Start_Syz(M:IDEAL or MODULE):NULL
```

Description

This command starts the Interactive Groebner Framework for calculating syzygies for M. For explanations and examples, see "The Interactive Groebner Framework" (IV-13.3 pg.118).

See Also: The Interactive Groebner Framework (IV-13.3 pg.118)

VI-1.91 GB.start_SyzMinGens

syntax

```
GB.Start_SyzMinGens:  COMMAND ELIMINATED
```

Description

The "`GB.Start_SyzMinGens`" command has been removed.

See Also: The Interactive Groebner Framework (IV-13.3 pg.118)

VI-1.92 GB.Stats

syntax

```
GB.Stats(M:IDEAL or MODULE):NULL
```

Description

This function displays information about the current status of a calculation started in the Interactive Groebner Framework. For explanations and examples, see "The Interactive Groebner Framework" (IV-13.3 pg.118).

See Also: GB.ResReport (VI-1.85 pg.187), The Interactive Groebner Framework (IV-13.3 pg.118)

VI-1.93 GB.Step, GB.Steps

syntax

```
GB.Step(M:IDEAL or MODULE):NULL
GB.Steps(M:IDEAL or MODULE, N:INT):NULL
```

Description

The first function performs one step in a calculation started in the Interactive Groebner Framework. The second, performs N steps. For explanations and examples, see "The Interactive Groebner Framework" (IV-13.3 pg.118).

See Also: The Interactive Groebner Framework (IV-13.3 pg.118)

VI-1.94 GBasis

syntax

```
GBasis(M:IDEAL, MODULE, or TAGGED(Quotient)):LIST
```

Description

If M is an ideal or module, this function returns a list whose components form a Groebner basis for M with respect to the term-ordering of the current ring. If M is a quotient of the current ring by an ideal I or of a free module by a submodule N, then the Groebner basis for M is defined to be that of I or N, respectively.

If M is a variable, then the result is stored in M for later use. It can be retrieved as M.GBasis and can also be seen using the command "Describe" (VI-1.48 pg.167).

For a reduced Groebner basis, use the command "ReducedGBasis" (VI-1.226 pg.254).
The coefficient ring must be a field.

example

```
Use R ::= Q[t,x,y];
I := Ideal(t^3-x,t^4-y);
Describe I;
Record[Type = IDEAL, Value = Record[Gens = [t^3 - x, t^4 - y]]]
-------------------------------
GBasis(I);
[t^3 - x, -tx + y, t^2y - x^2, x^3 - ty^2]
-------------------------------
Describe(I);  -- the Groebner basis has been stored in I
Record[Type = IDEAL, Value = Record[Gens = [t^3 - x, t^4 - y], GBasis
= [t^3 - x, -tx + y, t^2y - x^2, x^3 - ty^2]]]
-------------------------------
I.GBasis;
[t^3 - x, -tx + y, t^2y - x^2, x^3 - ty^2]
-------------------------------


For fine control and monitoring of Groebner basis calculations, see
``The Interactive Groebner Framework'' (\ref{The Interactive Groebner Framework} pg.\pageref{The Interact
```

See Also: Introduction to Groebner Bases in CoCoA (IV-13.1 pg.117)

VI-1.95 GBM, HGBM

syntax

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

Description

These functions compute the intersection of ideals corresponding to zero-dimensional schemes: GBM is for affine schemes, and HGBM for projective schemes. The list L must be a list of ideals. The function "`IntersectionList`" 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 Q[x,y,z];
I1:=IdealOfPoints([[1,2,1], [0,1,0]]);     -- a simple affine scheme
I2:=IdealOfPoints([[1,1,1], [2,0,1]])^2;   -- another affine scheme
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)
-------------------------------


Use Q[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: Finite Point Sets: Buchberger-Moeller (II-2.24 pg.35), IdealAndSeparatorsOfPoints (VI-1.123 pg.202), IdealAndSeparatorsOfProjectivePoints (VI-1.124 pg.203), IdealOfPoints (VI-1.125 pg.204), IdealOfProjectivePoints (VI-1.126 pg.204)

VI-1.96 GCD, LCM

syntax

```
GCD (F_1:INT,...,F_n:INT):INT
GCD (L:LIST of INT):INT
LCM (F_1:INT,...,F_n:INT):INT
LCM (L:LIST of INT):INT

GCD(F_1:POLY,...,F_n:POLY):POLY
GCD (L:LIST of POLY):POLY
LCM(F_1:POLY,...,F_n:POLY):POLY
LCM (L:LIST of POLY):POLY
```

Description

These functions return the greatest common divisor and least common multiple, respectively, of "`F_1,...,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 ::= Q[x,y];
F := x^2-y^2;
G := (x+y)^3;
GCD(F,G);
x + y
-------------------------------
LCM(F,G);
1/4x^4 + 1/2x^3y - 1/2xy^3 - 1/4y^4
-------------------------------
4It = (x+y)^3(x-y);
TRUE
-------------------------------
GCD(3*4,3*8,6*16);
12
-------------------------------
GCD([3*4,3*8,6*16]);
12
-------------------------------
```

See Also: Div, Mod (VI-1.55 pg.170)

VI-1.97 GenericPoints

syntax

```
GenericPoints(NumPoints:INT):LIST
GenericPoints(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 ::= Q[x,y];GenericPoints(7);
[[1, 0], [0, 1], [1, 1], [12, 59], [6, 63], [12, 80], [17, 63]]
-------------------------------
GenericPoints(7,500);
[[1, 0], [0, 1], [1, 1], [220, 162], [206, 452], [98, 106], [403, 449]]
-------------------------------
Use R ::= Z/(5)[x,y,z];
GenericPoints(7);
[[1, 0, 0], [0, 1, 0], [0, 0, 1], [1, 1, 1], [2, 1, 1], [2, 2, 4], [3, 1, 3]]
-------------------------------
GenericPoints(7,500);
[[1, 0, 0], [0, 1, 0], [0, 0, 1], [1, 1, 1], [1, 4, 2], [1, 3, 2], [2, 3, 3]]
-------------------------------
```

VI-1.98 GenRepr

syntax

```
GenRepr(X:POLY,I:IDEAL):LIST of POLY
GenRepr(X:VECTOR,I:MODULE):LIST of POLY
```

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,...,G_t]`”. If X is in I, then “`GenRepr`” will return a list “`[F_1,...,F_t]`” such that

```
X = F_1*G_1 + ... + F_t*G_t.
```

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

example

```
Use R ::= Q[xy];
I := Ideal(x+y^2,x^2-xy);
GenRepr(x^3-x^2y-y^3-xy,I);
[-y, x]
-------------------------------
-y I.Gens[1] + x I.Gens[2];
x^3 - x^2y - y^3 - xy
-------------------------------
GenRepr(x+y,I);
[ ]
-------------------------------
x+y IsIn I;  -- the empty list was returned above since x+y is not in I
FALSE
-------------------------------
V1:= Vector(x,y,y^2); V2:= Vector(x-y,0,x^2);
X := x^2 V1 - y^2 V2;
```

```
M := Module(V1,V2);
GenRepr(X,M);
[x^2, -y^2]
-------------------------------
```

See Also: DivAlg (VI-1.56 pg.171), IsIn (VI-1.146 pg.215), NF (VI-1.190 pg.237)

VI-1.99 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. Note that I.Gens and M.Gens will give the same lists of generators. The advantage of "`Gens`" is that its argument can be any expression evaluating to an ideal or module.

example
```
Use R ::= Q[x,y];
I := Ideal(y^2-x^3,xy);
Gens(I);
[-x^3 + y^2, xy]
-------------------------------
I.Gens;
[-x^3 + y^2, xy]
-------------------------------
Gens(I^2);
[x^6 - 2x^3y^2 + y^4, -x^4y + xy^3, x^2y^2]
-------------------------------
```

See Also: Minimalize, Minimalized (VI-1.175 pg.230)

VI-1.100 Get

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

Description

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);

Note: ‘‘\verb&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.
```

See Also: Introduction to IO (III-7.1 pg.57), OpenIFile, OpenOFile (VI-1.198 pg.240), OpenIString, OpenOString (VI-1.199 pg.241)

VI-1.101 GetErrMesg

syntax
```
GetErrMesg(E:ERROR):STRING
```

Description

This function returns the error message associated with an error.

example
```
Str := GetErrMesg(1/0);
PrintLn(Str);
Division by zero

-------------------------------
```

See Also: Catch (VI-1.19 pg.152), Error (VI-1.62 pg.174)

VI-1.102 Gin

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

Description

This function returns the [probabilistic] gin (generic initial ideal) of the ideal I. It is attained by computing the leading term ideal of g(I), where g is a random change of coordinates with integer coefficients in [-Range, Range], the default is [-100, 100]. This process is repeated until 4 consecutive change of coordinates give the same leading term ideal.

example
```
Use R ::= Q[x,y,z];  -- the default term-ordering is DegRevLex
Gin(Ideal(y^2-xz, x^2z-yz^2));
Ideal(x^2, xy^2, y^4)
-------------------------------
Use R ::= Q[x,y,z], Lex;
Gin(Ideal(y^2-xz, x^2z-yz^2), 20);
Ideal(x^2, xy^2, xyz^2, xz^4, y^6)
-------------------------------
```

VI-1.103 GlobalMemory

syntax
```
GlobalMemory():TAGGED(Memory)
```

Description

This function prints the contents of the global memory which are not bound to rings: variables prefixed by "`MEMORY`" but not by "`MEMORY.ENV`". Untagging the value returned by "`GlobalMemory`" gives a list of strings which are identifiers for the global variables. The command "`Fields(MEMORY)`" gives the same set of strings.

For more information about memory in CoCoA, see the chapter entitled "Memory Management" (III-8 pg.63).

example

```
Use R ::= Q[x,y,z];
A := 3;
ENV.R.B := 7;
MEMORY.C := 6;
GlobalMemory();
["C", "DEV", "ENV", "ERR", "PKG"]
-------------------------------
MEMORY.ENV;  -- the record holding the rings defined during the
             -- CoCoA session
Record[Q = Q, Qt = Q[t], R = Q[x,y,z], Z = Z]
-------------------------------
Memory();  -- the working memory
["A", "It"]
-------------------------------
Memory(R);  -- the global variables bound to the ring R
["B"]
-------------------------------
```

See Also: Memory (VI-1.173 pg.229), Memory Management (III-8 pg.63)

VI-1.104 H.Browse

syntax

```
H.Browse():NULL;
H.Browse(N:INT):NULL
```

Description

This function browses the online help system. Without an argument, it displays the next section of the online manual or the next command. With integer argument N, it skips ahead (or skips back, if N is negative) N sections of the manual or N commands.

See Also: H.Commands (VI-1.105 pg.194), ?, Man (VI-1.3 pg.142)

VI-1.105 H.Commands

syntax

```
H.Commands():NULL;
H.Commands(S:STRING):NULL
```

Description

This function prints a list of commands associated with the string S. For example, "`H.Commands("poly")`" will list all documented commands having to do with polynomials. Unlike "?", this function searches only the list of commands (not both the list of commands and online manual, both of which are a part of the online help system). Also, unlike "?", this function does not try to use the string S to identify a unique function. Instead it looks for all functions whose "*type*" is S, i.e., that are somehow related with the search string S. The types are often the names of data types in CoCoA. A complete list of these types, along with additional information, can be found by entering "`H.Commands()`", without any argument.

After a command name is found, complete information on the command can be obtained using "`?`". The function "`H.Syntax`" prints just the syntax for the command.

Note: entering "`H.Commands("")`" will produce a complete list of the documented commands.

See Also: ?, Man (VI-1.3 pg.142), H.Syntax (VI-1.110 pg.196)

VI-1.106 H.Man

syntax

```
H.Man():NULL
H.Man(S:STRING):NULL
H.Man(S:STRING,N:INT):NULL

where N = 0 or 1.
```

Description

This function is synonymous with "`Man`", which performs the same task as "`?`". See "?, Man" (VI-1.3 pg.142) for more information.

See Also: H.Commands (VI-1.105 pg.194), H.Syntax (VI-1.110 pg.196), ?, Man (VI-1.3 pg.142)

VI-1.107 H.OutCommands

syntax

```
H.OutCommands(S:STRING):NULL
H.OutCommands(S:STRING,A:INT):NULL
H.OutCommands(S:STRING,A:INT,B:INT):NULL
```

Description

The function prints the online descriptions of commands to the text file named S. Warning: if a file named S already exists, it is appended to. The first form prints all of the command descriptions, the second prints only the command with number A, and the last prints commands with numbers A to B. The total number of commands is given by Len(MEMORY.Doc.Commands). The name of the command with number I is MEMORY.Doc.Commands[I].Title. Entering "`H.Commands("")`" will list the documented commands, in order.

example

```
H.OutCommands("CommandFile",1,10);
```

To print sections of the online manual, use the function "`H.OutManual`".

See Also: H.OutManual (VI-1.108 pg.195)

VI-1.108 H.OutManual

syntax

```
H.OutManual(S:STRING):NULL
H.OutManual(S:STRING,P:INT):NULL
H.OutManual(S:STRING,P:INT,C:INT):NULL
H.OutManual(S:STRING,P:INT,C:INT,S:INT):NULL
```

Description

This function prints sections of the manual to a text file named S. Warning: if a file named S already exists, it is appended to. The first form prints the entire manual to a file. The others are used to print part P, chapter C, section S. Recall that the online help consists of a manual *and* a list of commands. To print out the commands, use "`H.OutCommands`".

example

```
H.OutManual("part1.chp2",1,2);
```

See Also: H.OutCommands (VI-1.107 pg.195), H.Toc (VI-1.112 pg.197)

VI-1.109 H.SetMore, H.UnSetMore

syntax

```
H.SetMore(N:INT):NULL
H.SetMore():NULL

H.UnSetMore():NULL
```

Description

The purpose of these functions is to turn on and off filtering of the online help system through the function "`More`". When the online help system filters through "`More`" any output from online help is stored in a "`MoreDevice`" then printed to the screen, N lines at a time. The number N is stored in the global variable MEMORY.MoreCount and may be set directly by the user with the command "`MEMORY.MoreCount := X`" where X is an integer. The idea is to keep the output from scrolling off of the screen. See "More" (VI-1.183 pg.233) for more information.

The function "`H.SetMore`" turns on filtering through "`More`"; and if the optional argument N is supplied, it sets MEMORY.MoreCount to N.

The function "`H.UnSetMore`" turns off filtering through "`More`", without affecting MEMORY.MoreCount.

See Also: More (VI-1.183 pg.233)

VI-1.110 H.Syntax

syntax

```
H.Syntax():NULL
H.Syntax(S:STRING):NULL
```

Description

The first form of the command, with no arguments, just prints this message. The second form looks for a command with associated keywords containing S as a substring. If S is exactly the keyword of a command or if S is the substring of a keyword of only one command, then the syntax for that command is displayed. (The command "H.Browse" (VI-1.104 pg.194) can then be called to display additional information.) Otherwise, H.Syntax(S) lists the names all commands with associated keywords containing S as a substring. Note: the search is case insensitive.

example

```
H.Syntax("dense");
DensePoly(N:INT):POLY

Description: the sum of all power-products of a given degree

--> "H.Browse();" for more information. <--

-------------------------------
```

VI-1.111 H.Tips

syntax

```
H.Tips():NULL
```

Description

This function prints advice on using CoCoA's online help system effectively.

example

```
H.Tips();
```

```
============ Quick Tips for Using Online Help =============

Here are some tips for using the online help system:

1. Searches are case insensitive and your search string need only be a
substring of a keyword to make a match.  Thus, for instance, to find
the section of the manual entitled "Commands and Functions for
Polynomials", it is enough to type: "?for poly".

             ---> Output suppressed <---
```

See Also: Online Help (V-2.1 pg.131)

VI-1.112 H.Toc

syntax

```
H.Toc():NULL;
H.Toc(P:INT):NULL
H.Toc(P:INT,C:INT):NULL
H.Toc(all):NULL
```

Description

The first form of this function, with no arguments, lists the titles of the parts and chapters of the manual. The second prints the table of contents for part P. The third prints the table of contents for part P, chapter C. The last form, with the string "*all*" as argument, prints the entire table of contents.

The contents of each section can be read online by giving enough of its title as an argument to "?".

See Also: ?, Man (VI-1.3 pg.142)

VI-1.113 H.Tutorial

syntax

```
H.Tutorial():NULL
```

Description

The CoCoA tutorial is part of the online manual. This function displays the first section of the tutorial. The following sections can then be browsed using "`H.Browse`".

See Also: H.Browse (VI-1.104 pg.194)

VI-1.114 Head

syntax

```
Head(L:LIST):OBJECT
```

Description

This function returns the first element of the list L.

example

```
Head([3,2,1]);
3
-------------------------------
```

See Also: First (VI-1.68 pg.177), Last (VI-1.156 pg.219), Tail (VI-1.265 pg.275)

VI-1.115 Help

syntax

```
Help(S:STRING):NULL
```

Description

This command is used for extending the online help to include information about user-defined functions. It is *not* the main command for getting information about CoCoA online. For information about online help in general, enter “`?`” or “`?online help`”.

There are two ways to let “`Help`” know about a help string associated with a user-defined function. First, one may use the line “`Help S`” where S is the help string, as the first line of the function definition.

example

```
Define AddThree(X)
  Help "adds 3 to its argument";
  Return X+3;
EndDefine;
Help("AddThree");
adds 3 to its argument
-------------------------------
F(0);
3
-------------------------------
```

The second way to provide a help string for “`Help`” is to define a function “`Help_F`” where F is the function identifier.

example

```
Define AddFive(X)
  Return X+5;
EndDefine;
Define Help_AddFive()
  Return "adds 5 to its argument";
EndDefine;
Help("AddFive");
adds 5 to its argument
-------------------------------
AddFive(0);
5
-------------------------------
```

See Also: Define (VI-1.41 pg.162), Online Help (V-2.1 pg.131)

VI-1.116 Hilbert

syntax

```
Hilbert(R:RING or TAGGED(Quotient)):TAGGED($cocoa/hp.Hilbert)
Hilbert(R:RING or TAGGED(Quotient),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`” instead of “`Hilbert(R,N)`” in order to speed up execution.

The coefficient ring must be a field.

example

```
Use R ::= Q[t,x,y,z];
Hilbert(R/Ideal(z^2-xy,xz^2+t^3));
H(0) = 1
H(1) = 4
H(t) = 6t-3   for t >= 2
-------------------------------
M := R^2/Module([x^2-t,xy-z^3],[zy,tz-x^3y+3]);
Hilbert(M);
H(0) = 2
H(1) = 8
H(2) = 20
H(3) = 39
H(t) = 3t^2 + 6t-7   for t >= 4
-------------------------------
Hilbert(M,3)
39
-------------------------------
Hilbert(M,5);
98
-------------------------------
```

See Also: EvalHilbertFn (VI-1.64 pg.175), HilbertPoly (VI-1.118 pg.199), HVector (VI-1.121 pg.201), Poincare, HilbertSeries (VI-1.209 pg.245)

VI-1.117 HilbertBasis

syntax

```
HilbertBasis(M:MAT): LIST

where M is a matrix over Z.
```

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]]);
HilbertBasis(M);
[[0, 3, 2, 0], [1, 4, 1, 1], [2, 5, 0, 2]]
-------------------------------
M * Transposed(Mat(It));
Mat([
  [0, 0, 0],
  [0, 0, 0]
])
-------------------------------
```

VI-1.118 HilbertPoly

syntax

```
Hilbert(R:RING or TAGGED(Quotient)):POLY in the ring Qt.
```

Description

This function returns the Hilbert polynomial for R as a polynomial in the standard CoCoA ring Qt (= Q[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” (VI-1.116 pg.198).

example

```
Use R ::= Q[w,x,y,z];
I := Ideal(z^2-xy,xz^2+w^3);
Hilbert(R/I);
H(0) = 1
H(1) = 4
H(t) = 6t-3   for t >= 2
-------------------------------
F := HilbertPoly(R/I);
F;  -- a polynomial in the ring Qt
Qt :: 6t-3
-------------------------------
Subst(F,Qt::t,3);
Qt :: 15
-------------------------------
```

See Also: EvalHilbertFn (VI-1.64 pg.175), Hilbert (VI-1.116 pg.198), HVector (VI-1.121 pg.201), Poincare, HilbertSeries (VI-1.209 pg.245)

VI-1.119 HIntersection, HIntersectionList

syntax

```
HIntersection(I_1:IDEAL,...,I_n:IDEAL):IDEAL
HIntersectionList(L:LIST of IDEAL):IDEAL
```

Description

The function “`HIntersection`” returns the intersection of “`I_1,...,I_n`” using a Hilbert-driven algorithm. It differs from “`Intersection`” only when the input is inhomogeneous, in which case, “`HIntersection`” may be faster.

The function “`HIntersectionList`” applies the function “`HIntersection`” to the elements of a list, i.e., “`HIntersectionList([I_1,...,I_n])`” is the same as “`HIntersection(I_1,...,I_n)`”.

The coefficient ring must be a field.

example

```
Use R ::= Q[x,y,z];
HIntersection(Ideal(x-z,y-2z),Ideal(x-2z,y-z));
Ideal(x + y - 3z, y^2 - 3yz + 2z^2)
-------------------------------
L := [Ideal(x-z,y-2z),Ideal(x-2z,y-z)];
HIntersectionList(L);
Ideal(x + y - 3z, y^2 - 3yz + 2z^2)
-------------------------------
```

See Also: Intersection, IntersectionList (VI-1.140 pg.212)

VI-1.120 Homogenized

syntax

```
Homogenized(X:INDET,E:T):T

where T is of type IDEAL or POLY, or T is a LIST recursively
constructed of types IDEAL, POLY, and LIST.
```

Description

This function returns the homogenization of E with respect to the indeterminate X, which must have weight 1. Note that in the case where E is an ideal, "`Homogenized`" returns the ideal generated by the homogenizations of all the elements of E, not just the homogenization of the generators of E (see the example, below). The coefficient ring must be a field for this function to work reliably.

example

```
Use R ::= Q[x,y,z,w];
Homogenized(w,x^3-y);
x^3 - yw^2
-------------------------------
Homogenized(w,[x^3-y,x^4-z]);
[x^3 - yw^2, x^4 - zw^3]
-------------------------------
I := Ideal(x^3-y,x^4-z);
Homogenized(w,I);  -- don't just get the homogenizations of
                   -- the generators of I
Ideal(x^3 - yw^2, -xy + zw, x^2z - y^2w, y^3 - xz^2)
-------------------------------
Homogenized(w,[[I,y-z^2],z-y^4]);
[[Ideal(x^3 - yw^2, -xy + zw, x^2z - y^2w, y^3 - xz^2), -z^2 + yw], -y^4 + zw^3]
-------------------------------
```

VI-1.121 HVector

syntax

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

Description

This function returns the h-vector of the ring R, i.e., the coefficients of the numerator of the simplified Poincare series for R.

The weights of the indeterminates of the current ring 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 ::= Q[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 (VI-1.116 pg.198), Poincare, HilbertSeries (VI-1.209 pg.245)

VI-1.122 Ideal

syntax

```
Ideal(P_1:POLY,...,P_n:POLY):IDEAL
Ideal(L:LIST):IDEAL
Ideal(M:MODULE):IDEAL

where L is a list of polynomials and M is contained in a free module
of rank 1.
```

Description

The first form returns the ideal generated by "`P_1,...P_n`". The second form returns the ideal generated by the polynomials in L. The third form returns the ideal generated by the polynomials in M; it is the same as "`Cast(M,IDEAL)`", and requires that the module be a submodule of the free module of rank 1.

example

```
Use R ::= Q[x,y,z];
I := Ideal(x-y^2,xy-z);
I;
Ideal(-y^2 + x, xy - z)
-------------------------------
L := [xy-z,x-y^2];
J := Ideal(L);
I = J;
TRUE
-------------------------------
M := Module([y^3-z],[x-y^2]);
Ideal(M) = I;
TRUE
-------------------------------
```

VI-1.123 IdealAndSeparatorsOfPoints

syntax

```
IdealAndSeparatorsOfPoints(Points:LIST):RECORD

where Points is a list of lists of coefficients representing a set of
*distinct* points in affine space.
```

Description

This function computes the results of "`IdealOfPoints`" and "`SeparatorsOfPoints`" 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; and 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`".

example

```
Use R ::= Q[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, HGBM (VI-1.95 pg.189), GenericPoints (VI-1.97 pg.191), IdealAndSeparatorsOfProjectivePoints (VI-1.124 pg.203), IdealOfPoints (VI-1.125 pg.204), IdealOfProjectivePoints (VI-1.126 pg.204), Interpolate (VI-1.138 pg.210), QuotientBasis (VI-1.216 pg.249), SeparatorsOfPoints (VI-1.241 pg.263), SeparatorsOfProjectivePoints (VI-1.242 pg.263)

VI-1.124 IdealAndSeparatorsOfProjectivePoints

syntax

```
IdealAndSeparatorsOfProjectivePoints(Points:LIST):RECORD

where Points is a list of lists of coefficients representing a set of
*distinct* points in projective space.
```

Description

This function computes the results of “`IdealOfProjectivePoints`” and “`SeparatorsOfProjectivePoints`” 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; and 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`”.

example

```
Use R ::= Q[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]
-------------------------------
```

See Also: GBM, HGBM (VI-1.95 pg.189), GenericPoints (VI-1.97 pg.191), IdealAndSeparatorsOfPoints (VI-1.123 pg.202), IdealOfPoints (VI-1.125 pg.204), IdealOfProjectivePoints (VI-1.126 pg.204), Interpolate (VI-1.138 pg.210), QuotientBasis (VI-1.216 pg.249), SeparatorsOfPoints (VI-1.241 pg.263), SeparatorsOfProjectivePoints (VI-1.242 pg.263)

VI-1.125 IdealOfPoints

syntax

```
IdealOfPoints(Points:LIST):IDEAL

where Points is a list of lists of coefficients representing a set of
*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; * 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:=IdealOfPoints(Pts); Foreach Element In Gens(I) Do PrintLn Element; EndForeach;

For ideals of points in projective space, see "`IdealOfProjectivePoints`".

example

```
Use R ::= Q[x,y];
Points := [[1, 2], [3, 4], [5, 6]];
I := IdealOfPoints(Points);
I;
Ideal(x - y + 1, y^3 - 12y^2 + 44y - 48)
-------------------------------
I.Gens;  -- the reduced Groebner basis
[x - y + 1, y^3 - 12y^2 + 44y - 48]
-------------------------------
```

See Also: GBM, HGBM (VI-1.95 pg.189), GenericPoints (VI-1.97 pg.191), IdealAndSeparatorsOfPoints (VI-1.123 pg.202), IdealAndSeparatorsOfProjectivePoints (VI-1.124 pg.203), IdealOfProjectivePoints (VI-1.126 pg.204), Interpolate (VI-1.138 pg.210), QuotientBasis (VI-1.216 pg.249), SeparatorsOfPoints (VI-1.241 pg.263), SeparatorsOfProjectivePoints (VI-1.242 pg.263)

VI-1.126 IdealOfProjectivePoints

syntax

```
IdealOfProjectivePoints(Points:LIST):IDEAL

where Points is a list of lists of coefficients representing a set of
*distinct* points in projective space.
```

Description

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

NOTE: * the current ring must have at least one more indeterminate than the dimension of the projective space in which the points lie, i.e, at least as many indeterminates as 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" (VI-1.125 pg.204).

example

```
Use R ::= Q[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, HGBM (VI-1.95 pg.189), GenericPoints (VI-1.97 pg.191), IdealAndSeparatorsOfPoints (VI-1.123 pg.202), IdealAndSeparatorsOfProjectivePoints (VI-1.124 pg.203), IdealOfPoints (VI-1.125 pg.204), Interpolate (VI-1.138 pg.210), QuotientBasis (VI-1.216 pg.249), SeparatorsOfPoints (VI-1.241 pg.263), SeparatorsOfProjectivePoints (VI-1.242 pg.263)

VI-1.127 Identity

syntax

```
Identity(N:INT):MAT
```

Description

This function returns the NxN identity matrix.

example

```
Identity(3);
Mat[
  [1, 0, 0],
  [0, 1, 0],
  [0, 0, 1]
]
-------------------------------
```

VI-1.128 If

syntax

```
If B Then C EndIf
If B_1 Then C_1 Else C_2 EndIf
If B_1 Then C_1 Elsif B_2 Then C_2 Elsif ... EndIf
If B_1 Then C_1 Elsif B_2 Then C_2 Elsif ... Else C_r EndIf

D If B

where the B's are boolean expressions, the C's are command
sequences, and D is a single command.
```

Description

If "`B_n`" is the first in the sequence of "`B_i`"'s to evaluate to TRUE, then "`C_n`" is executed. If none of the "`B_i`"'s evaluates to TRUE, nothing is done. The construct, "`Elsif B Then C`" can be repeated any number of times. Note: be careful not to type "`Elseif`" by mistake (it has an extraneous "`e`").

In the last form, the single command D is performed if B evaluates to TRUE. NOTE: the use of this form is discouraged. It will probably disappear from future versions of CoCoA.

For a conditional "`expression`", assignable to a variable, see "`Cond`".

example

```
Define Sign(A)
  If A > 0 Then Return 1
  Elsif A = 0 Then Return 0
  Else Return -1
  EndIf
End;

Sign(3);
1
-------------------------------
```

See Also: Cond (VI-1.34 pg.159)

VI-1.129 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
-------------------------------
```

VI-1.130 Image

syntax

```
Image(R::E:OBJECT,F:TAGGED(RMap)):OBJECT
Image(V:OBJECT,F:TAGGED(RMap)):OBJECT

where R is the identifier for a ring and F has the form
RMap(F_1:POLY,...,F_n:POLY) or the form RMap([F_1:POLY,...,F_n:POLY]).
The number n is the number of indeterminates of the ring R. In the
second form, V is a variable containing a CoCoA object dependent on R
or not dependent on any ring.
```

Description

This function maps the object E from one ring to another as determined by F. Suppose the current ring is S, and E is an object dependent on a ring R; then

```
Image(R::E,F)
```

returns the object in S obtained by substituting “`F_i`” for the i-th indeterminate of R in E. Effectively, we get the image of E under the ring homomorphism,

```
F: R   --->  S
   x_i |--> F_i,
```

where “`x_i`” denotes the i-th indeterminate of R.

Notes: 1. The coefficient rings for the domain and codomain must be the same. 2. If R = S, one may use “`Image(E,F)`” but in this case it may be easier to use “`Eval`” or “`Subst`”. 3. The exact domain is never specified by the mapping F. It is only necessary that the domain have the same number of indeterminates as F has components. Thus, we are abusing terminology somewhat in calling F a map. 4. The second form of the function does not require the prefix "`R::`" since the prefix is associated automatically. 5. If the object E in R is a polynomial or rational function (or list, matrix, or vector of these) which involves only indeterminates that are already in S, the object E can be mapped over to S without change using the command "BringIn" (VI-1.16 pg.150).

example

```
Use C ::= Q[u,v];   -- domain
Use B ::= Q[x,y];   -- another possible domain
I := Ideal(x^2-y);  -- an ideal in B
Use A ::= Q[a,b,c]; -- codomain
F := RMap(a,c^2-ab);
Image(B::xy, F);    -- the image of xy under F:B --> A
-a^2b + ac^2
-------------------------------
Image(C::uv,F);     -- the image of uv under F:C --> A
-a^2b + ac^2
-------------------------------
Image(I,F);         -- the image of the ideal I under F: B --> A
Ideal(a^2 + ab - c^2)
-------------------------------
I;  -- the prefix "B::" was not needed in the previous example since
    -- I is already labeled by B
B :: Ideal(x^2 - y)
-------------------------------
Image(B::Module([x+y,xy^2],[x,y]),F); -- the image of a module
Module([-ab + c^2 + a, a^3b^2 - 2a^2bc^2 + ac^4], [a, -ab + c^2])
-------------------------------
X := C:: u+v;  -- X is a variable in the current ring (the codomain), A,
X;             -- whose value is an expression in the ring C.
C :: u + v
-------------------------------
Image(X,F);    -- map X to get a value in C
-ab + c^2 + a
-------------------------------
```

See Also: Accessing Other Rings (IV-8.11 pg.100), BringIn (VI-1.16 pg.150), QZP, ZPQ (VI-1.217 pg.250), Ring Mappings: the Image Function (IV-8.12 pg.101), Subst (VI-1.257 pg.271), Using (VI-1.275 pg.280)

VI-1.131 In

syntax

```
[E:OBJECT | X In L:LIST And B:BOOL]:LIST
[X In L:LIST | B:BOOL]:LIST
[E:OBJECT | X In L]

where X is a variable identifier which may occur in B or E.
```

Description

In the first form, E is an arbitrary CoCoA expression and B is a boolean expression, both of which are functions of the variable X. Write E(X) for E and B(X) for B. The first listed command then returns the list of all E(X) such that X is in the list L and B(X) evaluates to TRUE.

example

```
[[X^2,X^3] | X In [-2,-1,0,1,2] And X <> 0];
[[4, -8], [1, -1], [1, 1], [4, 8]]
-------------------------------
```

```
[X In [1,2] >< [2,3,4] | X[1]+X[2]=4];
[[1, 3], [2, 2]]
-------------------------------
```

(Note: the ¿¡ operator is used to form Cartesian products; it is not the same as the "*not equal*" operator, ¡¿.)

The second form of the command is the same as the first with E = X.

example

```
[X In [1,2,3] | X > 1];
[2, 3]
-------------------------------
```

The third form is the same as the first with B = TRUE.

example

```
[X^2 | X In [1,2,3]];
[1, 4, 9]
-------------------------------
```

See Also: IsIn (VI-1.146 pg.215), NewList (VI-1.186 pg.235), Comp (VI-1.31 pg.158)

VI-1.132 Indet

syntax

```
Indet(N:INT):POLY
```

Description

This function returns the N-th indeterminate of the current ring.

example

```
Use R ::= Q[x,y,z];
Indet(2);
y
-------------------------------
```

See Also: IndetInd (VI-1.133 pg.208), IndetIndex (VI-1.134 pg.209), IndetName (VI-1.135 pg.209), Indets (VI-1.136 pg.209), NumIndets (VI-1.197 pg.240)

VI-1.133 IndetInd

syntax

```
IndetInd(X:INDET):LIST
```

Description

This function returns the index of the indeterminate X.

example

```
Use R ::= Q[x[1..3,1..2],y,z];
IndetInd(x[3,2]);
[3, 2]
-------------------------------
IndetInd(y);
[ ]
-------------------------------
```

See Also: Indet (VI-1.132 pg.208), IndetIndex (VI-1.134 pg.209), IndetName (VI-1.135 pg.209), Indets (VI-1.136 pg.209), NumIndets (VI-1.197 pg.240)

VI-1.134 IndetIndex

syntax

```
IndetIndex(X:INDET):INT
```

Description

This function returns the index of the named determinate. The index is determined by the order in which the indeterminate is listed when the corresponding ring was created.

example

```
Use R ::= Q[x,y,z]
IndetIndex(y);
2
-------------------------------
Use R ::= Q[x[1..2,1..2],y[1..2]];
Indets();
[x[1,1], x[1,2], x[2,1], x[2,2], y[1], y[2]]
-------------------------------
IndetIndex(x[2,1]);
3
-------------------------------
S ::= Q[a,b,c];
IndetIndex(S::b);
2
-------------------------------
```

See Also: Indet (VI-1.132 pg.208), IndetInd (VI-1.133 pg.208), IndetName (VI-1.135 pg.209), Indets (VI-1.136 pg.209), NumIndets (VI-1.197 pg.240)

VI-1.135 IndetName

syntax

```
IndetName(X:INDET):STRING
```

Description

This function returns the name of the indeterminate X as a string.

example

```
Use R ::= Q[x,y,z];
IndetName(Indet(2));
y
-------------------------------
Type(It);
STRING
-------------------------------
```

See Also: Indet (VI-1.132 pg.208), IndetInd (VI-1.133 pg.208), IndetIndex (VI-1.134 pg.209), NumIndets (VI-1.197 pg.240)

VI-1.136 Indets

syntax

```
Indets():LIST
```

Description

This function returns the list of indeterminates of the current ring.

example
```
Use R ::= Q[x,y,z];
Indets();
[x, y, z]
-------------------------------
```

See Also: Indet (VI-1.132 pg.208), IndetInd (VI-1.133 pg.208), IndetIndex (VI-1.134 pg.209), IndetName (VI-1.135 pg.209), NumIndets (VI-1.197 pg.240)

VI-1.137 Insert, Remove

syntax
```
Insert(V:LIST,N:INT,E:OBJECT):NULL
Remove(V:LIST,N:INT):NULL

where V is a variable containing a list.
```

Description

The first function inserts E into the list L as the N-th component.

example
```
L := ["a","b","d","e"];
Insert(L,3,"c");
L;
["a", "b", "c", "d", "e"]
-------------------------------
```

The second function removes the N-th component from L. (The function “`WithoutNth`” returns the list obtained by removing the N-th component of L without affecting L, itself.)

example
```
Use R ::= Q[x,y,z];
L := Indets();
L;
[x, y, z]
-------------------------------
Remove(L,2);
L;
[x, z]
-------------------------------
```

See Also: Append (VI-1.9 pg.146), WithoutNth (VI-1.281 pg.283)

VI-1.138 Interpolate

syntax
```
Interpolate(Points:LIST,Values:LIST):POLY

where Points is a list of lists of coefficients representing a set of
*distinct* points and Values is a list of the same size containing
numbers from the coefficient ring.
```

Description

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 Q[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
-------------------------------
```

VI-1.139 Interreduce, Interreduced

syntax

```
Interreduce(V:LIST of POLY):NULL
Interreduce(V:LIST of VECTOR):NULL

Interreduced(L:LIST of POLY):LIST of POLY
Interreduced(L:LIST of VECTOR):LIST of VECTOR

where V is a variable containing a list.
```

Description

These functions reduce each polynomial (resp., vector) using the other polynomials (resp., vectors) 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.

Note that the definition of normal form depends on the current value of the option FullRed of the panel GROEBNER. If FullRed is FALSE it means that a polynomial (resp., vector) is in normal form when its leading term with respect to the current term ordering cannot be reduced. If FullRed is TRUE it means that a polynomial (resp., vector) is in normal form if and only if each monomial cannot be reduced.

example

```
UnSet FullRed;  -- FullRed = FALSE
Use R ::= Q[x,y,z];
Interreduced([x^3-xy^2+yz,xy,z]);
[x^3 - xy^2 + yz, xy, z]
-------------------------------
Set FullRed;  -- FullRed = TRUE (the default value)
Interreduced([x^3-xy^2+yz,xy,z]);
[xy, z, x^3]
-------------------------------
```

```
L := [x^3-xy^2+yz,xy,z];
Interreduce(L);
L;
[xy, z, x^3]
-------------------------------
```

See Also: FullRed (V-1.13 pg.129)

VI-1.140 Intersection, IntersectionList

syntax
```
Intersection(E_1:IDEAL,...,E_n:IDEAL):IDEAL
Intersection(E_1:LIST,....,E_n:LIST):LIST
Intersection(E_1:MODULE,....,E_n:MODULE):MODULE

IntersectionList(L:LIST):OBJECT
```

Description

The function "`Intersection`" returns the intersection of "`E_1,...,E_n`". In the case where the "`E_i`"'s are lists, it returns the elements common to all of the lists.

The function "`IntersectionList`" applies the function "`Intersection`" to the elements of a list, i.e., "`IntersectionList([X_1,...,X_n])`" is the same as "`Intersection(X_1,...,X_n)`".

The coefficient ring must be a field.

NOTE: In order to compute the intersection of inhomogeneous ideals, it may be faster to use the function "`HIntersection`". To compute the intersection of ideals corresponding to zero-dimensional schemes, see the commands "`GBM`" and "`HGBM`".

example
```
Use R ::= Q[x,y,z];
Points := [[0,0],[1,0],[0,1],[1,1]]; -- a list of points in the plane
I := Ideal(x,y); -- the ideal for the first point
Foreach P In Points Do
  I := Intersection(I,Ideal(x-P[1]z,y-P[2]z));
EndForeach;
I;  -- the ideal for (the projective closure of) Points
Ideal(y^2 - yz, x^2 - xz)
-------------------------------
Intersection(["a","b","c"],["b","c","d"]);
["b", "c"]
-------------------------------
IntersectionList([Ideal(x,y),Ideal(y^2,z)]);
Ideal(yz, xz, y^2)
-------------------------------
It = Intersection(Ideal(x,y),Ideal(y^2,z));
TRUE
-------------------------------
```

See Also: GBM, HGBM (VI-1.95 pg.189), HIntersection, HIntersectionList (VI-1.119 pg.200)

VI-1.141 Inverse

syntax
```
Inverse(X:OBJECT):OBJECT
```

Description

This function computes the multiplicative inverse of its argument. It is included for use when writing Inverse(X) comes more naturally than writing "`X^(-1)`", though both notations are functionally equivalent.

example
```
Inverse(Mat([[1,2], [3,4]]));
Mat[
  [-2, 1],
  [3/2, -1/2]
]
-------------------------------
```

VI-1.142 IO.SprintTrunc

syntax
```
IO.SprintTrunc(E:OBJECT,N:INT):STRING
```

Description

This function works like "`Sprint`", 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 ::= Q[xy];
I := Ideal(x,y);
IO.SprintTrunc(I,4);
Idea...
-------------------------------
```

See Also: Format (VI-1.75 pg.181), Sprint (VI-1.252 pg.269)

VI-1.143 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 (VI-1.129 pg.206)

VI-1.144 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 (VI-1.154 pg.218)

VI-1.145 IsHomog

syntax

```
IsHomog(F:POLY or VECTOR):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.

example

```
Use R ::= Q[x,y];
IsHomog(x^2-xy);
TRUE
-------------------------------
IsHomog(x-y^2);
FALSE
-------------------------------
IsHomog([x^2-xy,x-y^2]);
FALSE
------------------------------
Use R ::= Q[x,y],Weights(Mat([[2,3],[1,2]]));
IsHomog(x^3y^2+y^4);
TRUE
-------------------------------
Use R ::= Q[x,y];
IsHomog(Ideal(x^2+y,y));
TRUE
-------------------------------
```

See Also: Deg (VI-1.43 pg.164), MDeg (VI-1.172 pg.228), Weights Modifier (IV-8.5 pg.97)

VI-1.146 IsIn

syntax

```
E IsIn F

where E and F are CoCoA objects.  For a precise description of
allowable objects, see the full online help entry.
```

Description

The semantics of IsIn is explained in the following table:

```
 ----------------------------------------------------------------
| POLY   IsIn  IDEAL   checks for ideal membership.              |
| VECTOR IsIn  MODULE  checks for module membership.             |
| OBJECT IsIn  LIST    checks if the list contains the object.   |
| STRING IsIn  STRING  checks if the first string is a substring |
|                      of the second one.                        |
 ----------------------------------------------------------------
                   IsIn operator
```

VI-1.147 IsNumber

syntax

```
IsNumber(E:OBJECT):BOOL
```

Description

This function returns TRUE if E has type INT, RAT, or ZMOD. Otherwise, it returns FALSE.

example

```
Use R ::= Q[x,y];
IsNumber(x+y);
FALSE
-------------------------------
IsNumber(3);
TRUE
-------------------------------
IsNumber(3%5);
TRUE
-------------------------------
```

VI-1.148 IsPosTo, IsToPos

syntax

```
IsPosTo(R:RING):BOOL
IsToPos(R:RING):BOOL

where R is an identifier for a ring.
```

Description

The first function determines whether the ring R has the PosTo module term-ordering. The second function determines whether R has the ToPos module term-ordering. (These are the only possible module term-orderings in CoCoA.)

example

```
Use R ::= Q[x,y,z];
S ::= Q[x,y],PosTo;
```

```
IsPosTo(R);
FALSE
-------------------------------
IsToPos(R);
TRUE
-------------------------------
IsPosTo(S);
TRUE
-------------------------------
IsToPos(Var(RingEnv()));  -- to check the current ring, R
TRUE
-------------------------------
```

VI-1.149 IsPrime

syntax

```
IsPrime(N:INT):BOOL
```

Description

This function determines whether a small integer is prime. The range of permitted values for N is the same the range of permitted values for the NextPrime function: on most platforms $N \leq 45000$ should work fine, on some platforms considerably larger values will work. For N greater than the limit an error is raised.

example

```
IsPrime(32003);
TRUE
-------------------------------
IsPrime(10^100);
ERROR: IsPrime: number too large
CONTEXT: Return(Error("IsPrime: number too large"))
-------------------------------
```

See Also: NextPrime (VI-1.189 pg.236)

VI-1.150 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: Expected non-negative INT
CONTEXT: Isqrt(-1)
-------------------------------
```

VI-1.151 IsStable, IsStronglyStable, IsLexSegment

syntax

```
IsStable(I: MONOMIAL IDEAL): BOOL
IsStronglyStable(I: MONOMIAL IDEAL): BOOL
IsLexSegment(I: MONOMIAL IDEAL): BOOL
```

Description

These functions return whether the monomial ideal I is stable, strongly stable (Borel-fixed in characteristic 0), or a lex-segment ideal.

example

```
Use R ::= Q[x,y,z];
I := Ideal(xy^3, y^4, x^3, x^2y, x^2z);
IsStable(I);
TRUE
-------------------------------
IsStronglyStable(I);
TRUE
-------------------------------
IsLexSegment(I);
FALSE
-------------------------------
```

VI-1.152 IsSubset

syntax

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

Description

This function returns TRUE is Set(L) is contained in Set(M); otherwise it returns FALSE.

NOTE: The obsolete function SubSet used to have the same behaviour.

example

```
IsSubset([1,1,2],[1,2,3,"a"]);
TRUE
-------------------------------
IsSubset([1,2],["a","b"]);
FALSE
-------------------------------
IsSubset([],[1,2]);
TRUE
-------------------------------
```

See Also: EqSet (VI-1.59 pg.173), Set (VI-1.243 pg.264), Subsets (VI-1.256 pg.271)

VI-1.153 IsTerm

syntax

```
IsTerm(X:POLY or VECTOR):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, xy^2z is a term, while $4xy^2z$ and $xy+2z^3$ are not. For a vector, a term is a power-product times a standard basis vector, e.g., $(0, xy^2z, 0)$.

example

```
Use R ::= Q[x,y,z];
IsTerm(x+y^2);
FALSE
-------------------------------
IsTerm(x^3yz^2);
TRUE
-------------------------------
IsTerm(5x^3yz^2);
FALSE
-------------------------------
IsTerm(Vector(0,0,xyz));
TRUE
-------------------------------
IsTerm(Vector(x^2,y^2));
FALSE
-------------------------------
IsTerm(5x^2);
FALSE
-------------------------------
```

VI-1.154 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 ::= Q[x,y];
IsZero(x^2+3y-1);
FALSE
-------------------------------
IsZero(Ideal(x^2,xy^3));
FALSE
-------------------------------
IsZero(Vector(0,0,0));
TRUE
-------------------------------
```

See Also: IsEven, IsOdd (VI-1.144 pg.214)

VI-1.155 Jacobian

syntax

```
Jacobian(L:LIST):MAT

where L is a list of polynomials.
```

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 ::= Q[x,y];
L := [x-y,x^2-y,x^3-y^2];
Jacobian(L);
Mat[
  [1, -1],
  [2x, -1],
  [3x^2, -2y]
]
-------------------------------
```

VI-1.156 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.

example
```
L := [1,2,3,4,5];
Last(L);
5
-------------------------------
Last(L,3);
[3, 4, 5]
-------------------------------
```

See Also: First (VI-1.68 pg.177), Head (VI-1.114 pg.197), Tail (VI-1.265 pg.275)

VI-1.157 Latex

syntax
```
Latex(X:OBJECT):TAGGED($cocoa/latex.Latex)
```

Description

This function returns its argument in a format suitable for inclusion in a LaTeX document.

example
```
Use R ::= Q[x,y,z];
F := x^3+2y^2z;
Latex(F);
x^{3} + 2y^{2}z
-------------------------------
M := Mat([[1,2],[3,4]]);
Latex(M);
 \left( \begin{array}{ll}
1 & 2 \\
3 & 4 \end{array}\right)

-------------------------------
```

```
F := (x+y)/(1-z)^3;
Latex(F);
\frac{ - x - y}{z^{3}-3z^{2} + 3z-1}
-------------------------------
Latex(Ideal(x^2,y+z));
( \ x^{2},
y + z\ )
-------------------------------
```

See Also: Format (VI-1.75 pg.181), Sprint (VI-1.252 pg.269)

VI-1.158 LC

syntax

```
LC(F:POLY or VECTOR):C

where C is one of INT, RAT, or ZMOD.
```

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 ::= Q[x,y];
LC(x+3x^2-5y^2);
3
-------------------------------
LC(Vector(0,5y+6x^2,y^2));
6
-------------------------------
```

See Also: Coefficients (VI-1.27 pg.156), CoeffOfTerm (VI-1.28 pg.156), LT (VI-1.168 pg.225)

VI-1.159 Len

syntax

```
Len(E:OBJECT):INT
```

Description

This function returns the “*length*” of an object, as summarized in the table below:

```
 ----------------------------------------------
| type   | length                              |
|----------------------------------------------|
| IDEAL  | length of Gens(E)                   |
| INT    | 1                                   |
| LIST   | number of items in the list         |
| MAT    | number of rows of the matrix        |
| MODULE | length of Gens(E)                   |
| POLY   | number of monomials                 |
| RATFUN | if E=F/G, then Len(E)=Len(F)+Len(G) |
| VECTOR | number of components                |
 ----------------------------------------------
         The operator ``\verb&Len&''
```

example

```
Use R ::= Q[x,y];
L := ["a",2,3,[4,5]];
Len(L);
4
-------------------------------
Len(L[1]);
1
-------------------------------
Len(L[4]);
2
-------------------------------
Len(x^2 + xy + y^2 + xy^4);
4
-------------------------------
Len((x^2+y^2)/(x+y+y^2));
5
-------------------------------
Len((x+y) - (xy/x));
1
-------------------------------
Len(Ideal(x,x^2));
2
-------------------------------
```

The function “`Size`” returns the amount of memory used by the object.

See Also: Count (VI-1.36 pg.160), Len (VI-1.159 pg.220), Size (VI-1.246 pg.266)

VI-1.160 LinKer

syntax

```
LinKer(M:MAT):LIST
LinKerModP(M:MAT):LIST

where M is a matrix over Q or Z.
```

Description

The first function returns a list whose components are lists representing a Z-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.

The second 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]]);
LinKer(M);
[[1, -1, -1, 1], [0, 1, -2, 1]]
-------------------------------
M*Transposed(Mat(It));
Mat[
  [0, 0],
  [0, 0],
  [0, 0]
]
-------------------------------
Use Z/(3)[x];
LinKerModP(M);
```

```
[[1, 1, 1, 0], [0, -1, -1, -1]]
-------------------------------
M*Transposed(Mat(It));

Mat([
  [0, 0],
  [0, 0],
  [0, 0]
])
-------------------------------
```

See Also: LinSol (VI-1.161 pg.222)

VI-1.161 LinSol

syntax

```
LinSol(M:MAT,L:LIST):LIST

where M is an m x n matrix over Z or Q, and L is a list of length m
with entries in Z or Q (or a list of such lists).
```

Description

This function finds a solution to the inhomogeneous system of equations represented by M and L. Specifically, it returns a list, X, of length n with entries in Z or Q, such that M*Transposed(Mat(X)) = Transposed(Mat([L])), if such X exists; otherwise, it returns the empty list. Once a solution is found, all solutions may be found using the function "`LinKer`".

NOTE: "`LinSol`" can solve several inhomogeneous systems at once. If L has the form $[L_1, ..., L_k]$ where each L_i is a list of length m with entries in Z or Q, then LinSol(M,L) returns a list $[X_1, ..., X_k]$ where X_i is a solution to the system of linear equations represented by M and L_i.

example

```
M := Mat([[3,1,4],[1,5,9],[2,6,5]]);
L := [123,456,789];
LinSol(M,L);
[199/5, 742/5, -181/5]
-------------------------------
M*Transposed(Mat([It]));
Mat[
  [123],
  [456],
  [789]
]
-------------------------------
LinSol(M,[L,[1,2,3]]);
[[199/5, 742/5, -181/5], [4/15, 7/15, -1/15]]
-------------------------------
```

See Also: LinKer (VI-1.160 pg.221)

VI-1.162 List

syntax

```
List(E:OBJECT):LIST

where E has type LIST, MAT, or VECTOR.
```

Description

This function converts the expression E into a list. It is the same as Cast(E,LIST).

example

```
Use R ::= Q[x,y];
M:=Jacobian([x^2y^2,x+y]);
M;
Mat[
  [2xy^2, 2x^2y],
  [1, 1]
]
-------------------------------
Head(M);

-------------------------------
ERROR: Bad parameters
CONTEXT: Head(M)
-------------------------------
Head(List(M));
[2xy^2, 2x^2y]
-------------------------------

The error occurs because the function ‘‘\verb&Head&’’ only accepts lists as
arguments.
```

See Also: NewList (VI-1.186 pg.235)

VI-1.163 LM

syntax

```
LM(P:POLY or VECTOR):same type as P
```

Description

This function returns the leading monomial of P. The monomial includes the coefficient. To get the leading term of P, (which does not included the coefficient), use “LT”.

example

```
Use R ::= Q[x,y];
LM(3x^2y+y);
3x^2y
-------------------------------
LM(Vector(2x,y));
Vector(2x, 0)
-------------------------------
LT(Vector(2x,y));
Vector(x, 0)
-------------------------------
```

See Also: LC (VI-1.158 pg.220), LPP (VI-1.167 pg.225), LT (VI-1.168 pg.225)

VI-1.164 Log

syntax

```
Log(F:POLY):LIST
```

Description

This function returns the list of exponents of the leading term of F.

example
```
Use R ::= Q[x,y,z];
F := x^3y^2z^5+x^2y+xz^4;
Log(F);
[3, 2, 5]
-------------------------------
```

See Also: LT (VI-1.168 pg.225), LogToTerm (VI-1.165 pg.224)

VI-1.165 LogToTerm

syntax
```
LogToTerm(L:LIST):POLY

where L is a list of integers.
```

Description

This function returns the power-product whose list of exponents is L.

example
```
Use R ::= Q[x,y,z];
LogToTerm([2,3,5]);
x^2y^3z^5
-------------------------------
Log(It);
[2, 3, 5]
-------------------------------
```

See Also: Log (VI-1.164 pg.223)

VI-1.166 LPos

syntax
```
LPos(V:VECTOR):INT
```

Description

This function returns the position of the leading power-product of V.

example
```
Use R ::= Q[x,y],ToPos;  -- ToPos is the default module term-ordering
LT(Vector(x,y^2));
Vector(0, y^2)
-------------------------------
LPP(Vector(x,y^2));
y^2
-------------------------------
LPos(Vector(x,y^2));
2
-------------------------------
Use R ::= Q[x,y],PosTo;
LT(Vector(x,y^2));
Vector(x, 0)
-------------------------------
```

```
LPP(Vector(x,y^2));
x
-------------------------------
LPos(Vector(x,y^2));
1
-------------------------------
```

See Also: LM (VI-1.163 pg.223), LPP (VI-1.167 pg.225), LT (VI-1.168 pg.225)

VI-1.167 LPP

syntax

```
LPP(P:POLY or VECTOR):POLY
```

Description

This function returns the leading power-product of P.

example

```
Use R ::= Q[x,y];
LPP(3x^2y+y);  -- LPP is the same as LT for polynomials
x^2y
-------------------------------
LPP(Vector(2x,y));
x
-------------------------------
LT(Vector(2x,y));  -- Note the difference between LPP and LT
                   -- for vectors.
Vector(x, 0)
-------------------------------
```

See Also: LC (VI-1.158 pg.220), LM (VI-1.163 pg.223), LT (VI-1.168 pg.225)

VI-1.168 LT

syntax

```
LT(E):same type as E

where E has type IDEAL, MODULE, POLY, or VECTOR.
```

Description

If E is a polynomial this function returns the leading term of the polynomial E with respect to the term-ordering of the current ring. For the leading monomial, which includes the coefficient, use “`LM`”.

example

```
Use R ::= Q[x,y,z];  -- the default term-ordering is DegRevLex
LT(y^2-xz);
y^2
-------------------------------
Use R ::= Q[x,y,z], Lex;
LT(y^2-xz);
xz
-------------------------------
```

If E is a vector, LT(E) gives the leading term of E with respect to the module term-ordering of the current ring. For the leading monomial, which includes the coefficient, use “`LM`”.

example

```
Use R ::= Q[x,y];
V := Vector(0,x,y^2);
LT(V); -- the leading term of V w.r.t. the default term-ordering, ToPos
Vector(0, 0, y^2)
-------------------------------
Use R ::= Q[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 ::= Q[x,y,z];
I := Ideal(x-y,x-z^2);
LT(I);
Ideal(x, z^2)
-------------------------------
```

See Also: LC (VI-1.158 pg.220), LM (VI-1.163 pg.223), LPP (VI-1.167 pg.225), Module Orderings (IV-8.10 pg.99), Orderings (IV-8.6 pg.98)

VI-1.169 MapDown

syntax

```
MapDown(F:POLY):RAT or ZMOD
```

Description

This function converts a constant polynomial to the equivalent coefficient. If the argument is not a constant polynomial, an error is signalled.

example

```
Use Q[x,y,z];
Type((x+1)^2 - x*(x+2));    -- value is seen as a polynomial
POLY
-------------------------------
MapDown((x+1)^2 - x*(x+2)); -- attempt to map down to the coeff ring
1
-------------------------------
Type(It);                   -- value is now simply a coefficient
RAT
-------------------------------
MapDown((x+1)^2 - x^2);     -- 2*x + 1 is not a coefficient
ERROR: Cannot MapDown non-const poly
CONTEXT: Error("Cannot MapDown non-const poly")
-------------------------------
```

VI-1.170 Mat

syntax

```
Mat(E):MAT
Mat[E]:MAT

where E is either: a rectangular lists of lists, a vector, or a
module.
```

Description

This function converts the expression E into a matrix. The first form is equivalent to Cast(E,MAT).

example

```
Use R ::= Q[x,y];
L := [[1,2],[3,4]];
Mat(L);
Mat[
  [1, 2],
  [3, 4]
]
-------------------------------
M := Module([x,x^2,y],[x^2,y,0]);
Mat(M);
Mat[
  [x, x^2, y],
  [x^2, y, 0]
]
-------------------------------
Mat([[1,2],[3,4]]);  -- note the syntax here
 Mat[
  [1, 2],
  [3, 4]
]
-------------------------------
Mat[[1,2],[3,4]];    -- and here
Mat[
  [1, 2],
  [3, 4]
]
-------------------------------
M:=Mat([["a","b"],["c",[1,2]]]);  -- a slightly more obscure example
N:=Mat([["d","e"],["f",[3,4]]]);
M+N;
Mat[
  ["ad", "be"],
  ["cf", [4, 6]]
]
-------------------------------
```

See Also: BlockMatrix (VI-1.14 pg.149), NewMat (VI-1.187 pg.235)

VI-1.171 Max, Min

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, these functions return a maximum and minimum, respectively, of $E_1, ..., E_n$. In the second form, they return a maximum and minimum, respectively, 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 ::= Q[x,y,z];
Max(x^3z, x^2y^2); -- x^2y^2 > x^3z in the default ordering, DegRevLex
x^2y^2
-------------------------------
Min(x^3z, x^2y^2);
x^3z
-------------------------------
Use R ::= Q[x,y,z], DegLex;
Max(x^3z, x^2y^2); -- x^3z > x^2y^2 in DegLex
x^3z
-------------------------------
Max(Ideal(x),Ideal(x^2),Ideal(x,y),Ideal(x-2,y-1));  -- ordered by inclusion
                        -- a maximal element in the list is returned
Ideal(x, y)
-------------------------------
```

See Also: Relational Operators (III-3.3 pg.48)

VI-1.172 MDeg

syntax

```
MDeg(F:POLY):LIST
```

Description

This function returns the multi-degree of F, as determined by the weights of the current ring. The function "Deg" returns the weight given by the first row of the weights matrix.

example

```
Use R ::= Q[x,y], Weights(Mat([[1,2],[3,4],[5,6]]));
MDeg(x);
[1, 3, 5]
-------------------------------
MDeg(y);
[2, 4, 6]
-------------------------------
Deg(y);
2
-------------------------------
MDeg(x^2+y);
[2, 6, 10]
-------------------------------
```

See Also: Deg (VI-1.43 pg.164), Weights Modifier (IV-8.5 pg.97)

VI-1.173 Memory

syntax

```
Memory():TAGGED(Memory)
Memory(R:RING):TAGGED(Memory)
```

Description

The first form of this function prints the contents of the working memory, i.e, all non-global variables. The second form lists all global variables bound to the ring R.

For more information about memory in CoCoA, see the chapter entitled “Memory Management” (III-8 pg.63).

example

```
Use R ::= Q[x,y,z];
I := Ideal(x-y^2,xy-z^3);
X := 14;
ENV.R.Y := 5;  --  a global variable bound to R
               -- recall that "ENV.R" is equivalent to "MEMORY.ENV.R"
Use S ::= Q[a,b];
J := Ideal(a,b);
ENV.S.Z := 7;
Memory();
["I", "J", "X"]
-------------------------------
Memory(R);
["Y"]
-------------------------------
```

See Also: GlobalMemory (VI-1.103 pg.193), Memory Management (III-8 pg.63)

VI-1.174 MinGens

syntax

```
MinGens(M:IDEAL or MODULE or TAGGED(Quotient)):LIST
```

Description

If M is an ideal or module, this function returns a list of minimal generators for M. If M is the quotient of the current ring by an ideal I or the quotient of a free module by the submodule N, then MinGens returns a set of minimal generators for I or N, respectively.

The coefficient ring must be a field.

The input must be homogeneous. The similar command “`Minimalized`”, will accept inhomogeneous input.

example

```
Use R ::= Q[x,y,z];
I:=Ideal(x-y,(x-y)^4,z+y,(z+y)^2);
I;
Ideal(x - y, x^4 - 4x^3y + 6x^2y^2 - 4xy^3 + y^4, y + z, y^2 + 2yz + z^2)
-------------------------------
MinGens(I);
[y + z, x + z]
-------------------------------
MinGens(R/I);
[y + z, x + z]
-------------------------------
M :=Module([x+y,x-y],[(x+y)^2,(x+y)(x-y)]);
MinGens(M);
[Vector(x + y, x - y)]
```

```
-------------------------------
MinGens(R^2/M);
[Vector(x + y, x - y)]
-------------------------------
```

See Also: Minimalize, Minimalized (VI-1.175 pg.230)

VI-1.175 Minimalize, Minimalized

syntax

```
Minimalize(X:IDEAL):NULL
Minimalize(X:MODULE):NULL

Minimalized(E:IDEAL):IDEAL
Minimalized(E:MODULE):MODULE

where X is a variable containing an ideal or module.
```

Description

In the inhomogeneous case the first form removes redundant generators from the ideal or module contained in X, storing the result in X; the original ideal or module is overwritten.

In the inhomogeneous case the second form returns the ideal or module obtained by removing redundant generators from E.

In the homogeneous case, one obtains a generating set with smallest possible cardinality. The minimal set of generators found by CoCoA is not necessarily a subset of the given generators. As with the inhomogeneous case, the first form overwrites the ideal or module contained in X and the second returns the minimalized ideal or module.

The coefficient ring is assumed to be a field.

example

```
Use R ::= Q[x,y,z];
I := Ideal(x-y^2,z-y^5,x^5-z^2);
I;
Ideal(-y^2 + x, -y^5 + z, x^5 - z^2)
-------------------------------
Minimalized(I);
Ideal(-y^2 + x, -y^5 + z)
-------------------------------
I;
Ideal(-y^2 + x, -y^5 + z, x^5 - z^2)
-------------------------------
Minimalize(I);
I;
Ideal(-y^2 + x, -y^5 + z)
-------------------------------
J := Ideal(x, x-y, y-z, z^2);
Minimalized(J);
Ideal(y - z, x - z, z)
-------------------------------
```

See Also: MinGens (VI-1.174 pg.229)

VI-1.176 Minors

syntax

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

Description

This function returns the list of all determinants of N x N submatrices of M.

example
```
M := Mat([[1,2,3],[-1,2,4]]);
Minors(2,M);
[4, 7, 2]
-------------------------------
```

See Also: Det (VI-1.50 pg.169)

VI-1.177 MinSyzMinGens

syntax
```
MinSyzMinGens:  FUNCTION ELIMINATED.
```

Description

The MinSyzMinGens function has been removed.
See Also: SyzMinGens (VI-1.261 pg.273)

VI-1.178 Mod2Rat

syntax
```
Mod2Rat(Residue:INT, Modulus:INT, DenomBound:INT):RAT
```

Description

This function determines a rational number equivalent to the given residue modulo the given modulus; the denominator will not exceed DenomBound, and the absolute value of the numerator will be at most Modulus/(2*DenomBound) – this guarantees uniqueness of the answer. If no such rational exists, or if DenomBound exceeds Modulus/2 then zero is returned. DenomBound must be positive; if not, zero is returned.

example
```
Mod2Rat(239094665,314159265,10000);
355/113
-------------------------------
Residue := 1234567;                          -- To compute inverse of
Modulus := 2^100;                            -- Residue modulo Modulus,
Mod2Rat(Residue, Modulus, Div(Modulus-1,2));-- result is the denominator.
1/284507170216309247716413542199
-------------------------------
```

See Also: Div, Mod (VI-1.55 pg.170)

VI-1.179 Module

syntax
```
Module(V_1:VECTOR,...,V_n:VECTOR):MODULE
Module(L:LIST):MODULE
Module(I:IDEAL):MODULE

where L is a list of elements cast-able to vectors.
```

Description

The first function returns the module generated by the vectors “`V_1,...,V_n`”. The second function returns the module generated by the (elements cast-able to) vectors in L. The third function returns the module generated by the generators of I. This is the same as the function “`Cast(E,MODULE)`”.

example

```
Use R ::= Q[x,y,z];
M := Module([x^2-y^3,x^3-yz],[x^3,z^2]);
M;
Module([-y^3 + x^2, x^3 - yz], [x^3, z^2])
-------------------------------
L := [[-y^3 + x^2, x^3 - yz], [x^3, z^2]];
Module(L) = M;
TRUE
-------------------------------
I := Ideal(x,y^2,z^3);
Module(I);
Module([x], [y^2], [z^3])
-------------------------------
```

See Also: Cast (VI-1.18 pg.151)

VI-1.180 Monic

syntax

```
Monic(F:POLY):POLY
Monic(L:LIST of POLY):LIST of POLY
```

Description

In the first form, this function returns F divided by its leading coefficient (see “LC” (VI-1.158 pg.220)) or, if F is zero, it returns zero.

In its second form, it returns the list obtained by applying the first form of Monic to each of the components of L.

example

```
Use R ::= Q[x,y];
L := [4x^5-y^2,3x-2y^4];
Monic(L);
[x^5 - 1/4y^2, y^4 - 3/2x]
-------------------------------
Use R ::= Z[x,y];
-- WARNING: Coeffs are not in a field
-- GBasis-related computations could fail to terminate or be wrong
-------------------------------

-------------------------------
L := [4x^5-y^2,3x-2y^4];
Monic(L);  -- can't invert coefficients over Z

-------------------------------
ERROR: Cannot divide
CONTEXT: Cond(Type(X) = LIST, [Monic(A)|A IN X], IsZero(X), X, X / LC(X))
-------------------------------
Use R ::= Z/(5)[x,y];
F := 2x^2+4y^3;
Monic(F);
```

```
y^3 - 2x^2
-------------------------------
```

See Also: LC (VI-1.158 pg.220)

VI-1.181 Monomials

syntax

```
Monomials(F:POLY or VECTOR):LIST
```

Description

This function returns the list of monomials of F. The function "`Support`" returns the list of terms (monomials without coefficients).

example

```
Use R ::= Q[xy];
F := 3x^2y+5y^3-xy^5;
Monomials(F);
[-xy^5, 3x^2y, 5y^3]
-------------------------------
Support(F);
[xy^5, x^2y, y^3]
-------------------------------
Monomials(Vector(3x^2y+y,5xy+4));
[Vector(3x^2y, 0), Vector(0, 5xy), Vector(y, 0), Vector(0, 4)]
-------------------------------
```

See Also: Coefficients (VI-1.27 pg.156), Support (VI-1.258 pg.272)

VI-1.182 MonsInIdeal

syntax

```
MonsInIdeal(I:IDEAL):IDEAL
```

Description

This function returns the ideal generated by all monomials in the original ideal I.

example

```
Use R ::= Q[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)
-------------------------------
```

VI-1.183 More

syntax

```
More(S:STRING):NULL
More():NULL
```

Description

The purpose of this function is to print the string S without scrolling off of the screen.

The first form of this function stores the string S in a "`MoreDevice`", then prints the first N lines from the MoreDevice where N is the integer stored in the global variable MEMORY.MoreCount. Subsequent calls to "`More`" print the next N lines from the MoreDevice, (each time removing the lines from the device) until the MoreDevice is empty. After each call to "`More`" a line with "`More();`" is printed as long as the MoreDevice is not empty. This line is easily cut-and-pasted.

The user may set the number of lines to print with the command "`MEMORY.MoreCount := X`", where X is an integer. The default is value is 20.

If E is *any* CoCoA object, the command "`Sprint(E)`" converts E into a string. The output may then be given to "`More`" for printing.

To use the more device with the online help system, see "`H.SetMore, H.UnSetMore`".

See Also: H.SetMore, H.UnSetMore (VI-1.109 pg.196), Commands and Functions for Strings (IV-3.5 pg.86)

VI-1.184 Multiplicity

syntax

```
Multiplicity(R:RING or TAGGED(Quotient)):INT
```

Description

This function computes the multiplicity (or degree) of the ring R, i.e., the leading coefficient of the Hilbert polynomial multiplied by the factorial of the degree of the Hilbert polynomial. The weights of the indeterminates of the current ring must all be 1.

example

```
Use R ::= Q[t,x,y,z];
Multiplicity(R/Ideal(x,y,z)^5);
35
-------------------------------
```

See Also: Hilbert (VI-1.116 pg.198), Poincare, HilbertSeries (VI-1.209 pg.245)

VI-1.185 NewId

syntax

```
NewId():STRING
```

Description

This function returns a string of the form "`V#N`" where N is an integer. Each time it is called, the integer N changes, producing a new string. The purpose is to produce identifiers for variables or rings. (CoCoA does not check for the unlikely event that variables of the same form have been defined without the use of "`NewId`".) The function "`NewId`" is often used with "`Var`".

The most important use for this function is for creating temporary rings within user-defined functions. For an example, see the section of the tutorial entitled "Rings Inside User-Defined Functions" (II-2.18 pg.31).

example

```
NewId();
V#0
-------------------------------
NewId();
V#1
-------------------------------
X := NewId();
X;
V#2
```

```
-------------------------------
Var(X) := 3;
Var(NewId()) := 4;
Describe Memory();
------------[Memory]-----------
It = V#2
V#2 = 3
V#3 = 4
X = V#2
-------------------------------
Y := NewId();
Var(Y) ::= Q[a,b];
Use Var(Y);
RingEnvs();
["Q", "Qt", "R", "V#6", "Z"]
-------------------------------
Y;
V#6
-------------------------------
Var(Y);
Q[a,b]
-------------------------------
```

See Also: Rings Inside User-Defined Functions (II-2.18 pg.31), Var (VI-1.276 pg.280)

VI-1.186 NewList

syntax

```
NewList(N:INT):LIST
NewList(N:INT,E:OBJECT)
```

Description

The second function returns a list of length N, filled by E. The first function, in which E is not indicated, returns a list of length N filled with “`Null`” values.

example

```
NewList(4,"a");
["a", "a", "a", "a"]
-------------------------------
NewList(4);
[Null, Null, Null, Null]
-------------------------------
```

See Also: List (VI-1.162 pg.222)

VI-1.187 NewMat

syntax

```
NewMat(M:INT,N:INT):MAT
NewMat(M:INT,N:INT,E:OBJECT):MAT
```

Description

The second function returns an MxN matrix, filled by E. The first function, in which E is not indicated, returns an MxN matrix filled with ”Null” values.

example

```
NewMat(2,3,"a");
Mat[
  ["a", "a", "a"],
  ["a", "a", "a"]
]
-------------------------------
NewMat(2,2);
Mat[
  [Null, Null],
  [Null, Null]
]
-------------------------------
```

See Also: Mat (VI-1.170 pg.226)

VI-1.188 NewVector

syntax

```
NewVector(N:INT):VECTOR
NewVector(N:INT,E:OBJECT):VECTOR

where E is cast-able to POLY.
```

Description

The second function returns a vector of length N, each of whose components is E. The first function, in which E is not indicated, returns a vector of length N filled with zeros.

example

```
Use R ::= Q[x,y];
NewVector(4);
Vector(0, 0, 0, 0)
-------------------------------
NewVector(3,x^2+y);
Vector(x^2 + y, x^2 + y, x^2 + y)
-------------------------------
```

See Also: Vector (VI-1.277 pg.281)

VI-1.189 NextPrime

syntax

```
NextPrime(N:INT):INT
```

Description

This function computes the smallest prime number greater than N. If N is negative or too large then the value zero is returned. This function may generate primes larger than permitted as the characteristic of a finite field in CoCoA. On most platforms primes up to about 45000 can be generated; in some cases a higher limit exists.

example

```
NextPrime(1000);
1009
-------------------------------
```

See Also: IsPrime (VI-1.149 pg.216)

VI-1.190 NF

syntax

```
NF(F:POLY,I:IDEAL):POLY
NF(V:VECTOR,M:MODULE):VECTOR
```

Description

The first function returns the normal form of F with respect to I. It also computes a Groebner basis of I if that basis has not been computed previously.

The second function returns the normal form of V with respect to M. It also computes a Groebner basis of M if that basis has not been computed previously.

The coefficient ring is assumed to be a field. Note that the definition of normal form depends on the current value of the option FullRed of the panel GROEBNER. If FullRed is FALSE it means that a polynomial is in normal form when its leading term with respect to the the current term ordering cannot be reduced. If FullRed is TRUE it means that a polynomial is in NF if and only if each monomial cannot be reduced.

example

```
Use R ::= Q[x,y,z];
Set FullRed;
I := Ideal(z);
NF(x^2+xy+xz+y^2+yz+z^2,I);
x^2 + xy + y^2
-------------------------------
UnSet FullRed;
NF(x^2+xy+xz+y^2+yz+z^2,I);
x^2 + xy + y^2 + xz + yz + z^2
-------------------------------
```

See Also: DivAlg (VI-1.56 pg.171), FullRed (V-1.13 pg.129), GenRepr (VI-1.98 pg.191), IsIn (VI-1.146 pg.215), NFsAreZero (VI-1.191 pg.237), NR (VI-1.194 pg.238)

VI-1.191 NFsAreZero

syntax

```
NFsAreZero(L:LIST of POLY,M IDEAL):BOOL
NFsAreZero(L:LIST of VECTOR,M:MODULE):BOOL
```

Description

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 ::= Q[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 (VI-1.146 pg.215), NF (VI-1.190 pg.237)

VI-1.192 NonZero

syntax

```
NonZero(L:LIST or VECTOR):LIST
```

Description

This function returns the list obtained by removing the zeroes from L.

example

```
Use R ::= Q[x,y,z];
NonZero(["a",0,0,3,Ideal(y),0]);
["a", 3, Ideal(y)]
-------------------------------
```

See Also: FirstNonZero, FirstNonZeroPos (VI-1.69 pg.177)

VI-1.193 Not, And, Or

syntax

```
Not E
E And F
E Or F

where E and F are of type BOOL.
```

Description

These operators have their usual meanings. Note that when two or more boolean expressions are combined with AND, they are evaluated one by one until a FALSE expression is found. The rest are not evaluated. For example, given the expression "`A And B`", the system does not attempt to evaluate B unless A evaluates to TRUE. Similarly, evaluation of a sequence of boolean expressions connected by OR stops as soon as a TRUE expression is found.

VI-1.194 NR

syntax

```
NR(X:POLY,L:LIST of POLY):POLY
NR(X:VECTOR,L:LIST of VECTOR):VECTOR
```

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`". 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`" or "`NF`" instead).

example

```
Use R::= Q[xyz];
F := x^2y+xy^2+y^2;
NR(F,[xy-1,y^2-1]);
x + y + 1
-------------------------------
V := Vector(x^2+y^2+z^2,xyz);
NR(V,[Vector(x,y),Vector(y,z),Vector(z,x)]);
Vector(z^2, z^3 - yz - z^2)
-------------------------------
```

See Also: DivAlg (VI-1.56 pg.171), GenRepr (VI-1.98 pg.191), NF (VI-1.190 pg.237)

VI-1.195 Num, Den

syntax

```
Num(N:INT or RAT):INT
Den(N:INT or RAT):INT

Num(N:POLY or RATFUN):POLY
Den(N:POLY or RATFUN):POLY
```

Description

These functions return the numerator and denominator of N. The numerator and denominator can also be found using "`.Num`" and "`.Den`".

example

```
Num(3/2);
3
-------------------------------
Den(x/(x+y));
x + y
-------------------------------
X := 2/3;
X.Num; X.Den;
2
-------------------------------
3
-------------------------------
```

See Also: Numerators and Denominators for Rational Functions (IV-10.2 pg.109), Numerators and Denominators for Rational Numbers (IV-2.3 pg.81)

VI-1.196 NumComps

syntax

```
NumComps(X:VECTOR or MODULE):INT
```

Description

If X is a vector, this function returns the number of components of X; it gives the same result as Len(X). If X is a module, then this function returns the rank of the free module in which X is defined.

example

```
Use R ::= Q[x,y];
NumComps(Vector(x,y,x^2+y^2,x^2-y^2));
4
-------------------------------
M := Module([x,y^2,2+x^2y],[x,0,y]);  -- a submodule of R^3
NumComps(M);
3
-------------------------------
M.NumComps;  -- alternative syntax
3
-------------------------------
```

See Also: Len (VI-1.159 pg.220)

VI-1.197 NumIndets

syntax

```
NumIndets():INT
NumIndet(R:RING):INT
```

Description

This function returns the number of indeterminates of the current ring or of R.

example

```
S ::= Q[x,y];
Use R ::= Q[x,y,z];
NumIndets();
3
-------------------------------
NumIndets(S);
2
-------------------------------
```

See Also: Indet (VI-1.132 pg.208), IndetInd (VI-1.133 pg.208), IndetIndex (VI-1.134 pg.209), IndetName (VI-1.135 pg.209), Indets (VI-1.136 pg.209)

VI-1.198 OpenIFile, OpenOFile

syntax

```
OpenIFile(S:STRING):DEVICE
OpenOFile(S:STRING):DEVICE
OpenOFile(S:STRING,w or W):DEVICE
```

Description

These functions open files for input or output. “`OpenIFile`” opens the file with name S. Input from that file can then be read with “`Get`”. “`OpenOFile`” opens the file with name S—creating it if it does not already exist—for output. The function “`Print On`” is then used for writing output to the file. If OpenOFile is used without a second argument or if the second argument is not “`w`” or “`W`” then “`Print On`” will append output to the file. Otherwise, any existing file with the name S will be erased before the output is written.

(Note: one would normally use “`Source`” 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"
Print " test" On D;  -- append to the file "mytest"
Close(D);  -- close the file
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);
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 (VI-1.24 pg.155), Introduction to IO (III-7.1 pg.57), OpenIString, OpenOString (VI-1.199 pg.241), Source, ¡¡ (VI-1.250 pg.268)

VI-1.199 OpenIString, OpenOString

syntax

```
OpenIString(S:STRING,T:STRING):DEVICE
OpenOString(S:STRING):DEVICE
```

Description

These functions open strings for input or output. 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`". "`OpenOString`" is used to write to a string with the help of "`Print On`".

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
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 = "CoCoAL"]
-------------------------------
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 (VI-1.24 pg.155), Introduction to IO (III-7.1 pg.57), OpenIFile, OpenOFile (VI-1.198 pg.240), Source, ¡¡ (VI-1.250 pg.268), Sprint (VI-1.252 pg.269)

VI-1.200 OpenLog, CloseLog

syntax

```
OpenLog(D:DEVICE):NULL
CloseLog(D:DEVICE):NULL
```

Description

The first function opens the output device D and starts to record the output from a CoCoA session on D. The second function 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`") or an output string (see "`OpenOString`"). 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 (III-7.1 pg.57), OpenIFile, OpenOFile (VI-1.198 pg.240), OpenIString, OpenOString (VI-1.199 pg.241), Set, Unset (VI-1.244 pg.264)

VI-1.201 Option

syntax

```
Option(O):BOOL

where O is a panel option.
```

Description

This function returns the status of a panel option (TRUE/FALSE). For a list of panels, use "`Panels`" and for the current status of each option for a panel with name P, use "`Panel(P)`". To toggle option values, use "`Set`" and "`Unset`". The function "`Option`" is particularly useful inside a user-defined function in order to temporarily change the value of an option (restoring the option's original value when the function is complete). See "Setting Options" (V-1.2 pg.125) for an example of this use of "`Option`".

example

```
Option(Indentation);
FALSE
-------------------------------
```

See Also: Introduction to Panels (V-1.1 pg.125), Panel (VI-1.204 pg.244), Panels (VI-1.205 pg.244), Setting Options (V-1.2 pg.125), Set, Unset (VI-1.244 pg.264)

VI-1.202 Ord

syntax

```
Ord():MAT
Ord(R:RING):MAT
Ord(M:MAT):MAT
```

Description

The first two forms return matrices which describe the term-ordering of the current ring or of the ring R, respectively. The last form is used as a modifier when creating a new ring. In that case, it determines the term-ordering for the ring (see “Orderings” (IV-8.6 pg.98)). Its argument is a matrix of small integers which defines a term-ordering; i.e. for a ring with N indeterminates it must be an NxN matrix of full rank where the first non-zero entry in each column is positive. The matrix entries must be in the range -32767 to +32767, otherwise an error results.

example

```
Use S ::= Q[x,y,z], Ord(Mat([[1,0,0],
                             [0,1,0],
                             [0,0,1]]));

M := Mat([[1,1],[0,-1]]);
T ::= Q[a,b], Ord(M);
U ::= Z/(101)[x,y,z,t], DegRevLex;
-- The term-order for the current ring, S.
Ord();
Mat[
  [1, 0, 0],
  [0, 1, 0],
  [0, 0, 1]
]
-------------------------------
Ord(T);
Mat[
  [1, 1],
  [0, -1]
]
-------------------------------
Ord(U);
Mat[
  [1, 1, 1, 1],
  [0, 0, 0, -1],
  [0, 0, -1, 0],
  [0, -1, 0, 0]
]
-------------------------------
```

See Also: DegLexMat, DegRevLexMat, LexMat, XelMat (VI-1.44 pg.165), Elim (VI-1.58 pg.172), Orderings (IV-8.6 pg.98), Predefined Term-Orderings (IV-8.7 pg.98)

VI-1.203 Packages

syntax

```
Packages():LIST
```

Description

This function returns the names of the loaded packages as a list of strings. The string “`$cocoa/user`” refers to the user-defined functions defined in the current CoCoA session.

example

```
Packages();
["‘‘\verb&$cocoa/builtin&’’", "‘‘\verb&$cocoa/coclib&’’", "‘‘\verb&$cocoa/user&’’", "$cocoa/help",
"$cocoa/io", "$cocoa/misc"]
-------------------------------
```

See Also: CoCoA Packages (III-9 pg.67), Supported Packages (III-9.11 pg.72)

VI-1.204 Panel

syntax

```
Panel(E)

where E is one of GENERAL or GROEBNER.
```

Description

This command prints the status of the options in one of the three panels. It returns no value.

example

```
Panel(GENERAL);

Echo............... : FALSE
Timer.............. : FALSE
Trace.............. : FALSE
Indentation........ : FALSE
TraceSources....... : FALSE
SuppressWarnings... : FALSE
ComputationStack... : FALSE
-------------------------------
```

See Also: Introduction to Panels (V-1.1 pg.125), Option (VI-1.201 pg.242), Panels (VI-1.205 pg.244), Set, Unset (VI-1.244 pg.264)

VI-1.205 Panels

syntax

```
Panels()
```

Description

This function returns a list of CoCoA's panels.

example

```
Panels();
["GENERAL", "GROEBNER"]
-------------------------------
Panel(It[1]);

Echo............... : FALSE
Timer.............. : FALSE
Trace.............. : FALSE
Indentation........ : FALSE
TraceSources....... : FALSE
SuppressWarnings... : FALSE
ComputationStack... : FALSE
-------------------------------
```

See Also: Introduction to Panels (V-1.1 pg.125), Option (VI-1.201 pg.242), Panel (VI-1.204 pg.244), Set, Unset (VI-1.244 pg.264)

VI-1.206 Partitions

syntax

```
Partitions(N: INT): LIST
```

Description

These function returns all integer partitions of N, positive integer

example

```
Partitions(3);
[[3], [1, 2], [1, 1, 1]]
-------------------------------
```

VI-1.207 Pfaffian

syntax

```
Pfaffian(M:MAT)

where M is skew-symmetric.  The resulting type depends on the entries
of the matrix.
```

Description

This function returns the Pfaffian of M.

example

```
Use R ::= Q[x,y];
Pfaffian(Mat([[0,y],[-y,0]]));
y
-------------------------------
```

See Also: Det (VI-1.50 pg.169)

VI-1.208 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 "`$cocoa/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();
$cocoa/gb
-------------------------------
$cocoa/gb.PkgName();
$cocoa/gb
-------------------------------
PkgName();
$cocoa/coclib
-------------------------------
```

VI-1.209 Poincare, HilbertSeries

syntax

```
Poincare(M:RING or TAGGED(Quotient)):TAGGED($cocoa/hp.PSeries)
HilbertSeries(M:RING or TAGGED(Quotient)):TAGGED($cocoa/hp.PSeries)
```

```
PoincareShifts(M: Module, ShiftsList: LIST):TAGGED($cocoa/hp.PSeries)
PoincareShifts(M: TAGGED(Quotient), ShiftsList: LIST)
                                    :TAGGED($cocoa/hp.PSeries)
```

Description

These functions all compute the Hilbert-Poincare series of M. The first two functions perform the same operations. The input, M, must be homogeneous (with respect to the first row of the weights matrix). In the standard case, where the weights of all indeterminates are 1, the result is simplified so that the power appearing in the denominator is the dimension of M.

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 R ::= Q[t,x,y,z];
Poincare(R);
(1) / (1-t)^4
-------------------------------
Q := R/Ideal(t^2,x,y^3); Poincare(Q);
(1 + 2t + 2t^2 + t^3) / (1-t)
-------------------------------
Poincare(R^2/Module([x^2,y],[z,y]));
(2 + t) / (1-t)^3
-------------------------------
Use R ::= Q[t,x,y,z], Weights([1,2,3,4]);
Poincare(R/Ideal(t^2,x,y^3));
---  Non Simplified Pseries  ---
(1-2t^2 + t^4 - t^9 + 2t^11 - t^13) / ( (1-t) (1-t^2) (1-t^3) (1-t^4) )
-------------------------------
Use R ::= Q[t,x,y,z], Weights(Mat([[1,2,3,4],[0,0,5,8]]));
Poincare(R/Ideal(t^2,x,y^3));
---  Non Simplified Pseries  ---
( - t^13x^15 + 2t^11x^15 - t^9x^15 + t^4-2t^2 + 1) / ( (1-t) (1-t^2) (1-t^3x^5) (1-t^4x^8) )
-------------------------------
Use P ::= Q[x,y,z];
M := Module([x,y^3], [x-z,0]);
PoincareShifts(M, [2,0]);     -- Poincare series of a shifted module
(2x^3) / (1-x)^3
-------------------------------
PoincareShifts(P^2/M, [3,1]); -- Poincare series of a shifted quotient module
(x + x^2 + 2x^3) / (1-x)^2
-------------------------------
```

See Also: Dim (VI-1.52 pg.169), Hilbert (VI-1.116 pg.198), HVector (VI-1.121 pg.201), Multiplicity (VI-1.184 pg.234), Weights Modifier (IV-8.5 pg.97), WeightsMatrix (VI-1.279 pg.282)

VI-1.210 Poly

syntax

```
Poly(E:OBJECT):POLY
```

Description

This function converts the expression E into a polynomial, if possible. It is the same as Cast(E,POLY).

example

```
F := 3;
G := Poly(3);
Type(F);
INT
-------------------------------
Type(G);
POLY
-------------------------------
Use R ::= Q[x];
Poly(Vector(x));
x
-------------------------------
```

See Also: Cast (VI-1.18 pg.151)

VI-1.211 PrimaryDecomposition

syntax

```
PrimaryDecomposition(I: SQUAREFREE MONOMIAL 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.

example

```
Use R ::= Q[x,y,z];
PrimaryDecomposition(Ideal(xy, xz, yz));
[Ideal(y, z), Ideal(x, z), Ideal(x, y)]
-------------------------------
```

See Also: EquiIsoDec (VI-1.61 pg.173)

VI-1.212 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`” or “`OpenOString`”.

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
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 ‘‘OpenOFile’’ (\ref{OpenOFile} pg.\pageref{OpenOFile}) for an example using output strings.  For prin
the CoCoA window, just use ‘‘\verb&Print E&’’ which is short for
‘‘\verb&Print E On DEV.OUT&’’.
```

See Also: Introduction to IO (III-7.1 pg.57), OpenIFile, OpenOFile (VI-1.198 pg.240), OpenIString, OpenOString (VI-1.199 pg.241), Print, PrintLn (VI-1.213 pg.248)

VI-1.213 Print, PrintLn

syntax

```
Print E_1,...,E_n :NULL
Print(E_1,...,E_n):NULL

PrintLn E_1,...,E_n :NULL
PrintLn(E_1,...,E_n):NULL

where the E_i are CoCoA expressions.
```

Description

The command “`Print`” displays the value of each of the expressions, E_i. The parentheses are optional. The argument, “`NewLine`”, (without quotes, but note the two capital letters), moves the cursor to the next line.

The command “`PrintLn`” is equivalent to “`Print`” with a final extra argument, “`NewLine`”; in other words, it prints the values of its arguments, then moves the cursor to the next line. The parentheses are optional.

example

```
For I := 1 To 10 Do
  Print(I^2, " ");
EndFor;
1 4 9 16 25 36 49 64 81 100
-------------------------------
For I := 1 To 3 Do
  PrintLn(I);
EndFor;
1
2
3

-------------------------------
Print "hello",NewLine,"world";
hello
world
-------------------------------
```

See Also: Print On (VI-1.212 pg.247)

VI-1.214 Product, Sum

syntax

```
Product(L:List):OBJECT
Sum(L:List):OBJECT
```

Description

These functions return the product and sum, respectively, of the objects in the list L.

example
```
Use R ::= Q[x,y];
Product([3,x,y^2]);
3xy^2
-------------------------------
Sum([3,x,y^2]);
y^2 + x + 3
-------------------------------
Product(1..40)=Fact(40);
TRUE
-------------------------------
Sum(["c","oc","oa"]);
cocoa
-------------------------------
```

See Also: Algebraic Operators (III-3.2 pg.47)

VI-1.215 Quit

syntax
```
Quit
```

Description

This command is used to quit CoCoA. Note, it is issued as follows:

Quit;

without parentheses.

See Also: Ciao (VI-1.22 pg.154)

VI-1.216 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 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.

example
```
Points:=[[Rand(-9,9) | N In 1..3] | S In 1..25];
Use Q[x,y,z];
I:=IdealOfPoints(Points);
QuotientBasis(I);     -- power products underneath the DegRevLex reduced GBasis
[1, z, z^2, z^3, z^4, y, yz, yz^2, yz^3, y^2, y^2z, y^2z^2, y^3, x,
xz, xz^2, xz^3, xy, xyz, xyz^2, xy^2, x^2, x^2z, x^2y, x^3]
-------------------------------
Use Q[x,y,z],Lex;
I:=IdealOfPoints(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, z^13,
y, yz, yz^2, yz^3, yz^4, yz^5, yz^6, y^2, y^2z, y^2z^2, y^2z^3]
-------------------------------
```

See Also: IdealOfPoints (VI-1.125 pg.204)

VI-1.217 QZP, ZPQ

syntax

```
QZP(F:POLY):POLY
ZPQ(F:POLY):POLY
QZP(F:LIST of POLY):LIST of POLY
ZPQ(F:LIST of POLY):LIST of POLY
QZP(I:IDEAL):IDEAL
ZPQ(I:IDEAL):IDEAL
```

Description

These functions 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 does the reverse, mapping a polynomial with finite field coefficients into one with rational (actually, integer) coefficients. The function ZPQ 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::=Q[x,y,z];
F:=1/2*x^3+34/567*x*y*z-890;   -- a poly with rational coefficients
Use S::=Z/(101)[x,y,z];
QZP(F);                        -- compute its image with coeffs in Z/(101)
-50x^3 - 19xyz + 19
-------------------------------
G:=It;
Use R;
ZPQ(G);                        -- now map that result back to Q[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 Z/(101)[x,y,z]
0
-------------------------------
```

See Also: Accessing Other Rings (IV-8.11 pg.100), BringIn (VI-1.16 pg.150), Image (VI-1.130 pg.206), Ring Mappings: the Image Function (IV-8.12 pg.101)

VI-1.218 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 ::= Q[x,y];
I := Ideal(x,y)^3;
Radical(I);
Ideal(y, x)
-------------------------------
```

See Also: EquiIsoDec (VI-1.61 pg.173), RadicalOfUnmixed (VI-1.219 pg.251)

VI-1.219 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 ::= Q[x,y];
I := Ideal(x^2 - y^2 - 4x + 4y, x - 2);
-------------------------------
RadicalOfUnmixed(I);
Ideal(x^2 - y^2 - 4x + 4y, x - 2, y - 2)
-------------------------------
Minimalized(It); -- the result may not be presented in its simplest form
Ideal(x - 2, y - 2)
-------------------------------
```

See Also: EquiIsoDec (VI-1.61 pg.173), Radical (VI-1.218 pg.250)

VI-1.220 Rand

syntax

```
Rand():INT
Rand(X:INT,Y:INT):INT
```

Description

In the first form, the function returns a random integer. In the second, it returns a random integer between X and Y, inclusive. (Note: —X-Y— should be less than $2^3$3 to assure a more random distribution.)

example

```
Rand();
6304433354
-------------------------------
Rand(1,100);
8
-------------------------------
Rand(100,1);
14
-------------------------------
Rand(-10^4,0);
-2747
-------------------------------
```

See Also: Randomize, Randomized (VI-1.221 pg.252), Seed (VI-1.240 pg.262)

VI-1.221 Randomize, Randomized

syntax

```
Randomize(V:POLY):POLY
Randomized(F:POLY or INT):POLY or INT

where V is a variable containing a polynomial.
```

Description

The first 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.

The second 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, the second function 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.

example

```
Use R ::= Q[x];
F := 1+x+x^2;
Randomized(F);
-2917104644x^2 + 3623608766x - 2302822308
-------------------------------
F;
x^2 + x + 1
-------------------------------
Randomize(F);
F;
-1010266662x^2 + 1923761602x - 4065654277
-------------------------------
Randomized(23);
-3997312402
-------------------------------
Use R ::= Z/(7)[x,y];
Randomized(x^2+3x-5);
3x^2 + 2x - 2
-------------------------------
```

See Also: Rand (VI-1.220 pg.251)

VI-1.222 Rank

syntax

```
Rank(M:MODULE):INT
Rank(M:MAT):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 NumComps which simply counts the number of components the module has.

example

```
Use R ::= Q[x,y,z];
Rank(Module([x,y,z,0]));
1
-------------------------------
Rank(Module([[1,2,3],[2,4,6]]));
```

```
1
-------------------------------
Rank(Module([[1,2,3],[2,5,6]]));
2
-------------------------------
```

VI-1.223 RealRootRefine

syntax

```
RealRootRefine(Root:RECORD, Precision:RAT):RECORD
```

Description

This functions computes a refinement of a real root of a univariate polynomial over Q to the desired precision (width of isolating interval). The starting root must be a record produced by RealRoots.

example

```
RR := RealRoots(x^2-2);
RealRootRefine(RR[1], 1/2);
Record[ CoeffList = [-8, 1456, -712], Inf = -91/64, Sup = -45/32]
-------------------------------
RR := [RealRootRefine(Root, 10^(-20)) | Root In RR];
FloatStr(RR[1].Inf);
-1.414213562*10^0
-------------------------------
```

See Also: RealRoots (VI-1.224 pg.253), RootBound (VI-1.237 pg.261)

VI-1.224 RealRoots

syntax

```
RealRoots(F:POLY):LIST
RealRoots(F:POLY, Precision:RAT):LIST
RealRoots(F:POLY, Precision:RAT, Interval:[RAT,RAT]):LIST
```

Description

This function computes isolating intervals for the real roots of a univariate polyomial over Q. It returns the list of the real roots, where a root is represented as a record containing either the exact root (if the fields Inf and Sup are equal), or an open interval (Inf, Sup) containing the root. A third field (called CoeffList) has an obscure meaning.

An optional second argument specifies the maximum width an isolating interval may have. An optional third argument specifies a closed interval in which to search for roots.

The interval represented by a root record may be refined by using the function RealRootRefine.

example

```
RealRoots(x^2-2);
[Record[CoeffList = [8, -16, 7], Inf = -4, Sup = 0],
Record[CoeffList = [8, 0, -1], Inf = 0, Sup = 4]]
-------------------------------
RR := RealRoots((x^2-2)*(x-1), 10^(-5));
FloatStr(RR[1].Inf);  -- left end of interval
-1.414213657*10^0
-------------------------------
FloatStr(RR[1].Sup);  -- right end of interval
-1.414213419*10^0
-------------------------------
RR := RealRoots(x^2-2, 10^(-20), [0, 2]);
```

```
RR[1].Inf;                               -- incomprehensible
3339217363285192246361/2361183241434822606848
-------------------------------
FloatStr(RR[1].Inf, 20);                 -- comprehensible
1.4142135623730950488*10^0
-------------------------------
```

See Also: RealRootRefine (VI-1.223 pg.253), RootBound (VI-1.237 pg.261)

VI-1.225 Record

syntax
```
Record[X_1 = OBJECT,...,X_n = OBJECT]

where each X_i is a variable.
```

Description

This function returns a record with fields “`X_1`”,...,“`X_n`”. The empty record is given by “`Record[]`”. The records are “*open*” in the sense that new fields may be added after the record is first defined.

example
```
P := Record[ Height = 10, Width = 5];
P.Height * P.Width;
50
-------------------------------
P.Area := It;
P;
Record[Area = 50, Height = 10, Width = 5]
-------------------------------
```

See Also: Fields (VI-1.67 pg.176)

VI-1.226 ReducedGBasis

syntax
```
ReducedGBasis(M:IDEAL, MODULE, or TAGGED(Quotient)):LIST
```

Description

If M is an ideal or module, this function returns a list whose components form a reduced Groebner basis for M with respect to the term-ordering of the current ring. If M is a quotient of the current ring by an ideal I or of a free module by a submodule N, then the Groebner basis for M is defined to be that of I or N, respectively.

example
```
Use R ::= Q[t,x,y,z];
I := Ideal(t^3-x,t^4-y,t^5-z);
GB.Start_GBasis(I);  -- start the Interactive Groebner Framework
GB.Step(I);  -- take one step towards computing the Groebner basis
I.GBasis;  -- the Groebner basis so far
[t^3 - x]
-------------------------------
GB.Complete(I);  -- finish the computation
I.GBasis;
[t^3 - x, -tx + y, -ty + z, -y^2 + xz, -x^2 + tz, t^2z - xy]
-------------------------------
ReducedGBasis(I);
[t^3 - x, tx - y, ty - z, y^2 - xz, x^2 - tz, t^2z - xy]
-------------------------------
```

See Also: Introduction to Groebner Bases in CoCoA (IV-13.1 pg.117), The Interactive Groebner Framework (IV-13.3 pg.118)

VI-1.227 Repeat

syntax

```
Repeat C Until B
Repeat C EndRepeat

where C is a sequence of commands and B is a boolean expression.
```

Description

In the first form, the command sequence C is repeated until B evaluates to FALSE. Unlike the "`While`" command, C is executed at least once. Note that there is no "`EndRepeat`" following B. In the second form, the command sequence C is repeated until a "`Break`" or "`Return`" is encountered within C.

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);
  Return If N = 5;
EndRepeat;
1
2
3
4
5

-------------------------------
```

See Also: For (VI-1.73 pg.179), Foreach (VI-1.74 pg.180), While (VI-1.280 pg.283)

VI-1.228 Res

syntax

```
Res(M):TAGGED($cocoa/gb.Res)

where M is of type IDEAL or MODULE or TAGGED(Quotient).
```

Description

This function returns the minimal free resolution of M. If M is a quotient of the current ring by an ideal I or a quotient of a free module by a submodule N, then the resolution of M is defined to be that of I or N, respectively.

"`Res`" only works in the homogeneous context, and the coefficient ring must be a field.

example

```
Use R ::= Q[x,y,z];
I := Ideal(x,y,z^2);
Res(R/I);
0 --> R(-4) --> R(-2)(+)R^2(-3) --> R^2(-1)(+)R(-2) --> R
-------------------------------
Describe It;

Mat[
  [y, x, z^2]
]
Mat[
  [x, z^2, 0],
  [-y, 0, z^2],
  [0, -y, -x]
]
Mat[
  [z^2],
  [-x],
  [y]
]
-------------------------------

For fine control and monitoring of Groebner basis calculations,
including various types of truncations, see ‘‘The Interactive Groebner
Framework’’ (\ref{The Interactive Groebner
Framework} pg.\pageref{The Interactive Groebner
Framework}) and ‘‘Introduction to Panels’’ (\ref{Introduction to Panels} pg.\pageref{Introduction to Panel
```

See Also: Example: Interactive Resolution Computation (IV-13.6 pg.120), Example: Truncations (IV-13.7 pg.121), Introduction to Groebner Bases in CoCoA (IV-13.1 pg.117)

VI-1.229 Reset

syntax

```
Reset():NULL
```

Description

This function resets the options in the CoCoA panels to their default values and executes “`Seed(0)`”.

example

```
Set Indentation;
Panel(GENERAL);

Echo............... : FALSE
Timer.............. : FALSE
Trace.............. : FALSE
Indentation........ : TRUE
TraceSources....... : FALSE
SuppressWarnings... : FALSE
ComputationStack... : FALSE
-------------------------------
Reset();
Panel(GENERAL);

Echo............... : FALSE
Timer.............. : FALSE
```

```
Trace.............. : FALSE
Indentation........ : FALSE
TraceSources....... : FALSE
SuppressWarnings... : FALSE
ComputationStack... : FALSE
-------------------------------
```

See Also: Introduction to Panels (V-1.1 pg.125), ResetPanels (VI-1.230 pg.257), Seed (VI-1.240 pg.262)

VI-1.230 ResetPanels

syntax

```
ResetPanels():NULL
```

Description

This function resets the options in the CoCoA panels to their default values.

example

```
Set Indentation;
Panel(GENERAL);

Echo............... : FALSE
Timer.............. : FALSE
Trace.............. : FALSE
Indentation........ : TRUE
TraceSources....... : FALSE
SuppressWarnings... : FALSE
ComputationStack... : FALSE
-------------------------------
ResetPanels();
Panel(GENERAL);

Echo............... : FALSE
Timer.............. : FALSE
Trace.............. : FALSE
Indentation........ : FALSE
TraceSources....... : FALSE
SuppressWarnings... : FALSE
ComputationStack... : FALSE
-------------------------------
```

See Also: Introduction to Panels (V-1.1 pg.125), Reset (VI-1.229 pg.256)

VI-1.231 Resultant

syntax

```
Resultant(F:POLY,G:POLY,X:INDET):POLY
```

Description

This function returns the resultant of the polynomials F and G with respect to the indeterminate X.

example

```
Use R ::= Q[p,q,x];
F := x^3+px-q; G := Der(F,x);
Resultant(F,G,x);
4p^3 + 27q^2
-------------------------------
```

See Also: Discriminant (VI-1.53 pg.170), Sylvester (VI-1.259 pg.272)

VI-1.232 Return

syntax

```
Return
Return E

where E is an expression.
```

Description

This function is used to exit from a structured command. The latter form returns the value of the expression E to the user. If executed within a nested loop or inside a user-defined function, it breaks out back to the "*top level*", not just to the next higher loop. (For breaks to the next higher loop, see "Break" (VI-1.15 pg.149).)

example

```
Define Rev(L) -- reverse a list
  If Len(L) < 2 Then Return L EndIf;
  M := Rev(Tail(L)); -- recursive function call
  H := Head(L);
  Return Concat(M,[H]);
EndDefine;
Rev([1,2,3,4];
[4, 3, 2, 1]
-------------------------------
-- Another example: returning from a nested loop
For I := 1 To 5 Do
  For J := 1 To 5 Do
    If J > 2 Then Return Else Print([I,J], " ") EndIf
  EndFor;
EndFor;
[1, 1] [1, 2]
-------------------------------
```

See Also: Break (VI-1.15 pg.149), Define (VI-1.41 pg.162)

VI-1.233 Reverse, Reversed

syntax

```
Reverse(V:LIST):NULL
Reversed(L:LIST):NULL

where V is a variable containing a list in the first case.
```

Description

The the function "`Reverse`" reverses the order of the elements of the list in V, returning Null. It does *not* return the reversed list, but instead changes L itself. The function "`Reversed`" returns the reversed list without changing L.

example

```
L := [1,2,3,4];
Reverse(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, Sorted (VI-1.248 pg.267)

VI-1.234 Ring

syntax

```
Ring(R:RING):RING
```

Description

This function returns the ring with identifier R.

example

```
Use R ::= Q[x,y,z];
S ::= Z/(3)[a,b];
Ring(S);
Z/(3)[a,b]
-------------------------------
Ring(R);
Q[x,y,z]
-------------------------------
R;  -- same as above, as long as there is no variable with identifier
    -- R in the working memory
Q[x,y,z]
-------------------------------
CurrentRing();
Q[x,y,z]
-------------------------------
R := 5;  -- a variable with identifier R; now there are two objects
         -- with the identifier R: a variable and a ring
R;
5
-------------------------------
Memory();  -- the variables of the working memory
["It", "R"]
-------------------------------
RingEnvs();  -- the list of rings
["Q", "Qt", "R", "S", "Z"]
-------------------------------
Ring(R);  -- the ring with identifier R
Q[x,y,z]
-------------------------------
```

See Also: CurrentRing (VI-1.37 pg.160), Introduction to Rings (IV-8.1 pg.95), RingEnv (VI-1.235 pg.259), RingEnvs (VI-1.236 pg.260)

VI-1.235 RingEnv

syntax

```
RingEnv():STRING
RingEnv(E:POLY, IDEAL, MODULE, RATFUN, VECTOR):STRING
```

Description

The first form of this function returns the identifier for the current ring. The second form returns the identifier of the ring on which the object E is dependent.

example
```
Use R ::= Q[x,y,z];
I := Ideal(x,y);  -- an object dependent on R
S ::= Z/(3)[a,b];  -- define S, but do not make S active
RingEnv();  -- the current ring
R
-------------------------------
RingEnvs();  -- your result here could be different
["Q", "Qt", "R", "S", "Z"]
-------------------------------
Ring(S);
Z/(3)[a,b]
-------------------------------
I;
Ideal(x, y)
-------------------------------
Use S;  -- S is now the active ring
I;  -- I is labeled by its ring.  The label appears explicitly when R
    -- is not the current ring.
R :: Ideal(x, y)
-------------------------------
RingEnv(I);  -- the ring labeling I
R
-------------------------------
CurrentRing();
Q[x,y,z]
-------------------------------
Use Q[ab];
RingEnv();
CurrentRingEnv
-------------------------------
```

See Also: CurrentRing (VI-1.37 pg.160), Ring (VI-1.234 pg.259), RingEnvs (VI-1.236 pg.260)

VI-1.236 RingEnvs

syntax
```
RingEnvs():TAGGED
```

Description

This function returns the tagged list of identifiers for the existing rings.

example
```
R_1 ::= Q[a,b,c];
R_2 ::= Q[x,y];
RingEnvs();   -- your result may be different
["Q", "Qt", "R", "R_1", "R_2", "Z"]
-------------------------------
```

See Also: CurrentRing (VI-1.37 pg.160), Ring (VI-1.234 pg.259), RingEnv (VI-1.235 pg.259)

VI-1.237 RootBound

syntax
```
RootBound(F:POLY):INT
```

Description

This function computes a bound on the absolute values of the complex roots of a univariate polynomial over Q.

example
```
RootBound(x^2-2);
4
-------------------------------
```

See Also: RealRootRefine (VI-1.223 pg.253), RealRoots (VI-1.224 pg.253)

VI-1.238 Saturation, HSaturation

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

Description

These functions return the saturation of I with respect to J: the ideal of polynomials F such that FG is in I for all G in J^d for some positive integer d.

The function "`HSaturation`" calculates the saturation using a Hilbert-driven algorithm. It differs from "`Saturation`" only when the input is inhomogeneous, in which case, "`HSaturation`" may be faster.

The coefficient ring must be a field.

example
```
Use R ::= Q[xyz];
I := Ideal(x-z,y-2z);
J := Ideal(x-2z,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
-------------------------------
Saturation(L,Ideal(x,y,z)) = K; -- saturating gets rid of the
                                -- irrelevant component
TRUE
-------------------------------
```

See Also: Colon, :, HColon (VI-1.29 pg.157)

VI-1.239 ScalarProduct

syntax
```
ScalarProduct(L, M):OBJECT

where each of L and M is of type VECTOR or LIST
```

Description

This function returns the sum of the product of the components of L and M; precisely:

ScalarProduct(L,M) = Sum([L[I]*M[I]—I In 1..Min(Len(L),Len(M))])).

Thus, the function works even if the lengths of L and M are different. The function works whenever the product of the components of L and M are defined (see “Algebraic Operators” (III-3.2 pg.47)).

example
```
ScalarProduct([1,2,3],[5,0,-1]);
2
-------------------------------
ScalarProduct([1,2,3],[5,0]);
5
-------------------------------
Use R ::= Q[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 (III-3.2 pg.47)

VI-1.240 Seed

syntax
```
Seed(N:INT):INT
```

Description

This function seeds the random number generator, “`Rand`”.

example
```
Seed(5);
Rand();
7321754624711838226149419O2
-------------------------------
Rand();
566765247850388894754751134 49
-------------------------------
Seed(5);  -- with the same seed, "Rand" generates the same sequence
Rand();
7321754624711838226149419O2
-------------------------------
Rand();
566765247850388894754751134 49
-------------------------------
-- The following shows how to make a seed based on the date.
D := Date();
D;
Mon Mar 02 14:43:44 1998
-------------------------------
F := Sum([Ascii(D[N])| N In 1..Len(D)]);
F;
[1455]
-------------------------------
Seed(F[1]);
```

See Also: Rand (VI-1.220 pg.251)

VI-1.241 SeparatorsOfPoints

syntax

```
SeparatorsOfPoints(Points:LIST):LIST

where Points is a list of lists of coefficients representing a set of
*distinct* points in affine space.
```

Description

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`".

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`".

example

```
Use R ::= Q[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 (VI-1.97 pg.191), IdealAndSeparatorsOfPoints (VI-1.123 pg.202), IdealAndSeparatorsOfProjectivePoints (VI-1.124 pg.203), IdealOfPoints (VI-1.125 pg.204), IdealOfProjectivePoints (VI-1.126 pg.204), Interpolate (VI-1.138 pg.210), SeparatorsOfProjectivePoints (VI-1.242 pg.263)

VI-1.242 SeparatorsOfProjectivePoints

syntax

```
SeparatorsOfProjectivePoints(Points:LIST):LIST

where Points is a list of lists of coefficients representing a set of
*distinct* points in projective space.
```

Description

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`".

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`".

example

```
Use R ::= Q[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 (VI-1.97 pg.191), IdealAndSeparatorsOfPoints (VI-1.123 pg.202), IdealAndSeparatorsOfProjectivePoints (VI-1.124 pg.203), IdealOfPoints (VI-1.125 pg.204), IdealOfProjectivePoints (VI-1.126 pg.204), Interpolate (VI-1.138 pg.210), SeparatorsOfPoints (VI-1.241 pg.263)

VI-1.243 Set

syntax

```
Set(L:LIST):LIST
```

Description

This function returns a list obtained by removing duplicates from L.

example

```
Set([2,2,2,1,2,1,1,3,3]);
[2, 1, 3]
-------------------------------

WARNING: to test two sets for equality use the function EqSet instead
of a normal equality test (because the latter yields false if the
elements are in a different order).

NOTE: there is a command named ``\verb&Set&'' for setting panel options.  Its
search key in the online help is ``\verb&Set, Unset&''.  The search key for the
present function is ``\verb&Set&''.
```

See Also: EqSet (VI-1.59 pg.173), Intersection, IntersectionList (VI-1.140 pg.212), Insert, Remove (VI-1.137 pg.210)

VI-1.244 Set, Unset

syntax

```
Set O
Set O := B:BOOL
UnSet O

where O is a panel option.
```

Description

The command “`Set`” in its first form sets a panel option to TRUE. The command “`UnSet`” sets a panel option to FALSE. The command “`Set`” in the second-listed form can be used to set an option to TRUE or FALSE. A list of panels is returned by “`Panels()`”, and a list of panel options for a panel with name P is printed by “`Panel(P)`”. The current status of an option is returned by “`Option`”.

example

```
Panel(GROEBNER);

Sugar........... : TRUE
FullRed......... : TRUE
SingleStepRed... : FALSE
Verbose......... : FALSE
-------------------------------
Set Verbose;
UnSet Sugar;
Set FullRed := FALSE;
Panel(GROEBNER);

Sugar........... : FALSE
FullRed......... : FALSE
SingleStepRed... : FALSE
Verbose......... : TRUE
-------------------------------

NOTE: there is also a function called ‘‘\verb&Set&’’ which takes a list
obtained by removing duplicate elements.  The search key for that
function in online help is ‘‘\verb&Set&’’ and the search key for the present
command is ‘‘\verb&Set, Unset&’’.
```

See Also: Introduction to Panels (V-1.1 pg.125), Panel (VI-1.204 pg.244), Option (VI-1.201 pg.242), Panels (VI-1.205 pg.244)

VI-1.245 Shape

syntax

```
Shape(E:LIST):LIST (of TYPE)
Shape(E:MAT):MAT (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 ::= Q[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("$cocoa/misc.IsSubShape");
Describe Function("$cocoa/misc.IsSubShapeOfSome");
```

See Also: Data Types (III-2.6 pg.44)

VI-1.246 Size

syntax

```
Size(E:OBJECT):INT
```

Description

This function returns the amount of memory used by the object E, expressed in words (1 word = 4 bytes = 32 bits).

example

```
Use R ::= Q[x,y];
Size(1);
1
-------------------------------
Size(2^32-1);
1
-------------------------------
Size(2^32);
2
-------------------------------
Size(2^64);
3
-------------------------------
Size(x);
32
-------------------------------
Size([x,y]);
64
-------------------------------
```

See Also: Count (VI-1.36 pg.160), Len (VI-1.159 pg.220)

VI-1.247 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;
```

VI-1.248 Sort, Sorted

syntax
```
Sort(V:LIST):NULL
Sorted(L:LIST):LIST

where V is a variable containing a list.
```

Description

The first function sorts the elements of the list in V with respect to the default comparisons related to their types; it overwrites V. The second function, "`Sorted`", returns the list of the sorted elements of L without affecting L, itself. For more on the default comparisons, see "Relational Operators" (III-3.3 pg.48) in the chapter on operators. For more complicated sorting, see "`SortBy, SortedBy`".

example
```
L := [3,2,1];
Sort(L);
L;
[1, 2, 3]
-------------------------------
Use R ::= Q[x,y,z];
L := [x,y,z];
Sort(L);
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"]
-------------------------------
Sorted([Ideal(x,y),Ideal(x)]); -- ideals are ordered by containment
[Ideal(x), Ideal(x, y)]
-------------------------------
```

See Also: Relational Operators (III-3.3 pg.48), SortBy, SortedBy (VI-1.249 pg.267)

VI-1.249 SortBy, SortedBy

syntax
```
SortBy(V:LIST,F:FUNCTION):NULL
SortedBy(L:LIST,F:FUNCTION):LIST

where V is a variable containing a list and F is a boolean-valued
comparison function of two arguments (e.g. representing less than).
```

Description

The first function sorts the elements of the list in V with respect to the comparisons made by F; it overwrites V. The second function, "`Sorted`", returns the list of the sorted elements of L without affecting L, itself. 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 := ["dog","mouse","cat"];
SortedBy(M,Function("ByLength"));
["mouse", "dog", "cat"]
-------------------------------
M;  -- M is not changed
["dog", "mouse", "cat"]
-------------------------------
Sorted(M);  -- the function "Sort" sorts using the default ordering:
            -- in this case, alphabetical order.
["cat", "dog", "mouse"]
-------------------------------
SortBy(M,Function("ByLength"));  -- sort M in place, changing M
M;
["mouse", "dog", "cat"]
-------------------------------
```

See Also: Sort, Sorted (VI-1.248 pg.267)

VI-1.250 Source, ¡¡

syntax

```
Source S:STRING
<< 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.coc";
```

or, equivalently,

```
    << "MyFile.coc";
```

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 (III-7.1 pg.57), Introduction to Packages (III-9.1 pg.67), User Initialization (V-3.1 pg.135)

VI-1.251 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 (VI-1.38 pg.161), Equals (VI-1.60 pg.173)

VI-1.252 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 ::= Q[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 (III-7.1 pg.57), IO.SprintTrunc (VI-1.142 pg.213), Print, PrintLn (VI-1.213 pg.248)

VI-1.253 StarPrint

syntax

```
StarPrint(F:POLY):NULL
StarPrintFold(F:POLY, LineWidth:INT)
```

Description

These functions print the polynomial F with asterisks added to denote multiplications. They may be useful when cutting and pasting from CoCoA to other mathematical software. StarPrint inserts newline characters (only between terms) to avoid lines much longer than about 70 characters. If a different approximate maximum length is desired this may be specified as the second argument to StarPrintFold; a negative value means no line length limit.

example

```
Use R ::= Q[x,y];
F := x^3+2xy-y^2;
StarPrint(F);
1*x^3+2*x*y-1*y^2
-------------------------------
StarPrintFold(F,0); -- this will print one term per line
1*x^3
+2*x*y
-1*y^2
-------------------------------
```

VI-1.254 Starting

syntax

```
Starting(S:STRING):LIST of STRING
```

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

```
Starting("Su");
["SubstPoly", "Support", "Subsets", "SubSet", "Submat", "Sum", "Subst"]
-------------------------------
```

See Also: Other Help (V-2.4 pg.132)

VI-1.255 Submat

syntax

```
Submat(M:LIST or 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);
Mat[
  [3, 4, 5],
  [13, 14, 15]
]
-------------------------------
L := [[1,2,3],[4,5,6]];
Submat(L,[2],[1,3]);
Mat[
  [4, 6]
]
-------------------------------
```

See Also: Introduction to Matrices (IV-7.1 pg.93), Minors (VI-1.176 pg.230)

VI-1.256 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(Set([2,3,3]));
[[ ], [3], [2], [2, 3]]
-------------------------------
```

See Also: Set (VI-1.243 pg.264), IsSubset (VI-1.152 pg.217)

VI-1.257 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 number,
polynomial, or rational function.
```

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`”.

example

```
Use R ::= Q[x,y,z,t];
F := x+y+z+t^2;
Subst(F,x,-2);
t^2 + y + z - 2
-------------------------------
Subst(F,x,z/y);
(yt^2 + y^2 + yz + z)/y
-------------------------------
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,3z-x]];
Subst(xyzt,MySubst);  -- substitute into the function xyzt
-x^2z + 3xz^2
-------------------------------
```

See Also: Eval (VI-1.63 pg.174), Evaluation of Polynomials (IV-9.2 pg.106), Image (VI-1.130 pg.206), QZP, ZPQ (VI-1.217 pg.250), Substitutions (II-2.15 pg.29)

VI-1.258 Support

syntax

```
Support(F:POLY or VECTOR):LIST
```

Description

This function returns the list of terms of F. To get a list of monomials, which includes coefficients, use "`Monomials`".

example

```
Use R ::= Q[x,y];
F := 3x^2-4xy+y^3+3;
Support(F);
[y^3, x^2, xy, 1]
-------------------------------
Monomials(F);
[y^3, 3x^2, -4xy, 3]
-------------------------------
Support(Vector(x^2y,x^3-3y^2,34));
[Vector(0, x^3, 0), Vector(x^2y, 0, 0), Vector(0, y^2, 0), Vector(0, 0, 1)]
-------------------------------
```

See Also: Coefficients (VI-1.27 pg.156), Monomials (VI-1.181 pg.233)

VI-1.259 Sylvester

syntax

```
Sylvester(F:POLY,G:POLY,X:INDET)
```

Description

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 ::= Q[p,q,x];
F := x^3+px-q; G := Der(F,x);
Sylvester(F,G,x);
Mat[
  [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 (VI-1.231 pg.257)

VI-1.260 Syz

syntax

```
Syz(L:LIST of POLY):MODULE
Syz(L:LIST of VECTOR):MODULE
Syz(M:IDEAL or MODULE, Index:INT):MODULE
```

Description

In the first two forms this function computes the syzygy module of a list of polynomials or vectors. 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.

The coefficient ring must be a field.

example

```
Use R ::= Q[x,y,z];
Syz([x^2-y,xy-z,xy]);
Module([0, xy, -xy + z], [z, x^2 - y, -x^2 + y], [yz, -y^2, y^2 - xz],
[xy, 0, -x^2 + y])
-------------------------------
I := Ideal(x^2-yz, xy-z^2, xyz);
Syz(I,0);
Module([x^2 - yz], [xy - z^2], [xyz])
-------------------------------
Syz(I,1);
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^2(-6) --> R(-4)(+)R^3(-5) --> R^2(-2)(+)R(-3)
-------------------------------

For fine control and monitoring of Groebner basis calculations, see
``The Interactive Groebner Framework'' (\ref{The Interactive Groebner Framework} pg.\pageref{The Interact
```

See Also: Introduction to Groebner Bases in CoCoA (IV-13.1 pg.117)

VI-1.261 SyzMinGens

syntax

```
SyzMinGens:  FUNCTION ELIMINATED
```

Description

The SyzMinGens function has been removed.

See Also: Syz (VI-1.260 pg.273), SyzOfGens (VI-1.262 pg.274)

VI-1.262 SyzOfGens

syntax
```
SyzOfGens(M:IDEAL, MODULE, or TAGGED(Quotient)):MODULE
```

Description

If M is an ideal or module, this function calculates the syzygy module for the given set of generators of M. If M is a quotient of the current 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.

The coefficient ring must be a field.

example
```
Use R ::= Q[x,y];
I := Ideal(x,y,x+y);
SyzOfGens(I);
Module([1, 1, -1], [y, -x, 0])
-------------------------------
```

See Also: Syz (VI-1.260 pg.273), SyzMinGens (VI-1.261 pg.273)

VI-1.263 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("MyTag")
-------------------------------
Tag(L);
MyTag
-------------------------------
```

See Also: Tagged Printing (III-7.6 pg.59)

VI-1.264 Tagged, Untagged, @

syntax
```
Tagged(E:OBJECT,S:STRING):TAGGED(S)
Untagged(E:TAGGED_OBJECT):UNTAGGED_OBJECT
@E:TAGGED_OBJECT:UNTAGGED_OBJECT
```

Description

The first function returns the object E, tagged with the string S. The second strips E of its tag, if any. The "*at sign*" can also be used to untag an object: @E is equivalent to Untagged(E). These functions 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
-------------------------------
Type(@M);
LIST
-------------------------------
```

See Also: Tagged Printing (III-7.6 pg.59)

VI-1.265 Tail

syntax
```
Tail(L:LIST):OBJECT
```

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 (VI-1.68 pg.177), Head (VI-1.114 pg.197), Last (VI-1.156 pg.219)

VI-1.266 TensorMat

syntax
```
TensorMat(M:Mat, N:Mat):MAT
```

Description

This function returns the tensor product of two matrices.

example
```
Use R ::= Q[x,y,z,w];
TensorMat(Mat([[1,-1],[2,-2],[3,-3]]),Mat([[x,y],[z,w]]));
Mat[
  [x, y, -x, -y],
  [z, w, -z, -w],
  [2x, 2y, -2x, -2y],
  [2z, 2w, -2z, -2w],
  [3x, 3y, -3x, -3y],
  [3z, 3w, -3z, -3w]
]
-------------------------------
```

VI-1.267 Toric

syntax
```
Toric(L:LIST of BINOMIAL):IDEAL
Toric(L:LIST of BINOMIAL,X:LIST of INDETS):IDEAL
Toric(M:MAT or LIST of LIST):IDEAL

where M is a matrix of integers with no zero column.  Elements of L
must be homogeneous (w.r.t. the first row of the weights matrix).
```

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 “*EATI*”. The first two examples below are motivated by B. Sturmfels, “*Groebner Bases and Convex Polytopes,*” Chapter 6, p. 51. They count the number of homogeneous primitive partition identities of degrees 8 and 9.

example
```
Use Q[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(HPPI8, [x[1],y[2]]);
Len(BL.Gens);
340
-------------------------------
Use Q[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(HPPI9, [x[1],y[2]]);
Len(BL.Gens);
-------------------------------
798
-------------------------------
Use R ::= Q[x,y,z,w];
Toric(Ideal(xz-y^2, xw-yz));
Ideal(-y^2 + xz, -yz + xw, z^2 - yw)
-------------------------------
Toric([xz-y^2, xw-yz]);
Ideal(-y^2 + xz, -yz + xw, z^2 - yw)
-------------------------------
Toric([xz-y^2, xw-yz], [y]);
Ideal(-y^2 + xz, -yz + xw, z^2 - yw)
-------------------------------
```

```
Use R ::= Q[x,y,z];
Toric([[1,3,2],[3,4,8]]);
Ideal(-x^16 + y^2z^5)
-------------------------------
Toric(Mat([[1,3,2],[3,4,8]]));
Ideal(-x^16 + y^2z^5)
-------------------------------
```

See Also: Toric.CheckInput (VI-1.268 pg.277)

VI-1.268 Toric.CheckInput

syntax

```
Toric.CheckInput(E:OBJECT):BOOL
Toric.CheckInput(E:OBJECT,X:LIST):BOOL
```

Description

This function checks if E or (E,X) is suitable input for "`Toric`". Thus, E should be either a list of homogeneous binomials (without coefficients) or a matrix of non-negative integers. In the former case, X must be a list of indeterminates (in the latter, X would be ignored by "`Toric`" anyway).

example

```
Use R ::= Q[x,y,z];
Toric.CheckInput([[1,2,3,4],[4,5,6,7]]);
TRUE
-------------------------------
Toric.CheckInput([[-1,2],[3,4]]);
ERROR: entries must be non-negative integers
CONTEXT: Return(Error(Toric_IntMatrix))
-------------------------------
Toric.CheckInput([xy-z^2,x^3-y^2z]);
TRUE
-------------------------------
Toric.CheckInput([3xy-z^2,x^3-y^2z]); -- the binomials should not
                                      -- have coefficients
ERROR: generators must be of type: power-product - power-product
CONTEXT: Return(Error(Toric_PP))
-------------------------------
Toric.CheckInput([xy-z^2,x^3-y^2z],[x]);
TRUE
-------------------------------
```

See Also: Toric (VI-1.267 pg.276)

VI-1.269 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;
Mat[
  [1, 2, 3],
  [4, 5, 6]
]
-------------------------------
Transposed(M);
Mat[
  [1, 4],
  [2, 5],
  [3, 6]
]
-------------------------------
```

VI-1.270 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: ¿¡ (VI-1.2 pg.141)

VI-1.271 Type

syntax

```
Type(E:OBJECT):TYPE
```

Description

This function returns the data type of E. The function “`Types`” returns the list of CoCoA data types.

example

```
Define CollectInts(L)
  Result := [];
  Foreach X In L Do
    If Type(X) = INT Then Append(Result,X) EndIf
  EndForeach;
  Return Result
EndDefine;

CollectInts([1,"a",2,"b",3,"c"]);
[1, 2, 3]
-------------------------------
Type(Type(INT));  -- Type returns a value of type TYPE
TYPE
-------------------------------
Types();
[NULL, BOOL, STRING, TYPE, ERROR, RECORD, DEVICE, INT, RAT, ZMOD,
POLY, RATFUN, VECTOR, IDEAL, MODULE, MAT, LIST, RING, TAGGED(""),
FUNCTION]
-------------------------------
```

See Also: Data Types (III-2.6 pg.44), Types (VI-1.273 pg.279)

VI-1.272 TypeOfCoeffs

syntax

```
TypeOfCoeffs():TYPE
```

Description

This function returns the type of the coefficients of the current ring.

example

```
Use R ::= Q[x,y,z];
TypeOfCoeffs();
RAT
-------------------------------
Use S ::= Z/(2)[t];
TypeOfCoeffs();
ZMOD
-------------------------------
```

See Also: Characteristic (VI-1.21 pg.153), Coefficients (VI-1.27 pg.156), CurrentRing (VI-1.37 pg.160), Indets (VI-1.136 pg.209)

VI-1.273 Types

syntax

```
Types()
```

Description

This function lists all CoCoA data types.

example

```
Types();
[NULL, BOOL, STRING, TYPE, ERROR, RECORD, DEVICE, INT, RAT, ZMOD,
POLY, RATFUN, VECTOR, IDEAL, MODULE, MAT, LIST, RING, TAGGED(""),
FUNCTION]
-------------------------------
```

See Also: Data Types (III-2.6 pg.44), Type (VI-1.271 pg.278)

VI-1.274 Use

syntax

```
Use N

where N is either the identifier of an existing ring or a ring
itself.
```

Description

"`Use`" is the command for making a ring active, i.e. for making a ring the current ring. The command
Use N ::= E;
where E is a ring, is a shorthand for
N ::= E; Use N;

example

```
Use S ::= Q[x,y,z];
RingEnv();
S
```

```
-------------------------------
T::= Z/(3)[a,b];
Use T;
RingEnv();
T
-------------------------------
Use Q[u];  -- note that "Use" can be used w/out a ring identifier
RingEnv();
CurrentRingEnv
-------------------------------
CurrentRing();
Q[u]
-------------------------------
```

See Also: Accessing Other Rings (IV-8.11 pg.100), Using (VI-1.275 pg.280)

VI-1.275 Using

syntax

```
Using R Do C EndUsing

where R is the identifier for a ring and C is a sequence of commands.
```

Description

Suppose S is the current ring and R is another ring, then
Using R Do C; EndUsing;
is equivalent to
Use R; C; Use S;

example

```
Use S ::= Q[x,y];          -- the current ring is S
R ::= Q[a,b,c];             -- another ring
Using R Do Indets() EndUsing;
[a, b, c]
-------------------------------

Note: ‘‘\verb&Using Q[a,b] Do ...&’’ is not proper syntax and will produce an
error.
```

See Also: Accessing Other Rings (IV-8.11 pg.100), Use (VI-1.274 pg.279)

VI-1.276 Var

syntax

```
Var X
Var(X)
Var(S:STRING)

where X is the identifier of a CoCoA variable.
```

Description

In the first and second form “`Var`” is used as a formal parameter to a user-defined function. It is used to pass a variable—not its value—to the user-defined function. The following example should make the difference clear.

example

```
Define CallByRef(Var L )  -- "call by reference": The variable referred
  L := "new value";       -- to by L is changed.
EndDefine;
M := "old value";
CallByRef(M);
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
-------------------------------
L;
old value.
-------------------------------
```

In the third form, Var(S), references the value of the variable or ring whose identifier is S:

example

```
Var("a string") := 6;
Var("a string");
6
-------------------------------
P := Record[Name = "test", Value = 1];
X := "Name";
P.Var(X);
test
-------------------------------
Var("myring") ::= Q[a,b];
Var("myring");
Q[a,b]
-------------------------------
Using Var("myring") Do (a+b)^2 EndUsing;
a^2 + 2ab + b^2
-------------------------------
```

See Also: Define (VI-1.41 pg.162)

VI-1.277 Vector

syntax

```
Vector(F_1:POLY,...,F_n:POLY):VECTOR
Vector(L:LIST):VECTOR

where L is a list of polynomials.
```

Description

The first form returns the vector with components “`F_1,...,F_n`”; the second form returns the vector whose components are the components of the list L.

example

```
Use R ::= Q[x];
V := Vector(1,x,x^2);
```

```
Type(V);
VECTOR
Vector([1+x,x,x^2]);
Vector(x + 1, x, x^2)
-------------------------------
```

VI-1.278 WeightsList

syntax

```
WeightsList():LIST
```

Description

This function returns the first row of the weights matrix for the current ring as a list.

example

```
Use R ::= Q[t,x,y,z];
WeightsList();
[1, 1, 1, 1]
-------------------------------
Use R ::= Q[t,x,y,z], Weights(1,3,5,2);
WeightsList();
[1, 3, 5, 2]
-------------------------------
Use R ::= Q[x,y], Weights(Mat([[1,2],[3,4],[5,6]]));
WeightsList();
[1, 2]
-------------------------------
```

See Also: Deg (VI-1.43 pg.164), MDeg (VI-1.172 pg.228), Weights Modifier (IV-8.5 pg.97), WeightsMatrix (VI-1.279 pg.282)

VI-1.279 WeightsMatrix

syntax

```
WeightsMatrix():MAT
```

Description

This function returns the weights matrix for the current ring.

example

```
Use R ::= Q[x,y], Weights(Mat([[1,2],[3,4],[5,6]]));
WeightsMatrix();
Mat[
  [1, 2],
  [3, 4],
  [5, 6]
]
-------------------------------
MDeg(y);
[2, 4, 6]
-------------------------------
WeightsList();  -- the first row of the weights matrix
[1, 2]
-------------------------------
```

See Also: Deg (VI-1.43 pg.164), MDeg (VI-1.172 pg.228), Weights Modifier (IV-8.5 pg.97), WeightsList (VI-1.278 pg.282)

VI-1.280 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 (VI-1.73 pg.179), Foreach (VI-1.74 pg.180), Repeat (VI-1.227 pg.255)

VI-1.281 WithoutNth

syntax

```
WithoutNth(L:LIST,N:INT):NULL
```

Description

This function returns the list obtained by removing the N-th component of the list L. The list L is not affected (as opposed to the command “`Remove`”).

example

```
L := [1,2,3,4,5];
WithoutNth(L,3);
[1, 2, 4, 5]
-------------------------------
```

See Also: Insert, Remove (VI-1.137 pg.210)

VI-1.282 WLog

syntax

```
WLog(F:POLY):LIST of INT
```

Description

This function returns the weighted list of exponents of the leading term of F, as determined by the first row of the weights matrix. Thus, if all the weights are 1, this function returns the same thing as “`Log(F)`”.

example

```
Use R ::= Q[x,y];
F := x^2-y;
WLog(F);
```

```
[2, 0]
-------------------------------
Use R ::= Q[x,y],Weights(2,3);
F := x^2-y;
WLog(F);
[4, 0]
-------------------------------
```

See Also: Log (VI-1.164 pg.223)