S M P™

a symbolic manipulation program

Chris A. Cole Stephen Wolfram

and

Geoffrey C. Fox
Jeffrey M. Greif
Eric D. Mjolsness
Larry J. Romans
Timothy Shaw
Anthony E. Terrano

version one

July 1981

California Institute of Technology

S M P™

a symbolic manipulation program

Chris A. Cole Stephen Wolfram

and

Geoffrey C. Fox
Jeffrey M. Greif
Eric D. Mjolsness
Larry J. Romans
Timothy Shaw
Anthony E. Terrano

version one

July 1981

California Institute of Technology

SMP Handbook

Preface

This handbook contains:

Implementation notes

Details for particular SMP installations.

Summary

Complete concise description of facilities available in SMP. Outline of SMP library providing additional facilities.
(Available separately).

Primer

Pedagogical discussion of basic features of SMP. List of common difficulties. Glossary of terms.

Reference manual

Description of facilities available in SMP, together with illustrative examples. Index of topics and SMP facilities.

SMP library

Topic and section directory. Text and description of external files in library.

"SMP implementation guide" is also available.

Updates to this handbook will appear periodically in the "SMP News". The next full edition is scheduled for December 1981.

Documents which report work for which SMP was used should cite this document as:
C.Cole, S.Wolfram et al., "SMP Handbook", Caltech 1981.

Short history of SMP

The original motivation for the construction of SMP came from the desire to evaluate complicated Feynman diagrams in theoretical high-energy physics. Experience with existing systems had indicated that they became inadequate for the very large expressions to be manipulated, and that they lacked the generality necessary to handle easily the mathematical constructs required.

In November 1979, design work on SMP was started by C.Cole and S.Wolfram. The first specifications for SMP were drawn up by S.Wolfram in December 1979, and it became clear that a very general symbolic manipulation program could be written.

Design work continued intermittantly between January and March 1980. Design meetings during this period were attended by C.Cole, G.Fox, D.Mitchell, R.Pike, H.Redelmeier, A.Terrano and S.Wolfram. The important decision that SMP should be written in the C language, rather than in LISP, was made at this stage. C was favoured primarily because of the greater flexibility in internal representation which it afforded, because of its potential portability and because the task of coding was expected to be easier.

In April 1980, the Caltech High-Energy Physics group obtained a VAX-11/780 computer (mainly through the extensive efforts of G.Fox). C.Cole started the coding of SMP in C, and by the beginning of June 1980, a very basic system was operational.

In the period June 1980 to August 1980, about 30000 lines of C code were written, and by the end of August a very preliminary version of SMP was completed. In July 1981, a second VAX 11/780 was obtained to continue SMP development and use.

The August 1981 version of SMP contains about 80000 lines of C code, corresponding to 900000 bytes of compiled code.

The basic structure of SMP was designed almost exclusively by C.Cole and S.Wolfram. The majority of the internal code was also written by them. The top-level form of SMP is mostly due to S.Wolfram. A.Terrano assisted in the implementation of many of the basic modules (pattern matcher, substitutor, domain controller) and is responsible for much of the mathematical functions package. L.Romans wrote `Ex`, `Dist`, `Powdist`, `G`, the basic forms of `Fac`, `Pf`, `Int` and `Pgcd`, and the three-dimensional plotting routines. T.Shaw designed and implemented the printing routines, and contributed some ideas to the basic structure of SMP. J.Greif wrote the matrix manipulation and equation solving programs. G.Fox wrote an efficient univariate factorization program. E.Mjolsness was responsible for the series approximations package.

Most of this handbook was written by S.Wolfram; the "Primer" was written in collaboration with A.Terrano. Most of the existing external files were written by S.Wolfram; those in [8] are due to A.Terrano.

Acknowledgements

We are grateful to many people for suggestions, assistance and encouragement. Among these were
B.Barish (Caltech), M.Bartelt (Caltech), K.Ellis (CERN), R.Fateman (Berkeley), R.Feynman (Caltech), W.Furmanski (Cracow), M.Gell-Mann (Caltech), W.Gosper (Xerox PARC), T.Gottschalk (Caltech), P.Gladstone (Rutherford), M.Griss (Utah), Y.Gursel (Caltech), J.Harvey (Caltech), A.Hearn (Utah), K.Kolbig (CERN), B.Lautrup (NBI,Copenhagen), M.Minsky (MIT), D.Mitchell (Caltech), J.Moses (MIT), E.Ng (JPL), A.Norman (Cambridge), S.Otto (Caltech), R.Pike (Caltech/Bell), D.Politzer (Caltech), P.Ramond (Caltech/Florida), H.Redelmeier (Toronto), D.Reiss (Caltech), T.Robinson (Oxford), D.Ross (Caltech/Southampton), R.Sjogren (Caltech), M.Veltman (Utrecht), N.Wilson (Caltech), L.Yaffe (Caltech).

We have discussed SMP with many other people, and have received numerous valuable suggestions. These people are too numerous to list; we thank them all.

We are particularly grateful to T.Gottschalk and N.Wilson for their help and understanding.

R.Hughes and S.Rabin contributed early versions of computer code. M.Abeln, J.Byrne, C.de Bardeleben, J.Raemsch and D.Sherwood made important contributions in the latter stages of the project.

We have benefitted from use of previous symbolic manipulation programs: particularly MACSYMA, REDUCE, SCHOONSCHIP and ASHMEDAI.

The development of SMP was funded by the California Institute of Technology.

Implementation Notes

(To be filled in separately for each installation)

[Facilities not provided in particular installation are left blank]

Installation:

For assistance, contact:

Connecting to computer/logging in:

To initiate an SMP job:

To terminate an SMP job:

Input

⊂*newline*⊃
⊂*character delete*⊃
⊂*line delete*⊃
⊂*tab*⊃
Special characters:

Real-time interrupt characters [1.4]:
⊂*termination character*⊃
⊂*quit interrupt*⊃
⊂*break interrupt*⊃
⊂*status interrupt*⊃

Output

To initialize terminal in SMP type: (terminal classes for Open [10.3] given in parentheses)

Graphics output available on:

To clear screen (without erasing graphics) type:

To continue printing after end-of-page pause on video terminal type:

External files

Modifications to standard external file names:

Special external files:

Procedure for creating external files:

Record file [1.3]:

Procedure for saving record file:

Procedure for reading record file:

(a) Outside an SMP job:

(b) From within an SMP job:

Procedure for printing record file:

(a) Outside an SMP job:

(b) From within an SMP job:

Default "user" file directory [A.2]:

Library file directory:

Procedure for initiating SMP job with initialization files:

Default Initialization file [A.4]:

External programs

Syntax for pre/post-processors on standard input/output in Run [10.5,A.3]:

Option for compiler to include SMPIO library:

Information on SMPIO library:

Codes for Cons [10.9]:

Implementation dependent features

Tick (check) indicates available as described in Reference Manual; blank indicates not available.

monitor escape

parallel/asynchronous processing

binary files

Gc

Miscellaneous

Procedure for initiating batch SMP job:

Procedure for determining status of batch SMP job:

Total number of memory blocks [10.8] available:

Actual CPU time corresponding to one click [10.8]:

Command interpreter invoked by monitor escape [1.6]:

Editor invoked by \e in edit mode [1.7]:

Information and assistance

To access central file of SMP news and information type:

Procedure for sending message to obtain on-line assistance:

(a) Outside an SMP job:

(b) From within an SMP job:

Procedure for sending message to central SMP report file:

(a) Outside an SMP job:

(b) From within an SMP job:

If a suspected SMP error is encountered, first consult the appendix to the SMP Primer, then use any on-line assistance available. If no resolution is apparent, fill out a copy of "Report of suspected SMP error" (given as following page of Handbook), or contact:

Report of Suspected SMP Error

Originator:

Installation:

Date:

SMP version:

What priority should this report be given?

Is the suspected error reproducible, in the sense that identical input yields identical output?

Try to find the simplest case under which the error occurs.

Look in the "Common Difficulties" appendix to the SMP Primer.

Attach a printout of record file (`smp.out`), with suspected error marked.

Enclose a self-addressed envelope.

Circle type of error:

1. SMP_INTERNAL_ERROR was printed.
   Additional messages:
   Did job terminate after error?
   (If so, send tape of core dump if possible).
   Was the error recorded before, after or during printout of the #0 line?
2. SMP apparently entered an infinite loop.
   Was a processing impasse recorded?
   Did quit interrupt stop it?
   If not, how was it stopped?
   How rapidly did memory usage (as printed after status interrupt) increase
   (a) Before quit interrupt:
   (b) After quit interrupt:
3. A built-in projection in SMP gave an unexpected result.
   Give section of SMP summary in which facility used is described:
   Was result given mathematically incorrect?
   Could error be in documentation?
4. An external file yielded an expected result
   Give name of external file and object used:
   Enclose listing of external file, marking parts used, together with any apparent errors.
   If the required result could be obtained by generalization of existing definitions, enclose suggested additions.
5. SMP crashed the local operating system.
6. Other.
   Attach details.

Further comments:

Signature of user contact:

Mail completed form and enclosures to:
SMP Error Report, 452-48, Caltech, Pasadena CA 91125, USA.

SMP Summary

Stephen Wolfram

Introduction

This summary provides an essentially complete but concise description of the standard facilities available in SMP. The same text is given, supplemented by examples, in the "Reference Manual". The "SMP primer" offers a more pedagogical but incomplete treatment.

This summary describes all standard system-defined projections [2.3] in SMP. Many enhancements and additional facilities are provided in the "SMP Library". Directories of this library are given in the summary; its complete contents are given in a separate section.

The procedure for initiating an SMP job should be described in the "Implementation Notes", together with other implementation-dependent features.

CONTENTS

0. Conventions

Text printed in `this font` represents literal SMP input or output, to be typed as it appears. Text in *italics* stands for SMP input or output expressions. Arbitrary characters or textual forms are given as ⊂*textual form*⊃ The actual characters corresponding to these textual forms in particular implementations should be given in the "Implementation Notes".

In descriptions of system-defined projections [2.3], the following notations for filters [2.3] are used

filt	Compulsory filter.
filt	Filter used in an unsimplified or partially simplified form. • [3.5, 3.6] • `Nosmp` [4]
(*filt*:*val*)	Optional filter assumed to have value *val* if omitted or given as `Null` (a blank [2.10]). An optional filter may be omitted entirely only when it would appear after the last filter explicitly specified in a projection.
f1, *f2*, ...	Sequence containing any number of similar filters.
{*f1*, *f2*, ...}	Filter to be given as a list of expressions.
⟨*filt*⟩	Filter to be given either as a single expression or as a list of expressions.
⟨*f1*, *f2*, ...⟩	Filter or sequence of filters to be given either as single expressions or as lists of expressions.

Properties carried by projections are indicated by
`<prop1, prop2,...>`

References to sections in the text of this summary are given as [⊂*number*⊃ References to additional or related material in the summary are indicated by •. References to external files or objects defined in them are indicated with ▫.

1. Basic system operation

1.1 Input and output

In standard mode the prompt for the *i*th input line is `#I [i]: :`
In subsidiary mode, the prompt is `%I [i]: :`

Input is terminated by a ⊂*newline*⊃ ⊂*delete character*⊃ may be used in unfinished input lines. All ⊂*tab*⊃ and unnecessary [2.10] spaces are ignored (unless they appear in quoted strings [2.2]). Input may be continued for several lines by placing \ (backslash) at the end of each intermediate line. If no text is entered before the terminating newline, the input is considered null, and the prompt is reissued.

The result generated by processing an input line is usually printed. No output is printed if ; (semicolon) is placed at the end of the input expression [6.3](or in general if the output expression is `Null` [2.2]).

Input of expressions with ambiguous syntax [2] yields a message on the nature of the ambiguity, and causes edit mode [1.7] to be entered. The input text is placed in the edit buffer, with the cursor positioned under the point at which the ambiguity was detected.

- [2.10]

Printed output is given in a two-dimensional format based on standard mathematical notation.

- [2.12]
- `Lpr` [10.1]

1.2 Global objects

`%`	The last expression (other than `Null` [2.2]) generated.
`#I [i]`	The *i*th (unsimplified) input expression. `<Ldist>`
`#O [i]`	The *i*th output expression. `<Ldist>`
`#T [i]`	The approximate time (in clicks [10.8]) required to generate `#O [i]`. `<Ldist>` • `Time` [10.8] • `Clock` [10.10]
`@i`	Equivalent to `#O [i]` `<Ldist>`

- `%%, %I, %O, %T` [1.8]

Any values assigned [3.2] to the special symbols `Pre` and `Post` are taken as templates [2.7] and applied respectively as a "preprocessor" on each input expression and a "postprocessor" on each output expression.

1.3 External files and job recording

External files are input by *<file*

- [10.3]

All input and output expressions are entered into a record file (usually named `smp.out`). (Implementation dependent)

• Open, Output [10.3]

1.4 Termination and real-time interrupts

(Implementation dependent)

⊂*input termination character*⊃
signifies termination of the current procedure. In standard and edit mode, it causes termination of the present job; in subsidiary mode [1.8] it results in return to the previous mode.
• Ret [6.3]
• Exit [10.5]

⊂*quit interrupt*⊃
attempts to terminate current processing or printing. Subsequent references to incompletely processed expressions cause their simplification to be completed.

⊂*break interrupt*⊃
causes any processing to be suspended and initiates an interactive subsidiary procedure [1.8].
• Proc [6.3]

⊂*status interrupt*⊃
prints the status of processing, including a stack of the last few projections simplified.
• State [10.8]
• Trace [4]

1.5 Processing impasses

If the complete processing of an input expression appears to require infinite time or memory space, its processing is suspended, and the user is consulted on the extent to which it should be continued. A response of Inf causes no further consultation. 0 results in immediate termination.

In implementations with finite available memory space, processing is suspended if only a small fraction of the memory remains. Memory is reclaimed when processing is complete.

1.6 Monitor escapes

(Implementation dependent)

! ⊂*monitor command*⊃
given directly at an input prompt executes the specified monitor (shell) command; the original input prompt is then repeated.
• Run [10.5]

1.7 Edit mode

In edit mode, one line of text is treated at a time. The editing of each line proceeds by a repetition of two steps:

1. The present form of the line is printed.
2. Following the prompt <edit> any editing commands for the line are entered. The commands are terminated by ⊂*newline*⊃ ⊂*tab*⊃ and ⊂*character delete*⊃ may be used for positioning in local editing.

If no editing commands are given in step 2 (the terminating newline is entered directly after the <edit> prompt), then the present line is left unchanged. If further lines of text exist, the editor moves to the next line; otherwise, edit mode is exited, and the edited text is returned as an input expression. If at the exit from edit mode,

no editing has actually been performed, the text is discarded, leaving a null line.

Two types of editing may be performed:

Local editing, in which individual characters in the edit command affect the characters appearing directly above them in the present line.

Global editing, in which an edit command prefaced by \ affects the whole present line or the complete edit buffer.

Local editing

#	Delete character above.
⊂*space*⊃ or ⊂*tab*⊃	Leave character(s) above unchanged.
^⊂*text*⊃	Insert ⊂*text*⊃ terminated by a space or tab (not appearing between " " [2.2]), into the present line before the character above.
\d	Delete the remainder of the present line.
⊂*other character*⊃	Replace character above by character given.

Global editing

\p	Print the complete text in the edit buffer; then return to the present line.
\q	Exit from edit mode, discarding the edited expression.
\n	Leave the present line unaltered, and move *n* lines forward. \- and \+ move one line forward and backward respectively.
\u	Undo the changes effected in the present line by the previous edit command.
\! ⊂*monitor command*⊃	Execute specified monitor command [1.6]
\m	Display levels of parentheses, braces and brackets in the present line.
\mg	Display levels of parentheses, braces and brackets in the complete text.
\s/⊂*e1*⊃/⊂*e2*⊃	Textually substitute ⊂*e2*⊃ for the character string ⊂*e1*⊃ throughout the present line. (The delimiter here given as / may be replaced by any character.)
\s*n*/⊂*e1*⊃/⊂*e2*⊃	Textually substitute for the first *n* occurrences of ⊂*e1*⊃ on or following the present line.
\sg/⊂*e1*⊃/⊂*e2*⊃	Textually substitute for ⊂*e1*⊃ throughout the text.
\e	Invoke external editor on complete text, and return modified text. (Implementation dependent)

• Ed, Edh [10.6]

1.8 Procedures and subsidiary mode

Procedures consist of sequences of expressions ("segments") to be simplified in turn. Segments in procedures may either be provided as successive filters of a Proc projection [6.3], or may be entered "interactively" on successive input lines. The #I[*i*] are the segments of the outermost procedure, and are entered in standard input mode. Interactive subsidiary procedures are initiated by ⊂*break interrupt*⊃ or Proc[] [6.3], and terminated with ⊂*input termination character*⊃ or Ret[] [6.3]. Their segments are entered in subsidiary input mode. Unless specified otherwise [6.3], all

expressions may be accessed and affected in any procedure. Objects local to a single subsidiary procedure are

%% The last expression (other than Null [2.2]) generated. The last value assigned to %% in a procedure is used as the value of the complete procedure.

%I [*i*] The *i*th (unsimplified) segment.
<Ldist>

%O [*i*] The value of the *i*th segment.
<Ldist>

%T [*i*] The approximate time (in clicks [10.8]) required to generate %O [*i*].
<Ldist>

1.9 Information and elaboration

?*name* or **Info [*name*]**
prints any information on *name* given in this summary.

? or **Info []**
initiates interactive information mode.

Elaborations provide commentary and interactive assistance. Each elaboration printed is assigned a unique name of the form *#n⊂name⊃([i, j, ..])*. The integer *n* specifies the "class" of the elaboration:

1 Prompt for respelling of unknown names.

2 Unexpected filters or undefined operations reported.

3 Unusual filters or operations reported.

4 Intermediate operations described.

On [{*elab1, elab2, ...*}]
causes the elaborations *elab1,elab2,..* to be printed when appropriate. *elabi* may be an integer corresponding to a class of elaborations.

On []
turns on all elaborations.

Off [{*elab1, elab2, ...*}]
stops printing of elaborations or classes of elaborations specified by *elab1,elab2,...*

Off []
turns off all elaborations.

1.10 External programs and program construction

(Implementation dependent)

External programs may be run with SMP output expressions as input using Run [10.4]; their output may be taken as SMP input.

External programs may be constructed from SMP expressions using Cons [10.9] and

may be run in a compiled form.

1.11 Parallel processing

(Implementation dependent)

SMP operations may be performed asynchronously and in parallel using `Fork` and `Wait` [10.10].

Values in the list `Rti` give expressions to be simplified on receipt of real-time interrupts.

Future implementations may employ automatic parallel processing: filters in projections which do not carry property `Ser` are then simplified in parallel rather than sequentially.

2. Syntax

2.1 Numbers

Numbers input with no decimal point are taken as exact integers.

Floating point numbers may be entered in the form *x**^*p* representing $x*10^p$.

Numbers are output if possible in integer or rational form, except when the corresponding input expression contains explicit floating point numbers or N projections [3.4]. Output floating point numbers are given to the precision specified in input N projections, or, by default, to 6 significant figures.

Numbers are by default treated to a finite precision (fractional accuracy of order 10^-13).

Further numerical constructs (which may be used in any numerical operations) are:

A [*x*, (*p*:0)] Arbitrary magnitude number $x*10^p$. Floating point numbers are usually converted to this form when the modulus of their exponent exceeds 10.

B [*nk*, . . ., *n2*, *n1*, *n0*] Arbitrary length integer $n0 + n1*10^4 + n2*10^8 + \cdots$ generated when integers with more than 10 digits are input.

F [*n1*, *n2*, . . ., (*expt*:0), (*dig*:6)] Floating point number $(n1*10^{-4} + n2*10^{-8} + \cdots)*10^{expt}$ with *dig* significant digits. F projections are generated if possible when the precision specified in an N projection exceeds 10. Expressions involving several F projections are treated to the numerical accuracy of the least precise F given.

Err [*x*, *dx*] Number *x* with one-standard-deviation error *dx* ($x \pm dx$). Errors are combined assuming statistical independence.

Cx [*x*, *y*] Complex number $x + I\ y$. Automatically generated when required.

2.2 Symbols

Symbols are the basic objects of SMP. They may be assigned values [3.2] and properties [4].

Symbol names may be

Strings of arbitrary length containing only alphanumeric characters, together with #, $ and % and not starting with numeric characters.

Arbitrary character strings (possibly containing control characters) enclosed between " ". The " " are not included in the symbol name; they are used on output only when necessary.
- [10.6]

Symbols whose names have the following forms are taken to have special characteristics (*ccc* represents any valid symbol name):

$*ccc* Generic symbol (representing an arbitrary expression [2.6])
- Gen [4, 2.6]

$$*ccc* Multi-generic symbol (representing an arbitrary sequence of expressions [2.6]).
- Mgen [4]

##ccc Chameleonic symbol (whose name changes whenever it is evaluated [2.8]).
• **Cham** [4]

No values may be assigned to symbols of these types.

• **%** [1.2]

The following naming conventions are used (*C* denotes any upper case alphabetic character):

Cccc System-defined symbol.

#ccc Internally-generated symbol.

%ccc Symbol used locally within a procedure [1.8,6.3]

Some special system-defined symbols are:

Null Input and output as a blank [2.10]. When the value assigned for a projection would simplify to **Null**, the projection is left unevaluated [3.1].

Inf Infinity. Used primarily to specify indefinite continuation of a process, rather than as a signal for mathematical infinities.

I The imaginary unit.

• **%, #I, #0, #T** [1.2]
• **%%, %I, %0, %T** [1.8]
• **Pi, E, Euler,...** [8.4]

2.3 Projections

Projections specify parts of general structures. They are analogous to array subscriptings or function calls.

In for example the projection **f[x,y]** the symbol **f** is termed the "projector", and **x,y** its "filters". These filters are used successively or together to select a part in the value of **f**.

The projector *f* in a projection is maintained in an unsimplified "held" [3.5] form; only its projections are simplified.

f[x,y] is equivalent to **f[x][y]** unless **f** carries the property **Tier** [4].
• **Proj** [7.3]

Common system-defined projections may be input in special forms [2.10].

Filters for system-defined projections input as **Null** [2.2] (a blank [2.10]) are taken to have their default values.

[*x1,x2,...*] or **Np[*x1,x2,...*]**
is a special system-defined null projection representing a sequence of expressions. When **[*x1,x2,...*]** appears as a filter in a projection, that filter is replaced by the set of filters *x1,x2,...* Hence **f[x,[y,z],[[w]]]** becomes **f[x,y,z,w]**. Null projections may also be used to specify sets of entries in lists [2.4]. No values may be assigned [3.2] to null projections.
• **Seq, Repl** [7.1]

`*expr* or **Mark[*expr*]**
is used in a variety of cases to designate expressions with special characteristics.
(Prefix form is "backquote" or "grave accent" character.)
• [2.7]

2.4 Lists

Lists are indexed and ordered sets of expressions.

Lists may be input directly in the form {[*index1*] : *value1*, [*index2*] : *value2*, ...}. Entries with delayed rather than immediate values [3.2] are input with :: instead of : .

If the expressions *indexi* are successive integers starting at 1, they may be omitted. The list may then be input as {*value1*, *value2*, ...}. Such lists are termed "contiguous".

• List [7.1]

{} is a zero-length list containing no entries.

List entries may also be specified indirectly through assignments for projections [3.2].

A value in a list may be extracted by a projection [2.3] with its index as a filter.

• Ar [7.1]

2.5 Expressions

Expressions are combinations of symbols, projections and lists.

Parts of expressions are selected by projections [2.3,7.3]. Values in a list are specified by their indices [2.4]. Parts in projections are specified by numerical filters: 0 specifies the projector (which is always in a "held" form [3.5]) and 1,2,3,.. label each of its filters. Numerical coefficients of symbols and projections are specified by the filter -1. The 0 part of a symbol is the symbol itself, without any associated numerical coefficients.

• Nc [7.9]
• Pos, Dis [7.3]

The *nth* "level" in an expression is the set of parts which may be selected by *n* filters (when *n* is a positive integer). The "depth" of an expression is one plus the maximum number of filters necessary to specify any part [7.4]. The -*n*th level in an expression is the set of parts which have depth *n*.

A domain is a set of parts in an expression. Domains may be specified by a parameter *levspec* giving the levels at which its elements may occur (*ni* are positive integers):

n	Levels 0 through *n*.
-*n*	Levels -Inf through -*n*.
{±*n*}	Level ±*n*.
{±*n1*, ±*n2*}	Levels ±*n1* through ±*n2*.
{-*n1*, *n2*}	Levels -Inf through -*n1* and 0 through *n2*.
{*l1*, *l2*, *levcrit*}	Levels specified by {*l1*, *l2*} on which application of the template [2.7] *levcrit* does not yield false [5].

Domains may also be selected by requiring that application of a template [2.7] *domcrit* to any of their parts should yield true [5].

Many system-defined projections may carry filters which select particular domains. A repeat count *rpt* is usually given to specify the number of times an operation is to be performed successively on a particular domain (Inf causes repetition to continue until the domain no longer changes or a processing impasse is reached [1.5]). A parameter *max* is used to determine the total maximum number of operations performed on any domain. The filters *levspec*,*rpt*,*domcrit*,*max* (or some subset of these) constitute a "domain specification". In treatment of a domain containing positive levels, larger subparts of an expression are treated first, following by their progressively

smaller subparts. In a domain containing negative levels, the smallest subparts are treated first. Different parts at the same level are ordered by their "positions" (the sequences of filters necessary to select them). When *rpt* is specified, the subparts appearing in a domain are re-determined each time the operation is performed.

2.6 Patterns

A "pattern" or "generic expression" is an expression containing generic symbols [2.2]. A pattern represents a possibly infinite set of expressions, in which arbitrary expressions replace the generic symbols. Every occurrence of a particular generic symbol in a pattern corresponds to the same expression.

Two patterns are considered equivalent if the sets of expressions which they represent are identical; they are literally equivalent if all their parts are identical.

Determination of literal equivalence takes account of filter reordering [4,7.7] properties of projections, and of the correspondence between numerical coefficients [2.5] and explicit `Mult` projections.

A pattern *p2* "matches" *p1* if it represents a superset of the expressions represented by *p1* (so that *p1* may be obtained by replacing some or all of the generic symbols in *p2*). *p1* is then considered "more specific" than *p2*.

`Match[`*expr2*`,` *expr1*`]`
: yields *0* if *expr2* does not match *expr1*, 1 if *expr2* is equivalent to *expr1*, or a list of replacements for generic symbols in *expr2* necessary to obtain *expr1*.

A pattern *p2* matches *p1* if the simplified form of *p2* after replacement of generic symbols is literally equivalent to *p1*. However, for at least one occurrence of each generic symbol in *p2* the necessary replacement must follow from literal comparison with *p1* (and must thus appear as an explicit part of *p1*).

The value *t* of the property [4] `$`*x*`[Gen]` restricts all occurrences of the generic symbol *x* to match only those expressions on which application of the template *t* yields "true" [5].

pat `_.` *cond* or `Gen[`*pat*`,` *cond*`]`
: represents a pattern equivalent to *pat*, but restricted to match only those expressions for which *cond* is determined to be true [5] after necessary replacements for generic symbols.

Multi-generic symbols [2.2] are a special class of generic symbols which represent sequences of expressions, corresponding to null projections [2.3] of any length. In a projection (or list), the sequence of filters matched by a multi-generic symbol is terminated when all subsequent filters can be otherwise matched. For projections with `Comm` or `Reor` [4] properties, all but one multi-generic symbol is required to match a zero-length sequence of expressions `[]`. In a projection from a projector *f* with property `Flat` [4], a multi-generic symbol matches a sequence of filters corresponding to a further projection from *f:* in the original projection, it is not represented by a single ordinary generic symbol.

2.7 Templates

A template *t* is an expression specifying an action to be taken on a set of expressions {*s1*,*s2*,...}.

The result from an application of *t* depends on its structure:

number or `Null` *t*

pattern	The value of *t* obtained by replacement of generic symbols *di* occurring in it by any corresponding *si*. The association of *si* with *di* is determined by first sorting the *di* into canonical order. Any unpaired *di* are left unreplaced. If any of the *di* are multi-generic symbols, the first is paired with the maximal set of *si*.
`` `u `` or Mark[*u*]	*u*.
other expression	*t*[*s1*, *s2*, ...]

• Ap [7.2]

2.8 Chameleonic expressions

A chameleonic expression is an expression containing chameleonic symbols [2.2]. If a chameleonic expression is assigned as a delayed value [3.2], then each chameleonic symbol which it contains is given a unique new name whenever the value is used.

• Make [10.6]

2.9 Commentary input

Any input (including newlines) between /* and */ is treated as commentary, and is not processed. Comments may be nested.

2.10 Input forms

input form	projection	grouping
(*x*)	(parentheses)	
{*x1, x2, x3, ...*} {[*i1*]:*x1*, [*i2*]:*x2, ...*}	(List) (List)	
*x1**^*x2*	*x1* * 10^*x2*	(*x1**^*x2*)*^*x3*
@*x*	#0[*x*]	
_*x*	Prop[*x*]	
f[*x1, x2, x3*]	(projection)	f[*x1, x2, x3*] (_f[Tier]:0) ((f[*x1*]):[*x2*])[*x3*] (_f[Tier]:1)
[*x1, x2, x3*]	Np[*x1, x2, x3*]	[*x1, x2, x3*]
<*x*	Input[*x*]	
x!! *x*!	Dfctl[*x*] Fctl[*x*]	(*x*!!)!
x1.*x2*.*x3* *x1* ** *x2* ** *x3*	Dot[*x1, x2, x3*] Omult[*x1, x2, x3*]	*x1*.*x2*.*x3* *x1* ** *x2* ** *x3*
x1^*x2*	Pow[*x1, x2*]	*x1*^(*x2*^*x3*)
-*x*	Mult[-1, *x*]	
x1/*x2*	Div[*x1, x2*]	(*x1*/*x2*)/*x3*
x1 * *x2* * *x3*	Mult[*x1, x2, x3*]	*x1* * *x2* * *x3* (see also below)
x1 + *x2* + *x3* *x1* - *x2* + *x3*	Plus[*x1, x2, x3*] Plus[*x1*, -*x2, x3*]	*x1* + *x2* + *x3* *x1* + (-*x2*) + *x3*
x1 = *x2* *x1* ~= *x2* *x1* > *x2* *x1* >= *x2* *x1* < *x2* *x1* <= *x2*	Eq[*x1, x2*] Uneq[*x1, x2*] Gt[*x1, x2*] Ge[*x1, x2*] Gt[*x2, x1*] Ge[*x2, x1*]	See note below See note below See note below See note below See note below See note below
~*x*	Not[*x*]	
x1 & *x2* & *x3*	And[*x1, x2, x3*]	*x1* & *x2* & *x3*
x1 | *x2* | *x3* *x1* || *x2* || *x3*	Or[*x1, x2, x3*] Xor[*x1, x2, x3*]	*x1* | *x2* | *x3* *x1* || *x2* || *x3*
x1 => *x2*	Imp[*x1, x2*]	*x1*=>(*x2*=>*x3*)
x1 _= *x2*	Gen[*x1, x2*]	(*x1* _= *x2*) _= *x3*
x1 .. *x2*	Seq[*x1, x2*]	(*x1* .. *x2*) .. *x3*
x1 : *x2* *x* : *x1* :: *x2* *x1* -> *x2* *x1* --> *x2* *x1* _; *x2* *x1* _*x2* *x1* := *x2* *x1* :-	Set[*x1, x2*] Set[*x*] Setd[*x1, x2*] Rep[*x1, x2*] Repd[*x1, x2*] Preset[*x1, x2*] Tyset[*x1, x2*] Sxset[*x1, x2*] Sxset[*x1*]	*x1* : (*x2* : *x3*) *x1* :: (*x2* :: *x3*) *x1* -> (*x2* -> *x3*) *x1* --> (*x2* --> *x3*) *x1* _; (*x2* _; *x3*) *x1* _(*x2* _*x3*) *x1* := (*x2* := *x3*)
x1; *x2*; *x3* *x1*; *x2*; ... *xn*;	Proc[*x1, x2, x3*] Proc[*x1, x2, ..., xn,*]	*x1*; *x2*; *x3* *x1*; *x2*; ... *xn*;
!*x* '*x* `*x*	(pre-simplification) Hold[*x*] Mark[*x*]	
?*x*	Info[*x*]	

Forms given in the same box have the same precedence; the boxes are in order of decreasing precedence.

If @@ is a form with precedence higher than ##, then *x1* @@ *x2* ## *x3* is treated as *(x1* @@ *x2)* ## *x3*, while *x1* ## *x2* @@ *x3* is treated as *x1* ## *(x2* @@ *x3)*.

The last column of the table indicates how multiple appearances of the same form are grouped. When no parentheses appear, the corresponding projection is Flat [4,7.7]. These groupings also govern the treatment of different forms with the same precedence.

Combinations of the relational operators [5] =, ~=, >, >=, <, <= may be input together; if # and ## represent input forms for two such operators, then *x1* # *x2* ## *x3* is treated as *(x1* # *x2)* & *(x2* ## *x3)*.

Products may be input without explicit * when their terms are suitably distinguished. For any numbers *n*, symbols *s* and *f* and expressions *x* the following input forms are taken as products:

n n	*s n*	*f*[*x*] *n*	(*x*) *n*	{*x*} *n*
ns	*s s*	*f*[*x*] *s*	(*x*) *s*	{*x*} *s*
nf[*x*]	*s f*[*x*]	*f*[*x*] *f*[*x*]	(*x*) *f*[*x*]	{*x*} *f*[*x*]
n(*x*)	*s*(*x*)	*f*[*x*] (*x*)	(*x*) (*x*)	{*x*} (*x*)
n{*x*}	*s*{*x*}	*f*[*x*] {*x*}	(*x*) {*x*}	{*x*} {*x*}

Precedence of such forms is as when explicit * are given.

The symbol Null [2.2] may be input as a blank when it appears as the value of an entry in a list, or as a filter in a projection given in standard form. Hence f[] is equivalent to f[Null] and g[,,] to g[Null,Null,Null].

2.11 Syntax modification

cc*:= *dd or **Sxset[*cc*, *dd*, (*class*:0), (*prec*:0)]**

assigns the name ⊂*cc*⊃ of the symbol *cc* to be replaced textually on input by ⊂*dd*⊃ according to one of the following classes of transformations

0 ⊂*cc*⊃ → ⊂*dd*⊃

1 ⊂*cc*⊃ *x* → ⊂*dd*⊃[*x*]

2 *x* ⊂*cc*⊃ → ⊂*dd*⊃[*x*]

3 *x1* ⊂*cc*⊃ *x2* ⊂*cc*⊃ *x3* → ⊂*dd*⊃[*x1*, *x2*, *x3*]

4 *x1* ⊂*cc*⊃ *x2* → ⊂*dd*⊃[*x1*, *x2*]
x1 ⊂*cc*⊃ *x2* ⊂*cc*⊃ *x3* → ⊂*dd*⊃[*x1*, ⊂*dd*⊃[*x2*, *x3*]]

5 *x1* ⊂*cc*⊃ *x2* → ⊂*dd*⊃[*x1*, *x2*]
x1 ⊂*cc*⊃ *x2* ⊂*cc*⊃ *x3* → ⊂*dd*⊃[⊂*dd*⊃[*x1*, *x2*], *x3*]

and with precedence *prec*. Textual replacements are performed before input expressions are simplified. Input text, excluding any strings enclosed in " ", is scanned once only from the beginning, and at each point, textual replacements for the longest possible character strings are made. Transformations 1 through 5 may carry precedences from 1 to 3: 1 is the same as Input, 2 as Plus and 3 as Info.

cc:= or Sxset[*cc*] removes any textual replacements assigned for *cc*.

Sxset[] removes all assignments for textual replacements.

• [10.8]

2.12 Output forms

Most input forms are also used for output.

Parentheses are omitted in output when the required groupings follow the precedences of forms given in [2.10].

Common additional output forms:

```
Mult[x1,x2,x3]      x1 x2 x3

                      x1
Div[x1,x2]            --
                      x2

                     x2
Pow[x1,x2]           x1
```

The output form of the projection `Plot` [10.2] is a "plot".

`Fmt` and `Sx` are printed in a special formatted form.

Assignment of a `Pr` property [4] defines a special output form for a projection.

The presence of special forms in an output expression is indicated by * at the beginning of the output. The labelling of parts in such special cutput forms may not be manifest. A direct and unambiguous representation of any expression is printed by `Lpr` [10.1].

3. Fundamental operations

3.1 Automatic simplification

Any input expression is automatically "simplified" to the maximum extent possible. Unless further assignments are made, the resulting output expression can be simplified no further; if it is input again, it will be output unchanged.

Even if part of an expression is apparently meaningless, no message is printed; that part is merely returned without further simplification. Processing impasses occur if simplification apparently requires infinite time or memory space [1.5].

The simplification of expressions proceeds as follows:

Numbers — Ordinary numbers remain unchanged.

Symbols — A symbol is replaced by the simplified form of any value assigned [3.2] to it.

Projections

1. Unless the projector carries the property `Nosmp` [4], each filter is simplified in turn. If any filter is found to be extedned [4] with respect to the projector, then the projection is replaced or encased as specified in the relevant property list [4].
2. Projections with `Flat` or `Reor` properties [4,7.7] are cast into canonical form. Built-in simplification routines are invoked for system-defined projections.
3. If a value v has been assigned for the projector, then the filters of the projection are used in an attempt to select a part of v. If the required part is present, any necessary replacements for generic symbols [2.2] are performed, and the projection is replaced by the simplified value of the part. When v is a list, its entries are scanned sequentially until one is found whose indices match [2.6] the filters of the projection (and whose value is not `Null`). `Flat` and `Reor` properties are accounted for in this matching process.

Lists — Immediate values [3.2] for entries of a list are simplified in turn.

Whenever an expression to which an immediate value has been assigned (through `:` [3.2]) is simplified, the old value is replaced by the new simplified form. Delayed values (assigned by `::` [3.2]) are simplified whenever they are required, but are not replaced by the resulting simplified forms.

When the value of a projection involves further projections from the same projector (recursion), these further projections are not immediately simplified; after one pass through the whole expression, further passes are made until the projections are completely simplified.

"Static" (manifestly non-terminating) recursive assignments in which the literal form of an expression appears in its simplified value are evaluated to one level only.

Smp[*expr*, (*n*:Inf)]
simplifies *expr* making at most *n* passes.

3.2 Assignment and deassignment

Assignment is used to define a value for an expression. Simplification [3.1] replaces an expression by any value assigned to it.

Values for expressions come in two types:

Immediate — The value is simplified when it is assigned, and is maintained in a simplified form, being updated, if necessary, whenever it is used.

Delayed — The value is maintained in an unsimplified form; a new simplified form is obtained whenever it is used.

expr1*: *expr2 or **Set[*expr1*, *expr2*]**
assigns *expr2* to be the immediate value of the expression *expr1*.

expr1*:: *expr2 or **Setd[*expr1*, *expr2*]**
assigns *expr2* to be the delayed value of the expression *expr1*.

Values *expr2* are assigned to expressions *expr1* of different types as follows:

Numbers — No assignment is made.

Symbols — *expr2* replaces any value previously assigned to *expr1*. No assignment is made for generic and chameleonic symbols [2.2].

Projections — First, the filters of *expr1* are simplified in turn, and any **Flat** or **Reor** properties [4] of its projector *f* are used. Then the specified part in the simplified form *v* of *f* is assigned the value *expr2*. The actions of the assignment for various types of *v* are as follows:

- Symbols — The value of *f* becomes a list (or nested set of lists) with one entry whose indices give the filters of *expr1* and whose value is *expr2*.
- Lists — If the indices of an existing entry of *v* are equivalent [2.6] to the filters of *expr1*, then the value of that entry is replaced by *expr2*. If no such entry is present, then an additional entry with indices given by the filters of *expr1* and with value *expr2* is introduced. New entries are positioned in the list immediately after any entries with more specific [2.6] indices.
- Projections — Unless the relevant filter of *expr1* is an integer, no assignment is made. If the filter corresponds to an existing part of *v*, this part is replaced by the expression *expr2*; otherwise, additional **Null** filters are introduced so as to include the specified part.

Lists — If *expr2* is a list, then each entry in *expr1* is assigned (in parallel) the value of the corresponding *expr2* entry (or **Null** if no such entry exists); otherwise, all entries in *expr1* are assigned the value *expr2*.

Any overall numerical coefficient in *expr1* is divided into *expr2* before assignment.

***expr*:** or ***expr*:Null** or **Set[*expr*]** or **Set[*expr*,Null]**
removes any values assigned for the literal expression *expr*. If filters in projections are removed, the projections are correspondingly shortened. "Removal" of a projector results in its replacement by **Np** [2.3]. If *expr* is a list, values for each entry are removed. ***_symb*:** removes properties [4] assigned to the symbol *symb* (restoring any initial properties if *symb* is system-defined).
• **Del** [7.3]

Set[]
removes values assigned to all expressions.

Inc [*expr*, (*step*:1)]
increments the value of *expr* by *step*.
• Do [6.2]

Dec [*expr*, (*step*:1)]
decrements the value of *expr* by *step*.

Assignment and deassignment projections have the special capability to effect permanent changes on their filters; all other projections must leave their filters unchanged.

3.3 Replacements and substitutions

Values assigned for expressions [3.2] are used whenever they are applicable. More controlled evaluation is provided by the use of substitution projections.

expr1 -> *expr2* or **Rep** [*expr1*, *expr2*, (*levspec*:Inf), (*rpt*:Inf), (*max*:Inf)]
is a purely syntactic construct which represents a replacement of *expr1* by *expr2*. The replacement is specified to be active when used in S projection substitutions on an expression *expr* only for the first *max* occurrences of *expr1* appearing at or below level *lev* in *expr*, and during the first *rpt* passes through *expr*. *expr1->expr2->expr3* is equivalent to *expr1->expr3*.

expr1--> *expr2* or **Repd** [*expr1*, *expr2*, (*levspec*:Inf), (*rpt*:Inf), (*max*:Inf)]
represents a replacement in which *expr2* is maintained in an unsimplified form; a new simplified form is obtained whenever the replacement is performed.

S [*expr*, {*rep1*, *rep2*, ...}, (*rpt*:1), (*levspec*:Inf), (*domcrit*:1)]
performs substitutions in *expr* specified by the replacements *rep1*,*rep2*,... in the domain defined by *levspec* and *domcrit* [2.5]. Each successive subpart of *expr* is compared with the left members of each active replacement *repi* in turn; if a match [2.6] is found, then the part is replaced by the corresponding right member, with any necessary substitutions for generic symbols made. The resulting complete expression is scanned until no further replacements can be used, or at most *rpt* times. (*rpt* may be Inf). When a replacement is used, the resulting subexpression is not scanned for possible further substitutions until it is reached on at least the next pass through the complete expression.

Many relations involving mathematical functions are given in external files [1.3] in the form of replacements, to be applied using S.

Si [*expr1*, {*rep1*, *rep2*, ...}, (*levspec*:Inf), (*domcrit*:1)]
is equivalent to S [*expr*, {*rep1*, *rep2*, ...}, Inf , (*levspec*:Inf), (*domcrit*:1)]
; the *repi* are repeatedly used in *expr* until the result no longer changes.

Arep [{*rep1*, *rep2*, ...}]
performs explicit assignments using Set or Setd projections [3.2] on the Rep or Repd replacements *repi*.

Irep [{*rep1*, *rep2*, ...}]
yields a list of "inverted" replacements.

3.4 Numerical evaluation

N [*expr*, (*acc*:6), (*trunc*:10**^-12)]
<ldist>
yields the numerical value of *expr* in terms of real or complex decimal numbers, maintaining if possible an accuracy of *acc* significant figures, and setting all numerical coefficients smaller than *trunc* to zero.

A , F and Cx projections [2.1] are generated when required.

Numerical values for arbitrary expressions may be defined by assignments [3.2] for the corresponding N[*expr*] *or* N[*expr*, *n*] .

Neq[*expr1*, *expr2*, (*acc*:1*^-8), (*n*:2), (*range*:{-10, 10})]
tests for numerical equality of *expr1* and *expr2* within fractional accuracy *acc* by evaluating them at least *n* times with random numerical choices (in the specified *range*) for all symbolic parameters appearing in them.
- Eq [5]

- D, Int [9.4]
- Sum, Prod [9.2]

3.5 Deferred simplification

'*expr* or Hold[*expr*]
yields an expression entirely equivalent to *expr* but all of whose parts are "held" in an unsimplified form, until "released" by Rel.
(Prefix form is "forward-quote" or "acute accent" character.)

Rel[*expr*]
releases all "held" parts of *expr* for simplification.

- Ev [3.7]

Projectors in simplified projections are maintained in "held" form.

Nosmp [4] filters in projections are converted to "held" form.

3.6 Pre-simplification

!*expr*
represents the simplified form of *expr* regardless of its environment.

Simplifications indicated by ! are performed during input.

! may be used to insert simplified forms as filters in Nosmp [4] projections.

3.7 Partial simplification

Ev[*expr*, (*k*:1)]
yields the result of *k* partial evaluations on the held [3.5] form of *expr*. In each partial evaluation, symbols and projections are replaced by the held forms of any values assigned to them.

4. Properties

_symb or **Prop[*symb*]**
is a list giving properties of the symbol ***symb*** used to specify treatment of ***symb*** or its projections.

The operations of projection [2.3,7.3], assignment [3.2] and deassignment [3.2] may be applied to ***_symb*** just as to symbols.

_symb: causes the properties of a system-defined symbol ***symb*** to revert to their initial form [3.2].

A symbol is considered to "carry" a particular property ***p*** if the value of ***_symb*[*p*]** is not "false" [5].

Properties such as **Flat** which affect treatment of assignments for projections must be defined before the projections are assigned.

The following are system-defined properties for a symbol *s* :

Sys System-defined symbol [2.2].

Gen Generic symbol [2.2], representing the class of expressions on which application of the template ***_s*[Gen]** yields a non-zero number [2.6].

Mgen Multi-generic symbol [2.2].

Cham Chameleonic symbol [2.2].

Init Any value assigned for ***_s*[Init]** is simplified, and then removed, when *s* next appears as a projector.

Tier The projections **s[x]**[y] and **s[x,y]** are distinguished [2.3].

Nosmp The *i*th filter in a projection from *s* is automatically simplified [3.1] if ***_s*[Nosmp][*i*]** is not "false", and is otherwise placed in a "held" form [3.5].

Pr The value of ***_s*[Pr]** is used as the print form for *s* and its projections.
• **Fmt, Sx** [10.1]

Trace If the value of ***_s*[Trace]** is "true", then any projections from *s* are printed before evaluation. Any other values are applied as templates to projections from *s*.
• [10.7]

Flat All projections from *s* are flattened [7.7], so that *s* is treated as an "associative n-ary function".

Reor Filters of projections from *s* are placed in canonical order, by reordering them using permutation symmetries given as the value of ***_s*[Reor]** [7.7].

Comm Equivalent to ***_s*[Reor]:Sym** [7.7]. Causes filters of projections from *s* to be placed in canonical order, so that *s* represents a "commutative function".

Dist ***_f*[Dist]::⦃*g1*,*g*(*h1*:*g1*),(*k1*:*f1*),(*pspec1*:Inf)⦄** defines *f* to be distributive over projections from *gi* appearing as its filters in positions specified by *pspeci*, yielding projections of *hi* and *ki*. The definitions are used as defaults in Ex [7.8].

Powdist ***_f*[Powdist]::⦃*g1*,(*h1*:Plus)⦄** defines projections of *f* to be "power expandable" over projections of *gi* appearing as their first filters, and yielding projections from *hi*. The definitions are used as defaults in Ex [7.8].

Ldist Projections from *s* are "distributed" over the entries of lists appearing as filters in projections from *s* [7.7].

Const *s* is treated as a numerical constant.

Extr The projector *f* of a projection containing *s* (as an isolated symbol or as a projector) in its filters is replaced by a template given as the value of *_s*[Extr,*f*] (see below).

Exte The value of *_s*[Exte] is applied as a template to any projection whose filters involve *s* (as an isolated symbol or as a projector), and for whose projector *f* no entry *_s*[Extr,*f*] exists (see below). Exte is also effective for entries in lists.

Type *s* is treated as carrying the additional properties assigned to a symbol given as the value of *_s*[Type]. These additional properties are considered only if properties given directly for *s* are inadequate. Any number of Type indirection levels may be used.

- Ser [1.11]
- Cons [10.9]

Some additional properties used for special purposes are described below.

All system-defined symbols carry the property Tier.

symb ***_: p*** or Prset[*symb*,*p*]
is equivalent to *_symb*[*p*]:1 and assigns the property *p* to the symbol *symb*.

symb ***_st*** or Tyset[*symb*,*st*]
is equivalent to *_symb*[Type]::*st* and assigns the symbol *symb* to have the type of the symbol *st*.

Entries *_s*[Extr] and *_s*[Exte] in the property list of a symbol *s* allow for "type extension". Standard projections may be replaced by special projections if some of their filters are of some special extended type. Hence, for example, a product may be entered as a projection of Mult, but is replaced by a projection of Psmult if one or more of its filters is a power series (Ps projection [9.5]). Projections reached by type extension are usually not described separately in this manual.

5. Relational and logical operations

An expression is treated as "false" if it is zero, and "true" if it is any non-zero number.

P [*expr*]
yields 1 if *expr* is "true", and 0 otherwise.

The relational and logical projections described below yield 0 or 1 if their "truth" or "falsity" can be determined (by syntactic comparison or linear elimination of symbolic parameters); otherwise they yield simplified forms of undetermined truth value. When only one filter is given, the relational projections defined below yield as images that filter. When more than two filters are given, they yield the conjunction of results for each successive pair of filters.

expr1* = *expr2 or **Eq [*expr1*, *expr2*]**
<Comm>
represents an equation asserting the equality of *expr1* and *expr2*. Equations represented by Eq projections are used in Sol [9.3].
• Neq [9.6]

expr1* ~= *expr2 or **Uneq [*expr1*, *expr2*]**
<Comm>
asserts the inequality of *expr1* and *expr2*.

expr1* > *expr2 or **Gt [*expr1*, *expr2*]**
asserts that *expr1* is numerically larger than *expr2*.

expr1* >= *expr2 or **Ge [*expr1*, *expr2*]**
asserts that *expr1* is numerically larger than or equal to *expr2*.

expr1* < *expr2
is equivalent to *expr2* > *expr1*

expr1* <= *expr2
is equivalent to *expr2* >= *expr1*.

The assignment [3.2] `expr1 > expr2 : 1` defines the expression `expr1` to be greater than `expr2`. The projection Ge tests for assignments of relevant Gt projections.

If # and ## represent special input forms for relational projections then *expr1 # expr2 ## expr3* is converted to *(expr1 # expr2)* & *(expr2 ## expr3)* [2.10].

~*expr* or **Not [*expr*]**
yields 1 if *expr* is 0 and 0 if *expr* is "true".

***expr1* & *expr2* & *expr3* . . .** or **And [*expr1*, *expr2*, *expr3*, . . .]**
<Comm, Flat>
forms the conjunction of the *expri*.

***expr1* | *expr2* | *expr3* . . .** or **Or [*expr1*, *expr2*, *expr3*, . . .**
<Comm, Flat>
forms the inclusive disjunction of the *expri*.

***expr1* || *expr2* || *expr3* . . .** or **Xor [*expr1*, *expr2*, *expr3*, . . .]**
<Comm, Flat>
forms the exclusive disjunction of the *expri*.

expr1* => *expr2 or **Imp [*expr1*, *expr2*]**
represents the logical implication "if *expr1* then *expr2*".

Is[*expr*]
yields 1 if *expr* represents a logical tautology "true" regardless of the "truth" or "falsity" of any symbols appearing in it, and yields 0 otherwise.
- Neq [9.6]

- If [6.1]
- Intp,.. [7.6]

Ord[*expr1*,*expr2*]
yields +1 if expr1 is lexically [10.6] ordered before expr2, -1 if *expr1* is lexically ordered after *expr2* and 0 if they are literally equivalent [2.6].
- Reor [7.7]
- Sort [7.7]

6. Control Structures

6.1 Conditional statements

If [*pred*, (*expr1*:Null), (*expr2*:Null), (*expr3*:Null)]
yields *expr1* if the predicate expression *pred* is determined to be "true" [5]; *expr2* if it is determined to be "false", and *expr3* if its truth or falsity cannot be determined [5]. Only the *expri* selected by evaluation of *pred* is simplified.

Swtch [*pred1*, *expr1*, *pred2*, *expr2*, . . .]
tests *pred1*, *pred2*, ... in turn, selecting the *expri* associated with the first one determined to be "true" [5] (Null if none are "true"). Only the *predi* tested and the *expri* selected are simplified.

6.2 Iteration

Rpt [*expr*, (*n*:1)]
simplifies *expr n* times, yielding the last value found.

Loop [(*precond*:1), *expr*, (*postcond*:1)]
repeatedly simplifies *precond*, *expr* and *postcond* in turn, yielding the last value of *expr* found before *precond* or *postcond* ceases to be "true" [5].

For [*init*, *test*, *next*, *expr*]
first simplifies *init*, and then repeatedly simplifies *expr* and *next* in turn until *test* fails to be "true" (according to the sequence *init* <*test expr next*>), yielding the last form of *expr* found.

Do [*var*, (*start*:1), *end*, (*step*:1), *expr*]
first sets *var* to *start* and then repeatedly evaluates *expr*, successively incrementing the value of *var* by *step* until it reaches the value *end* ; the image is the last form of *expr* found.

- Inc, Dec [3.2]
- Ret, Jmp [6.3]

6.3 Procedures and flow control

***expr1*; *expr2*; . . . or Proc [*expr1*, *expr2*, . . . , *exprn*]**
represents a procedure [1.8] in which the expressions *expri* are simplified in turn, yielding finally the value of *exprn*.

In the form expr1;expr2;...;exprk; the last filter of Proc is taken to be Null [1.1].

The value of %% [1.8] is reassigned at each segment in a Proc to be the last non-Null *expri* simplified.

The unsimplified form of each *expri* is maintained throughout the execution of a Proc, so as to allow resimplification if required by a control transfer.

Proc [] initiates an interactive procedure [1.8].

Lcl [*s1*, *s2*, . . .]
declares the symbols *s1*,*s2*,... to be "local variables" in the current Proc (and any nested within it); the original values and properties of the *si* are removed, to be restored upon exit from the Proc.

Local variables are conventionally given names beginning with the character % [2.2].

Lcl may be used in interactive procedures [1.8].

Lbl [*expr*]
represents a "label" within a Proc ; its "identifier" *expr* is resimplified when a Jmp might transfer control to its position.

Jmp [*expr*]
causes "control" to be "transferred" to the nearest label which matches Lbl [*expr*] The current Proc, and then any successive enclosing Proc are scanned to find a suitable Lbl. After Jmp has acted, the remaining segments of the Proc containing the Lbl are (re)simplified. For positive integer n, Jmp [n] transfers control to the nth segment in the current procedure. Jmp may be used in interactive procedures: if the specified Lbl is not present, input lines are read without simplification until it is encountered.

Ret [*expr*, (n:1)]
exits at most n nested control structures, yielding *expr* as the value of the outermost one. Ret is effective in Proc, Rpt, For and Do. It may be used to exit interactive subsidiary procedures [1.8]. Ret [*expr*, Inf] returns from any number of nested procedures to standard input mode [1.8].

- [10.7]

7. Structural operations

7.1 Projection and list generation

x **..** ***y*** or **Seq[*x*, *y*]**
yields if possible a null projection [2.3] consisting of a sequence of expressions whose integer parameters form linear progressions between those in x and y. For two integers m and n, (with $n > m$) $m..n$ yields $[m, m+1, m+2, \ldots, n-1, n]$. Similarly, $f[m1,m2]..f[n1,n2]$ yields $[f[m1,m2], f[m1+1, m2+i], f[m1+2, m2+2i], \ldots, f[n1,n2]]$ if $i = (n2-m2)/(n1-m1)$ is an integer. Only the values of integer parameters may differ between x and y, and all such differences must be integer multiples of the smallest.

Ar [*spec*, (*temp*:Eq), (*icrit*:1), (*vcrit*:1)]
generates a list whose entries have sets of indices with ranges specified by *spec*, and whose values are obtained by application of the template *temp* to these indices. Sequences of indices at each level in the list are defined by

n	$1, 2, \ldots n$
$\{s, e, (i:1)\}$	$s,\ s+i,\ s+2i, \ldots,\ s+k*i$ where k is the largest integer such that $s + k*i$ is not greater than e.
$\{\{x1,x2,..\}\}$	$x1, x2, \ldots$

and collected into a complete specification $\{spec1, spec2, \ldots\}$. For a contiguous [2.4] list with one level, *spec* may be given as n. Entries with sets of indices on which application of the template *icrit* would yield $\emptyset$ are omitted. Entries whose values would yield $\emptyset$ on application of the template *vcrit* are also omitted.

- **Outer** [9.6]
- **Dim** [7.4]
- [3.6]

Repl [*expr*, (*n*:1)]
yields a null projection [2.3] containing n replications of *expr*.

List [*expr1*, *expr2*, ...]
yields the contiguous list $\{expr1, expr2, \ldots\}$

7.2 Template application

Ap [*temp*, {*expr1*, (*expr2*, ...)}]
applies the template *temp* to the *expri* [2.7].

Map [*temp*, *expr*, (*levspec*:1), (*domcrit*:1), (*ltemp*:Null)]
yields the expression obtained by recursively applying the template *temp* at most *max* times to the parts of *expr* in the domain specified by *levspec* and *domcrit* [2.5]. Any non-Null result obtained by applying the template *ltemp* to the (positive or negative) integer specifying the level in *expr* reached is used as a second expression on which to apply *temp*.

7.3 Part extraction and removal

***expr*[*filt1*, *filt2*, ...] or Proj[*expr*, {*filt1*, *filt2*, ...}]**
extracts the part of *expr* specified by successive projections with the filters *filt1*, *filt2*, ... [2.3].
• [2.10]

Pos[{*form1*}, *expr*, (*levspec*:Inf), (*max*:Inf), (*domcrit*:1)]
gives a list of sets of filters specifying the positions of (at most *max*) occurrences of parts in *expr* (in the domain specified by *levspec* and *domcrit* [2.5]) matching any of the *formi*. If *formi* are patterns [2.6], the actual subexpressions matched are also given.
• In [7.5]

Elem[*expr*, {*n1*, *n2*, ...}]
extracts successively the *n*ith values in the list *expr*, irrespective of their indices.

Last[*list*]
yields the value of the last entry in *list*.

Ind[*list*, *n*]
yields the index of the *n*th entry in *list*.

Dis[*expr*, (*lev*:1)]
yields a list in which all projections in *expr* (below level *lev*) are "disassembled" into lists with the same parts.

As[*expr*, (*lev*:1)]
yields an expression in which any suitable lists (at or below level *lev*) in *expr* are "assembled" into projections with the same parts.

Del[*form*, *expr*, (*lev*:Inf), (*n*:Inf)]
yields an expression in which (at most *n*) parts matching *form* (and appearing at or below level *lev*) in *expr* have been deleted.
• Set [3.2]

7.4 Structure determination

Tree[*expr*, (*nlev*:1)]
yields a list of successive replacements specifying the construction of *expr* from its level -*nlev* parts.
• Dis [7.3]

Len[*expr*]
the number of filters or entries in a projection or list *expr*, and 0 for a symbol *expr*. (The "length" of *expr*).

Dep[*expr*]
the maximum number of filters necessary to specify any part of *expr*. (The "depth" of *expr* [2.5]).

Dim[*list*, (*llev*:Inf)]
gives a description of the ranges of indices at or below level *llev* in *list* (in the form used by Ar [7.1]).

Hash[*expr*, (*n*:2^15)]
a positive integer less than *n* which provides an almost unique "hash code" for *expr*.

7.5 Content determination

In[{*form1*}, *expr*, (*lev*:Inf)]
yields 1 if a part matching any of the *formi* occurs in *expr* at or below level *lev*, and 0 otherwise.

Cont[<*expr*: (all symbols)>, (*crit*:1)]
yields an ordered list of the symbols in *expr* on which application of the template *crit* does not yield 0 (default is to omit symbols appearing as projectors).

7.6 Character determination

The following projections yield 1 if *expr* is determined to be of the specified character, and 0 otherwise. The determination includes use of any assignments made for the testing projection; Intp[x]:1 thus defines x to be an integer.

Symbp[*expr*] Single symbol

Numbp[*expr*] Real or complex number.

Realp[*expr*] Real number.

Imagp[*expr*] Purely imaginary number.

Intp[*expr*] Integer.

Natp[*expr*] Natural number (positive integer).

Evenp[*expr*] Even integer.

Oddp[*expr*] Odd integer.

Ratp[*expr*, (*maxden*:1*^8), (*acc*:1*^-13)]
Rational number (to within accuracy *acc*) with denominator not greater than *maxden*.

Projp[*expr*] Projection.

Listp[*expr*] List.

Contp[*expr*, (*lev*:Inf)]
List or list of lists contiguous [2.4] to at least level *lev*.

Fullp[*expr*, (*lev*:Inf)]
"Full" list whose indices (and those of its sublists at or below level *lev*) are "contiguous" and have ranges independent of the values of any other indices (so that the list is "rectangular").

Valp[*expr*] Value other than Null.

Heldp[*expr*] "Held" expression [3.5].

Polyp[*expr*, ({*form1*})]
Polynomial in "bases" matching *formi* [9.1].

7.7 List and projection manipulation

Cat[*list1*, *list2*, . . .]
<Flat>
yields a contiguous list [2.4] obtained by concatenating the entries in *list1*, *list2*,... Cat[*list*] renders the indices of entries in *list* contiguous, without affecting their values.

Sort[*list*, (*card*:Null), (*ord*:Ord)]
yields a list *v* obtained by sorting the top-level entries in *list* so that either the expressions Ap[*card*, {Ent[*v*, {*i*}]] are in canonical order (for increasing *i*) , or

Ap[*ord*, {Ent[*v*, {*i*}], Ent[*v*, {*j*}]}] is non-negative for $i > j$ and is non-positive for $i < j$.

Cyc[*expr*, (*n*:1)]
gives a list or projection obtained by cycling entries or filters in *expr* to the left by *n* positions with respect to their first index.

Rev[*expr*]
yields a list or projection obtained from *expr* by reversing the order of entries or filters with respect to their first index.

Reor[*expr*, (*f*:*expr*[0]), (*reord*:Sym)]
places filters of projections from *f* in *expr* in canonical order using the reordering (permutation) symmetries specified by *reord*. Symmetries are represented by the following codes (or lists of such codes applied in the order given):

Sym	Completely symmetric under interchange of all filters.
Asym	Completely antisymmetric. • Sig [9.6]
Cyclic	Completely cyclic.
Sym[*i1*, *i2*, . .]	Symmetric with respect to interchanges of filters *i1*, *i2*, ...
Asym[*i1*, *i2*, . .]	Antisymmetric with respect to interchanges of filters *i1*, *i2*, ...
Cyclic[*i1*, *i2*, . .]	Symmetric under cyclic interchange of filters *i1*, *i2*, ...
Greor[{*perm1*, (*wt1*:1)}, {*perm2*, (*wt2* :1)}, . .]	Projections are multiplied by the weights obtained by application of *wti* to them when their filters are permuted so that the *j*th becomes the *permi*[*j*]th.

Projections from a symbol *f* are automatically reordered according to any permutation symmetries given (using the above codes) as the value of the property *_f*[Reor] [4].

• Comm [4]

Ldist[*expr*, (*f*:*expr*[0]), (*levspec*:Inf), (*domcrit*:1)]
"distributes" projections from *f* in the domain of *expr* specified by *levspec* and *domcrit* over lists appearing as their filters.
f[{[i1]:v11, [i2]:v12, . .}, {[i1]:v21, [i2]:v22, . .}, a, . .] becomes {[i1]:f[v11, v21, a, . .], [i2]:f[v12, v22, a, . .], . .} where a is not a list.

Projections from symbols carrying the property Ldist [4] are automatically distributed over lists appearing as their filters.

Flat[*expr*, (*levspec*:Inf), (*f*:*expr*[0] or), (*domcrit*:1)]
"flattens" nested projections from *f* or lists in *expr* in the domain specified by *levspec* and *domcrit* [2.5]. *f* projections appearing as filters within *f* projections are replaced by null projections [2.3]. Sublists in lists are "unraveled". Indices in flattened lists are contiguous [2.4].

Projections from symbols carrying the property Flat are automatically "flattened" [4].

Union[*list1*, *list2*, . . .]
<Comm, Flat>
yields a sorted contiguous [2.4] list of all entries appearing in *list1*, *list2*, . . .

Inter [*list1*, *list2*, ...]
<Comm, Flat>
yields a sorted contiguous [2.4] list of the entries common to *list1*, *list2*, ...

7.8 Distribution and expansion

Ex [*expr*, (*dlist*), (*ndlist*: { }), (*rpt*: Inf), (*levspec*: Inf), (*domcrit*: 1)]
"expands" *expr* in the domain specified by *levcrit* and *domcrit* [2.5] using distributive replacements for projections specified in *dlist* but not in ndlist, performing replacements on a particular level at most *rpt* times. The default replacements in *dlist* are standard mathematical results for Mult, Div, Dot, Pow, Exp, Log, G and their extensions [4], together with any replacements defined by Dist or Powdist properties [4].

Dist [*expr*, {{*f1*, *g1*, ((*h1*: *g1*), (*k1*: *f1*)), (*pspec*: Inf)}, (*rpt*: Inf), (*levspec*: Inf), (*domcrit*: 1)]
distributes occurrences of the projectors *fi* in *expr* (in the domain specified by *levspec* and *domcrit* [2.5]) over projections from *gi* appearing as their filters, yielding projections of *hi* and *ki*. Distributions on a particular level are performed at most *rpt* times. A distribution specified by {*f*, *g*, *h*, *k*, *pspec*} corresponds to the replacement
f[$$1, *g*[*x1*, *x2*, ...], $$2] -> *h*[*k*[$$1, *x1*, $$2], *k*[$$1, *x2*, $$2], ...]. The possible positions at which the *g* projection may appear in *f* are specified by *pspec* according to

i	1 through *i*.
{i}	*i* only.
{*i*, *j*}	*i* through *j*.
{*i*, *j*, *pcrit*}	*i* through *j* and such that application of the template *pcrit* yields "true".

Powdist [*expr*{{*fpow1*, *fmult1*, (*fplus1*: Plus)}(*rpt*: Inf), (*levspec*: Inf), (*domcrit*: 1)]
performs a "power expansion" on projections from *fpowi* appearing in *expr* (in the domain specified by *levspec* and *domcrit*) over projections from *fplus* appearing as their first filters, and yielding projections from *qmult*. Expansions on a particular level are performed at most *rpt* times. A power expansion specified by {*fpow*, *fmult*, *fplus*} corresponds to the replacement *fpow*[*fplus*[$$1], $n_?Natp[$n]] -> *fmult*[Rep[*fplus*[$$1], $n] followed by distribution of *fmult* over *fplus*.

• Ldist [4,7.7]

7.9 Rational expression manipulation and simplification

Nc [*expr*]
gives the overall numerical coefficient of *expr*.
• [2.5]

Coef [*term*, *expr*, (*temp*: Plus)]
yields the result of applying *temp* to the set of coefficients of *term* in *expr*.

Expt [*form*, *expr*, (*temp*: Max)]
yields the result of applying *temp* to the set of exponents for *form* in *expr*.

Num [*expr*]
numerator of *expr*.

Den [*expr*]
denominator of *expr*.

Rat[*expr*, (*crit*:1), (*levspec*:Inf), (*domcrit*:1)]
combines over a common denominator terms in the domain of *expr* specified by *levspec* and *domcrit* [2.5], and on which application of the template *crit* does not yield **0**.

Col[*expr*, (*crit*:1), (*levspec*:Inf), (*domcrit*:1)]
collects terms with the same denominator in the domain of *expr* specified by *levspec* and *domcrit*, but on which application of the template *crit* does not yield **0**.

Cb[*expr*, {*form1*}, (*levspec*:Inf), (*domcrit*:1)]
combines coefficients of terms matching *form1*, *form2*, .. in the domain of *expr* specified by *levspec* and *domcrit*.

• Fac, Pf [9.1]

7.10 Statistical expression generation and analysis

Rex[(*n*:10), ({*unit1*, (*uwt1*:1)}, ...}), ({*temp1*, (*twt1*:1), (*n1*:1)}, ...})]
generates a pseudorandom expression containing on average *n* "units" selected from the *uniti* with statistical weights *uwti*, and combined by application of templates selected from the *tempi* with weights *twti;* each *tempi* acts on *ni* expressions (or an average of *-ni* expressions for negative *ni*). Simple defaults are provided for the *uniti* and *tempi*.
• Rand [8.3]

Aex[*expr1*, *expr2*, ...]
performs a simple statistical analysis on the *expri* and yields a held [3.5] Rex projection necessary to generate further expressions with the same "statistical properties".

8. Mathematical functions

8.1 Introduction

Reductions of mathematical functions occur through simplification or numerical evaluation. Simplification yields an exact transformation; numerical evaluation is performed only in the absence of symbolic parameters, and may be approximate.

Simplifications for arithmetic operations [8.2] and numerical functions [8.3] are automatically performed. Elementary transcendental functions [8.5] are simplified only when the result is a rational number or multiple of a constant [8.4]. Many additional relations and transformations are given as replacements [3.3] defined in external files, and applied selectively by S projections [3.3].

Real or complex numerical values for any of the functions described below are obtained by N projections [3.4].

Whenever a mathematical "function" is used to represent solutions of an equation, it may take on several values for any particular set of arguments, as conventionally parametrized by Riemann sheets. For such multivalued "functions", numerical values are always taken on a single "principal" Riemann sheet; at the conventional positions of branch cuts, the limiting values in a counter-clockwise approach to the cut are given.

Differentiation, integration [9.4] and power series expansion [9.5] of mathematical functions is performed where possible.

All projections representing mathematical functions carry the property `Ldist` [4].

Definitions of mathematical functions are based primarily on four references:

AS "Handbook of Mathematical Functions", ed. M.Abramowitz and I.Stegun, NBS AMS 55 (1964); Dover (1965).

GR "Table of Integrals, Series and Products", I.Gradshteyn and I.Ryzhik, Academic Press (1965).

MOS "Formulas and Theorems for the Special Functions of Mathematical Physics", W.Magnus, F.Oberhettinger and R.P.Soni, 3rd ed, Springer-Verlag (1966).

BMP "Higher Transcendental Functions", Bateman Manuscript Project (A.Erdelyi et al.) Vols 1-3, McGraw-Hill (1953).

Citations in which notations or conventions differ from those used here are indicated by †.

8.2 Elementary arithmetic operations

expr1 **+** ***expr2*** **+ ...,** ***expr1*** **-** ***expr2*** **+ ...,** or **Plus[*expr1*, *expr2*, ...]**
<Comm,Flat>

expr1 ***** ***expr2*** *** ...,** or **Mult[*expr1*, *expr2*, ...]**
<Comm,Flat>

expr1**/*****expr2*** or **Div[*expr1*, *expr2*]**

expr1**^*****expr2*** or **Pow[*expr1*, *expr2*]**

Sqrt[*expr*]

expr1 **.** ***expr2*** **. ...** or **Dot[*expr1*, *expr2*, ...]**
<Flat>

forms the inner product of *expr1*,*expr2*,.... For two lists `x` and `y` the inner product is a list obtained by summing `x[i[1],i[2],...,i[n-1],k] * y[k,j[2],...,j[m]]` over all values of the index `k` for which entries are

present in both x and y.
• Inner [9.6]

***expr1* ** *expr2* ** ...or Omult[*expr1*, *expr2*, ...]**
<Comm,Flat>
forms the outer product of *expr1*, *expr2*, ...
• Outer [9.6]

Multiplication of an expression by a numerical coefficient does not involve an explicit Mult projection. Such products may nevertheless be matched by a pattern in which a generic symbol representing the coefficient appears in a Mult projection [2.6].

Plus[*expr*] and Mult[*expr*] are taken as *expr*; Plus[] is 0 and Mult[] is 1.

• Fctl, Dfctl, Comb [8.6]

8.3 Numerical functions

Gint[*x*]
the greatest integer not larger than the real number *x*
• Mod [8.10]

Sign[*x*]
1 or -1 if *x* is a positive or negative real number, and 0 if *x* is zero.

Theta[*x*]
Heavyside step function $\vartheta(x)$.

Delta[*x*]
Dirac's delta function $\delta(x)$.

Abs[*x*]
the absolute value of a real or complex number *x*.

Conj[*expr*]
complex conjugate.

Re[*expr*]
real part.

Im[*expr*]
imaginary part.

Max[*x1*, *x2*, ...]
<Comm,Flat>
the numerically largest of *x1*, *x2*, .. if this can be determined.

Min[*x1*, *x2*, ...]
<Comm,Flat>
the numerically smallest of *x1*, *x2*, .. if this can be determined.

Rand[(*x*:1), (*seed*)]
pseudorandom number uniformly distributed between 0 and *x*. A number *seed* may be used to determine the sequence of numbers generated.

8.4 Mathematical constants

Pi	π = 3.14159..
E	e = 2.71828..
Euler	Euler-Mascheroni constant γ = 0.577216.. [AS 6.1.3; GR 9.73; MOS 1.1]
Deg	$\pi/180$ = 0.0174533..

Omult Gint Sign Theta Delta Abs Conj Re Im Max Min Rand Pi E Euler Deg

Phi Golden ratio φ = 1.61803..

Catalan Catalan's constant 0.915966.. [AS 23.2.23; GR 9.73]

• I, Inf [2.2]

8.5 Elementary transcendental functions

Exp[z]

Log[z, (*base*:E)]
logarithm with branch cut along negative real axis [AS 4.1.1].

Sin[z]	**Asin[z]**	**Sinh[z]**	**Asinh[z]**
Cos[z]	**Acos[z]**	**Cosh[z]**	**Acosh[z]**
Tan[z]	**Atan[z]**	**Tanh[z]**	**Atanh[z]**
Csc[z]	**Acsc[z]**	**Csch[z]**	**Acsch[z]**
Sec[z]	**Asec[z]**	**Sech[z]**	**Asech[z]**
Cot[z]	**Acot[z]**	**Coth[z]**	**Acoth[z]**

trigonometric and inverse trigonometric functions with arguments in radians [AS 4.4.1-4.4.6; AS 4.6.1-4.6.6].

• Deg [8.4]

Gd[z], Agd[z]
Gudermannian functions gd(z), $\mathrm{gd}^{-1}(z)$ [AS 4.3.117].

8.6 Gamma, Zeta and related functions

Gamma[z, (*a*:*0*)]
Euler Γ function $\Gamma(z)$ [AS 6.1.4; GR 8.310; MOS 1.1; BMP 1.1] and incomplete Γ function $\Gamma(z,a)$ [AS 6.5.3; GR 8.350.2; MOS 9.1.1; BMP 6.9.2.21].

***n*!** or **Fctl[*n*]**
factorial [AS 6.1.6].

***n*!!** *or* **Dfctl[*n*]**
double factorial [AS 6.1.49 (footnote); GR p.xliii]

Poc[*x*, *n*]
Pochhammer symbol $(x)_n$ [AS 6.1.22; MOS 1.1].

Comb[*n*, *m*[*1*], (*m*[*2*], . . .*m*[*k*-*1*], (*m*[*k*]:*n*-*m*[*1*]-*m*[*2*]-. . .-*m*[*k*-*1*]))]
Multinomial coefficient $(n;m[1],m[2],\cdots,m[k-1],m[k])$ [AS 24.1.2]; **Comb[*n*, *m*]** gives binomial coefficient $\binom{n}{m}$ [AS 6.1.21; MOS 1.1]

Ei[z]
Exponential integral Ei(z) [AS 5.1.2; GR 8.2; MOS 9.2.1; BMP 6.9.2.(25)].

• Erf [8.7]

Expi[(*n*:1), z]
Exponential integrals $E_n(z)$ [AS 5.1.4; MOS 9.2.1].

Logi[z]
Logarithm integral function li(z) [AS 5.1.3; GR 8.24; MOS 9.2.1].

Sini[z]
Sine integral function Si(z) [AS 5.2.1; GR 8.230.1; MOS 9.2.2].

Cosi[z]
Cosine integral function Ci(z) [AS 5.2.2; GR 8.230.2; MOS 9.2.2].

Sinhi[z]
Hyperbolic sine integral function Shi(z) [AS 5.2.3; MOS 9.2.2].

Coshi [z]
Hyperbolic cosine integral function Chi(z) [AS 5.2.4; MOS 9.2.2].

Lob [z]
Lobachevskiy's function $L(z)$ [GR 8.26].

Beta [x, y, (a:1)]
Euler B function $B(x,y)$ [AS 6.2.1; GR 8.380; MOS 1.1; BMP 1.5] and incomplete B function $B(x,y,a)$ [AS 6.6.1; GR 8.391; MOS 9.2.5; BMP 2.5.3].

Psi [z, (n:1)]
Digamma function $\psi(z)$ [AS 6.3.1; GR 8.360; MOS 1.2; BMP 1.7.1] and polygamma functions $\psi^{(n-1)}(z)$ [AS 6.4.1; MOS 1.2; BMP 1.16.1].

Ler [z, (s:2), (a:0)]
Lerch's transcendent $\Phi(z,s,\alpha)$ [GR 9.55; MOS 1.6; BMP 1.11].

Zeta [z, (a:1)]
Riemann ζ function $\zeta(z)$ [AS 23.2; GR 9.513,9.522; MOS 1.3; BMP 1.12] and generalized ζ function $\zeta(z,\alpha)$ [GR 9.511,9.521; MOS 1.4; BMP 1.10].

Li [(n:2), z]
Dilogarithm (Spence's function) $\mathrm{Li}_2(z)$ [† AS 27.7; MOS 1.6; BMP 1.11.1] and polylogarithm function $\mathrm{Li}_n(z)$ [BMP 1.11.(14)].

Catb [n]
Catalan's β function $\beta(n)$ [AS 23.2.21].

EpsZ [{g1, g2, ...}, {h1, h2, ...}, s, phi]
Epstein's Z function $Z\left[\begin{matrix} g1 & g2 & \cdots \\ h1 & h2 & \cdots \end{matrix}\right](s)_\varphi$ [BMP 17.8].

Ber [n, (x:0)]
Bernoulli numbers B_n [AS 23.1.2; GR 9.61; MOS 1.5.1; BMP 1.13.(1)] and polynomials $B_n(x)$ [AS 23.1.1; GR 9.62; MOS 1.5.1; BMP 1.13.(2)].

Eul [n, (x)]
Euler numbers E_n [AS 23.1.2; GR 9.63; MOS 1.5.2; BMP 1.14.(1)] and Euler polynomials $E_n(x)$ [AS 23.1.1; MOS 1.5.2; BMP 1.14.(2)] (note relative normalization between numbers and polynomials).

8.7 Confluent hypergeometric and related functions

Chg [a, c, z]
Confluent hypergeometric (Kummer) function ${}_1F_1(a;c;z)$ [AS 13.1.2; GR 9.210; MOS 6.1.1].

KumU [a, b, z]
Kummer's U function $U(a,b,z)$ [AS 13.1.3; GR 9.210.(2); MOS 6.1.1].

WhiM [l, m, z]
Whittaker's M function $M_{l,m}(z)$ [AS 13.1.32; GR 9.220.(2); MOS 7.1.1].

WhiW [l, m, z]
Whittaker's W function $W_{l,m}(z)$ [AS 13.1.33; GR 9.220.(4); MOS 7.1.1].

Par [p, z]
Parabolic cylinder functions $D_p(z)$ [† AS 19.3.7; GR 9.240; MOS 6.1.1].

CouF [l, e, r]
Regular Coulomb wave function $F_L(\eta,r)$ [AS 14.1.3].

CouG [l, e, r]
Irregular Coulomb wave function $G_L(\eta,r)$ [AS 14.1.14].

BesJ [n, z]
Regular Bessel function $J_n(z)$ [AS 9.1.10; GR 8.402; MOS 3.1].

Coshi Lob Beta Psi Ler Zeta Li Catb EpsZ Ber Eul Chg KumU
WhiM WhiW Par CouF CouG BesJ

BesY[*n*, *z*]
Irregular Bessel function (Weber's function) $Y_n(z)$ [AS 9.1.11; GR 8.403.(1); MOS 3.1].

Besj[*n*, *z*]
Regular spherical Bessel function $j_n(z)$ [AS 10.1.1; MOS 3.3].

Besy[*n*, *z*]
Irregular spherical Bessel function $y_n(z)$ [AS 10.1.1; MOS 3.3].

BesK[*n*, *z*]
Modified Bessel function $K_n(z)$ [AS 9.6.2; GR 8.407.(1); MOS 3.1].

BesI[*n*, *z*]
Modified Bessel function $I_n(z)$ [AS 9.6.3; GR 8.406; MOS 3.1].

BesH1[*n*, *z*]
Hankel function $H_n^{(1)}(z)$ [AS 9.1.3; GR 8.405.(1)].

BesH2[*n*, *z*]
Hankel function $H_n^{(2)}(z)$ [AS 9.1.4; GR 8.405.(1)].

Kelbe[*n*, *z*]
Complex Kelvin functions $\mathrm{ber}_n(z) + i\ \mathrm{bei}_n(z)$ [AS 9.9.1; GR 8.561].

Kelke[*n*, *z*]
Complex Kelvin functions $\mathrm{ker}_n(z) + i\ \mathrm{kei}_n(z)$ [AS 9.9.2; GR 8.563.(2)].

StrH[*n*, *z*]
Struve function $\mathrm{H}_n(z)$ [AS 12.1; GR 8.550.(1)].

StrL[*n*, *z*]
Modified Struve function $\mathrm{L}_n(z)$ [AS 12.2.1; GR 8.550.(2)].

AngJ[*n*, *z*]
Anger function $\mathrm{J}_n(z)$ [AS 12.3.1; GR 8.580.(1)].

WebE[*n*, *z*]
Weber's function $\mathrm{E}_n(z)$ [AS 12.3.3; GR 8.580.(2)].

Lom[*m*, *n*, *z*]
Lommel's function $s_{m,n}(z)$ [GR 8.570.(1); MOS 3.10.1].

AirAi[*z*]
Airy's function Ai(z) [AS 10.4.2].

AirBi[*z*]
Airy's function Bi(z) [AS 10.4.3].

Erf[*z*]
Error function erf(z) [AS 7.1.1; GR 8.250.(1)].

Erfc[*z*]
Complementary error function erfc(z) [AS 7.1.2].

• `Gamma, Ei` [8.6]

FreC[*z*]
Fresnel's function $C(z)$ [AS 7.3.1; GR 8.250.(1)].

FreS[*z*]
Fresnel's function $S(z)$ [AS 7.3.2; GR 8.250.(2)].

Lag[*n*, (*a*:1), *z*]
(Generalized) Laguerre function $L_n^{(a)}(z)$ [AS 22.2.12; GR 8.970].

Her[*n*, *z*]
Hermite's function $H_n(z)$ [AS 22.2.14; GR 8.950].

Pcp[*n*, *nu*, *z*]
Poisson-Charlier polynomials $\rho_n(\nu, z)$ [AS 13.6.11; MOS 6.7.2].

`BesY Besj Besy BesK BesI BesH1 BesH2 Kelbe Kelke StrH StrL`
`AngJ WebE Lom AirAi AirBi Erf Erfc FreC FreS Lag Her Pcp`

Tor [*m*, *n*, *z*]
Toronto function $T(m,n,z)$ [AS 13.6.20; MOS 6.7.2].

Batk [*n*, *z*]
Bateman's function $k_\nu(z)$ [AS 13.6.33; MOS 6.7.2].

8.8 Hypergeometric and related functions

Hg [*a*, *b*, *c*, *z*]
Gauss hypergeometric function ${}_2F_1(a,b;c;z)$ [AS 15.1.1; GR 9.10; MOS 2.1].

JacP [*n*, *a*, *b*, *z*]
Jacobi functions $P_n^{(a,b)}(z)$ [AS 22.2.1; GR 8.960; MOS 5.2.1].

Geg [*n*, *l*, *x*]
Gegenbauer (ultraspherical) functions $C_n^{(l)}(x)$ [AS 22.2.3; GR 8.930; MOS 5.3.1].

CheT [*n*, *x*]
Chebyshef function of first kind $T_n(x)$ [AS 22.2.4; MOS 5.3.1].

CheU [*n*, *x*]
Chebyshef function of second kind $U_n(x)$ [AS 22.2.5; MOS 5.3.1].

LegP [*l*, (*m*:0), *z*]
(Associated) Legendre functions $P_l^m(z)$ [AS 8.1.2; GR 8.702; MOS 4.1.2, 5.4.1].

LegQ [*l*, (*m*:0), *z*]
(Associated) Legendre functions of second kind $Q_l^m(z)$ [AS 8.1.3; GR 8.703; MOS 4.1.2, 5.4.2].

• Beta [8.6]

EllK [*k*, (*t*:Pi/2)]
First kind elliptic integral $K(k\,|\,t)$ [† AS 17.2.6; † GR 8.111.(2); MOS 10.1]

EllE [*k*, (*t*:Pi/2)]
Second kind elliptic integral $E(k\,|\,t)$ [† AS 17.2.8; † GR 8.111.(3); MOS 10.1]

EllPi [*k*, (*t*:Pi/2)]
Third kind elliptic integral $\Pi(k\,|\,t)$ [† AS 17.2.14; † GR 8.111.(4); MOS 10.1]

JacSn [*x*, (*m*:0)] **JacCn [*x*, (*m*:0)]** **JacDn [*x*, (*m*:0)]**
JacCd [*x*, (*m*:0)] **JacSd [*x*, (*m*:0)]** **JacNd [*x*, (*m*:0)]**
JacDc [*x*, (*m*:0)] **JacNc [*x*, (*m*:0)]** **JacSc [*x*, (*m*:0)]**
JacNs [*x*, (*m*:0)] **JacDs [*x*, (*m*:0)]** **JacCs [*x*, (*m*:0)]**

JacAm [*x*, (*m*:0)]
Jacobian elliptic functions Sn(x|m) etc. [AS 16.1; GR 8.144; MOS 10.3].

Jacth [*i*, *u*]
Jacobi ϑ functions $\vartheta_i(u)$ [AS 16.27; GR 8.18; MOS 10.2].

WeiP [*u*]
Weierstrass function $P(u)$ [AS 18; GR 8.160; MOS 10.5].

Weiz [*u*]
Weierstrass ζ function $\zeta(u)$ [GR 8.171.(1); MOS 10.5].

Weis [*u*]
Weierstrass σ function $\sigma(u)$ [GR 8.171.(2); MOS 10.5].

Tor Batk Hg JacP Geg CheT CheU LegP LegQ EllK EllE EllPi
JacAm Jacth WeiP Weiz Weis

8.9 Further special functions

Ghg[*p*, *q*, {*a1*, *a2*, ...}, {*b1*, *b2*, ...}, *z*]
Generalized hypergeometric function [MOS 2.9].

Mei[*m*, *n*, *p*, *q*, {*a1*, ..., *ap*}, {*b1*, ..., *bq*}, *z*]
Meijer G function [GR 9.30].

MacE[*a*, *b*, *z*]
MacRobert E function [GR 9.4; MOS 6.7.2].

Matce[*n*, *x*, *q*]
Matse[*n*, *x*, *q*]
Mathieu functions $\mathrm{ce}_n(x,q)$, $\mathrm{se}_n(x,q)$ [AS 20; GR 8.61].

Wig[{*j1*, *m1*}, {*j2*, *m2*}, {*j3*, *m3*}]
Wigner 3-j symbol (Clebsch-Gordan coefficient) $\begin{pmatrix} j1 & j2 & j3 \\ m1 & m2 & m3 \end{pmatrix}$ [AS 27.8.1].

Rac[*j1*, *j2*, *j3*, *j4*, *j5*, *j6*]
Racah 6-j symbol $\begin{Bmatrix} j1 & j2 & j3 \\ j4 & j5 & j6 \end{Bmatrix}$

8.10 Number theoretical functions

Mod[*n*, *m*]
the integer n modulo the integer m.

Gcd[*n1*, *n2*, ...]
the greatest common divisor of the integers *n1*, *n2*, ...

Divis[*n*]
a list of the integer divisors of an integer n.

Nfac[*n*]
a list of the prime factors of an integer or rational number n, together with their exponents.

Prime[*n*]
the n th prime number

Sti1[*n*, *m*]
First kind Stirling numbers $S_n^{(m)}$ [AS 24.1.3].

Sti2[*n*, *m*]
Second kind Stirling numbers $\mathrm{S}_n^{(m)}$ [AS 24.1.4].

Mob[(*k*:1), *n*]
Mobius μ function $\mu_k(n)$ of order k [AS 24.3.1].

Jacsym[*p*, *q*]
Jacobi symbol $\left(\frac{p}{q}\right)$ [BMP 17.5].

Divsig[(*k*:1), *n*]
Divisor function $\sigma_k(n)$ ($\sigma_0(n)=d(n)$) [AS 24.3.3].

ManL[*n*]
Mangoldt Λ function [BMP 17.1.1].

Part[*n*]
Partition function [AS 24.2.1].

Rrs[*n*]
List containing reduced residue system modulo n.

Totient[*n*]
Euler's totient function $\varphi(n)$ [AS 24.3.2].

Lio [*n*]
Liouville's function $\nu(n)$ [BMP 17.1.1].

Jor [*k*, *n*]
Jordan's function $J_k(n)$ (kth totient of n) [BMP 17.1.1].

• **Comb** [8.6]

9. Mathematical operations

9.1 Polynomial manipulation

Polynomials consist of sums of powers of "base" expressions. The bases in an expression are by default taken as the literal first filters of Pow projections.

• Polyp [7.6.]

Pdiv[*expr1*, (*expr2*:1), (*form*)]
the polynomial quotient of *expr1* and *expr2* with respect to the "base" *form*.

Pmod[*expr1*, *expr2*, (*form*)]
the remainder from division of *expr1* by *expr2* with respect to *form* (polynomial modulus).

• Mod [8.10]

Pgcd[*expr1*, *expr2*, (*form*)]
greatest common divisor of the polynomials *expr1* and *expr2* with respect to *form*.

• Gcd [8.10]

Fac[*expr*, (*lev*:1), (|*form1*|), (*crit*:1), (|*rep1*|)]
factors polynomials appearing at or below level *lev* in *expr* (and not yielding "false" on application of the template *crit*) with respect to "bases" matching *form1*, *form2*, .. The smallest available bases are taken as default. (The replacements *rep1*, *rep2*, .. will specify polynomial equations defining algebraic extensions to the default real integer factorization field.)

• Nfac [8.10]

Pf[*expr*, (*form*)]
yields a partial fraction form of *expr* with respect to the "base" *form*.

9.2 Evaluation of sums and products

Sum[*expr*, *var*, (*start*:0), (*end*:Inf), (*step*:1), (*test*:1)]
sums the values of *expr* obtained when *var* takes on values from *start* to *end* with increment *step* (and such that the value of *test* is not 0).

Prod[*expr*, *var*, (*start*:0), (*end*:Inf), (*step*:1), (*test*:1)]
forms the product of the values of *expr* obtained when *var* takes on values from *start* to *end* with increment *step* (and such that the value of *test* is not 0).

Sums and products with infinite limits may be evaluated numerically with an N projection [3.4].

• Do [6.2]

9.3 Solution of equations

Sol[|*eqn1*|, |*form1*|, (|*elim1*|)]
takes the equations *eqn1*, .. (represented as Eq projections [6]) and yields a list of simplified equations or, if possible, replacements giving solutions for forms matching *form1*, .. after eliminating forms matching *elim1*, .. where possible. Undetermined parameters in solutions appear as indices in the resulting list.

Solutions for classes of equations may be defined by assignments for the relevant Sol projections [3.2]. The assignment `Sol[f[$x]=$y,$x]::Sol[$x=fi[$y],$x]` thus defines an "inverse" for the "function" `f`.

• Mdiv [9.6]

9.4 Differentiation and integration

A "variable" is an expression containing a single symbol (either on its own or in a projection; in the latter case the necessary Jacobian factors are extracted).

D[*expr*,{*var1*,(*n1*:1),(*pt1*:*var1*)},{*var2*,(*n2*:1),(*pt2*:*var2*)}...]
forms the partial derivative of *expr* successively *ni* times with respect to the "variables" *vari*, evaluating the final result at the point *vari* → *pti*.

Dt[*expr*,{*var1*,(*n1*:1),(*pt1*:*var1*)},{*var2*,(*n2*:1),(*pt2*:*var2*)}...]
forms the total derivative of *expr* with respect to the variables *vari*.

Dt[*expr*]
forms the total differential of *expr*.

Derivatives which cannot be performed explicitly are converted into a canonical form with internally-generated symbols [2.2] for the *vari*, and explicit values for *ni* and *pti*.

Derivatives may be defined by assignments for the relevant D or Dt projections. D[f[$x,$y],{$x,1,$z}]:g[$z,$y] defines the derivative of the "function" f with respect to its first "argument".

In D projections, distinct symbols are assumed independent, while in Dt projections, they are assumed to be interdependent, unless the corresponding derivative has explicitly been assigned the value 0. Symbols or projections carrying the property Const [4] are assumed independent of all variables.

N projections [3.4] yield when possible numerical values for derivatives with definite *pti*.

Int[*expr*,{*var1*,(*lower1*),(*upper1*:*var1*)}...]
forms the integral of *expr* successively with respect to the variables *var1*,.. between the limits *lower1*,.. and *upper1*,... If no lower limit is specified, an internally-generated symbol [2.2] is used.

Integrals which cannot be performed explicitly are converted into a canonical form with internally-generated symbols for the *vari*.

Integrals may be defined by assignments for the relevant Int projections.

All distinct symbols are assumed independent.

N projections [3.4] yield when possible numerical values for integrals with definite limits *loweri* and *upperi*.

9.5 Series approximations and limits

Ps[(*expr*:1),{*var1*},{*pt1*},{{*sord1*:0),*ord1*},
(*ser*:{[*sord1*]:0,..,[-1]:0,[0]:1,[1]:0,..,[*ord1*]:0})]
Power (Taylor-Laurent) series in *var1*,... about the points *pt1*,... to order *ord1*,.... Terms proportional to *var1^j1 var2^j2* ... are given when *j1,j2*,... lie within a simplex with vertices *(j1:sord1)*, *(j1:ord1, j1:sord1)*, *(j2:ord2, j1:sord1)*, ... *ser[i]* gives the coefficient of *var1^i* in the power series.

Ra[*expr*, *var*, *pt*,{*degn*,(*degd*:*degn*)},(*crit*:$1=*degn*&&$2=*degd*),
(*sern*:{[0]:1,..,[*degn*]:0}),(*serd*: {[0]:1,..,[*degd*]:0})]
Rational (Pade) approximants in *var* about the point *pt*, to order *degn* in numerator and *degd* in denominator series. All order *(m,n)* approximants with *m+n<degn+degd* such that application of the template *crit* to *m,n* yields "true" are given (in a list if necessary). *sern[i]* is the coefficient of *var^i* in the numerator series, and *serd[j]* of *var^j* in the denominator.

Cf[*expr*,(*var*:1),(*pt*:0),{(*sord*:0),(*ord*:0)},(*ser*:{[0]:1,[1]:0,..,[*ord*]:0})]
continued fraction approximation in *var* about the point *pt*. *ser*[*i*] gives the coefficient of *var* in the *i*th partial quotient of the continued fraction.

expr gives an overall factor for the series. Input Ps, Ra and Cf projections are simplified so that all possible terms are transferred from *expr* to coefficients in the series *serk*.

Arithmetic and mathematical operations and substitutions (compositions) may be performed on series approximations: the results are taken to the highest permissible order.

Ax[*expr*]
yields an ordinary expression obtained by truncating all higher order terms in the series approximation *expr*.

If the expression *expr* in Ps,Ra and Cf is a series approximation, it is converted to the specified form, maintaining the highest permissible order.

Series approximations may be defined by assignments for Ps projections. Ps[exp[$x],$x,0,$n]:Ps[1,$x,0,$n,{[$i]:1/$i!}] defines the power series for the exponential function around the origin. Ra and Cf use assignments made for Ps projections.

Numerical values for series approximations are obtained using N.

Lim[*expr*,*var*,*pt*]
forms the limit of *expr* as *var* tends to *pt*. A sequence of Ps projections of increasing order are formed until a definite limit is found.

9.6 Matrix and explicit tensor manipulation

Outer[*temp*,*list1*,*list2*,...]
forms the generalized outer "product" of the lists *list1*, *list2*, .. with respect to the template *temp*. If entries in the lists t and u are specified as t[i1,i2,..,ik] and u[j1,j2,..,jk] then Outer[f,t,u] is a list whose entries are given by Ap[f,{t[i1,i2,..,ik],u[j1,j2,..,jk]}]

Inner[(*temp1*:Mult),*list1*,*list2*,(*temp2*:Plus)]
forms the generalized inner "product" of *list1* and *list2* with respect to the templates *temp1*,*temp2*.
• Dot [8.2]

Trans[*list*, (*levs*:2),]
yields a list obtained from *list* by transposing entries between two levels specified by *levs* (the *ni* are positive integers):

n	1 and *n*
{*n1*,*n2*}	*n1* and *n2*

Tr[*list*, (*temp*:Plus)]
the generalized trace obtained by applying *temp* to the set of entries in *list* whose indices are all equal.

Sig[*list*, *base*]
the signature of the permutation between *base* and *list* (0 if no permutation suffices). (If *base* is omitted, entries of *list* may appear directly as filters for *Sig*).
• Asym [7.7]

Det[*list*]
the determinant of a "full" *list*.

Minv[*list*]
the inverse of a non-singular matrix represented by *list*.

Mdiv[*list1*, *list2*]
yields a matrix *mat* such that *list2*. *mat* is equal to *list1*.

Triang[*list*]
gives the triangularized form of a matrix represented by *list*.

Eig[*list*]
yields a list of eigenvalues and normalized eigenvectors for the matrix *list*.

Simtran[*list*]
the similarity transformation matrix necessary to diagonalize *list*.

10. Non-computational operations

10.1 Input and output operations

Lpr [*expr*, (*file*:Null)]
prints *expr* to the specified file [10.3] (terminal as default) in a direct linear format suitable to be used as input, and in which labelling of parts is manifest [2.10]. It yields Null as an image.

Pr [*expr1*, (*expr2*, ...)]
prints *expr1*, *expr2*, .. in turn (separated by tabs) with standard two-dimensional format [2.12], and yields the last *expri* as an image.

Prh [*expr1*, (*expr2*, ...)]
prints the unsimplified forms of the *expri*.

Rd [(*prompt*:Null), (*file*:Null)]
prints the expression *prompt*, then reads and simplifies one line of input (terminated by newline) from the specified file [10.3]. Default is input from standard input/output medium (usually terminal). A null line is read as Null.

Rdh [(*prompt*:Null), (*file*:Null)]
prints the expression *prompt*, then reads one line of input from *file* and yields its "held" [3.5] form.

Fmt [(*prspec*:Null), *expr1*, *expr2*, ...]
yields a print form with the *expri* in a format specified by *prspec*:

Null	*expr1* followed immediately by *expr2*...
positive integer	*expr1* followed by *expr2*... after *prspec* blank spaces.
list	*expri* appear with horizontal and vertical offsets defined by *prspec*[*i*], according to {*hori*, *veri*}.expri with equal horizontal or vertical offsets are aligned. Those with larger horizontal offsets are further to the right, and those with larger vertical offsets are higher up. If no entry exists in *prspec* for a particular *expri*, it appears immediately to the right of the last printed expression. A horizontal or vertical offset Inf specifies a position to the right or above all other expressions.

Sx [(*cc*:), {*x1*, *x2*, ...}, (*class*:1), (*prec*:1)]
yields a print form with the *xi* appearing in association with ⊂cc⊃ with a syntax and precedence defined by *class* and *prec* as specified in [2.11].

Special output forms may be defined by assigning a suitable print form as the value of _s[Pr] [4].

10.2 Graphical output

Graph [{(*x1*), *y1*, (*z1*)}, {*u*, (*v*)}, {*umin*, (*vmin*)}, {*umax*, (*vmax*)}, {*form1*}, {(*xv*:0), (*yv*:0), (*zv*:Inf)}, {*upt*, (*vpt*)}, {({(*xmin*), *xmax*}), {(*ymin*), *ymax*}, ({(*zmin*), *zmax*})}]
generates a Plot projection which prints as a plot of curves or surfaces defined by the numerical values of *xi*, *yi* and *zi* as functions of the parameters *u* and *v*, between *umin* and *umax* (with *upt* samples), and *vmin* and *vmax* (with *vpt* samples). *xv*, *yv*, *zv* specify the point of observation for three-dimensional plots (contour plots by default). The *formi* define the style of curves plotted: integer codes give standard curve styles; other *formi* are printed explicitly on the curves. Only points in the region bounded by *xmin*, *xmax*, *ymin*, *ymax*, *zmin*, *zmax* are plotted.

Plot[*plist*,{(*xv*:0),(*yv*:0),(*zv*:Inf)},
{{(*xmin*),*xmax*},{(*ymin*),*ymax*},{(*zmin*),*zmax*}}]
prints as a plot containing points, lines, curves, surfaces and regions specified in *plist*. Ranges of coordinates default to include all forms given in *plist*. *xmin*, *xmax*, ... define boundaries of the region in which points are plotted. In two-dimensional plots, forms given later in *plist* overwrite those given earlier when they overlap. In three (and higher) dimensions, explicit intersections and perspective are used.

High-resolution graphics output is generated if a suitable device is available. **Plot**[] clears the plotting area. (Implementation dependent)

The list *plist* (whose sublists are flattened) contains:

Pt[{*x*,*y*,(*z*)},(*form*)]
represents a point with coordinates *x*, *y* and *z*, to be printed as *form*. When *form* does not print as a single character, the coordinates are taken to specify its lower left-hand corner.

Line[*ptlist*,(*form*)]
represents a succession of straight lines between the point specified by **Pt** projections in the list *ptlist*: *form* specifies the style of line.

Curve[*ptlist*,(*form*)]
represents a smooth curve through the points specified by **Pt** projections in the list *ptlist*: *form* specifies the style of curve.

Zone[{*clist*},*form*]
represents the interior of a region bounded by curves or lines specified in *clist* (with end points identified) to be filled with a texture or colour *form*. Infinity is taken as exterior.

Node[{*x*,*y*,*z*},{{*u1*,*v1*},{*u2*,*v2*},...},{(*fconti*:*z*),...},
{(*bconti*:*u*),(*bconti*:*v*),...}]
represents a node with coordinates *x*,*y*,*z* connected to nodes with parameters *u1*,*v1*, *u2*,*v2*, .. in a triangular network. Contour lines with integer spacing in each of the additional coordinates *fconti* are drawn on the front of the surface defined by the triangular network (and represented by a **Surf** projection). *bconti* specify contours for the back of the surface. The default back contour lines correspond to a square grid in the *u*,*v* parameter space. The front normal to a surface and the directions of increasing *u* and *v* respectively are taken to form a right-handed triple.

Surf[{[*u1*,*v1*]:*node1*,[*u2*,*v2*]:*node2*,... },(*form*)]
represents a surface spanned by a triangular network defined by **Node** projections *nodei* with parameter values *ui*,*vi*, and with a texture or colour specified by *form*.

Hull[{*pt1*,*pt2*...},*form*]
represents the surface formed by the convex hull of the points *pt1*, *pt2*,... with a texture or colour specified by *form*.

Axes[{(*x*:Inf),(*y*:Inf),(*z*:Inf)},{(*xtemp*),(*ytemp*),(*ztemp*)}]
represents a labelled set of orthogonal axes intersecting at **Pt**[*x*,*y*,*z*]. *xtemp* is a template applied to *xmin* and *xmax* to obtain a list of *x* values at which the x axis is to be labelled. *ytemp* and *ztemp* are analogous templates for the y and z axes. If *xtemp*, *ytemp*, *ztemp* are omitted, a heuristic procedure is used.

10.3 File input and output

Files are specified by single symbols (with names enclosed in " " if necessary [2.2]). The portion of a file after the *n*th input line is labelled by {*filespec,n*}. {*filespec,-n*} specifies the portion following the *n*th line from the end of the file. Output is by default appended to files. Specification of {*file,0*} causes new output to overwrite existing contents of *file*. The standard input/output medium (usually terminal) is considered as a special file denoted by the symbol Null. Input and output characteristics of a file are specified by a numerical code (defined in [A.3]).

<*file* or **Input[*file*]**
reads input up to the first ⊂*input termination character*⊃ from the specified file. If ambiguous syntax [1.1,2] is encountered, a message is printed, and no further input occurs. If *file* is successfully input, the projection yields the last output line generated. (In some implementations [A.3], Input may be used to load a binary file into the data space of the current job.)
• Rd, Rdh [10.1]

Output[*expr1*, (*expr2*, . . .), (*file*:Null)]
outputs assignments defining values given for the *expri* to the specified file in a form suitable for subsequent input.

Open[*file*, (*code*:1)]
initiates entry of all subsequent input and output expressions into the specified file, in a mode defined by *code* [A.3].

Close[*file1*, *file2*, . . .]
terminates entry of input and output expressions into the specified files.

Close[] stops the printing of any output on the standard output medium until Open[] occurs.

• [1.3]

10.4 Memory management

(Implementation dependent)

Gc[]
reclaims memory not required for further processing (by "compacting garbage collection").

10.5 External operations

Init[(*expr1*, *expr2*, . . .)]
defines various external parameters as specified in [A.3].

Exit[(*expr*)]
terminates the current job, passing the textual form of *expr* as an "exit code" [A.3] to the monitor (shell).

Run[*expr*, (*arg1*, *arg2*, . . .)]
executes the textual form of *expr* (printed by Lpr [10.1]) as a monitor (shell) program [A.2], using the textual forms of the *argi* as input [A.2]; the text of any output [A.2] generated is simplified and given as the image [1.10].

• [1.6]

10.6 Character string manipulation

Ed[*expr*]
enters edit mode [1.7], with the textual form of ***expr***, as printed by Lpr, in the edit buffer, and yields as a result the edited expression.

Edh[*expr*]
enters edit mode with the text of a partially simplified form [3.5] of ***expr*** in the edit buffer.
• Ev [3.7]

Make[(*start*:#),(*expr*:*(next integer)*)]
generates a symbol with name obtained by concatenating the textual form of ***start*** with the textual form of ***expr***, or, by default, with the smallest positive integer necessary to form a previously unused name.

Expl[*expr*]
gives a list of numerical codes for each of the characters appearing in the textual form of ***expr*** (as printed by Lpr). Characters are numbered from 0 to 94 in the order:
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
⊂space⊃!"#$%&\'()*+,-./:;<=>?@[\]^_\`{|}~

Impl[{*n1*,*n2*,*n3*,...}]
generates a symbol whose name consists of the characters specified by the numerical codes ***n1***, ***n2***, ***n3***,...

10.7 Programming aids

Step[*expr*,(*nest*:1)]
steps through the simplification of ***expr***. Each segment in procedures [6.3] or iteration structures [6.2] nested to depth less than ***nest*** is printed, and an interactive subsidiary procedure [1.8] is initiated.

• Trace [4]
• Ev [3.7]

Struct[*expr*]
prints a schematic picture of the internal representation of ***expr***.

10.8 Performance analysis

State[]
yields a list giving the number of memory "blocks" used (after last memory reclamation, at present, and maximum so far). One block is the memory required to store a single symbol (usually 16 bytes [A.5]).

• [1.4]

Size[*expr*]
yields a list whose first entry is the actual number of memory blocks occupied by ***expr***, and whose second entry is the number which would be occupied if all common subexpressions were stored separately.

Time[*expr*,(*n*:1)]
simplifies ***expr*** ***n*** times, and yields an Err projection [2.1] giving the approximate average CPU time in "clicks" [A.5] used for each simplification.

10.9 Program construction

(Implementation dependent)

Cons [{*f1*, . . .}, {(*file1*:), . . .}, (*code*:0)]
constructs if possible an external program which obtains the values assigned to projections of the *fi*. Resulting programs are placed in the files *filei*. The language and treatment of the external programs is determined by *code* as specified in the implementation notes. With certain *code*, the external programs assume that all symbols take on numerical values. Non-local variables are assumed to have constant values. Projections whose evaluation may be carried out by external programs carry property **Cons**.

10.10 Asynchronous and parallel operations

(Implementation dependent)

Some implementations allow a set of independent processes to be performed in parallel, either as asynchronous jobs on a single computing unit, or as jobs in separate computing units.

Processes (including procedures within them) are specified by a unique expression used as a name: the basic process is **Null**. A particular expression may be modified by only one of a set of parallel processes. The order of operations in different processes is usually not determined.

Fork [(*expr*:**Null**), (*name*: *(next integer)*), (*pri*:1)]
initiates the named parallel process to simplify *expr* at priority *pri*, yielding *name*. If *name* is not specified, a unique integer name is assigned to the process. Any existing process *name* is terminated. Several processes competing for a single computing unit are executed at higher priorities for lower *pri*. Processes on separate computing units are executed when possible with instruction times in ratios given by *pri*.

Wait [{*name1*, *name2*, . . .}]
waits for completion of the processes *name1*, *name2*, ..., yielding the resulting {*expr1*, *expr2*, . . .}

Para [*expr1*, *expr2*, . . .]
is equivalent to **Wait** [{**Fork** [*expr1*], **Fork** [*expr2*], ...}] and simplifies the *expri* in parallel, yielding a list of the results.

Fork [*mess*, *code*] may be used to transfer *mess* to **Wait** [*code*] in another process.

Fork [, *name*] terminates the process *name*, possibly from within *name*.

Clock [(*name*: *(present process)*)]
yields a list of the total elapsed computing unit time (in clicks) and total elapsed real time (in seconds) since the initiation of the specified process (0 if the process is not executing).

Rti [*code*]
represents a real-time interrupt whose value is simplified immediately on receipt of the interrupt *code*.

- [1.4]

Appendix. External interface

A.1 Introduction

This appendix describes in general terms features of SMP affected by its external enviroment. Details of these features vary between different implementations of SMP. Information for a particular implementation should be given in the "Implementation Notes". Mechanisms for features under different operating systems will not be described; they are discussed in the "SMP Implementation Guide" (available separately).

A.2 External operations

Typical external operations provided in SMP implementations are:

Hard[(*expr*),(*code*)]
generates a hard copy of *expr* on the device specified by *code*. Hard[] yields a hard copy of all input and output expressions. Graphics output is given if possible.

Dsp[(*file*:smp.out)]
prints the specified file.

Save[(*rec*:smp.out),*file*]
creates a permanent copy *file* of the record file *rec*.

Send[(*uname*)]
enters send mode: arbitrary text terminated by ⊂*input termination character*⊃ is sent to the location or user identified by *uname*. ⊂*break interrupt*⊃ may be used to include SMP expressions. Send[] sends the text to a central SMP report file at each installation.

Dir[*dir*]
changes the default "user" file directory [A.4] to *dir*. Dir[] resets to the directory given at initialization.

A.3 Terminal characteristics

The following are ASCII equivalents for non-alphabetic characters used in this handbook: ! (041) 042 () # (043) $ (044) % (045) & (046) ' (047) ((050)) (051) * (052) + (053) , (054) - (055) . (056) / (057) : (072) ; (073) < (074) = (075) > (076) ? (077) @ (100) [(133) 134 ()] (135) ^ (136) _ (137) ` (140) { (173) | (174) } (175) ~ (176)

Replacements for input text may be specified using Sxset [2.11].

Characteristics of a terminal or file may be specified in Open or Init as a list whose entries usually include

Number of lines printed before pause for end of page. (Input of newline continues printing). (Inf for no pause).

Position of first character to be printed on left.

Position of last character to be printed on right.

Hardware tab spacing (0 if none).

Type of graphics mode (0 if none).

Graphics mode entry code.

Graphics mode return code.

Screen clear/form feed code.

Many other parameters may be provided in a particular implementation.

Common classes of terminals may be specified by a single integer code, as defined in the implementation notes.

A.4 External files

If no explicit file directory is specified, external files are first assumed to be in a default "user" directory, and failing that in a central "library" directory. The default directories are specified by `Init` [A.5].

The names of external files provided with releases of SMP all begin with the letter `X`. Names of new external files should begin with letters other than `X`. Files whose names end with `SX` contain syntax modifications [2.11].

Most external files contain SMP input lines. In some versions of SMP, external files may also contain direct binary forms of SMP expressions. Such files are recognized and treated appropriately by `Input` [10.3]. The names of binary files should usually end with `#B` or `.B`

The record file [1.3] for each SMP job is placed in the "user" directory, and usually named `smp.out`

A.5 Initialization

`Init`[(*udir*:),(*libdir*:),(*term*:)]
: specifies default "user" and "library" directories, and defines characteristics of the terminal.

If provided, SMP jobs read an initialization file, usually named `smp.init`

When an SMP job is initiated, it is often possible to pass "arguments" to the job, giving files to be read for initialization (after `smp.init`).

When an SMP job terminates, it passes by default a "successful completion" exit code to the monitor; other exit codes may be specified in `Exit`.

If provided, a termination file `smp.end` is read before termination of an SMP job.

A.6 System characteristics

The "block" is the basic unit of memory used by SMP. Its physical size in terms of bytes may vary from one implementation to another: in most cases, one block is 16 bytes.

- `State` [10.8]

The "click" is the basic unit of CPU time used in SMP. A "click" is defined by the time taken to execute certain initialization procedures; the time for operations in different installations should be roughly a fixed number of "clicks". On a VAX 11/780 one "click" corresponds to approximately one second.

- `#T` [1.2]
- `Time` [10.8]

• Clock [10.10]

A.7 External programs

External programs may be entered explicitly or may be constructed from SMP definitions using Cons [10.9]. They may be run explicitly using Run [1.10,10.9], or may be defined by Cons to be used automatically by SMP in simplifying particular projections. Each invocation of an external program receives only one set of parameters from Run or its associated projection. External programs may usually return any number of input lines to SMP. The simplification of a projection involving an external program is complete when the external program terminates.

External programs invoked by Run may communicate with SMP by one of several mechanisms:

1. Take input and output on the standard input and output media, but pass them through pre- and post-processors which direct them to and from SMP.
2. Use the SMPIO library functions to obtain input directly from SMP, and pass output directly to SMP. In this case, additional input and output may occur on the standard input/output medium.

The SMPIO library is usually included in external programs by an option for the compiler used.

For the C language the SMPIO library contains the functions `fromsmp` and `tosmp`, analogous to `scanf` and `printf` respectively. The following conversion characters may be used in the control string:

`%n` Decimal number (in SMP *^ format [2.1]).

`%s` Character string.

`fromsmp` and `tosmp` usually use the input-output channel 3.

External programs constructed with Cons are usually linked directly as the values of projections in a running SMP job.

INDEX

SMP library topic directory

A. Mathematics

1. Fundamentals

XAck Ackermann function
XDios Solution of Diophantine equations
XFib Fibonacci numbers
XGr Basic graph theory
XLCM Lowest common multiple
XLogic Elementary laws in propositional calculus
XLogic2 Elementary logic with quantifiers
XLogicPr Logical truth tables
XSets Elementary finite set theory
XSetsSX Set theory notation
XTup n-tuples

2. Algebra

XArperm Generation of permutations
XBase Conversion of integers in arbitrary number bases
XFrac Fractions
XMat1 Matrix input and generation
XMat2 Structural matrix operations
XMat3 Matrix character tests
XMat4 Algebraic matrix operations
XPerm0 Permutations
XPerm1 Elementary operations on permuations
XPermC Cycle decomposition of permutations
XRslt Polynomial resultants
XSympol Generate symmetric polynomials

3. Analysis

XAbs Extensions for Abs
XAir Airy and related functions
XAng Anger and Weber functions
XBer Bernoulli polynomials
XBes1 Functional relations for Bessel functions
XBes2 Recurrence relations for Bessel functions
XBes3 Bessel functions of integer order
XBes4 Bessel functions of half odd integer order
XBes5 Special cases for half odd integer order Bessel functions
XBeta Euler beta function
XChe Chebychef polynomials
XChg Confluent hypergeometric function
XDSol Series solution of differential equations

7. Astrophysics and gravitation
   - `XLevi` Generate Levi-Civita tensor
8. Chemistry
9. Earth sciences
10. Physical quantities
    - `XDim` Dimensional analysis
    - `XMKS` MKS/SI units
    - `XNAT` Natural units
    - `XPhys` Fundamental physical constants

C. Life and social sciences
1. Biology
2. Medicine
3. Sociology
4. Economics

D. Technology
1. Mechanical engineering
2. Civil engineering
3. Hydraulic and aeronautical engineering
4. Electrical and optical engineering
5. Chemical engineering
6. Systems engineering

E. Miscellaneous

SMP library section directory

3. Replacements and substitutions
4. Numerical evaluation
5. Deferred simplification
6. Pre-simplification
7. Partial simplification

4. Properties

5. Relational and logical operations
   - XLogic Elementary laws in propositional calculus

6. Control structures
   1. Conditional statements
   2. Iteration
   3. Procedures and flow control

7. Structural operations
   1. Projection and list generation
      - XArperm Generation of permutations
      - XTup n-tuples
   2. Template application
      - XDap Directional application
      - XNMap Multi-element generalization of Map
   3. Part extraction and removal
      - XLev Isolate single level
   4. Structure determination
      - XLenex Length of expanded expressions
      - XLPart List positions of all parts
   5. Content determination
      - XAny Test for any elements of list satisfying condition
   6. Character determination
      - XGenp Test for generic symbols
      - XIntp Additional rules for integer testing
   7. List and projection manipulation
      - XContig Make list contiguous
      - XDap Directional application
      - XInd Manipulation of indices in lists
      - XLArith Arithmetic operations on lists
      - XList0 Basic list manipulations
      - XList1 Operations on sublists
      - XMaxind Find maximal index
      - XPeel Peel away sublists
      - XProj Projection manipulation
      - XUnFlat List unflattening

7. Confluent hypergeometric and related functions
   - XAir Airy and related functions
   - XAng Anger and Weber functions
   - XBes1 Functional relations for Bessel functions
   - XBes2 Recurrence relations for Bessel functions
   - XBes3 Bessel functions of integer order
   - XBes4 Bessel functions of half odd integer order
   - XBes5 Special cases for half odd integer order Bessel functions
   - XChg Confluent hypergeometric function
   - XHer Hermite polynomials
   - XKumU Kummer U function
   - XLag Laguerre polynomials
   - XLom Lommel function
   - XPar Parabolic cylinder function
   - XStr Struve functions
   - XWhi Whittaker function
8. Hypergeometric and related functions
   - XChe Chebychef polynomials
   - XGeg Gegenbauer polynomials
   - XHg1 Hypergeometric functions - 1
   - XHg2 Hypergeometric functions - 2
   - XHg3 Hypergeometric functions - 3
   - XHg4 Hypergeometric functions - 4
   - XHg5 Functional relations for hypergeometric functions
   - XJac Jacobi polynomials
   - XLegP Legendre polynomials
   - XOp1 Orthogonal polynomials - 1
   - XOp2 Orthogonal polynomials - 2
   - XOp3 Orthogonal polynomials - 3
   - XOpR Rodrigues formulae for orthogonal polynomials
9. Further special functions
10. Number theoretical functions
    - XAbs Extensions for Abs
    - XAck Ackermann function
    - XFib Fibonacci numbers
    - XHarm Harmonic sequence
    - XLCM Lowest common multiple

9. Mathematical operations
   1. Polynomial manipulation
      - XPoly Information on polynomials
      - XRslt Polynomial resultants

2. Evaluation of sums and products
   - XIter General iterated forms
   - XSum Summation of series
3. Solution of equations
   - XDios Solution of Diophantine equations
   - XLdEq Lists of equations
   - XSol Inverses of elementary transcendental functions
4. Differentiation and integration
   - XInt Elementary definite integrals
   - XVecan Three-dimensional vector analysis
5. Series approximations and limits
6. Matrix and explicit tensor manipulation
   - XCon Tensor contraction
   - XDap Directional application
   - XLevi Generate Levi-Civita tensor
   - XMat1 Matrix input and generation
   - XMat2 Structural matrix operations
   - XMat3 Matrix character tests
   - XMat4 Algebraic matrix operations
   - XNorm Norm of a vector
   - XTEx Tensor expansion

10. Non-computational operations
    1. Input and output operations
    2. Graphical output
       - XPlhist Plot histogram
       - XRot2 Rotations in two dimensions
       - XRot3 Rotations in three dimensions
    3. File input and output
       - XWatch Watching external file input
    4. Memory management
       - XKillIO Kill Input/Output
    5. External operations
    6. Character string manipulation
       - XChar Character manipulation
       - XStr0 Basic character string manipulation
       - XStr1 Further character string manipulation
    7. Programming aids
    8. System performance analysis
    9. Program construction
    10. Asynchronous and parallel operations

SMP Primer

Anthony E. Terrano

and

Stephen Wolfram

CONTENTS

0. Preliminaries

This and the following sections provide a pedagogical introduction to the basic features of SMP. Knowledge of these features suffices for many applications of SMP. Reference is made when appropriate to the "SMP Reference Manual", which gives a complete and systematic description of the facilities in SMP.

This primer assumes no prior experience with computer systems. A glossary of some technical terms used is given as an appendix.

As with all computer systems, SMP is most effectively learned through use. The reader is therefore strongly encouraged to experiment on an actual SMP system.

Many superficial features of SMP differ from one installation to another. The details pertinent to a particular installation should be given in the "Implementation Notes" section of this handbook.

First find a suitable terminal, connect it to the computer, and log in (as described in the "Implementation Notes"). A video terminal will probably be much more convenient than a printing one; SMP always keeps a record of your work. On video terminals there is usually a "cursor" (often a square or an underscore) which indicates the position at which the next character will appear. Under normal circumstances, any character typed on the terminal should appear at this position*. (If this fails to happen, check that the terminal is connected to the computer, and that it is not on "local". Terminals sometimes become "locked": switching power off momentarily may "unlock" them. If each character is printed twice, switch the terminal or telephone modem from "half duplex" to "full duplex".) Next identify the characters on the terminal referred to in the "Implementation Notes". "Control" characters are typed in direct analogy with upper-case "shifted" characters: hold down the key marked "CTRL" and press the required character.

On most systems, no typed input is processed until a ⊂*newline*⊃ is entered. Except for very complicated operations, the computer should respond to any input within a few seconds; if an expected response is not forthcoming, the computer is probably waiting for ⊂*newline*.⊃ (An extra ⊂*newline*⊃ is never detrimental.) Before the ⊂*newline*⊃ is entered, text on a line may be deleted and retyped. ⊂*character delete*⊃ causes the last character typed to be discarded; typing two ⊂*character delete*⊃ discards the last two characters, and so on. When a character is discarded, the cursor usually backspaces over its position (however, on some systems, the discarded character is reprinted). A replacement character may then be typed. ⊂*line delete*⊃ discards all characters typed so far on the present line.

Characters in examples given below may be different for particular systems. Any such differences are listed in the "special characters" section of the "Implementation Notes".

Notice that upper and lower case letters are distinguished in SMP. (If your terminal does not allow lower case letters, the "Implementation Notes" will give the necessary instructions.)

Now, using the instructions in the "Implementation Notes", start an SMP job.

---

* Except when a "password" is being entered.

1. Fundamentals

When SMP is called, it begins by printing*

```
SMP version 1

#I[1]::   ▫
```

placing the terminal cursor at the position marked ▫. With this prompt, the SMP job is ready to receive its first input. #I [1] is the name to be assigned to this first input line. As a first example of an input expression, type 2+5 followed by ⊂*newline*⊃

```
#I[1]::  2+5

#O[1]:   7

#I[2]::  ▫
```

SMP read the input expression 2+5, simplified it, and printed the result 7 as the first output expression #O [1]. It is now ready to receive a second line of input.

SMP simplifies any input line to which it can assign a unique meaning. If an input line is ambiguous or meaningless, SMP enters "edit mode" (see sect. 4 below), allowing the line to be modified. Typing two ⊂*newline*⊃ 's in succession exits the editor, discards the original line and causes the input prompt to be reissued. (If additional editor commands have unwittingly been entered, it may be necessary to type \q to exit the editor.)

```
#I[1]::  2++5
+ unexpected
         2++5
<edit>     ▫
         2++5
<edit>
#I[1]::  ⊂3+6

#O[1]:   9
```

In this example, the first ▫ marks the position of the cursor on entry to edit mode, and the second ▫ its position after exit from edit mode achieved by input of two ⊂*newline*⊃ 's

SMP does assign a unique meaning to many unintended input lines; in some cases the input may be output again with little or no modification, in others, surprising results may be obtained. In most such cases, the intended input may simply be entered as the next input line.

```
#I[1]::  1..5+3.2

#O[1]:   [1,2,3,4,5,6,7,8]

#I[2]::  1.5+3.2

#O[2]:   4.7
```

* On some systems, there may be a wait of several seconds between the request for SMP and this response.

At all stages in an SMP job, a ⊂*quit interrupt*⊃ terminates the computation or output, and issues a new input prompt.

SMP is usually very taciturn: it prints essentially no messages. A running commentary may be obtained by typing `On[]`. (The commentary is switched off by `Off[]`). Information on a particular object or topic is obtained by typing *?name*.

The appendix to this primer lists a variety of common difficulties, and should be consulted if unexpected behaviour occurs.

An SMP job is usually terminated by typing ⊂*input termination character*⊃ after an input prompt*. All input and output in an SMP job is saved in an external file for possible later use, as discussed in sect. 4.

Most mathematical operations and functions are represented in SMP in close analogy with their written forms. The standard arithmetic operations are typed as:

x+*y*	*x* plus *y*
x-*y*	*x* minus *y*
*x***y* or *x* ⊂*space*⊃ *y*	*x* multiplied by *y*
x/*y*	*x* divided by *y*
x^*y*	*x* raised to the power *y*.

Here *x* and *y* stand for numbers or other SMP expressions. Additional ⊂*space*⊃ 's may always be typed on either side of arithmetic operators. (Note that *x***y* represents the "outer product" of *x* and *y* not *x* raised to the power *y*.)

Arithmetic operations are performed in the conventional order: parenthesized expressions are evaluated first, followed by ^, /, *, -, +. ⊂*space*⊃ 's or parentheses usually suffice to indicate multiplication: no explicit * need be typed.

Divisions group to the left (so that `4/3/2` means `(4/3)/2`) but powers group to the right (`4^3^2` means `4^(3^2)`).

```
#I[1]::  1+2+7
#O[1]:   18
#I[2]::  1 +7-15
#O[2]:   -7
#I[3]::  2*3*4
#O[3]:   24
#I[4]::  2 3 4
#O[4]:   24
#I[5]::  -23 4
#O[5]:   -92
#I[6]::  1/2+7/3
#O[6]:   17/6
#I[7]::  242/12
```

* A computation may usually be terminated at an intermediate stage by typing ⊂*quit interrupt*⊃ (without an input prompt); the job may be terminated by ⊂*break interrupt*⊃ followed by `Exit[]` (after the prompt `%I[1]::`).

```
#O[7]:   121/6
#I[8]::  4^6
#O[8]:   4096
#I[9]::  (2+3)^2
#O[9]:   25
#I[10]::  2+3^2
#O[10]:   11
#I[11]::  2+3*4
#O[11]:   14
#I[12]::  3*4+2
#O[12]:   14
#I[13]::  3 4+ 2
#O[13]:   14
#I[14]::  3*(4+2)
#O[14]:   18
#I[15]::  3(4+2)
#O[15]:   18
#I[16]::  (1+2)(3+4) (5-6+7)
#O[16]:   126
#I[17]::  2/3 4
#O[17]:   8/3
#I[18]::  2/(3 4)
#O[18]:   1/6
#I[19]::  4^3^2
#O[19]:   262144
#I[20]::  (4^3)^2
#O[20]:   4096
#I[21]::  27^(1/3)
#O[21]:   3
#I[22]::  27^(1/2)
                1/2
#O[22]:   27
```

In lines 6 and 7 above, answers were given as rational numbers. On line 22, the answer could not be given as a simple rational number, and so was left in a symbolic form. In such cases, a decimal number result may be obtained using the SMP "projection" `N`. For any expression *expr*, `N[`*expr*`]` yields a real or complex numerical result if this is possible. Note the use of square brackets around *expr*, to be distinguished from the parentheses used to indicate grouping in expressions.

```
#I[1]::  1/2+7/3
#O[1]:   17/6
#I[2]::  N[1/2+7/3]
#O[2]:   2.833333
#I[3]::  27^(1/2)
                1/2
#O[3]:   27
#I[4]::  N[27^(1/2)]
#O[4]:   5.196152
#I[5]::  N[27^0.5]
#O[5]:   5.196152
```

SMP treats a large number of mathematical functions, including all common special functions of mathematical physics*. A list of the functions is given in [8]. (Here, as throughout this primer, references to sections of the Reference Manual and Summary are given in square brackets.) Any of the functions listed may be evaluated numerically† by use of N. Note that all "system-defined" or "built-in" functions in SMP have names which begin with a capital letter. Their "arguments" are enclosed in square brackets (just as for N), and separated by commas.

```
#I[1]::  Exp[2]
#O[1]:   Exp[2]
#I[2]::  N[Exp[2]]
#O[2]:   7.389056
#I[3]::  N[BesK[2,4.56]]
#O[3]:   .008898988
#I[4]::  N[BesJ[0.2,1.5] Sin[0.2^3]]
#O[4]:   .004834921
```

In line 3, for example, BesK [n, z] represents the modified Bessel function $K_n(z)$.
At all stages in an SMP job, the symbol % stands for the latest output line generated.

```
#I[1]::  1/2+7/3
#O[1]:   17/6
#I[2]::  %
#O[2]:   17/6
#I[3]::  %^2
#O[3]:   289/36
```

* Sect. 8 describes the procedure for defining values and characteristics of further functions.
† When branch cuts are present, all functions are evaluated on their principal Riemann sheets.

```
#I[4]:: N[%]

#O[4]:  8.027778

#I[5]:: %+%^3

#O[5]:  525.3797
```

SMP manipulates not only numbers, but also expressions containing symbolic parameters or "symbols". Symbols may be used to represent quantities whose numerical value is undetermined. Mathematical simplifications performed on expressions hold for any possible values of the symbols which appear in them.

Any sequence of letters and numbers (not starting with a number) may be used to denote a symbol. As throughout SMP, upper and lower case letters are distinguished. System-defined symbols, such as N, Pi (see below) and Exp, always begin with a capital letter. To avoid confusion, symbols introduced by the user should therefore begin with a lower case letter.

```
#I[1]:: x

#O[1]:  x

#I[2]:: x+x

#O[2]:  2x

#I[3]:: %-3x

#O[3]:  -x

#I[4]:: x1+x2-3x1/4

#O[4]:  x1/4 + x2

#I[5]:: %^2-4%

                                  2
#O[5]:  -x1 - 4x2 + (x1/4 + x2)

#I[6]:: (x+a)(x+b)-(x+y)^(a+b)/(2a+b)

                  a + b
        - (x + y)
#O[6]:  -------------- + (a + x) (b + x)
           2a + b

#I[7]:: (Exp[x^2]-1)Sin[x phi/4]

                    2          phi x
#O[7]:  (-1 + Exp[x ]) Sin[-----]
                                4
```

Notice that in, for example, line 3, 3x denotes 3*x. However, x2 in line 4 is a single symbol.

One use of symbols is to represent mathematical constants, for which numerical values are defined [8.4].

```
#I[1]:: Pi^2-9

                   2
#O[1]:  -9 + Pi

#I[2]:: N[%]

#O[2]:  .8696044
```

```
#I[1]:  1/2+7/3
#O[1]:  17/6
#I[2]:  N[1/2+7/3]
#O[2]:  2.833333
#I[3]:  27^(1/2)
                1/2
#O[3]:  27
#I[4]:  N[27^(1/2)]
#O[4]:  5.196152
#I[5]:  N[27^0.5]
#O[5]:  5.196152
```

SMP treats a large number of mathematical functions, including all common special functions of mathematical physics*. A list of the functions is given in [8]. (Here, as throughout this primer, references to sections of the Reference Manual and Summary are given in square brackets.) Any of the functions listed may be evaluated numerically† by use of N. Note that all "system-defined" or "built-in" functions in SMP have names which begin with a capital letter. Their "arguments" are enclosed in square brackets (just as for N), and separated by commas.

```
#I[1]:  Exp[2]
#O[1]:  Exp[2]
#I[2]:  N[Exp[2]]
#O[2]:  7.389056
#I[3]:  N[BesK[2,4.56]]
#O[3]:  .008898988
#I[4]:  N[BesJ[0.2,1.5] Sin[0.2^3]]
#O[4]:  .004834921
```

In line 3, for example, BesK $[n, z]$ represents the modified Bessel function $K_n(z)$.
At all stages in an SMP job, the symbol % stands for the latest output line generated.

```
#I[1]:  1/2+7/3
#O[1]:  17/6
#I[2]:  %
#O[2]:  17/6
#I[3]:  %^2
#O[3]:  289/36
```

* Sect. 8 describes the procedure for defining values and characteristics of further functions.
† When branch cuts are present, all functions are evaluated on their principal Riemann sheets.

```
#I[4]:  N[%]

#O[4]:  8.827778

#I[5]:  %+%^3

#O[5]:  525.3797
```

SMP manipulates not only numbers, but also expressions containing symbolic parameters or "symbols". Symbols may be used to represent quantities whose numerical value is undetermined. Mathematical simplifications performed on expressions hold for any possible values of the symbols which appear in them.

Any sequence of letters and numbers (not starting with a number) may be used to denote a symbol. As throughout SMP, upper and lower case letters are distinguished. System-defined symbols, such as N, Pi (see below) and Exp, always begin with a capital letter. To avoid confusion, symbols introduced by the user should therefore begin with a lower case letter.

```
#I[1]:  x

#O[1]:  x

#I[2]:  x+x

#O[2]:  2x

#I[3]:  %-3x

#O[3]:  -x

#I[4]:  x1+x2-3x1/4

#O[4]:  x1/4 + x2

#I[5]:  %^2-4%

                                    2
#O[5]:  -x1 - 4x2 + (x1/4 + x2)

#I[6]:  (x+a)(x+b)-(x+y)^(a+b)/(2a+b)

                    a + b
        - (x + y)
#O[6]:  -------------- + (a + x) (b + x)
           2a + b

#I[7]:  (Exp[x^2]-1)Sin[x phi/4]

                      2         phi x
#O[7]:  (-1 + Exp[x ]) Sin[-------]
                                 4
```

Notice that in, for example, line 3, 3x denotes 3*x. However, x2 in line 4 is a single symbol.

One use of symbols is to represent mathematical constants, for which numerical values are defined [8.4].

```
#I[1]:  Pi^2-9

                   2
#O[1]:  -9 + Pi

#I[2]:  N[%]

#O[2]:  .8696044
```

```
#I[3]::  Sin[Pi/2]

#O[3]:   1

#I[4]::  Sin[45Deg]

#O[4]:   Sin[45Deg]

#I[5]::  N[%]

#O[5]:   .7071068

#I[6]::  Exp[Euler]-1

#O[6]:   -1 + Exp[Euler]

#I[7]::  N[%]

#O[7]:   .7810724
```

The argument of the trigonometric function `Sin` in line 4 is in radians; `Deg` is a constant with value `Pi/180`. `Euler` is the Euler-Mascheroni constant γ=0.5772..

Some mathematical operations which may be performed on symbolic expressions are described in [9]. Examples are

`Ex[`*expr*`]`	"Expand" *expr* by using distributive rules for various functions.
`Fac[`*expr*`]`	Factor *expr*.
`D[`*expr*`,` *var*`]`	Form the partial derivative of *expr* with respect to *var*.
`Int[`*expr*`,` *var*`]`	Form the integral of *expr* with respect to *var*.

```
#I[1]::  (x+1)(x+a)(x+b)

#O[1]:   (1 + x) (a + x) (b + x)

#I[2]::  Ex[%]

                          2             2             2    3
#O[2]:   a b + a x + a x  + b x + b x  + a b x + x  + x

#I[3]::  Fac[%]

#O[3]:   (1 + x) (a + x) (b + x)

#I[4]::  Sin[x]/(1+x+x^2)

           Sin[x]
#O[4]:   ----------
                2
         1 + x + x

#I[5]::  D[%,x]

                 (1 + 2x) Sin[x]
         Cos[x] - ---------------
                             2
                    1 + x + x
#O[5]:   -------------------------
                         2
               1 + x + x

#I[6]::  Ex[%]

           Cos[x]              2x Sin[x]                Sin[x]
#O[6]:   ---------- - ------------------------ - ------------------------
                2                    2    3   4                2    3   4
         1 + x + x    1 + 2x + 3 x  + 2 x  + x    1 + 2x + 3 x  + 2 x  + x
```

```
#I[7]::  x/(1-4x+x^2)

              x
#O[7]:   -----------
                   2
         1 - 4x + x

#I[8]::  Int[%,x]

                             1/2                           1/2
               -4 + 2#1 - 12                 -4 + 2x - 12
         -2Log[------------------]    2Log[----------------]
                             1/2                           1/2                2
               -4 + 2#1 + 12                 -4 + 2x + 12       Log[1 - 4#1 + #1 ]
#O[8]:   ------------------------ + ---------------------- - ------------------
                     1/2                           1/2                 2
                   12                            12

                                 2
                   Log[1 - 4x + x ]
                 + ----------------
                          2
```

In line 8, the symbol #1 was introduced as a constant of integration.

A symbol may be assigned an expression as a value. ***symb*: *expr*** assigns the expression ***expr*** to be the value of the symbol ***symb*** [3.2]. After this assignment, the symbol ***symb*** becomes essentially just a short notation for its value ***expr***. Whenever ***symb*** appears, it is replaced by ***expr***.

```
#I[1]::  x

#O[1]:   x

#I[2]::  x:a+b

#O[2]:   a + b

#I[3]::  x

#O[3]:   a + b

#I[4]::  x^2 + x
                          2
#O[4]:   a + b + (a + b)

#I[5]::  y:x+1

#O[5]:   1 + a + b

#I[6]::  x-y

#O[6]:   -1

#I[7]::  y

#O[7]:   1 + a + b

#I[8]::  x

#O[8]:   a + b
```

Note that ***symb*: *expr*** performs the specified assignment, and then yields the result ***expr***. Thus the output from the assignment on line 2 was the assigned value a+b.

a:b:c assigns the value c to both a and b.

If an assignment is made for a symbol which already carries a value, the newly assigned value replaces the old one.

```
#I[1]::  x:2
#O[1]:   2
#I[2]::  x^2
#O[2]:   4
#I[3]::  x:3
#O[3]:   3
#I[4]::  x^2
#O[4]:   9
```

Once an assignment for a symbol, say x, has been made, it continues to be used throughout the SMP job. A common source of unexpected results is the use of symbols for which a value has been assigned much earlier in the job. *symb*: removes any value assigned to *symb*.

```
#I[1]::  x:2
#O[1]:   2
#I[2]::  x^2+x
#O[2]:   6
#I[3]::  x:
#I[4]::  x^2+x
                  2
#O[4]:   x + x
```

Assignment of a value for a symbol causes the symbol to be replaced by that value whenever it appears. Substitutions for a symbol in a particular expression may also be made. S[*expr*, *symb*->*subst*] substitutes the value *subst* for the symbol *symb* in the expression *expr*. The "arrow" is typed as - followed by >.

```
#I[1]::  x^2+x
                  2
#O[1]:   x + x
#I[2]::  S[%,x->a+b]
                              2
#O[2]:   a + b + (a + b)
#I[3]::  x
#O[3]:   x
#I[4]::  x:3+c
#O[4]:   3 + c
#I[5]::  x^2+x
```

```
#O[5]:   3 + c + (3 + c)^2

#I[6]::  S[%,c->4]

#O[6]:   56
```

At the beginning of this section we described the procedure for discarding ambiguous SMP input lines. Often simple changes in such lines give them a definite meaning. When the editor is entered, the cursor is placed under the position in the line where modification is required. Any characters typed at this stage are used to replace the characters appearing above them in the original input line. Spaces may be used to position the cursor. # deletes the character appearing above it. ^⊂*text*⊃⊂*space*⊃ inserts the sequence of characters ⊂*text*⊃ in the line immediately before the character above the ^. The text to be inserted is terminated by a space: insert * to denote multiplication. The editing commands are performed when a ⊂*newline*⊃ is entered, and the edited line is then printed. If no editing commands are given before a ⊂*newline*⊃ the editing is assumed complete, and the edited line is used as input for SMP.

```
#I[1]::  2++5
+ unexpected
         2++5
<edit>     #
         2+5
<edit>

#O[1]:   7

#I[2]::  t:f]x]
] unexpected
         t:f]x]
<edit>     [
         t:f[x]
<edit>

#O[2]:   f[x]

#I[3]::  (c+d)(a++b+c
+ unexpected
         (c+d)(a++b+c
<edit>            ^x  +1)
         (c+d)(a+x+b+c+1)
<edit>

#O[3]:   (c + d) (1 + a + b + c + x)
```

Notice that <edit> is the prompt for input of editor commands.

Further examples

Example 1
Find the factors of $1-x^{12}$.

```
#I[1]:  Fac[1-x^12]

                                    2          2          2          2    4
#O[1]:  -(-1 + x) (1 + x) (1 + x ) (1 - x + x ) (1 + x + x ) (1 - x  + x )

#I[2]:  Ex[%]

                 12
#O[2]:  1 - x
```

Example 2
Find the numerical value of $\Gamma(5.2)$.

```
#I[1]:  N[Gamma[5.2]]

#O[1]:  32.5781
```

Example 3
Verify that $x=1$ is a root of the polynomial $x^3+9x^2+11x-21$.

```
#I[1]:  x^3+9x^2+11x-21

                                2    3
#O[1]:  -21 + 11x + 9 x  + x

#I[2]:  S[%,x->1]

#O[2]:  0
```

Example 4
Find the value of the first derivative of $e^{x^a/(1+x)}$ at the point $x \to 1$.

```
#I[1]:  Exp[x^a/(1+x)]

                a
               x
#O[1]:  Exp[-------]
            1 + x

#I[2]:  D[%,x]

                a          a
               x        - x          -1 + a
            Exp[-------] (------- + a x       )
                1 + x    1 + x
#O[2]:  ---------------------------------------
                         1 + x

#I[3]:  S[%,x->1]

            Exp[1/2] (-1/2 + a)
#O[3]:  -------------------
                    2

#I[4]:  N[%]

#O[4]:  -.4121883 + .8243606a
```

Example 5

Verify that the roots of a quadratic equation are given by the usual formulae.

```
#I[1]::  r1:(-b+Sqrt[b^2-4a c])/(2a)

                        2  1/2
         -b + (-4a c + b )
#O[1]:   ----------------------
                  2a

#I[2]::  r2:(-b-Sqrt[b^2-4a c])/(2a)

                        2  1/2
         -b - (-4a c + b )
#O[2]:   ----------------------
                  2a

#I[3]::  a(x-r1)(x-r2)

                                  2  1/2                    2  1/2
                       -b - (-4a c + b )         -b + (-4a c + b )
#O[3]:   a (x - ----------------------) (x - ----------------------)
                            2a                        2a

#I[4]::  Ex[%]

                    2
#O[4]:   c + a x  + b x
```

2. Lists, projections and parts

SMP manipulates symbolic expressions. Symbols and numbers form the fundamental units. They are combined into more complex expressions through projections and lists. Symbols were discussed in the previous section. In this section, we introduce lists and projections. A thorough understanding of these constructs is crucial for all but the most superficial use of SMP.

A list is an ordered (and indexed) collection of expressions. In its simplest form, a list consists of a set of expressions separated by commas and enclosed in brace brackets. An example is {a+b,c+d,3,1}. Such lists are called "contiguous".

It is often necessary to perform the same operation on several expressions. This may be achieved conveniently by collecting the expressions into a list, and then performing the operation on the complete list. This yields a list containing the results of performing the operation on each element of the original list.

```
#I[1]:  t:{a+b,c+d,3,1}

#O[1]:  {a + b,c + d,3,1}

#I[2]:  5t

#O[2]:  {5(a + b),5(c + d),15,5}

#I[3]:  Exp[t]

#O[3]:  {Exp[a + b],Exp[c + d],Exp[3],E}

#I[4]:  x+t^2

                    2          2
#O[4]:  {x + (a + b) ,x + (c + d) ,9 + x,1 + x}

#I[5]:  t+%

                        2          2
#O[5]:  {a + b + x + (a + b) ,c + d + x + (c + d) ,12 + x,2 + x}
```

Notice that in line 5 two lists of the same length were added; the result was a list of the sums of their corresponding elements.

Sets of expressions in several lists may be combined by concatenating the lists using `Cat`.

```
#I[1]:  t:{a+b,x+1,c}

#O[1]:  {a + b,1 + x,c}

#I[2]:  Cat[t,t,{2,3}]

#O[2]:  {a + b,1 + x,c,a + b,1 + x,c,2,3}

#I[3]:  Rev[%]

#O[3]:  {3,2,c,1 + x,a + b,c,1 + x,a + b}
```

`Rev[`*list*`]` reverses the order of elements in *list*.

A second important use of contiguous lists is to represent vectors. The dot product of two vectors *list1* and *list2* is given by *list1*. *list2* [8.3].

```
#I[1]:  {a,b,c}.{x,y,z}

#O[1]:  a x + b y + c z
```

The entries in a list may themselves be lists. A list of equal length lists may be considered as a matrix, with each list corresponding to a row of the matrix. Higher-rank tensors are represented as lists of matrices, lists of lists of matrices, and so on. Multiplication of two matrices or of a matrix by a vector is then obtained simply as a dot product of the corresponding lists [8.3].

```
#I[1]:: m:{{a,b},{c,d}}
#O[1]:  {{a,b},{c,d}}
#I[2]:: v:{p,q}
#O[2]:  {p,q}
#I[3]:: m.v
#O[3]:  {a p + b q,c p + d q}
#I[4]:: v.m
#O[4]:  {a p + c q,b p + d q}
#I[5]:: v.m.v
#O[5]:  p (a p + c q) + q (b p + d q)
#I[6]:: n:{{w,x},{y,z}}
#O[6]:  {{w,x},{y,z}}
#I[7]:: 3n
#O[7]:  {{3w,3x},{3y,3z}}
#I[8]:: n^2+2*m
                    2      2      2      2
#O[8]:  {{2a + w ,2b + x },{2c + y ,2d + z }}
#I[9]:: m.n
#O[9]:  {{a w + b y,a x + b z},{c w + d y,c x + d z}}
#I[10]:: m.n.m
#O[10]:  {{a (a w + b y) + c (a x + b z),b (a w + b y) + d (a x + b z)},
          {a (c w + d y) + c (c x + d z),b (c w + d y) + d (c x + d z)}}
```

Note that in line 8, `n^2` gave the square of each element of the matrix `n`, not `n.n`.

Some mathematical matrix operations [9.7] are: `Det` (determinant), `Minv` (matrix inverse) and `**` (outer product).

```
#I[1]:: m:{{2,3},{-5,6}}
#O[1]:  {{2,3},{-5,6}}
#I[2]:: Det[m]
#O[2]:  27
#I[3]:: Minv[m]
#O[3]:  {{2/9,-1/9},{5/27,2/27}}
#I[4]:: n:{{w,x},{y,z}}
#O[4]:  {{w,x},{y,z}}
```

```
#I[5]::  mesn

#O[5]:   {{{{2w,2x},{2y,2z}},{{3w,3x},{3y,3z}}},
                {{{-5w,-5x},{-5y,-5z}},{{6w,6x},{6y,6z}}}}
```

Entries in a contiguous list are indexed by their positions. With `v:{a,b,c,d}`, the "projection" `v[3]` extracts the third component of `v`, and thus yields `c`.

```
#I[1]::  v:{a,x^2,3a/(b+c),1}

                  2  3a
#O[1]:   {a,x ,-----,1}
                   b + c

#I[2]::  v[2]

          2
#O[2]:   x

#I[3]::  v[2]+v[4-1]

           3a       2
#O[3]:   ----- + x
         b + c
```

A sequence of projections may be used to extract entries in lists of lists.

```
#I[1]::  w:{{a,b},{c,d}}

#O[1]:   {{a,b},{c,d}}

#I[2]::  w[2]

#O[2]:   {c,d}

#I[3]::  w[2][2]

#O[3]:   d

#I[4]::  w[1,1]

#O[4]:   a
```

Notice that `w[1][1]` is equivalent to `w[1,1]` (see, however, sect. 8).

Projections which specify an entry not present in a list are left unevaluated.

```
#I[1]::  v:{a,b,c}

#O[1]:   {a,b,c}

#I[2]::  v[7]

#O[2]:   v[7]

#I[3]::  v[3]+v[-2]

#O[3]:   c + v[-2]
```

Entries in a list may be specified by assignments. Existing entries may be changed by such assignments, or additional entries may be introduced.

```
#I[1]::  v:{a,b,c}

#O[1]:   {a,b,c}

#I[2]::  v[2]:p+q
```

```
#O[2]:   p + q

#I[3]::  v

#O[3]:   {a,p + q,c}

#I[4]::  v[4]:d

#O[4]:   d

#I[5]::  v

#O[5]:   {a,p + q,c,d}

#I[6]::  v[6]:p^2+q^2

          2    2
#O[6]:   p  + q

#I[7]::  v

                2    2
#O[7]:   {[6]: p  + q , [1]: a, [2]: p + q, [3]: c, [4]: d}
```

On line 6, a value is defined for `v[6]`, even though no value has been assigned to `v[5]`. The resulting list is no longer contiguous: the indices of its entries are not given by their positions in the list. Instead, the index associated with each entry is displayed explicitly in square brackets. The resulting form for `v` is given on line 7.

The components of a vector may be defined in any order. When values have been specified for a suitable set of components, the list representing the vector becomes contiguous.

```
#I[1]::  u[1]:a

#O[1]:   a

#I[2]::  u

#O[2]:   {a}

#I[3]::  u[3]:c

#O[3]:   c

#I[4]::  u

#O[4]:   {[3]: c, [1]: a}

#I[5]::  u[2]:b

#O[5]:   b

#I[6]::  u

#O[6]:   {a,b,c}
```

The index of an entry in a list need not be a number: it may be any expression.

```
#I[1]::  f[x]:a

#O[1]:   a

#I[2]::  f

#O[2]:   {[x]: a}

#I[3]::  f[1+y]:b
```

```
#O[3]:   b

#I[4]::  f

#O[4]:   {[1 + y]: b, [x]: a}

#I[5]::  f[x]^2+f[z]

                2
#O[5]:   a   + f[z]

#I[6]::  f[z]:3

#O[6]:   3

#I[7]::  f

#O[7]:   {[z]: 3, [1 + y]: b, [x]: a}

#I[8]::  f[x]^2+f[z]

                  2
#O[8]:   3 + a
```

In line 1, the value a is assigned to the projection f[x] of f with "filter" x. The value of the symbol f given on line 2 is then a list with one entry. The index of this entry is x and its value is a. The projection f[x] specifies that "part" of f indexed by the filter x, and hence extracts the value a. The assignment for f[1+y] on line 3 adds an entry with index 1+y to the list giving the value of f. The list for f given on line 4 thus specifies the values of "parts" of f indexed with filters x or 1+y. Parts of f indexed by other filters have not been specified. Thus on line 5, the projection f[z] is left in a symbolic unevaluated form, since no value for f[z] has been assigned. On line 6, an assignment for f[z] is made, so that on line 8, the projection f[z] may be evaluated.

Lists have analogues in other computer languages. Contiguous lists may be used as "arrays". Lists of contiguous lists are analogous to multi-dimensional arrays. Lists such as line 7 above resemble "records" or "structures".

Just as for symbols, assignment of a new value to a projection replaces any previous value.

```
#I[1]::  f[x]:a

#O[1]:   a

#I[2]::  f[y]:b

#O[2]:   b

#I[3]::  f

#O[3]:   {[y]: b, [x]: a}

#I[4]::  f[x]:c^2

                2
#O[4]:   c

#I[5]::  f

                                2
#O[5]:   {[y]: b, [x]: c    }

#I[6]::  f[x]+f[y]

                  2
#O[6]:   b + c
```

Values assigned to projections such as `f[x]` may be removed by `f[x]:` `f:` removes all values assigned to projections of `f`.

```
#I[1]::  f[x]:a

#O[1]:   a

#I[2]::  f[y]:b

#O[2]:   b

#I[3]::  f

#O[3]:   {[y]: b, [x]: a}

#I[4]::  f[y]:

#I[5]::  f

#O[5]:   {[x]: a}

#I[6]::  f[y]+f[x]

#O[6]:   a + f[y]

#I[7]::  f:

#I[8]::  f

#O[8]:   f

#I[9]::  f[y]+f[x]

#O[9]:   f[x] + f[y]
```

Projections may be used to represent "functions". `f[2]` may be considered as the "function" `f` with "argument" 2. Assignments for projections then define values for "functions" at particular "points".

"System-defined" functions, such as `Log[2]` are also projections. When no system-defined value exists for a projection, a value may be assigned to it.

```
#I[1]::  Log[x]

#O[1]:   Log[x]

#I[2]::  Log[x]:a

#O[2]:   a

#I[3]::  Log

#O[3]:   {[x]: a}

#I[4]::  Log[x]+Log[y]+Log[0]

#O[4]:   a + Log[0] + Log[y]

#I[5]::  Log[0]:Inf

#O[5]:   Inf

#I[6]::  Log[0]^2+Log[x^2]+Log[x]^2

                 2        2    2
#O[6]:   Log[x ] + Inf  + a
#I[7]::  Log[1]

#O[7]:   0
```

In line 4, `Log[0]` was entered. This has no system-defined value; a value was assigned to it on line 5.

Notice that entry of `Log[0]` did not result in the printing of any warning message. Like other symbolic expressions for which no value has been defined, `Log[0]` was simply left unevaluated. Unless `On[]` has been used (sect. 1), SMP does not usually print any warning messages.

In line 7, `Log[1]` was automatically simplified to `0`. Such automatic evaluations of mathematical functions are performed only when the results are very simple. Automatic simplifications are performed before explicitly assigned values are used: `Log[1]` may thus not be assigned another value.

Commoner system-defined projections may be input in special forms. A complete table of such forms is given in [2.10]. For example, the factorial function `Fctl[x]` may be input as $x!$. In all cases, $x!$ is equivalent to `Fctl[x]`.

```
#I[1]::  5!

#O[1]:   120

#I[2]::  Fctl[5]

#O[2]:   120

#I[3]::  x!:a

#O[3]:   a

#I[4]::  Fctl

#O[4]:   {[x]: a}

#I[5]::  x!^2

          2
#O[5]:   a

#I[6]::  (1/2)!:Sqrt[Pi]

            1/2
#O[6]:   Pi

#I[7]::  Fctl

                    1/2
#O[7]:   {[1/2]: Pi    , [x]: a}
```

We discussed above the extraction of an element in a matrix by two successive projections. Elements in a matrix may be introduced by assignments for projections with two filters.

```
#I[1]::  m[2,2]:d

#O[1]:   d

#I[2]::  m

#O[2]:   {[2]: {[2]: d}}

#I[3]::  m[1,1]:a

#O[3]:   a

#I[4]::  m

#O[4]:   {{a},{[2]:d}}
```

```
#I[5]::   m[2][1]:c
#O[5]:    c
#I[6]::   m
#O[6]:    {{a},{c,d}}
#I[7]::   m[1,2]:b
#O[7]:    b
#I[8]::   m
#O[8]:    {{a,b},{c,d}}
```

Projections with several symbolic filters may also be assigned values.

```
#I[1]::   a^2:3
#O[1]:    3
#I[2]::   Pow
#O[2]:    {[a]: {[2]: 3}}
#I[3]::   a^2+a^3
                 3
#O[3]:    3 + a
#I[4]::   8^8
                 8
#O[4]:    8
#I[5]::   8^8:1
#O[5]:    1
#I[6]::   Pow
#O[6]:    {[8]: {[8]: 1}, [a]: {[2]: 3}}
#I[7]::   (1-1)^8
#O[7]:    1
```

In the assignment and evaluation of, for example, Plus projections, the commutative and associative properties of the Plus operation were used. (The specification of such properties will be discussed in sect. 8.)

```
#I[1]::   a+b:e^2
                 2
#O[1]:    e
#I[2]::   Plus
                          2
#O[2]:    {[a]: {[b]: e }}
#I[3]::   a+b+c+d
                     2
#O[3]:    c + d + e
#I[4]::   a+c:f^2
```

```
#O[4]:   f
                2

#I[5]::  Plus

#O[5]:   {[a]: {[c]: f , [b]: e }}
                      2        2

#I[6]::  a+b+c+d

#O[6]:   b + d + f
                 2
```

Not only mathematical functions but all SMP operations are specified by projections. Thus, for example, `a:b` is equivalent to the explicit projection `Set[a,b]`.

Just as parts of a list may be obtained by projections, so also parts of an arbitrary expression may be specified by suitable projections. In a function such as `f[a,b,c]` the filter `a` is specified by index 1, `b` by 2 and `c` by 3. The "projector" `f` is specified by index 0.

```
#I[1]::  t:f[a,b,c]

#O[1]:   f[a,b,c]

#I[2]::  t[1]

#O[2]:   a

#I[3]::  t[3]

#O[3]:   c

#I[4]::  t[0]

#O[4]:   ' f

#I[5]::  t[5]

#O[5]:   t[5]

#I[6]::  t[3]:d

#O[6]:   d

#I[7]::  t

#O[7]:   f[a,b,d]
```

In line 6, the part `t[3]` was re-assigned to have value `d`. The resulting complete projection `f[a,b,d]` was given on line 7.

The quote (') on line 4 will be discussed in sect. 9.

Parts in functions input in special form are indexed according to their positions in the corresponding explicit projections.

```
#I[1]::  t:a^b

#O[1]:   a
              b

#I[2]::  t[1]

#O[2]:   a

#I[3]::  t[0]

#O[3]:   ' Pow
```

```
#I[4]::  u:a+b+c+d

#O[4]:   a + b + c + d

#I[5]::  u[3]

#O[5]:   c
```

Some system-defined projections are output in forms for which the labelling of parts is not manifest. Such cases are indicated by a * at the beginning of the output. Lpr [*expr*] prints *expr* in a simple linear form in which the labelling of parts is manifest.

```
#I[1]::  a+b I+c

#O[1]:*  (a + c) + b I

#I[2]::  Lpr[%]

Cx[a + c,b]
```

Note that I is the complex unit (square root of -1).

Elements in lists of lists may be extracted by several successive projections (or equivalently a single projection with several filters). Similarly, parts in compound expressions may be extracted by suitable projections with several filters.

```
#I[1]::  t:(a+b)^2

                  2
#O[1]:   (a + b)

#I[2]::  t[1]

#O[2]:   a + b

#I[3]::  t[1][2]

#O[3]:   b

#I[4]::  t[1,1]

#O[4]:   a

#I[5]::  t[2]

#O[5]:   2

#I[6]::  t[2,1]

#O[6]:   t[2,1]
```

Notice that since no part t[2,1] is present, the projection on line 6 is left unchanged.

Symbols and projections may carry numerical coefficients without explicit Mult projections. Numerical coefficients are thus irrelevant in defining positions of parts, even when they are rational numbers.

```
#I[1]::  t:2/3(a+2b)

         2(a + 2b)
#O[1]:   ---------
             3

#I[2]::  t[1]

#O[2]:   a
```

```
#I[3]::  t[2]

#O[3]:   2b

#I[4]::  t[-1]

#O[4]:   2/3
```

In line 4, projection with -1 was used to extract the numerical coefficient.

The system-defined projection Pos[*form*, *expr*] gives a list of "positions" at which the subexpression *form* appears in the expression *expr*. Each position is specified by a list of the filters necessary to extract it by a projection.

```
#I[1]::  t:(a+b)^a+c^b

                  b          a
#O[1]:   c  + (a + b)

#I[2]::  Pos[a,t]

#O[2]:   {{2,1,1},{2,2}}

#I[3]::  t[2,1,1]

#O[3]:   a

#I[4]::  Proj[t,{2,2}]

#O[4]:   a

#I[5]::  Pos[c^b,t]

#O[5]:   {{1}}
```

Notice that in line 1 parts were rearranged using the commutativity of Plus. The positions of parts are always determined by their location in the simplified form of an expression.

The projection Proj[*expr*, {*x1*, *x2*, ...}] used on line 4 is equivalent to *expr*[*x1*, *x2*, ...].

The specification of parts is important in performing operations on expressions. In many system-defined projections, a "domain" may be used to define a set of parts in an expression on which an operation is to be performed. Domains are described in [2.5].

In rearranging or simplifying expressions, it is often convenient to modify a particular part, while leaving other parts unchanged.

```
#I[1]::  t:x^2+3+(x^2-1)^2

                2          2 2
#O[1]:   3 + x  + (-1 + x )

#I[2]::  t[3]:S[t[3],x->a]

                 2 2
#O[2]:   (-1 + a )

#I[3]::  t

                2          2 2
#O[3]:   3 + x  + (-1 + a )
```

Further examples

Example 1

Find the angle between the vectors {1,-2,a} and {2a,-3,5} for the cases a=0 and a=1/2.

```
#I[1]:  v1:{1,-2,a}

#O[1]:  {1,-2,a}

#I[2]:  v2:{2a,-3,5}

#O[2]:  {2a,-3,5}

#I[3]:  ang:Acos[v1.v2/Sqrt[v1.v1 v2.v2]]

                                6 + 7a
#O[3]:  Acos[--------------------------]
                  2 1/2             2 1/2
        (5 + a )     (34 + 4 a )

#I[4]:  S[ang,a->0]

                6
#O[4]:  Acos[----------]
              1/2   1/2
        5     34

#I[5]:  N[%]

#O[5]:  1.8926

#I[6]:  S[ang,a->1/2]

                  19
#O[6]:  Acos[---------------]
                1/2   1/2
        2 21/4      35

#I[7]:  N[%]

#O[7]:  .7942422
```

Example 2

Find the characteristic polynomial for the matrix

$$\begin{bmatrix} 1 & 2 & -1 \\ 4 & 3 & -1 \\ 4 & 2 & 1 \end{bmatrix}$$

```
#I[1]:  {{1,2,-1},{4,3,-1},{4,2,1}}

#O[1]:  {{1,2,-1},{4,3,-1},{4,2,1}}

#I[2]:  %-lam{{1,0,0},{0,1,0},{0,0,1}}

#O[2]:  {{1 - lam,2,-1},{4,3 - lam,-1},{4,2,1 - lam}}

#I[3]:  Det[%]

#O[3]:  -12 + 4 lam + (1 - lam) (2 + (1 - lam) (3 - lam))

#I[4]:  Ex[%]

                                2      3
#O[4]:  -7 - 5 lam + 5 lam  - lam
```

Example 3

Find the real part of $e^{\frac{2}{5}\pi i}$.

```
#I[1]:  Exp[2I Pi/5]

#O[1]:* Exp[2Pi/5 I]

#I[2]:   N[%]

#O[2]:* .309017 + .95105651

#I[3]:  Re[%]

#O[3]:  .309017
```

Example 4

Form the 3×3 matrix M such that M_{ij} is ij.

```
#I[1]:  v:{1,2,3}

#O[1]:  {1,2,3}

#I[2]:  v**v

#O[2]:  {{1,2,3},{2,4,6},{3,6,9}}
```

Example 5

Change the circled $\sin(x)$ in the expression t to $\sqrt{1-\cos^2(x)}$.

```
#I[1]:  t:4Sin[x] +3z^3(1+z+Sin[x])Sin[x]^2/(4(1+Sin[x]))

                    3                          2
               3 z  (1 + z + Sin[x]) Sin[x]
#O[1]:  ------------------------------------ + 4Sin[x]
                  4(1 + Sin[x])

#I[2]:  Pos[Sin,t]

#O[2]:  {{1,1,2,3,0},{1,1,3,1,0},{1,2,2,0},{2,0}}

#I[3]:  t[1,2,2]:Sqrt[1-Cos[x]^2]

                        2 1/2
#O[3]:  (1 - Cos[x] )

#I[4]:  t

                    3                          2
               3 z  (1 + z + Sin[x]) Sin[x]
#O[4]:  ------------------------------------ + 4Sin[x]
                                2 1/2
           4(1 + (1 - Cos[x] )    )
```

Example 6

Consider the network defined by the adjacency matrix

$$\begin{bmatrix} 0 & 2 & 0 \\ 2 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Find the number of 3-step paths from node 1 to node 2.

```
#I[1]::   m:{{0,2,0},{2,0,1},{0,1,1}}
#O[1]:    {{0,2,0},{2,0,1},{0,1,1}}
#I[2]::   m.m.m
#O[2]:    {{0,10,2},{10,1,6},{2,6,3}}
#I[3]::   %[1,2]
#O[3]:    10
```

3. Patterns

The assignment `f[x]:x^2` defines a projection of `f` with the specific filter `x` to be `x^2`. This assignment does not define any value for `f[y]`. To define a mathematical function `g` which forms the square of any expression which appears as its "argument", one performs the assignment `g[$x]:$x^2`. Here the "generic symbol" `$x` stands for any expression. Any symbol which begins with `$` is taken as a generic symbol, and may stand for any expression.

```
#I[1]::  f[x]:x^2

                2
#O[1]:   x

#I[2]::  f[y]+f[x]

                2
#O[2]:   x  + f[y]

#I[3]::  g[$x]:$x^2

                2
#O[3]:   $x

#I[4]::  g

                          2
#O[4]:   {[$x]: $x }

#I[5]::  g[2]+g[x]+g[a^2]

                    4     2
#O[5]:   4 + a  + x

#I[6]::  g[0]:c

#O[6]:   c

#I[7]::  g

                                  2
#O[7]:   {[0]: c, [$x]: $x }

#I[8]::  g[0]+g[2]

#O[8]:   4 + c
```

The assignment for `g[0]` in line 6 defined a special value for the function `g`. This is used in preference to the more general assignment for `g[$x]` given in line 3.

When values for projections are assigned, those for more specific cases are placed before those for more general cases. The most restricted applicable assignment is therefore used in the evaluation of a particular projection.

```
#I[1]::  f[0]:a

#O[1]:   a

#I[2]::  f

#O[2]:   {[0]: a}

#I[3]::  f[$x]:1/$x

                1
#O[3]:   --
                $x

#I[4]::  f
```

```
#O[4]:   {[0]: a, [$x]: 1/$x}

#I[5]::  f[1]:b

#O[5]:   b

#I[6]::  f

#O[6]:   {[1]: b, [0]: a, [$x]: 1/$x}

#I[7]::  f[z]+f[0]^2

#O[7]:   1/z + a^2

#I[8]::  g[$n]:$n g[$n-1]

#O[8]:   $n g[-1 + $n]

#I[9]::  g

#O[9]:   {[$n]: $n g[-1 + $n]}

#I[10]::  g[5]

#O[10]:   0

#I[11]::  g[1]:1

#O[11]:   1

#I[12]::  g

#O[12]:   {[1]: 1, [$n]: $n g[-1 + $n]}

#I[13]::  g[5]

#O[13]:   120
```

In line 8, a "recursive" definition for the factorial function g was given. With no end condition specified, the result for g[5] in line 10 is 0. With the specification on line 11, evaluation of g[5] uses the recursion relation on line 8 until the end point of line 11 is reached.

Assignments may be made not only for projections such as f[$x] but also for more complicated "patterns" such as f[$x^2,$y].

```
#I[1]::  Exp[1+Log[$x]]:E $x

#O[1]:   $x E

#I[2]::  Exp

#O[2]:   {[1 + Log[$x]]: $x E}

#I[3]::  Exp[1+Log[a+b]]

#O[3]:   E (a + b)

#I[4]::  Exp[2+Log[a+b]]

#O[4]:   Exp[2 + Log[a + b]]

#I[5]::  Exp[1+$x]:h[$x]

#O[5]:   h[$x]
```

```
#I[6]::  Exp

#O[6]:   {[1 + Log[$x]]: $x E, [1 + $x]: h[$x]}

#I[7]::  Exp[1+a^2]+Exp[1+Log[b]]

                    2
#O[7]:   E b + h[a ]
```

All occurrences of a particular generic symbol in a pattern must stand for the same expression.

```
#I[1]::  $x^$x:q[$x]

#O[1]:   q[$x]

#I[2]::  Pow

#O[2]:   {[$x]: {[$x]: q[$x]}}

#I[3]::  x^2+y^y

          2
#O[3]:   x  + q[y]

#I[4]::  $x^($x+$y):p[$x,$y]

#O[4]:   p[$x,$y]

#I[5]::  x^(x+a)

#O[5]:   p[x,a]

#I[6]::  Log[a+b]^(Log[a+b]+c^2)

                          2
#O[6]:   p[Log[a + b],c ]
```

Patterns match expressions whenever replacements for generic symbols may be deduced without solving explicit equations. For example, while `x^$n` matches `x^2`, `x^$n` does not match `x` alone, since in the latter case, determination of the necessary replacement for `$n` would require solution of the equation `x^$n=x`. Nevertheless, `f[$n,x^$n]` does match `f[1,x]` since the replacement for `$n` may be deduced from a direct comparison between the first filters, without solving an equation. Similarly, `f[1-$x,$x]` matches `f[-2,3]`, while `f[1-$x,1+$x]` matches `f[1-2a,1+2a]`, but does not match `f[-2,4]`.

```
#I[1]::  f[2/$x,$x]:h[$x]

#O[1]:   h[$x]

#I[2]::  f[1,2]

#O[2]:   h[2]

#I[3]::  f[2,1]

#O[3]:   h[1]

#I[4]::  f[2a,1/a]

           1
#O[4]:   h[-]
           a

#I[5]::  g[2/$x,1/$x]:i[$x]

#O[5]:   i[$x]
```

```
#I[6]::  g[2/x,1/x]

#O[6]:   i[x]

#I[7]::  g[2,1]

#O[7]:   g[2,1]

#I[8]::  gp[2$x_,$x_]:i[1/$x_]

                1
#O[8]:   i[-----]
           $x_

#I[9]::  gp[2,1]

#O[9]:   i[1]

#I[10]::  j[2$x]:jp[$x]

#O[10]:   jp[$x]

#I[11]::  j[x]

#O[11]:   jp[x/2]

#I[12]::  j[1]

#O[12]:   jp[1/2]
```

Notice that in line 10, 2$x is treated as a single generic symbol matching any expression, regardless of its numerical coefficient.

Pattern matching takes account of commutativity and associativity properties of functions.

```
#I[1]::  b+$x:u[$x]

#O[1]:   u[$x]

#I[2]::  a+b

#O[2]:   u[a]

#I[3]::  a+b+c

#O[3]:   c + u[a]

#I[4]::  a $x^2:v[$x]

#O[4]:   v[$x]

#I[5]::  a b c^2 d^2

                2
#O[5]:   b d   v[c]

#I[6]::  f[$x+$y]:g[$x]g[$y]

#O[6]:   g[$x] g[$y]

#I[7]::  f[x+y]+f[x+y+z]

#O[7]:   g[x] g[y] + f[x + y + z]
```

In line 7, $x+$y matches x+y but not x+y+z. Multi-generic symbols such as $$x are used to represent a sequence of arguments to a function. $$x+$y would match x+y+z with $y replaced by x and $$x replaced by y+z.

```
#I[1]::  f[$$x+$y]:g[$$x]g[$y]
#O[1]:   g[$$x] g[$y]
#I[2]::  f[x+y+z]
#O[2]:   g[x] g[y + z]
#I[3]::  f[x]+f[x+y]+f[x+y+z+w]
#O[3]:   g[w] g[x + y + z] + g[x] g[y] + f[x]
#I[4]::  mu.$$x.mu:h[$$x]
#O[4]:   h[$$x]
#I[5]::  p.mu.q.r.mu.p
#O[5]:   p.h[q.r].p
#I[6]::  a.mu.a1.b1.mu.c1.mu.d1.e1.mu.b.c
#O[6]:   a.h[a1.b1].c1.h[d1.e1].b.c
#I[7]::  a.mu.a1.b1.mu.c1.mu.b
#O[7]:   a.h[a1.b1].c1.mu.b
#I[8]::  Log[a b c d]
#O[8]:   Log[a b c d]
#I[9]::  Log[$x $$x]:Log[$x]+Log[$$x]
#O[9]:   Log[$$x] + Log[$x]
#I[10]:: Log[a b c d]
#O[10]:  Log[a] + Log[b] + Log[c] + Log[d]
```

The class of expressions matched by a particular pattern may be restricted by imposing conditions on the generic symbols in it. Whereas `$x` stands for any expression, `$x_;$x>0` stands only for expressions such that `$x>0` is determined to be true.

```
#I[1]::  g[$n_;$n>0]:h[$n]
#O[1]:   h[$n]
#I[2]::  g[2]+g[-3]+g[x]
#O[2]:   g[-3] + g[x] + h[2]
#I[3]::  gamma[$x_;Natp[$x]]:($x-1)!
#O[3]:   (-1 + $x)!
#I[4]::  gamma[4]+gamma[-2]
#O[4]:   6 + gamma[-2]
```

In line 3, `Natp[x]` is a projection which yields `1` (representing "true") if x is determined to be a natural number (positive integer), and `0` (representing "false") otherwise. Similarly, `Intp` tests for an integer, `Numbp` for a number and `Listp` for a list.

$x=y$ yields `1` if x and y are equal, and `0` if they are determined to be unequal. If no definite truth value is obtained, the expression $x=y$ is left unchanged. Other relational operators are `~=` (unequal), `>` (greater than), `<` (less than), `>=` and `<=`.

Finally, $\sim x$ means "not x", $x \& y$ means "x and y" and $x \mid y$ means "x or y".

```
#I[1]::  3=2
#0[1]:   0
#I[2]::  x=2
#0[2]:   x = 2
#I[3]::  3>2>=1 & x~=y
#0[3]:   x ~= y
#I[4]::  f[$x,$y_:$x>$y & 0<$y<1]:g[$x,$y]
#0[4]:   g[$x,$y]
#I[5]::  f[2,1/2]
#0[5]:   g[2,1/2]
#I[6]::  f[x,1/2]
#0[6]:   f[x,1/2]
```

Without giving a definite numerical value for a symbol `x`, one may assert that `x` is positive by the assignment `x>0:1`. Similarly, `x` may be defined as an integer by `Intp[x]:1`.

Further examples

Example 1

Find the set of all subparts of an expression.

```
#I[1]:=  t:6(1+4x)(2+3x+x z)

#O[1]:   6(1 + 4x) (2 + 3x + x z)

#I[2]:=  Pos[$x,t]

#O[2]:   {{1,{1,1}},{4x,{1,2}},{1 + 4x,{1}},{2,{2,1}},{3x,{2,2}},{x,{2,3,1}},
           {z,{2,3,2}},{x z,{2,3}},{2 + 3x + x z,{2}},
           {(1 + 4x) (2 + 3x + x z),{0}}}
```

Example 2

Define rules to simplify $e^{i\pi\frac{n}{2}}$ where n is an integer.

```
#I[1]:=  Exp[I Pi $x_]:=Intp[2$x]:I^Mod[2$x,4]

                Mod[2$x,4]
#O[1]:=  I

#I[2]:=  Exp[3I Pi]

#O[2]:   -1

#I[3]:=  Exp[3/2 I Pi]

#O[3]:=  -I

#I[4]:=  Exp[I Pi]

#O[4]:=  Exp[Pi I]
```

Example 3

Find the 5^{th} and 10^{th} terms in the series defined by

$$f(x) = f(x-1) + a\, f(x-2)$$
$$f(1) = f(2) = 1$$

```
#I[1]:=  f[$x]:f[$x-1]+a f[$x-2]

#O[1]:   a f[-2 + $x] + f[-1 + $x]

#I[2]:=  f[1]:f[2]:1

#O[2]:   1

#I[3]:=  f[5]

#O[3]:   1 + 2a + a (1 + a)

#I[4]:=  f[10]

#O[4]:   1 + 2a + a (1 + a) + a (1 + 2a) + a (1 + 2a + a (1 + a))
           + a (1 + 2a + a (1 + a) + a (1 + 2a))
           + a (1 + 2a + a (1 + a) + a (1 + 2a) + a (1 + 2a + a (1 + a)))
           + a (1 + 2a + a (1 + a) + a (1 + 2a) + a (1 + 2a + a (1 + a))
```

```
                    + a (1 + 2a + a (1 + a) + a (1 + 2a)))

#I[5]:: Ex[%]

                                  2        3        4
#O[5]:  1 + 8a + 21 a  + 20 a  + 5 a
```

Example 4

Define a function of three variables which vanishes if the difference between any arguments is greater than 1 and is equal to one otherwise.

```
#I[1]:: f[$1,$2,$3]:=~(Abs[$1-$2]>1 | Abs[$2-$3]>1 | Abs[$3-$1]>1)

#O[1]:  ~ (Abs[-$1 + $3] > 1 | Abs[$1 - $2] > 1 | Abs[$2 - $3] > 1)

#I[2]:: f[0,0,0]

#O[2]:  1

#I[3]:: f[2,0,0]

#O[3]:  0

#I[4]:: f[1,2,3]

#O[4]:  0

#I[5]:: f[0,.5,.7]

#O[5]:  1
```

4. Building up calculations

One important feature of SMP is the possibility of constructing a system of rules and definitions, and then using this system repeatedly for many different cases. SMP remembers and organizes each definition or rule given.

Everything typed during an SMP job is kept in a "record file" which may be re-read or printed either during the job or after the job has finished. Details of the "record file" are given in the Implementation Notes; in most implementations, it will be named `smp.out`. At most installations, each SMP job run by a particular user generates the same record file: if the record is to be saved permanently, it must be moved to another file before the next SMP job is started. The record file may also be printed on paper to provide a permanent record. The necessary procedures are given in the implementation notes. Note that only input and output SMP expressions are included in the record file: operations such as editing are omitted.

The input and output lines in an SMP job may be manipulated just like other expressions. The ith output expression is given by `@`i. `@{`$i,j,\ldots$`}` gives a list of output expressions $i,j,\ldots$. Each output line is assigned as the value of a projection of `#0`, as indicated by the `#0[`i`]:` at the beginning of each output line; `@`i is a short notation for `#0[`i`]`. `#0` is a list of all output expressions.

```
#I[1]::  a:x^2+1

                  2
#0[1]:   1 + x

#I[2]::  a+a^2

                  2          2 2
#0[2]:   1 + x  + (1 + x )

#I[3]::  @1

                  2
#0[3]:   1 + x

#I[4]::  @{2,1}

                  2          2 2       2
#0[4]:   {1 + x  + (1 + x ) ,1 + x }

#I[5]::  #0

                        2          2 2       2              2
#0[5]:   {[4]: {1 + x  + (1 + x ) ,1 + x }, [3]: 1 + x ,

                       2          2 2            2
                 [2]: 1 + x  + (1 + x ) , [1]: 1 + x ,

                 [0]: {"/u1/smp/smp.init"}}

#I[6]::   {1,5..9}

#0[6]:   {1,5,6,7,8,9}

#I[7]::  @{2..4,6}

                  2          2 2       2          2          2 2       2
#0[7]:   {1 + x  + (1 + x ) ,1 + x ,{1 + x  + (1 + x ) ,1 + x },{1,5,6,7,8,9}}
```

In line 5, `#0[0]` is a list of "initialization files" read in when an SMP job is initiated, and containing SMP input which is simplified at the beginning of the job.

In line 6, $i..j$ gives $i, i+1, \ldots, j$.

Input expressions are stored in an unevaluated form in the list #I. The #I[i]:: which begins each input line indicates that the unsimplified input expression is assigned as a "delayed value" for the projection #I[i]. An important distinction exists between the "immediate assignment" a:b used, for example, for the output lines #O, and the "delayed assignment" a::b used for the input lines #I. In immediate assignment, b is simplified, and the result is assigned as the value of a. In delayed assignment, the unsimplified form of b is maintained as the value of a, and is simplified afresh each time a is used. Thus subsequent re-definitions relevant to the simplification of b are used with delayed assignment but not with immediate assignment.

```
#I[1]::  a:x+1

#O[1]:   1 + x

#I[2]::  a^2+a

                          2
#O[2]:   1 + x + (1 + x)

#I[3]::  a:x-2

#O[3]:   -2 + x

#I[4]::  @2

                          2
#O[4]:   1 + x + (1 + x)

#I[5]::  #I[2]

                          2
#O[5]:   -2 + x + (-2 + x)

#I[6]::  t::b+2

#O[6]:   ' b + 2

#I[7]::  u:b+2

#O[7]:   2 + b

#I[8]::  b:3

#O[8]:   3

#I[9]::  t

#O[9]:   5

#I[10]::  u

#O[10]:   5

#I[11]::  b:4

#O[11]:   4

#I[12]::  t

#O[12]:   6

#I[13]::  u

#O[13]:   5
```

Both t and u as defined in lines 6 and 7 depend on b. After the value of b is reassigned in line 11, the value obtained for t uses the new value of b while that for u does not.

The "apostrophe", "acute accent" or "quote mark" (') in line 6 indicates that the expression b+2 is unsimplified.

As illustrated in the example, an input expression or "command" may be "re-executed" in the current environment or with current definitions simply by asking for its value. This operation may be considered as a simple case of a "program" executed many times under different conditions or using different data.

```
#I[1]::  x^2-1

                  2
#O[1]:   -1 + x

#I[2]::  S[%,x->3]

#O[2]:   8

#I[3]::  x^3+4x-1

                  3
#O[3]:   -1 + 4x + x

#I[4]::  #I[2]

#O[4]:   38

#I[5]::  (x+y+4)^2

                        2
#O[5]:   (4 + x + y)

#I[6]::  #I[2]

                  2
#O[6]:   (7 + y)

#I[7]::  Edh[#I[2]]

          S[%,x->3]
<edit>      #^@3 5
          S[@3,x->5]
<edit>

#O[7]:   144

#I[8]::  f[$a]::S[$a,x->3]

#O[8]:   ' S[$a,x -> 3]

#I[9]::  f[x^2-1]

#O[9]:   8

#I[10]::  f[(x+1)(x-y)^2]

                        2
#O[10]:   4 (3 - y)

#I[11]::  f

#O[11]:   {[$a]:: S[$a,x -> 3]}
```

On line 7, Edh was used to change the unevaluated input line #I[2] using the editor.

After the definition on line 8, f acts as a "function" which substitutes 3 for x in any expression. Notice that if f had been defined by an immediate (:) rather than a delayed (::) assignment, the substitution would have been performed on the symbol $a at the time of assignment, rather than on each separate occasion when a projection of f is used, with the result that f[$a] would have been set to $a.

When a sequence of commands is typed on successive input lines, the results of each command are printed on the corresponding output line. A sequence of commands whose results need not be displayed may be entered on a single input line, separated by ;. A semicolon at the end of an input line causes the input expression or expressions to be simplified without printing the result.

```
#I[1]::  a:3;a^2-a

#O[1]:   6

#I[2]::  t:a^a-a^2;

#I[3]::  t

#O[3]:   18
```

A set of definitions which will be used in several SMP jobs may be saved in an external file, and read into each SMP job as required. An external file consists of SMP input lines which are simplified in order when the file is read, just as if they were typed as explicit input. An external file with name *file* is read into an SMP job by *<file*. Many external files containing functions with specialized applications are provided with SMP, as described in the SMP Library section of this handbook. An example is the file XLCM (note that file names may differ between installations, as specified in the Implementation Notes):

```
                    /**  Lowest common multiple  **/

/* LCM[n1,n2,...]
         yields the lowest common multiple of n1,n2,... */

         LCM_:Flat
         LCM[$n1,$n2] :: ($n1 $n2)/Gcd[$n1,$n2]
```

The text enclosed between /* and */ is treated as "commentary" and is not processed.

```
#I[1]::  <XLCM

#I[2]::  LCM[12,21,7]

#O[2]:   84

#I[3]::  LCM[2,6,8]

#O[3]:   48

#I[4]::  <Xlcm

#O[4]:   <Xlcm
```

In line 4, an attempt was made to read the non-existent file Xlcm: return of the unevaluated command signifies failure of the attempt. Files other than the standard SMP ones may be specified by complicated names, such as /u1/swolf/smp/fun; such file names must be enclosed in quotes: <"/u1/swolf/smp/fun".

Many external files contain tables of relations and transformations for mathematical functions, given in the form of "replacements" to be applied when required using S (see sect. 1). For example, the file XTr4 gives replacements for hyperbolic functions:

```
        /** Elementary Transcendental Functions - 4 **/

        /* 4.  Hyperbolic Functions  */

        /* 4.1  Relations to Other Functions  */

STr_Ldist
STr[4,1,1]:   Sinh[$x] -> (Exp[$x]-Exp[-$x])/2

STr[4,1,2]:   Cosh[$x] -> (Exp[$x]+Exp[-$x])/2

STr[4,1,3]:   Tanh[$x] -> (Exp[$x]-Exp[-$x])/(Exp[$x]+Exp[-$x])

STr[4,1,4]:   Sinh[$x] -> -I Sin[I $x]

STr[4,1,5]:   Cosh[$x] -> Cos[I $x]

STr[4,1,6]:   Tanh[$x] -> -I Tan[I $x]

/* All other relations may be obtained by using [4,1,1] - [4,1,6] to convert
to circular functions and using the relations in Sec 2. */


#I[1]::  <XTr4

#I[2]::  t:Sinh[x+1]+Sinh[x-1]+Tanh[2x]

#O[2]:   Sinh[-1 + x] + Sinh[1 + x] + Tanh[2x]

#I[3]::  S[t,STr[4,1,1]]

                -Exp[-(-1 + x)]   Exp[-1 + x]   Exp[-(1 + x)]   Exp[1 + x]
#O[3]:   ---------------------- + ------------ - ------------- + ----------- + Tanh[2x]
                      2                2               2              2

#I[4]::  S[t,STr[4,1,3]]

                -Exp[-2x] + Exp[2x]
#O[4]:   ------------------------------- + Sinh[-1 + x] + Sinh[1 + x]
                Exp[-2x] + Exp[2x]

#I[5]::  S[t,STr[4,1,{4..6}]]

#O[5]:*  (-Sin[(-1 + x) I] - Sin[(1 + x) I] - Tan[2x I]) I

#I[6]::  S[t,STr[4,1,1],STr[4,1,6]]

                 -Exp[-(-1 + x)]   Exp[-1 + x]   Exp[-(1 + x)]   Exp[1 + x]
#O[6]:*  (---------------------- + ------------ - ------------- + -----------)
                        2                2               2              2

                + -Tan[2x I] I

#I[7]::  S[t,STr[4,1]]

                -Exp[-2x] + Exp[2x]   Exp[-(-1 + x)]   Exp[-1 + x]   Exp[-(1 + x)]
#O[7]:   ------------------------------- - ---------------- + ------------ - -------------
                Exp[-2x] + Exp[2x]              2                2               2

                          Exp[1 + x]
                        + -----------
                              2
```

In line 7, `STr[4,1]` is the list of all relations defined in `XTr4`. `S` applies the relations in the order given: in this case, the first three relations transform `t` so that the last replacements have no effect.

The relations in `XTr4` were given as replacements and used selectively with `S`. `Arep[STr[4,1]]` would have converted the list of replacements to assignments and

thus defined the transformations to be performed automatically whenever applicable.

The Reference Manual lists many external files. Some simply define additional relations for mathematical functions. Others introduce entirely new mathematical objects. Still others define projections to be used in manipulating expressions. These external files may be used without understanding their construction. They may also be studied as examples of more advanced uses of SMP.

The external file XLdEq from [9.3] defines the function LdEq used in manipulating lists of equations:

```
                    /** Lists of equations **/

/* LdEq[expr]
        distributes Eq over lists in expr, thus converting equations
        involving lists into lists of equations. */
        LdEq[$expr] :: Ldist[$expr,'Eq]


#I[1]::  <XLdEq

#I[2]::  {x,y}={a,b}

#O[2]:   {x,y} = {a,b}

#I[3]::  LdEq[%]

#O[3]:   {x = a,y = b}
```

New external files may be prepared outside an SMP job (usually using an editor) according to the procedure outlined in the Implementation Notes. Each input line is terminated by a ⊂*newline*⊃ line must end with a backslash (\). Descriptive comments (whose value should not be underestimated) are enclosed between /* and */, and may extend over several lines.

External files may also be prepared using definition or results in an SMP job. Output[*x1*, *x2*, ..., *file*] places assignments for *x1*, *x2*, ... or their projections in *file* in a form suitable for subsequent input.

```
#I[1]::  Log[$x/$y]:Log[$x]-Log[$y]

#O[1]:   Log[$x] - Log[$y]

#I[2]::  Log[$x $$x]:Log[$x]+Log[$$x]

#O[2]:   Log[$$x] + Log[$x]

#I[3]::  Log[$x^$y]:$y Log[$x]

#O[3]:   $y Log[$x]

#I[4]::  Log

              $y
#O[4]:   {[$x  ]: $y Log[$x], [$$x $x]: Log[$$x] + Log[$x],

                  $x
                 [--]: Log[$x] - Log[$y]}
                  $y

#I[5]::  t:Log[x^2 y^3/(s+1)^3]

#O[5]:   2Log[x] + 3Log[y] - 3Log[1 + s]

#I[6]::  Output[Log,t,"log.ex"]
```

The file log.ex is then

```
Log[$x/$y] : Log[$x] + -Log[$y]
Log[$$x*$x] : Log[$$x] + Log[$x]
Log[$x^$y] : $y*Log[$x]
t : 2Log[x] + 3Log[y] + -3Log[1 + s]
```

This file may be used as input in another SMP job, or may be edited and manipulated outside of SMP.

```
#I[1]::  Log[x y^2]

                  2
#O[1]:   Log[x y ]

#I[2]::  <"log.ex"

#I[3]::  @1

#O[3]:   Log[x] + 2Log[y]

#I[4]::  t-%

#O[4]:   Log[x] + Log[y] - 3Log[1 + s]

#I[5]::  Output[#0,res0]

#I[6]::  Output[#I,resI]
```

On lines 5 and 6 the complete lists of output and input expressions were saved in the files res0

```
#O[0,1] : "/u1/smp/SRC/smp.in"
#O[1] : Log[x] + 2Log[y]
#O[2] : Null
#O[3] : Log[x] + 2Log[y]
#O[4] : Log[x] + Log[y] + -3Log[1 + s]
```

and resI

```
#I[0] :: Init
#I[1] :: Log[x*y^2]
#I[2] :: <"log.ex"
#I[3] :: #O[1]
#I[4] :: t + -%
#I[5] :: Output[#0,res0]
```

In most implementations, printed hard copies of expressions may be generated from within SMP by Hard[*expr*]. When possible, hard copies of graphics output (see sect. 7) may be generated in this way.

During an SMP job, it is sometimes necessary to manipulate external files or perform other operations outside SMP. The remainder of an input line whose first character is ! is passed as a command to the monitor (shell or command line interpreter), and is ignored by SMP. The details of some possible commands are given in the Implementation Notes.

```
#I[1]::  t:2
#O[1]:    2

#I[2]::   !p log.ex
Log[$x/$y] : Log[$x] + -Log[$y]
Log[$$x*$x] : Log[$$x] + Log[$x]
Log[$x^$y] : $y*Log[$x]
t : Log[x^2*y^3/(s + 1)^3]
#I[2]::  t^2
```

```
#0[2]:    4
```

The monitor command `p log.ex` used to print the file `log.ex` is specific to the implementation used for the example.

Further examples

Example 1

Define a function to convert $\sin(x)$ and $\cos(x)$ in an expression to complex exponential form. Save the function in the file `csrep`.

```
#I[1]::  r1:Cos[$x]->(Exp[I $x]+Exp[-I $x])/2

                      Exp[-$x I] + Exp[$x I]
#0[1]:*  Cos[$x] -> ----------------------
                              2

#I[2]::  r2:Sin[$x]->-I(Exp[I $x]-Exp[-I $x])/2

                      -(-Exp[-$x I] + Exp[$x I])
#0[2]:*  Sin[$x] -> --------------------------- I
                              2

#I[3]::  ef[$e]::S[$e,r1,r2]

#0[3]:   ' S[$e,r1,r2]

#I[4]::  Sin[3.2]+(1+Cos[Pi/4]^2)^(1/2)

                          2 .5
#0[4]:   (1 + Cos[.25Pi] )    + Sin[3.2]

#I[5]::  ef[%]

                                                2
          (Exp[-Pi/4 I] + Exp[Pi/4 I])  1/2    Exp[-16I/5]   Exp[16I/5]
#0[5]:*  (1 + ----------------------------)  + (----------- - ----------) I
                        4                        2             2

#I[6]::  N[%]

#0[6]:   1.166371

#I[7]::  Output[r1,r2,ef,csrep]

#0[7]:   {[$e]:: S[$e,r1,r2]}
```

Example 2

Define a function to evaluate the norm of a vector.

```
#I[1]::  norm[$l]::N[Sqrt[$l.$l]]

#0[1]:   ' N[Sqrt[$l.$l]]

#I[2]::  v:{.23,-1.1,2.7}

#0[2]:   {.23,-1.1,2.7}

#I[3]::  norm[v]

#0[3]:   2.924534
```

Example 3

Simplify $\Gamma(\frac{13}{4})\,\Gamma(\frac{15}{4})$ using the transformations given in the external file `XGamma`.

```
#I[1]::  <<Gamma

#I[2]::  SGamma[8]

                                  1 - 2$z    1/2
                                 2          Pi     Gamma[2$z]
#O[2]:   Gamma[$z] -> ------------------------------------
                                    Gamma[1/2 + $z]

#I[3]::  Gamma[3.75] S[Gamma[3.25],SGamma[8]]

           .5
         Pi    Gamma[6.5]
#O[3]:   ---------------
                5.5
               2

#I[4]::  SGamma[6]

                                                 1        1/2 - $n    1/2
#O[4]:   Gamma[$n _= Natp[$n - -]] -> 2           Pi     (-2 + 2$n)!!
                                     2

#I[5]::  S[@3,SGamma[6]]

         10395Pi
#O[5]:   --------
             11/2
         64 2

#I[6]::  N[%]

#O[6]:   11.27533

#I[7]::  N[Gamma[3.25]Gamma[3.75]]

#O[7]:   11.27533
```

5. Manipulating expressions

The two expressions b+c+a and a+c+b, while superficially different, are mathematically equal. This equality is made manifest by casting both expressions into the canonical form a+b+c in which the arguments of the commutative function Plus have been ordered. Simple transformations such as this are performed automatically by SMP.

In the more complicated case of the expressions x+x^2 and x(1+x), a canonical may be obtained by expansion (using Ex). However, no algorithm exists for placing an arbitrarily complicated expression in a canonical form. The beginning of this section concentrates on the variety of functions in SMP for transforming rational expressions. The external files discussed in sect. 4 contain many transformations for transcendental and special functions.

```
#I[1]:: a:(1+x^2)/((1-x)(2-x)(3-x))

                          2
                    1 + x
#O[1]:  ------------------------
        (1 - x) (2 - x) (3 - x)

#I[2]:: Int[%,x]

#O[2]:  Log[1 - #1] - Log[1 - x] - 5Log[2 - #1] + 5Log[2 - x] + 5Log[3 - #1]

                  - 5Log[3 - x]

#I[3]:: D[%,x]

          1         5         5
#O[3]:  ----- - ----- + -----
        1 - x   2 - x   3 - x

#I[4]:: b:%;

#I[5]:: Neq[a,b]

#O[5]:  1

#I[6]:: Rat[b]

                          2
                    1 + x
#O[6]:  ------------------------
        (1 - x) (2 - x) (3 - x)

#I[7]:: Ex[a]

                                        2
                  1                    x
#O[7]:  ------------------- + -------------------
                      2    3                  2    3
        6 - 11x + 6 x  - x    6 - 11x + 6 x  - x

#I[8]:: Col[%]

                        2
                  1 + x
#O[8]:  -------------------\
                      2    3
        6 - 11x + 6 x  - x

#I[9]:: Fac[%]
```

```
                                  2
                          -(1 + x )
#O[9]:    ---------------------------------
          (-3 + x) (-2 + x) (-1 + x)

#I[10]::  Pf[a,x]

            1       5       5
#O[10]:   ----- - ----- + -----
          1 - x   2 - x   3 - x
```

Since b is the result of differentiating the integral of a, a and b must be mathematically equal. They are, however, written in different forms. The projection Neq [*expr1*, *expr2*] used on line 5 tests for the numerical equality of *expr1* and *expr2* by substituting sets of random numbers for symbols which appear in them*.

The projection Rat [*expr*] used on line 6 introduces a common denominator for all terms in *expr*, multiplying the numerators by the necessary factors, and in this case casts b into the same form as a.

Lines 7 through 10 give further forms for a. On line 7, Ex was used to perform all the possible distributions, leaving an expression containing no parentheses. In line 8, terms with the same denominator were collected using Col. The result was then factorized in line 9. Finally, in line 10, the expression a was placed in a partial fraction form with respect to x using Pf.

```
#I[1]::   (1-x^2)(x+a)(x-3)+(2-a)(x^2+1)(1+x)^6

                                 2                          6       2
#O[1]:    (-3 + x) (1 - x ) (a + x) + (1 + x)  (1 + x ) (2 - a)

#I[2]::   Ex[%]

                                          2       3       4       5       6
#O[2]:    2 - 4a + 9x - 5a x - 13a x  - 27a x  - 30a x  - 26a x  - 16a x

                 7       8       2       3       4       5       6       7
            - 6a x  - a x  + 33 x  + 55 x  + 59 x  + 52 x  + 32 x  + 12 x

                 8
            + 2 x

#I[3]::   Fac[%]

                                                  2       3       4       5
#O[3]:    -(1 + x) (-2 + 4a - 7x + a x + 12a x  + 15a x  + 15a x  + 11a x

                           6       7       2       3       4       5       6
                    + 5a x  + a x  - 26 x  - 29 x  - 30 x  - 22 x  - 10 x

                           7
                    - 2 x )

#I[4]::   Cb[@2,a]

                                          2       3       4       5       6       7
#O[4]:    2 + 9x + a (-4 - 5x - 13 x  - 27 x  - 30 x  - 26 x  - 16 x  - 6 x

                           8       2       3       4       5       6       7
                      - x ) + 33 x  + 55 x  + 59 x  + 52 x  + 32 x  + 12 x

                 8
            + 2 x

#I[5]::   Coef[a,@2]
```

* This test is clearly probabilistic in nature; the chance that it fails is however infinitesimal.

```
#0[5]:   -4 - 5x - 13 x^2 - 27 x^3 - 30 x^4 - 26 x^5 - 16 x^6 - 6 x^7 - x^8

#I[6]::  Cb[@2,x^$i]

#0[6]:   2 - 4a + 9x - 5a x + x^2 (33 - 13a) + x^3 (55 - 27a) + x^4 (59 - 30a)
           + x^5 (52 - 26a) + x^6 (32 - 16a) + x^7 (12 - 6a) + x^8 (2 - a)

#I[7]::  Cb[@2,{x^$i,x}]

#0[7]:   2 - 4a + x (9 - 5a) + x^2 (33 - 13a) + x^3 (55 - 27a) + x^4 (59 - 30a)
           + x^5 (52 - 26a) + x^6 (32 - 16a) + x^7 (12 - 6a) + x^8 (2 - a)
```

Application of Ex on line 2 gave a comparatively large expression: to avoid unintentional generation of huge expressions, Ex should be used with forethought. In line 3, Fac could not entirely reverse the effect of Ex. Cb[*expr*,*form*] combines coefficients of *form* in *expr*, or of patterns matching *form*. Thus in line 6, Cb[@2,x^$i] combines coefficients of each power of x. In line 7, the list {x^$i,x} causes collection of terms proportional to x as well as to powers of x. On line 5, Coef was used to extract the total coefficient of a.

The various projections illustrated in the examples above provide tools for manipulating expressions into "simpler" forms. The optimal procedure in particular cases is usually obtained through trial and error.

It is often convenient to apply the same operation or "simplification" to a set of expressions. Map[*temp*,*expr*] applies the "template" *temp* to parts in *expr* lying on the "first level".

```
#I[1]::  r:{a,b,c}

#0[1]:   {a,b,c}

#I[2]::  Map[f,r]

#0[2]:   {f[a],f[b],f[c]}

#I[3]::  s:a^2+b+c(c+1)

#0[3]:   b + c (1 + c) + a^2

#I[4]::  Map[f,s]

#0[4]:   f[b] + f[c (1 + c)] + f[a^2]

#I[5]::  s^2

#0[5]:   (b + c (1 + c) + a^2)^2

#I[6]::  Map[$x^2,s]

#0[6]:   c^2 (1 + c)^2 + a^4 + b^2

#I[7]::  Map[f,s,Inf]

#0[7]:   f[b] + f[f[c] f[f[1] + f[c]]] + f[f[a]^f[2]]
```

When the "template" is a single symbol, such as f in lines 2 and 4, the application of the template to *exp* by Map consists in forming f[*exp*]. In application of a template containing a generic symbol, as on line 6, the expression *exp* is inserted in place of the generic symbol.

On line 4, Map applied the template f to each subexpression in the "first level" of s. The *n*th "level" in an expression is in general defined to be the set of all parts which may be extracted by projection with *n* filters. Thus the "first level" in s consists of the arguments of Plus. By default, Map treats only the first level in an expression. In line 7, however, Map was explicitly specified to treat an infinite number of levels, so that f was applied to all subparts of s.

In general, it is possible to treat a "domain" containing an arbitrary set of subparts of an expression. The parts to be included in a domain are determined by their levels and by a "criterion" which they must satisfy, as described in [2.5].

Domains may also be specified directly for projections such as Fac and Cb, as described in the Reference Manual.

It is often convenient to apply some operation or simplification automatically to each output expression. Any value assigned to the global object Post is applied to every output expression before it is printed. To obtain all expressions in expanded form Post:Ex. To obtain a numerical value for each output expression Post:N.

```
#I[1]::  (1+x)(1-x)

#O[1]:   (1 - x) (1 + x)

#I[2]::  Post:Ex

#O[2]:   Ex

#I[3]::  @1

                  2
#O[3]:   1 - x

#I[4]::  a(a+1)^3

                2       3    4
#O[4]:   a + 3 a  + 3 a  + a

#I[5]::  Post:N

#O[5]:   N

#I[6]::  Sin[1]+Gamma[4.5]

#O[6]:   12.4732

#I[7]::  Post:

#I[8]::  Sin[1]+Gamma[4.5]

#O[8]:   Gamma[4.5] + Sin[1]
```

Further examples

Example 1
Express $(2-a)\,(a+\sin(bx))^2\,(1+a^2+3\sin(bx))$ as a polynomial in $\sin(bx)$.

```
#I[1]:  (2-a)(a+Sin[b x])^2(1+a^2+3Sin[b x])

#O[1]:  (2 - a) (a + Sin[b x])^2 (1 + a^2 + 3Sin[b x])

#I[2]:  Ex[%]

#O[2]:  11a Sin[b x]^2 - 3a Sin[b x]^3 + 4a Sin[b x] - 4 a^2 Sin[b x]^2

                  2          3          2    3          4
        + 4 a^2 Sin[b x] - a  Sin[b x] + a  Sin[b x] - 2 a  Sin[b x]

                  2    3     4    5               2          3
        + 2 a^2 - a  + 2 a  - a  + 2 Sin[b x]^2 + 6 Sin[b x]

#I[3]:  Cb[%,Sin[b x]^$n]

#O[3]:  4a Sin[b x] + (6 - 3a) Sin[b x]^3 + (2 + 11a - 4 a^2 - a^3) Sin[b x]^2

                  2          3          4          2    3
        + 4 a^2 Sin[b x] + a  Sin[b x] - 2 a  Sin[b x] + 2 a  - a

                  4    5
        + 2 a^4 - a

#I[4]:  Cb[%,Sin[b x]]

#O[4]:  (6 - 3a) Sin[b x]^3 + (2 + 11a - 4 a^2 - a^3) Sin[b x]^2

                  2    3     4                2    3     4    5
        + (4a + 4 a^2 + a  - 2 a ) Sin[b x] + 2 a  - a  + 2 a  - a
```

Example 2
Find the residues at the poles of $\dfrac{1+ax+3x^2}{x(x+1)(x-2)}$.

```
#I[1]:  (1+a x+3x^2)/(x(x+1)(x-2))

                      2
          1 + a x + 3 x
#O[1]:  ------------------
        x (-2 + x) (1 + x)

#I[2]:  Pf[%,x]

        -1    4/3 - a/3    13/6 + a/3
#O[2]:  -- + ---------- + -----------
        2x       1 + x        -2 + x
```

Example 3
Find the coefficient of x^2 in the second derivative of $(1+x^2)\,(1-ax)^3$.

```
#I[1]:  D[(1+x^2)(1-a x)^3,x,x]

                      2                2                2          3
#O[1]:  -12a x (1 - a x)  + 6 a^2 (1 - a x) (1 + x^2) + 2 (1 - a x)
```

```
#I[2]::  Ex[%]

                                  2  2       3           3  3       2
#O[2]:   2 - 18a x + 36 a^2 x^2 - 6 a  x - 20 a  x^3 + 6 a  x^2

#I[3]::  Coef[x^2,%]

                 2
#O[3]:   36 a^2
```

Example 4

Find the 5^{th} iterant of the map $x \to (1-\lambda x)^2$ at $x \to 1$ and $\lambda \to \frac{1}{2}$.

```
#I[1]::  r:$s->(1-lam $s)^2

                                2
#O[1]:   $s -> (1 - $s lam)^2

#I[2]::  S[x,r,5]

                                                                  2 2 2 2 2
#O[2]:   (1 - lam (1 - lam (1 - lam (1 - lam (1 - lam x)^2)^2)^2)^2)^2

#I[3]::  S[%,x->1,lam->1/2]

                    2
              26527/32768  2
#O[3]:   (1 - ------------)^2
                    2

#I[4]::  N[%]

#O[4]:   .4520178
```

6. Mathematical operations

In written mathematics, it is commonplace to use shortened notation and to infer the omitted details from convention or context of usage. For example, the indefinite integral $\int x\,dx$ is usually taken as $\frac{x^2}{2} = \int_0^x y\,dy$, while $\int \frac{dx}{x}$ is usually taken as $\log(x) = \int_1^x \frac{dy}{y}$. Similarly, the notation $f'(x^2)$ may mean either $\left.\frac{df(y)}{dy}\right|_{y=x^2}$ or $\left.\frac{df(y^2)}{dy}\right|_{y=x}$. Without further information, SMP could not resolve these ambiguities.

Mathematical operations in SMP must be defined precisely. General definitions may be used to introduce particular short notations.

```
#I[1]:: Int[x,x]

                 2    2
             - #1     x
#O[1]:       ------ + --
               2      2

#I[2]:: Int[1/x,x]

#O[2]:  -Log[#2] + Log[x]

#I[3]:: Int[x,{x,a,b}]

                2    2
            - a     b
#O[3]:      ---- + --
             2     2

#I[4]:: Int[f[x],{x,0,1}]

#O[4]:  Int[f[#3],{#3,0,1}]

#I[5]:: Int[f[x],x]

#O[5]:  Int[f[#5],{#5,#4,x}]
```

In the "indefinite" integrals on lines 1 and 2, SMP introduced the internally-generated symbols #1 and #2 to represent the lower limit of integration; the upper limit was taken as x. On line 3, the upper and lower limits were specified explicitly as b and a.

In line 4, the definite integral of the undefined function f was requested. The result was simply a canonical form of the integral, with x replaced by the "dummy" variable #3. On line 5, the canonical form for the indefinite integral involved both a dummy variable #5 and a lower limit of integration #4.

The definite integral of f required in line 4 may be defined to be c by the direct assignment `Int[f[$x],{$x,0,1}]:c`.

```
#I[1]:: Int[f[$x],{$x,0,1}]:c

#O[1]:  c

#I[2]:: Int[f[x]+x,{x,0,1}]

#O[2]:  1/2 + c

#I[3]:: Int[f[$x],{$x,$1,$2}]:cf[$2-$1]

#O[3]:  cf[-$1 + $2]
```

```
#I[4]::  Int[f[x],x]

#O[4]:   cf[-#2 + x]
```

On line 3, the indefinite integral of f was defined. As usual, the patterns are organized so that a definite integral of f[x] from 0 to 1 would use the specific assignment on line 1 rather than the more general one of line 3.

Int yields explicit results for the majority of rational, logarithmic, exponential and trigonometric expressions whose indefinite integrals lie in the same class of expressions*. To avoid generation of potentially huge expressions, Int does not automatically expand the integrand. Explicit use of Ex may therefore be required to cast the integrand into a suitable form for integration.

```
#I[1]::  Int[(1+x^2)/(1+x^3),x]

                                  1/2                                1/2
                 -1 + 2#1 - (-3)                    -1 + 2x - (-3)
         -3Log[-------------------]      3Log[------------------]
                                  1/2                                1/2
                 -1 + 2#1 + (-3)                    -1 + 2x + (-3)              2Log[1 + #1]
#O[1]:   ------------------------------ + ---------------------------- - ------------
                           1/2                                1/2                     3
                    2 (-3)                              2 (-3)

                                                            2                     2
                 2Log[1 + x]     Log[1 - #1 + #1 ]      Log[1 - x + x ]
               + -----------  + -----------------  - ---------------
                      3                      6                       6

#I[2]::  Int[x(1+x),x]

#O[2]:   Int[#3 (1 + #3),{#3,#2,x}]

#I[3]::  Ex[%]

              2        3       2       3
          - #2      #2       x       x
#O[3]:    ----- - --- + -- + --
            2        3       2       3

#I[4]::  Int[1/(1+x^2+x^3)^2,{x,0,1}]

                        28/31 + 2#4/31
#O[4]:   -7/93 + Int[---------------,{#4,0,1}]
                               2      3
                        1 + #4  + #4
```

The integral in line 1, while explicit, is rather complicated. The integral on line 4 was not completely evaluated by the built-in integrator; the part not done is however given in its simplest canonical form.

Definite integrals for which an explicit result is not found may be evaluated numerically using N if their value is a single number.

```
#I[1]::  Int[Gamma[x],{x,1,4}]

#O[1]:   Int[Gamma[#1],{#1,1,4}]

#I[2]::  N[%]

#O[2]:   5.852363
```

* All such integrals could be performed explicitly if it were possible to give an explicit solution for an arbitrary polynomial equation.

We next discuss the slightly more involved case of differentiation.

```
#I[1]::  D[x^a,x]

                -1 + a
#O[1]:   a x

#I[2]::  D[f[x],x]

#O[2]:   D[f[#1],{#1,1,x}]

#I[3]::  D[%,x]

#O[3]:   D[f[#1],{#1,2,x}]

#I[4]::  D[f[g[x]],x]

#O[4]:   D[f[#3],{#3,1,g[x]}] D[g[#2],{#2,1,x}]

#I[5]::  D[f[x,x],x]

#O[5]:   D[f[#4,x],{#4,1,x}] + D[f[x,#5],{#5,1,x}]
```

The expression on line 2 represents $\left.\frac{d^1 f(\#1)}{d\#1^1}\right|_{\#1=x}$. #1 is a dummy variable of differentiation; the complete result is a function only of the point x at which the derivative is evaluated. The form in line 3 represents the second derivative. In line 4, the chain rule is used to cast the derivative into a canonical form. The "function" $f(z_1,z_2)$ depends on two independent "variables" z_1 and z_2. The derivative of this function with respect to x along the curve $z_1=z_2=x$ is evaluated on line 5, yielding the result $\left.\frac{\partial f(z_1,z_2=x)}{\partial z_1}\right|_{z_1=x} + \left.\frac{\partial f(z_1=x,z_2)}{\partial z_2}\right|_{z_2=x}$.

Derivatives need not always be taken with respect to symbols: any expression containing only one distinct symbol may be used.

```
#I[1]::  D[f[x],g[x]]

         D[f[#2],{#2,1,x}]
#O[1]:   -----------------
         D[g[#1],{#1,1,x}]

#I[2]::  D[f[a x^2],Log[x]]

                  2
#O[2]:   2a x  D[f[#3],{#3,1,a x }]

#I[3]::  S[%,f->Sin]

                  2      2
#O[3]:   2a x  Cos[a x ]

#I[4]::  D[Sin[a x^2],Log[x]]

                  2      2
#O[4]:   2a x  Cos[a x ]

#I[5]::  D[%,x,x,x]

                  2          2          3  3          2        4  5          2
#O[5]:   2(-24 a  x Sin[a x ] - 36 a  x  Cos[a x ] + 8 a  x  Sin[a x ])

#I[6]::  D[@4,{x,3}]

                  2          2          3  3          2        4  5          2
#O[6]:   2(-24 a  x Sin[a x ] - 36 a  x  Cos[a x ] + 8 a  x  Sin[a x ])
```

The arbitrary function f in the derivative on line 2 was specified to be Sin on line 3: the result is the same as that obtained on line 4 where f was replaced by Sin before the derivative was taken. Lines 5 and 6 show two equivalent forms which yield the third derivative of @4.

Derivatives may be defined by direct assignments, just like integrals.

```
#I[1]::  D[f[$x],{$x,1,$y}]:g[$y]

#O[1]:   g[$y]

#I[2]::  D[f[x],x]

#O[2]:   g[x]

#I[3]::  D[f[x^2] x,x]

                  2     2          2
#O[3]:   2 x  g[x ] + f[x ]

#I[4]::  D[%,x]

                 2          3                2
#O[4]:   6x g[x ] + 4 x  D[g[#3],{#3,1,x }]

#I[5]::  D[h[$x,$y],{$x,1,1}]:c[$y]

#O[5]:   c[$y]

#I[6]::  D[h[x^2,y-1](x+3)^2,{x,1,1}]

#O[6]:   32c[-1 + y] + 8h[1,-1 + y]
```

Line 1 defined the "function" g to be the derivative of the function f. In line 4, the second derivative of f was required, and was written as a derivative of g. On line 5,

$\left.\frac{\partial h(x,y)}{\partial x}\right|_{x=1}$ was defined to be $c(y)$.

In the partial derivatives represented by D, all distinct symbols are assumed independent. Dt represents a total derivative, in which all symbols are assumed interdependent, unless they are defined to be constants (by *symb_*:Const [4]), or are explicit defined to have zero total derivatives.

```
#I[1]::  t:Pi x^a

                 a
#O[1]:   Pi x

#I[2]::  D[t,x]

                   -1 + a
#O[2]:   Pi a x

#I[3]::  Dt[t,x]

                        -1 + a     a
#O[3]:   Pi (a x              + x  Dt[a,x] Log[x])

#I[4]::  Dt[t]

                        -1 + a          a
#O[4]:   Pi (a x              Dt[x] + x  Dt[a] Log[x])
```

On line 4, the total differential of t was given.

Ps[*expr*, *x*, *x0*, *n*] yields the power (Taylor-Laurent) series of *expr* with respect to *x* about the point *x*=*x0* to *n* terms.

```
#I[1]::  t:Ps[x Exp[x],x,0,4]

                        3    4
                   2   x    x
#0[1]:*  x + x  + 1 + -- + --
                       2    6

#I[2]::  t^2+1/t

                                  2
          -1                   5 x
#0[2]:*  x   - 1 + x/2 + 1 + ----
                               6

#I[3]::  Ax[@2]

                                  2
                        1    5 x
#0[3]:   -1 + x/2 + 1 + - + ----
                        x    6

#I[4]::  @2^2

           -2        -1
#0[4]:*  x   - 2 x   + 2 + 2x/3

#I[5]::  @3^2

                                  2
                        1    5 x   2
#0[5]:   (-1 + x/2 + 1 + - + ----)
                        x    6
```

As mentioned in sect. 3 above, the star at the beginning of the output on line 1 indicates that the result is represented* and treated in a special manner: in manipulations such as line 2, the power series is truncated as necessary. The projection Ax used on line 3 converts the power series into an ordinary polynomial, whose square in line 5 is left unexpanded and untruncated.

The assignment of power series expansions for functions is described in [9.5]; if no expansion has been assigned, information on derivatives is used if appropriate.

Essential singularities are factored out of power series when they are encountered.

The specification of multi-variate power series is described in [9.5].

In addition to ordinary power series, [9.5] describes the projections Ra and Cf used to obtain rational (Pade) and continued fraction approximations.

Lim[*expr*, *x*, *x0*] yields the limit of *expr* when *x* tends to *x0*.

```
#I[1]::  Lim[Sin[x]/x,x,0]

#0[1]:   1

#I[2]::  Lim[Cos[x]/x,x,0]

           -1
#0[2]:*  x   - x/2
```

As revealed on line 2, Lim in general yields a power series.

* The parts in the representation are not manifest from the output form: the projection Lpr[*expr*] prints *expr* in an explicit form.

Sum[*expr*, *i*, *imin*, *imax*] yields $\sum_{i=imin}^{imax} expr$; Prod yields the corresponding product.

```
#I[1]:  Prod[f[i],i,3,10]

#O[1]:  f[3] f[4] f[5] f[6] f[7] f[8] f[9] f[10]

#I[2]:  Sum[i^2,i,1,20]

#O[2]:  2870
```

An equation *expr1=expr2* is automatically simplified by linear elimination of parameters. If *expr1* and *expr2* are determined to be identical, it yields 1; if they are found unequal, it gives 0. If possible, an explicit solution or solutions for a parameter *x* appearing in an equation *eqn* is found by Sol[*eqn*, *x*].

```
#I[1]:  2x+3a-5=6x-4a+3

#O[1]:  7a = 8 + 4x

#I[2]:  Sol[%,x]

                  -8 + 7a
#O[2]:  {x -> --------}
                   4

#I[3]:  Sol[x^2-3x+7=0,x]

                          1/2                          1/2
                  (-19)                        (-19)
#O[3]:  {x -> 3/2 + --------,x -> 3/2 - --------}
                     2                            2

#I[4]:  N[%]

#O[4]:* {x -> 1.5 + 2.179449I,x -> 1.5 - 2.179449I}

#I[5]:  Sol[Gamma[x]=x,x]

#O[5]:  {Gamma[x] = x}

#I[6]:  Sol[{x+2y=3a,x-5y=a},{x,y}]

#O[6]:  {x -> 17a/7,y -> 2a/7}
```

Linear elimination was automatically on the equation in line 1: an explicit solution for x was obtained on line 2, and given as a replacement. Similar replacements were obtained for the two solutions to the quadratic equation on line 3; complex numerical values of these roots were obtained on line 4. The transcendental equation on line 5 could not be solved or simplified, and the result was an equation, rather than a replacement representing an explicit solution. Line 6 is an example of solution to a list of simultaneous equations.

Further examples

Example 1
Find the value of $\int_0^1 e^{x^3} dx$.

```
#I[1]::  Int[Exp[x^3],{x,0,1}]

                    3
#O[1]:   Int[Exp[#1 ],{#1,0,1}]

#I[2]::  N[%]

#O[2]:   1.341904
```

Example 2
Define a function to obtain Legendre polynomials using Rodrigues's formula:

$$P_l(x) = \frac{1}{2^l l!} \frac{d^l}{dx^l}(x^2-1)^l$$

```
#I[1]::  p[$l,$x]::Ex[1/(2^$l $l!)D[(z^2-1)^$l,{z,$l,$x}]]

                  1          2      $l
#O[1]:   ' Ex[--------- D[(z  - 1)  ,{z,$l,$x}]]
               $l
             2   $l!

#I[2]::  p[0,x]

#O[2]:   1

#I[3]::  p[2,x]

                    2
                 3 x
#O[3]:   -1/2 + ----
                 2

#I[4]::  p[10,x]

                          2         4         6         8          10
                    3465 x    15015 x    45045 x    109395 x    46189 x
#O[4]:   -63/256 + ------- - --------- + --------- - ---------- + ----------
                     256       128       128        256         256
```

Example 3
Find the gradient of the function $f(x)=2x^2+3x+4xy-5y^2+7$, and deduce the positions of any saddle points.

```
#I[1]::  2x^2+3x+7+4x y-5y^2

                                 2       2
#O[1]:   7 + 3x + 4x y + 2 x^2 - 5 y

#I[2]::  {D[%,x],D[%,y]}

#O[2]:   {3 + 4x + 4y,4x - 10y}

#I[3]::  Sol[%,{x,y}]

#O[3]:   {x -> 3/4,y -> -3/10}
```

Example 4

Find the power series expansion of $r\,e^{r^2}$ where $r=1+x+x^2$ about the point $x=0$ to fifth order.

```
#I[1]::  r:1+x+x^2

                    2
#O[1]:   1 + x + x

#I[2]::  Ps[r Exp[r^2],x,0,5]

                          2          3          4          5
#O[2]:*  E + 3E x + 8E x  + 49E/3 x  + 61E/2 x  + 1523E/30 x
```

Example 5

Calculate the Bernoulli numbers up to B_{10} using the generating function:

$$\frac{t}{e^t-1}=\sum_{n=0}^{\infty} B_n \frac{t^n}{n!}$$

```
#I[1]::  Ps[t/(Exp[t]-1),t,0,11]

                      2     4       6                          8                          10
                     t     t       t
#O[1]:*  1 - t/2 + -- - --- + ----- - 8.267196*^-07 t  + 2.087676*^-08 t
                    12   720   30240

#I[2]::  Ax[%]

                      2     4       6                          8                          10
                     t     t       t
#O[2]:   1 - t/2 + -- - --- + ----- - 8.267196*^-07 t  + 2.087676*^-08 t
                    12   720   30240

#I[3]::  S[%,t^$n->t^$n $n!]

                      2     4     6     8       10
                     t     t     t     t     5 t
#O[3]:   1 - t/2 + -- - -- + -- - -- + -----
                    6     30    42    30     66

#I[4]::  S[%,Plus->List,t->1,Inf]

#O[4]:   {1,-1/2,1/6,-1/30,1/42,-1/30,5/66}
```

Example 6

Calculate the exponential constant e from power series and continued fraction approximations to e^x.

```
#I[1]::  Ps[Exp[x],x,0,4]

                     2     3     4
                    x     x     x
#O[1]:*  1 + x + -- + -- + --
                    2     6    24

#I[2]::  S[Ax[%],x->1]

#O[2]:   65/24

#I[3]::  N[%]

#O[3]:   2.708333
```

```
#I[4]:  Cf[Exp[x],x,0,4]

          1    -x  x/2  -x/6  x/6
#O[4]:* ----- ---- ---- ----- ----
        1 + 1 + 1 + 1 + 1 +

#I[5]:  #I[2]

#O[5]:  19/7

#I[6]:  N[%]

#O[6]:  2.714286

#I[7]:  N[E]

#O[7]:  2.718282
```

Example 7

Define the derivatives of the Γ function using the relations:

$$\frac{d}{dx}\Gamma(x) = \psi^{(0)}(x)\Gamma(x)$$

$$\frac{d}{dx}\psi^{(n)}(x) = \psi^{(n+1)}(x)$$

```
#I[1]:  D[Gamma[$x],{$x,1,$y}]:Psi[1,$y] Gamma[$y]

#O[1]:  Gamma[$y] Psi[1,$y]

#I[2]:  D[Gamma[Exp[x^2]],x]

                  2                  2                  2
#O[2]:  2x Exp[x ] Gamma[Exp[x ]] Psi[1,Exp[x ]]

#I[3]:  D[Psi[$n,$x],{$x,1,$y}]:Psi[$n+1,$y]

#O[3]:  Psi[1 + $n,$y]

#I[4]:  D[Gamma[x],x,x,x]

                                3
#O[4]:  Gamma[x] Psi[1,x]  + Gamma[x] Psi[3,x] + 3Gamma[x] Psi[1,x] Psi[2,x]
```

7. Presentation of results

Many results are best presented in the form of tables or graphs. SMP allows tables and graphs not only to be generated, but also to be manipulated as symbolic expressions in their own right.

Lists may be considered as tables. For example, values of a function at a set of points may be tabulated in a list whose indices correspond to the points.

`Ar[`*n*`,` *temp*`]` yields a contiguous list whose entries have indices 1,...,*n* and values obtained by application of the "template" *temp* (see sect. 5) to these indices.

```
#I[1]:: Ar[5,f]
#O[1]:  {f[1],f[2],f[3],f[4],f[5]}
#I[2]:: Ar[5,Log]
#O[2]:  {0,Log[2],Log[3],Log[4],Log[5]}
#I[3]:: N[%]
#O[3]:  {0,.6931472,1.098612,1.386294,1.609438}
#I[4]:: Ar[5,$x^2]
#O[4]:  {1,4,9,16,25}
#I[5]:: Ar[{{0,1.2,0.1}},f]
#O[5]:  {[0]: f[0], [.1]: f[.1], [.2]: f[.2], [.3]: f[.3], [.4]: f[.4],
         [.5]: f[.5], [.6]: f[.6], [.7]: f[.7], [.8]: f[.8],
         [.9]: f[.9], [1]: f[1], [1.1]: f[1.1]}
#I[6]:: Ar[{{0,1.2,0.1}},$x+$x^2]
#O[6]:  {[0]: 0, [.1]: .11, [.2]: .24, [.3]: .39, [.4]: .56, [.5]: .75,
         [.6]: .96, [.7]: 1.19, [.8]: 1.44, [.9]: 1.71, [1]: 2,
         [1.1]: 2.31}
#I[7]:: %^2+0.6
#O[7]:  {[0]: .6, [.1]: .6121, [.2]: .6576, [.3]: .7521, [.4]: .9136,
         [.5]: 1.1625, [.6]: 1.5216, [.7]: 2.0161, [.8]: 2.6736,
         [.9]: 3.5241, [1]: 4.6, [1.1]: 5.9361}
```

In line 5, the indices in the list were specified to run from `0` to `1.2` in steps of `0.1`. Notice that in line 7, the specified operation was performed on each element of the list, and a new transformed table was generated.

The tables in this example are analogous to vectors. `Ar` may also be used to generate tables in which each element carries a set of indices, as in a matrix or tensor.

```
#I[1]:: Ar[{2,3},f]
#O[1]:  {{f[1,1],f[1,2],f[1,3]},{f[2,1],f[2,2],f[2,3]}}
#I[2]:: Ar[{2,3},f[$i,$j^2]]
#O[2]:  {{f[1,1],f[1,4],f[1,9]},{f[2,1],f[2,4],f[2,9]}}
#I[3]:: Ar[{2,3},1/($i+$i^2+$j)]
```

```
#O[3]:   {{1/3,1/4,1/5},{1/7,1/8,1/9}}
#I[4]::  Ar[{{0,1,0.3},{4,5}},f]
#O[4]:   {[0]: {[4]: f[0,4], [5]: f[0,5]},
                [.3]: {[4]: f[.3,4], [5]: f[.3,5]},
                [.6]: {[4]: f[.6,4], [5]: f[.6,5]},
                [.9]: {[4]: f[.9,4], [5]: f[.9,5]}}
#I[5]::  Ar[{{0,1,0.3},{4,5}},N[Log[$x+$y^2]]]
#O[5]:   {[0]: {[4]: 2.772589, [5]: 3.218876},
                [.3]: {[4]: 2.791165, [5]: 3.230804},
                [.6]: {[4]: 2.809403, [5]: 3.242592},
                [.9]: {[4]: 2.827314, [5]: 3.254243}}
#I[6]::  %^2
#O[6]:   {[0]: {[4]: 2.772589^2, [5]: 3.218876^2},
                [3/10]: {[4]: 2.791165^2, [5]: 3.230804^2},
                [3/5]: {[4]: 2.809403^2, [5]: 3.242592^2},
                [9/10]: {[4]: 2.827314^2, [5]: 3.254243^2}}
#I[7]::  Ar[{2,2,2},f]
#O[7]:   {{{f[1,1,1],f[1,1,2]},{f[1,2,1],f[1,2,2]}},
                {{f[2,1,1],f[2,1,2]},{f[2,2,1],f[2,2,2]}}}
```

In line 1, Ar was used to generate a 2×3 matrix, each of whose entries was given by applying f to its two indices. On line 2, a 2×3 matrix whose i,jth element is $f(i,j^2)$ was obtained by applying the template f[$i,$j^2] to the indices for each element. When a template containing several generic symbols is applied to a set of expressions, the generic symbol which appears first in lexical ordering is associated with the first expression, the second with the second expression, and so on.

In line 7 above, a rank-3 2×2×2 tensor whose i,j,kth element is $f(i,j,k)$ was generated.

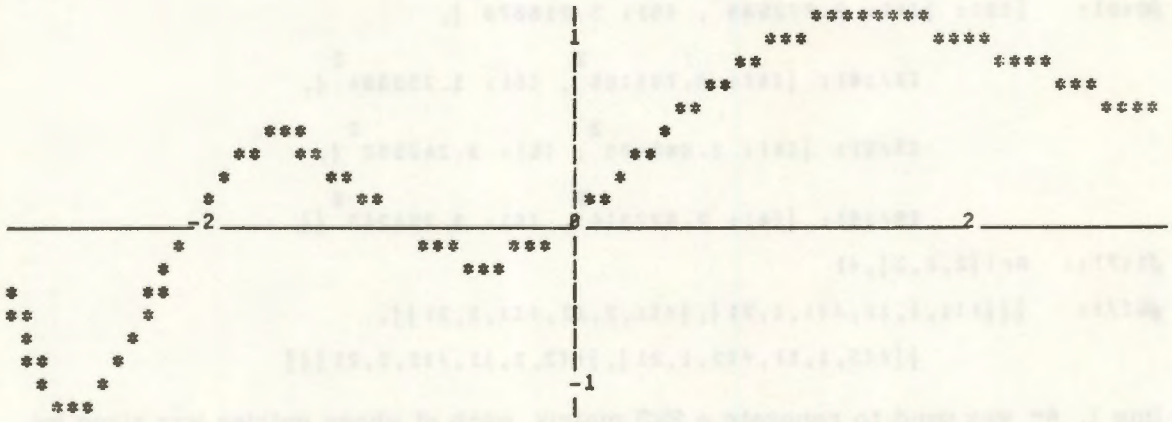
In some cases, it suffices to generate and print a table once, without retaining a permanent record of its elements in the form of a list. Do[*i, imin, imax, istep, expr*] evaluates *expr* with i equal to *imin*, *imin+istep*, ... , *imax*. If *istep* is omitted, it is taken as 1, and if in addition, *imin* is omitted, it also takes the value 1. Pr[*expr1, expr2,...*] prints a sequence of expressions.

```
#I[1]::  Do[i,5,Pr[i,i!,i!!,i^i]]

1       1       1       1
2       2       2       4
3       6       3       27
4       24      8       256
5       120     15      3125

#O[1]:   3125
```

```
#I[2]::  Do[u,x+1,x+5,Pr[u,Ex[u^2]]]

                    2
1 + x   1 + 2x + x
                   2
2 + x   4 + 4x + x
                   2
3 + x   9 + 6x + x
                    2
4 + x   16 + 8x + x
                     2
5 + x   25 + 10x + x

                     2
#O[2]:   25 + 10x + x
```

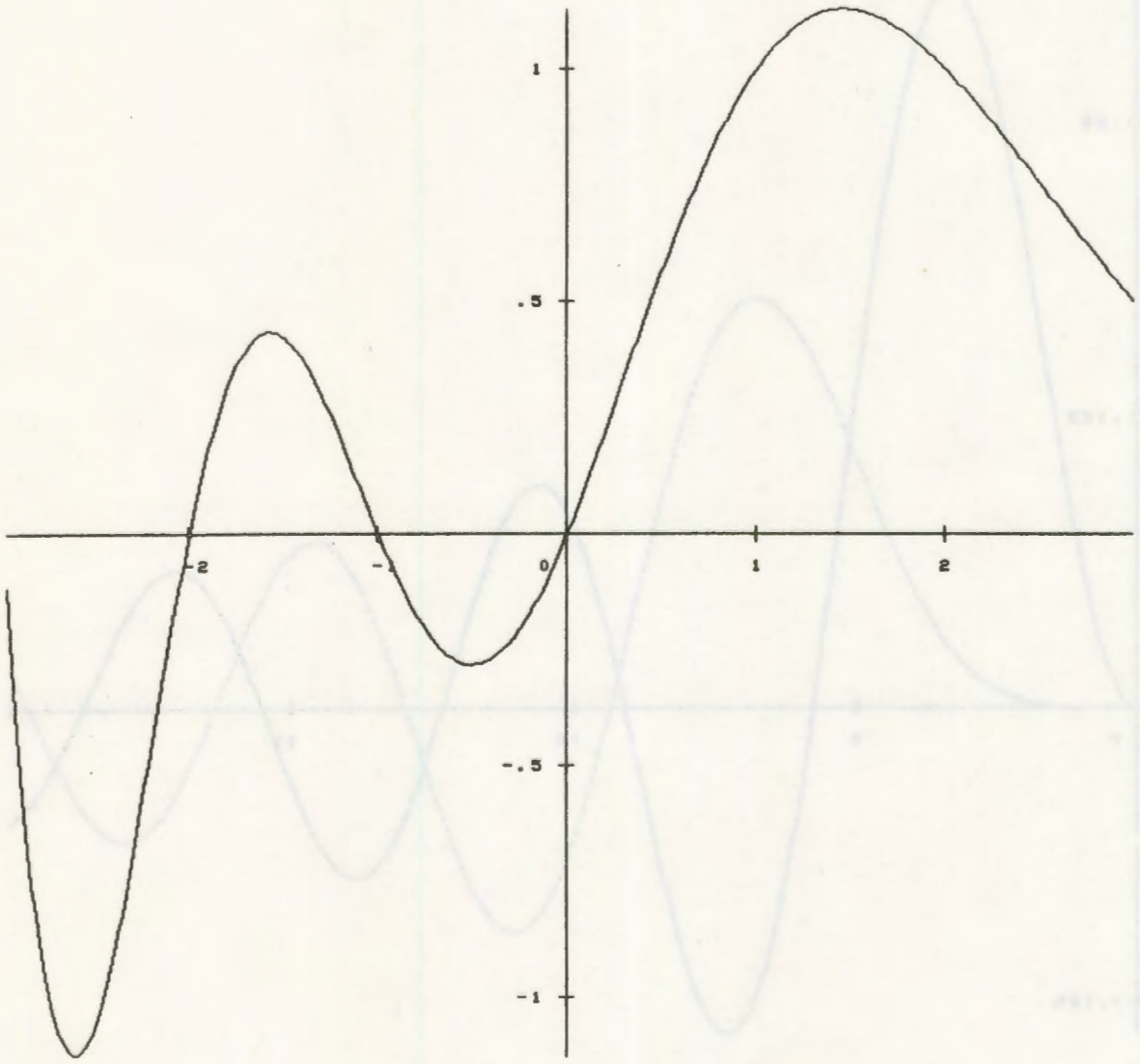
Numerical results may be presented as graphs. Graph[*expr*, *x*, *xmin*, *xmax*] plots a graph of the numerical value of *expr* with *x* between *xmin* and *xmax*.

```
#I[1]::  Graph[1/Gamma[x],x,-3,3]

#O[1]:
```



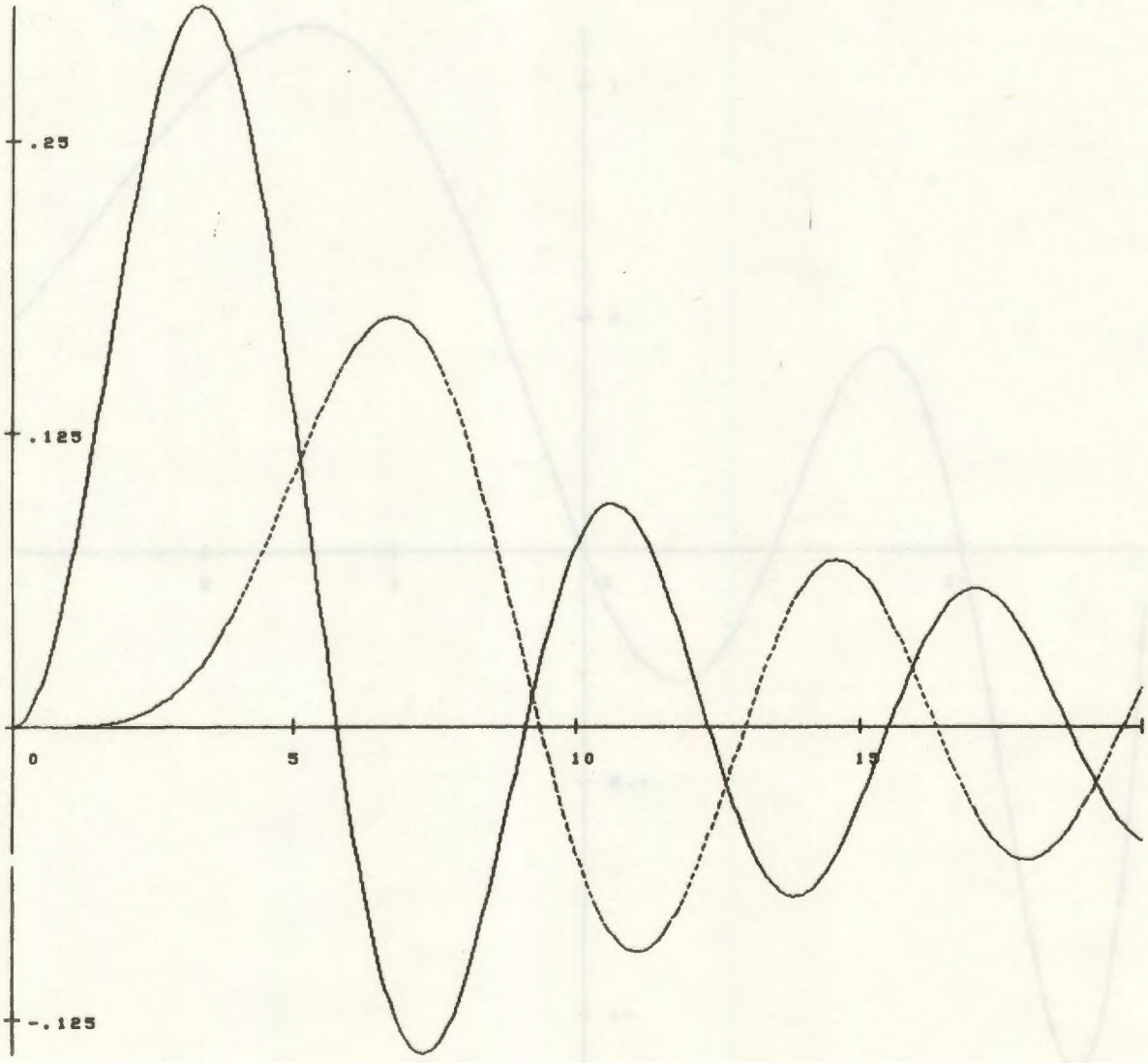
At some installations, graphics output devices may be available for which graphs are automatically generated with continuous high-resolution lines.

```
#I[1]:: Graph[1/Gamma[x],x,-3,3]
```

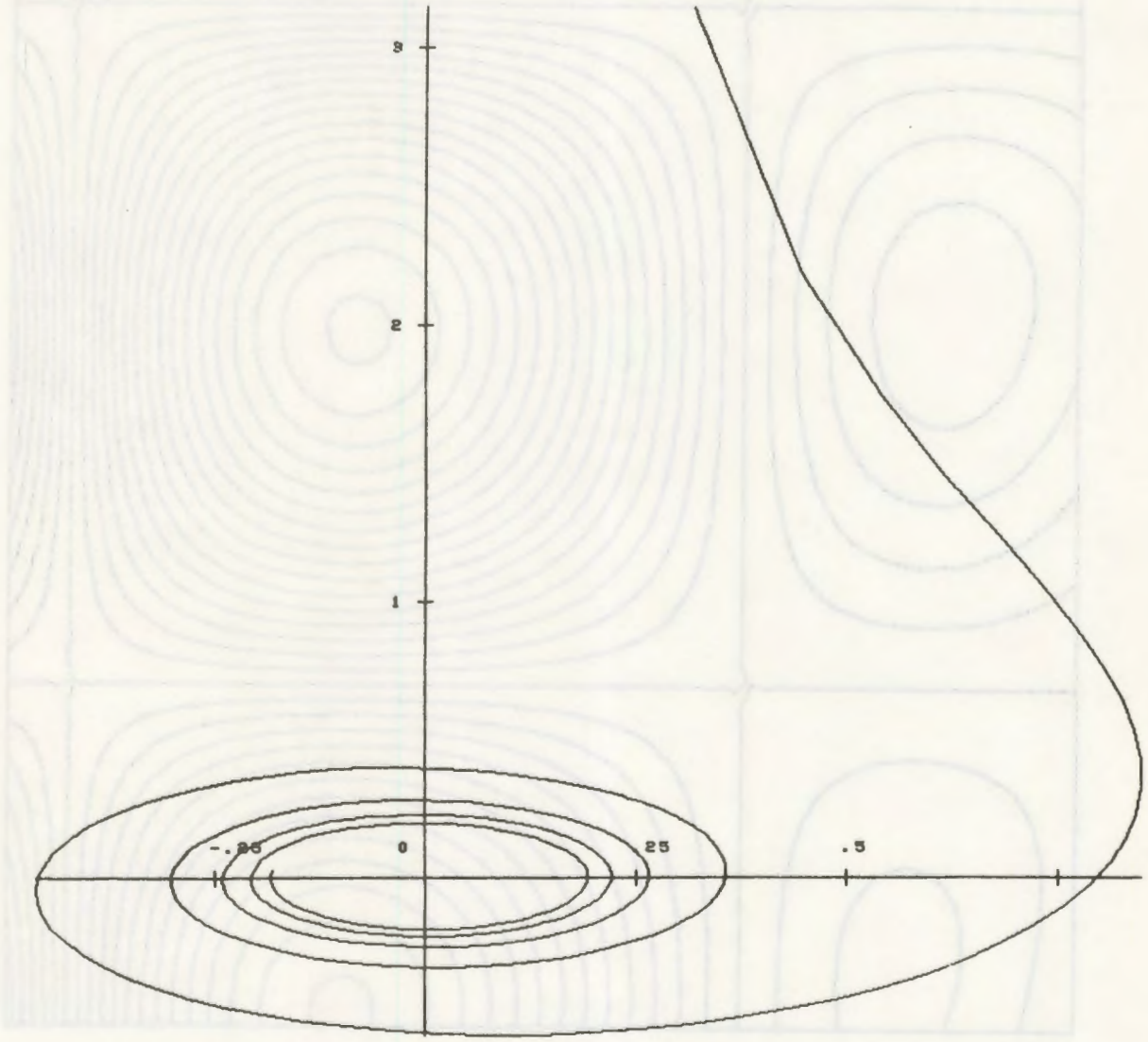


Graph[{*expr1*}, {*expr2*}, ...}, *x*, *xmin*, *xmax*] plots the curves for *expr1*, *expr2*, ... on a single set of axes.

```
#I[1]:  Graph[{Bes][2,x]},{Bes][5,x]}},x,0,20]
```

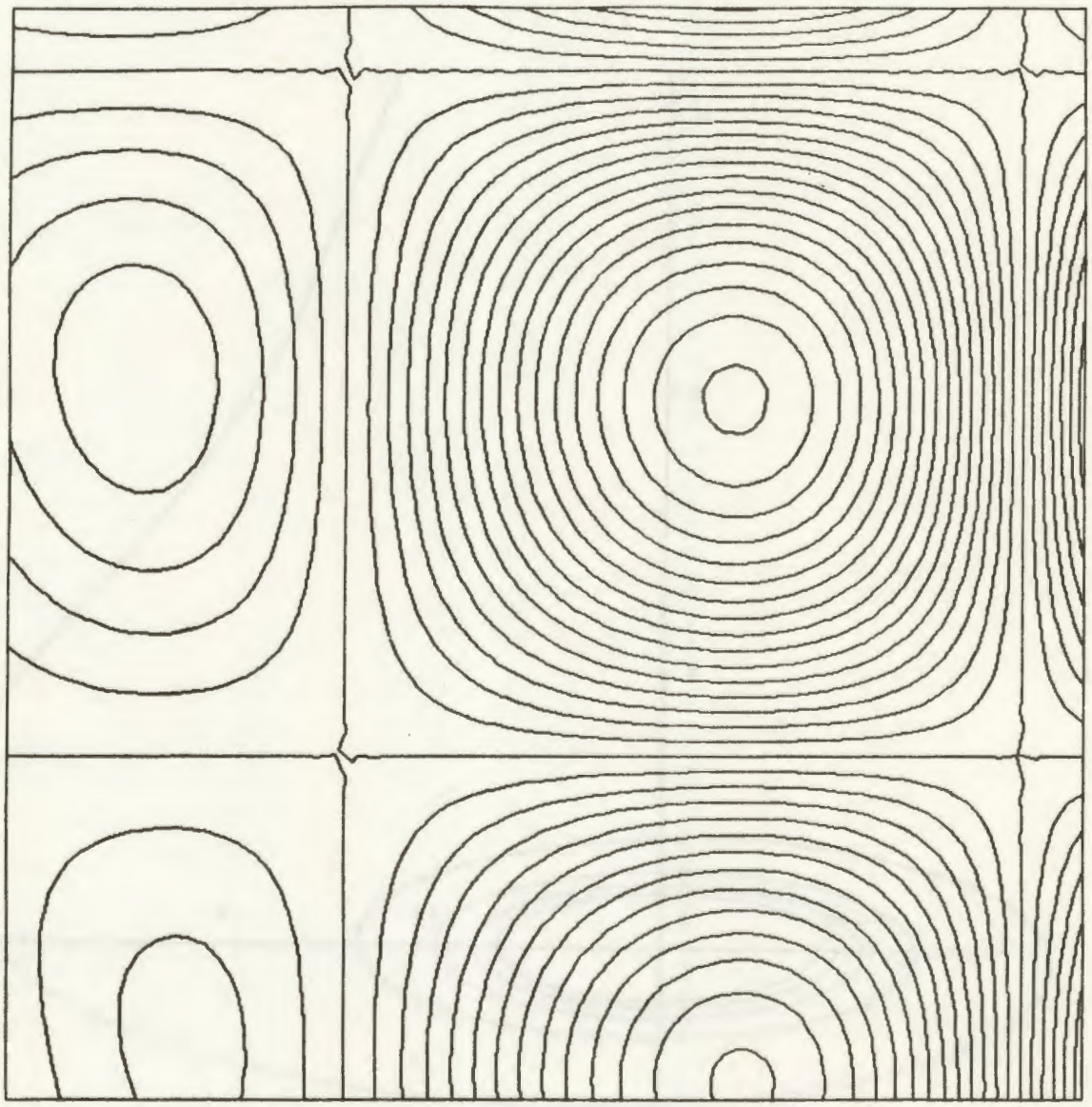


`Graph[{x, y}, u, umin, umax]` yields a parametric plot of x,y with the parameter u in the range *umin* to *umax*.

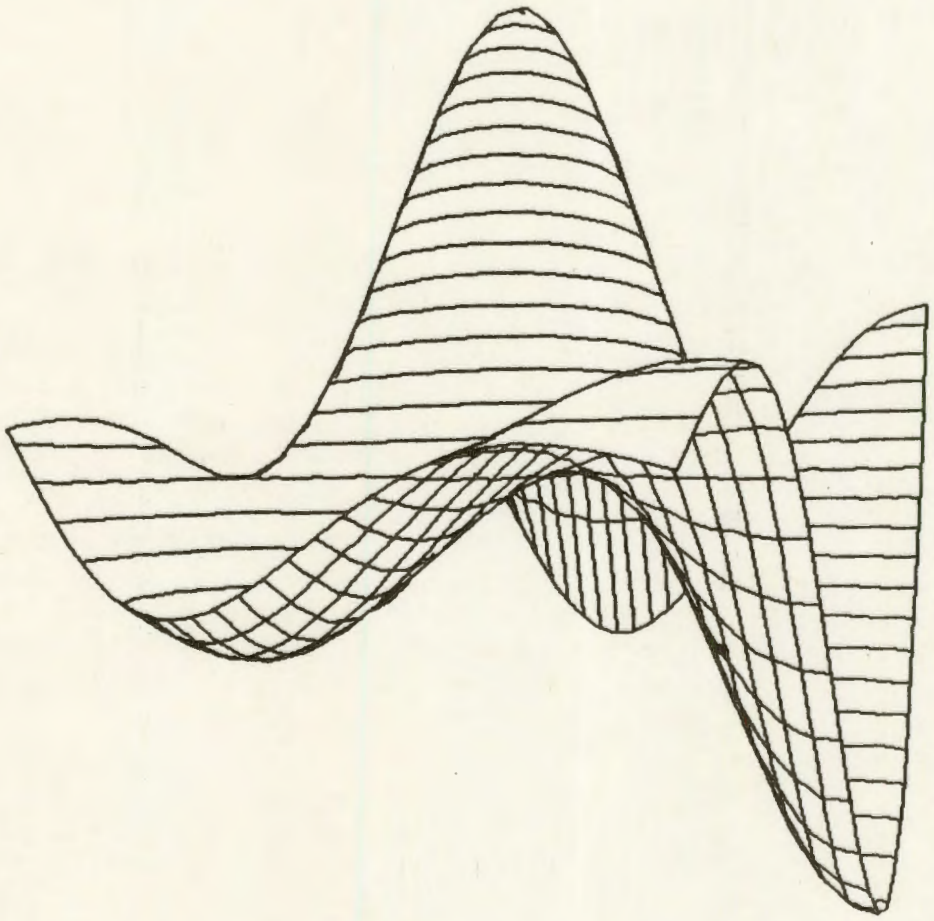


Graph[z, {x, y}, {xmin, ymin}, {xmax, ymax}] gives a contour plot of z as a function of x and y.

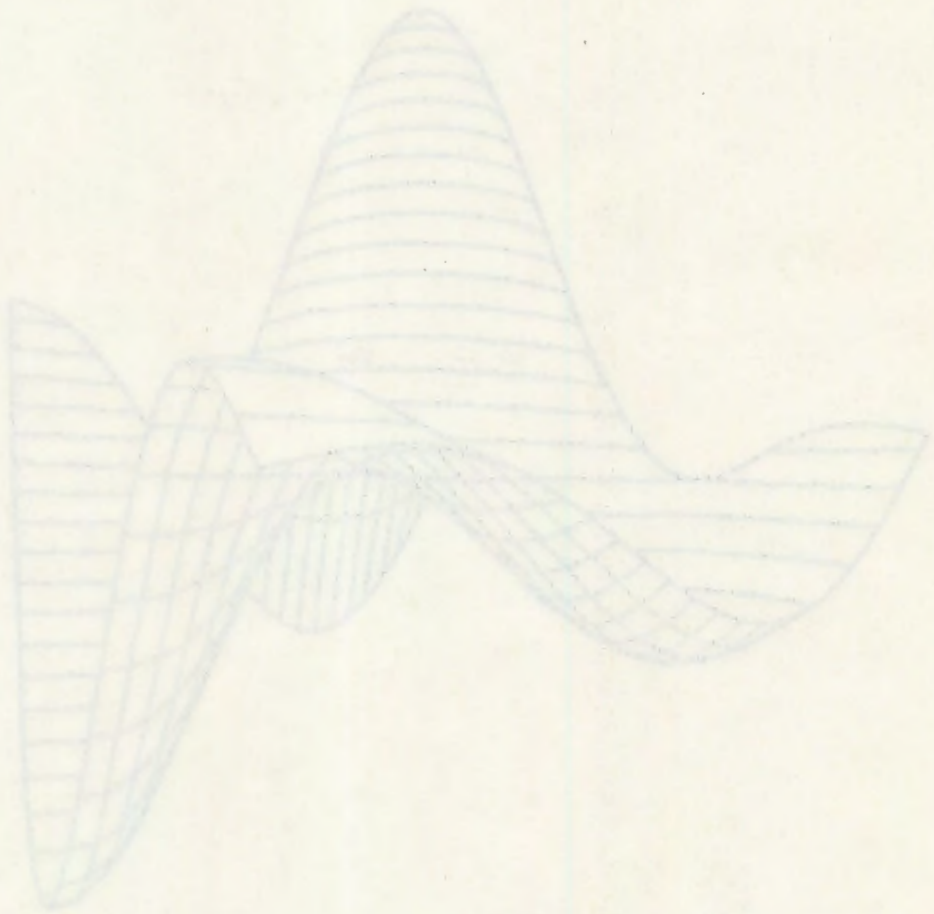
```
#I[1]:: Graph[(4x+y)/5 Cos[x/5] Cos[y/5],{0,0},{25,25}]
```



```
#I[2]::   Graph[(4x+y)/5 Cos[x/5] Cos[y/5], {x,y}, {8,0}, {25,25},, {-10,-20,0}]
```



Plots in SMP are symbolic expressions consisting of a set of point, line and surface specifications with special printing characteristics. Two, three and higher-dimensional graphs may thus be manipulated extensively, as indicated in the Reference Manual.



Further examples

Example 1

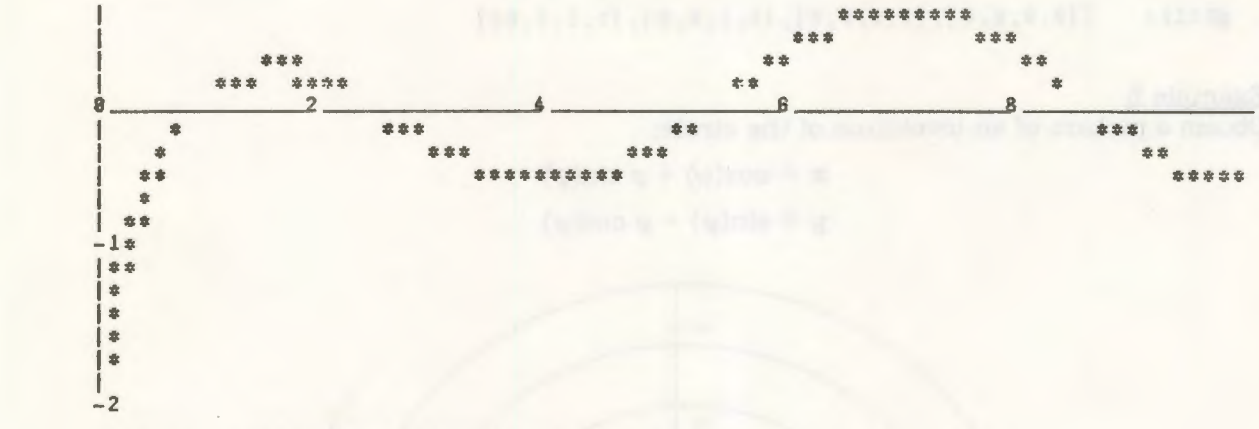
Form the 4×4 matrix $M_{ij} = \frac{1}{i+j+1}$. Find the inverse of M.

```
#I[1]::  Ar[{4,4},1/($i+$j+1)]

#O[1]:   {{1/3,1/4,1/5,1/6},{1/4,1/5,1/6,1/7},{1/5,1/6,1/7,1/8},
                {1/6,1/7,1/8,1/9}}

#I[2]::  Minv[%]

#O[2]:   {{1200,-6300,10080,-5040},{-6300,35280,-58800,30240},
                {10080,-58800,100800,-52920},{-5040,30240,-52920,28224}}
```

Example 2

Find graphically the approximate position of the first zero of $\log(x)\, J_0(x)$. Make tables of the values of the function to locate the zero more accurately.

```
#I[1]::  Graph[Log[x]BesJ[0,x],x,0,10]

#O[1]:
```



```
#I[2]::  t[$x]::N[Log[$x]BesJ[0,$x]]

#O[2]:   ' N[BesJ[0,$x] Log[$x]]

#I[3]::  Ar[{{2.4,2.8,.025}},t]

#O[3]:   {[2.4]: .002195398, [2.425]: -.00923844, [2.45]: -.02081321,
                [2.475]: -.03251591, [2.5]: -.04433361, [2.525]: -.05625344,
                [2.55]: -.06826262, [2.575]: -.08034843, [2.6]: -.09249824,
                [2.625]: -.1046995, [2.65]: -.1169398, [2.675]: -.1292067,
                [2.7]: -.1414881, [2.725]: -.1537717, [2.75]: -.1660456,
                [2.775]: -.1782979}

#I[4]::  Ar[{{2.4,2.41,.001}},t]

#O[4]:   {[2.4]: .002195398, [2.401]: .001740914, [2.402]: .001286185,
```

```
[2.403]: .0008312102, [2.404]: .000375992,
[2.405]: -7.946916*^-05, [2.406]: -.0005351726,
[2.407]: -.0009911173, [2.408]: -.001447303,
[2.409]: -.001903727, [2.41]: -.002360391}
```

Example 3

Generate the totally antisymmetric (Levi-Civita) tensor in three dimensions.

```
#I[1]:  Ar[{3,3,3},Sig]
#O[1]:  {{{0,0,0},{0,0,1},{0,-1,0}},{{0,0,-1},{0,0,0},{1,0,0}},
          {{0,1,0},{-1,0,0},{0,0,0}}}
#I[2]:  Pos[1,%]
#O[2]:  {{1,2,3},{2,3,1},{3,1,2}}
```

Example 4

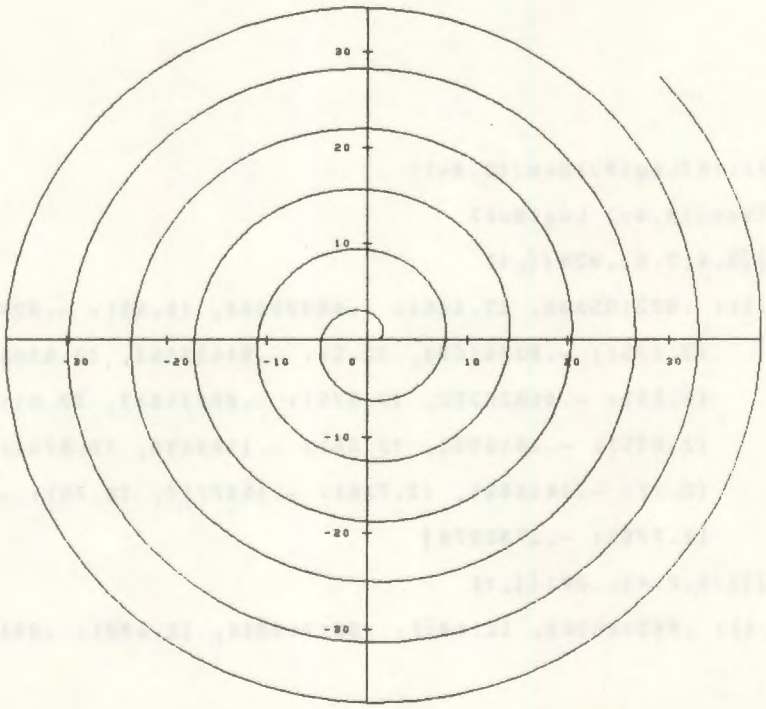
Form the unit lower-triangular 4×4 matrix.

```
#I[1]:  Ar[{4,4},Gt]
#O[1]:  {{0,0,0,0},{1,0,0,0},{1,1,0,0},{1,1,1,0}}
```

Example 5

Obtain a picture of an involution of the circle:

$$x = \cos(\varphi) + \varphi \sin(\varphi)$$
$$y = \sin(\varphi) - \varphi \cos(\varphi)$$



8. Defining new mathematical constructs

One of the most powerful features of SMP is the simplicity with which mathematical constructs may be introduced and manipulated. The SMP definition of a new construct usually coincides very closely with its precise mathematical definition.

As a first example, consider introduction of new differentiation projection `newdiff` which obeys the standard rules:

$$\frac{d}{dx}x = 1$$

$$\frac{d}{dx}y = 0 \qquad (y \neq x)$$

$$\frac{d}{dx}(u+v) = \frac{du}{dx} + \frac{dv}{dx}$$

$$\frac{d}{dx}(u\,v) = v\,\frac{du}{dx} + u\,\frac{dv}{dx}$$

where x and y are symbols.

```
#I[1]::  newdiff[$x,$x]:1

#O[1]:   1

#I[2]::  newdiff[$y_=Numbp[$y],$x]:0

#O[2]:   0

#I[3]::  newdiff[$y_=Symbp[$y],$x_=(~P[$x=$y])]:0

#O[3]:   0

#I[4]::  newdiff[$u+$$v,$x]::newdiff[$u,$x]+newdiff[$$v,$x]

#O[4]:   ' newdiff[$u,$x] + newdiff[$$v,$x]

#I[5]::  newdiff[$u $$v,$x]::$$v newdiff[$u,$x]+$u newdiff[$$v,$x]

#O[5]:   ' $$v newdiff[$u,$x] + $u newdiff[$$v,$x]

#I[6]::  newdiff

#O[6]:   {[$$v $u]: {[$x]:: $$v newdiff[$u,$x] + $u newdiff[$$v,$x]},
                [$$v + $u]: {[$x]:: newdiff[$u,$x] + newdiff[$$v,$x]},
                [$y _= Symbp[$y]]: {[$x _= (~ P[$x = $y])]: 0},
                [$y _= Numbp[$y]]: {[$x]: 0}, [$x]: {[$x]: 1}}

#I[7]::  newdiff[2x+3,x]

#O[7]:   2

#I[8]::  t:(x+a+b)(3/2x+2b)(4x+3)+x+a x

#O[8]:   x + a x + (3 + 4x) (2b + 3x/2) (a + b + x)

#I[9]::  newdiff[t,x]

#O[9]:   1 + a + (3 + 4x) (3a/2 + 7b/2 + 3x) + 4(2b + 3x/2) (a + b + x)

#I[10]::  Ex[%]

                                                                  2      2
#O[10]:  1 + 11a/2 + 21b/2 + 9x + 8a b + 12a x + 28b x + 8 b  + 18 x

#I[11]::  Ex[D[t,x]]
```

```
#0[11]:   1 + 11a/2 + 21b/2 + 9x + 8a b + 12a x + 28b x + 8 b^2 + 18 x^2

#I[12]::  Output[newdiff,defdiff];
```

The rules for differentiation stated in mathematical form above were given as assignments for newdiff in lines 1 through 5. The resulting form for newdiff was displayed in line 6. Examples of newdiff were given on lines 7 and 9, the latter being compared with the result obtained with the built-in differentiation projection D on line 11. On line 12, the definitions for newdiff were saved in the file defdiff:

```
newdiff[$x,$x] : 1
newdiff[$y_=Numbp[$y],$x] : 0
newdiff[$y_=Symbp[$y],$x_=(~P[$x = $y])] : 0
newdiff[$$v + $u,$x] :: newdiff[$u,$x] + newdiff[$$v,$x]
newdiff[$$v*$u,$x] :: $$v*newdiff[$u,$x] + $u*newdiff[$$v,$x]
```

In lines 2 and 3, Numbp[*y*] tests whether *y* is a number, and Symbp[*y*] tests whether it is a symbol. In line 3, P[*expr*] yields 1 if *expr* is 1 ("true") and 0 otherwise. Its use in this case implements the assumption that distinct symbols represent independent variables. The generic symbol $u in line 4 may represent either a composite expression or a single symbol and thus an independent variable. The alternative realized in a particular case depends on the explicit form of $u provided. To avoid evaluation before this form is specified, delayed assignments are used in lines 4 and 5.

On lines 4 and 5, $$v represents a sequence of arguments to Plus and Mult, and, through recursion, applies to derivatives of sums and products with any number of terms. As discussed in sect. 3, use of $v would give rules only for sums or products with exactly two terms.

Notice that in all assignments for newdiff, there was no explicit requirement that the variable of differentiation $x be a symbol. Such a requirement may be introduced by use of $x_=Symbp[$x] rather than $x on the left-hand side in each case.

In contrast to the rather direct definition for newdiff given above, conventional programming languages would require a definition such as:

```
#I[1]::  newdiff[$y,$x]::Swtch[Symbp[$y],If[P[$y=$x],1,0],\
          Numbp[$y],0,$y[0]=Plus,newdiff[$y[1],$x]+newdiff[$y[2],$x],\
          $y[0]=Mult,$y[2]newdiff[$y[1],$x]+$y[1]newdiff[$y[2],$x]]

#0[1]:   ' Swtch[Symbp[$y],If[P[$y = $x],1,0],Numbp[$y],0,$y[0] = Plus,

                  newdiff[$y[1],$x] + newdiff[$y[2],$x],$y[0] = Mult,

                  $y[2] newdiff[$y[1],$x] + $y[1] newdiff[$y[2],$x]]
```

Notice that this assignment cannot treat sums and products containing more than two terms.

The "switch" projection Swtch[*pred1, expr1, pred2, expr2, ...*] tests each of the "predicate expressions" *predi* in turn, yielding the *expri* associated with the first one found to be "true". If[*pred, expr1, expr2*] tests *pred*, yielding *expr1* if it is found to be "true" and *expr2* if it is found "false".

In the second example the function newdiff accepts any expression $y as a first argument, and then uses Swtch and If to test the possible forms of $y and select an appropriate case. In the first example, however, individual assignment are given directly for each possible form, so that explicit Swtch or If tests are unnecessary. The $y[0]=Plus case in the Swtch of the second example is covered by an assignment for expressions of the form $u+$$v on line 4 of the first example. The terms in the sum are identified in the first example as $u and $$v. In the second example,

they must be identified and extracted by the explicit projections `$y[1]` and `$y[2]`.

In addition to its simplicity and directness, the approach used in the first example has the virtue that further definitions may be given as additional assignments. In the second example, the entire `Switch` must be rebuilt to accomodate the additional case.

For the next example, we consider a function $f(x)$ obeying the relations:

$$(a) \qquad f(n\pi) = -1^n \qquad (n \text{ integer})$$

$$(b) \qquad f(-x) = -f(x)$$

```
#I[1]::  f[($n_?Intp[$n]) Pi]:(-1)^$n

                 $n
#O[1]:   (-1)

#I[2]::  f[3Pi]

#O[2]:   -1

#I[3]::  f[x]

#O[3]:   f[x]

#I[4]::  f[N[3Pi]]

#O[4]:   f[9.4247781

#I[5]::  f[$x_?Mod[$x,!N[Pi]]=0]:(-1)^($x/N[Pi])

                 .3183099$x
#O[5]:   (-1)

#I[6]::  f[N[3Pi]]

#O[6]:   -1

#I[7]::  f[3]

#O[7]:   f[3]

#I[8]::  f[$x_?Nc[$x]<0]:-f[-$x]

#O[8]:   -f[-$x]

#I[9]::  f[-x]+f[y]

#O[9]:   -f[x] + f[y]
```

The assignment in line 1 implements relation (a) when `Pi` appears explicitly. The assignment on line 4 treats cases such as line 3 in which the argument of `f` is a real number divisible by `Pi`. The function `Mod[`x, y`]` yields the remainder from the division of x by y.

The definition on line 5 requires the numerical value of `Pi`. `N[Pi]` is evaluated immediately on the right-hand side of the assignment. However, in the condition on the left-hand side, `N[Pi]` would usually be re-evaluated whenever the condition is tested. `!`*expr* causes *expr* to be simplified on input, and thus avoids re-evaluation in this case.

Line 8 uses the reflection formula (b) to produce a canonical form by removing any explicit overall minus sign in the argument of `f`. `Nc[`x`]` gives the overall numerical coefficient of the expression x. When no explicit overall minus sign is present, projections of `f` are left unchanged. If no condition had been given in the assignment on line 8, `f[x]` would have been replaced by `-f[-x]` which would in turn have been replaced

by f[x] ad infinitum.

We now consider manipulation of permutations (see also the external file XPerm). Permutations are represented as usual by a list of integers giving the final positions of each element. The projection comp[*perm1*,*perm2*] yields the permutation obtained by applying first *perm1* then *perm2* (composition).

```
#I[1]:: comp[$p1,$p2]::Ar[Len[$p1],$p2[$p1[$1]]]
#O[1]:  ' Ar[Len[$p1],$p2[$p1[$1]]]
#I[2]:: comp[{3,2,1},{1,3,2}]
#O[2]:  {2,3,1}
#I[3]:: comp[{2,3,1},{3,2,1},{1,3,2}]
#O[3]:  comp[{2,3,1},{3,2,1},{1,3,2}]
#I[4]:: comp[comp[{2,3,1},{3,2,1}],{1,3,2}]
#O[4]:  {3,1,2}
#I[5]:: f[f[a,b],c]
#O[5]:  f[f[a,b],c]
#I[6]:: comp_:Flat
#O[6]:  Flat
#I[7]:: _comp
#O[7]:  {[Flat]: 1}
#I[8]:: _f[Flat]:1
#O[8]:  1
#I[9]:: f[f[a,b],c]
#O[9]:  f[a,b,c]
#I[10]:  comp[{2,3,1},{3,2,1},{1,3,2}]
#O[10]:  {3,1,2}
#I[11]:: Output[comp,perms]
```

In line 1, Len[*expr*] gives the "length" of *expr*: the number of elements in a list, or the number of filters in a projection. On line 2, $1 in the definition of comp runs over the values 1, 2 and 3. The inner projection in the template yields the position of each element after the first permutation; the outer projection then gives the final position after the element has been subjected to the second permutation.

comp is defined on line 1 as a function of two arguments only. The expression on line 3 is therefore not simplified. However, the mathematical operation of composition is associative, allowing composition of three permutations to be performed as on line 4. Notice that the arguments of the function comp on line 4 are simplified before the function itself.

Line 5 assigns comp to have the property Flat, and defines its projections to act as associative n-ary functions. This property is used in evaluating the composition of three permutations on line 8.

Line 6 defines f to have the property Flat. The nested projection on line 4 is then reduced to a "flattened" canonical form on line 8. The built-in functions Plus, Mult

and Dot also carry the property Flat, so that (a+b)+c is reduced to a+b+c.

Any properties or attributes assigned to a symbol *s* are given in the list *_s*. When a symbol carries a property *prop*, a nonzero entry *_s*[*prop*] appears in the list *_s*. Line 7 shows that comp carries the property Flat. The form *s_:prop* used on line 6 is a short notation for *_s*[*prop*]:1. Projections of *_s* may be assigned or removed just like projections of a symbol.

Properties are used to specify many characteristics of projections. Projections carrying the property Comm are treated as commutative functions, and their arguments are sorted into a canonical order. The property Reor may be used to specify more complicated reordering symmetries for the filters of a projection (as required for indicial tensors). For example, *_s*[Reor]:Asym defines *s* to be totally antisymmetric with respect to reordering of its arguments.

When a projection carrying the property Ldist has lists as its filters, the projection is applied separately to each element of the lists, yielding a list of the results. All common mathematical functions have this property, as illustrated in sect. 2.

a_b assigns the symbol *a* to have the "type" *b*, and to share any properties of *b*. Such assignments are often required to distinguish objects of different classes when using external files.

It is often convenient to represent several related functions as projections from the same object, distinguishing them by the number of arguments given. Thus both the ordinary gamma function $\Gamma(x)$ and the incomplete gamma function $\Gamma(x,a)$ are represented as projections of Gamma. Projections with fewer filters usually correspond to functions obtained when omitted arguments take on simple "default" values. The ordinary gamma function is obtained from $\Gamma(x,a)$ when $a=0$. Assignment of the property Tier to a symbol *s* indicates that projections of *s* with different numbers of filters represent different functions. Without Tier, an additional filter in a projection represents extraction of a deeper subpart of the same function or object. f[1,2] is then equivalent to f[1][2]. When f has the property Tier, however, f[1,2] and f[1][2] are distinguished. Most built-in functions allow several optional arguments used to specify their action more precisely; they therefore carry the property Tier.

Composition may be considered as the analogue for permutations of the Dot operation. Lists used to represent permutations may be distinguished from other lists by enclosing them in a projection as Perm[*list*]. No value is assigned to Perm; it is merely used to identify permutation lists.

```
#I[1]:: Perm[$!1].Perm[$!2]::Perm[comp[$!1,$!2]]

#O[1]:  ' Perm[comp[$!1,$!2]]

#I[2]:: Perm[{2,3,1}].Perm[{3,2,1}].Perm[{1,3,2}]

#O[2]:  Perm[comp[comp[{2,3,1},{3,2,1}],{1,3,2}]]

#I[3]:: <perms

#I[4]:: @2

#O[4]:  Perm[{3,1,2}]

#I[5]:: _Perm[Pr][$x]::Fmt[{{0,0},{1,1},{1,-1},{2,0}},"[",$x,Ar[Len[$x]],"]"]

#O[5]:  ' $x
       [           ]
       Ar[Len[$x]]
```

```
#I[6]:: @4

          {3,1,2}
#O[6]:* [       ]
          {1,2,3}

#I[7]:: Lpr[%]

Perm[{3,1,2}]

#I[8]:: Perm[{2,1,3,4}].Perm[{1,4,2,3}]

          {4,1,2,3}
#O[8]:* [         ]
          {1,2,3,4}
```

Line 5 defines a special output form for `Perm`. When a suitable property is defined, projections from *f* are printed in the form assigned for the corresponding projections of *_f*[Pr]. `Fmt[`*spec*, *expr1*, *expr2*, ...`]` prints with the expressions *expr1*, *expr2*, ... in relative positions defined by the lists of horizontal and vertical offsets in *spec*. `Lpr` prints expressions without using special output forms, as on line 7.

If permutations are represented as ordinary lists, composition may be represented by a special input form.

```
#I[1]:: Sxset["<>",comp,4]

#O[1]: comp

#I[2]:: {2,3,1}<>{3,2,1}<>{1,3,2}

#O[2]: comp[{2,3,1},comp[{3,2,1},{1,3,2}]]

#I[3]:: <perms

#I[4]:: @2

#O[4]: {3,1,2}

#I[5]:: ^D
```

Line 1 defines `<>` to be an operator of class 4 (see [2.10]) representing `comp`.

External files in the SMP library serve as further examples. See particularly the files `XLap` (Laplace transforms), `XGr` (graph theory) and `XFrac` (fractions).

9. Programming

Assignments may be considered to define programs. In most of the cases discussed above, the "programs" have involved only mathematical functions or single SMP commands. In some cases, it is necessary to use a sequence of commands to manipulate or transform an expression.

In the simplest case, the sequence is formed by nesting commands. The nested projection *f*[*g*[*x*]] performs the command *g* on the expression *x*, and then performs *f* on the result. This method is adequate when the output from each step is required only as the input to the next step.

A procedure consists of a sequence of independent commands. The commands are given in order, and separated by semicolons. Procedures are necessary for manipulations which require intermediate expressions to be used repeatedly. Such expressions may be assigned as the values of temporary "local" variables in a procedure. Lc|[*x1*, *x2*, ..] defines *x1*, *x2*,... to be local variables whose original values are restored upon completion of the procedure.

```
#I[1]::  Ex[(1+x+a)^3]

                                  2      2      2    3      2    3
#O[1]:   1 + 3a + 3x + 6a x + 3a x  + 3 a  x + 3 a  + a  + 3 x  + x

#I[2]::  Coef[x,%]

                        2
#O[2]:   3 + 6a + 3 a

#I[3]::  Fac[%]

                    2
#O[3]:   3 (1 + a)

#I[4]::  Fac[Coef[x,Ex[(1+x+a)^3]]]

                    2
#O[4]:   3 (1 + a)

#I[5]::  t:Ex[(1+x+a)^3];t:Coef[x,t];t:Fac[t]

                    2
#O[5]:   3 (1 + a)

#I[6]::  t

                    2
#O[6]:   3 (1 + a)

#I[7]::  t:Ex[(1+x+a)^3];t:Coef[x,t]+Coef[x^2,t];Fac[t]

#O[7]:   3(1 + a) (2 + a)

#I[8]::  t

                        2
#O[8]:   6 + 9a + 3 a

#I[9]::  Lc|[t];t:Ex[(1+x+a)^3];t:Coef[x^2,t]/Coef[x,t]

           3 + 3a
#O[9]:   ----------
                  2
         3 + 6a + 3 a
```

```
#I[10]:: t

#O[10]:   6 + 9a + 3 a^2
```

In lines 1, 2 and 3, a succession of commands were used on a single expression. On line 4, the commands were nested together. In line 5, the commands were performed sequentially in a procedure, using t to carry intermediate results. Note that even after the procedure has been completed, t retained the last intermediate value assigned to it in the procedure.

In the procedure on line 7, the intermediate result from Ex was used twice; to perform the same operation using nested commands would have required two separate invocations of Ex. On line 9, a similar procedure was executed, but with t defined as a local variable. The original value of t was then restored on exit from the procedure, as shown on line 10.

Variables used only in intermediate stages of a procedure should be defined as local to avoid potential conflicts with existing or future variables of the same name used outside the procedure. It is conventional for the names of local variables to start with %.

If a procedure contains conditional statements (If or Switch), the result of the procedure may be returned before the final expression in the procedure is executed. When Ret[*expr*] is encountered in a procedure, the procedure is terminated, and the result *expr* is returned.

A procedure may contain invocations of further procedures. In a set of nested procedures, a variable defined as local in an outer procedure is shared with all inner procedures in which it is not explicitly defined as local. Ret[*expr*, *n*] returns the value *expr* through *n* nested procedures.

It is often necessary to perform an operation repetitively a fixed number of times, or until a particular form is reached. Many SMP functions may be specified to apply an operation repeatedly. For example, S[*expr*, *a*->*b*, *n*] performs substitutions in *expr* using the replacement *a*->*b* successively until no further change occurs, or at most *n* times.

Loop[*cond*, *expr*] simplifies *expr* repetitively until the condition *cond* ceases to be "true".

```
#I[1]:: S[x,$x->f[$x]]

#O[1]:  f[x]

#I[2]:: S[x,$x->f[$x],4]

#O[2]:  f[f[f[f[x]]]]

#I[3]:: t:x;n:0;Loop[n<4,t:f[t];Inc[n]];t

#O[3]:  f[f[f[f[x]]]]

#I[4]:: For[t:x;n:0,n<4,Inc[n],t:f[t]]

#O[4]:  f[f[f[f[x]]]]

#I[5]:: t:x;Do[i,4,t:f[t]]

#O[5]:  f[f[f[f[x]]]]
```

Inc[*x*] increments the value of *x* by 1.

A natural mathematical definition of the Fibonacci numbers is

$$f(n) = f(n-1) + f(n-2) \quad \text{(a)}$$

$$f(1) = f(2) = 1 \quad \text{(b)}$$

The evaluation of Fibonacci numbers using this definition requires nested "recursive" application of the function *f*. *f* is used repeatedly until the end conditions (b) are encountered.

```
#I[1]::  f[$n]:f[$n-1]+f[$n-2]
#0[1]:   f[-2 + $n] + f[-1 + $n]
#I[2]::  f[1]:f[2]:1
#0[2]:   1
#I[3]::  f[3]
#0[3]:   2
#I[4]::  f[10]
#0[4]:   55
#I[5]::  g[$n]::(g[$n]:g[$n-1]+g[$n-2])
#0[5]:   g[$n] : g[$n - 1] + g[$n - 2]
#I[6]::  g[1]:g[2]:1
#0[6]:   1
#I[7]::  f
#0[7]:   {[1]: 1, [2]: 1, [$n]: f[-2 + $n] + f[-1 + $n]}
#I[8]::  g
#0[8]:   {[1]: 1, [2]: 1, [$n]:: g[$n] : g[$n - 1] + g[$n - 2]}
#I[9]::  g[5]
#0[9]:   5
#I[10]::  g
#0[10]:   {[5]: 5, [4]: 3, [3]: 2, [1]: 1, [2]: 1,
                [$n]:: g[$n] : g[$n - 1] + g[$n - 2]}
```

The function g on line 6 is defined to be a program which recursively computes values of g and then stores the result ("dynamic programming"). After the evaluation of g[5] the list g contains the explicit values for all the cases computed, as shown on line 10.

Many common definitions implicitly use recursion. For example, Log[$x $$x]:Log[$x]+Log[$$x] is applied recursively to Log[a b c] to yield first Log[b c] + Log[a] and then Log[a] + Log[b] + Log[c].

Recursive assignments are used until end conditions are encountered. Use of the definition

$$f(n) = f(n+2) - f(n+1)$$

would miss the end conditions and cause the attempted evaluation of say *f(4)* to continue forever. (In practice, SMP eventually suspends apparently fruitless recursive

evaluations.) S allows recursive replacements to be applied a fixed number of times, instead of continuing until end conditions are reached.

In investigating or verifying the operation of a definition or program, it is often convenient to assign *f_*:Trace so that each projection from *f* is printed before evaluation.

```
#I[1]::  Log[$x $$x]:Log[$x]+Log[$$x]

#0[1]:   Log[$$x] + Log[$x]

#I[2]::  Log_:Trace

#0[2]:   Trace

#I[3]::  Log[a b c d e]

Log[a b c d e]

Log[b c d e]

Log[a]

Log[b]

Log[c d e]

Log[c]

Log[d e]

Log[d]

Log[e]

#0[3]:   Log[a] + Log[b] + Log[c] + Log[d] + Log[e]
```

Many programs require arguments of particular types. For example, the Fibonacci program above allows only integer arguments. Such type restrictions are enforced by placing conditions on the input variables: `f[$n_=Natp[$n]]:f[$n-1]+f[$n-2]` defines the reduction of `f[$n]` only when `$n` is a positive integer.

Expressions are usually simplified whenever they appear. However, in assignments performed with `::`, the expression on the right hand side is not immediately simplified. It is simplified each time the assignment is used, after generic symbols have been replaced by the specific values required. Use of `::` in the definition of the Fibonacci function `g[$n]` above causes the assignment on the right-hand side to be performed separately for each value of `$n`. Whenever simplification of an expression or program on the right-hand side of an assignment followed by replacements for generic symbols would yield a different result from simplification after the replacement of particular values for generic symbols, a delayed assignment `::` should be used. This case is realized when structural operations are performed on the expressions represented by the generic symbols, or when an assignment occurs.

Only values of arguments are usually passed to functions. However, when projections or assignments of an argument are required, it is necessary to pass the argument "by name" in an unsimplified form. Some system-defined functions automatically maintain their arguments in unsimplified form. For example, in `a:1`, `a` is maintained unsimplified. In `!a:1`, `a` is simplified before assignment. If the value of `a` were `b`, then in this case, `b` rather than `a` would be assigned the value `1`. Similarly, functions such as `Plus` to be used as templates are passed in an unevaluated form.

The simplification of any expression may be prevented by explicit use of `'`. *'expr* is an expression equivalent to *expr*, but maintained in an unsimplified form. Structural

operations may be performed on the "frozen" or "held" expression *'expr* just as on *expr*, but the results are left unsimplified until explicitly "released" by Rel[*expr*].

External files in the SMP library serve as further examples. See particularly XHorn (Horner representation of a polynomial), XLItp (Lagrangian interpolation) and XYoung (Young graphs for group theory).

10. Epilogue

The reader of this primer should by now have a gained a basic working knowledge of SMP. Experience with an actual SMP system is crucial in consolidating and extending this knowledge.

Casual reading of the Reference Manual and the external files in the SMP Library will indicate further capabilities and facilities of SMP.

The keyword index to the Reference Manual, and the topic directory of the SMP Library may be used to locate a detailed description of facilities relevant to a specific calculation or application. In some cases, a single built-in function, or an existing library function, may perform the complete calculation required. In all but the most mundane cases, however, several functions will be required. The first time a complicated operation is performed, it may be convenient to carry out each step on a separate line, inspecting and checking the intermediate results. However, once the procedure for the operation is clear, a single function should be devised to implement it. It is almost always worthwhile to give the most general definition feasible, and to save it in an external file - later changes of parameters or interpretation may then be investigated without devising necessary definitions again. It is usually valuable to include a short commentary with each external file describing the operation or nature of the definitions given.

An operation or construct requires precise formulation to be defined in SMP. Mathematical definitions may very often be used directly in SMP. Definitions of procedures are often more difficult and lengthy to implement.

The appearance of global variables in definitions should usually be avoided - when the definition is used again, the values of the variables may have been changed.

Separate functions should be defined for each part of a complicated operation, and controlled by an overall dispatching function.

When no complicated mathematical formulae are involved, a rough rule is that individual SMP definitions should usually be less than two or three lines long. If they are longer, an alternative formulation is probably desirable. The shorter its definition, the easier a function is to test, verify and understand.

Once a definition has been devised, it should be checked carefully. Incorrect definitions may often give plausible results. Simple cases with known results should be tried to verify the defintion and its implementation.

It is common for a definition or function not to behave in the way intended or expected. To understand discrepancies, it is often valuable to trace the operations by hand in a simple case.

SMP does what it is defined or programmed to do exactly and pedantically. It can determine what the user intends only through the definitions given.

If an unexpected result is obtained, it is always possible that SMP may be in error. Such errors should be considered as a very last resort in tracing the source of a difficulty. The definitions given should first be examined and investigated in extreme detail.

When performing a calculation, needless complication should be avoided. A more complicated calculation will take longer to perform, and the result will often be lengthy and difficult to assimilate. There is nevertheless no limit in principle to the size of calculations and expressions handled by SMP. (Some installations may however impose limits.) SMP was in fact designed to perform calculations of extreme complexity.

SMP is a very concise language. Very complicated operations can be specified on a single input line, so that the processing of the line may require a substantial amount

of time. The internal operation of SMP is nevertheless extremely efficient.

Most scientific investigations involve three phases: development of conceptual framework, mundane calculation and interpretation and exposition of results. It is in the second of these phases that SMP may be used to greatest advantage. It should enable calculations to be performed quickly and without error, and make accessible problems of immense complexity. Judging from the authors' experience, it will prove an invaluable aid in almost any scientific investigation.

Appendix Some common difficulties

Some of the commoner difficulties encountered by SMP users are collected here.

1. A "system-defined" projection is returned unsimplified when simplification is expected.

   All system-defined projections begin with capital letters. `Ex[(x+1)(x-1)]` will work; `ex[(x+1)(x-1)]` will not.

   Check the spelling of names for system-defined objects.

   Filters for system-defined projections ("arguments" to "functions") are enclosed in square brackets, not parentheses.

2. An input expression is not printed as expected.

   `1/2x` means `(1/2)*x` not `1/(2x)`.

   `x^2a` means `(x^2)*a` not `x^(2a)`.

   `a/b/c` means `a/(b*c)` not `(a*c)/b`.

   Filters for projections are given in square brackets: `f(x)` means `f*x` not `f[x]`.

   `ab` is a single symbol, not the product of symbols `a b` or `a*b`.

3. SMP runs for a long time without printing a result.

   Try to stop processing using ⊂*quit interrupt*⊃ or ⊂*break interrupt*⊃

   Estimate how complicated the calculation requested should have been. Try an analogous but simpler calculation.

   A non-terminating recursive assignment may have been performed. Investigate relevant assignments to see whether a special case of the left-hand side may appear on the right-hand side. If so, insert suitable conditions to prevent infinite recursion. If the assignment itself leads to infinite recursion, use `::` rather than `:`.

   A non-terminating sequence of substitutions may have been specified through `S` or `Si`. In `S` projections, the last filter may be a repetition count: a level specification must be given as second-to-last filter.

   A non-terminating `For` or `Do` loop may have been invoked; usually these yield a processing impasse after 1000 iterations. A non-terminating recursive projection definition may have been made and used; usually the simplifier suspends processing after 100 nested recursive projections.

4. A strange result involving unexpected objects is obtained.

   Some of the objects used in the current expression were probably assigned values earlier in the job. The value of *expr* is removed by *expr:*

   Subexpressions may be simplified when not intended: *'expr* is equivalent to *expr* but remains in an unsimplified form.

   See also 5.

5. An input ambiguity is reported on an input line with apparently correct syntax.

   Syntax modifications may have been performed. These are removed by `Sxset[]`

   Multiplication must be input as an explicit `*` when ⊂*space*⊃ would be ambiguous: `a <b` means `Gt[b,a]` not `a*<b`.

6. An assignment is not used when expected, or a pattern does not match as expected.

   `f[x]:x^2` assigns the value of the projection `f[x]` with the particular filter `x` to be `x^2`, and has no consequences for `f[y]`. `f[$x]:$x^2` assigns the

projection of f with an arbitrary expression $x to be $x^2.

x^$n:*expr* does not assign a value for x. Similarly, $x+x:*expr*, $x x:*expr* and x/$x:*expr* do not assign values for x.

f[$x+$y,$x-$y]:*expr* does not assign a value for f[4,1].

f[$x+$y]:*expr* assigns a value for f[x+y] but not for f[x+y+z]; f[$x+$$y] assigns a value for any number of summands.

f[1-$x,$x] does not match f[a,1-a] since 1-$x and 1-1+a do not match, because Plus is a projection with property Flat; f[1-$$x,$$x] would match.

f[$$x_?(Len[$$x]>3)]:*expr* causes f[a,b,c,d] to test Len[a,b,c,d]; f[$$x_?(Len[List[$$x]]>3)] is a suitable test for projections of f with more than three filters.

Properties such as Flat, Tier and Comm should be assigned to projections before any values are assigned to the projections.

Indices in lists are not simplified or modified except when they are first assigned.

f[$x]:=$x^2 specifies a syntax modification, not an assignment.

7. The value obtained from a projection is not the one which was supposed to have been assigned, or which appears to be present.

   Positions of parts are determined by the simplified form of an expressions. Particularly in commutative functions, positions of parts may change when assignments for other parts are made.

   f[$x]:S[$x,x->1] makes the assignment f[$x]:$x; f[$x]::S[$x,x->1] is a delayed assignment which defines the S to be simplified again for each specific form of $x provided.

   If a projection unexpectedly yields a list, the projection requested probably had a different number of filters from the one assigned. Alternatively, the projection should have been assigned property Tier.

   a:b;c means (a:b);c not a:(b;c).

   Expressions may be printed in a form which differs from their internal form. Such expressions are indicated by a * at the beginning of output. Direct forms in which labelling of parts are manifest is obtained by Lpr.

8. A replacement in an S projection did not behave as expected.

   f[$x]->S[$x,x->1] becomes f[$x]->$x; f[$x]-->S[$x,x->1] is a delayed replacement which causes the S projection to be simplified again each time the replacement is used.

   Unless the *rpt* filter is specified, S does not perform further replacements on expressions obtained as a result of replacements.

9. Run or < fail to find the specified files.

   All file names must be symbols: smp.in must therefore be written as "smp.in".

10. A template was not used as expected.

    Generic symbols in a template are ordered lexically (with $$x before $%x before $x) and associated with expressions in that order. In a sequence of nested template applications, the outermost is usually performed first.

    Ap[f[x],{1}] yields Proj[f[x],{1}]; Ap[`f[x],{1}] yields f[x].

    Templates are not simplified before use, except when given as !*temp*.

Glossary

Terminology or usage specific to SMP is indicated by *.
References are to sections in the summary/reference manual.

array Example of a contiguous list.

ASCII "American Standard Code for Information Interchange"; a standard set of numerical codes for characters.

* ***assignment*** The association of an attribute (usually value) to an expression [3.2]; definition; assertion; assumption; declaration.

asynchronous Not occuring in a fixed time sequence.

binary file File containing direct binary machine code instructions.

* ***block*** Unit of computer memory (usually 16 bytes); space required to store one symbol.

break interrupt Real time interrupt used to suspend processing and enter interactive subsidiary procedure [1.4].

buffer Temporary text storage region.

bug An error in a program or procedure.

bus error A class of program errors involving illegal memory fetches.

byte Usually 8 binary bits of computer memory.

C The programming language in which SMP was written.

* ***chameleonic expression*** An expression containing chameleonic symbols [2.8].

* ***chameleonic symbol*** Symbol whose name changes whenever it is evaluated [2.8].

character string See string.

* ***click*** Unit of computing time in SMP, arranged to be approximately independent of the actual of a particular computing unit.

comment Input text used for descriptive purposes and not processed [2.9].

compiler Program for transforming code in a high-level language into machine code.

computing unit Central processing unit; processor.

* ***contiguous list*** A list whose indices are successive integers starting at 1 [2.4].

control character Character typed at a terminal while "CTRL" key is depressed.

core Main computer memory.

* ***criterion*** Template [2.7] applied to expressions to determine whether an operation should be preformed on them.

data space Portion of computer memory used to store expressions manipulated in a job.

* ***deassignment*** The removal of an attribute (usually value) from an expression [3.2].

default Attribute assumed if not explicitly specified; "standard".

* ***delayed value*** A value maintained in an unsimplified form, and simplified afresh when used [3.2].

* ***depth*** The maximum number of filters necessary to specify any part of an expression [2.5].

* ***domain*** A set of parts in an expression [2.5].

* ***editor*** Interactive facility for performing textual modifications on input lines [1.7].

* ***entry*** Expression appearing in a list [2.4].

* ***evaluation*** Process of replacing symbols and projections by values assigned to them [3.7].

exception Processing error detected by CPU.

* ***expression*** Combination of symbols, projections and lists manipulated by SMP [2.5].

* ***extension*** See type extension.

external file A file containing SMP expressions to be input when required [1.3].

external program A program to be run in the monitor taking input from SMP and passing input back to SMP [1.10,A.2].

file A named area (usually on a magnetic disk) used to provide long-term storage of text or data.

* ***filter*** An expression appearing in a projection [2.3] (e.g. x or y in f[x,y]).

* ***full list*** List forming a "rectangular array" [7.6].

function Example of a projection.

garbage collection Process by which areas of computer memory not required for further processing are reclaimed.

* ***generic expression*** An expression containing generic symbols; pattern [2.6].

* ***generic symbol*** A symbol representing any one of a possibly infinite class of expressions [2.6]; dummy symbol; pattern variable.

global object An object whose value or attributes can affect expressions which do not directly contain it [1.2].

* ***held form*** A form of an expression in which simplification is prevented (e.g. 'x) [3.5].

* ***immediate value*** A value simplified when assigned, and maintained in a simplified form [3.2].

* ***impasse*** See processing impasse.

* ***index*** An expression labelling an entry in a list [2.4].

* ***interactive procedure*** Procedure in which results from one segment are output before the next segment is input.

* ***interpreter*** Program which executes procedures in a high-level language interactively.

interrupt See real-time interrupt.

iteration Successive simplification or evaluation of a set of expressions [6.2].

job One execution of a program, from initiation to termination; process; task.

* ***length*** The number of entries in a list or number of filters in a projection [7.4].

* ***level*** A set of parts of an expression [2.5].

* ***list*** Indexed and ordered set of expressions (e.g. {x,y} or {[a1]:x,[a2]:y}) [2.4]; vector; structure or record; lambda function; rule.

local variables Symbols used within a procedure, and restored on exit from the procedure [6.3].

logging in Signing on; connecting to computer.

* ***match*** A pattern ***p2*** matches ***p1*** if it represents a superset of the expressions represented by ***p1*** [2.6].

memory fault A class of program errors involving use of illegal memory addresses.

modem A device for acoustic or electronic connection of computer equipment to a telephone.

monitor Shell; command line interpreter; operating system.

* ***multi-generic symbol*** Generic symbol representing a sequence of expressions [2.6].

newline Line termination character; carriage return; line feed; enter.

* ***null projection*** Special projection representing a sequence of expressions [2.3].

paging The ability to read into main memory areas of virtual memory only when they are requested by a running program.

parallel processing Independent and usually simultaneous execution of several operations.

parsing Process by which textual input is transformed into internal representation.

* ***pattern*** An expression containing generic symbols; generic expression [2.6].

postprocessor A procedure or operation performed before output on each expression after processing (Post [1.3]).

precedence Attribute determining the grouping of input forms [2.10].

preprocessor A procedure or operation performed on each input expression before further processing (Pre [1.3]).

precedence Ordering for treatment of input forms [2.10].

procedure A set of expressions to be simplified in order when required [6.3]; program.

process Independent simplification of a procedure; job.

* ***processing impasse*** A state achieved when complete processing of an input expression appears to require infinite time or memory space [1.5].

* ***projection*** Expression specifying part of a general structure (e.g. `f[x,y]`) [2.3]; array subscripting; function or procedure call.

* ***projector*** The object from which a projection is taken [2.3] (e.g. `f` in `f[x,y]`).

prompt An indication that further input may be given.

* ***property*** Attribute of a symbol used to specify its treatment [4].

prompt Character or message printed by computer to indicate that input is required.

quit interrupt Real time interrupt used to terminate current processing [1.4].

real-time interrupt A command to be issued at any time (not necessarily after an input prompt) [1.4].

recursion Simplification of an expression involving further simplification of similar expressions [3.1].

* ***replacement*** A construct specifying a substitution (e.g. x->y) [3.3].

segment An expression forming part of a procedure [1.8,6.3].

* ***simplification*** The process performed by SMP on input expressions [3.1].

string A sequence of textual characters.

* ***status interrupt*** Real time interrupt used to obtain information on the status of processing.

subscripted variable Example of a projection.

* ***subsidiary mode*** Input mode for entry of segments in interactive subsidiary procedures [1.8].

* ***symbol*** Fundamental symbolic object (e.g. x and Mult) [2.2]; variable; parameter; unknown.

syntax Construction of expressions [2].

* ***system-defined*** Object or procedure with a built-in definition in SMP; primitive.

systems programming Computer programming intended as a service to many users rather than directed to a particular application.

tab ASCII HT; usually CTRL I; usually has fixed "stops" 8 spaces apart.

* ***template*** Expression specifying an action on a set of expressions [2.7].

terminal Computer input and output device; teletype; console; VDU.

termination character Character signifying termination of the current procedure and used to terminate a job [1.4].

text space Portion of computer memory in which the executable text of SMP resides.

* ***type extension*** Mechanism through which operations may be modified by the nature of their operands [4].

* ***user-defined*** Input by the "user" and not intrinsic to SMP.

user name Name by which a person or group is specified to a computer.

* ***value*** An expression specified to replace an expression or class of expressions on simplification [3.1].

virtual memory System by which effective computer memory may include not only physical main memory but also secondary storage media and thus be of practically arbitrary size.

SMP Reference Manual

Stephen Wolfram

Introduction

This reference manual gives an essentially complete description of the facilities available in SMP. Extensive illustrative examples are included. The same text is given without these examples, and slightly shortened, in the "Summary". The "SMP Primer" gives a more pedagogical outline of some important features of SMP.

Many additional facilities are included in the "SMP Library".

The procedure for initiating an SMP job should be described in the "Implementation Notes", together with other implementation-dependent features.

CONTENTS

0. Conventions

Text printed in `this font` represents literal SMP input or output, to be typed as it appears. Text in *italics* stands for SMP input or output expressions. Arbitrary characters or textual forms are given as ⊂*textual form*⊃ The actual characters corresponding to these textual forms in particular implementations should be given in the "Implementation Notes".

In descriptions of system-defined projections [2.3], the following notations for filters [2.3] are used

filt	Compulsory filter.
filt	Filter used in an unsimplified or partially simplified form. • [3.5, 3.6] • `Nosmp` [4]
(*filt*:*val*)	Optional filter assumed to have value *val* if omitted or given as `Null` (a blank [2.10]). An optional filter may be omitted entirely only when it would appear after the last filter explicitly specified in a projection.
f1, *f2*, ...	Sequence containing any number of similar filters.
{*f1*, *f2*, ...}	Filter to be given as a list of expressions.
⟨*filt*⟩	Filter to be given either as a single expression or as a list of expressions.
⟨*f1*, *f2*, ...⟩	Filter or sequence of filters to be given either as single expressions or as lists of expressions.

Properties carried by projections are indicated by
`<prop1, prop2,...>`

References to sections in the text of this summary are given as [⊂*number*⊃ References to additional or related material in the summary are indicated by •. References to external files or objects defined in them are indicated with ▫.

1. Basic system operation

1.1 Input and output

In standard mode the prompt for the *i*th input line is `#I [i]::`
In subsidiary mode, the prompt is `%I [i]::`

Input is terminated by a ⊂*newline*⊃ ⊂*delete character*⊃ may be used in unfinished input lines. All ⊂*tab*⊃ and unnecessary [2.10] spaces are ignored (unless they appear in quoted strings [2.2]). Input may be continued for several lines by placing \ (backslash) at the end of each intermediate line. If no text is entered before the terminating newline, the input is considered null, and the prompt is reissued.

The result generated by processing an input line is usually printed. No output is printed if ; (semicolon) is placed at the end of the input expression [6.3](or in general if the output expression is `Null` [2.2]).

Input of expressions with ambiguous syntax [2] yields a message on the nature of the ambiguity, and causes edit mode [1.7] to be entered. The input text is placed in the edit buffer, with the cursor positioned under the point at which the ambiguity was detected.

- [2.10]

Printed output is given in a two-dimensional format based on standard mathematical notation.

- [2.12]
- `Lpr` [10.1]

```
#I[1]::  a*y-x+2x

#O[1]:   x + a y

#I[2]::  a   *y- x + 2  x

#O[2]:   x + a y

#I[3]::  "a  *y- x + 2  x"

#O[3]:   "a  *y- x + 2  x"

#I[4]::  x+y+z\
         +w+z\
         +y

#O[4]:   w + x + 2y + 2z

#I[5]::  8/9-1/7

#O[5]:   47/63

#I[6]::  8/9-1/7;

#I[7]::  Pr[8/9-1/7];

47/63

#I[8]::  (a+b+)*c
) unexpected
         (a+b+)*c
<edit>         ^f
         (a+b+f)*c
<edit>

#O[8]:   c (a + b + f)
```

```
#I[9]:: (x^2+y^2)/(x^(2-a)+z^3)^4

                 2     2
                x  + y
#O[9]:  ---------------
          2 - a     3 4
         (x       + z )
```

1.2 Global objects

% The last expression (other than Null [2.2]) generated.

#I[*i*] The *i*th (unsimplified) input expression. (Assigned after the corresponding output expression is generated.)
<Ldist>

#O[*i*] The *i*th output expression.
<Ldist>

#T[*i*] The approximate time (in clicks [10.8]) required to generate #O[*i*].
<Ldist>
- Time [10.8]
- Clock [10.10]

@*i* Equivalent to #O[*i*]
<Ldist>

- %%, %I, %O, %T [1.8]

```
#I[1]::  t:(a+b)

#O[1]:   a + b

#I[2]::  %^2-2%

                                  2
#O[2]:   -2a - 2b + (a + b)

#I[3]::  3t-4;

#I[4]::  2/%

               2
#O[4]:  ------------
        -4 + 3a + 3b

#I[5]::  #O

                    2                                  2
#O[5]:  {[4]: ------------, [3]: , [2]: -2a - 2b + (a + b) , [1]: a + b,
              -4 + 3a + 3b

          [8]: {"/u1/smp/SRC/smp.init"}}

#I[6]::  Pr[t]

a + b

#O[6]:   a + b

#I[7]::  #O[6]

#O[7]:   a + b

#I[8]::  I[6]

#O[8]:   I[6]

#I[9]::  Ev[#I[6]]

#O[9]:   ' Pr[t]
```

```
#I[10]::  #T
#O[10]:   {[9]: 0, [8]: 0, [7]: 0, [6]: 0, [5]: 0, [4]: 0, [3]: 0, [2]: 1/30,
                [1]: 0, [0]: 0}
#I[11]::  2/@1-b
                    2
#O[11]:   -b + -------
              a + b
#I[12]::  @1[2]
#O[12]:   b
#I[13]::  @{20,22,1,12}
#O[13]:   {#O[20],#O[22],a + b,b}
#I[14]::  Ar[4,#T]
#O[14]:   {0,1/30,0,0}
```

Any values assigned [3.2] to the special symbols `Pre` and `Post` are taken as templates [2.7] and applied respectively as a "preprocessor" on each input expression and a "postprocessor" on each output expression.

```
#I[1]::  1+1/6
#O[1]:   7/6
#I[2]::  Post:N
#O[2]:   N
#I[3]::  1+1/6
#O[3]:   1.166667
#I[4]::  Post:
#I[5]::  1+1/6
#O[5]:   7/6
```

1.3 External files and job recording

External files are input by *<file*
- [10.3]

All input and output expressions are entered into a record file (usually named `smp.out`). (Implementation dependent)
- `Open, Output` [10.3]

1.4 Termination and real-time interrupts

(Implementation dependent)

⊂*input termination character*⊃
signifies termination of the current procedure. In standard and edit mode, it causes termination of the present job; in subsidiary mode [1.8] it results in return to the previous mode.
- Ret [6.3]
- Exit [10.5]

⊂*quit interrupt*⊃
attempts to terminate current processing or printing. Subsequent references to incompletely processed expressions cause their simplification to be completed.

⊂*break interrupt*⊃
causes any processing to be suspended and initiates an interactive subsidiary procedure [1.8].
- Proc [6.3]

⊂*status interrupt*⊃
prints the status of processing, including a stack of the last few projections simplified.
- State [10.8]
- Trace [4]

```
#I[1]::  D[x^x^x,x,2]
⊂quit interrupt⊃

#I[2]::  D[x^x,x]

          x              x
#O[2]:   x  Log[x] + x

#I[3]::  @1

                x                          x
          -1 + x     x            x     -2 + x + x              x
#O[3]:   x         (x  Log[x] + x ) + x             (-1 + x + x )
⊂quit interrupt⊃


#I[4]::  For[i:0;d[0]:x^x;s[0]:1,i<=6,Inc[i],d[i+1]:Ex[D[d[i],x]];\
         s[i+1]:S[d[i+1],x->1]];s

⊂status interrupt⊃
<simplifying Plus>
<simplifying Plus>
<simplifying Plus>
<simplifying Plus>
<simplifying Pow>
<simplifying Mult>
<simplifying Log>
<simplifying Pow>
<simplifying Plus>
<2649 simplifier entries; 21395 Blocks used>

⊂status interrupt⊃
<simplifying Rep>
<simplifying Plus>
<simplifying Pow>
<simplifying Plus>
<simplifying Pow>
<simplifying Log>
<simplifying Pow>
```

```
<simplifying Mult>
<11424 simplifier entries; 69874 Blocks used>

⊂break interrupt⊃

%I[1]::    i

%O[1]:     5

%I[2]::    s

%O[2]:     {[4]: 8, [3]: 3, [2]: 2, [1]: 1, [0]: 1}

%I[3]::    Size[d[4]]

%O[3]:     {168,164}

%I[3]::   ⊂input termination character⊃

#O[4]:     {[6]: 54, [5]: 10, [4]: 8, [3]: 3, [2]: 2, [1]: 1, [0]: 1}
⊂status interrupt⊃
<reclaiming memory>

#I[5]::
```

1.5 Processing impasses

If the complete processing of an input expression appears to require infinite time or memory space, its processing is suspended, and the user is consulted on the extent to which it should be continued. A response of `Inf` causes no further consultation. `0` results in immediate termination.

```
#I[1]::  f[$x]:f[$x-1]
#O[1]:   f[-1 + $x]
#I[2]::  f[5]
100 levels of recursion in simplifier - how many more ? 0
#O[2]:   f[-1 - 95]
#I[3]::  %
100 levels of recursion in simplifier - how many more ? 20
120 levels of recursion in simplifier - how many more ? 0
#O[3]:   f[-1 - 216]
#I[4]::  a:a+1
#O[4]:   2 + a
```

In implementations with finite available memory space, processing is suspended if only a small fraction of the memory remains. Memory is reclaimed when processing is complete.

```
#I[1]::   Ar[100000000]
95% of available memory used - processing suspended.
#I[2]::   Ev[%]
#O[2]:    ' Ar[100000000]
```

1.6 Monitor escapes

(Implementation dependent)

! ⊂*monitor command*⊃
: given directly at an input prompt executes the specified monitor (shell) command; the original input prompt is then repeated.

• Run [10.5]

```
#I[1]::  !date
Wed Dec 31 03:38:39 1980 PST
#I[1]::
```

1.7 Edit mode

In edit mode, one line of text is treated at a time. The editing of each line proceeds by a repetition of two steps:

1. The present form of the line is printed.
2. Following the prompt `<edit>` any editing commands for the line are entered. The commands are terminated by ⊂*newline*⊃ ⊂*tab*⊃ and ⊂*character delete*⊃ may be used for positioning in local editing.

If no editing commands are given in step 2 (the terminating newline is entered directly after the `<edit>` prompt), then the present line is left unchanged. If further lines of text exist, the editor moves to the next line; otherwise, edit mode is exited, and the edited text is returned as an input expression. If at the exit from edit mode, no editing has actually been performed, the text is discarded, leaving a null line.

Two types of editing may be performed:

Local editing, in which individual characters in the edit command affect the characters appearing directly above them in the present line.

Global editing, in which an edit command prefaced by \ affects the whole present line or the complete edit buffer.

Local editing

#	Delete character above.
⊂*space*⊃ or ⊂*tab*⊃	Leave character(s) above unchanged.
^⊂*text*⊃	Insert ⊂*text*⊃ terminated by a space or tab (not appearing between " " [2.2]), into the present line before the character above.
\d	Delete the remainder of the present line.
⊂*other character*⊃	Replace character above by character given.

Note that a character may be replaced with # by the command #^#

Global editing

\p	Print the complete text in the edit buffer; then return to the present line.
\q	Exit from edit mode, discarding the edited expression.
\n	Leave the present line unaltered, and move *n* lines forward. \- and \+ move one line forward and backward respectively.
\u	Undo the changes effected in the present line by the previous edit command.
\! ⊂*monitor command*⊃	Execute specified monitor command [1.6]
\m	Display levels of parentheses, braces and brackets in the present line.
\mg	Display levels of parentheses, braces and brackets in the complete text.
\s/⊂*e1*⊃/⊂*e2*⊃	Textually substitute ⊂*e2*⊃ for the character string ⊂*e1*⊃ throughout the present line. (The delimiter here given as / may be replaced by any character.)

\s*n*/⊂*e1*⊃/⊂*e2*⊃ Textually substitute for the first *n* occurrences of ⊂*e1*⊃ on or following the present line.

\sg/⊂*e1*⊃/⊂*e2*⊃ Textually substitute for ⊂*e1*⊃ throughout the text.

\e Invoke external editor on complete text, and return modified text. (Implementation dependent)

• Ed, Edh [10.6]

```
#I[1]::  (a++b**a-x)^3
+ unexpected
         (a++b**a-x)^3
<edit>      #  ^c ^2
         (a+b*c*a-2x)^3
<edit>      \d
         (a+b
<edit>   \u
         (a+b*c*a-2x)^3
<edit>   \s/a/(ap+1)
         ((ap+1)+b*c*(ap+1)-2x)^3
<edit>   \m
         12    2     2    2   1
         ((ap+1)+b*c*(ap+1)-2x)^3
<edit>
                                  3
#O[1]:   (1 + ap - 2x + b c (1 + ap))

#I[2]::  2//3%
/ unexpected

         2//3%
<edit>     ^1
         2/1/3%
<edit>   \q

#I[2]::  (2/3%
% unexpected
         (2/3%
<edit>

         (2/3%
<edit>

#I[2]::  2/3

#O[2]:   2/3
```

1.8 Procedures and subsidiary mode

Procedures consist of sequences of expressions ("segments") to be simplified in turn. Segments in procedures may either be provided as successive filters of a Proc projection [6.3], or may be entered "interactively" on successive input lines. The #I[*i*] are the segments of the outermost procedure, and are entered in standard input mode. Interactive subsidiary procedures are initiated by ⊂*break interrupt*⊃ or `Proc[]` [6.3], and terminated with ⊂*input termination character*⊃ or `Ret[]` [6.3]. Their segments are entered in subsidiary input mode. Unless specified otherwise [6.3], all expressions may be accessed and affected in any procedure. Objects local to a single subsidiary procedure are

%% The last expression (other than `Null` [2.2]) generated. The last value assigned to **%%** in a procedure is used as the value of the complete procedure.

%I[*i*] The *i*th (unsimplified) segment.
`<Ldist>`

%O[*i*] The value of the *i*th segment.
`<Ldist>`

%T[*i*] The approximate time (in clicks [10.8]) required to generate %O[*i*].
`<Ldist>`

```
#I[1]::  t:E^2-1

                 2
#O[1]:   -1 + E

#I[2]::  Proc[]

%I[1]::  N[t]

%O[1]:   6.389056

%I[2]::  tp:t+2

                 2
%O[2]:   1 + E

%I[3]::  Proc[]

%I[1]::  tp

                 2
%O[1]:   1 + E

%I[2]::  tq:N[tp]

%O[2]:   8.389056

%I[3]::  %%

%O[3]:   8.389056

%I[4]::  %O

                                                    2
%O[4]:   {[3]: 8.389056, [2]: 8.389056, [1]: 1 + E , [0]: Init}

%I[5]::  %

                 2
%O[5]:   -1 + E
```

```
%I[6]::  Ret[]

%I[4]::  %I

%O[4]:   {[3]:: Proc[], [2]:: tp : t + 2, [1]:: N[t], [0]:: Init}

%I[5]::  tq-1

%O[5]:   7.389056

%I[6]::  Ret[]

#I[3]::  %^2

                  2
#O[3]:   7.389056

#I[4]::  tp^3

                2 3
#O[4]:   (1 + E )
```

1.9 Information and elaboration

?*name* or **Info[*name*]**
prints any information on *name* given in this summary.

```
#I[1]::   ?D

    D[expr,|var1,(n1:1),(pt1:var1)|,|var2,(n2:1),(pt2:var2)|...]
         forms the partial derivative of expr successively ni  times
         with  respect  to  the  "variables"  vari,  evaluating  the
         results at the points vari -> pti.
```

? or **Info[]**
initiates interactive information mode.

Elaborations provide commentary and interactive assistance. Each elaboration printed is assigned a unique name of the form *#n⊂name⊃([i,j,..])*. The integer *n* specifies the "class" of the elaboration:

1 Prompt for respelling of unknown names.
2 Unexpected filters or undefined operations reported.
3 Unusual filters or operations reported.
4 Intermediate operations described.

On[{*elab1, elab2, ...*}]
causes the elaborations *elab1,elab2,..* to be printed when appropriate. *elabi* may be an integer corresponding to a class of elaborations.

On[]
turns on all elaborations.

Off[{*elab1, elab2, ...*}]
stops printing of elaborations or classes of elaborations specified by *elab1,elab2,...*

Off[]
turns off all elaborations.

```
#I[1]::  On[]

Elaboration modes now on. To answer questions type
y<newline> for yes, n<newline> or <newline> for no,
and o<newline> to turn off question.

#I[2]::  x+exp[x]

#1exp:  Replace  exp[x]  by  Exp[x] ? y

#0[2]:   x + Exp[x]

#I[3]::  exp[x+2]

#1exp:  Replace  exp[x+2]  by  Exp[x+2] ? n

#0[3]:   exp[x + 2]

#I[4]::  Off[#1exp]

Turning off prompt for replacement of  exp  by  Exp.

#I[5]::  exp[x]
```

```
#O[5]:   exp[x]

#I[6]::  D[y,x]

#3O[2]:  D[y,x] gives 0 since variable y is assumed independent of x.

#O[6]:   0

#I[7]::  Off[#3O]

Turning off all level 3 elaboration on differentiation D.

#I[8]::  D[z,x]

#O[8]:   0
```

1.10 External programs and program construction

(Implementation dependent)

External programs may be run with SMP output expressions as input using Run [10.4]; their output may be taken as SMP input.

External programs may be constructed from SMP expressions using Cons [10.9] and may be run in a compiled form.

1.11 Parallel processing

(Implementation dependent)

SMP operations may be performed asynchronously and in parallel using `Fork` and `Wait` [10.10].

Values in the list `Rti` give expressions to be simplified on receipt of real-time interrupts.

Future implementations may employ automatic parallel processing: filters in projections which do not carry property `Ser` are then simplified in parallel rather than sequentially.

2. Syntax

2.1 Numbers

Numbers input with no decimal point are taken as exact integers.

Floating point numbers may be entered in the form *x**^*p* representing $x*10^p$.

Numbers are output if possible in integer or rational form, except when the corresponding input expression contains explicit floating point numbers or N projections [3.4]. Output floating point numbers are given to the precision specified in input N projections, or, by default, to 6 significant figures.

Numbers are by default treated to a finite precision (fractional accuracy of order 10^-13).

```
#I[1]:  55/7+8/5
#O[1]:  331/35
#I[2]:  c|:3*^-8
#O[2]:  3.*^-88
#I[3]:  N[@1]
#O[3]:  9.457143
#I[4]:  55.0/7+8/5
#O[4]:  9.457143
#I[5]:  N[%,14]
#O[5]:  9.45714285714286
#I[6]:  1+(1/1000)^8
#O[6]:  1
#I[7]:  %-1
#O[7]:  0
```

Further numerical constructs (which may be used in any numerical operations) are:

A[*x*, (*p*:0)] Arbitrary magnitude number $x*10^p$. Floating point numbers are usually converted to this form when the modulus of their exponent exceeds 10.

B[*nk*, . . ., *n2*, *n1*, *n0*]
Arbitrary length integer $n0 + n1*10^4 + n2*10^8 + \cdots$ generated when integers with more than 10 digits are input.

F[*n1*, *n2*, . . ., (*expt*:0), (*dig*:6)]
Floating point number
$(n1*10^{-4} + n2*10^{-8} + \cdots)*10^{expt}$ with *dig* significant digits. F projections are generated if possible when the precision specified in an N projection exceeds 10. Expressions involving several F projections are treated to the numerical accuracy of the least precise F given.

Err[*x*, *dx*] Number *x* with one-standard-deviation error *dx* ($x \pm dx$). Errors are combined assuming statistical independence.

Cx[*x*, *y*] Complex number $x + I\ y$. Automatically generated when required.

The presence of any A, B or F projections in an expression forces conversion of all numbers to these forms.

```
#I[1]::  h:6.62559*^-34
#O[1]:   A[6.62559,-34]
#I[2]::  h^8
#O[2]:   A[3.713611,-266]
#I[3]::  12123123412345123456
#O[3]:   (12123123412345123456)
#I[4]::  %^2
#O[4]:   (146978121270950470243047013943881383936)
#I[5]::  %/124234
                (73485066063547523512152350697194069196 8)
#O[5]:   ------------------------------------------
                                (62117)
#I[6]::  100!
#O[6]:   (933262154439441526816992388562667084907159682643816214685929638952\
           175999932299156089414639761565182862536979208272237582511852109 1\
           6864000000000000000000000000)
#I[7]::  N[%]
#O[7]:   A[9.332622,157]
#I[8]::  N[1/7,50]
#O[8]:   (0.14285714285714285714285714285714285714285714285714285714)
#I[9]::  N[1.1/7,100]
#O[9]:   .1571429
#I[10]::  Err[6.1,0.5]+Err[8.9,0.7]
#O[10]:   Err[15,.86023251
#I[11]::  (5+6I)(8+9I)
#O[11]:*  -14 + 93 I
#I[12]::  Lpr[%]
Cx[-14,93]
#O[12]:*  -14 + 93 I
```

2.2 Symbols

Symbols are the basic objects of SMP. They may be assigned values [3.2] and properties [4].

Symbol names may be

Strings of arbitrary length containing only alphanumeric characters, together with #, $ and % and not starting with numeric characters.

Arbitrary character strings (possibly containing control characters) enclosed between " ". The " " are not included in the symbol name; they are used on output only when necessary.
- [10.6]

Symbols whose names have the following forms are taken to have special characteristics (*ccc* represents any valid symbol name):

$*ccc* Generic symbol (representing an arbitrary expression [2.6])
- Gen [4, 2.6]

$$*ccc* Multi-generic symbol (representing an arbitrary sequence of expressions [2.6]).
- Mgen [4]

##*ccc* Chameleonic symbol (whose name changes whenever it is evaluated [2.8]).
- Cham [4]

No values may be assigned to symbols of these types.
- % [1.2]

The following naming conventions are used (*C* denotes any upper case alphabetic character):

*C**ccc* System-defined symbol.

#*ccc* Internally-generated symbol.

%*ccc* Symbol used locally within a procedure [1.8,6.3]

Some special system-defined symbols are:

Null Input and output as a blank [2.10]. When the value assigned for a projection would simplify to Null, the projection is left unevaluated [3.1].

Inf Infinity. Used primarily to specify indefinite continuation of a process, rather than as a signal for mathematical infinities.

I The imaginary unit.

- %, #I, #O, #T [1.2]
- %%, %I, %O, %T [1.8]
- Pi, E, Euler,... [8.4]

```
#I[1]:: 1+EighthOrderTerm

#O[1]:  1 + EighthOrderTerm

#I[2]:: {a2,"a2",44,"44","8th order",""}

#O[2]:  {a2,a2,44,"44","8th order"," 87 87 87"}

#I[3]:: Pr[number,"square of number"];Do[i,3,Pr[i,i^2]]

number  square of number
1       1
2       4
```

```
3        9
#O[3]:   9
#I[4]::  f[$x]:$x^2
                 2
#O[4]:   $x
#I[5]::  f[a]+f[b]
           2     2
#O[5]:   a  + b
#I[6]::  _$x
#O[6]:   {[Gen]: 1}
#I[7]::  g[$$x]:h[1,$$x,2]
#O[7]:   h[1,$$x,2]
#I[8]::  g[a,b]+g[]+g[a]
#O[8]:   h[1,,2] + h[1,a,2] + h[1,a,b,2]
#I[9]::  _$$x
#O[9]:   {[Mgen]: 1, [Gen]: 1}
#I[10]::  t::##v+##w
#O[10]:   ' ##v + ##w
#I[11]::  t
#O[11]:   ##v1 + ##w1
#I[12]::  t
#O[12]:   ##v2 + ##w2
#I[13]::  _##v
#O[13]:   {[Cham]: 1}
#I[14]::  {_log,_loge,_log,_Pi,_pi}
#O[14]:   {{[Ldist]: 1, [Tier]: 1, [Sys]: 1},_loge,_log,{[Const]: 1, [Sys]: 1},
                 _pi}
#I[15]::  Tree[h[x^2]]
                                  2
#O[15]:   {#1 -> h[#2],#2 -> x  }
#I[16]::  D[h[x],x]+D[h1[x^2],x]
                                                  2
#O[16]:   D[h[#4],{#4,1,x}] + 2x D[h1[#5],{#5,1,x  }]
#I[17]::  Lpr[p[a,,b,]+q[]+r[{,}]]
p[a,Null,b,Null] + q[Null] + r[{Null,Null}]
#I[18]::  w[x]:Null
#I[19]::  a+w[x]
#O[19]:   a + w[x]
```

```
#I[20]:: Map[q,a+b^(c+d),Inf]

                              q[q[c] + q[d]]
#O[20]:  q[a] + q[q[b]                      ]

#I[21]:: I^2

#O[21]:  -1
```

2.3 Projections

Projections specify parts of general structures. They are analogous to array subscriptings or function calls.

In for example the projection `f[x,y]` the symbol `f` is termed the "projector", and `x,y` its "filters". These filters are used successively or together to select a part in the value of `f`.

The projector *f* in a projection is maintained in an unsimplified "held" [3.5] form; only its projections are simplified.

`f[x,y]` is equivalent to `f[x][y]` unless `f` carries the property `Tier` [4].
• `Proj` [7.3]

Common system-defined projections may be input in special forms [2.10].

Filters for system-defined projections input as `Null` [2.2] (a blank [2.10]) are taken to have their default values.

```
#I[1]::  f:{[x]:{[y]:a1,[yp]:a2},[xp]:b}
#O[1]:   {[x]: {[y]: a1, [yp]: a2}, [xp]: b}
#I[2]::  f[x,y]
#O[2]:   a1
#I[3]::  f[x]
#O[3]:   {[y]: a1, [yp]: a2}
#I[4]::  f[x][y]
#O[4]:   a1
#I[5]::  u[x][y]
#O[5]:   u[x,y]
#I[6]::  g:a^2+b*c
                  2
#O[6]:   b c + a
#I[7]::  g[1]
#O[7]:   b c
#I[8]::  g[0]
#O[8]:   ' Plus
#I[9]::  g[1,1]
#O[9]:   b
#I[10]::  f_:Tier
#O[10]:   Tier
#I[11]::  f[x]
#O[11]:   f[x]
#I[12]::  f[x,y]
#O[12]:   a1
```

```
#I[13]::  u_Tier
#0[13]:   Tier
#I[14]::  u[x][y]
#0[14]:   Proj[u[x],{y}]
#I[15]::  g[x]
#0[15]:   g[x]
#I[16]::  Mult[a,b,c]
#0[16]:   a b c
#I[17]::  a*b*c
#0[17]:   a b c
#I[18]::  Rand[]
#0[18]:   .81579895
```

[*x1,x2,...*] or **Np[*x1,x2,...*]**
is a special system-defined null projection representing a sequence of expressions. When **[*x1,x2,...*]** appears as a filter in a projection, that filter is replaced by the set of filters *x1,x2,...* Hence `f[x,[y,z],[[w]]]` becomes `f[x,y,z,w]`. Null projections may also be used to specify sets of entries in lists [2.4]. No values may be assigned [3.2] to null projections. Removal of input `Np` projections occurs through a type extension mechanism [4].
• Seq, Repl [7.1]

```
#I[1]::  r:[[x,y,z]]
#0[1]:   [x,y,z]
#I[2]::  f[r,w,[r,w]]
#0[2]:   f[x,y,z,w,x,y,z,w]
#I[3]::  {[a,b],c,[[d],e]}
#0[3]:   {a,b,c,d,e}
#I[4]::  1..5
#0[4]:   [1,2,3,4,5]
#I[5]::  f[$$x]:$$x
#0[5]:   $$x
#I[6]::  f[a,b,c]
#0[6]:   [a,b,c]
```

`*expr* or **Mark[*expr*]**
is used in a variety of cases to designate expressions with special characteristics.
(Prefix form is "backquote" or "grave accent" character.)
• [2.7]

```
#I[1]:: f[`1,`x+y,a+`b,``c,{`g[x],`{{{{{}]
#O[1]:  f[` 1,` x + y,a + (` b),` (` c),{` g[x],` {{{{{}]
#I[2]:: Ap[f,{a,b}]
#O[2]:  f[a,b]
#I[3]:: Ap[`f,{a,b}]
#O[3]:  f
```

2.4 Lists

Lists are indexed and ordered sets of expressions.

Lists may be input directly in the form {[*index1*] : *value1*, [*index2*] : *value2*, ...}. Entries with delayed rather than immediate values [3.2] are input with :: instead of : .

If the expressions *indexi* are successive integers starting at 1, they may be omitted. The list may then be input as {*value1*, *value2*, ...}. Such lists are termed "contiguous".

• List [7.1]

{} is a zero-length list containing no entries.

List entries may also be specified indirectly through assignments for projections [3.2].

A value in a list may be extracted by a projection [2.3] with its index as a filter.

• Ar [7.1]

```
#I[1]::  r:{[x]:a, [y]::Rand[], [$x]:$x-3}
#O[1]:   {[x]: a, [y]:: Rand[], [$x]: -3 + $x}
#I[2]::  t:{a,b,c}
#O[2]:   {a,b,c}
#I[3]::  t[5]:e
#O[3]:   e
#I[4]::  t
#O[4]:   {[5]: e, [1]: a, [2]: b, [3]: c}
#I[5]::  t[4]:d
#O[5]:   d
#I[6]::  t
#O[6]:   {a,b,c,d,e}
#I[7]::  Lpr[{{},{,}}]
{{},{Null,Null}}
#I[8]::  r[x]:b
#O[8]:   b
#I[9]::  r
#O[9]:   {[x]: b, [y]:: Rand[], [$x]: -3 + $x}
#I[10]::  r[x]
#O[10]:   b
#I[11]::  r[y]
#O[11]:   .4172875
```

```
#I[12]::  r[y]
#O[12]:   .9003638
#I[13]::  r[z]
#O[13]:   -3 + z
```

2.5 Expressions

Expressions are combinations of symbols, projections and lists.

Parts of expressions are selected by projections [2.3,7.3]. Values in a list are specified by their indices [2.4]. Parts in projections are specified by numerical filters: `0` specifies the projector (which is always in a "held" form [3.5]) and `1,2,3,..` label each of its filters. Numerical coefficients of symbols and projections are specified by the filter `-1`. The `0` part of a symbol is the symbol itself, without any associated numerical coefficients.

- Nc [7.9]
- Pos, Dis [7.3]

```
#I[1]::  t:{[x]:c,[y]:{a1,a2}}

#O[1]:   {[x]: c, [y]: {a1,a2}}

#I[2]::  t[y]

#O[2]:   {a1,a2}

#I[3]::  t[y][1]

#O[3]:   a1

#I[4]::  t[y,2]

#O[4]:   a2

#I[5]::  u:2a-3f[x,y^2]/5

                      2
                  3f[x,y ]
#O[5]:   2a - ----------
                    5

#I[6]::  u[1]

#O[6]:   2a

#I[7]::  u[2,1]

#O[7]:   x

#I[8]::  u[2,2]

          2
#O[8]:   y

#I[9]::  %[1]

#O[9]:   y

#I[10]::  u[0]

#O[10]:   ' Plus

#I[11]::  u[1][-1]

#O[11]:   2

#I[12]::  u[2,-1]

#O[12]::  -3/5

#I[13]::  u[1,0]
```

```
#O[13]:   a
#I[14]::  u[5]
#O[14]:   u[5]
#I[15]::  Pos[x,u]
#O[15]:   {{2,1}}
#I[16]::  Dis[u]
#O[16]:   {[0]: ' Plus, [1]: {[-1]: 2, [0]: b},
                [2]: {[-1]: -3/5, [0]: ' f, [1]: x, [2]: y^2}}
```

The *nth* "level" in an expression is the set of parts which may be selected by *n* filters (when *n* is a positive integer). The "depth" of an expression is one plus the maximum number of filters necessary to specify any part [7.4]. The *-n*th level in an expression is the set of parts which have depth *n*.

A domain is a set of parts in an expression. Domains may be specified by a parameter *levspec* giving the levels at which its elements may occur (*ni* are positive integers):

n	Levels 0 through *n*.
-n	Levels -Inf through *-n*.
{±*n*}	Level ±*n*.
{±*n1*, ±*n2*}	Levels ±*n1* through ±*n2*.
{*-n1*, *n2*}	Levels -Inf through *-n1* and 0 through *n2*.
{*l1*, *l2*, *levcrit*}	Levels specified by {*l1*, *l2*} on which application of the template [2.7] *levcrit* does not yield false [5].

Domains may also be selected by requiring that application of a template [2.7] *domcrit* to any of their parts should yield true [5].

Many system-defined projections may carry filters which select particular domains. A repeat count *rpt* is usually given to specify the number of times an operation is to be performed successively on a particular domain (Inf causes repetition to continue until the domain no longer changes or a processing impasse is reached [1.5]). A parameter *max* is used to determine the total maximum number of operations performed on any domain. The filters *levspec*,*rpt*,*domcrit*,*max* (or some subset of these) constitute a "domain specification". In treatment of a domain containing positive levels, larger subparts of an expression are treated first, following by their progressively smaller subparts. In a domain containing negative levels, the smallest subparts are treated first. Different parts at the same level are ordered by their "positions" (the sequences of filters necessary to select them). (Depth-first traverse.) When *rpt* is specified, the subparts appearing in a domain are re-determined each time the operation is performed.

```
#I[1]::   Dep[a]
#O[1]:    1
#I[2]::   Dep[a+b+c]
#O[2]:    2
```

```
#I[3]::  t:a+b^(c d)+g[{x,{y,g[z]}}]

                 c d
#O[3]:   a + b     + g[{x,{y,g[z]}}]

#I[4]::  Dep[t]

#O[4]:   6

#I[5]::  Pos[b,t]

#O[5]:   {{2,1}}

#I[6]::  Pos[z,t]

#O[6]:   {{3,1,2,2,1}}

#I[7]::  Pos[g,t]

#O[7]:   {{3,0},{3,1,2,2,0}}

#I[8]::  Map[f,t]

                      c d
#O[8]:   f[a] + f[b     ] + f[g[{x,{y,g[z]}}]]

#I[9]::  Map[f,t,{2}]

                   f[c d]
#O[9]:   a + f[b]        + g[f[{x,{y,g[z]}}]]

#I[10]::  Map[f,t,2]

                        f[c d]
#O[10]:  f[a] + f[f[b]         ] + f[g[f[{x,{y,g[z]}}]]]

#I[11]::  Map[f,t,Inf]

                        f[f[c] f[d]]
#O[11]:  f[a] + f[f[b]              ] + f[g[f[{f[x],f[{f[y],f[g[f[z]]]}]}]]]

#I[12]::  Map[f,t,-1]

                f[c] f[d]
#O[12]:  f[b]           + f[a] + g[{f[x],{f[y],g[f[z]]}}]

#I[13]::  Map[f,t,{-1}]

                f[c] f[d]
#O[13]:  f[b]           + f[a] + g[{f[x],{f[y],g[f[z]]}}]

#I[14]::  Map[f,t,{-2}]

                   f[c d]
#O[14]:  a + b          + g[{x,{y,f[g[z]]}}]

#I[15]::  Map[f,t,-2]

                        f[f[c] f[d]]
#O[15]:  f[b]                       + f[a] + g[{f[x],{f[y],f[g[f[z]]]}}]

#I[16]::  Map[f,t,-Inf]

                        f[f[c] f[d]]
#O[16]:  f[a] + f[f[b]              ] + f[g[f[{f[x],f[{f[y],f[g[f[z]]]}]}]]]

#I[17]::  Map[f,t,{2,3}]

                      f[f[c] f[d]]
#O[17]:  a + f[b]                  + g[f[{f[x],f[{y,g[z]}]}]]
```

```
#I[18]:: Map[f,t,{-2,3}]

                f[f[c] f[d]]
#O[18]: f[b]               + f[a] + g[{f[x],{y,g[z]}}]

#I[19]:: Map[f,t,{0,Inf,Evenp}]

                  f[c d]
#O[19]: a + f[b]        + g[f[{x,{f[y],f[g[z]]}}]]

#I[20]:: Map[f,t,0]

          c d
#O[20]: a + b    + g[{x,{y,g[z]}}]

#I[21]:: Map[f,t,Inf,,$x]

                          f[f[c,3] f[d,3],2]
#O[21]: f[a,1] + f[f[b,2]                  ,1]
                 + f[g[f[{f[x,3],f[{f[y,4],f[g[f[z,5]],4]},3]},2]],1]

#I[22]:: Map[f,t,-Inf,,$x]

                            f[f[c,-1] f[d,-1],-2]
#O[22]: f[a,-1] + f[f[b,-1]                     ,-3]
                  + f[g[f[{f[x,-1],f[{f[y,-1],f[g[f[z,-1]],-2]},-3]},-4]],-5]

#I[23]:: i:0;Map[f[$x,Inc[i]],t]

                    c d
#O[23]: f[a,1] + f[b   ,2] + f[g[{x,{y,g[z]}}],3]

#I[24]:: i:0;Map[f[$x,Inc[i]],t,-1]

              f[c,3] f[d,4]
#O[24]: f[b,2]               + f[a,1] + g[{f[x,5],{f[y,6],g[f[z,7]]}}]

#I[25]:: i:0;Map[f[$x,Inc[i]],t,Inf]

                          f[f[c,3] f[d,4],5]
#O[25]: f[a,1] + f[f[b,2]                  ,6]
                 + f[g[f[{f[x,7],f[{f[y,8],f[g[f[z,9]],10]},11]},12]],13]

#I[26]:: i:0;Map[f[$x,Inc[i]],t,-Inf]

                          f[f[c,3] f[d,4],5]
#O[26]: f[a,1] + f[f[b,2]                  ,6]
                 + f[g[f[{f[x,7],f[{f[y,8],f[g[f[z,9]],10]},11]},12]],13]

#I[27]:: Map[f,t,Inf,Listp]

          c d
#O[27]: a + b    + g[f[{x,f[{y,g[z]}]}]]

#I[28]:: Map[f,t,Inf,~In[z,$x]]

                    f[f[c] f[d]]
#O[28]: f[a] + f[f[b]            ] + g[{f[x],{f[y],g[z]}}]

#I[29]:: Map[f,t,2,Listp]

          c d
#O[29]: a + b    + g[f[{x,{y,g[z]}}]]

#I[30]:: i:0;Map[f[$x,Inc[i]],t,Inf,,,7]

                          f[f[c,3] f[d,4],5]
#O[30]: f[a,1] + f[f[b,2]                  ,6] + g[{f[x,7],{y,g[z]}}]
```

2.6 Patterns

A "pattern" or "generic expression" is an expression containing generic symbols [2.2]. A pattern represents a possibly infinite set of expressions, in which arbitrary expressions replace the generic symbols. Every occurrence of a particular generic symbol in a pattern corresponds to the same expression.

Two patterns are considered equivalent if the sets of expressions which they represent are identical; they are literally equivalent if all their parts are identical.

Determination of literal equivalence takes account of filter reordering [4,7.7] properties of projections, and of the correspondence between numerical coefficients [2.5] and explicit Mult projections.

A pattern *p2* "matches" *p1* if it represents a superset of the expressions represented by *p1* (so that *p1* may be obtained by replacing some or all of the generic symbols in *p2*). *p1* is then considered "more specific" than *p2*.

Match[*expr2*, *expr1*]
yields 0 if *expr2* does not match *expr1*, 1 if *expr2* is equivalent to *expr1*, or a list of replacements for generic symbols in *expr2* necessary to obtain *expr1*.

```
#I[1]:: Match[x^2,x^2]
#O[1]:  1
#I[2]:: Match[x^2,y^2]
#O[2]:  0
#I[3]:: Match[$x^2,y^2]
#O[3]:  {$x -> y}
#I[4]:: Match[$x^2,(a+b)^2]
#O[4]:  {$x -> a + b}
#I[5]:: Match[y^2,$x^2]
#O[5]:  0
#I[6]:: Match[$x^2,$y^2]
#O[6]:  {$x -> $y}
#I[7]:: Match[f[$x],f[x]]
#O[7]:  {$x -> x}
#I[8]:: Match[f[x],f[x,y]]
#O[8]:  0
#I[9]:: Match[f[$x,$y],f[x,a+b]]
#O[9]:  {$x -> x,$y -> a + b}
#I[10]:: Match[$x^$x,x^x]
#O[10]:  {$x -> x}
#I[11]:: Match[$x^$x,x^y]
#O[11]:  0
```

```
#I[12]::  Match[{$x,a},{x+y,a}]
#O[12]:   {$x -> x + y}
#I[13]::  Match[{[x]:$a},{[x]:b+c}]
#O[13]:   {$a -> b + c}
#I[14]::  Match[{[$x]:a},{[x]:a}]
#O[14]:   {$x -> x}
#I[15]::  Match[f[$x,$y,$x],f[a,b+c,a]]
#O[15]:   {$x -> a,$y -> b + c}
#I[16]::  Match[f[$x,$y,$x],f[a,b+c,c]]
#O[16]:   0
#I[17]::  Match[$f[$x],f[x]]
#O[17]:   {$f -> (' f),$x -> x}
#I[18]::  Match[$x a,2a]
#O[18]:   {$x -> 2}
#I[19]::  Match[b c $x,a b c]
#O[19]:   {$x -> a}
#I[20]::  h_:Comm
#O[20]:   Comm
#I[21]::  Match[h[$x,a],h[a,b]]
#O[21]:   {$x -> b}
#I[22]::  Match[f[$x,a]f[a,b]]
#O[22]:   0
#I[23]::  r[x]:a^x
                  x
#O[23]:   a
#I[24]::  r[y]
#O[24]:   r[y]
#I[25]::  r[$x]:b^$x
                  $x
#O[25]:   b
#I[26]::  {r[y],r[a+b]}
                y   a + b
#O[26]:   {b ,b       }
#I[27]::  r
                          x          $x
#O[27]:   {[x]: a , [$x]: b  }
```

```
#I[28]:: S[f[x^2,y^2,x^3],$x^2->h[$x]]

                                3
#O[28]:  f[h[x],h[y],x ]

#I[29]:: Pos[$x^$y,f[x^2,y^2,x^3,z]]

                    2          2          3
#O[29]:  {{x ,{1}},{y ,{2}},{x ,{3}}}

#I[30]:: $x f[$x]:hp[$x]

#O[30]:  hp[$x]

#I[31]:: {a b f[a], 2f[3], 2 a f[2], y^2 f[x^2] a x^2}

                                    2     2
#O[31]:  {b hp[a],2f[3],2a f[2],a y  hp[x ]}

#I[32]:: Match[a+c,a+b+c+d]

#O[32]:  0

#I[33]:: a+c:p

#O[33]:  p

#I[34]:: a+b+c+d

#O[34]:  b + d + p

#I[35]:: a.$x:j[$x]

#O[35]:  j[$x]

#I[36]:: {x.a.b.y,a.x.y,a.x}

#O[36]:  {x.j[b].y,j[x].y,j[x]}

#I[37]:: a*$x:jp[$x]

#O[37]:  jp[$x]

#I[38]:: {x a b y,a x y,a x}

#O[38]:  {x y jp[b],y jp[x],jp[x]}
```

A pattern *p2* matches *p1* if the simplified form of *p2* after replacement of generic symbols is literally equivalent to *p1*. However, for at least one occurrence of each generic symbol in *p2* the necessary replacement must follow from literal comparison with *p1* (and must thus appear as an explicit part of *p1*).

```
#I[1]:: Match[f[$x,1-$x],f[x,1-x]]

#O[1]:  {$x -> x}

#I[2]:: Match[f[$x,1-$x],f[x+y,1-x-y]]

#O[2]:  {$x -> x + y}

#I[3]:: Match[f[$x,1-$x],f[3,-2]]

#O[3]:  {$x -> 3}

#I[4]:: Match[f[$x+$y,$x-$y],f[2,0]]

#O[4]:  0
```

```
#I[5]:: Match[f[$x+$y,$x,$y],f[2,1,1]]

#O[5]:  {$x -> 1,$y -> 1}

#I[6]:: q[$a+$b,g[$a,$b]]:h[$a,$b]

#O[6]:  h[$a,$b]

#I[7]:: {q[7,g[3,4]],q[a^2+b^2-1,g[a^2-2,b^2+1]],q[a,g[b,c]]}

                                2    2    2    2
#O[7]:  {h[3,4],q[-1 + a  + b  ,g[-2 + a  ,1 + b  ]],q[a,g[b,c]]}

#I[8]:: h_:Comm

#O[8]:  Comm

#I[9]:: h[$a,$a+1]:hp[$a]

#O[9]:  hp[$a]

#I[10]:: {h[4,3],h[2,3],h[b,a],h[a,b]}

#O[10]:  {h[3],h[2],h[a,b],h[a,b]}
```

The value *t* of the property [4]_$*x*[Gen] restricts all occurrences of the generic symbol *x* to match only those expressions on which application of the template *t* yields "true" [5].

```
#I[1]:: _$i

#O[1]:  {[Gen]: 1}

#I[2]:: _$i[Gen]:Intp

#O[2]:  Intp

#I[3]:: g[$i]:2$i

#O[3]:  2$i

#I[4]:: g[x]+g[4]

#O[4]:  8 + g[x]
```

pat _= ***cond*** or Gen[***pat***, ***cond***]
represents a pattern equivalent to ***pat***, but restricted to match only those expressions for which ***cond*** is determined to be true [5] after necessary replacements for generic symbols.

```
#I[1]:: f[$n_:Intp[$n] & $n>0]:$n!

#O[1]:  $n!

#I[2]:: f[2]+f[-3]+f[x]

#O[2]:  2 + f[-3] + f[x]

#I[3]:: g[$x,$y_:$x+$y > 0]:t[$x,$y]

#O[3]:  t[$x,$y]

#I[4]:: g[x,y]+g[-1,0]+g[5,1]

#O[4]:  g[-1,0] + g[x,y] + t[5,1]
```

```
#I[5]:: u[$x_?Nc[$x]<0]:-u[-$x]
#O[5]:  -u[-$x]
#I[6]:: u[-x]+u[a+b]+u[u[-7]]
#O[6]:  -u[x] + u[a + b] - u[u[7]]
```

Multi-generic symbols [2.2] are a special class of generic symbols which represent sequences of expressions, corresponding to null projections [2.3] of any length. In a projection (or list), the sequence of filters matched by a multi-generic symbol is terminated when all subsequent filters can be otherwise matched. For projections with Comm or Reor [4] properties, all but one multi-generic symbol is required to match a zero-length sequence of expressions []. In a projection from a projector *f* with property Flat [4], a multi-generic symbol matches a sequence of filters corresponding to a further projection from *f:* in the original projection, it is not represented by a single ordinary generic symbol.

```
#I[1]:: Match[f[$x],f[x,y]]
#O[1]:  0
#I[2]:: Match[f[$$x],f[x,y]]
#O[2]:  {$$x --> [x,y]}
#I[3]:: Match[f[$$x,w,$$z],f[a,b,w,z]]
#O[3]:  {$$z --> [z],$$x --> [a,b]}
#I[4]:: Match[f[$$x,w,$$z],f[a,w,c,w,z]]
#O[4]:  {$$z --> [c,w,z],$$x --> [a]}
#I[5]:: Match[f[$$x,w,$$z],f[a,b,c,w]]
#O[5]:  {$$z --> [],$$x --> [a,b,c]}
#I[6]:: Match[f[$$x,w,$$z],f[w]]
#O[6]:  {$$z --> [],$$x --> []}
#I[7]:: Match[f[$$x,$x],f[a,b,c,d]]
#O[7]:  {$x -> d,$$x --> [a,b,c]}
#I[8]:: Match[f[$$x,$x,$y],f[a,b,c,d]]
#O[8]:  {$x -> c,$y -> d,$$x --> [a,b]}
#I[9]:: Match[f[$$x,$x,$y],f[a,b]]
#O[9]:  {$x -> a,$y -> b,$$x --> []}
#I[10]:: Match[f[$$x,$x,$y],f[a]]
#O[10]:  0
#I[11]:: Match[f[$$x,$$y],f[a,b,c,d]]
#O[11]:  {$$y --> [a,b,c,d],$$x --> []}
#I[12]:: Match[f[$$x,a,$$x],f[b,c,a,b,c]]
#O[12]:  {$$x --> [b,c]}
#I[13]:: Match[f[$$x,a,$$x],f[b,c,a,b,d]]
```

```
#O[13]:   0
#I[14]::  Match[f[$$x,a,$$y],f[a,b,c,d,e]]
#O[14]:   {$$y --> [b,c,d,e],$$x --> []}
#I[15]::  Match[f[$x],f[x]]
#O[15]:   {$x -> x}
#I[16]::  Match[f[$$x],f[x]]
#O[16]:   {$$x --> [x]}
#I[17]::  pal:f[$$x_;List[$$x]=Rev[List[$$x]]]
#O[17]:   f[$$x _;  (List[$$x] = Rev[List[$$x]])]
#I[18]::  Match[pal,f[a,b,a]]
#O[18]:   {$$x --> [a,b,a]}
#I[19]::  Match[pal,f[a,b,b]]
#O[19]:   0
#I[20]::  g_;Comm
#O[20]:   Comm
#I[21]::  Match[g[$$x,c],g[a,c,x,y]]
#O[21]:   {$$x --> [a,x,y]}
#I[22]::  Match[g[$$x,$x],g[a,b,c,d]]
#O[22]:   {$x -> a,$$x --> [b,c,d]}
#I[23]::  Match[g[$$x,$x,$y],g[a,b,c,d]]
#O[23]:   {$x -> a,$y -> b,$$x --> [c,d]}
#I[24]::  Match[g[$$x,$x_;Natp[$x]],g[a,2,b,c]]
#O[24]:   {$x -> 2,$$x --> [a,b,c]}
#I[25]::  Match[g[$$x,w,$$y],g[a,b,w,z]]
#O[25]:   {$$x --> [a,b,z]}
#I[26]::  h_;Flat
#O[26]:   Flat
#I[27]::  Match[h[$$x],h[a,b,c]]
#O[27]:   {$$x -> h[a,b,c]}
#I[28]::  Match[h[$$x],h[a]]
#O[28]:   {$$x -> h[a]}
#I[29]::  Match[h[a,$$x],h[a,b]]
#O[29]:   {$$x -> h[b]}
#I[30]::  Match[h[a,$$x],h[a]]
#O[30]:   0
#I[31]::  p[$x $y]:p[$x]+p[$y]
```

```
#O[31]:   p[$x] + p[$y]
#I[32]::  {p[a b],p[a b c]}
#O[32]:   {p[a] + p[b],p[a b c]}
#I[33]::  q[$x $$x]:q[$x]+q[$$x]
#O[33]:   q[$$x] + q[$x]
#I[34]::  {q[a b],q[a b c],q[a b c d]}
#O[34]:   {q[a] + q[b],q[a] + q[b] + q[c],q[a] + q[b] + q[c] + q[d]}
#I[35]::  S[a.b.a.c.d.a,a.$$x.a->r[$$x]]
#O[35]:   r[b].c.d.a
```

2.7 Templates

A template *t* is an expression specifying an action to be taken on a set of expressions {*s1*,*s2*,...}.

The result from an application of *t* depends on its structure:

number or Null	*t*
pattern	The value of *t* obtained by replacement of generic symbols *di* occurring in it by any corresponding *si*. The association of *si* with *di* is determined by first sorting the *di* into canonical order. Any unpaired *di* are left unreplaced. If any of the *di* are multi-generic symbols, the first is paired with the maximal set of *si*.
`*u* or Mark[*u*]	*u*.
other expression	*t*[*s1*, *s2*, ...]

• Ap [7.2]

All operations represented by templates may equivalently be specified by suitable lists: templates are provided for ease of use in simple cases.

```
#I[1]::  Ap[3,{a,b}]
#O[1]:   3
#I[2]::  Ap[f[$x,$y],{a,b}]
#O[2]:   f[a,b]
#I[3]::  Ap[f[$y,$x],{a,b}]
#O[3]:   f[b,a]
#I[4]::  Ap[f[$y],{a,b}]
#O[4]:   f[a]
#I[5]::  Ap[f[$z,$y,$x],{a,b}]
#O[5]:   f[$z,b,a]
#I[6]::  Ap[f[$x,$x,$y]*g[$y^$y],{a,b}]
                          b
#O[6]:   f[a,a,b] g[b ]
#I[7]::  Ap[f[$$x],{a,b}]
#O[7]:   f[a,b]
#I[8]::  Ap[f[$$x,1,$$x],{a,b}]
#O[8]:   f[a,b,1,a,b]
#I[9]::  Ap[$f[$x],{g,y}]
#O[9]:   g[y]
#I[10]:: Ap[`(x+y),{a,b}]
#O[10]:  x + y
```

```
#I[11]::  Ap[(x+y),{a,b}]
#O[11]:   Proj[x + y,{a,b}]
#I[12]::  Ap[f,{a,b}]
#O[12]:   f[a,b]
#I[13]::  Ap[{a1,a2,a3,a4,a5},{3}]
#O[13]:   a3
#I[14]::  Ar[5,f]
#O[14]:   {f[1],f[2],f[3],f[4],f[5]}
#I[15]::  Ar[5,`f]
#O[15]:   {f,f,f,f,f}
#I[16]::  Ar[5,f[$x]]
#O[16]:   {f[1],f[2],f[3],f[4],f[5]}
#I[17]::  Ar[5,`f[$x]]
#O[17]:   {f[$x],f[$x],f[$x],f[$x],f[$x]}
#I[18]::  Ar[{2,3},$y^$x]
#O[18]:   {{1,2,3},{1,4,9}}
```

2.8 Chameleonic expressions

A chameleonic expression is an expression containing chameleonic symbols [2.2]. If a chameleonic expression is assigned as a delayed value [3.2], then each chameleonic symbol which it contains is given a unique new name whenever the value is used.
• Make [10.6]

```
#I[1]::   t::f[##a,##b]-##c
#O[1]:    ' f[##a,##b] - ##c
#I[2]::   t
#O[2]:    f[##1,##2] - ##3
#I[3]::   t-t
#O[3]:    f[##4,##5] - f[##7,##8] - ##6 + ##9
```

2.9 Commentary input

Any input (including newlines) between /* and */ is treated as commentary, and is not processed. Comments may be nested.

```
#I[1]::   a + /* begin comment
          comment1
             /*inner comment*/
          comment2*/ b+c

#O[1]:    a + b + c
```

2.10 Input forms

input form	projection	grouping
(*x*)	(parentheses)	
{*x1, x2, x3, ...*} {[*i1*]:*x1*, [*i2*]:*x2, ...*}	(List) (List)	
*x1**^*x2*	*x1* * 10^*x2*	(*x1**^*x2*)*^*x3*
@*x*	#0[*x*]	
_*x*	Prop[*x*]	
f[*x1, x2, x3*]	(projection)	f[*x1, x2, x3*] (_f[Tier]:0) ((f[*x1*])[*x2*])[*x3*] (_f[Tier]:1)
[*x1, x2, x3*]	Np[*x1, x2, x3*]	[*x1, x2, x3*]
<*x*	Input[*x*]	
x!! *x*!	Dfct![*x*] Fct![*x*]	(*x*!!)!
x1.*x2*.*x3* *x1* ** *x2* ** *x3*	Dot[*x1, x2, x3*] Omult[*x1, x2, x3*]	*x1*.*x2*.*x3* *x1* ** *x2* ** *x3*
x1^*x2*	Pow[*x1, x2*]	*x1*^(*x2*^*x3*)
-*x*	Mult[-1, *x*]	
x1/*x2*	Div[*x1, x2*]	(*x1*/*x2*)/*x3*
x1 * *x2* * *x3*	Mult[*x1, x2, x3*]	*x1* * *x2* * *x3* (see also below)
x1 + *x2* + *x3* *x1* - *x2* + *x3*	Plus[*x1, x2, x3*] Plus[*x1*, -*x2, x3*]	*x1* + *x2* + *x3* *x1* + (-*x2*) + *x3*
x1 = *x2* *x1* ~= *x2* *x1* > *x2* *x1* >= *x2* *x1* < *x2* *x1* <= *x2*	Eq[*x1, x2*] Uneq[*x1, x2*] Gt[*x1, x2*] Ge[*x1, x2*] Gt[*x2, x1*] Ge[*x2, x1*]	See note below See note below See note below See note below See note below See note below
~*x*	Not[*x*]	
x1 & *x2* & *x3*	And[*x1, x2, x3*]	*x1* & *x2* & *x3*
x1 | *x2* | *x3* *x1* || *x2* || *x3*	Or[*x1, x2, x3*] Xor[*x1, x2, x3*]	*x1* | *x2* | *x3* *x1* || *x2* || *x3*
x1 => *x2*	Imp[*x1, x2*]	*x1*=>(*x2*=>*x3*)
x1 _= *x2*	Gen[*x1, x2*]	(*x1* _= *x2*) _= *x3*
x1 .. *x2*	Seq[*x1, x2*]	(*x1* .. *x2*) .. *x3*
x1 : *x2* *x*: *x1* :: *x2* *x1* -> *x2* *x1* --> *x2* *x1* _: *x2* *x1* _*x2* *x1* := *x2* *x1* :=	Set[*x1, x2*] Set[*x*] Setd[*x1, x2*] Rep[*x1, x2*] Repd[*x1, x2*] Preset[*x1, x2*] Tyset[*x1, x2*] Sxset[*x1, x2*] Sxset[*x1*]	*x1* : (*x2* : *x3*) *x1* :: (*x2* :: *x3*) *x1* -> (*x2* -> *x3*) *x1* --> (*x2* --> *x3*) *x1* _: (*x2* _: *x3*) *x1* _(*x2* _*x3*) *x1* := (*x2* := *x3*)
x1; *x2*; *x3* *x1*; *x2*; ... *xn*;	Proc[*x1, x2, x3*] Proc[*x1, x2, ..., xn,*]	*x1*; *x2*; *x3* *x1*; *x2*; ... *xn*;
!*x* '*x* `*x*	(pre-simplification) Hold[*x*] Mark[*x*]	
?*x*	Info[*x*]	

Forms given in the same box have the same precedence; the boxes are in order of decreasing precedence.

If @@ is a form with precedence higher than ##, then *x1 @@ x2 ## x3* is treated as *(x1 @@ x2) ## x3*, while *x1 ## x2 @@ x3* is treated as *x1 ## (x2 @@ x3)*.

```
#I[1]::  x+y^2*a

                 2
#0[1]:   x + a y

#I[2]::  (x+y)^(2*a)

                 2a
#0[2]:   (x + y)

#I[3]::  a/b*c - d/e/f + k^-d*c - x^y^z

                                        z
         -d     a c        -d      y
#0[3]:   --- + --- + c k     - x
         e f     b

#I[4]::  {a+b:c, a+(b:c)}

#0[4]:   {c,a + c}

#I[5]::  {'a:b+c, (`a):b+c}

#0[5]:   {' a : b + c,2c}

#I[6]::  {@4[1], @(4[1])}

#0[6]:   {c,#0[Proj[4,{1}]]}

#I[7]::  {3+0>1, 3+(0>1)}

#0[7]:   {1,3}

#I[8]::  {k::(1;2), k::1;2}

#0[8]:   {' 1 ; 2,2}

#I[9]::  {1>0&1>0, 1>(0&1)>0}

#0[9]:   {1,0}

#I[10]::  h1_Tier

#0[10]:   Tier

#I[11]::  {h1[x][y][z], h2[x][y][z]}

#0[11]:   {Proj[Proj[h1[x],{y}],{z}],h2[x,y,z]}

#I[12]::  {-1^(1/2),a*-b,a^-b,-2!}

                    -c
#0[12]:   {-1,-a c,a  ,-2}

#I[13]::  {100*^-1!, (100*^-1)!}

#0[13]:   {18,3628800}
```

The last column of the table indicates how multiple appearances of the same form are grouped. When no parentheses appear, the corresponding projection is Flat [4.7.7]. These groupings also govern the treatment of different forms with the same precedence.

```
#I[1]:: {a/b/c, a/(b/c)}

#O[1]:  {a/b/c, a/(b/c)}

#I[2]:: {a^b^c, (a^b)^c}

#O[2]:  {a^(b^c), (a^b)^c}

#I[3]:: a:b:c

#O[3]:  c

#I[4]:: {a,b}

#O[4]:  {c,c}

#I[5]:: a_b_c_type

#O[5]:  type

#I[6]:: {S[x,x->y->z], S[x,(x->y)->z]}

#O[6]:  {y -> z,x}

#I[7]:: {3!!!, (3!)!!}

#O[7]:  {6,48}
```

Combinations of the relational operators [5] =, ~=, >, >=, <, <= may be input together; if # and ## represent input forms for two such operators, then *x1 # x2 ## x3* is treated as *(x1 # x2)* & *(x2 ## x3)*.

```
#I[1]:: x<y

#O[1]:  y > x

#I[2]:: x=y=z<3

#O[2]:  x = y & y = z & 3 > z
```

Products may be input without explicit * when their terms are suitably distinguished. For any numbers *n*, symbols *s* and *f* and expressions *x* the following input forms are taken as products:

n n	*s n*	*f[x] n*	*(x) n*	*{x} n*
ns	*s s*	*f[x] s*	*(x) s*	*{x} s*
nf[x]	*s f[x]*	*f[x] f[x]*	*(x) f[x]*	*{x} f[x]*
n(x)	*s(x)*	*f[x] (x)*	*(x) (x)*	*{x} (x)*
n{x}	*s{x}*	*f[x] {x}*	*(x) {x}*	*{x} {x}*

Precedence of such forms is as when explicit * are given.

```
#I[1]:: 22 2 3

#O[1]:  132

#I[2]:: {2a/4, a2/4}

#O[2]:  {a/2,a2/4}
```

```
#I[3]::  {3f[x]4, 3 4f[x], 34f[x]}

#O[3]:   {12f[x],12f[x],34f[x]}

#I[4]::  c(x+y) + 2(x+y)(y+z)^3 + g[x]f[x]^2

                                        3       2
#O[4]:   c (x + y) + 2(x + y) (y + z)  + f[x]  g[x]

#I[5]::  a/2b

         a b
#O[5]:   ---
          2

#I[6]::  a/(2b)

         a
#O[6]:   --
         2b

#I[7]::  a^2b^c

          2  c
#O[7]:   a  b

#I[8]::  2{{a,b},{c,d}}

#O[8]:   {{2a,2b},{2c,2d}}

#I[9]::  {a,b}{c,d}

#O[9]:   {a c,b d}
```

The symbol Null [2.2] may be input as a blank when it appears as the value of an entry in a list, or as a filter in a projection given in standard form. Hence f[] is equivalent to f[Null] and g[,,] to g[Null,Null,Null].

```
#I[1]::  Lpr[f[] + g[,,1,2,,3,]]

f[Null] + g[Null,Null,1,2,Null,3,Null]

#I[2]::  Lpr[{a,,b,c,}]

{a,Null,b,c,Null}

#I[3]::  Lpr[{}]

{}
```

2.11 Syntax modification

cc:= dd or **Sxset[cc, dd, (class:0), (prec:0)]**
assigns the name ⊂cc⊃ of the symbol *cc* to be replaced textually on input by ⊂*dd*⊃ according to one of the following classes of transformations

0 ⊂*cc*⊃ → ⊂*dd*⊃

1 ⊂*cc*⊃ *x* → ⊂*dd*⊃[*x*]

2 *x* ⊂*cc*⊃ → ⊂*dd*⊃[*x*]

3 *x1* ⊂*cc*⊃ *x2* ⊂*cc*⊃ *x3* → ⊂*dd*⊃[*x1*, *x2*, *x3*]

4 *x1* ⊂*cc*⊃ *x2* → ⊂*dd*⊃[*x1*, *x2*]
x1 ⊂*cc*⊃ *x2* ⊂*cc*⊃ *x3* → ⊂*dd*⊃[*x1*, ⊂*dd*⊃[*x2*, *x3*]]

5 *x1* ⊂*cc*⊃ *x2* → ⊂*dd*⊃[*x1*, *x2*]
x1 ⊂*cc*⊃ *x2* ⊂*cc*⊃ *x3* → ⊂*dd*⊃[⊂*dd*⊃[*x1*, *x2*], *x3*]

and with precedence *prec*. Textual replacements are performed before input expressions are simplified. Input text, excluding any strings enclosed in " ", is scanned once only from the beginning, and at each point, textual replacements for the longest possible character strings are made. Transformations 1 through 5 may carry precedences from 1 to 3: 1 is the same as Input, 2 as Plus and 3 as Info.

cc:= or Sxset[*cc*] removes any textual replacements assigned for *cc*.

Sxset[] removes all assignments for textual replacements.

• [10.8]

```
#I[1]::  in:=out

#O[1]:   out

#I[2]::  {in,in in,inin,inin in}

                  2
#O[2]:   {out,out ,outout,out outout}

#I[3]::  ")(":=ШШШ

#O[3]:   ШШШ

#I[4]::  )(^)(

            ШШШ
#O[4]:   ШШШ

#I[5]::  {2()(+1), a)(, a )(, a )( )( )()( }

                                        2
#O[5]:   {2(1 + ШШШ),aШШШ,a ШШШ,a ШШШ  ШШШШШШ}

#I[6]::  (a+b)(c+d)

#O[6]:   a + bШШШc + d

#I[7]::  (a+b) (c+d)

#O[7]:   (a + b) (c + d)

#I[8]::  (a+b")("c+d)

#O[8]:   a + d + ")(" b c
```

```
#I[9]::  "++":="@"
#O[9]:   "@"
#I[10]:: {++x,++1}
#O[10]:  {#O[x],out}
#I[11]:: ".":=x
#O[11]:  x
#I[12]:: "..":=y
#O[12]:  y
#I[13]:: "...":=z
#O[13]:  z
#I[14]:: {.,..,...,....,.....,......,.......,........,.........}
#O[14]:  {x,y,z,zx,zy,zz,zzx,zzy,zzz}
#I[15]:: "  ":="+"
#O[15]:  "+"
#I[16]:: {a b,2 3,a  b,2  3,a  b c,a   b}
#O[16]:  {a b,6,a + b,5,a + b c,a + b}
#I[17]:: "*":="^"
#O[17]:  "^"
#I[18]:: {a*b,a b,a^b}
                 b       b
#O[18]:  {a ,a b,a }
#I[19]:: "^":="*"
#O[19]:  "*"
#I[20]:: {a*b,a b,a^b,a*b*c,a^b^c}
                          c
                 b       b
#O[20]:  {a ,a b,a b,a  ,a b c}
#I[21]:: "^":=
#I[22]:: a^b
          b
#O[22]:  a
#I[23]:: Sxset[]
#I[24]:: a*b
#O[24]:  a b
#I[25]:: Sxset["++",Oplus,5,2]
#O[25]:  Oplus
#I[26]:: {a++b,a+b,a++b++c,a++b c}
#O[26]:  {Oplus[a,b],a + b,Oplus[a,Oplus[b,c]],Oplus[a,b c]}
```

```
#I[27]:: Sxset["^^",Opow,5,1]
#O[27]: Opow
#I[28]:: {a^^b,a^^b c,a^^b++c,a^^b!}
#O[28]: {Opow[a,b],c Opow[a,b],Opow[a,Oplus[b,c]],Opow[a,b]!}
```

2.12 Output forms

Most input forms are also used for output.

Parentheses are omitted in output when the required groupings follow the precedences of forms given in [2.10].

Common additional output forms:

```
Mult[x1,x2,x3]      x1 x2 x3

                          x1
Div[x1,x2]                --
                          x2

                            x2
Pow[x1,x2]                x1
```

The output form of the projection Plot [10.2] is a "plot".

Fmt and Sx are printed in a special formatted form.

Assignment of a Pr property [4] defines a special output form for a projection.

The presence of special forms in an output expression is indicated by * at the beginning of the output. The labelling of parts in such special output forms may not be manifest. A direct and unambiguous representation of any expression is printed by Lpr [10.1].

```
#I[1]:  Ps[Exp[x],x,0,4]

                      2    3    4
                     x    x    x
#O[1]:*  1 + x + ---- + -- + --
                     2    6    24

#I[2]:  Lpr[%]

Ps[1,x,0,{0,4},{[0]: 1,[1]: 1,[2]: 1/2,[3]: 1/6,[4]: 1/24}]
```

3. Fundamental operations

3.1 Automatic simplification

Any input expression is automatically "simplified" to the maximum extent possible. Unless further assignments are made, the resulting output expression can be simplified no further; if it is input again, it will be output unchanged.

```
#I[1]::  a:b
#O[1]:   b
#I[2]::  b:c
#O[2]:   c
#I[3]::  a
#O[3]:   c
#I[4]::  a+1
#O[4]:   1 + c
#I[5]::  c:d
#O[5]:   d
#I[6]::  a
#O[6]:   d
```

Even if part of an expression is apparently meaningless, no message is printed; that part is merely returned without further simplification. Processing impasses occur if simplification apparently requires infinite time or memory space [1.5].

```
#I[1]::  f[x]+0^0
                0
#O[1]:   0   + f[x]
#I[2]::  0^0:1
#O[2]:   1
#I[3]::  @1
#O[3]:   1 + f[x]
#I[4]::  f[x]:a
#O[4]:   a
#I[5]::  @1
#O[5]:   1 + a
#I[6]::  f[$x]:f[$x-1]+1
#O[6]:   1 + f[-1 + $x]
#I[7]::  f[100]
100 levels of recursion in simplifier - how many more ? 0
#O[7]:   101 + f[-1 - 98]
```

The simplification of expressions proceeds as follows:

Numbers — Ordinary numbers remain unchanged.

Symbols — A symbol is replaced by the simplified form of any value assigned [3.2] to it.

Projections

1. Unless the projector carries the property Nosmp [4], each filter is simplified in turn. (Future parallel-processing implementations may not respect this ordering.) If any filter is found to be extedned [4] with respect to the projector, then the projection is replaced or encased as specified in the relevant property list [4].
2. Projections with Flat or Reor properties [4,7.7] are cast into canonical form. Built-in simplification routines are invoked for system-defined projections.
3. If a value v has been assigned for the projector, then the filters of the projection are used in an attempt to select a part of v. If the required part is present, any necessary replacements for generic symbols [2.2] are performed, and the projection is replaced by the simplified value of the part. When v is a list, its entries are scanned sequentially until one is found whose indices match [2.6] the filters of the projection (and whose value is not Null). Flat and Reor properties are accounted for in this matching process.

Lists — Immediate values [3.2] for entries of a list are simplified in turn.

```
#I[1]:  a1:a2

#O[1]:  a2

#I[2]:  a1

#O[2]:  a2

#I[3]:  f[Pr[1],Pr[1+1],Pr[3]]

1
2
3

#O[3]:  f[1,2,3]

#I[4]:  f_:Nosmp

#O[4]:  Nosmp

#I[5]:  f[Pr[1],Pr[1+1],Pr[3]]

#O[5]:  f[' Pr[1],' Pr[1 + 1],' Pr[3]]

#I[6]:  _:[Extr,h]:h1

#O[6]:  h1

#I[7]:  h[a,b]+h[j,a]+h[j[b]]

#O[7]:  h[a,b] + h1[j[b]] + h1[j,a]

#I[8]:  j_:Flat

#O[8]:  Flat
```

```
#I[9]::  j[j[a],j[b,j[c]]]

#O[9]:   j[a,b,c]

#I[10]:: v:b^c+a+d

                    c
#O[10]:  a + d + b

#I[11]:: v[3,1]+v[5]

#O[11]:  b + v[5]

#I[12]:: v:{[0]:a, [$x]:1/$x}

                          1
#O[12]:  {[0]: a, [$x]: --}
                         $x

#I[13]:: v[0]+v[x]+v[y]

                 1   1
#O[13]:  a + --- + -
             x     y

#I[14]:: vp:{[$x]:1/$x, [0]:a}

                   1
#O[14]:  {[$x]: --, [0]: a}
                  $x

#I[15]:: vp[0]+vp[x]+vp[y]

         1   1   1
#O[15]:  - + - + -
         0   x   y

#I[16]:: a*c:t1

#O[16]:  t1

#I[17]:: Mult

#O[17]:  {[a]: {[c]: t1}}

#I[18]:: a*b*c

#O[18]:  b t1

#I[19]:: {[Pr[i1]]:Pr[v1],[Pr[i2]]::Pr[v2], [Pr[i3]]:Pr[v3]}

v1
v3

#O[19]:  {[Pr[i1]]: v1, [Pr[i2]]:: Pr[v2], [Pr[i3]]: v3}
```

Whenever an expression to which an immediate value has been assigned (through : [3.2]) is simplified, the old value is replaced by the new simplified form. Delayed values (assigned by :: [3.2]) are simplified whenever they are required, but are not replaced by the resulting simplified forms.

```
#I[1]::  a:b;b:c

#O[1]:   c

#I[2]::  ap::b

#O[2]:   ' b
```

```
#I[3]::  {a,ap}
#O[3]:   {c,c}
#I[4]::  b:d
#O[4]:   d
#I[5]::  {a,ap}
#O[5]:   {c,d}
#I[6]::  f1[$x]:D[$x,x]
#O[6]:   0
#I[7]::  f2[$x]::D[$x,x]
#O[7]:   ' D[$x,x]
#I[8]::  {f1[x^2],f2[x^2]}
#O[8]:   {0,2x}
#I[9]::  f[$x]::f[$x]:f[$x-1]+f[$x-2]
#O[9]:   ' f[$x] : f[$x - 1] + f[$x - 2]
#I[10]::  f[1]:f[0]:1
#O[10]:   1
#I[11]::  f
#O[11]:   {[1]: 1, [0]: 1, [$x]:: f[$x] : f[$x - 1] + f[$x - 2]}
#I[12]::  f[10]
#O[12]:   89
#I[13]::  f
#O[13]:   {[10]: 89, [9]: 55, [8]: 34, [7]: 21, [6]: 13, [5]: 8, [4]: 5,
                [3]: 3, [2]: 2, [1]: 1, [0]: 1,
                [$x]:: f[$x] : f[$x - 1] + f[$x - 2]}
```

When the value of a projection involves further projections from the same projector (recursion), these further projections are not immediately simplified; after one pass through the whole expression, further passes are made until the projections are completely simplified.

```
#I[1]::  g[$x]:$x g[$x-1]
#O[1]:   $x g[-1 + $x]
#I[2]::  g
#O[2]:   {[$x]: $x g[-1 + $x]}
#I[3]::  g[10]
#O[3]:   0
#I[4]::  g[1]:1
#O[4]:   1
```

```
#I[5]::  g[10]
#0[5]:   3628800
```

"Static" (manifestly non-terminating) recursive assignments in which the literal form of an expression appears in its simplified value are evaluated to one level only.

```
#I[1]::  a:a+1
#0[1]:   1 + a
#I[2]::  a
#0[2]:   1 + a
#I[3]::  f:f[x]
#0[3]:   f[x]
#I[4]::  f
#0[4]:   f[x]
```

Smp[*expr***, (***n***:Inf)]**
simplifies *expr* making at most *n* passes.

```
#I[1]::  g[$x]:$x g[$x-1]
#0[1]:   $x g[-1 + $x]
#I[2]::  Smp[g[10],2]
#0[2]:   90g[9 - 1]
```

3.2 Assignment and deassignment

Assignment is used to define a value for an expression. Simplification [3.1] replaces an expression by any value assigned to it.

Values for expressions come in two types:

Immediate — The value is simplified when it is assigned, and is maintained in a simplified form, being updated, if necessary, whenever it is used.

Delayed — The value is maintained in an unsimplified form; a new simplified form is obtained whenever it is used.

expr1*: *expr2 or **Set[*expr1*, *expr2*]**
assigns *expr2* to be the immediate value of the expression *expr1*.

expr1*:: *expr2 or **Setd[*expr1*, *expr2*]**
assigns *expr2* to be the delayed value of the expression *expr1*.

```
#I[1]::  r1:Rand[]
#O[1]:   .4768823
#I[2]::  r2::Rand[]
#O[2]:   ' Rand[]
#I[3]::  {r1,r2}
#O[3]:   {.4768823,.1359726}
#I[4]::  {r1,r2}
#O[4]:   {.4768823,.4165893}
```

Values *expr2* are assigned to expressions *expr1* of different types as follows:

Numbers — No assignment is made.

Symbols — *expr2* replaces any value previously assigned to *expr1*. No assignment is made for generic and chameleonic symbols [2.2].

Projections — First, the filters of *expr1* are simplified in turn, and any **Flat** or **Reor** properties [4] of its projector *f* are used. Then the specified part in the simplified form *v* of *f* is assigned the value *expr2*. The actions of the assignment for various types of *v* are as follows:

- Symbols — The value of *f* becomes a list (or nested set of lists) with one entry whose indices give the filters of *expr1* and whose value is *expr2*.
- Lists — If the indices of an existing entry of *v* are equivalent [2.6] to the filters of *expr1*, then the value of that entry is replaced by *expr2*. If no such entry is present, then an additional entry with indices given by the filters of *expr1* and with value *expr2* is introduced. New entries are positioned in the list immediately after any entries with more specific [2.6] indices.
- Projections — Unless the relevant filter of *expr1* is an integer, no assignment is made. If the filter corresponds to an existing part of *v*, this part is replaced by the expression *expr2*; otherwise, additional **Null** filters are introduced so as to include the specified part.

Lists

If *expr2* is a list, then each entry in *expr1* is assigned (in parallel) the value of the corresponding *expr2* entry (or Null if no such entry exists); otherwise, all entries in *expr1* are assigned the value *expr2*.

Any overall numerical coefficient in *expr1* is divided into *expr2* before assignment.

```
#I[1]::  a:1
#O[1]:   1
#I[2]::  a
#O[2]:   1
#I[3]::  a:2
#O[3]:   2
#I[4]::  a
#O[4]:   2
#I[5]::  g_:Comm
#O[5]:   Comm
#I[6]::  g[b,a]:x
#O[6]:   x
#I[7]::  g
#O[7]:   {[b]: {[2]: x}}
#I[8]::  g[a,b]
#O[8]:   x
#I[9]::  f[x,y]:axy
#O[9]:   axy
#I[10]::  f
#O[10]:   {[x]: {[y]: axy}}
#I[11]::  f[x,y]:bxy
#O[11]:   bxy
#I[12]::  f
#O[12]:   {[x]: {[y]: bxy}}
#I[13]::  f[x^2,z]:c
#O[13]:   c
#I[14]::  f
                  2
#O[14]:   {[x ]: {[z]: c}, [x]: {[y]: bxy}}
#I[15]::  f[$x,$y]:d*$y+$x
#O[15]:   $y d + $x
#I[16]::  f
                  2
#O[16]:   {[x ]: {[z]: c}, [x]: {[y]: bxy}, [$x]: {[$y]: $y d + $x}}
```

```
#I[17]::  f[$q,a^$q]:p[$q]
#O[17]:   p[$q]
#I[18]::  f
#O[18]:   {[x^2]: {[z]: c}, [x]: {[y]: bxy}, [$q]: {[2^$q]: p[$q]},
                [$x]: {[$y]: $y d + $x}}
#I[19]::  f[3,8]
#O[18]:   3 + 8d
#I[20]::  v[1]:v1
#O[20]:   v1
#I[21]::  v
#O[21]:   {v1}
#I[22]::  v[3]:v3
#O[22]:   v3
#I[23]::  v
#O[23]:   {[3]: v3, [1]: v1}
#I[24]::  v[2]:v2
#O[24]:   v2
#I[25]::  v
#O[25]:   {v1,v2,v3}
#I[26]::  v[x]:vx
#O[26]:   vx
#I[27]::  v
#O[27]:   {[x]: vx, [1]: v1, [2]: v2, [3]: v3}
#I[28]::  Log[0]
#O[28]:   Log[0]
#I[29]::  Log[0]:inf
#O[29]:   inf
#I[30]::  Log
#O[30]:   {[0]: inf}
#I[31]::  a+b:2
#O[31]:   2
#I[32]::  Plus
#O[32]:   {[b]: {[2]: 2}}
#I[33]::  a+b+c
#O[33]:   c + 2
#I[34]::  t:r[x,y^2,b*x]
```

```
                 2
#O[34]:   r[x,y ,b x]

#I[35]::  t[2][1]:c

#O[35]:   c

#I[36]::  t

                 2
#O[36]:   r[x,c ,b x]

#I[37]::  t[3,8]:Pow

#O[37]:   Pow

#I[38]::  t

                 2  x
#O[38]:   r[x,c ,b ]

#I[39]::  t[2,-1]:3/2

#O[39]:   3/2

#I[40]::  t

                    2
                3 c    x
#O[40]:   r[x,----,b ]
                2

#I[41]::  t[4]:e4

#O[41]:   e4

#I[42]::  t

                    2
                3 c    x
#O[42]:   r[x,----,b ,e4]
                2

#I[43]::  t[7]:e7

#O[43]:   e7

#I[44]::  t

                    2
                3 c    x
#O[44]:   r[x,----,b ,e4,,,e7]
                2

#I[45]::  t[3,8]:Plus

#O[45]:   t[3,8] : {[b]: {[2]: 2}}

#I[46]::  t[3,8]::Plus

#O[46]:   ' Plus

#I[47]::  t

                    2
                3 c
#O[47]:   r[x,----,b + x,e4,,,e7]
                2
#I[48]::  p:1;q:2

#O[48]:   2
```

```
#I[49]::  {a,p,q}:{5,q,p}
#O[49]:   {5,2,1}
#I[50]::  {a,p,q}
#O[50]:   {5,2,1}
#I[51]::  a:b:c:8
#O[51]:   8
#I[52]::  {a,b,c}
#O[52]:   {8,8,8}
#I[53]::  {a,b,c}:7
#O[53]:   7
#I[54]::  {a,b,c+a}
#O[54]:   {7,7,14}
#I[55]::  {a,b,c}:{4}
#O[55]:   {4}
#I[56]::  {a,b,c,a+b,b+c}
#O[56]:   {7,b,c,7 + b,b + c}
```

expr: or *expr*:Null or Set[*expr*] or Set[*expr*,Null]
removes any values assigned for the literal expression *expr*. If filters in projections are removed, the projections are correspondingly shortened. "Removal" of a projector results in its replacement by Np [2.8]. If *expr* is a list, values for each entry are removed. *_symb*: removes properties [4] assigned to the symbol *symb* (restoring any initial properties if *symb* is system-defined).
• Del [7.3]

Set[]
removes values assigned to all expressions.

```
#I[1]::   a:b:3
#O[1]:    3
#I[2]::   {a,b,c}
#O[2]:    {3,3,c}
#I[3]::   a:
#I[4]::   {a,b,c}
#O[4]:    {a,3,c}
#I[5]::   f[x]:x^2;f[y]:4;f[$x]:$x^3
#O[5]:    $x^3
#I[6]::   f
#O[6]:    {[y]: 4, [x]: x^2, [$x]: $x^3}
```

```
#I[7]::  f[x]:
#I[8]::  f
#O[8]:   {[y]: 4, [$x]: $x^3}
#I[9]::  f[$y]:
#I[10]::  f
#O[10]:   {[y]: 4, [$x]: $x^3}
#I[11]::  f[$x]:
#I[12]::  f
#O[12]:   {[y]: 4}
#I[13]::  f:
#I[14]::  f
#O[14]:   f
#I[15]::  f[x]
#O[15]:   f[x]
#I[16]::  t:r[1,2,3,4,5]
#O[16]:   r[1,2,3,4,5]
#I[17]::  t[2]:
#I[18]::  t
#O[18]:   r[1,3,4,5]
#I[19]::  t[0]:
#I[20]::  t
#O[20]:   [1,3,4,5]
#I[21]::  a:b:c:4
#O[21]:   4
#I[22]::  {a,b,c}
#O[22]:   {4,4,4}
#I[23]::  {a,b}:
#I[24]::  {a,b,c}
#O[24]:   {a,b,4}
#I[25]::  Set[]
#I[26]::  {a,b,c}
#O[26]:   {a,b,c}
```

Inc[*expr*, (*step*:1)]
increments the value of *expr* by *step*.
• Do [6.2]

Dec[*expr*, (*step*:1)]
decrements the value of *expr* by *step*.

```
#I[1]::  i:1;x:a;
#I[2]::  Dec[i,6]
#O[2]:   -5
#I[3]::  i
#O[3]:   -5
#I[4]::  Inc[x]
#O[4]:   1 + a
#I[5]::  x
#O[5]:   1 + a
```

Assignment and deassignment projections have the special capability to effect permanent changes on their filters; all other projections must leave their filters unchanged.

3.3 Replacements and substitutions

Values assigned for expressions [3.2] are used whenever they are applicable. More controlled evaluation is provided by the use of substitution projections.

expr1 -> expr2 or **Rep[*expr1*, *expr2*, (*levspec*:Inf), (*rpt*:Inf), (*max*:Inf)]**
is a purely syntactic construct which represents a replacement of *expr1* by *expr2*. The replacement is specified to be active when used in S projection substitutions on an expression *expr* only for the first *max* occurrences of *expr1* appearing at or below level *lev* in *expr*, and during the first *rpt* passes through *expr*. *expr1->expr2->expr3* is equivalent to *expr1->expr3*.

expr1*--> *expr2 or **Repd[*expr1*, *expr2*, (*levspec*:Inf), (*rpt*:Inf), (*max*:Inf)]**
represents a replacement in which *expr2* is maintained in an unsimplified form; a new simplified form is obtained whenever the replacement is performed.

S[*expr*,{*rep1*, *rep2*, ...}, (*rpt*:1), (*levspec*:Inf), (*domcrit*:1)]
performs substitutions in *expr* specified by the replacements *rep1*,*rep2*,... in the domain defined by *levspec* and *domcrit* [2.5]. Each successive subpart of *expr* is compared with the left members of each active replacement *repi* in turn; if a match [2.6] is found, then the part is replaced by the corresponding right member, with any necessary substitutions for generic symbols made. The resulting complete expression is scanned until no further replacements can be used, or at most *rpt* times. (*rpt* may be Inf). When a replacement is used, the resulting subexpression is not scanned for possible further substitutions until it is reached on at least the next pass through the complete expression.

```
#I[1]:  S[x^2+y^2,x->a,y->1]

                 2
#O[1]:  1 + a

#I[2]:  S[3x-x(x+1),x->2b-2]

#O[2]:  -6 + 8b - (-2 + 2b) (-1 + 2b)

#I[3]:  S[{[i]:3i-2,[2i+1]:5i},2i->3]

#O[3]:  {[i]: 5/2, [2 i + 1]: 15/2}

#I[4]:  gt:g[3/2]+g[4]+g[x]

#O[4]:  g[3/2] + g[4] + g[x]

#I[5]:  r1:g[$x]->f1[$x]

#O[5]:  g[$x] -> f1[$x]

#I[6]:  r2:g[$x_:Natp[$x]]->f2[$x]

#O[6]:  g[$x _: Natp[$x]] -> f2[$x]

#I[7]:  r3:g[$x_:Numbp[$x]]->f3[$x]

#O[7]:  g[$x _: Numbp[$x]] -> f3[$x]

#I[8]:  S[gt,r1]

#O[8]:  f1[3/2] + f1[4] + f1[x]

#I[9]:  S[gt,r2]

#O[8]:  f2[4] + g[3/2] + g[x]
```

```
#I[10]:: S[gt,r2,r1]
#0[10]:  f1[3/2] + f1[x] + f2[4]
#I[11]:: S[gt,r3,r2,r1]
#0[11]:  f1[x] + f3[3/2] + f3[4]
#I[12]:: S[gt,r2,r3,r1]
#0[12]:  f1[x] + f2[4] + f3[3/2]
#I[13]:: S[x^2,x->x+1]
                  2
#0[13]:  (1 + x)
#I[14]:: S[x^2,x->x+1,4]
                  2
#0[14]:  (4 + x)
#I[15]:: S[f[x]+f[y],f[x]->1,f[$x]->$x^2]
                2
#0[15]:  1 + y
#I[16]:: rf:f[$x]->f[$x-1]+f[$x-2]
#0[16]:  f[$x] -> f[-2 + $x] + f[-1 + $x]
#I[17]:: S[f[4],rf]
#0[17]:  f[2] + f[3]
#I[18]:: S[f[4],rf,4]
#0[18]:  f[-4] + 4f[-3] + 6f[-2] + 4f[-1] + f[0]
#I[19]:: rfe:{f[1]->1,f[0]->1}
#0[19]:  {f[1] -> 1,f[0] -> 1}
#I[20]:: S[f[4],rf,rfe,4]
#0[20]:  f[-4] + 4f[-3] + 5f[-2] + 4f[-1] + f[0]
#I[21]:: S[f[4],rfe,rf,4]
#0[21]:  5
#I[22]:: S[f[4],rfe,rf,Inf]
#0[22]:  5
#I[23]:: ht:S[x,$x->h[$x],5]
#0[23]:  h[h[h[h[h[x]]]]]
#I[24]:: hr:h[$x]->y[$x]
#0[24]:  h[$x] -> y[$x]
#I[25]:: S[ht,hr]
#0[25]:  y[h[h[h[h[x]]]]]
#I[26]:: S[ht,hr,2]
#0[26]:  y[y[h[h[h[x]]]]]
#I[27]:: S[ht,hr,-1,1]
```

```
#O[27]:   h[h[h[h[h[x]]]]]
#I[28]::  S[ht,hr,-2,1]
#O[28]:   h[h[h[h[y[x]]]]]
#I[29]::  S[ht,hr,-2,2]
#O[29]:   h[h[h[h[y[x]]]]]
#I[30]::  S[ht,hr,-3,1]
#O[30]:   h[h[h[h[y[x]]]]]
#I[31]::  S[ht,hr,-3,2]
#O[31]:   h[h[h[y[y[x]]]]]
#I[32]::  S[ht,hr,-4,3]
#O[32]:   h[h[y[y[y[x]]]]]
#I[33]::  hrp:Rep[h[$x],yp[$x],1]
#O[33]:   h[$x] -> yp[$x] -> 1
#I[34]::  S[ht,hrp]
#O[34]:   yp[h[h[h[h[x]]]]]
#I[35]::  S[ht,hrp,2]
#O[35]:   yp[yp[h[h[h[x]]]]]
#I[36]::  S[ht,hrp,3]
#O[36]:   yp[yp[h[h[h[x]]]]]
#I[37]::  S[ht,hrp,3,3]
#O[37]:   yp[yp[h[h[h[x]]]]]
#I[38]::  S[ht,hrp,-3,1]
#O[38]:   h[h[h[h[h[x]]]]]
#I[39]::  S[ht,h->y,1,1]
#O[39]:   y[h[h[h[h[x]]]]]
#I[40]::  S[ht,h->y,-1,1]
#O[40]:   y[h[h[h[h[x]]]]]
#I[41]::  rfn:Rep[f[$x],f[$x-1]+fn,Inf,2]
#O[41]:   f[$x] -> fn + f[-1 + $x] -> Inf -> 2
#I[42]::  rgn:g[$x]->g[$x-1]+gn
#O[42]:   g[$x] -> gn + g[-1 + $x]
#I[43]::  S[f[10],rfn,rgn,6]
#O[43]:   f[10]
#I[44]::  rk:k[$x]-->(Pr[$x];$x^2)
#O[44]:   k[$x] --> (Pr[$x] ; $x^2)
#I[45]::  tk:Ar[4,k]
```

```
#O[45]:   {k[1],k[2],k[3],k[4]}

#I[46]::  S[tk,rk]

1
2
3
4
#O[46]:   {1,4,9,16}
```

Many relations involving mathematical functions are given in external files [1.3] in the form of replacements, to be applied using S.

Si [*expr1*,{*rep1*, *rep2*, . . .}, (*levspec*:Inf), (*domcrit*:1)]
is equivalent to S[*expr*,{*rep1*, *rep2*, . . .}, Inf , (*levspec*:Inf), (*domcrit*:1)]
; the *repi* are repeatedly used in *expr* until the result no longer changes.

Arep [{*rep1*, *rep2*, . . .}]
performs explicit assignments using Set or Setd projections [3.2] on the Rep or Repd replacements *repi*.

```
#I[1]::   r:a->b+c

#O[1]:    a -> b + c

#I[2]::   a

#O[2]:    a

#I[3]::   Arep[r]

#O[3]:    b + c

#I[4]::   a

#O[4]:    b + c
```

Irep [{*rep1*, *rep2*, . . .}]
yields a list of "inverted" replacements.

```
#I[1]::   f[$x]->a^$x

                    $x
#O[1]:    f[$x] -> a

#I[2]::   Irep[%]

           $x
#O[2]:    a   -> f[$x]
```

3.4 Numerical evaluation

N[*expr*, (*acc*:6), (*trunc*:10*^-12)]
<ldist>
yields the numerical value of *expr* in terms of real or complex decimal numbers, maintaining if possible an accuracy of *acc* significant figures, and setting all numerical coefficients smaller than *trunc* to zero.

A , F and C× projections [2.1] are generated when required.

Numerical values for arbitrary expressions may be defined by assignments [3.2] for the corresponding N[*expr*] *or* N[*expr*, *n*] .

```
#I[1]::  N[Gamma[3.2]]

#O[1]:   2.423965

#I[2]::  N[Gamma[3.2],12]

#O[2]:   2.423965479935

#I[3]::  N[(1.2+5I)^(6+7I)]

#O[3]:*  1.308713 + .9422779 I

#I[4]::  N[Log[2+2I]-Sin[Exp[1+Asin[2I]]],10]

#O[5]:*  -1.476712 + -6.159398 I

#I[6]::  Pi

#O[6]:   Pi

#I[7]::  N[Pi]

#O[7]:   3.141593

#I[8]::  N[Pi2]:1.1124553503148

#O[8]:   1.112455

#I[9]::  Pi2^2+5Pi

           2
#O[9]:   Pi2  + 5Pi

#I[10]::  N[%]

#O[10]:   16.94552

#I[11]::  N[f[$x_?Numbp[$x]]]::N[Sum[$x^2/i^5,i,1,10]]

                    2
                 $x
#O[11]:   ' N[Sum[---,i,1,10]]
                   5
                  i

#I[12]::  N[f[x]+a*f[2]+b*f[3+4I]+c*f[2-Sin[17Deg]]]

#O[16]:*  (f[x] + 3.023616a - 7.258351b + 3.023616c) + 24.88578b I

#I[17]::  u:g[0.1,0.001,0.0001,0.000001]

#O[17]:   g[.1,.001,.0001,1.*^-05]

#I[18]::  N[u,,8.001]
```

```
#O[18]:   g[.1,.001,0,0]

#I[19]::  N[u,0.1,0.001]

#O[19]:   g[.1,.001,0,0]
```

Neq[*expr1*, *expr2*, (*acc*:1*^-8), (*n*:2), (*range*:{-10,10})]
tests for numerical equality of *expr1* and *expr2* within fractional accuracy *acc* by evaluating them at least *n* times with random numerical choices (in the specified *range*) for all symbolic parameters appearing in them.
• Eq [5]

```
#I[1]::   e1:(x+y)^3

                  3
#O[1]:    (x + y)

#I[2]::   e2:Ex[e1]

                2       2       3     3
#O[2]:    3x y  + 3 x  y + x  + y

#I[3]::   e1=e2

                  3       2       2       3     3
#O[3]:    (x + y)  = 3x y  + 3 x  y + x  + y

#I[4]::   Neq[e1,e2]

#O[4]:    1

#I[5]::   Neq[e1+1,e2]

#O[5]:    0

#I[6]::   Neq[f[x],f[x+1]]

#O[6]:    Neq[f[x],f[1 + x]]
```

• D, Int [9.4]
• Sum, Prod [9.2]

3.5 Deferred simplification

'*expr* or **Hold[*expr*]**
yields an expression entirely equivalent to *expr* but all of whose parts are "held" in an unsimplified form, until "released" by **Rel**.
(Prefix form is "forward-quote" or "acute accent" character.)

Rel[*expr*]
releases all "held" parts of *expr* for simplification.

• Ev [3.7]

Projectors in simplified projections are maintained in "held" form.

Nosmp [4] filters in projections are converted to "held" form.

Use of Hold projections allows expressions to be treated by "name" rather than by "value".

```
#I[1]::  t:'1+a+3
#O[1]:   ' 1 + a + 3
#I[2]::  t[2]
#O[2]:   ' a
#I[3]::  Rel[t]
#O[3]:   4 + a
#I[4]::  t[3]:4
#O[4]:   4
#I[5]::  t
#O[5]:   ' 1 + a + 4
#I[6]::  u:f[x]
#O[6]:   f[x]
#I[7]::  u[8]
#O[7]:   ' f
#I[8]::  S[u,f->t]
#O[8]:   Proj[' 1 + a + 4,{x}]
#I[9]::  S[u,f->'t]
#O[9]:   t[x]
#I[10]::  j_:Nosmp
#O[10]:   Nosmp
#I[11]::  j[1+1,t]
#O[11]:   j[' 1 + 1,' t]
```

3.6 Pre-simplification

! *expr*
represents the simplified form of *expr* regardless of its environment.

Simplifications indicated by ! are performed during input.

! may be used to insert simplified forms as filters in Nosmp [4] projections.

! denotes pre-simplification only if it is not the first character of an input line [1.6].

```
#I[1]::  f_:Nosmp
#O[1]:   Nosmp
#I[2]::  f[!1+1,1+1]
#O[2]:   f[' 2,' 1 + 1]
#I[3]::  a:b
#O[3]:   b
#I[4]::  a:c
#O[4]:   c
#I[5]::  {a,b,c}
#O[5]:   {c,b,c}
#I[6]::  (!a):d
#O[6]:   d
#I[7]::  {a,b,c}
#O[7]:   {d,b,d}
#I[8]::  Pr[1];!Pr[2]
2

1
#O[8]:   2
#I[9]::  j:$x^2
                 2
#O[9]:   $x
#I[10]::  Ar[4,j]
#O[10]:   {$x,2,j[3],j[4]}
#I[11]::  Ar[4,!j]
#O[11]:   {1,4,9,16}
```

3.7 Partial simplification

Ev[*expr*, (*k*:1)]
yields the result of *k* partial evaluations on the held [3.5] form of *expr*. In each partial evaluation, symbols and projections are replaced by the held forms of any values assigned to them.

```
#I[1]::  a:b
#O[1]:   b
#I[2]::  b:c
#O[2]:   c
#I[3]::  Ev[a]
#O[3]:   ' b
#I[4]::  Ev[#I[1]]
#O[4]:   ' a : b
```

4. Properties

_symb or **Prop[*symb*]**
is a list giving properties of the symbol *symb* used to specify treatment of *symb* or its projections.

The operations of projection [2.3,7.3], assignment [3.2] and deassignment [3.2] may be applied to *_symb* just as to symbols.

***_symb*:** causes the properties of a system-defined symbol *symb* to revert to their initial form [3.2].

```
#I[1]:: _a
#O[1]:  _a
#I[2]:: _a:{[p1]:1,[p2]:0}
#O[2]:  {[p1]: 1, [p2]: 0}
#I[3]:: _a[p2]
#O[3]: 0
#I[4]:: _a[p3]:x
#O[4]:  x
#I[5]:: _a
#O[5]:  {[p3]: x, [p1]: 1, [p2]: 0}
#I[6]:: _a[p1]:
#I[7]:: _a
#O[7]:  {[p3]: x, [p2]: 0}
#I[8]:: _Dot
#O[8]:  {[Tier]: 1, [Flat]: 1, [Sys]: 1}
#I[9]:: _Dot[Comm]:1
#O[9]:  1
#I[10]:: _Dot
#O[10]:  {[Comm]: 1, [Tier]: 1, [Flat]: 1, [Sys]: 1}
#I[11]:: _Dot:
```

A symbol is considered to "carry" a particular property *p* if the value of *_symb*[*p*] is not "false" [5].

Properties such as `Flat` which affect treatment of assignments for projections must be defined before the projections are assigned.

The following are system-defined properties for a symbol *s* :

Sys	System-defined symbol [2.2].
Gen	Generic symbol [2.2], representing the class of expressions on which application of the template *_s*[Gen] yields a non-zero number [2.6].
Mgen	Multi-generic symbol [2.2].

Cham Chameleonic symbol [2.2].

Init Any value assigned for _*s*[Init] is simplified, and then removed, when *s* next appears as a projector.

Tier The projections s[x][y] and s[x,y] are distinguished [2.3].

Nosmp The *i*th filter in a projection from *s* is automatically simplified [3.1] if _*s*[Nosmp][*i*] is not "false", and is otherwise placed in a "held" form [3.5].

Pr The value of _*s*[Pr] is used as the print form for *s* and its projections.
• Fmt, Sx [10.1]

Trace If the value of _*s*[Trace] is "true", then any projections from *s* are printed before evaluation. Any other values are applied as templates to projections from *s*.
• [10.7]

Flat All projections from *s* are flattened [7.7], so that *s* is treated as an "associative n-ary function".

Reor Filters of projections from *s* are placed in canonical order, by reordering them using permutation symmetries given as the value of _*s*[Reor] [7.7].

Comm Equivalent to _*s*[Reor]:Sym [7.7]. Causes filters of projections from *s* to be placed in canonical order, so that *s* represents a "commutative function".

Dist _*f*[Dist]::{*g1*,*g*(*h1*:*g1*),(*k1*:*f1*),(*pspec1*:Inf)} defines *f* to be distributive over projections from *gi* appearing as its filters in positions specified by *pspeci*, yielding projections of *hi* and *ki*. The definitions are used as defaults in Ex [7.8].

Powdist _*f*[Powdist]::{*g1*,(*h1*:Plus)} defines projections of *f* to be "power expandable" over projections of *gi* appearing as their first filters, and yielding projections from *hi*. The definitions are used as defaults in Ex [7.8].

Ldist Projections from *s* are "distributed" over the entries of lists appearing as filters in projections from *s* [7.7].

Const *s* is treated as a numerical constant.

Extr The projector *f* of a projection containing *s* (as an isolated symbol or as a projector) in its filters is replaced by a template given as the value of _*s*[Extr,*f*] (see below).

Exte The value of _*s*[Exte] is applied as a template to any projection whose filters involve *s* (as an isolated symbol or as a projector), and for whose projector *f* no entry _*s*[Extr,*f*] exists (see below). Exte is also effective for entries in lists.

Type *s* is treated as carrying the additional properties assigned to a symbol given as the value of _*s*[Type]. These additional properties are considered only if properties given directly for *s* are inadequate. Any number of Type indirection levels may be used.

• Ser [1.11]
• Cons [10.9]

Some additional properties used for special purposes are described below.

All system-defined symbols carry the property Tier.

```
#I[1]::  _$x
#O[1]:   {[Gen]: 1}
#I[2]::  _$x[Gen]:Intp
#O[2]:   Intp
#I[3]::  f[$x]:$x^2
                2
#O[3]:   $x
#I[4]::  f[x]+f[3]
#O[4]:   9 + f[x]

#I[1]::  _f[Init]::Pr["Using f"]
#O[1]:   ' Pr["Using f"]
#I[2]::  f
#O[2]:   f
#I[3]::  f[x]+f[y]
"Using f"
#O[3]:   f[x] + f[y]
#I[4]::  _f
#O[4]:   _f

#I[1]::  f[x][y]
#O[1]:   f[x,y]
#I[2]::  f_:Tier
#O[2]:   Tier
#I[3]::  f[x][y]
#O[3]:   Proj[f[x],{y}]

#I[1]::  g_:Nosmp
#O[1]:   Nosmp
#I[2]::  _b[Nosmp]:{1,0,0}
#O[2]:   {1,0,0}
#I[3]::  _j[Nosmp][$x]::Evenp[$x]
#O[3]:   ' Evenp[$x]
#I[4]::  {f[1+1,1+1], g[1+1,1+1,1+1], h[1+1,1+1,1+1,1+1], i[1+1,1+1,1+1,1+1,1+1]}
#O[4]:   {f[2,2],g[' 1 + 1,' 1 + 1,' 1 + 1],h[' 1 + 1,2,2,' 1 + 1],
                i[2,' 1 + 1,2,' 1 + 1,2]}
```

```
#I[1]::  _f[Pr]:a

#O[1]:   a

#I[2]::  f^2+f[x]

          2
#O[2]:*  a  + f[x]

#I[3]::  Lpr[%]

f^2 + f[x]

#I[4]::  _q[Pr]:Fmt[{{0,0},{0,1}},"[]","||"]

         ||
#O[4]:   []

#I[5]::  q+f[q^q]

                ||
         ||     ||[]
#O[5]:*  [] + f[[]  ]

#I[6]::  _g[Pr][$x,$y]:Fmt[{{0,0},{0,-1},{1,-1}},$x,";",$y]

#O[6]:   $x
         ; $y

#I[7]::  {g[a+b,c],g[x^2+1,x/(x+y)],g[3]}

                          2
#O[7]:*  {a + b ,1 + x      ,g[3]}
          ;    c         x
                 ;     -----
                       x + y


#I[1]::  f_:Trace

#O[1]:   Trace

#I[2]::  f[f[x]]+f[y]

f[x]

f[f[x]]

f[y]

#O[2]:   f[y] + f[f[x]]

#I[3]::  _g[Trace]::Pr[Len[$x]]

#O[3]:   ' Pr[Len[$x]]

#I[4]::  g[g[1,2,3],1]+g[4]

3
2
1

#O[4]:   g[4] + g[g[1,2,3],1]

#I[5]::  Set_:Trace

#O[5]:   Trace
```

```
#I[6]::  a:b:c
a : (b : c)
b : c
#O[6]:   c

#I[1]::  f_Flat
#O[1]:   Flat
#I[2]::  f[f[a,b],f[c+f[d,e],f[f[c]]],e]
#O[2]:   f[a,b,c + f[d,e],c,e]
#I[3]::  g_Comm
#O[3]:   Comm
#I[4]::  g[b,a]+g[d,c]+g[c,d]
#O[4]:   g[a,b] + 2g[c,d]
#I[5]::  _h[Reor]:Asym
#O[5]:   Asym
#I[6]::  {h[b,a],h[a,b],h[a,a],h[c,b,a],h[a,c,b],h[a,b,a]}
#O[6]:   {-h[a,b],h[a,b],0,-h[a,b,c],-h[a,b,c],0}
#I[7]::  _j[Reor]:{Cyclic[1,2,3],Asym[4,5]}
#O[7]:   {Cyclic[1,2,3],Asym[4,5]}
#I[8]::  {i[b,a,c,e,d],i[b,b,a,d,e],i[b,c,a,e,d]}
#O[8]:   {-i[a,c,b,d,e],i[a,b,b,d,e],-i[a,b,c,d,e]}


#I[1]::  _f[Dist]::Plus
#O[1]:   ' Plus
#I[2]::  Ex[f[a+f[b+c]]]
#O[2]:   f[a] + f[f[b]] + f[f[c]]
#I[3]::  _f[Dist]::{Mult,Dot}
#O[3]:   ' {Mult,Dot}
#I[4]::  Ex[f[a b c]+f[u.v.w]+f[x+y]]
#O[4]:   f[u].f[v].f[w] + f[a] f[b] f[c] + f[x + y]
#I[5]::  _j[Dist]::{{g,h,k}}
#O[5]:   ' {{g,h,k}}
#I[6]::  t:j[g[a,b],gp[x,y],g[c,d,e]]
#O[6]:   j[g[a,b],gp[x,y],g[c,d,e]]
#I[7]::  Ex[t]
#O[7]:   h[k[a,gp[x,y],g[c,d,e]],k[b,gp[x,y],g[c,d,e]]]
```

```
#I[8]::  _j[Dist]::{{g,h,j},gp}
#O[8]:   ' {{g,h,j},gp}
#I[9]::  Ex[t]
#O[9]:   h[h[gp[j[a,x,c],j[a,y,c]],gp[j[a,x,d],j[a,y,d]],
             gp[j[a,x,e],j[a,y,e]]],
           h[gp[j[b,x,c],j[b,y,c]],gp[j[b,x,d],j[b,y,d]],
             gp[j[b,x,e],j[b,y,e]]]]


#I[1]::  _f[Powdist]::Mult
#O[1]:   ' Mult
#I[2]::  Ex[f[a+b]^3]
                    3
#O[2]:   f[a + b]
#I[3]::  _f[Powdist]::{{Mult,Plus},{Dot,g}}
#O[3]:   ' {{Mult,Plus},{Dot,g}}
#I[4]::  Ex[f[a+b+c,2]]
                                  2    2    2
#O[4]:   2a b + 2a c + 2b c + a  + b  + c
#I[5]::  Ex[f[g[a,b],3]]
#O[5]:   g[a.a.a,a.a.b,a.b.a,a.b.b,b.a.a,b.a.b,b.b.a,b.b.b]


#I[1]::  f_;Ldist
#O[1]:   Ldist
#I[2]::  f[{a1,a2,a3},{b1,b2},c]
#O[2]:   {f[a1,b1,c],f[a2,b2,c],f[a3,c]}
#I[3]::  f[{{1,2},{3,4}},c,{a1,a2,a3}]
#O[3]:   {{f[1,c,a1],f[2,c,a1]},{f[3,c,a2],f[4,c,a2]},f[c,a3]}
#I[4]::  f[{[x]:ax,[y]:ay},{[z]:bz,[x]:bx},c]
#O[4]:   {[x]: f[ax,bx,c], [y]: f[ay,c], [z]: f[bz,c]}
#I[5]::  {4,5,3}+a{3,4,{2,5}}
#O[5]:   {4 + 3a,5 + 4a,{3 + 2a,3 + 5a}}
#I[6]::  r[$x]:$x+3
#O[6]:   3 + $x
#I[7]::  r
#O[7]:   {[$x]: 3 + $x}
#I[8]::  rp:r^2
```

```
#O[8]:   {[$x]: (3 + $x)^2}

#I[9]::  rp[x]

#O[9]:   (3 + x)^2


#I[1]::  f_:Nosmp

#O[1]:   Nosmp

#I[2]::  f[1+1] + g[1+1]

#O[2]:   f[' 1 + 1] + g[2]

#I[3]::  g_:f

#O[3]:   f

#I[4]::  _g

#O[4]:   {[Type]: f}

#I[5]::  f[1+1] + g[1+1]

#O[5]:   f[' 1 + 1] + g[' 1 + 1]
```

symb ***_: p*** or Prset[*symb*, *p*]
is equivalent to *_symb*[*p*]:1 and assigns the property *p* to the symbol *symb*.

symb ***_st*** or Tyset[*symb*, *st*]
is equivalent to *_symb*[Type]::*st* and assigns the symbol *symb* to have the type of the symbol *st*.

```
#I[1]::  f_:p1

#O[1]:   p1

#I[2]::  _f

#O[2]:   {[p1]: 1}

#I[3]::  f_:g_:p2

#O[3]:   p2

#I[4]::  _f

#O[4]:   {[p2]: 1, [p1]: 1}

#I[5]::  f_g_:1

#O[5]:   t1

#I[6]::  _f

#O[6]:   {[Type]: t1, [p2]: 1, [p1]: 1}
```

Entries *_s*[Extr] and *_s*[Exte] in the property list of a symbol *s* allow for "type extension". Standard projections may be replaced by special projections if some of their filters are of some special extended type. Hence, for example, a product may be entered as a projection of Mult, but is replaced by a projection of Psmult if one or more of its filters is a power series (Ps projection [9.5]). Projections reached by

type extension are usually not described separately in this manual.

```
#I[1]::  _f[Extr,g]:fg
#O[1]:   fg
#I[2]::  _f[Extr,Mult]:fmult
#O[2]:   fmult
#I[3]::  g[f[x],y]+g[x,y]*f[a,b]+c*a*b*f
#O[3]:   a c fmult[b,f] + fg[f[x],y] + fmult[g[x,y],f[a,b]]
#I[4]::  _f[Exte]:fgen
#O[4]:   fgen
#I[5]::  g[f[x],y]+f[f[x],y]
#O[5]:   fg[f[x],y] + fgen[f[f[x],y]]
#I[6]::  j_f
#O[6]:   f
#I[7]::  a*f+b*j
#O[7]:   fmult[a,f] + fmult[b,j]
#I[8]::  {f[1],f[2]}
#O[8]:   fgen[{f[1],f[2]}]
```

5. Relational and logical operations

An expression is treated as "false" if it is zero, and "true" if it is any non-zero number.

P[*expr*]
yields 1 if *expr* is "true", and 0 otherwise.

The relational and logical projections described below yield 0 or 1 if their "truth" or "falsity" can be determined (by syntactic comparison or linear elimination of symbolic parameters); otherwise they yield simplified forms of undetermined truth value. When only one filter is given, the relational projections defined below yield as images that filter. When more than two filters are given, they yield the conjunction of results for each successive pair of filters.

expr1* = *expr2 or **Eq[*expr1*, *expr2*]**
<Comm>
represents an equation asserting the equality of *expr1* and *expr2*. Equations represented by Eq projections are used in Sol [9.3].
• Neq [9.6]

expr1* ~= *expr2 or **Uneq[*expr1*, *expr2*]**
<Comm>
asserts the inequality of *expr1* and *expr2*.

expr1* > *expr2 or **Gt[*expr1*, *expr2*]**
asserts that *expr1* is numerically larger than *expr2*.

expr1* >= *expr2 or **Ge[*expr1*, *expr2*]**
asserts that *expr1* is numerically larger than or equal to *expr2*.

expr1* < *expr2
is equivalent to *expr2* > *expr1*

expr1* <= *expr2
is equivalent to *expr2* >= *expr1*.

The assignment [3.2] `expr1 > expr2 : 1` defines the expression `expr1` to be greater than `expr2`. The projection Ge tests for assignments of relevant Gt projections.

If # and ## represent special input forms for relational projections then *expr1 # expr2 ## expr3* is converted to *(expr1 # expr2)* & *(expr2 ## expr3)* [2.10].

```
#I[1]:  {2+1=3,x=1,P[x=1],P[2+1=3]}
#O[1]:  {1,x = 1,0,1}
#I[2]:  {Eq[2],Uneq[2],2~=2+1,Uneq[1,2,1],Uneq[1,3,-2,4]}
#O[2]:  {2,2,1,0,1}
#I[3]:  {x<y,x+3>x+2,x+a<x+b,2x+a>x-3a}
#O[3]:  {y > x,1,b > a,4a + x > 0}
#I[4]:  {x>y>z,x>y>=z,x=y=z}
#O[4]:  {x > y & y > z,y >= z & x > y,x = y & y = z}
#I[5]:  a>0:1
#O[5]:  1
```

```
#I[6]:: {a>0,a<0,0<a,a>=0,a>1}
#O[6]:  {1,0 > a,1,1,a > 1}
#I[7]:: f[$x]>f[$y]:$x>$y
#O[7]:  $x > $y
#I[8]:: {f[3]>f[2],f[x]>f[y],f[x+a]>f[x],f[x+b]>f[x]}
#O[8]:  {1,x > y,1,b > 0}
```

~*expr* or **Not[*expr*]**
yields 1 if *expr* is 0 and 0 if *expr* is "true".

***expr1* & *expr2* & *expr3*...**or **And[*expr1*, *expr2*, *expr3*, ...]**
<Comm,Flat>
forms the conjunction of the *expri*.

***expr1* | *expr2* | *expr3*...**or **Or[*expr1*, *expr2*, *expr3*, ...**
<Comm,Flat>
forms the inclusive disjunction of the *expri*.

***expr1* || *expr2* || *expr3*...**or **Xor[*expr1*, *expr2*, *expr3*, ...]**
<Comm,Flat>
forms the exclusive disjunction of the *expri*.

expr1* => *expr2 or **Imp[*expr1*, *expr2*]**
represents the logical implication "if *expr1* then *expr2*".

```
#I[1]:: 1 & 0 | 4>3
#O[1]:  1
#I[2]:: Outer[f,{1,0},{1,0}]
#O[2]:  {{f[1,1],f[1,0]},{f[0,1],f[0,0]}}
#I[3]:: Outer[And,{1,0},{1,0}]
#O[3]:  {{1,0},{0,0}}
#I[4]:: Outer[Or,{1,0},{1,0}]
#O[4]:  {{1,1},{1,0}}
#I[5]:: Outer[Xor,{1,0},{1,0}]
#O[5]:  {{0,1},{1,0}}
#I[6]:: Outer[Imp,{1,0},{1,0}]
#O[6]:  {{1,0},{1,1}}
```

Is[*expr*]
yields 1 if *expr* represents a logical tautology "true" regardless of the "truth" or "falsity" of any symbols appearing in it, and yields 0 otherwise.
• Neq [9.6]

```
#I[1]:: {Is[p | ~p], Is[p || p], Is[p & ~p], Is[(p=>q) = ((~q) => (~p))]}
#O[1]:  {1,0,0,1}
```

```
#I[2]:: base: (even | odd) => (int & real)

#O[2]:  even | odd => int & real

#I[3]:: {Is[base=>odd=>int], Is[base=>int=>odd], Is[base=>(~real)=>(~even)]}

#O[3]:  {1,0,1}
```

- If [6.1]
- Intp,.. [7.6]

Ord[*expr1*,*expr2*]
yields +1 if expr1 is lexically [10.6] ordered before expr2, -1 if *expr1* is lexically ordered after *expr2* and 0 if they are literally equivalent [2.6].

- Reor [7.7]
- Sort [7.7]

```
#I[1]:: {Ord[a,b],Ord[b,a],Ord[f[x],x],Ord[a,a]}

#O[1]:  {1,-1,-1,0}
```

6. Control Structures

6.1 Conditional statements

If[*pred*, (*expr1*:Null), (*expr2*:Null), (*expr3*:Null)]
yields *expr1* if the predicate expression *pred* is determined to be "true" [5]; *expr2* if it is determined to be "false", and *expr3* if its truth or falsity cannot be determined [5]. Only the *expri* selected by evaluation of *pred* is simplified.

```
#I[1]:: f1[$x]::If[$x>0,a]
#0[1]:  ' If[$x > 0,a]
#I[2]:: f2[$x]::If[$x>0,a,b]
#0[2]:  ' If[$x > 0,a,b]
#I[3]:: f3[$x]::If[$x>0,a,b,c]
#0[3]:  ' If[$x > 0,a,b,c]
#I[4]:: {f1[1],f2[1],f3[1]}
#0[4]:  {a,a,a}
#I[5]:: {f1[-1],f2[-1],f3[-1]}
#0[5]:  {f1[-1],b,b}
#I[6]:: {f1[x],f2[x],f3[x]}
#0[6]:  {f1[x],f2[x],c}
```

Switch[*pred1*, *expr1*, *pred2*, *expr2*, . . .]
tests *pred1*, *pred2*, ... in turn, selecting the *expri* associated with the first one determined to be "true" [5] (Null if none are "true"). Only the *predi* tested and the *expri* selected are simplified.

```
#I[1]:: f[$x]::Switch[$x>5,a,$x<3,b]
#0[1]:  ' Switch[$x > 5,a,3 > $x,b]
#I[2]:: {f[7],f[4],f[2],f[x]}
#0[2]:  {a,f[4],b,f[x]}
#I[3]:: g[$x]::Switch[$x>5,a,$x<3,b,Pr["no value for",$x]]
#0[3]:  ' Switch[$x > 5,a,3 > $x,b,Pr["no value for",$x]]
#I[4]:: {g[7],g[4],g[2],g[x]}
"no value for"  4
"no value for"  x
#0[4]:  {a,g[4],b,g[x]}
```

6.2 Iteration

Rpt[*expr*, (*n*:1)]
simplifies *expr* *n* times, yielding the last value found.

```
#I[1]::  Rpt[Pr[x],3]

x
x
x

#O[1]:   x

#I[2]::  r:x

#O[2]:   x

#I[3]::  Rpt[r:1/(1+r),3]

                1
#O[3]:  -------------
                1
        1 + ---------
                1
            1 + -----
                1 + x
```

Loop[(*precond*:1), *expr*, (*postcond*:1)]
repeatedly simplifies *precond*, *expr* and *postcond* in turn, yielding the last value of *expr* found before *precond* or *postcond* ceases to be "true" [5].

```
#I[1]::  i:0;Loop[i<5,Pr[i,i!];Inc[i]]

0        1
1        1
2        2
3        6
4        24

#O[1]:   5

#I[2]::  i

#O[2]:   5

#I[3]::  i:0;Loop[,Pr[i];Inc[i],i<3]

0
1
2

#O[3]:   3

#I[4]::  i:0;Loop[i<0,Pr[i];Inc[i]]

#I[5]::  i:0;Loop[,Pr[i];Inc[i],i<0]

0

#O[5]:   1
```

For[*init*, *test*, *next*, *expr*]
first simplifies *init*, and then repeatedly simplifies *expr* and *next* in turn until *test* fails to be "true" (according to the sequence *init* <*test* *expr* *next*>), yielding the last form of *expr* found.

```
#I[1]::  For[i:0,i<5,Inc[i],Pr[i,!!]]

0       1
1       1
2       2
3       6
4       24

#O[1]:   24

#I[2]::  !

#O[2]:   5

#I[3]::  For[x:10000,x~=0,x:Gint[x/33],Pr[x]]

10000
303
9

#O[3]:   9
```

Do [*var*, (*start*:1), *end*, (*step*:1), *expr*]
first sets *var* to *start* and then repeatedly evaluates *expr*, successively incrementing the value of *var* by *step* until it reaches the value *end* ; the image is the last form of *expr* found.

```
#I[1]::  Do[i,4,Pr[i,!!]]

1       1
2       2
3       6
4       24

#O[1]:   24

#I[2]::  !

#O[2]:   !

#I[3]::  Do[i,2,8,3,Pr[i,!!]]

2       2
5       120
8       40320

#O[3]:   40320

#I[4]::  Do[i,x,x+2,f[1/i]:!]

#O[4]:   2 + x

#I[5]::  f

                1                   1                1
#O[5]:   {[-------]: 2 + x, [-------]: 1 + x, [-]: x}
           2 + x               1 + x            x
```

- Inc, Dec [3.2]
- Ret, Jmp [6.3]

6.3 Procedures and flow control

***expr1*; *expr2*; ...** or **Proc[*expr1*, *expr2*, ..., *exprn*]**
represents a procedure [1.8] in which the expressions *expri* are simplified in turn, yielding finally the value of *exprn*.

In the form `expr1;expr2;...;exprk;` the last filter of `Proc` is taken to be `Null` [1.1].

The value of `%%` [1.8] is reassigned at each segment in a `Proc` to be the last non-`Null` *expri* simplified.

The unsimplified form of each *expri* is maintained throughout the execution of a `Proc`, so as to allow resimplification if required by a control transfer.

`Proc[]` initiates an interactive procedure [1.8].

```
#I[1]::  Pr[1];Pr[2];Pr[7]

1
2
7

#O[1]:   7

#I[2]::  Pr[1];Pr[2];

1
2

#I[3]::  f[$x]::($x-1;%% - %%^2)

                                  2
#O[3]:   ' $x - 1 ; %% - %%

#I[4]::  {f[x],f[2]}

                                  2
#O[4]:   {-1 + x - (-1 + x) ,0}

#I[5]::  Proc[]

%I[1]::  y-1

%O[1]:   -1 + y

%I[2]::  %%^2-4

                                2
%O[2]:   -4 + (-1 + y)

%I[3]::  Ret[14]

%O[3]:   14

#O[5]:   14
```

Lcl[*s1*, *s2*, ...]
declares the symbols *s1*,*s2*,... to be "local variables" in the current `Proc` (and any nested within it); the original values and properties of the *si* are removed, to be restored upon exit from the `Proc`.

Local variables are conventionally given names beginning with the character `%` [2.2].

`Lcl` may be used in interactive procedures [1.8].

```
#I[1]::  Lbl[a];a:5;Prt[a];a
5
#O[1]:   5
#I[2]::  a
#O[2]:   a
#I[3]::  b:1
#O[3]:   1
#I[4]::  Proc[]
%I[1]::  c:2
%O[1]:   2
%I[2]::  {b,c}
%O[2]:   {1,2}
%I[3]::  Lbl[b,c]
%O[3]:   Lbl[b,c]
%I[4]::  {b,c}
%O[4]:   {b,c}
%I[5]::  b:c:5
%O[5]:   5
%I[6]::  {b,c}
%O[6]:   {5,5}
%I[7]::  Ret[b]
%O[7]:   5
#O[4]:   5
#I[5]::  {b,c}
#O[5]:   {1,2}
#I[6]::  Lbl[%x];%y:%x:3;(Lbl[%x];%x:7;Prt[%x,%y]);Prt[%x,%y];%x
7        3
3        3
#O[6]:   3
```

Lbl[*expr*]
represents a "label" within a Proc ; its "identifier" ***expr*** is resimplified when a Jmp might transfer control to its position.

Jmp[*expr*]
causes "control" to be "transferred" to the nearest label which matches Lbl[*expr*] The current Proc, and then any successive enclosing Proc are scanned to find a suitable Lbl. After Jmp has acted, the remaining segments of the Proc containing the Lbl are (re)simplified. For positive integer ***n***, Jmp[*n*] transfers control to the ***n***th segment in the current procedure. Jmp may be used in interactive procedures: if the specified Lbl is not present, input lines are read without simplification until it is encountered.

```
#I[1]::  Pr[1];Jmp[on];Pr[2];Lb![on];Pr[3]

1
3

#0[1]:   3

#I[2]::  Pr[1];(x;Jmp[on]);Pr[2];Lb![on];Pr[3]

1
3

#0[2]:   3

#I[3]::  Pr[1];Jmp[on];Pr[2]

1

#0[3]:   Jmp[on]

#I[4]::  i:0;Lb![a];Pr[i];Inc[i];If[i<3,Jmp[a]]

0
1
2

#I[5]::  Pr[a];Jmp[4];Pr[y];Pr[z];Pr[w]

a
w

#0[5]:   w

#I[6]::  i:x;Lb![i+x];Pr[i];Inc[i];Jmp[x+1];Jmp[x+2]

x

#0[6]:   Jmp[x + 1]
```

Ret[*expr*, (*n*:1)]

exits at most *n* nested control structures, yielding *expr* as the value of the outermost one. Ret is effective in Proc, Rpt, For and Do. It may be used to exit interactive subsidiary procedures [1.8]. Ret[*expr*, Inf] returns from any number of nested procedures to standard input mode [1.8].

```
#I[1]::  a;Ret[b];c

#0[1]:   b

#I[2]::  f[Ret[a],b]

#0[2]:   f[a,b]

#I[3]::  a;(b;Ret[x]);c

#0[3]:   c

#I[4]::  a;(b;Ret[x,2]);c

#0[4]:   x

#I[5]::  a;(b;Ret[x,Inf]);c

#0[5]:   x
```

```
#I[6]::  Do[i,10,If[i>3,Ret[i]]]

#O[6]:   4

#I[7]::  Do[i,10,If[i>3,Ret[i]];Pr[i]]

1
2
3

#O[7]:   10

#I[8]::  Do[i,10,If[i>3,Ret[i,2]];Pr[i]]

1
2
3

#O[8]:   4
```

- [10.7]

7. Structural operations

7.1 Projection and list generation

x .. y or **Seq[x, y]**
yields if possible a null projection [2.3] consisting of a sequence of expressions whose integer parameters form linear progressions between those in x and y. For two integers m and n, (with $n > m$) $m..n$ yields $[m,m+1,m+2,\ldots,n-1,n]$. Similarly, $f[m1,m2]..f[n1,n2]$ yields $[f[m1,m2], f[m1+1, m2+i], f[m1+2, m2+2i], \ldots, f[n1,n2]]$ if $i = (n2-m2)/(n1-m1)$ is an integer. Only the values of integer parameters may differ between x and y, and all such differences must be integer multiples of the smallest.

```
#I[1]:  1..5
#O[1]:  [1,2,3,4,5]
#I[2]:  {1..5}
#O[2]:  {1,2,3,4,5}
#I[3]:  f[1..5,18,2x..6x]
#O[3]:  f[1,2,3,4,5,18,2x,3x,4x,5x,6x]
#I[4]:  f[1]..f[5]
#O[4]:  [f[1],f[2],f[3],f[4],f[5]]
#I[5]:  f[8,8]..f[4,8]
#O[5]:  [f[8,8],f[1,2],f[2,4],f[3,6],f[4,8]]
#I[6]:  f[8,8]..f[4,9]
#O[6]:  f[8,8] .. f[4,9]
#I[7]:  x^2..x^4
                 2  3  4
#O[7]:  [x ,x ,x ]
#I[8]:  f[g[10(x+y),8],100]..f[g[60(x+y),5],600]
#O[8]:  [f[g[10(x + y),8],100],f[g[20(x + y),1],200],f[g[30(x + y),2],300],
         f[g[40(x + y),3],400],f[g[50(x + y),4],500],f[g[60(x + y),5],600]]
```

Ar [*spec*, (*temp*:Eq), (*icrit*:1), (*vcrit*:1)]
generates a list whose entries have sets of indices with ranges specified by *spec*, and whose values are obtained by application of the template *temp* to these indices. Sequences of indices at each level in the list are defined by

n	$1,2,\ldots n$
$\{s,e,(i:1)\}$	$s,\ s+i,\ s+2i,\ldots,\ s+k*i$ where k is the largest integer such that $s+k*i$ is not greater than e.
$\{\{x1,x2,..\}\}$	$x1,\ x2,\ldots$

and collected into a complete specification $\{spec1, spec2, \ldots\}$. For a contiguous [2.4] list with one level, *spec* may be given as n. Entries with sets of indices on which application of the template *icrit* would yield $\emptyset$ are omitted. Entries whose values would yield $\emptyset$ on application of the template *vcrit* are also omitted.

- Outer [9.6]
- Dim [7.4]
- [3.6]

```
#I[1]::  Ar[5,f]
#O[1]:   {f[1],f[2],f[3],f[4],f[5]}
#I[2]::  Ar[{{2,12,3}},f]
#O[2]:  {[2]: f[2], [5]: f[5], [8]: f[8], [11]: f[11]}
#I[3]::  Ar[{{x,x+4}},f]
#O[3]:   {[x]: f[x], [1 + x]: f[1 + x], [2 + x]: f[2 + x], [3 + x]: f[3 + x]}
#I[4]::  Ar[{{{i1,i2,i3}}},f]
#O[4]:   {[i1]: f[i1], [i2]: f[i2], [i3]: f[i3]}
#I[5]::  Ar[{2,2},f]
#O[5]:   {{f[1,1],f[1,2]},{f[2,1],f[2,2]}}
#I[6]::  Ar[{2,2,2},f]
#O[6]:   {{{f[1,1,1],f[1,1,2]},{f[1,2,1],f[1,2,2]}},
                {{f[2,1,1],f[2,1,2]},{f[2,2,1],f[2,2,2]}}}
#I[7]::  Ar[{2,{2,5}},f]
#O[7]:   {{[2]: f[1,2], [3]: f[1,3], [4]: f[1,4], [5]: f[1,5]},
                {[2]: f[2,2], [3]: f[2,3], [4]: f[2,4], [5]: f[2,5]}}
#I[8]::  Ar[5]
#O[8]:   {1,2,3,4,5}
#I[9]::  Ar[5,Fct]
#O[9]:   {1,2,6,24,120}
#I[10]:: Ar[{3,3}]
#O[10]:  {{1,0,0},{0,1,0},{0,0,1}}
#I[11]:: Ar[{4,2},($x+1)^($y-1)]
#O[11]:  {{1,2},{1,3},{1,4},{1,5}}
#I[12]:: Ar[2,@1]
#O[12]:  {f[1],f[2]}
#I[13]:: Ar[{{3,5}},@1]
#O[13]:  {[3]: f[3], [4]: f[4], [5]: f[5]}
#I[14]:: Ar[10,f,Evenp]
#O[14]:  {[2]: f[2], [4]: f[4], [6]: f[6], [8]: f[8], [10]: f[10]}
#I[15]:: Ar[{3,3},f,$x>$y-1]
#O[15]:  {{f[1,1]},{f[2,1],f[2,2]},{f[3,1],f[3,2],f[3,3]}}
#I[16]:: Ar[10,$x^2-3$x,Evenp]
```

```
#O[16]:   {[1]: -2, [2]: 4, [3]: 18, [4]: 40, [5]: 70}
#I[17]::  Ar[10,$x^2-3$x,1,Evenp]
#O[17]:   {[1]: -2, [2]: -2, [3]: 4, [4]: 10, [5]: 18, [6]: 28, [7]: 40,
           [8]: 54, [9]: 70}
#I[18]::  u:0;Ar[10,`Inc[u]]
#O[18]:   {1,2,3,4,5,6,7,8,9,10}
```

Repl[*expr*,(*n*:1)]
yields a null projection [2.3] containing *n* replications of *expr*.

```
#I[1]::  Repl[f[x],4]
#O[1]:   [f[x],f[x],f[x],f[x]]
#I[2]::  {Repl[0,5]}
#O[2]:   {0,0,0,0,0}
#I[3]::  Ar[4,Repl[$1,$1]]
#O[3]:   {[1],[2,2],[3,3,3],[4,4,4,4]}
```

List[*expr1*,*expr2*,...]
yields the contiguous list {*expr1*,*expr2*,...}

```
#I[1]::  List[x,y,z]
#O[1]:   {x,y,z}
#I[2]::  Ar[5,List]
#O[2]:   {{1},{2},{3},{4},{5}}
```

7.2 Template application

Ap [*temp*, {*expr1*, (*expr2*, ...)}]
applies the template *temp* to the *expri* [2.7].

```
#I[1]:  Ap[f,{a,b,c}]
#O[1]:  f[a,b,c]
#I[2]:  Ap[$z^$x-$x,{a,b}]
                    a
#O[2]:  -a + b
#I[3]:  Ap[{a,b,c,d},{2}]
#O[3]:  b
#I[4]:  f:g
#O[4]:  g
#I[5]:  Ap[f,{a,b}]
#O[5]:  f[a,b]
#I[6]:  Ap[!f,{a,b}]
#O[6]:  g[a,b]
#I[7]:  Ap[Ap,{f,{a,b}}]
#O[7]:  g[a,b]
```

Map [*temp*, *expr*, (*levspec*:1), (*domcrit*:1), (*ltemp*:Null)]
yields the expression obtained by recursively applying the template *temp* at most *max* times to the parts of *expr* in the domain specified by *levspec* and *domcrit* [2.5]. ("Multiple Apply") Any non-Null result obtained by applying the template *ltemp* to the (positive or negative) integer specifying the level in *expr* reached is used as a second expression on which to apply *temp*. (With *lev*: 0 Map is equivalent to Ap with a single *expri*.)

```
#I[1]:  t:{{a,b},{x^2,y^2},z}
                    2   2
#O[1]:  {{a,b},{x ,y },z}
#I[2]:  Map[f,t]
                      2   2
#O[2]:  {f[{a,b}],f[{x ,y }],f[z]}
#I[3]:  Map[f,t,2]
                          2     2
#O[3]:  {f[{f[a],f[b]}],f[{f[x ],f[y ]}],f[z]}
#I[4]:  Map[f,t,2,,$x]
                          2       2
#O[4]:  {f[{f[a,2],f[b,2]},1],f[{f[x ,2],f[y ,2]},1],f[z,1]}
```

```
#I[5]::  Map[f,t,Inf,,$x]

#O[5]:   {f[{f[a,2],f[b,2]},1],f[{f[f[x,3]^f[2,3],2],f[f[y,3]^f[2,3],2]},1],
           f[z,1]}

#I[6]::  Map[f,t,-1]

#O[6]:   {{f[a],f[b]},{f[x]^f[2],f[y]^f[2]},f[z]}

#I[7]::  Map[f,t,-Inf,,$x]

#O[7]:   {f[{f[a,-1],f[b,-1]},-2],
           f[{f[f[x,-1]^f[2,-1],-2],f[f[y,-1]^f[2,-1],-2]},-3],f[z,-1]}

#I[8]::  u:Ar[10]

#O[8]:   {1,2,3,4,5,6,7,8,9,10}

#I[9]::  Map[f,u,1,Evenp]

#O[9]:   {1,f[2],3,f[4],5,f[6],7,f[8],9,f[10]}

#I[10]::  v:a^(b+c a^2)+2g[a-b,c^2+d^2]

                 2
          b + a  c                2    2
#O[10]:  a           + 2g[a - b,c  + d ]

#I[11]::  Map[f,v,-1]

                   f[2]
          f[a]    f[c] + f[b]                          f[2]      f[2]
#O[11]:  f[a]                  + 2g[f[a] + f[-b],f[c]      + f[d]    ]

#I[12]::  Map[f,v,-1,~Numbp[$x]]

                 2
          f[a]  f[c] + f[b]                  2      2
#O[12]:  f[a]               + 2g[f[a] + f[-b],f[c]  + f[d] ]

#I[13]::  Map[f,v,-Inf,~In[a,$x]]

            f[2]
           a     f[c] + f[b]                          f[2]          f[2]
#O[13]:  a                   + 2g[a + f[-b],f[f[f[c]     ] + f[f[d]     ]]]

#I[14]::  Map[f,v,Inf,$y & ~In[a,$x],$1>2]

            f[2,1]
           a       f[c,1] + f[b,1]
#O[14]:  a
                                                 f[2,1]          f[2,1]
                  + 2g[a + f[-b,1],f[f[f[c,1]       ,1] + f[f[d,1]       ,1],0]]
```

7.3 Part extraction and removal

***expr*[*filt1*, *filt2*, ...] or Proj[*expr*, {*filt1*, *filt2*, ...}]**
extracts the part of *expr* specified by successive projections with the filters *filt1*, *filt2*, ... [2.3].
• [2.10]

Pos[{*form1*}, *expr*, (*levspec*:Inf), (*max*:Inf), (*domcrit*:1)]
gives a list of sets of filters specifying the positions of (at most *max*) occurrences of parts in *expr* (in the domain specified by *levspec* and *domcrit* [2.5]) matching any of the *formi*. If *formi* are patterns [2.6], the actual subexpressions matched are also given.
• In [7.5]

```
#I[1]:: t:a*h[a^2+4b^2+2c] + h[c^2*(h[a*c]-b^2)]

                    2      2        2      2
#O[1]:  a h[2c + a  + 4 b ] + h[c  (- b  + h[a c])]

#I[2]:: Pos[b^2,t]

#O[2]:  {{1,2,1,3},{2,1,2,1}}

#I[3]:: Pos[$x^2,t]

                  2                  2                  2                  2
#O[3]:  {{a ,{1,2,1,2}},{b ,{1,2,1,3}},{c ,{2,1,1}},{b ,{2,1,2,1}}}

#I[4]:: Pos['h,t]

#O[4]:  {{1,2,0},{2,0},{2,1,2,2,0}}

#I[5]:: Pos[{a,b},t]

#O[5]:  {{1,1},{1,2,1,2,1},{2,1,2,2,1,1},{1,2,1,3,1},{2,1,2,1,1}}

#I[6]:: Pos[a,t,3]

#O[6]:  {{1,1}}

#I[7]:: Pos[h[$x],t,-3]

                    2      2
#O[7]:  {{h[2c + a  + 4 b ],{1,2}},{h[a c],{2,1,2,2}}}

#I[8]:: Pos[a,t,Inf,2]

#O[8]:  {{1,1},{1,2,1,2,1}}

#I[9]:: u:a^b+c

                  b
#O[9]:  c + a

#I[10]:: Pos[$x,u]

                                                  b                  b
#O[10]:  {{c,{1}},{a,{2,1}},{b,{2,2}},{a ,{2}},{c + a ,{0}}}
```

Elem[*expr*, {*n1*, *n2*, ...}]
extracts successively the *ni*th values in the list *expr*, irrespective of their indices.

```
#I[1]::  t:{[x]:a,[y]:b,[1]:c}
#O[1]:   {[x]: a, [y]: b, [1]: c}
#I[2]::  Elem[t,{1}]
#O[2]:   a
#I[3]::  t[1]
#O[3]:   c
```

Last[*list*]
yields the value of the last entry in *list*.

```
#I[1]::  t:{[x]:a,[y]:b}
#O[1]:   {[x]: a, [y]: b}
#I[2]::  Last[t]
#O[2]:   b
#I[3]::  Last[a]
#O[3]:   Last[a]
```

Ind[*list*, *n*]
yields the index of the *n*th entry in *list*.

```
#I[1]::  Ind[{[x]:a,[y]:b},1]
#O[1]:   x
#I[2]::  Ind[{x,y,z},2]
#O[2]:   2
```

Dis[*expr*, (*lev*:1)]
yields a list in which all projections in *expr* (below level *lev)* are "disassembled" into lists with the same parts.

```
#I[1]::  t:f[g[a,b],c^2,d]
                      2
#O[1]:   f[g[a,b],c ,d]
#I[2]::  Dis[t]
#O[2]:   {[0]: ' f, [1]: {[0]: ' g, [1]: a, [2]: b},
                [2]: {[0]: ' Pow, [1]: c, [2]: 2}, [3]: d}
#I[3]::  Dis[t,1]
                                  2
#O[3]:   {[0]: ' f, [1]: g[a,b], [2]: c , [3]: d}
#I[4]::  Dis[5a/6]
#O[4]:   {[-1]: 5/6, [0]: a}
```

As [*expr*, (*lev*:1)]
yields an expression in which any suitable lists (at or below level *lev*) in *expr* are "assembled" into projections with the same parts.

```
#I[1]:  Dis[f[x,y,z^2]]

                                    2
#O[1]:  {[0]: ' f, [1]: x, [2]: y, [3]: z }

#I[2]:  As[%]

                      2
#O[2]:  f[x,y,z ]
```

De l [*form*, *expr*, (*lev*:Inf), (*n*:Inf)]
yields an expression in which (at most *n*) parts matching *form* (and appearing at or below level *lev*) in *expr* have been deleted.
• Set [3.2]

```
#I[1]:  De![a,{a,b,a,c}]

#O[1]:  {b,c}

#I[2]:  t:(a+b^2+c*a)^(a+b*c)

                          2 a + b c
#O[2]:  (a + a c + b )

#I[3]:  De![a,t]

                 2 b c
#O[3]:  (c + b )
```

7.4 Structure determination

Tree[*expr*, (*nlev*:1)]
yields a list of successive replacements specifying the construction of *expr* from its level -*nlev* parts.
• Dis [7.3]

```
#I[1]:  t:a^(b+c)+d

                  b + c
#O[1]:  d + a

#I[2]:  Tree[%]

                                  #3
#O[2]:  {#1 -> d + #2, #2 -> a    , #3 -> b + c}
```

Len[*expr*]
the number of filters or entries in a projection or list *expr*, and 0 for a symbol *expr*. (The "length" of *expr*).

```
#I[1]:  t:a^2+b^2+c+d

                       2    2
#O[1]:  c + d + a  + b

#I[2]:  Len[t]

#O[2]:  4

#I[3]:  Len[{a,b}]

#O[3]:  2
```

Dep[*expr*]
the maximum number of filters necessary to specify any part of *expr*. (The "depth" of *expr* [2.5]).

```
#I[1]:  Dep[a]

#O[1]:  1

#I[2]:  Dep[a^2]

#O[2]:  2

#I[3]:  Dep[a^2+b^2]

#O[3]:  3

#I[4]:  Dep[a^2+b^2+c^2]

#O[4]:  3

#I[5]:  Dep[f[f[f[f[a]]]+f[a]]

#O[5]:  5
```

Dim[*list*, (*llev*:Inf)]
gives a description of the ranges of indices at or below level *llev* in *list* (in the form used by Ar [7.1]).

```
#I[1]::  {{a},{b}}
#O[1]:   {{a},{b}}
#I[2]::  Dim[%]
#O[2]:   {2,1}
#I[3]::  Ar[{2,3,1}]
#O[3]:   {{{1},{0},{0}},{{0},{0},{0}}}
#I[4]::  Dim[%]
#O[4]:   {2,3,1}
#I[5]::  Dim[%]
#O[5]:   {3}
```

Hash[*expr*, (*n*:2^15)]
: a positive integer less than *n* which provides an almost unique "hash code" for *expr*. Two expressions with different hash codes cannot be identical.

```
#I[1]::  Hash[a]
#O[1]:   935360
#I[2]::  Hash[a^2+b^2/c]
#O[2]:   5823468
```

7.5 Content determination

In[{*form1*}, *expr*, (*lev*:Inf)]
yields 1 if a part matching any of the *formi* occurs in *expr* at or below level *lev*, and 0 otherwise.

```
#I[1]:: t:x+a*x*y^2+x^2

                    2    2
#O[1]:  x + a x y  + x

#I[2]:: In[y^2,t]

#O[2]:  1

#I[3]:: In[{z,x^2},t]

#O[3]:  1

#I[4]:: In[x^$i_;Evenp[$i],t]

#O[4]:  1

#I[5]:: In[z,t]

#O[5]:  0

#I[6]:: In[y^2,t,1]

#O[6]:  0
```

Cont[<*expr*: (all symbols)>, (*crit*:1)]
yields an ordered list of the symbols in *expr* on which application of the template *crit* does not yield 0 (default is to omit symbols appearing as projectors).

```
#I[1]:: t:a+b^c

              c
#O[1]:  a + b

#I[2]:: Cont[t]

#O[2]:  {a,b,c}

#I[3]:: Cont[t,1]

#O[3]:  {Plus,Pow,a,b,c}

#I[4]:: a_:b_:d_:spec

#O[4]:  spec

#I[5]:: _a

#O[5]:  {[spec]: 1}

#I[6]:: Cont[t,_$x[spec]]

#O[6]:  {a,b}

#I[7]:: Cont[,_$x[spec]]

#O[7]:  {a,b,d}
```

7.6 Character determination

The following projections yield 1 if *expr* is determined to be of the specified character, and 0 otherwise. The determination includes use of any assignments made for the testing projection; `Intp[x]:1` thus defines x to be an integer.

Symbp[*expr*] Single symbol

Numbp[*expr*] Real or complex number.

Realp[*expr*] Real number.

Imagp[*expr*] Purely imaginary number.

Intp[*expr*] Integer.

Natp[*expr*] Natural number (positive integer).

Evenp[*expr*] Even integer.

Oddp[*expr*] Odd integer.

Ratp[*expr*, (*maxden*:1*^8), (*acc*:1*^-13)]
Rational number (to within accuracy *acc*) with denominator not greater than *maxden*.

Projp[*expr*] Projection.

Listp[*expr*] List.

Contp[*expr*, (*lev*:Inf)]
List or list of lists contiguous [2.4] to at least level *lev*.

Fullp[*expr*, (*lev*:Inf)]
"Full" list whose indices (and those of its sublists at or below level *lev*) are "contiguous" and have ranges independent of the values of any other indices (so that the list is "rectangular").

Valp[*expr*] Value other than Null.

Heldp[*expr*] "Held" expression [3.5].

Polyp[*expr*, (*form1*)]
Polynomial in "bases" matching *formi* [9.1].

```
#I[1]:: {Intp[3], Intp[3/2], Intp[x]}
#O[1]:  {1,0,0}
#I[2]:: Intp[x]:1
#O[2]:  1
#I[3]:: {Intp[x], Intp[x^2], Intp[y]}
#O[3]:  {1,0,0}
#I[4]:: Intp[$x_:_$x[int]]:1
#O[4]:  1
#I[5]:: Intp[$x^2]::Intp[$x]
#O[5]:  ' Intp[$x]
#I[6]:: y_:int
#O[6]:  int
```

```
#I[7]:: {Intp[x^2],Intp[y],Intp[y^2],Intp[z^2]}
#O[7]:  {1,1,1,0}
#I[8]:: {Ratp[0.773],Ratp[N[Pi]],Ratp[2/3,4],Ratp[2/3,2]}
#O[8]:  {1,0,1,0}
#I[9]:: {Contp[{a,b,c}],Contp[{[x]:a,[y]:b}]}
#O[9]:  {1,0}
#I[10]:: t:{{a,b},{c}}
#O[10]:  {{a,b},{c}}
#I[11]:: u:{{a,b},{[x]:c}}
#O[11]:  {{a,b},{[x]: c}}
#I[12]:: {Contp[u],Contp[u,1],Contp[u,2],Contp[t]}
#O[12]:  {0,1,0,1}
#I[13]:: v:{{a,b},{c,d}}
#O[13]:  {{a,b},{c,d}}
#I[14]::  {Fullp[t],Fullp[t,1],Fullp[t,2],Fullp[v]}
#O[14]:  {1,1,1,1}
#I[15]:: f[x]:xv
#O[15]:  xv
#I[16]:: {Valp[f],Valp[f[x]],Valp[f[y]],Valp[1]}
#O[16]:  {1,1,0,0}
#I[17]:: {Heldp[`f],Heldp[#I[15][0]],Heldp[f]}
#O[17]:  {0,1,0}
#I[18]:: {Polyp[x^2-x+1],Polyp[x+Exp[x]]}
#O[18]:  {1,0}
```

7.7 List and projection manipulation

Cat[*list1*, *list2*, . . .]
<Flat>
yields a contiguous list [2.4] obtained by concatenating the entries in *list1*, *list2*,... Cat[*list*] renders the indices of entries in *list* contiguous, without affecting their values.

```
#I[1]:  t1:{a,b,c}
#O[1]:  {a,b,c}
#I[2]:  t2:{d,{e,f}}
#O[2]:  {d,{e,f}}
#I[3]:  Cat[t1,t2,t1,t1]
#O[3]:  {a,b,c,d,{e,f},a,b,c,a,b,c}
#I[4]:  Cat[{[x]:1,[y]:2},t1]
#O[4]:  {1,2,a,b,c}
#I[5]:  Cat[{[a]:x,[b]:y,[c]:z}]
#O[5]:  {x,y,z}
```

Sort[*list*, (*card*:Null), (*ord*:Ord)]
yields a list v obtained by sorting the top-level entries in *list* so that either the expressions Ap[*card*, {Ent[v, {i}]] are in canonical order (for increasing i), or Ap[*ord*, {Ent[v, {i}], Ent[v, {j}]}] is non-negative for $i > j$ and is non-positive for $i < j$.

```
#I[1]:  t:{2,1,3,4,1}
#O[1]:  {2,1,3,4,1}
#I[2]:  Sort[t]
#O[2]:  {1,1,2,3,4}
#I[3]:  Sort[t,Evenp]
#O[3]:  {1,1,3,2,4}
#I[4]:  Sort[t,,Ord[$x,$y]]
#O[4]:  {4,3,2,1,1}
#I[5]:  Sort[{b,c,a}]
#O[5]:  {a,b,c}
#I[6]:  Sort[{[b]:3,[c]:1, [a]:2}]
#O[6]:  {[c]: 1, [a]: 2, [b]: 3}
```

Cyc[*expr*, (*n*:1)]
gives a list or projection obtained by cycling entries or filters in *expr* to the left by n positions with respect to their first index.

Rev[*expr*]
yields a list or projection obtained from *expr* by reversing the order of entries or filters with respect to their first index.

```
#I[1]:: t:{a,b,c,d}
#O[1]:  {a,b,c,d}
#I[2]:: Cyc[t]
#O[2]:  {b,c,d,a}
#I[3]:: Cyc[t,-2]
#O[3]:  {c,d,a,b}
#I[4]:: Rev[t]
#O[4]:  {d,c,b,a}
#I[5]:: r:f[x,y,z,w]
#O[5]:  f[x,y,z,w]
#I[6]:: Cyc[%,15]
#O[6]:  f[w,x,y,z]
#I[7]:: Rev[%]+Cyc[%,-3]
#O[7]:  f[x,y,z,w] + f[z,y,x,w]
#I[8]:: Rev[{[a]:1,[b]:2}]
#O[8]:  {[b]: 2, [a]: 1}
```

Reor[*expr*, (*f*:*expr*[0]), (*reord*:Sym)]
places filters of projections from *f* in *expr* in canonical order using the reordering (permutation) symmetries specified by *reord*. Symmetries are represented by the following codes (or lists of such codes applied in the order given):

Sym	Completely symmetric under interchange of all filters.
Asym	Completely antisymmetric. • Sig [9.6]
Cyclic	Completely cyclic.
Sym[*i1*, *i2*, . .]	Symmetric with respect to interchanges of filters *i1*, *i2*, ...
Asym[*i1*, *i2*, . .]	Antisymmetric with respect to interchanges of filters *i1*, *i2*, ...
Cyclic[*i1*, *i2*, . .]	Symmetric under cyclic interchange of filters *i1*, *i2*, ...
Greor[{*perm1*, (*wt1*:1)}, {*perm2*, (*wt2* :1) }, . .]	Projections are multiplied by the weights obtained by application of *wti* to them when their filters are permuted so that the *j*th becomes the *permi*[*j*]th.

Projections from a symbol *f* are automatically reordered according to any permutation symmetries given (using the above codes) as the value of the property *_f*[Reor] [4].
• Comm [4]

```
#I[1]:: _ts[Reor]:Sym
#O[1]: Sym
#I[2]:: {ts[a,b,c],ts[b,a,c],ts[c,a,b],ts[c,a,a]}
#O[2]: {ts[a,b,c],ts[a,b,c],ts[a,b,c],ts[a,a,c]}
#I[3]:: _ta[Reor]:Asym
#O[3]: Asym
#I[4]:: {ta[a,b,c],ta[b,a,c],ta[c,a,b],ta[c,a,a]}
#O[4]: {ta[a,b,c],-ta[a,b,c],ta[a,b,c],0}
#I[5]:: _tc[Reor]:Cyclic
#O[5]: Cyclic
#I[6]:: {tc[a,b,c],tc[b,a,c],tc[c,a,b],tc[c,a,a]}
#O[6]: {tc[a,b,c],tc[a,c,b],tc[a,b,c],tc[a,a,c]}
#I[7]:: _t[Reor]:{Cyclic[1,3,4],Asym[2,5]}
#O[7]: {Cyclic[1,3,4],Asym[2,5]}
#I[8]:: {t[a,c,b,a,e],t[a,b,d,c,e],t[a,a,b,c,a],t[c,b,d,a,a],t[a,c,b,a,e,f,g]}
#O[8]: {t[a,c,a,b,e],t[a,b,d,c,e],0,-t[a,a,c,d,b],t[a,c,a,b,e,f,g]}
#I[9]:: _tg[Reor]:Greor[{{3,1,2},w1},{{1,3,2},w2}]
#O[9]: Greor[{{3,1,2},w1},{{1,3,2},w2}]
#I[10]:: {tg[a,b,c],tg[b,a,c],tg[c,a,b],tg[c,a,a],tg[c,a,b,d]}
```

Ldist[*expr*,(*f*:*expr*[0]),(*levspec*:Inf),(*domcrit*:1)]
"distributes" projections from *f* in the domain of *expr* specified by *levspec* and *domcrit* over lists appearing as their filters.
f[{[i1]:v11,[i2]:v12,..},{[i1]:v21,[i2]:v22,..},a,..] becomes {[i1]:f[v11,v21,a,..],[i2]:f[v12,v22,a,..],..} where a is not a list.

Projections from symbols carrying the property Ldist [4] are automatically distributed over lists appearing as their filters.

```
#I[1]:: t:f[{a,b},e,{c,d}]
#O[1]: f[{a,b},e,{c,d}]
#I[2]:: Ldist[t]
#O[2]: {f[a,e,c],f[b,e,d]}
#I[3]:: u:f[{a,b},{c,d},f[{i,j},{k,l}]]
#O[3]: f[{a,b},{c,d},f[{i,j},{k,l}]]
#I[4]:: Ldist[u]
#O[4]: {f[a,c,f[i,k]],f[b,d,f[j,l]]}
#I[5]:: Ldist[u,f,1]
#O[5]: {f[a,c,f[{i,j},{k,l}]],f[b,d,f[{i,j},{k,l}]]}
```

```
#I[6]:: {x,y,z}={1,2,3}
#O[6]:  {x,y,z} = {1,2,3}
#I[7]:: Ldist[%]
#O[7]:  {x = 1,y = 2,z = 3}
```

Flat[*expr*, (*levspec*:Inf), (*f*:*expr*[0] or), (*domcrit*:1)]
"flattens" nested projections from *f* or lists in *expr* in the domain specified by *levspec* and *domcrit* [2.5]. *f* projections appearing as filters within *f* projections are replaced by null projections [2.3]. Sublists in lists are "unraveled". Indices in flattened lists are contiguous [2.4].

```
#I[1]:: t:f[f[a,b],f[c],f[f[d]],e]
#O[1]:  f[f[a,b],f[c],f[f[d]],e]
#I[2]:: Flat[t]
#O[2]:  f[a,b,c,d,e]
#I[3]:: Flat[t,2]
#O[3]:  f[a,b,c,d,e]
#I[4]:: u:{{a,b},{c,{d}},e}
#O[4]:  {{a,b},{c,{d}},e}
#I[5]:: Flat[u]
#O[5]:  {a,b,c,d,e}
#I[6]:: v:{[x]:a,[y]:b,[z]:c}
#O[6]:  {[x]: a, [y]: b, [z]: c}
#I[7]:: Flat[v]
#O[7]:  {a,b,c}
```

Projections from symbols carrying the property Flat are automatically "flattened" [4].

Union[*list1*, *list2*, . . .]
<Comm,Flat>
yields a sorted contiguous [2.4] list of all entries appearing in *list1*, *list2*, . . .

```
#I[1]:: t:{a,b,a,a,d,c}
#O[1]:  {a,b,a,a,d,c}
#I[2]:: Union[t]
#O[2]:  {a,b,c,d}
#I[3]:: u:{a,b,f,e}
#O[3]:  {a,b,f,e}
#I[4]:: Union[u,t,u]
#O[4]:  {a,b,c,d,e,f}
```

Inter [*list1*, *list2*, . . .]
<Comm, Flat>
yields a sorted contiguous [2.4] list of the entries common to *list1*, *list2*, . . .

```
#I[1]:: t:{a,b,a,a,c,d}
#O[1]:  {a,b,a,a,c,d}
#I[2]:: u:{b,a,e,f}
#O[2]:  {b,a,e,f}
#I[3]:: Inter[t,u]
#O[3]:  {a,b}
```

7.8 Distribution and expansion

Ex[*expr*, (*dlist*), (*ndlist*: {}), (*rpt*: Inf), (*levspec*: Inf), (*domcrit*: 1)]

"expands" *expr* in the domain specified by *levcrit* and *domcrit* [2.5] using distributive replacements for projections specified in *dlist* but not in ndlist, performing replacements on a particular level at most *rpt* times. The default replacements in *dlist* are standard mathematical results for Mult, Div, Dot, Pow, Exp, Log, G and their extensions [4], together with any replacements defined by Dist or Powdist properties [4].

```
#I[1]:: t:(a+x)^2 (b+x(x+a)+(b+x)^2)

                          2                    2
#O[1]:  (a + x)  (b + x (a + x) + (b + x) )

#I[2]:: Ex[t]

                3       2       3     2      2  2      2  2    3      2  2
#O[2]:  5a x  + b x  + 2b x  + a  b + a  b  + 4 a  x  + a  x + b  x  + 2a b x

                      2       2       2          4
             + 4a b x  + 2a b  x + 2 a  b x + 2 x

#I[3]:: Ex[t,Pow]

                   2  2                                2  2
#O[3]:  (2a x + a  + x ) (b + 2b x + x (a + x) + b  + x )

#I[4]:: Ex[t,'Mult]

                   2          3          2          2
#O[4]:  b (a + x)  + x (a + x)  + (a + x)  (b + x)

#I[5]:: Ex[t,,Pow]

                   2          3          2          2
#O[5]:  b (a + x)  + x (a + x)  + (a + x)  (b + x)

#I[6]:: Ex[t,,,1]

                   2          3          2          2
#O[6]:  b (a + x)  + x (a + x)  + (a + x)  (b + x)

#I[7]:: Ex[t,,,2]

                     2  2          2      2      3    3
#O[7]:  b (2a x + a  + x ) + x (3a x  + 3 a  x + a  + x )

                          2  2          2  2
                + (2a x + a  + x ) (2b x + b  + x )

#I[8]:: Ex[t,,,,1]

                   2          3          2          2
#O[8]:  b (a + x)  + x (a + x)  + (a + x)  (b + x)

#I[9]:: Ex[t,,,,2]

                3       2       3     2      2  2      2  2    3      2  2
#O[9]:  5a x  + b x  + 2b x  + a  b + a  b  + 4 a  x  + a  x + b  x  + 2a b x

                      2       2       2          4
             + 4a b x  + 2a b  x + 2 a  b x + 2 x
```

```
#I[10]:: Ex[t,,,,-1]

                    2                    2
#O[10]:  (a + x)  (b + x (a + x) + (b + x)  )

#I[11]:: Ex[t,,,,-3]

                  2    2                    2      2
#O[11]:  (2a x + a  + x ) (b + a x + 2b x + b  + 2 x )

#I[12]:: Ex[t,,,,,~In[b,$x]]

                  2    2                 2      2
#O[12]:  (2a x + a  + x ) (b + a x + x  + (b + x) )

#I[13]:: Ex[t,,,,,Len[$x]=2]

          3      2      3    2      2 2      2 2      3      2 2
#O[13]:  3a x  + b x  + 2b x  + a  b + a  b  + 4 a  x  + a  x + b  x  + 2a b x

                                 2    2      2      4
              + 2a x (2b x + b  + x ) + 2 a  b x + 2 x

#I[14]:: Ex[2(u+v)]

#O[14]:  2u + 2v

#I[15]:: Ex[a(u+v)]

#O[15]:  a u + a v

#I[16]:: Ex[(u+v)^3]

            2      2      3    3
#O[16]:  3u v  + 3 u  v + u  + v

#I[17]:: Ex[(u+v)/a]

#O[17]:  u   v
         - + -
         a   a

#I[18]:: Ex[a.(u+v)]

#O[18]:  a.u + a.v

#I[19]:: Ex[Exp[u v]]

#O[19]:  Exp[u v]

#I[20]:: Ex[Log[u v]]

#O[20]:  Log[u] + Log[v]

#I[21]:: Ex[Log[u/v]]

#O[21]:  Log[u] - Log[v]

#I[22]:: Ex[Log[u~v]]

#O[22]:  v Log[u]

#I[23]:: Ex[f[a+b+c]]

#O[23]:  f[a + b + c]

#I[24]:: _f[Dist]::Plus

#O[24]:  ' Plus
```

```
#I[25]:: Ex[f[a+b+c]]

#O[25]:  f[a] + f[b] + f[c]

#I[26]:: _g[Powdist]::h

#O[26]:  ' h

#I[27]:: Ex[g[a+b,3]]

#O[27]:  h[h[a,a],a] + h[h[a,a],b] + h[h[a,b],a] + h[h[a,b],b] + h[h[b,a],a]
                + h[h[b,a],b] + h[h[b,b],a] + h[h[b,b],b]
```

Dist[*expr*,{{*f1*, *g1*, ((*h1*:*g1*), (*k1*:*f1*)), (*pspec1*:Inf)}}, (*rpt*:Inf), (*levspec*:Inf), (*domcrit*:1)]

distributes occurrences of the projectors *fi* in *expr* (in the domain specified by *levspec* and *domcrit* [2.5]) over projections from *gi* appearing as their filters, yielding projections of *hi* and *ki*. Distributions on a particular level are performed at most *rpt* times. A distribution specified by {*f*, *g*, *h*, *k*,*pspec*} corresponds to the replacement
f[$$1, *g*[*x1*,*x2*,...], $$2] -> *h*[*k*[$$1, *x1*, $$2], *k*[$$1, *x2*, $$2], ...]. The possible positions at which the *g* projection may appear in *f* are specified by *pspec* according to

i	1 through *i*.
{*i*}	*i* only.
{*i*,*j*}	*i* through *j*.
{*i*,*j*,*pcrit*}	*i* through *j* and such that application of the template *pcrit* yields "true".

```
#I[1]:: t:f[g[a,b],c,g[x,y,z]]

#O[1]:  f[g[a,b],c,g[x,y,z]]

#I[2]:: Dist[t,{f,g}]

#O[2]:  g[g[f[a,c,x],f[a,c,y],f[a,c,z]],g[f[b,c,x],f[b,c,y],f[b,c,z]]]

#I[3]:: Dist[t,{f,g,h,k}]

#O[3]:  h[k[a,c,g[x,y,z]],k[b,c,g[x,y,z]]]

#I[4]:: Dist[t,{f,g,h,k,2}]

#O[4]:  h[k[a,c,g[x,y,z]],k[b,c,g[x,y,z]]]

#I[5]:: Dist[t,{f,g,h,k,{3}}]

#O[5]:  h[k[g[a,b],c,x],k[g[a,b],c,y],k[g[a,b],c,z]]

#I[6]:: Dist[t,{{f,g,h,k},{k,g}}]

#O[6]:  h[g[k[a,c,x],k[a,c,y],k[a,c,z]],g[k[b,c,x],k[b,c,y],k[b,c,z]]]

#I[7]:: s:a+b(c+d)

#O[7]:  a + b (c + d)

#I[8]:: Dist[s,{Mult,Plus}]

#O[8]:  a + b c + b d
```

```
#I[9]:: Dist[s,{Plus,Mult}]

#O[9]:  (a + b) (a + c + d)
```

Powdist[*expr*,{{*fpow1*,*fmult1*,(*fplus1*:Plus)}},(*rpt*:Inf),(*levspec*:Inf),(*domcrit*:1)]

performs a "power expansion" on projections from *fpowi* appearing in *expr* (in the domain specified by *levspec* and *domcrit*) over projections from *fplus* appearing as their first filters, and yielding projections from *qmult*. Expansions on a particular level are performed at most *rpt* times. A power expansion specified by {*fpow*,*fmult*,*fplus*} corresponds to the replacement *fpow*[*fplus*[$$1],$n_?Natp[$n]] -> *fmult*[Rep1[*fplus*[$$1],$n] followed by distribution of *fmult* over *fplus*.

```
#I[1]:: t:(a+b)^3

                  3
#O[1]:  (a + b)

#I[2]:: Powdist[t,{Pow,Mult}]

                2       2       3     3
#O[2]:  3a b  + 3 a  b + a  + b

#I[3]:: Powdist[t,{Pow,Dot}]

#O[3]:  a.a.a + a.a.b + a.b.a + a.b.b + b.a.a + b.a.b + b.b.a + b.b.b

#I[4]:: Powdist[f[g[a,b],3],{f,h,g}]

#O[4]:  g[h[h[a,a],a],h[h[a,a],b],h[h[a,b],a],h[h[a,b],b],h[h[b,a],a],
         h[h[b,a],b],h[h[b,b],a],h[h[b,b],b]]
```

• Ldist [4,7.7]

7.9 Rational expression manipulation and simplification

Nc[*expr*]
gives the overall numerical coefficient of ***expr***.
- [2.5]

```
#I[1]::  {Nc[a/b], Nc[2a/b], Nc[2/(3b)], Nc[2/3]}

#O[1]:   {1,2,2/3,2/3}
```

Coef[*term*, *expr*, (*temp*:Plus)]
yields the result of applying ***temp*** to the set of coefficients of ***term*** in ***expr***.

```
#I[1]::  t:Ex[(2a+x+b*x)^3]

              2        3        2         2  3      3  3           2        2  2
#O[1]:   6a x  + 3b x  + 12 a  x + 3 b  x  + b  x  + 12a b x  + 6a b  x

                    2        3    3
              + 12 a  b x + 8 a  + x

#I[2]::  Coef[x^2,t]

                          2
#O[2]:   6a + 12a b + 6a b

#I[3]::  Coef[a*x^2,t]

                        2
#O[3]:   6 + 12b + 6 b

#I[4]::  Coef[a*b*x,t]

#O[4]:   0

#I[5]::  Coef[x,t,f]

              2      2
#O[5]:   f[12 a ,12 a  b]
```

Expt[*form*, *expr*, (*temp*:Max)]
yields the result of applying ***temp*** to the set of exponents for ***form*** in ***expr***.

```
#I[1]::  t:Ex[(x^3+f[y]^b+z/x^2)^2]

                   b    2
         2z f[y]       z                3       b     6        2b
#O[1]:   -------- + -- + 2x z + 2 x  f[y]  + x  + f[y]
             2       4
            x       x

#I[2]::  Expt[x,t]

#O[2]:   6

#I[3]::  Expt[f[y],t]

#O[3]:   Max[b,2b]

#I[4]::  Expt[x,t,h]

#O[4]:   h[-2,-4,1,3,6]
```

Num[*expr*]
numerator of *expr*.

```
#I[1]:: {Num[2a/(3b*c)], Num[5/6]}

#O[1]: {2a/3,5}

#I[2]:: {Num[0.75], Num[N[Pi]]}

#O[2]: {3,3.141593}
```

Den[*expr*]
denominator of *expr*.

```
#I[1]:: {Den[2a/(3*b*c)], Den[5/6]}

#O[1]: {b c,6}

#I[2]:: {Den[0.75], Den[N[Pi]]}

#O[2]: {4,1}
```

Rat[*expr*, (*crit*:1), (*levspec*:Inf), (*domcrit*:1)]
combines over a common denominator terms in the domain of *expr* specified by *levspec* and *domcrit* [2.5], and on which application of the template *crit* does not yield **0**.

```
#I[1]:: t:1/a+1/b+1/(a b)

                1   1     1
#O[1]:          - + - + ---
                a   b   a b

#I[2]:: Rat[t]

                1 + a + b
#O[2]:          ---------
                   a b
```

Col[*expr*, (*crit*:1), (*levspec*:Inf), (*domcrit*:1)]
collects terms with the same denominator in the domain of *expr* specified by *levspec* and *domcrit*, but on which application of the template *crit* does not yield **0**.

```
#I[1]:: t:(x+a)(x+b)(x+c)/(a*b)+(y+a)(y+b)/b

                (a + y) (b + y)   (a + x) (b + x) (c + x)
#O[1]:          --------------- + ----------------------
                       b                    a b

#I[2]:: u:Ex[t]

                                                    2    2    2    3    2
                          a y   c x   c x   c x     x    x    x    x    y
#O[2]:  a + c + x + y + --- + --- + --- + ---- + -- + -- + -- + --- + --
                         b     a     b    a b    a    b    a b   b

#I[3]:: Col[u]

                                  2     2    3                   2    2
                          c x + x     c x  + x     a y + c x + x  + y
#O[3]:  a + c + x + y + -------- + --------- + -------------------
                            a          a b              b
```

Cb[*expr*,{*form1*}, (*levspec*:Inf), (*domcrit*:1)]
combines coefficients of terms matching *form1*, *form2*, .. in the domain of *expr* specified by *levspec* and *domcrit*.

```
#I[1]:: t:Ex[(x+a+b)^4]

                3       3       3       2 2       2 2       3       3       2 2
#O[1]: 4a b  + 4a x  + 4b x  + 6 a  b  + 6 a  x  + 4 a  b + 4 a  x + 6 b  x

                    3           2       2       2       4   4   4
       + 4 b  x + 12a b x  + 12a b  x + 12 a  b x + a  + b  + x

#I[2]:: Cb[t,x]

                3       3       3           2       2       3       3       2 2
#O[2]: 4a b  + 4a x  + 4b x  + x (12a b  + 12 a  b + 4 a  + 4 b ) + 6 a  b

                  2 2       3       2 2       2   4   4   4
       + 6 a  x  + 4 a  b + 6 b  x  + 12a b x  + a  + b  + x

#I[3]:: Cb[t,{b^2,a^2,b}]

                3       3           2       3       3       2           2       3
#O[3]: 4a b  + 4a x  + b (12a x  + 4 a  + 4 x ) + a  (12b x + 6 x ) + 4 a  x

                  2           2       2       3       4   4   4
       + b  (12a x + 6 a  + 6 x ) + 4 b  x + a  + b  + x

#I[4]:: Cb[t,{x,x^$i}]

                3           2       2       3       3       2 2       3
#O[4]: 4a b  + x (12a b  + 12 a  b + 4 a  + 4 b ) + 6 a  b  + 4 a  b

                  2           2       2       3           4   4   4
       + x  (12a b + 6 a  + 6 b ) + x  (4a + 4b) + a  + b  + x
```

• Fac, Pf [9.1]

7.10 Statistical expression generation and analysis

Rex[(*n*:10),({{*unit1*,(*uwt1*:1)},...}),({{*temp1*,(*twt1*:1),(*n1*:1)},...})]
generates a pseudorandom expression containing on average *n* "units" selected from the *uniti* with statistical weights *uwti*, and combined by application of templates selected from the *tempi* with weights *twti*; each *tempi* acts on *ni* expressions (or an average of *-ni* expressions for negative *ni*). Simple defaults are provided for the *uniti* and *tempi*.
• Rand [8.3]

```
#I[1]::  Rex[]

                 1       2      2
#O[1]:   -7 + x + - - 2a x  - 12 x
                 x

#I[2]::  Rex[]

              2y (3 + 2a + 2y + 2z + 4x z)
#O[2]:   Exp[f[----------------------------]]
                      f[f[96x/7]]

#I[3]::  Rex[5]

                     2
#O[3]:   -2g[2,x + 6 z ]

#I[4]::  p[$n]::Rex[$n,{x,1,-1},{{Mult,1,-3},{Plus,1,-4}}]

#O[4]:   ' Rex[$n,{x,1,-1},{{Mult,1,-3},{Plus,1,-4}}]

#I[5]::  p[10]

                  4
#O[5]:   -2 + x + x  (1 + 2x)

#I[6]::  p[10]

               2
#O[6]:   (2 + x ) (3 + 4x)

#I[7]::  p[10]

          2
#O[7]:   x  (1 - 3x)

#I[8]::  p[20]

#O[8]:   -3 + 15x

#I[9]::  p[20]

                                                2
#O[9]:   -x (-3 + 2x) (1 - x) (1 + x) (x - x )
```

Aex[*expr1*,*expr2*,...]
performs a simple statistical analysis on the *expri* and yields a held [3.5] Rex projection necessary to generate further expressions with the same "statistical properties".

```
#I[1]:: Rex[a+b]

#O[1]:  ' Rex[2,{{a,1},{b,1}},{{Plus,1,2}}]

#I[2]:: Rex[]

                -12a x z f[2x]
#O[2]:  --------------------------
                    4 + x

#I[3]:: v:Rex[%]

#O[3]:  ' Rex[6,{{a,1},{x,3},{z,1},{4,1}},
              {{Div,1,2},{Mult,1,4},{f,1,1},{Plus,1,2}}]

#I[4]:: Rel[v]

                                  a
#O[4]:  f[f[f[-----------------------------]]]
                                        a
              a + x + f[a + x + --------]
                                      3
                             f[a x ]

#I[5]:: Rel[v]

          4  2        x
#O[5]:  a  x  f[---------]
                    2  2
               f[a  x ]

#I[6]:: Rex[@4,@5]

#O[6]:  ' Rex[9,{{a,7},{x,6},{3,1},{4,1},{2,3}},
              {{f,7,1},{Div,3,2},{Plus,2,3},{Mult,3,-1.333333},{Pow,5,2}}]
```

8. Mathematical functions

8.1 Introduction

Reductions of mathematical functions occur through simplification or numerical evaluation. Simplification yields an exact transformation; numerical evaluation is performed only in the absence of symbolic parameters, and may be approximate.

Simplifications for arithmetic operations [8.2] and numerical functions [8.3] are automatically performed. Elementary transcendental functions [8.5] are simplified only when the result is a rational number or multiple of a constant [8.4]. Many additional relations and transformations are given as replacements [3.3] defined in external files, and applied selectively by **S** projections [3.3].

Real or complex numerical values for any of the functions described below are obtained by **N** projections [3.4].

Whenever a mathematical "function" is used to represent solutions of an equation, it may take on several values for any particular set of arguments, as conventionally parametrized by Riemann sheets. For such multivalued "functions", numerical values are always taken on a single "principal" Riemann sheet; at the conventional positions of branch cuts, the limiting values in a counter-clockwise approach to the cut are given.

Differentiation, integration [9.4] and power series expansion [9.5] of mathematical functions is performed where possible.

All projections representing mathematical functions carry the property **Ldist** [4].

Definitions of mathematical functions are based primarily on four references:

AS "Handbook of Mathematical Functions", ed. M.Abramowitz and I.Stegun, NBS AMS 55 (1964); Dover (1965).

GR "Table of Integrals, Series and Products", I.Gradshteyn and I.Ryzhik, Academic Press (1965).

MOS "Formulas and Theorems for the Special Functions of Mathematical Physics", W.Magnus, F.Oberhettinger and R.P.Soni, 3rd ed, Springer-Verlag (1966).

BMP "Higher Transcendental Functions", Bateman Manuscript Project (A.Erdelyi et al.) Vols 1-3, McGraw-Hill (1953).

Citations in which notations or conventions differ from those used here are indicated by †.

8.2 Elementary arithmetic operations

expr1 **+** ***expr2*** **+ ...,** ***expr1*** **-** ***expr2*** **+ ...,** or **Plus[***expr1, expr2, ...***]**
<Comm,Flat>

expr1 ***** ***expr2*** *** ...,** or **Mult[***expr1, expr2, ...***]**
<Comm,Flat>

expr1**/*****expr2*** or **Div[***expr1, expr2***]**

expr1**^*****expr2*** or **Pow[***expr1, expr2***]**

Sqrt[*expr***]**

expr1 **.** ***expr2*** **. ...** or **Dot[***expr1, expr2, ...***]**
<Flat>
forms the inner product of *expr1,expr2,....* For two lists x and y the inner product is a list obtained by summing x[i[1],i[2],...,i[n-1],k] * y[k,j[2],...,j[m]] over all values of the index k for which entries are present in both x and y.
• Inner [9.6]

```
#I[1]:: {a,b,c}.{x,y,z}
#O[1]:  a x + b y + c z
#I[2]:: {v,w}.{{a,b},{c,d}}.{r,s}
#O[2]:  r (a v + c w) + s (b v + d w)
#I[3]:: {[x]:a,[y]:b}.{[x]:{u1,u2},[y]:{v1,v2}}
#O[3]:  {a u1 + b v1,a u2 + b v2}
#I[4]:: {b,c}.{a}
#O[4]:  {b,c}.{a}
```

expr1 ****** ***expr2*** **** ...**or **Omult[***expr1, expr2, ...***]**
<Comm,Flat>
forms the outer product of *expr1, expr2, ...*
• Outer [9.6]

```
#I[1]:: {a,b}**{c,d}
#O[1]:  {{a c,a d},{b c,b d}}
#I[2]:: {{1,2,3},{4}}**{a,b}**{c,d}
#O[2]:  {{{{a c,a d},{b c,b d}},{{2a c,2a d},{2b c,2b d}},
                        {{3a c,3a d},{3b c,3b d}}},
                {{{4a c,4a d},{4b c,4b d}}}}
```

Multiplication of an expression by a numerical coefficient does not involve an explicit Mult projection. Such products may nevertheless be matched by a pattern in which a generic symbol representing the coefficient appears in a Mult projection [2.6].

```
#I[1]:: t:x*a
#O[1]:  a x
#I[2]:: t[0]
#O[2]:  ' Mult
#I[3]:: u:5x/6
#O[3]:  5x/6
#I[4]:: u[0]
#O[4]:  5x/6
#I[5]:: h[$n*x]:j[$n]
#O[5]:  j[$n]
#I[6]:: {h[t],h[u],h[6],h[x/3]}
#O[6]:  {j[a],j[5/6],h[6],j[1/3]}
```

Plus[*expr*] and Mult[*expr*] are taken as *expr*; Plus[] is 0 and Mult[] is 1.

• Fctl, Dfctl, Comb [8.6]

8.3 Numerical functions

Gint[*x*]
the greatest integer not larger than the real number x
• Mod [8.10]

```
#I[1]:  {Gint[2.4], Gint[-2.4], Gint[-x]}
#O[1]:  {2,-3,Gint[-x]}
```

Sign[*x*]
1 or −1 if x is a positive or negative real number, and 0 if x is zero.

```
#I[1]:  {Sign[2.3], Sign[-2.3], Sign[-x]}
#O[1]:  {1,-1,Sign[-x]}
```

Theta[*x*]
Heavyside step function $\vartheta(x)$.

Delta[*x*]
Dirac's delta function $\delta(x)$.

Integrals and derivatives involving `Theta` and `Delta` are treated.

Abs[*x*]
the absolute value of a real or complex number x.

```
#I[1]:  {Abs[-4/5], Abs[2+I], Abs[a+b*I], Abs[-x]}
                                  1/2   2     2 1/2
#O[1]:  {4/5,5    ,(a  + b )    ,Abs[x]}
```

Conj[*expr*]
complex conjugate.

Re[*expr*]
real part.

Im[*expr*]
imaginary part.

```
#I[1]:  (a+b*I)*(c+d*I)/(a+f*I)
#O[1]:* a (a c + d f) - b (a d - c f)    a (a d - c f) + b (a c + d f)
       ------------------------------ + ------------------------------ I
                    2    2                           2    2
                   a  + f                           a  + f

#I[2]:  Realp[x]:Imagp[y]:1
#O[2]:  1
#I[3]:  {Conj[2I+3],Conj[a],Conj[a+b*I],Conj[x],Conj[y],Conj[x+I*y]}
#O[3]:* {3 - 2I,Conj[a],Conj[a] + -Conj[b] I,x,-y,x + y I}
#I[4]:  {Re[3+4*I],Re[a+I*b],Re[x],Re[y],Re[x+a+y]}
#O[4]:  {3,-Im[b] + Re[a],x,0,Re[a + x + y]}
```

Max[*x1*, *x2*, . . .]
<Comm, Flat>
the numerically largest of *x1*, *x2*, .. if this can be determined.

Min[*x1*, *x2*, . . .]
<Comm, Flat>
the numerically smallest of *x1*, *x2*, .. if this can be determined.

```
#I[1]:  Max[-2,3,5,3,2]

#O[1]:  5

#I[2]:  Min[x+2,y-1,y+3]

#O[2]:  Min[-1 + y,2 + x]
```

Rand[(*x*:1), (*seed*)]
pseudorandom number uniformly distributed between 0 and *x*. A number *seed* may be used to determine the sequence of numbers generated.

```
#I[1]:  Rand[]

#O[1]:  .9919248

#I[2]:  Rand[]

#O[2]:  .1170436

#I[3]:  Ar[5, `Rand[-10]]

#O[3]:  {-7.743285,-7.426181,-.3662663,-2.85419,-9.781653}
```

8.4 Mathematical constants

Pi	π = 3.14159..
E	e = 2.71828..
Euler	Euler-Mascheroni constant γ = 0.577216.. [AS 6.1.3; GR 9.73; MOS 1.1]
Deg	$\pi/180$ = 0.0174533..
Phi	Golden ratio φ = 1.61803..
Catalan	Catalan's constant 0.915966.. [AS 23.2.23; GR 9.73]

• I,Inf [2.2]

```
#I[1]:: Catalan-1
#O[1]:  -1 + Catalan
#I[2]:: N[%]
#O[2]:  -.88403441
#I[3]:: N[@1,15]
#O[3]:  -.884034405822780 97
```

8.5 Elementary transcendental functions

Exp[z]

Log[z,(*base*:E)]
logarithm with branch cut along negative real axis [AS 4.1.1].

Sin[z]	**Asin[z]**	**Sinh[z]**	**Asinh[z]**
Cos[z]	**Acos[z]**	**Cosh[z]**	**Acosh[z]**
Tan[z]	**Atan[z]**	**Tanh[z]**	**Atanh[z]**
Csc[z]	**Acsc[z]**	**Csch[z]**	**Acsch[z]**
Sec[z]	**Asec[z]**	**Sech[z]**	**Asech[z]**
Cot[z]	**Acot[z]**	**Coth[z]**	**Acoth[z]**

trigonometric and inverse trigonometric functions with arguments in radians [AS 4.4.1-4.4.6; AS 4.6.1-4.6.6].

• Deg [8.4]

Gd[z], Agd[z]
Gudermannian functions $\mathrm{gd}(z)$, $\mathrm{gd}^{-1}(z)$ [AS 4.3.117].

```
#I[1]:=  {Log[1], Log[5], Log[-5], Log[3+4I]}

#O[1]:=  {0,Log[5],Log[-5],Log[3 + 4I]}

#I[2]:=  N[%]

#O[2]:=  {0,1.609438,1.609438 + 3.141593I,1.609438 + .9272952I}

#I[3]:=  {Sin[5Pi/2], Sin[5Pi/3], Tan[Pi/2], Asin[1], Acos[6], Acos[6+I]}

#O[3]:=  {1,Sin[5Pi/3],Tan[Pi/2],Pi/2,Acos[6],Acos[6 + I]}

#I[4]:=  N[%]

#O[4]:=  {1,-.8660254,Tan[1.570796],1.570796,Acos[6],.167383 - 2.49216I}
```

8.6 Gamma, Zeta and related functions

Gamma[z,(a:0)]
Euler Γ function Γ(z) [AS 6.1.4; GR 8.310; MOS 1.1; BMP 1.1] and incomplete Γ function Γ(z,a) [AS 6.5.3; GR 8.350.2; MOS 9.1.1; BMP 6.9.2.21].

n! or **Fct[n]**
factorial [AS 6.1.6].

n!! or **Dfct[n]**
double factorial [AS 6.1.49 (footnote); GR p.xliii]

Poc[x,n]
Pochhammer symbol $(x)_n$ [AS 6.1.22; MOS 1.1].

Comb[n,m[1],(m[2],...m[k-1],(m[k]:n-m[1]-m[2]-...-m[k-1]))]
Multinomial coefficient $(n;m[1],m[2],\cdots,m[k-1],m[k])$ [AS 24.1.2]; **Comb[n,m]** gives binomial coefficient $\binom{n}{m}$ [AS 6.1.21; MOS 1.1]

Ei[z]
Exponential integral Ei(z) [AS 5.1.2; GR 8.2; MOS 9.2.1; BMP 6.9.2.(25)].

• **Erf** [8.7]

Expi[(n:1),z]
Exponential integrals $E_n(z)$ [AS 5.1.4; MOS 9.2.1].

Logi[z]
Logarithm integral function li(z) [AS 5.1.3; GR 8.24; MOS 9.2.1].

Sini[z]
Sine integral function Si(z) [AS 5.2.1; GR 8.230.1; MOS 9.2.2].

Cosi[z]
Cosine integral function Ci(z) [AS 5.2.2; GR 8.230.2; MOS 9.2.2].

Sinhi[z]
Hyperbolic sine integral function Shi(z) [AS 5.2.3; MOS 9.2.2].

Coshi[z]
Hyperbolic cosine integral function Chi(z) [AS 5.2.4; MOS 9.2.2].

Lob[z]
Lobachevskiy's function $L(z)$ [GR 8.26].

Beta[x,y,(a:1)]
Euler B function $B(x,y)$ [AS 6.2.1; GR 8.380; MOS 1.1; BMP 1.5] and incomplete B function $B(x,y,a)$ [AS 6.6.1; GR 8.391; MOS 9.2.5; BMP 2.5.3].

Psi[z,(n:1)]
Digamma function $\psi(z)$ [AS 6.3.1; GR 8.360; MOS 1.2; BMP 1.7.1] and polygamma functions $\psi^{(n-1)}(z)$ [AS 6.4.1; MOS 1.2; BMP 1.16.1].

Ler[z,(s:2),(a:0)]
Lerch's transcendent $\Phi(z,s,\alpha)$ [GR 9.55; MOS 1.6; BMP 1.11].

Zeta[z,(a:1)]
Riemann ζ function $\zeta(z)$ [AS 23.2; GR 9.513,9.522; MOS 1.3; BMP 1.12] and generalized ζ function $\zeta(z,\alpha)$ [GR 9.511,9.521; MOS 1.4; BMP 1.10].

Li[(n:2),z]
Dilogarithm (Spence's function) $\mathrm{Li}_2(z)$ [† AS 27.7; MOS 1.6; BMP 1.11.1] and polylogarithm function $\mathrm{Li}_n(z)$ [BMP 1.11.(14)].

Catb[n]
Catalan's β function $\beta(n)$ [AS 23.2.21].

EpsZ[{*g1*, *g2*, ...}, {*h1*, *h2*, ...}, *s*, *phi*]
Epstein's Z function $Z\left[\begin{matrix} g1 & g2 & \cdot\cdot \\ h1 & h2 & \cdot\cdot \end{matrix}\right](s)_{\varphi}$ [BMP 17.8].

Ber[*n*, (*x*:0)]
Bernoulli numbers B_n [AS 23.1.2; GR 9.61; MOS 1.5.1; BMP 1.13.(1)] and polynomials $B_n(x)$ [AS 23.1.1; GR 9.62; MOS 1.5.1; BMP 1.13.(2)].

Eul[*n*, (*x*)]
Euler numbers E_n [AS 23.1.2; GR 9.63; MOS 1.5.2; BMP 1.14.(1)] and Euler polynomials $E_n(x)$ [AS 23.1.1; MOS 1.5.2; BMP 1.14.(2)] (note relative normalization between numbers and polynomials).

```
#I[1]:: {10!, 100!, 7!!, 8!!}

#O[1]:  {3628800,100!,105,384}

#I[2]:: {Comb[17,4], Comb[1,1], Comb[1,2]}
```

8.7 Confluent hypergeometric and related functions

Chg [a, c, z]
Confluent hypergeometric (Kummer) function ${}_1F_1(a;c;z)$ [AS 13.1.2; GR 9.210; MOS 6.1.1].

KumU [a, b, z]
Kummer's U function $U(a,b,z)$ [AS 13.1.3; GR 9.210.(2); MOS 6.1.1].

WhiM [l, m, z]
Whittaker's M function $M_{l,m}(z)$ [AS 13.1.32; GR 9.220.(2); MOS 7.1.1].

WhiW [l, m, z]
Whittaker's W function $W_{l,m}(z)$ [AS 13.1.33; GR 9.220.(4); MOS 7.1.1].

Par [p, z]
Parabolic cylinder functions $D_p(z)$ [† AS 19.3.7; GR 9.240; MOS 6.1.1].

CouF [l, e, r]
Regular Coulomb wave function $F_L(\eta,r)$ [AS 14.1.3].

CouG [l, e, r]
Irregular Coulomb wave function $G_L(\eta,r)$ [AS 14.1.14].

BesJ [n, z]
Regular Bessel function $J_n(z)$ [AS 9.1.10; GR 8.402; MOS 3.1].

BesY [n, z]
Irregular Bessel function (Weber's function) $Y_n(z)$ [AS 9.1.11; GR 8.403.(1); MOS 3.1].

Besj [n, z]
Regular spherical Bessel function $j_n(z)$ [AS 10.1.1; MOS 3.3].

Besy [n, z]
Irregular spherical Bessel function $y_n(z)$ [AS 10.1.1; MOS 3.3].

BesK [n, z]
Modified Bessel function $K_n(z)$ [AS 9.6.2; GR 8.407.(1); MOS 3.1].

BesI [n, z]
Modified Bessel function $I_n(z)$ [AS 9.6.3; GR 8.406; MOS 3.1].

BesH1 [n, z]
Hankel function $H_n^{(1)}(z)$ [AS 9.1.3; GR 8.405.(1)].

BesH2 [n, z]
Hankel function $H_n^{(2)}(z)$ [AS 9.1.4; GR 8.405.(1)].

Kelbe [n, z]
Complex Kelvin functions $\mathrm{ber}_n(z) + i\ \mathrm{bei}_n(z)$ [AS 9.9.1; GR 8.561].

Kelke [n, z]
Complex Kelvin functions $\mathrm{ker}_n(z) + i\ \mathrm{kei}_n(z)$ [AS 9.9.2; GR 8.563.(2)].

StrH [n, z]
Struve function $\mathrm{H}_n(z)$ [AS 12.1; GR 8.550.(1)].

StrL [n, z]
Modified Struve function $\mathrm{L}_n(z)$ [AS 12.2.1; GR 8.550.(2)].

AngJ [n, z]
Anger function $\mathrm{J}_n(z)$ [AS 12.3.1; GR 8.580.(1)].

WebE [n, z]
Weber's function $\mathrm{E}_n(z)$ [AS 12.3.3; GR 8.580.(2)].

Lom [m, n, z]
Lommel's function $s_{m,n}(z)$ [GR 8.570.(1); MOS 3.10.1].

AirAi[z]
Airy's function Ai(z) [AS 10.4.2].

AirBi[z]
Airy's function Bi(z) [AS 10.4.3].

Erf[z]
Error function erf(z) [AS 7.1.1; GR 8.250.(1)].

Erfc[z]
Complementary error function erfc(z) [AS 7.1.2].

• **Gamma, Ei** [8.6]

FreC[z]
Fresnel's function $C(z)$ [AS 7.3.1; GR 8.250.(1)].

FreS[z]
Fresnel's function $S(z)$ [AS 7.3.2; GR 8.250.(2)].

Lag[*n*, (*a*:1), *z*]
(Generalized) Laguerre function $L_n^{(a)}(z)$ [AS 22.2.12; GR 8.970].

Her[*n*, *z*]
Hermite's function $H_n(z)$ [AS 22.2.14; GR 8.950].

Pcp[*n*, *nu*, *z*]
Poisson-Charlier polynomials $\rho_n(\nu,z)$ [AS 13.6.11; MOS 6.7.2].

Tor[*m*, *n*, *z*]
Toronto function $T(m,n,z)$ [AS 13.6.20; MOS 6.7.2].

Batk[*n*, *z*]
Bateman's function $k_\nu(z)$ [AS 13.6.33; MOS 6.7.2].

8.8 Hypergeometric and related functions

Hg[a, b, c, z]
Gauss hypergeometric function ${}_2F_1(a,b;c;z)$ [AS 15.1.1; GR 9.10; MOS 2.1].

JacP[n, a, b, z]
Jacobi functions $P_n^{(a,b)}(z)$ [AS 22.2.1; GR 8.960; MOS 5.2.1].

Geg[n, l, x]
Gegenbauer (ultraspherical) functions $C_n^{(l)}(x)$ [AS 22.2.3; GR 8.930; MOS 5.3.1].

CheT[n, x]
Chebyshef function of first kind $T_n(x)$ [AS 22.2.4; MOS 5.3.1].

CheU[n, x]
Chebyshef function of second kind $U_n(x)$ [AS 22.2.5; MOS 5.3.1].

LegP[l, (m:0), z]
(Associated) Legendre functions $P_l^m(z)$ [AS 8.1.2; GR 8.702; MOS 4.1.2, 5.4.1].

LegQ[l, (m:0), z]
(Associated) Legendre functions of second kind $Q_l^m(z)$ [AS 8.1.3; GR 8.703; MOS 4.1.2, 5.4.2].

• Beta [8.6]

EllK[k, (t:Pi/2)]
First kind elliptic integral $K(k\,|\,t)$ [† AS 17.2.6; † GR 8.111.(2); MOS 10.1]

EllE[k, (t:Pi/2)]
Second kind elliptic integral $E(k\,|\,t)$ [† AS 17.2.8; † GR 8.111.(3); MOS 10.1]

EllPi[k, (t:Pi/2)]
Third kind elliptic integral $\Pi(k\,|\,t)$ [† AS 17.2.14; † GR 8.111.(4); MOS 10.1]

JacSn[x, (m:0)] **JacCn[x, (m:0)]** **JacDn[x, (m:0)]**
JacCd[x, (m:0)] **JacSd[x, (m:0)]** **JacNd[x, (m:0)]**
JacDc[x, (m:0)] **JacNc[x, (m:0)]** **JacSc[x, (m:0)]**
JacNs[x, (m:0)] **JacDs[x, (m:0)]** **JacCs[x, (m:0)]**

JacAm[x, (m:0)]
Jacobian elliptic functions Sn(x|m) etc. [AS 16.1; GR 8.144; MOS 10.3].

Jacth[i, u]
Jacobi ϑ functions $\vartheta_i(u)$ [AS 16.27; GR 8.18; MOS 10.2].

WeiP[u]
Weierstrass function $P(u)$ [AS 18; GR 8.160; MOS 10.5].

Weiz[u]
Weierstrass ζ function $\zeta(u)$ [GR 8.171.(1); MOS 10.5].

Weis[u]
Weierstrass σ function $\sigma(u)$ [GR 8.171.(2); MOS 10.5].

```
#I[1]:  {EllK[1/4], LegP[2,0], LegP[2.2,0], Hg[3,5,4,2.8]}

#O[1]:  {EllK[.25],LegP[2,0],LegP[2.2,0],Hg[3,5,4,2.8]}

#I[2]:  N[%]

#O[2]:  {1.596242,-.5,-.4581419,.02857796}
```

8.9 Further special functions

Ghg[*p*, *q*, {*a1*, *a2*, ...}, {*b1*, *b2*, ...}, *z*]
Generalized hypergeometric function [MOS 2.9].

Mei[*m*, *n*, *p*, *q*, {*a1*, ..., *ap*}, {*b1*, ..., *bq*}, *z*]
Meijer G function [GR 9.30].

MacE[*a*, *b*, *z*]
MacRobert E function [GR 9.4; MOS 6.7.2].

Matce[*n*, *x*, *q*]
Matse[*n*, *x*, *q*]
Mathieu functions $\mathrm{ce}_n(x,q)$, $\mathrm{se}_n(x,q)$ [AS 20; GR 8.61].

Wig[{*j1*, *m1*}, {*j2*, *m2*}, {*j3*, *m3*}]
Wigner 3-j symbol (Clebsch-Gordan coefficient) $\begin{pmatrix} j1 & j2 & j3 \\ m1 & m2 & m3 \end{pmatrix}$ [AS 27.8.1].

Rac[*j1*, *j2*, *j3*, *j4*, *j5*, *j6*]
Racah 6-j symbol $\begin{Bmatrix} j1 & j2 & j3 \\ j4 & j5 & j6 \end{Bmatrix}$

```
#I[1]:  Wig[{5,1},{4,-2},{6,1}]

                  1/2
#O[1]:  16/6435
```

8.10 Number theoretical functions

Mod[*n*, *m*]
the integer n modulo the integer m.

Gcd[*n1*, *n2*, . . .]
the greatest common divisor of the integers $n1$, $n2$, ...

Divis[*n*]
a list of the integer divisors of an integer n.

Nfac[*n*]
a list of the prime factors of an integer or rational number n, together with their exponents.

Prime[*n*]
the n th prime number

Stil[*n*, *m*]
First kind Stirling numbers $S_n^{(m)}$ [AS 24.1.3].

Sti2[*n*, *m*]
Second kind Stirling numbers $S_n^{(m)}$ [AS 24.1.4].

Mob[(*k*:1), *n*]
Mobius μ function $\mu_k(n)$ of order k [AS 24.3.1].

Jacsym[*p*, *q*]
Jacobi symbol $\left(\frac{p}{q}\right)$ [BMP 17.5].

Divsig[(*k*:1), *n*]
Divisor function $\sigma_k(n)$ ($\sigma_0(n)=d(n)$) [AS 24.3.3].

ManL[*n*]
Mangoldt Λ function [BMP 17.1.1].

Part[*n*]
Partition function [AS 24.2.1].

Rrs[*n*]
List containing reduced residue system modulo n.

Totient[*n*]
Euler's totient function $\varphi(n)$ [AS 24.3.2].

Lio[*n*]
Liouville's function $\nu(n)$ [BMP 17.1.1].

Jor[*k*, *n*]
Jordan's function $J_k(n)$ (kth totient of n) [BMP 17.1.1].

- Comb [8.6]

```
#I[1]:  {Gcd[15,25],Gcd[9/5,6/7]}
#O[1]:  {5,3/35}
#I[2]:  {Mod[5,3],Mod[-5,3],Mod[N[Pi],1]}
#O[2]:  {2,1,.1415927}
#I[3]:  Divis[12]
#O[3]:  {1,2,4,3,6,12}
```

```
#I[4]::  Nfac[666]

#O[4]:   {{2,1},{3,2},{37,1}}

#I[5]::  Nfac[35/99]

#O[5]:   {{5,1},{7,1},{3,-2},{11,-1}}

#I[6]::  Prime[188]

#O[6]:   541

#I[7]::  {Sti1[9,4],Sti2[9,4]}

#O[7]:   {-67284,7778}

#I[8]::  {Mob[7],Mob[12],Mob[2,12]}

#O[8]:   {-1,0,-1}

#I[9]::  Jacsym[888,1999]

#O[9]:   -1

#I[10]::  {Divsig[12],Divsig[Pi,12]}

                          3Pi          2Pi
                (-1 + 2    ) (-1 + 3    )
#O[10]:  {28,-----------------------}
                      Pi           Pi
                 (-1 + 2  ) (-1 + 3  )

#I[11]::  ManL[125]

#O[11]:   Log[5]

#I[12]::  Part[58]

#O[12]:   284226

#I[13]::  Rrs[12]

#O[13]:   {1,5,7,11}

#I[14]::  {Len[%],Totient[12],Jor[1,12]}

#O[14]:   {4,4,4}

#I[15]::  Jor[Pi,12]

                 Pi        -Pi          -Pi
#O[15]:  12    (1 - 2    ) (1 - 3    )

#I[16]::  Lio[12]

#O[16]:   -1
```

9. Mathematical operations

9.1 Polynomial manipulation

Polynomials consist of sums of powers of "base" expressions. The bases in an expression are by default taken as the literal first filters of **Pow** projections.

• **Polyp** [7.6.]

Pdiv[*expr1*, (*expr2*:1), (*form*)**]**
the polynomial quotient of *expr1* and *expr2* with respect to the "base" *form*.

```
#I[1]:  Pdiv[x+a,x+b,x]

#O[1]:  1

#I[2]:  Pdiv[x+a,x+b,a]

          a         x
#O[2]:  ----- + -----
        b + x   b + x

#I[3]:  Pdiv[x^3,x-1,x]

                   2
#O[3]:  1 + x + x

#I[4]:  Pdiv[x^4+x^2,x+Sqrt[x]+2,Sqrt[x]]

                          1/2       3/2       2    5/2    3
#O[4]:  252 + 28x - 84 x      - 9 x     + 3 x  - x    + x

#I[5]:  Pdiv[a^(3b),a^b-1,a^b]

             b     2b
#O[5]:  1 + a  + a
```

Pmod[*expr1*, *expr2*, (*form*)**]**
the remainder from division of *expr1* by *expr2* with respect to *form* (polynomial modulus).
• Mod [8.10]

```
#I[1]:  Pmod[x+a,x+b,x]

#O[1]:  a - b

#I[2]:  Pmod[x+a,x+b,a]

#O[2]:  0

#I[3]:  Pmod[x^7,x^4+1,x]

             3
#O[3]:  - x
```

Pgcd[*expr1*, *expr2*, (*form*)**]**
greatest common divisor of the polynomials *expr1* and *expr2* with respect to *form*.
• Gcd [8.10]

```
#I[1]:  t1:Ex[(x+1)(x+2)(x+3)]

                          2    3
#O[1]:  6 + 11x + 6 x  + x
```

```
#I[2]:: t2:Ex[(x+1)(x+2)(x-4)]

                          2   3
#O[2]:  -8 - 10x - x  + x

#I[3]:: Pgcd[t1,t2,x]

                        2
#O[3]:  14 + 21x + 7 x

#I[4]:: Fac[%]

#O[4]:  7(1 + x) (2 + x)

#I[5]:: Pgcd[x+a,x+b,x]

#O[5]:  a - b

#I[6]:: Pgcd[x+a,x+b,a]

#O[6]:  b + x
```

Fac[*expr*, (*lev*:1), ({*form1*}), (*crit*:1), ({*rep1*})]
factors polynomials appearing at or below level *lev* in *expr* (and not yielding "false" on application of the template *crit*) with respect to "bases" matching *form1*, *form2*, .. The smallest available bases are taken as default. (The replacements *rep1*, *rep2*, .. will specify polynomial equations defining algebraic extensions to the default real integer factorization field.)

```
#I[1]:: Fac[x^4-1]

                              2
#O[1]:  (-1 + x) (1 + x) (1 + x )

#I[2]:: t:(1+x)(x-7)^2*(x^2+2)

                    2                 2
#O[2]:  (-7 + x)  (1 + x) (2 + x )

#I[3]:: u:Ex[t]

                          2       3       4     5
#O[3]:  98 + 70x + 23 x  + 37 x  - 13 x  + x

#I[4]:: Fac[u]

                    2                 2
#O[4]:  (-7 + x)  (1 + x) (2 + x )

#I[5]:: Fac[x^4-1,{x^2}]

                  2         2
#O[5]:  (-1 + x ) (1 + x )

#I[6]:: Fac[x^2-1,{x^(1/3)}]

                 1/3       1/3       1/3    2/3       1/3    2/3
#O[6]:  (-1 + x   ) (1 + x   ) (1 - x    + x   ) (1 + x    + x   )

#I[7]:: v:(a+1)(b+2)(c-7)^2

                    2
#O[7]:  (-7 + c)  (1 + a) (2 + b)
```

```
#I[8]::  w:Ex[v]

#O[8]:   98 + 98a + 49b - 28c + 49a b - 28a c + 2a c^2 - 14b c + b c^2 - 14a b c

                    2      2
            + a b c  + 2 c

#I[9]::  Fac[w]

                  2
#O[9]:   (-7 + c)  (1 + a) (2 + b)
```

• Nfac [8.10]

Pf[*expr*, (*form*)]

yields a partial fraction form of *expr* with respect to the "base" *form*.

```
#I[1]::  t:1/((x-a)(x-b))

                  1
#O[1]:   ------------------
         (-a + x) (-b + x)

#I[2]::  Pf[t,x]

                -1                    1
#O[2]:   ------------------ + ------------------
         (-a + b) (-a + x)   (-a + b) (-b + x)

#I[3]::  Pf[x^3/(1-x),x]

                    1      2
#O[3]:   -1 - x + ----- - x
                  1 - x

#I[4]::  u:(x^2+1)/(1+Sqrt[x]+x)

                 2
            1 + x
#O[4]:   ------------
                1/2
         1 + x + x

#I[5]::  Pf[u,x]

                                1/2          1/2
                  1 - (-1 - x    ) (1 + x    )     1/2
#O[5]:   -1 + x + ------------------------------ - x
                                  1/2
                       1 + x + x

#I[6]::  Pf[u,Sqrt[x]]

                      1/2
                5 - 4 x          1/2
#O[6]:   2 + x + ------------ - x
                       1/2
              1 + x + x
```

9.2 Evaluation of sums and products

Sum[*expr*, *var*, (*start*:0), (*end*:Inf), (*step*:1), (*test*:1)]
sums the values of *expr* obtained when *var* takes on values from *start* to *end* with increment *step* (and such that the value of *test* is not 0).

```
#I[1]:: Sum[f[i],i,1,5]
#O[1]:  f[1] + f[2] + f[3] + f[4] + f[5]
#I[2]:: Sum[f[i],i,x,x+5,2]
#O[2]:  f[x] + f[2 + x] + f[4 + x]
#I[3]:: Sum[f[i],i,1,20,2,Mod[i,3]]
#O[3]:  f[1] + f[5] + f[7] + f[11] + f[13] + f[17] + f[19]
#I[4]:: Sum[1/i^2,i,1,100]
#O[4]:  1.634984
#I[5]:: Sum[f[i],i,1,n]
#O[5]:  Sum[f[i],i,1,n]
#I[6]:: Sum[S[x,$x->f[$x],i],i,1,4]
#O[6]:  f[x] + f[f[x]] + f[f[f[x]]] + f[f[f[f[x]]]]
```

Prod[*expr*, *var*, (*start*:0), (*end*:Inf), (*step*:1), (*test*:1)]
forms the product of the values of *expr* obtained when *var* takes on values from *start* to *end* with increment *step* (and such that the value of *test* is not 0).

```
#I[1]:: Prod[(x-2i),i,1,4]
#O[1]:  (-8 + x) (-6 + x) (-4 + x) (-2 + x)
```

Sums and products with infinite limits may be evaluated numerically with an N projection [3.4].

• Do [6.2]

9.3 Solution of equations

Sol [{*eqn1*},{*form1*}, ({*elim1*})]
takes the equations *eqn1*, .. (represented as Eq projections [6]) and yields a list of simplified equations or, if possible, replacements giving solutions for forms matching *form1*, .. after eliminating forms matching *elim1*, .. where possible. Undetermined parameters in solutions appear as indices in the resulting list.

Solutions for classes of equations may be defined by assignments for the relevant Sol projections [3.2]. The assignment `Sol[f[$x]=$y,$x]::Sol[$x=fi[$y],$x]` thus defines an "inverse" for the "function" `f`.

• Mdiv [9.6]

```
#I[1]:  Sol[a*x+b=c,x]

                  -b + c
#O[1]:  {x -> -------}
                   a

#I[2]:  t:Ex[(x-4)(2x-3)(5-x)(x+1)]

                          2      3      4
#O[2]:  60 - 7x - 46 x  + 19 x  - 2 x

#I[3]:  Sol[t=0,x]

#O[3]:  {x -> -1,x -> 3/2,x -> 4,x -> 5}

#I[4]:  Sol[x^3=3,x]

                 1/3                    1/3                    1/3
#O[4]:  {x -> 3    Exp[4Pi/3 I],x -> 3    Exp[2Pi/3 I],x -> 3    }

#I[5]:  N[%]

#O[5]:  {x -> -.7211248 - 1.249025I,x -> -.7211248 + 1.249025I,x -> 1.44225}

#I[6]:  Sol[f[x]=y,x]

#O[6]:  {f[x] = y}

#I[7]:  Sol[f[$x]=$y,$x]::Sol[$x=finv[$y],$x]

#O[7]:  ' Sol[$x = finv[$y],$x]

#I[8]:  Sol[f[a*x+b]=y+c,x]

                  -b + finv[c + y]
#O[8]:  {x -> ----------------}
                        a

#I[9]:  u:{2x+3y=5,7y-x=9a}

#O[9]:  {2x + 3y = 5,7y = 9a + x}

#I[10]:  Sol[u,{x,y}]

                  70/17 - 54a/17       2(5/2 + 9a)
#O[10]:  {x -> --------------,y -> -----------}
                        2                17

#I[11]:  Sol[{x^3+2x^2 y + 2y (y-2) x+y^2=4,x^2+2x y+2y^2-5y+2=0},{x,y}]

#O[11]:  {{x -> -1,y -> 3},{x -> -5,y -> 3},{x -> 8,y -> 2},{x -> -4,y -> 2}}
```

```
#I[12]::  Sol[{x=t^2-t+1, y=2t^2+t-3},,t]

#O[12]:   {(-1 + x) (-18 + 4x - 2y) + (-3 - y)^2
                                = 7 + 2x + 3y + (-1 + x) (7 + 2y)}
```

9.4 Differentiation and integration

A "variable" is an expression containing a single symbol (either on its own or in a projection; in the latter case the necessary Jacobian factors are extracted).

D[*expr*,{*var1*, (*n1*:1), (*pt1*:*var1*)},{*var2*, (*n2*:1), (*pt2*:*var2*)}...]
forms the partial derivative of *expr* successively *ni* times with respect to the "variables" *vari*, evaluating the final result at the point *vari* → *pti*.

Dt[*expr*,{*var1*, (*n1*:1), (*pt1*:*var1*)},{*var2*, (*n2*:1), (*pt2*:*var2*)}...]
forms the total derivative of *expr* with respect to the variables *vari*.

Dt[*expr*]
forms the total differential of *expr*.

Derivatives which cannot be performed explicitly are converted into a canonical form with internally-generated symbols [2.2] for the *vari*, and explicit values for *ni* and *pti*.

Derivatives may be defined by assignments for the relevant D or Dt projections. D[f[$x,$y],{$x,1,$z}]:g[$z,$y] defines the derivative of the "function" f with respect to its first "argument".

In D projections, distinct symbols are assumed independent, while in Dt projections, they are assumed to be interdependent, unless the corresponding derivative has explicitly been assigned the value 0. Symbols or projections carrying the property Const [4] are assumed independent of all variables.

N projections [3.4] yield when possible numerical values for derivatives with definite *pti*.

```
#I[1]:   t:a x^b

                 b
#O[1]:   a x

#I[2]:   D[t,x]

                -1 + b
#O[2]:   a b x

#I[3]:   D[t,x,x]

                -2 + b
#O[3]:   a b x         (-1 + b)

#I[4]:   D[t,{x,3}]

                -3 + b
#O[4]:   a b x         (-2 + b) (-1 + b)

#I[5]:   D[t,{x,0}]

                 b
#O[5]:   a x

#I[6]:   D[t,{x,n}]

                 b
#O[6]:   D[#1    a,{#1,n,x}]

#I[7]:   D[t,Log[x]]

                 b
#O[7]:   a b x
```

```
#I[8]:: D[t,x,b]

                  -1 + b          -1 + b
#O[8]:  a x          + a b x          Log[x]

#I[9]:: D[t,{x,1,y}]

                -1 + b
#O[9]:  a b y

#I[10]:: D[t,{x,0,y}]

                b
#O[10]:  a y

#I[11]:: D[t,{x,1,y},{y,1,x}]

                     -2 + b
#O[11]:  a b x          (-1 + b)

#I[12]:: D[t,{x,2,x^2}]

                     -4 + 2b
#O[12]:  a b x          (-1 + b)

#I[13]:: D[{x^2,x^3},x]

                   2
#O[13]:  {2x,3 x }

#I[14]:: D[Sin[Exp[x]],x]

#O[14]:  Cos[Exp[x]] Exp[x]

#I[15]:: D[Exp[a x]=b x^2,x]

#O[15]:  a Exp[a x] = 2b x

#I[16]:: D[f[x],x]

#O[16]:  D[f[#1],{#1,1,x}]

#I[17]:: D[f[x^2],x]

                                 2
#O[17]:  2x D[f[#1],{#1,1,x }]

#I[18]:: S[%,f->Log]

         2
#O[18]:  -
         x

#I[19]:: D[f[g[x]],x]

#O[19]:  D[f[#1],{#1,1,g[x]}] D[g[#1],{#1,1,x}]

#I[20]:: D[f[x,x],x]

#O[20]:  D[f[#1,x],{#1,1,x}] + D[f[x,#2],{#2,1,x}]

#I[21]:: D[f[g1[x],g2[x]],x]

#O[21]:  D[f[#1,g2[x]],{#1,1,g1[x]}] D[g1[#1],{#1,1,x}]
                + D[f[g1[x],#2],{#2,1,g2[x]}] D[g2[#1],{#1,1,x}]
```

```
#I[22]:: D[f[x],g[x]]

#O[22]:  D[f[#1],{#1,1,x}]
         ----------------
         D[g[#1],{#1,1,x}]

#I[23]:: D[f[g[x]],g[x]]

#O[23]:  D[f[#1],{#1,1,g[x]}]

#I[24]:: D[f[x],x,x]

#O[24]:  D[f[#1],{#1,2,x}]

#I[25]:: D[f[x,x],x,x]

#O[25]:  D[f[#1,x],{#1,2,x}] + D[f[x,#2],{#2,2,x}]
                + D[f[#1,#2],{#1,1,x},{#2,1,x}]
                + D[f[#1,#2],{#2,1,x},{#1,1,x}]

#I[26]:: D[f[g[x]],x,x]

#O[26]:  D[f[#1],{#1,1,g[x]}] D[g[#1],{#1,2,x}]

                + D[f[#1],{#1,2,g[x]}] D[g[#1],{#1,1,x}]^2

#I[27]:: D[p[$x],{$x,1,$y}] : q[$y]

#O[27]:  q[$y]

#I[28]:: D[p[x],x]

#O[28]:  q[x]

#I[29]:: D[p[x^2],x]

                  2
#O[29]:  2x q[x^2]

#I[30]:: D[%,x]

                    2                  2          2
#O[30]:  2(2 x^2 D[q[#1],{#1,1,x^2}] + q[x^2])

#I[31]:: D[p[x,2],x]

#O[31]:  D[p[#1,2],{#1,1,x}]

#I[32]:: D[j[$x,$y],{$x,1,$z}]:j1[$z,$y]

#O[32]:  j1[$z,$y]

#I[33]:: D[j[$x,$y],{$y,1,$z}]:j2[$x,$z]

#O[33]:  j2[$x,$z]

#I[34]:: D[j[x,3],x]

#O[34]:  j1[x,3]

#I[35]:: D[j[x,x],x]

#O[35]:  j1[x,x] + j2[x,x]
```

```
#I[36]::  D[j[x^2,x] p[x+j[x,x-1]],x]

                    2              2
#O[36]:   (2x j1[x ,x] + j2[x ,x]) p[x + j[x,-1 + x]]

                                                    2
                  + (1 + j1[x,-1 + x] + j2[x,-1 + x]) j[x ,x]

                           * q[x + j[x,-1 + x]]


#I[1]::  D[y,x]

#O[1]:   0

#I[2]::  Dt[y,x]

#O[2]:   Dt[y,x]

#I[3]::  t:a x^b

                b
#O[3]:   a x

#I[4]::  Dt[t,x]

                      -1 + b      b                      b
#O[4]:   a (b x             + x  Dt[b,x] Log[x]) + x  Dt[a,x]

#I[5]::  Dt[a,x]:0

#O[5]:   0

#I[6]::  Dt[t,x]

                      -1 + b     b
#O[6]:   a (b x             + x  Dt[b,x] Log[x])

#I[7]::  Dt[t]

                      -1 + b           b                 b
#O[7]:   a (b x             Dt[x] + x  Dt[b] Log[x]) + x  Dt[a]
```

Int[*expr*,{*var1*, (*lower1*), (*upper1*:*var1*)}...]
forms the integral of *expr* successively with respect to the variables *var1*,.. between the limits *lower1*,.. and *upper1*,... If no lower limit is specified, an internally-generated symbol [2.2] is used.

Integrals which cannot be performed explicitly are converted into a canonical form with internally-generated symbols for the *vari*.

Integrals may be defined by assignments for the relevant Int projections.

All distinct symbols are assumed independent.

N projections [3.4] yield when possible numerical values for integrals with definite limits *loweri* and *upperi*.

```
#I[1]::  t:a x^b

                b
#O[1]:   a x
```

```
#I[2]::  Int[t,x]

                 1 + b    1 + b
            - #1         x
#O[2]:   a (---------- + ------)
              1 + b      1 + b

#I[3]::  Int[t,{x,p1}]

               1 + b     1 + b
            - p1        x
#O[3]:   a (---------- + ------)
              1 + b      1 + b

#I[4]::  Int[t,{x,p1,p2}]

               1 + b       1 + b
            - p1        p2
#O[4]:   a (---------- + -------)
              1 + b       1 + b

#I[5]::  Int[x^2,x,x]

             3              4     4
         - #2  (-#3 + x)   #3    x
#O[5]:   ----------------- - --- + --
                 3            12    12

#I[6]::  D[%,x]

             3    3
         - #2    x
#O[6]:   ----- + --
           3      3

#I[7]::  D[%,x]

          2
#O[7]:   x

#I[8]::  Int[{x^a,x^b},{x,0,1}]

                     1 + a           1 + b
           1        0        1      0
#O[8]:   {----- - -----, ----- - -----}
          1 + a   1 + a  1 + b   1 + b

#I[9]::  t:(x^4+4)/((x-5)(x^2-x-1))

                      4
                 4 + x
#O[9]:   ----------------------
                               2
         (-5 + x) (-1 - x + x )

#I[10]::  Int[t,{x,0}]

                                  1/2                       1/2
                           -1 - 5                  -1 + 2x - 5
                   101Log[---------]      101Log[---------------]
                                  1/2                       1/2
                           -1 + 5                  -1 + 2x + 5          629Log[-5]
#O[10]:   105x/19 + ----------------- - ----------------------- - ----------
                                1/2                         1/2          19
                          38 5                        38 5

                                                          2        2
             22Log[-1]    629Log[-5 + x]   22Log[-1 - x + x ]   15 x
           + --------- + -------------- - ------------------ + ----
                19             19                 19            38
```

```
#I[11]::  Int[t,{x,0,1}]

                                     1/2                 1/2
                               -1 - 5               1 - 5
                      191Log[-----------]   191Log[----------]
                                     1/2                 1/2
                               -1 + 5               1 + 5        629Log[-5]
#O[11]:   225/38 + ----------------------- + -------------------- + ----------
                                1/2                 1/2             19
                            38 5                38 5

                  629Log[-4]
                + ----------
                     19

#I[12]::  N[%]

#O[12]:   .82179

#I[13]::  Int[(x+3)(x+2)/(x+1),{x,0,1}]

              (2 + #4) (3 + #4)
#O[13]:   Int[-----------------,{#4,0,1}]
                   1 + #4

#I[14]::  Ex[%]

#O[14]:   9/2 + 2Log[2]

#I[15]::  Int[f[x],x]

#O[15]:   Int[f[#6],{#6,#5,x}]

#I[16]::  Int[f[x],g[x]]

#O[16]:   Int[D[g[#10],{#10,1,#8}] f[#9],{#9,#7,g[x]}]

#I[17]::  Int[x f[x],x]

#O[17]:   Int[#12 f[#12],{#12,#11,x}]

#I[18]::  Int[a f[b x] + g[x+1] + x, x]

                                                                          2     2
                                                                       #13     x
#O[18]    Int[g[1 + #15],{#15,#13,x}] + a Int[f[#14 b],{#14,#13,x}] - ---- + --
                                                                         2      2

#I[19]::  Int[f[$x,$a],{$x,0,1}]:h[$a]

#O[19]:   h[$a]

#I[20]::  Int[3f[x,2]+5a f[x,a-1]+1/(x+1),{x,0,1}]

#O[20]:   3Int[f[#17,2],{#17,0,1}] + Log[2] + 5a h[-1 + a]

#I[21]::  Int[Gamma[x+1],{x,0,3}]

#O[21]:   Int[Gamma[1 + #18],{#18,0,3}]

#I[22]::  N[%]

#O[22]:   5.852363
```

9.5 Series approximations and limits

Ps[(*expr*:1),{*var1*},{*pt1*},{(*sord1*:0), *ord1*},
(*ser*:{[*sord1*]:0,..,[-1]:0,[0]:1,[1]:0,..,[*ord1*]:0})]

Power (Taylor-Laurent) series in *var1*,... about the points *pt1*,... to order *ord1*,.... Terms proportional to *var1^j1 var2^j2* ... are given when *j1,j2*,... lie within a simplex with vertices *(j1:ord1, j1:ord1, ji:sordi), (j2:ord2, ji:sordi)*, ... *ser[i]* gives the coefficient of *var1^i* in the power series.

Ra[*expr*, *var*, *pt*, {*degn*, (*degd*: *degn*)}, (*crit*: $1=*degn*&&$2=*degd*),
(*sern*:{[0]:1,..,[*degn*]:0}), (*serd*: {[0]:1,..,[*degd*]:0})]

Rational (Pade) approximants in *var* about the point *pt*, to order *degn* in numerator and *degd* in denominator series. All order *(m,n)* approximants with *m+n<degn+degd* such that application of the template *crit* to *m,n* yields "true" are given (in a list if necessary). *sern[i]* is the coefficient of *var^i* in the numerator series, and *serd[j]* of *var^j* in the denominator.

Cf[*expr*, (*var*:1), (*pt*:0), {(*sord*:0), (*ord*:0)}, (*ser*:{[0]:1, [1]:0, .., [*ord*]:0})]

continued fraction approximation in *var* about the point *pt*. *ser[i]* gives the coefficient of *var* in the *i*th partial quotient of the continued fraction.

expr gives an overall factor for the series. Input Ps, Ra and Cf projections are simplified so that all possible terms are transferred from *expr* to coefficients in the series *serk*.

Arithmetic and mathematical operations and substitutions (compositions) may be performed on series approximations: the results are taken to the highest permissible order.

Ax[*expr*]

yields an ordinary expression obtained by truncating all higher order terms in the series approximation *expr*.

If the expression *expr* in Ps, Ra and Cf is a series approximation, it is converted to the specified form, maintaining the highest permissible order.

Series approximations may be defined by assignments for Ps projections. Ps[exp[$x],$x,0,$n]:Ps[1,$x,0,$n,{[$i]:1/$i!}] defines the power series for the exponential function around the origin. Ra and Cf use assignments made for Ps projections.

Numerical values for series approximations are obtained using N.

```
#I[1]:  p:Ps[Log[1+x],x,0,6]

                    2   3   4   5   6
                   x   x   x   x   x
#O[1]:* x - x/2 + x/3 - x/4 + x/5 - x/6

#I[2]:  Lpr[p]

Ps[1,x,0,{1,6},{[1]: 1,[2]: -1/2,[3]: 1/3,[4]: -1/4,[5]: 1/5,[6]: -1/6}]

#I[3]:  Exp[p/5]

                          2       3       4        5         6
                       2 x     6 x     21 x     399 x     1596 x
#O[3]:* 1 + x/5 - ------ + ------ - ------- + ------- - --------
                     25      125      625     15625      78125
```

```
#I[4]::  %^5

#O[4]:*  1 + x

#I[5]::  Ps[(1-x)^n,x,0,3]

                                                  2                    3
#O[5]:*  1 + -n x + (n (-1/2 + n/2)) x  + -(n (-2/3 + n/3) (-1/2 + n/2)) x

#I[6]::  Ps[Sin[x]/x,x,a,1]

                                Sin[a]
                    Cos[a] - ------
         Sin[a]                  a
#O[6]:*  ------ + --------------- (x - a)
           a                a

#I[7]::  Ex[%]

         Sin[a]    Cos[a]   Sin[a]
#O[7]:*  ------ + (------ - ------) (x - a)
           a         a          2
                                a

#I[8]::  p:Ps[Exp[x],x,Inf,6]

#O[8]:*  Exp[x] 1

#I[9]::  p:p*Exp[1/x]

                                 -2    -3    -4    -5    -6
                        -1   x     x     x     x     x
#O[9]:*  Exp[x] (1 + x    + --- + --- + --- + --- + ---)
                           2     6    24   120   720

#I[10]::  s[0]:0;s[$i]:1/$i!^2;s

                                 1
#O[10]:  {[0]: 0, [$i]: ------}
                               2
                          ($i!)

#I[11]::  Ps[x^(n+2)*(1-x),x,,5,s]

                          4      5      6     7
           n   3    3 x     2 x     5 x     x
#O[11]:*  x  (x  - ---- - ---- - ---- - ---)
                    4      9     192   588

#I[12]::  Ps[f[$x],$x,0,$n]:Ps[1,$x,0,$n,s]

                                                  1
#O[12]:  Ps[1,$x,0,$n,{[0]: 0, [$i]: ------}]
                                               2
                                          ($i!)

#I[13]::  Ps[f[x],x,,5]

              2    3    4      5
             x    x    x      x
#O[13]:*  x + -- + -- + --- + -----
             4    36   576   14400

#I[14]::  f[%]

              2       3       4         5
             x    13 x    31 x    1187 x
#O[14]:*  x + -- + ----- + ----- + -------
             2     72      576     86400
```

```
#I[15]::  Ps[Exp[x+y],{x,y},0,3]

                   2      3                        2                     
                  y      y                        y                2            3
#O[15]:*  (1 + y + -- + --) + (1 + y + --) x + (1/2 + y/2) x  + (1/6) x
                   2      6                        2

#I[16]::  Ra[Exp[x],x,0,{3,2}]

                        2     3
                     3 x     x
          1 + 3x/5 + ---- + --
                      20    60
#O[16]:*  --------------------
                           2
                          x
               1 - 2x/5 + --
                          20

#I[17]::  Ps[%,x,0,5]

                   2    3    4    5
                  x    x    x    x
#O[17]:*  1 + x + -- + -- + -- + ---
                  2    6    24   120

#I[18]::  Ra[Exp[x],x,0,{3,2,$x=$y+1}]

                                 2                   2    3
                                x                 3 x    x
                    1 + 2x/3 + -- 1 + 3x/5 + ---- + --
           1 + x                 6                 20    60
#O[18]:*  {-----,-------------,-------------------}
             1        1 - x/3                     2
                                                 x
                                      1 - 2x/5 + --
                                                 20

#I[19]::  Ra[@9,x,Inf,3]

                          1      1
          Exp[x] (1 + -- + -----)
                          2x       2
                               12 x
#O[19]:*  ------------------------
                      1      1
                (1 - -- + -----)
                      2x      2
                          12 x

#I[20]::  Cf[Exp[x],x,0,3]

                              x
                              -
            1     -x          2    -x
#O[20]:   ---- ---- ---- --
          1 +  1 +  1 +  6
```

Lim[*expr*, *var*, *pt*]

forms the limit of *expr* as *var* tends to *pt*. A sequence of Ps projections of increasing order are formed until a definite limit is found.

```
#I[1]::   t:Sin[x]/x

          Sin[x]
#O[1]:    ------
            x
```

```
#I[2]::  Lim[t,x,0]

#O[2]:   1

#I[3]::  S[t,x->0]

          0
#O[3]:    -
          0

#I[4]::  Lim[(x+1)/Sin[x^2],x,0]

           -2    -1
#O[4]:*  x    + x
```

9.6 Matrix and explicit tensor manipulation

Outer [*temp*, *list1*, *list2*, . . .]
forms the generalized outer "product" of the lists *list1*, *list2*, .. with respect to the template *temp*. If entries in the lists t and u are specified as t[i1, i2, .., ik] and u[j1, j2, .., jk] then Outer[f, t, u] is a list whose entries are given by Ap[f, {t[i1, i2, .., ik], u[j1, j2, .., jk]}]

```
#I[1]:: Outer[f,{a,b},{c,d}]

#O[1]:  {{f[a,c],f[a,d]},{f[b,c],f[b,d]}}

#I[2]:: Outer[f,{{1,2},{3,4}},{[x]:a,[y]:b,[x]:c}]

#O[2]:  {{{[x]: f[1,a], [y]: f[1,b], [x]: f[1,c]},
                        {[x]: f[2,a], [y]: f[2,b], [x]: f[2,c]}},
                {{[x]: f[3,a], [y]: f[3,b], [x]: f[3,c]},
                        {[x]: f[4,a], [y]: f[4,b], [x]: f[4,c]}}}

#I[3]:: t:Ar[3]

#O[3]:  {1,2,3}

#I[4]:: Outer[Plus,t,t,t]

#O[4]:  {{{3,4,5},{4,5,6},{5,6,7}},{{4,5,6},{5,6,7},{6,7,8}},
                {{5,6,7},{6,7,8},{7,8,9}}}

#I[5]:: Dim[%]

#O[5]:  {3,3,3}

#I[6]:: Outer[Ge,t,t]

#O[6]:  {{1,0,0},{1,1,0},{1,1,1}}
```

Inner [(*temp1*:Mult), *list1*, *list2*, (*temp2*:Plus)]
forms the generalized inner "product" of *list1* and *list2* with respect to the templates *temp1*,*temp2*.
• Dot [8.2]

```
#I[1]:: Inner[f,{a,b},{c,d},g]

#O[1]:  g[f[a,c],f[b,d]]

#I[2]:: Inner[f,{{a,b},{c,d}},{{x,y},{z,t}},g]

#O[2]:  {{g[f[a,x],f[b,z]],g[f[a,y],f[b,t]]},
                {g[f[c,x],f[d,z]],g[f[c,y],f[d,t]]}}

#I[3]:: Inner[f,{a,{c,d}},{{x,y},{z,t}}]

#O[3]:  {f[a,x] + f[{c,d},z],f[a,y] + f[{c,d},t]}
```

Trans [*list*, (*levs*:2),]
yields a list obtained from *list* by transposing entries between two levels specified by *levs* (the *ni* are positive integers):

n	1 and *n*
{*n1*,*n2*}	*n1* and *n2*

```
#I[1]:: m:{{a,b},{c,d}}
#O[1]:  {{a,b},{c,d}}
#I[2]:: Trans[m]
#O[2]:  {{a,c},{b,d}}
#I[3]:: t:Ar[{3,2,2},f]
#O[3]:  {{{f[1,1,1],f[1,1,2]},{f[1,2,1],f[1,2,2]}},
                {{f[2,1,1],f[2,1,2]},{f[2,2,1],f[2,2,2]}},
                {{f[3,1,1],f[3,1,2]},{f[3,2,1],f[3,2,2]}}}
#I[4]:: Trans[t]
#O[4]:  {{{f[1,1,1],f[1,1,2]},{f[2,1,1],f[2,1,2]},{f[3,1,1],f[3,1,2]}},
                {{f[1,2,1],f[1,2,2]},{f[2,2,1],f[2,2,2]},{f[3,2,1],f[3,2,2]}}}
#I[5]:: Trans[t,3]
#O[5]:  {{{f[1,1,1],f[2,1,1],f[3,1,1]},{f[1,2,1],f[2,2,1],f[3,2,1]}},
                {{f[1,1,2],f[2,1,2],f[3,1,2]},{f[1,2,2],f[2,2,2],f[3,2,2]}}}
#I[6]:: Trans[t,{2,3}]
#O[6]:  {{{f[1,1,1],f[1,2,1]},{f[1,1,2],f[1,2,2]}},
                {{f[2,1,1],f[2,2,1]},{f[2,1,2],f[2,2,2]}},
                {{f[3,1,1],f[3,2,1]},{f[3,1,2],f[3,2,2]}}}
```

Tr [*list*, (*temp*:Plus)]
the generalized trace obtained by applying *temp* to the set of entries in *list* whose indices are all equal.

```
#I[1]:: Tr[{{a,b},{c,d}}]
#O[1]:  a + d
#I[2]:: Tr[{{a,b},{c,d}},f]
#O[2]:  f[a,d]
#I[3]:: Ar[{2,2,2},q]
#O[3]:  {{{q[1,1,1],q[1,1,2]},{q[1,2,1],q[1,2,2]}},
                {{q[2,1,1],q[2,1,2]},{q[2,2,1],q[2,2,2]}}}
#I[4]:: Tr[%]
#O[4]:  q[1,1,1] + q[2,2,2]
```

Sig[*list*, *base*]
the signature of the permutation between *base* and *list* (0 if no permutation suffices). (If *base* is omitted, entries of *list* may appear directly as filters for *Sig*).
• **Asym** [7.7]

```
#I[1]:  {Sig[2,1,3], Sig[{2,1,3}], Sig[1,1,3], Sig[1,4,3,2,5]}

#O[1]:  {-1,-1,0,-1}
```

Det[*list*]
the determinant of a "full" *list*.

```
#I[1]::  Det[{{a,b},{c,d}}]
#O[1]:   a d - b c
```

Minv[*list*]
the inverse of a non-singular matrix represented by *list*.

```
#I[1]::  m:{{1,3,5},{7,-5,2},{8,11,1}}
#O[1]:   {{1,3,5},{7,-5,2},{8,11,1}}
#I[2]::  n:Minv[m]
#O[2]:   {{-3/65,4/45,31/585},{1/65,-1/15,11/195},{1/5,1/45,-2/45}}
#I[3]::  m.n
#O[3]:   {{1,0,0},{0,1,0},{0,0,1}}
#I[4]::  Minv[{{1,1},{2,2}}]
#O[4]:   Minv[{{1,1},{2,2}}]
```

Mdiv[*list1*, *list2*]
yields a matrix *mat* such that *list2*. *mat* is equal to *list1*.

```
#I[1]::  m:{{1,3,5},{7,-5,2},{8,11,1}}
#O[1]:   {{1,3,5},{7,-5,2},{8,11,1}}
#I[2]::  v:{1,3,1}
#O[2]:   {1,3,1}
#I[3]::  x:Mdiv[v,m]
#O[3]:   {{32/117},{-5/39},{2/9}}
#I[4]::  m.x
#O[4]:   {{1},{3},{1}}
#I[5]::  x.m
#O[5]:   {{1,3,5},{7,-5,2},{8,11,1}}
```

Triang[*list*]
gives the triangularized form of a matrix represented by *list*.

```
#I[1]::  m:{{1,3,5},{7,-5,2},{8,11,1}}
#O[1]:   {{1,3,5},{7,-5,2},{8,11,1}}
#I[2]::  Triang[m]
#O[2]:   {{8,11,1},{0,-117/8,9/8},{0,0,5}}
```

Eig[*list*]
yields a list of eigenvalues and normalized eigenvectors for the matrix *list*.

Simtran[*list*]
the similarity transformation matrix necessary to diagonalize *list*.

```
#I[1]:: m:{{2,2,-1},{1,3,-1},{1,4,-2}}

#O[1]:  {{2,2,-1},{1,3,-1},{1,4,-2}}

#I[2]:: Eig[m]

                                1     -1          1     -1          2     3     -1
#O[2]:  {{-1,{0,-------,-------}},{1,{-------,-------,0}},{3,{-------,-------,-------}}}
                           1/2   1/2          1/2   1/2          1/2   1/2   1/2
                          2     2            2     2            14    14    14

#I[3]:: Simtran[m]

                      1     -1          1     -1          2     3     -1
#O[3]:  {{0,-------,-------},{-------,-------,0},{-------,-------,-------}}
                    1/2   1/2          1/2   1/2          1/2   1/2   1/2
                   2     2            2     2            14    14    14
```

10. Non-computational operations

10.1 Input and output operations

Lpr[*expr*, (*file*:Null)]
prints *expr* to the specified file [10.3] (terminal as default) in a direct linear format suitable to be used as input, and in which labelling of parts is manifest [2.10]. It yields Null as an image.

```
#I[1]:  r:x

#O[1]:  x

#I[2]:  Rpt[r:r^r,4]

                                                    1 + x
                                 1 + x     1 + x + x
                    1 + x + x          + x
                   x
#O[2]:  x

#I[3]:  Lpr[r]

x^(x^(1 + x + x^(1 + x) + x^(1 + x + x^(1 + x))))

#I[4]:  4+5I

#O[4]:  4 + 5I

#I[5]:  Lpr[%]

Cx[4,5]

#I[6]:  Lpr[r,"r.out"]

#I[7]:  Lpr[r^r,"r.out"]
```

The file `r.out` then contains:

```
x^(x^(1 + x + x^(1 + x) + x^(1 + x + x^(1 + x))))

x^(x^(1 + x + x^(1 + x) + x^(1 + x + x^(1 + x)) + x^(1 + x + x^(1 + x)\
                    + x^(1 + x + x^(1 + x)))))
```

Pr[*expr1*, (*expr2*, . . .)]
prints *expr1*, *expr2*, .. in turn (separated by tabs) with standard two-dimensional format [2.12], and yields the last *expri* as an image.

```
#I[1]:  Pr[a^a,b^b];Pr[c,d,e];Pr[];Pr[1,2,3,4]

 a       b
a       b
c       d       e

1       2       3       4

#O[1]:  4
```

Prh[*expr1*, (*expr2*, . . .)]
prints the unsimplified forms of the *expri*.

```
#I[1]:: Prh[1+1,a-1/b]

1 + 1   a - 1
            -
            b

#O[1]:  a - 1
            -
            b

#I[2]:: Pr[1+1]

2

#O[2]:  2
```

Rd[(*prompt*:Null),(*file*:Null)]
prints the expression *prompt*, then reads and simplifies one line of input (terminated by newline) from the specified file [10.3]. Default is input from standard input/output medium (usually terminal). A null line is read as Null.

```
#I[1]:: {a,Rd[]^2,Rd["3rd> "]}

b+c
3rd> 5

                  2
#O[1]:  {a,(b + c) ,5}
```

Rdh[(*prompt*:Null),(*file*:Null)]
prints the expression *prompt*, then reads one line of input from *file* and yields its "held" [3.5] form.

```
#I[4]:: {Rdh[],Rd[]}

1+1
1+3

#O[4]:  {' 1 + 1,4}
```

Fmt[(*prspec*:Null),*expr1*,*expr2*,...]
yields a print form with the *expri* in a format specified by *prspec*:

Null	*expr1* followed immediately by *expr2*...
positive integer	*expr1* followed by *expr2*... after *prspec* blank spaces.
list	*expri* appear with horizontal and vertical offsets defined by *prspec*[*i*], according to {*hori*, *veri*}.expri with equal horizontal or vertical offsets are aligned. Those with larger horizontal offsets are further to the right, and those with larger vertical offsets are higher up. If no entry exists in *prspec* for a particular *expri*, it appears immediately to the right of the last printed expression. A horizontal or vertical offset Inf specifies a position to the right or above all other expressions.

```
#I[1]:: Fmt[1,a,b+c,x^2]

                2
#O[1]:  ab + cx
```

```
#I[2]::  Fmt[3,a,b+c,x^2]

                      2
#O[2]:   a   b + c   x

#I[3]::  Fmt[,"|",x^2,"|"]^p+1

            2 p
#O[3]:   1 + |x |

#I[4]::  Fmt[{3,1,2},a,b+c,x^2]

              2
#O[4]:   b + cx  a

#I[5]::  Fmt[,a,,1+3,"---","x^2"]

#O[5]:   a4---x^2

#I[6]::  Fmt[{{0,1},{0,-1}},a,b]

         a
#O[6]:
         b

#I[7]::  Fmt[{{0,1},{0,-1}},a^2+b^2,x/y]

          2    2
         a  + b
#O[7]:
         x
         -
         y

#I[8]::  Fmt[{{0,0},{-1,1},{1,-2}},a+b,c,d]

         c
#O[8]:    a + b

               d

#I[9]::  Fmt[{{0,0},{1,1}},a,b,c,d]

          bcd
#O[9]:   a

#I[10]::  Fmt[{0,{1,1}},a,b,c,d]

           bcd
#O[10]:   a

#I[11]::  Fmt[{[$x_=Evenp[$x]]:{$x,1},[$x]:$x},a,b,c,d,e,f,g]

          b d f
#O[11]:   a c e g

#I[12]::  Fmt[{[1]:0,[3]:Inf,[$x]:{$x,1}},a,b,c,d,e,f,g]

            b defg
#O[12]:   a       c

#I[13]::  Fmt[{0,1,1},a,x^2+y^2,c]

           2    2
#O[13]:   ac  + y

#I[14]::  Lpr[%]

Fmt[{0,1,1},a,x^2 + y^2,c]
```

```
#I[15]:: ex:Fmt[{{0,3},{0,2},{0,1},{0,-1}},"|","|","|","#"]

#0[15]:
          |
          |
          |
          #

#I[16]:: Fmt[,x/y,ex]

#0[16]:
          |
          |
          x|
          -
          y#

#I[17]:: Fmt[{0,2},x/y,ex]

#0[17]:
          x |
          - |
          y
            #
```

Sx[(*cc*:), {*x1, x2, ...*}, (*class*:1), (*prec*:1)]
yields a print form with the *xi* appearing in association with ⊂cc⊃ with a syntax and precedence defined by *class* and *prec* as specified in [2.11].

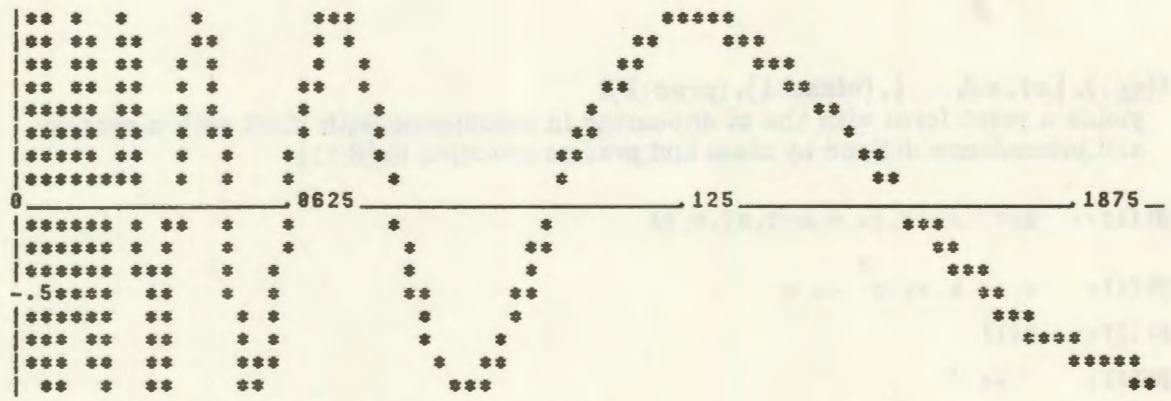
```
#I[1]:: Sx[" ++ ",{a,b,c^2,d},3,2]

                    2
#0[1]:  a ++ b ++ c  ++ d

#I[2]:: %[1]

#0[2]:  " ++ "

#I[3]:: 3x^2+Sx["~~",{a,b},5,1]

          2
#0[3]:  3 x  + a~~b
```

Special output forms may be defined by assigning a suitable print form as the value of *_s*[Pr] [*4*].

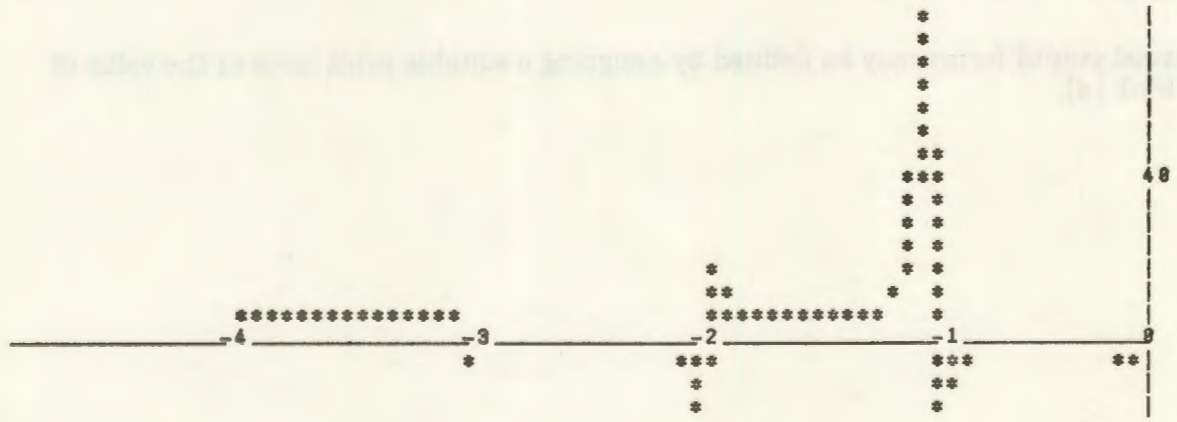
10.2 Graphical output

Graph [{(*x1*), *y1*, (*z1*)}, {*u*, (*v*)}, {*umin*, (*vmin*)}, {*umax*, (*vmax*)}, {*form1*}, {*xv*:0), (*yv*:0), (*zv*:Inf)}, {*upt*, (*vpt*)}, {{(*xmin*), *xmax*}), {(*ymin*), *ymax*}, ({(*zmin*), *zmax*})]
generates a Plot projection which prints as a plot of curves or surfaces defined by the numerical values of *xi*, *yi* and *zi* as functions of the parameters *u* and *v*, between *umin* and *umax* (with *upt* samples), and *vmin* and *vmax* (with *vpt* samples). *xv*, *yv*, *zv* specify the point of observation for three-dimensional plots (contour plots by default). The *formi* define the style of curves plotted: integer codes give standard curve styles; other *formi* are printed explicitly on the curves. Only points in the region bounded by *xmin*, *xmax*, *ymin*, *ymax*, *zmin*, *zmax* are plotted.

```
#I[1]:  Graph[Sin[1/x],x,0.02,0.2]

#O[1]:
```

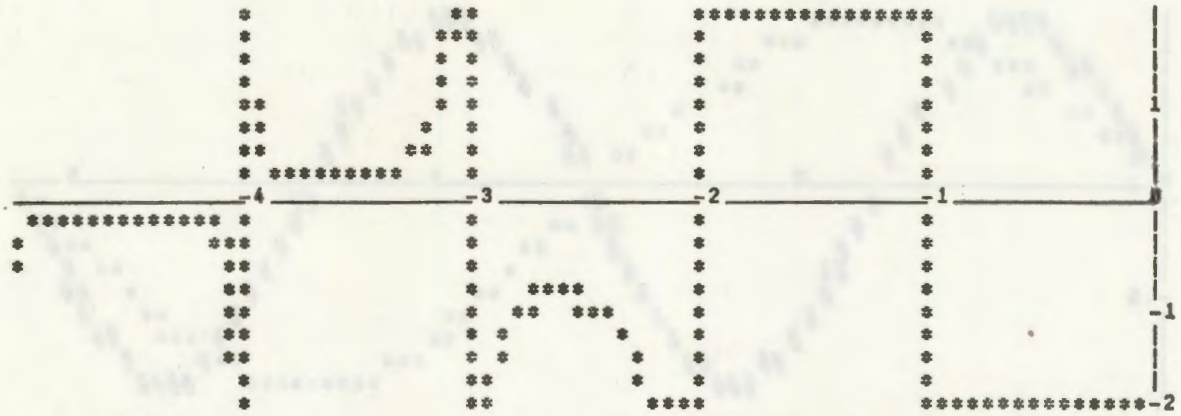



```
#I[2]:  Graph[Gamma[x],x,-5,0]

#O[2]:
```



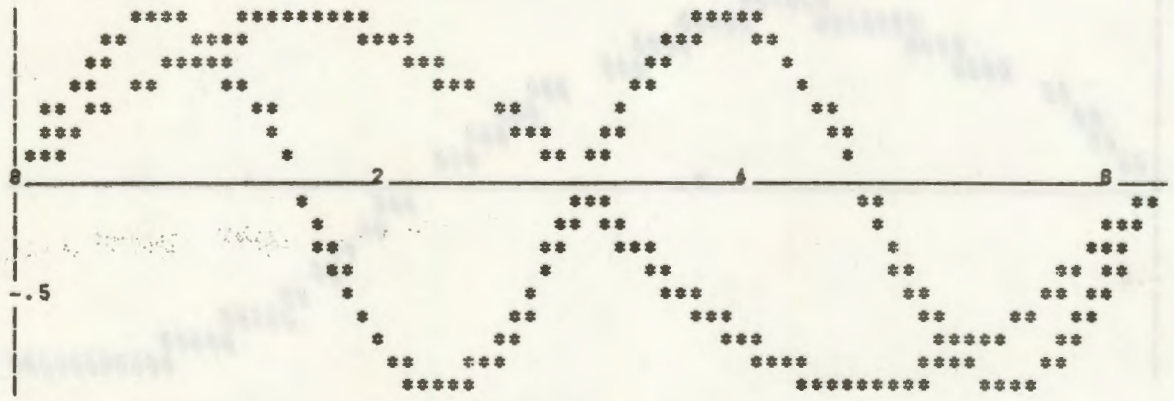
```
#I[3]:: Graph[Gamma[x],x,-5,0,,,,{-2,2}]
```

#O[3]:



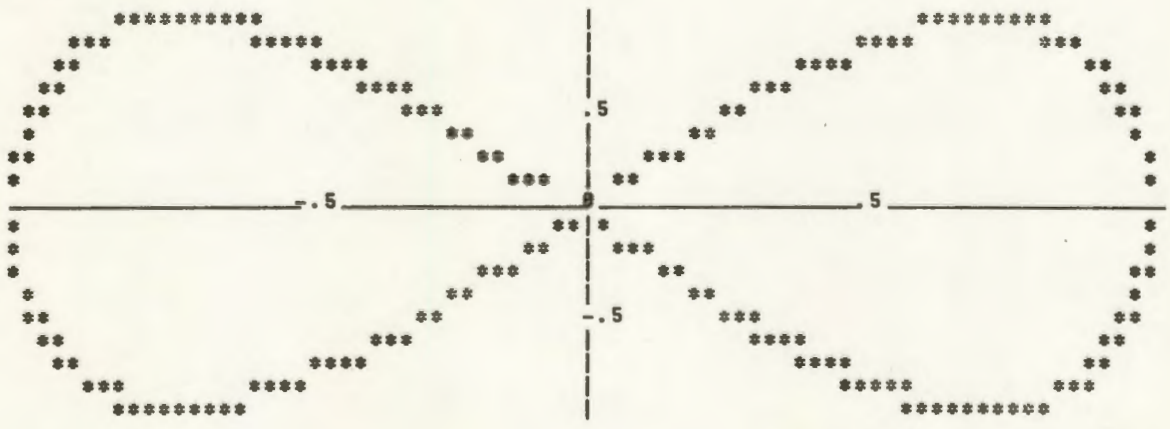

```
#I[4]:: Graph[{{Sin[t]},{Sin[2t]}},t,0,2Pi]
```

#O[4]:



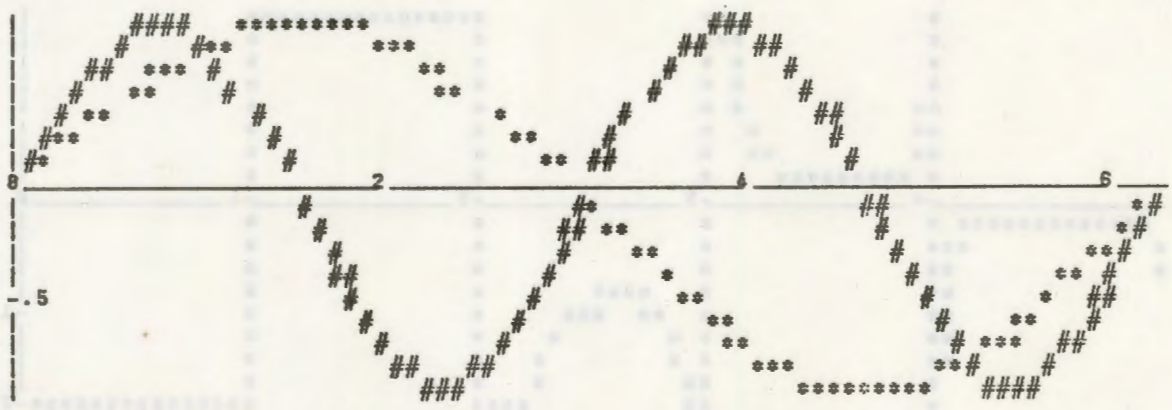

```
#I[5]:: Graph[{Sin[t],Sin[2t]},t,0,2Pi]
```

#O[5]:



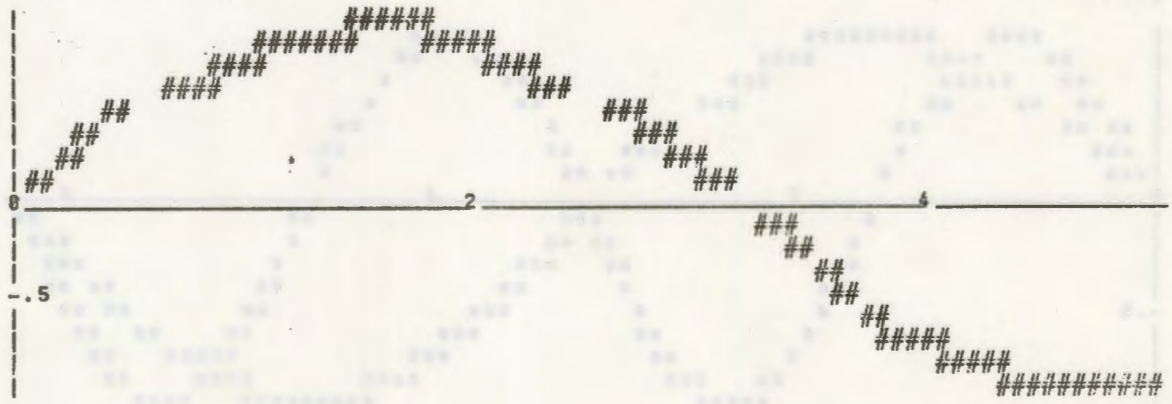
```
#I[6]:: Graph[{{Sin[t]},{Sin[2t]}},t,0,2Pi,{"*","#"}]
```

#O[6]:

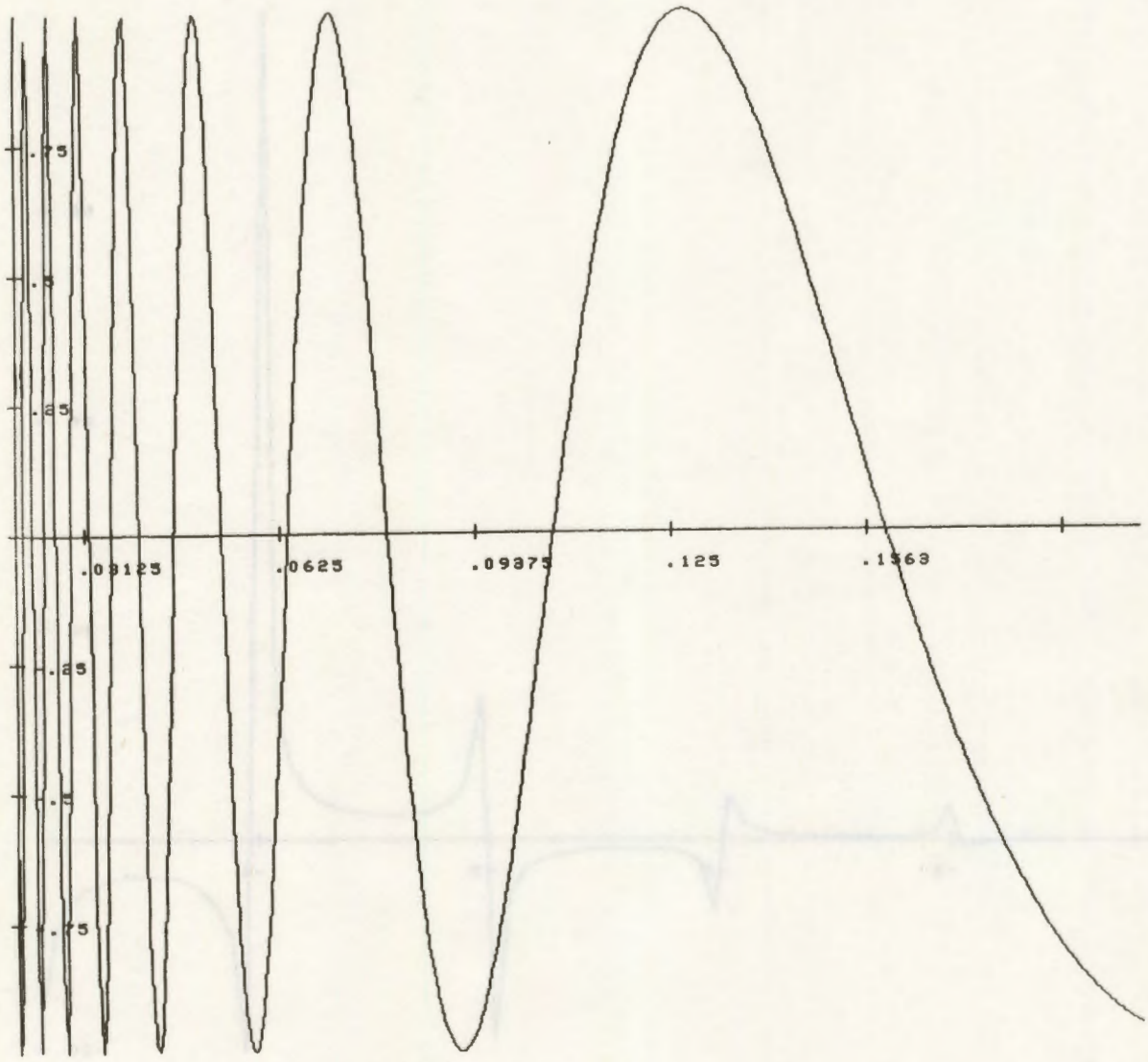



```
#I[7]:: Graph[Sin[x],x,0,5,"#",,18]
```

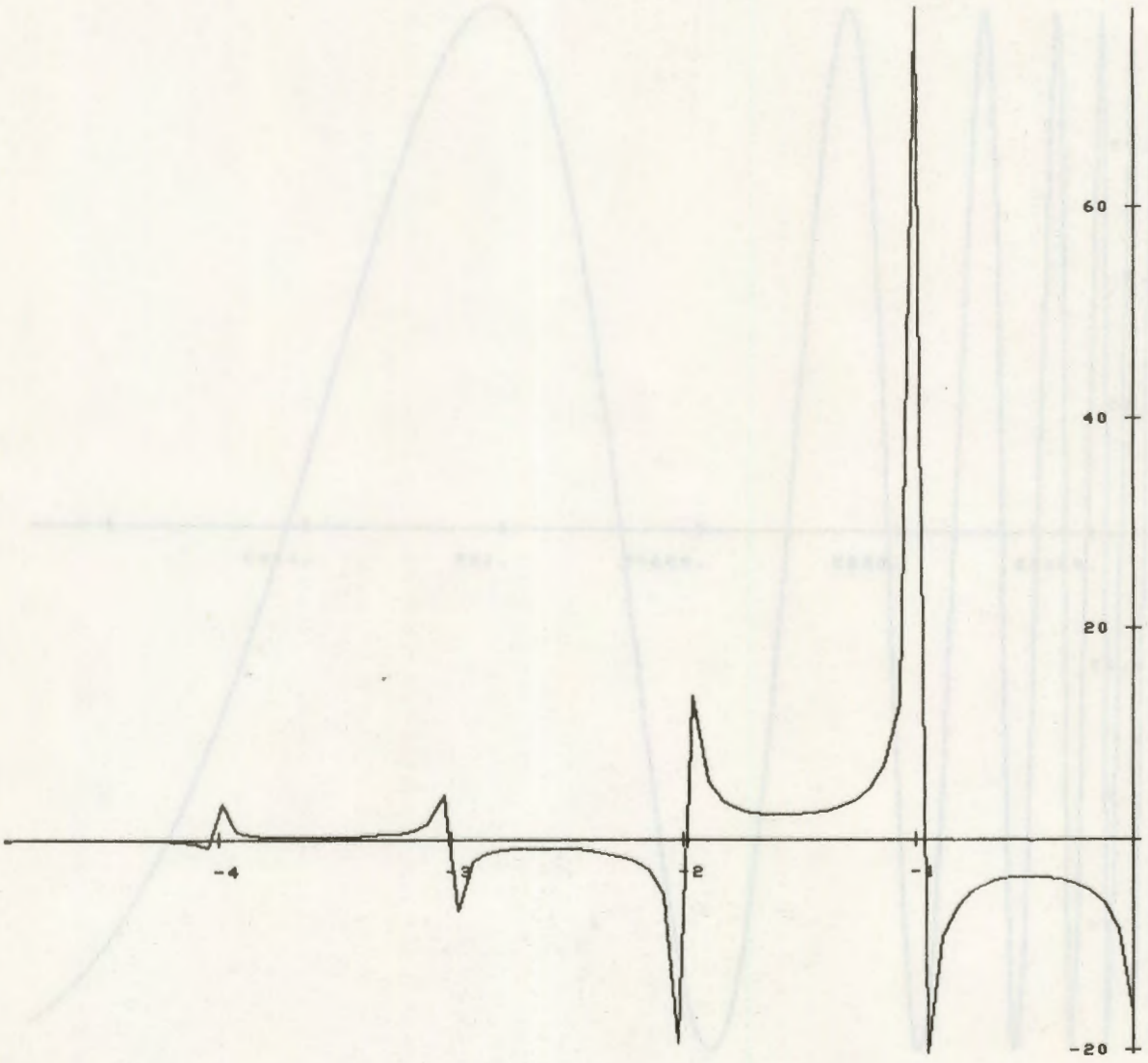
#O[7]:



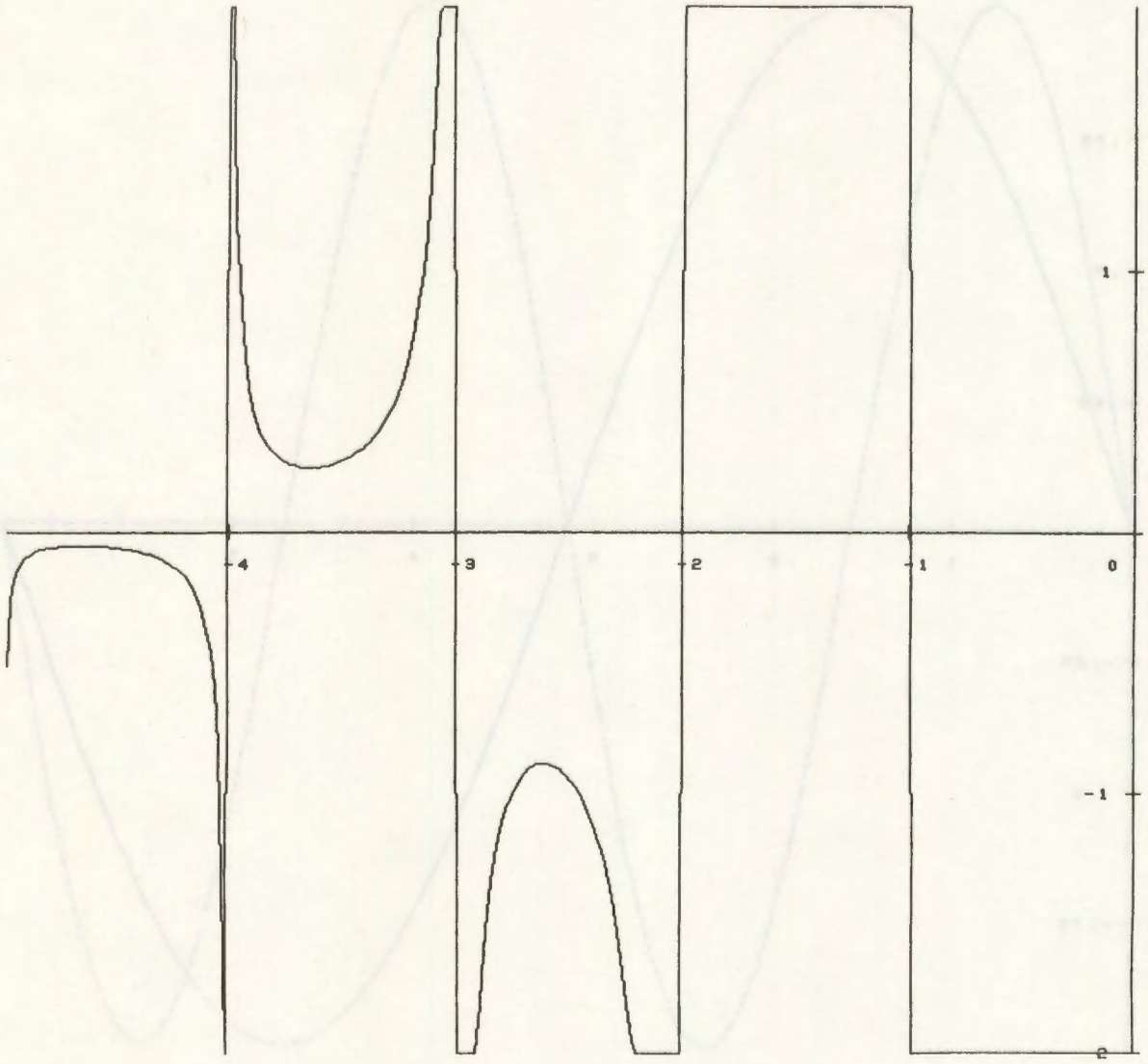
```
#I[1]:: Graph[Sin[1/x],x,0.02,0.2]
```



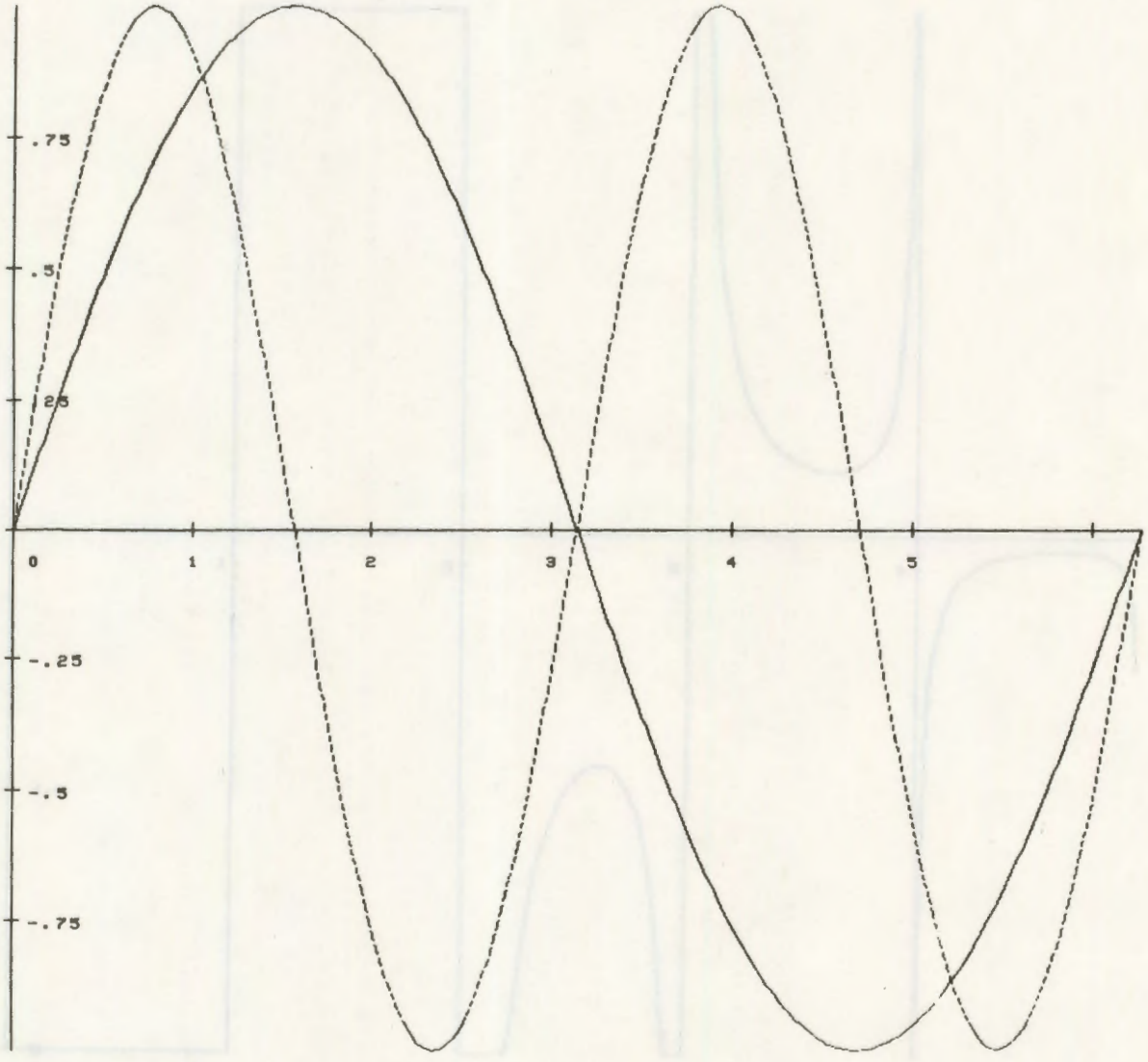
```
#I[2]:: Graph[Gamma[x],x,-5,0]
```



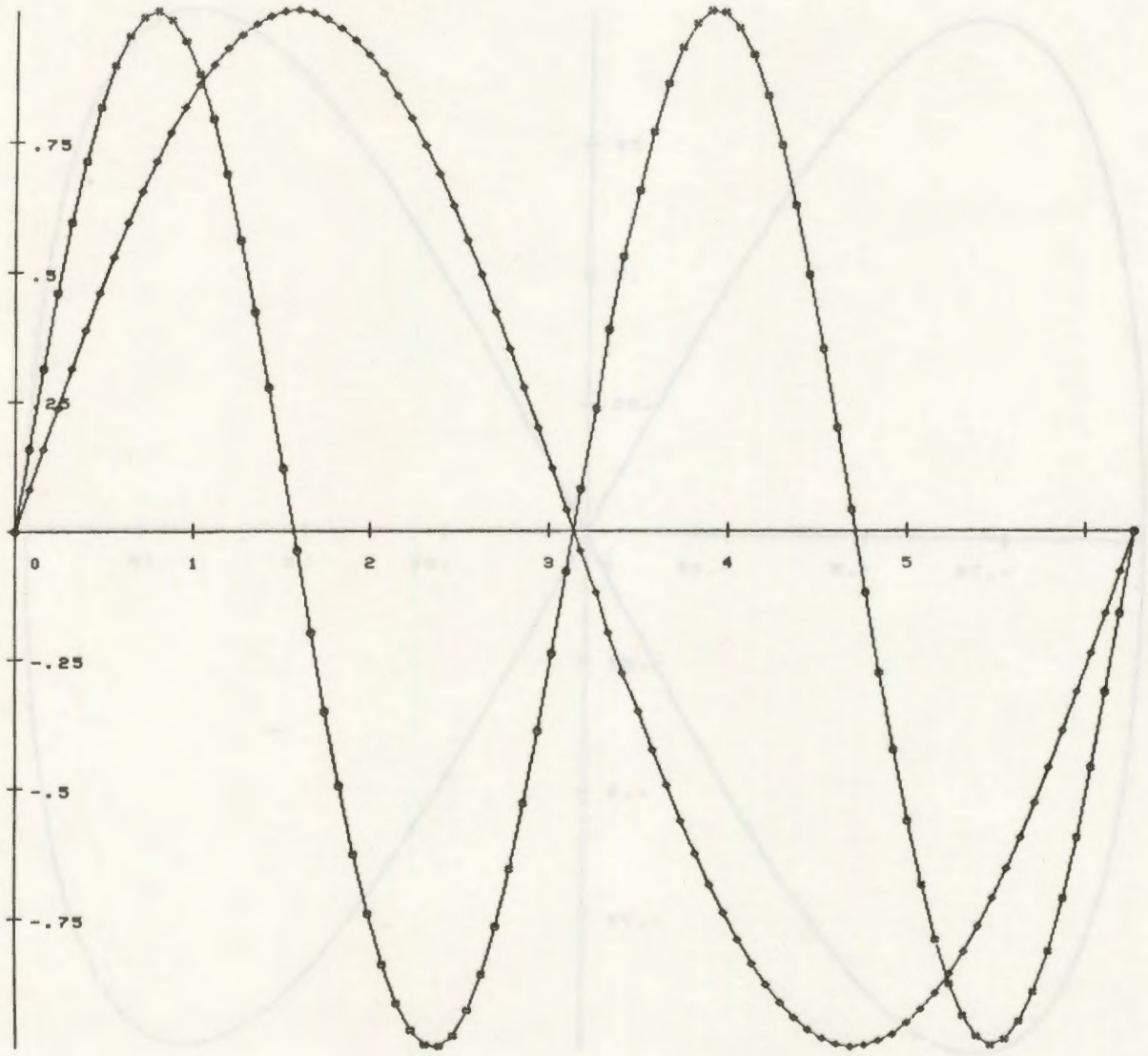
```
#I[3]::   Graph[Gamma[x],x,-5,0,,,,{-2,2}]
```



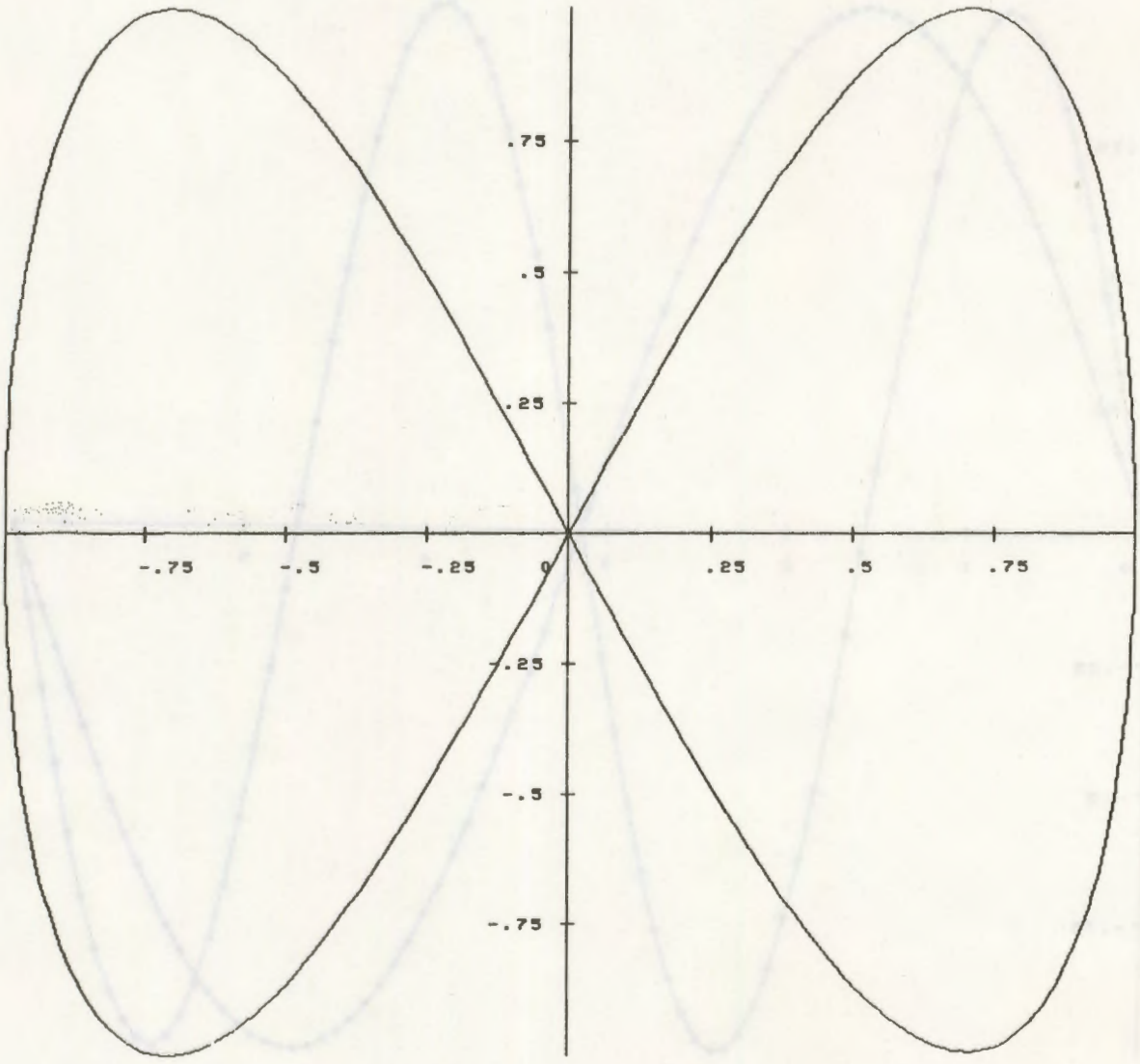
```
#I[4]::   Graph[{{Sin[t]},{Sin[2t]}},t,0,2Pi]
```



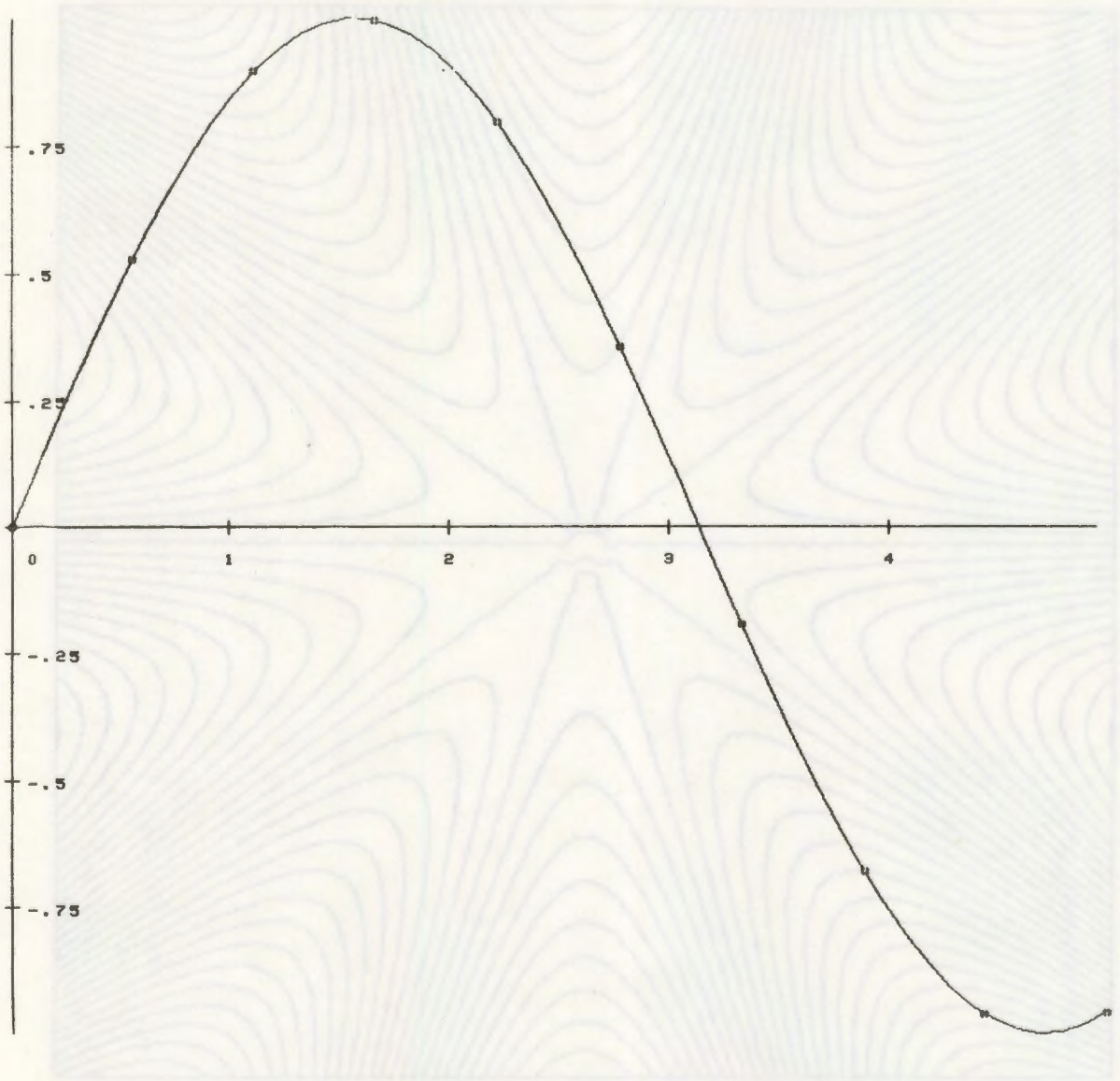
```
#I[6]::  Graph[{{Sin[t]},{Sin[2t]}},t,0,2Pi,{"*","#"}]
```



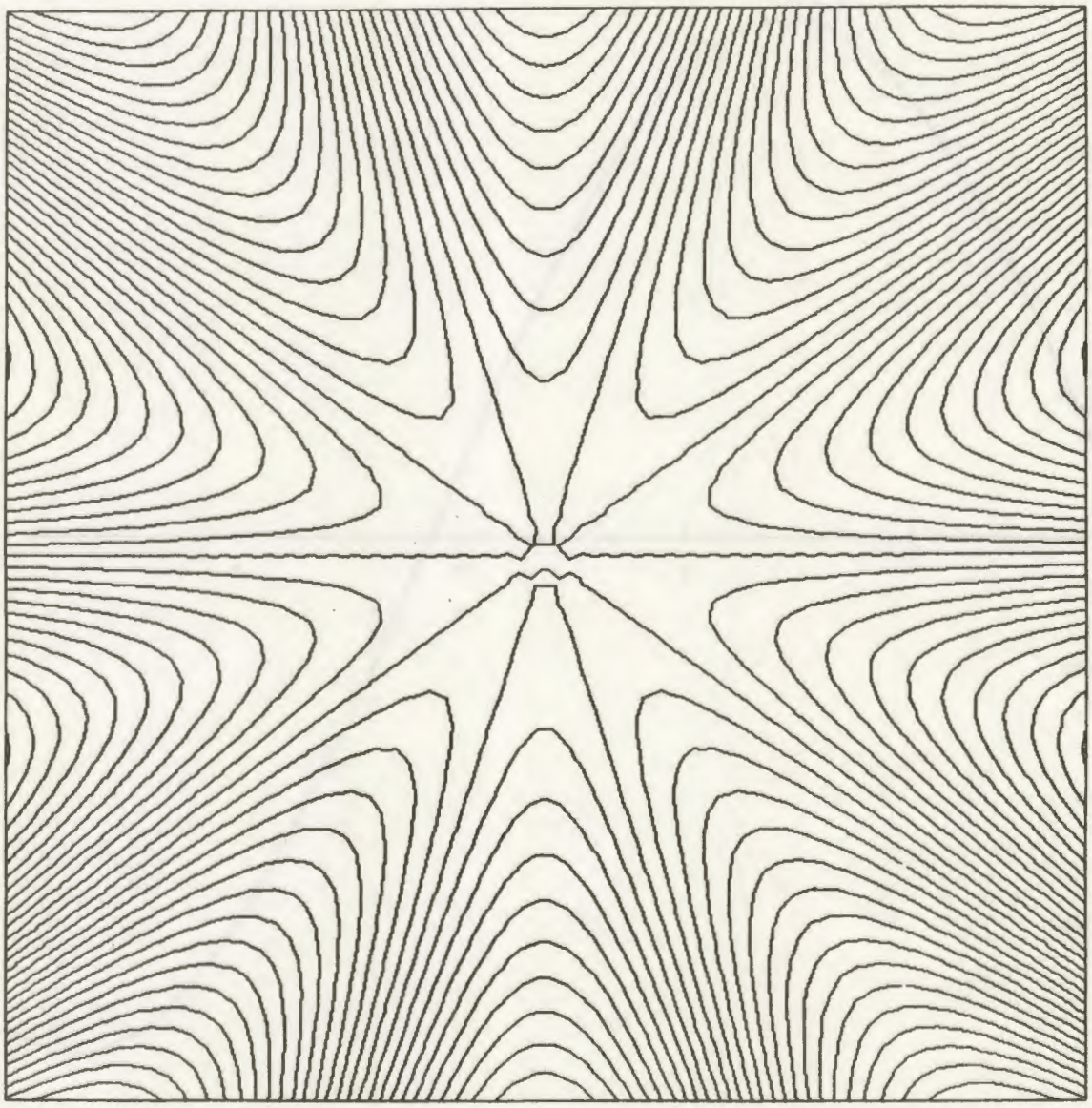
```
#I[5]:: Graph[{Sin[t],Sin[2t]},t,0,2Pi]
```



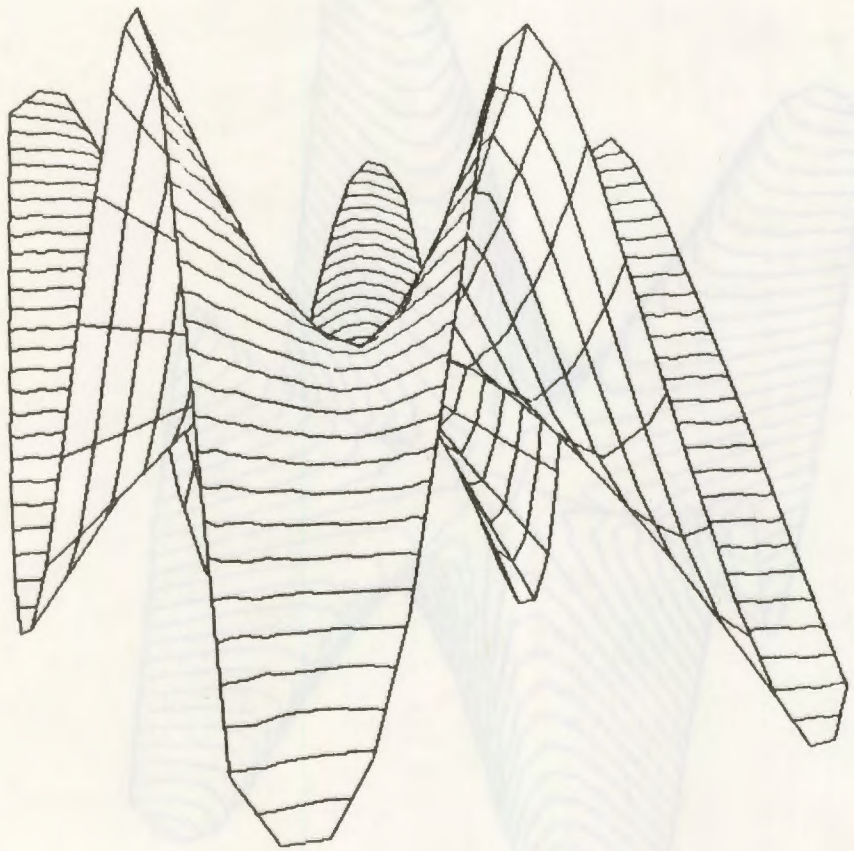
```
#I[7]::   Graph[Sin[x],x,0,5,"#",,10]
```



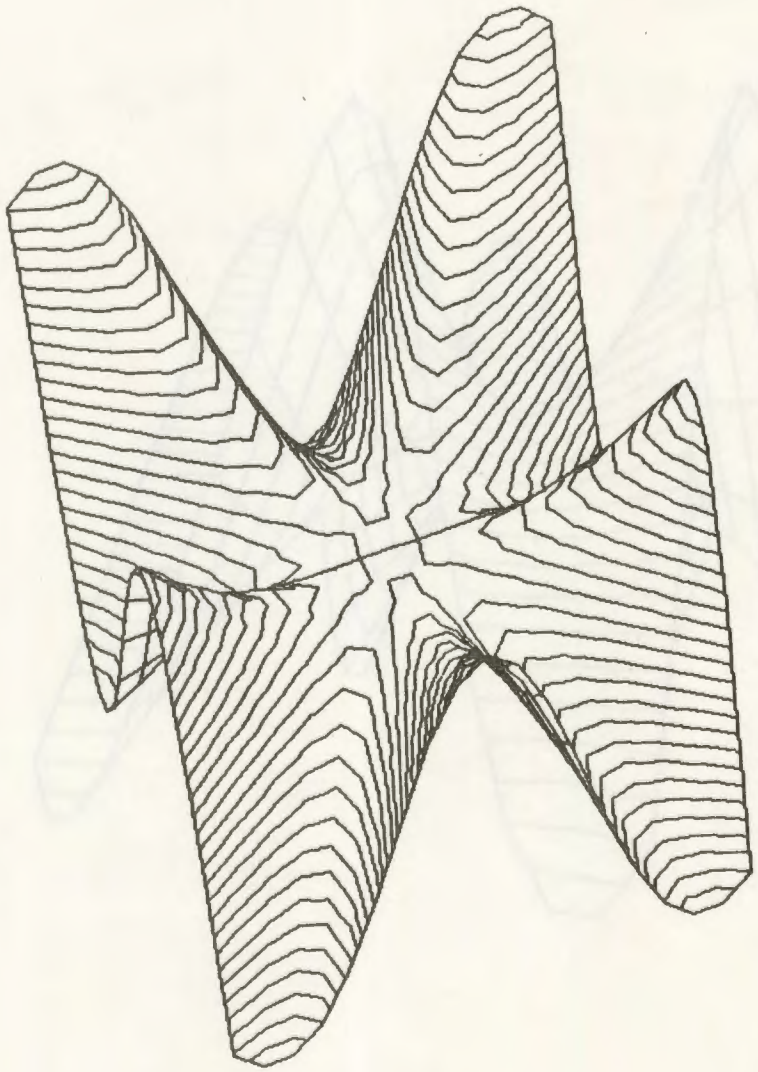
```
#I[8]::  Graph[(x^2+y^2)/100 Cos[5Atan[y/x]],{x,y},{-10,-10},{10,10}]
```



```
#I[9]::  Graph[(x^2+u^2)/100 Cos[5Atan[y/x]],{x,y},{-10,-10},{10,10} \
,,{20,30,-10}]
```



```
#I[18]:: Graph[(x^2+u^2)/100 Cos[5Atan[y/x]],{x,y},{-10,-10},{10,10} \
,,{50,150,300}
```



Plot[*plist***,{(***xv***:0),(***yv***:0),(***zv***:Inf)},**
{{(*xmin***),***xmax***},{(***ymin***),***ymax***},{(***zmin***),***zmax***}}]**

prints as a plot containing points, lines, curves, surfaces and regions specified in *plist*. Ranges of coordinates default to include all forms given in *plist*. *xmin*, *xmax*, ... define boundaries of the region in which points are plotted. In two-dimensional plots, forms given later in *plist* overwrite those given earlier when they overlap. In three (and higher) dimensions, explicit intersections and perspective are used.

High-resolution graphics output is generated if a suitable device is available. Plot[] clears the plotting area. (Implementation dependent)

The list *plist* (whose sublists are flattened) contains:

Pt[{*x*,*y*,(*z*)},(*form*)]
represents a point with coordinates *x*, *y* and *z*, to be printed as *form*. When *form* does not print as a single character, the coordinates are taken to specify its lower left-hand corner.

Line[*ptlist*,(*form*)]
represents a succession of straight lines between the point specified by Pt projections in the list *ptlist*: *form* specifies the style of line.

Curve[*ptlist*,(*form*)]
represents a smooth curve through the points specified by Pt projections in the list *ptlist*: *form* specifies the style of curve.

Zone[{*clist*},*form*]
represents the interior of a region bounded by curves or lines specified in *clist* (with end points identified) to be filled with a texture or colour *form*. Infinity is taken as exterior.

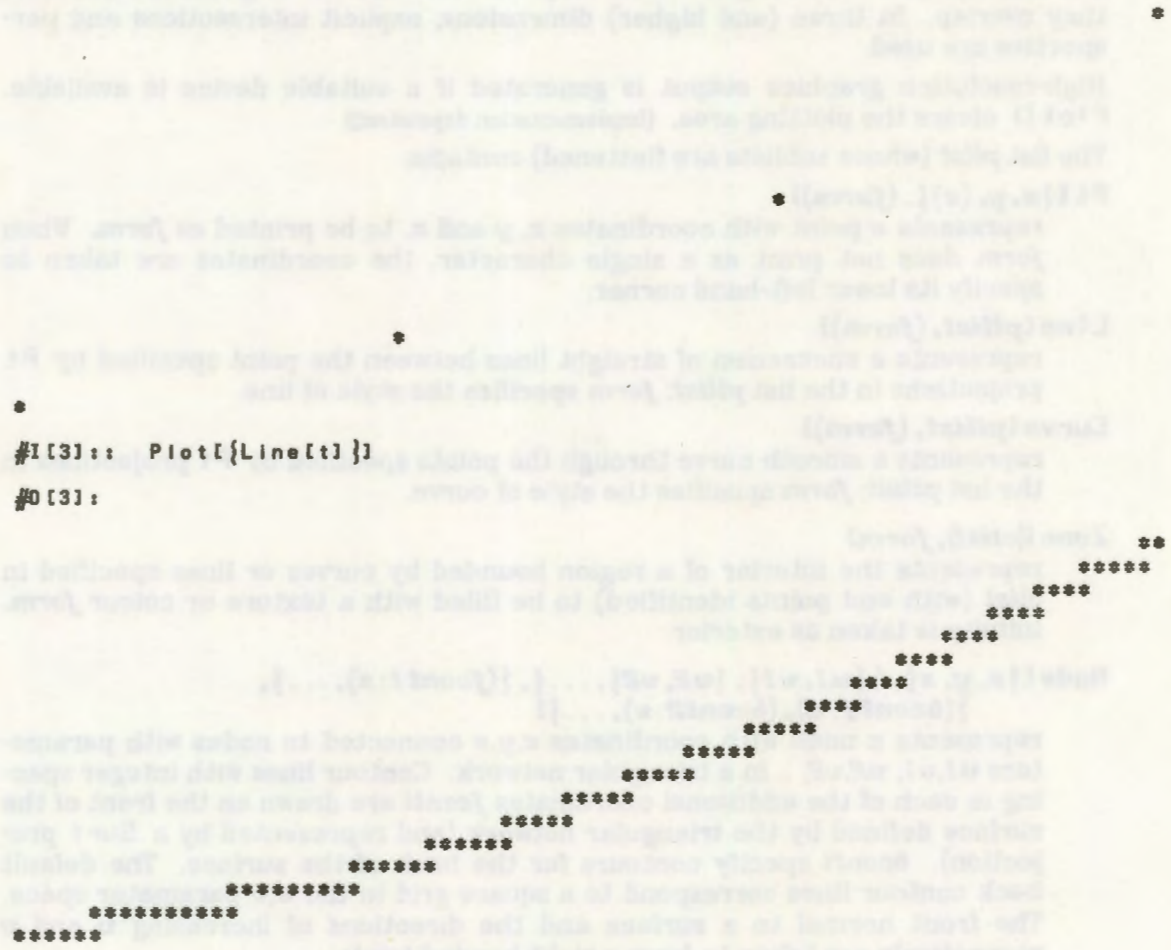
Node[{*x*,*y*,*z*},{{*u1*,*v1*},{*u2*,*v2*},...},{(*fcont1*:*z*),...},
{(*bcont1*:*u*),(*bcont2*:*v*),...}]
represents a node with coordinates *x*,*y*,*z* connected to nodes with parameters *u1*,*v1*, *u2*,*v2*, .. in a triangular network. Contour lines with integer spacing in each of the additional coordinates *fconti* are drawn on the front of the surface defined by the triangular network (and represented by a Surf projection). *bconti* specify contours for the back of the surface. The default back contour lines correspond to a square grid in the *u*,*v* parameter space. The front normal to a surface and the directions of increasing *u* and *v* respectively are taken to form a right-handed triple.

Surf[{[*u1*,*v1*]:*node1*,[*u2*,*v2*]:*node2*,... },(*form*)]
represents a surface spanned by a triangular network defined by Node projections *nodei* with parameter values *ui*,*vi*, and with a texture or colour specified by *form*.

Hull[{*pt1*,*pt2*...},*form*]
represents the surface formed by the convex hull of the points *pt1*, *pt2*,... with a texture or colour specified by *form*.

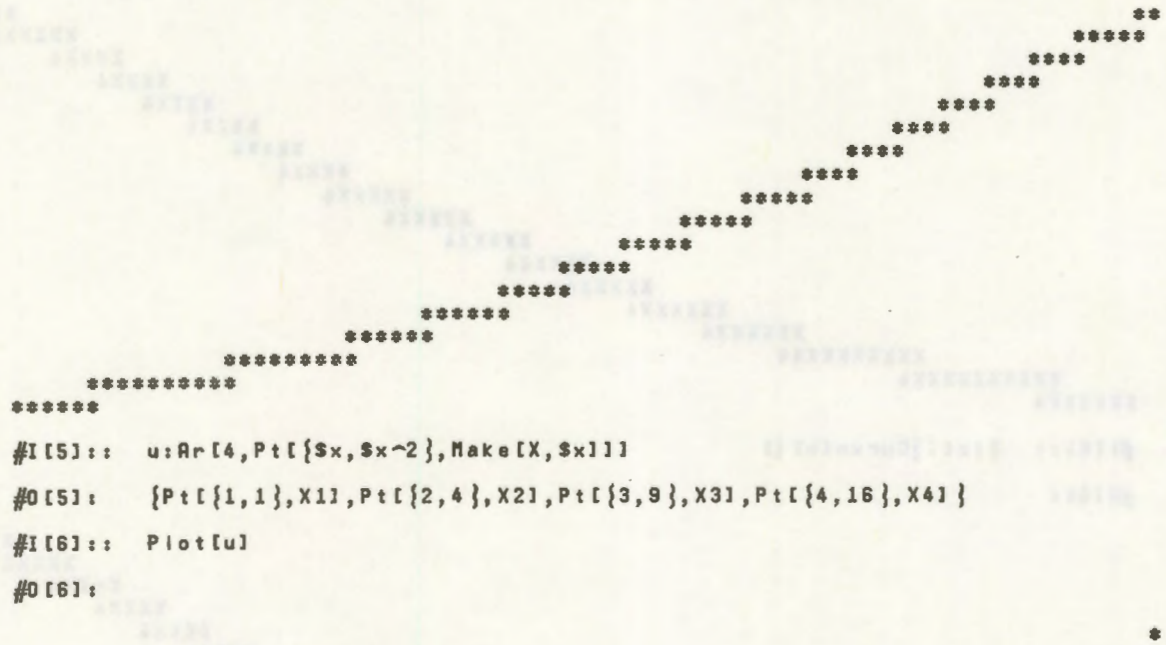
Axes[{(*x*:Inf),(*y*:Inf),(*z*:Inf)},{(*xtemp*),(*ytemp*),(*ztemp*)}]
represents a labelled set of orthogonal axes intersecting at Pt[*x*,*y*,*z*]. *xtemp* is a template applied to *xmin* and *xmax* to obtain a list of *x* values at which the x axis is to be labelled. *ytemp* and *ztemp* are analogous templates for the y and z axes. If *xtemp*, *ytemp*, *ztemp* are omitted, a heuristic procedure is used.

```
#I[1]::  t:Ar[4,Pt[{$x,$x^2}]]
#O[1]:   {Pt[{1,1}],Pt[{2,4}],Pt[{3,9}],Pt[{4,16}]}
#I[2]::  Plot[t]
#O[2]:
```

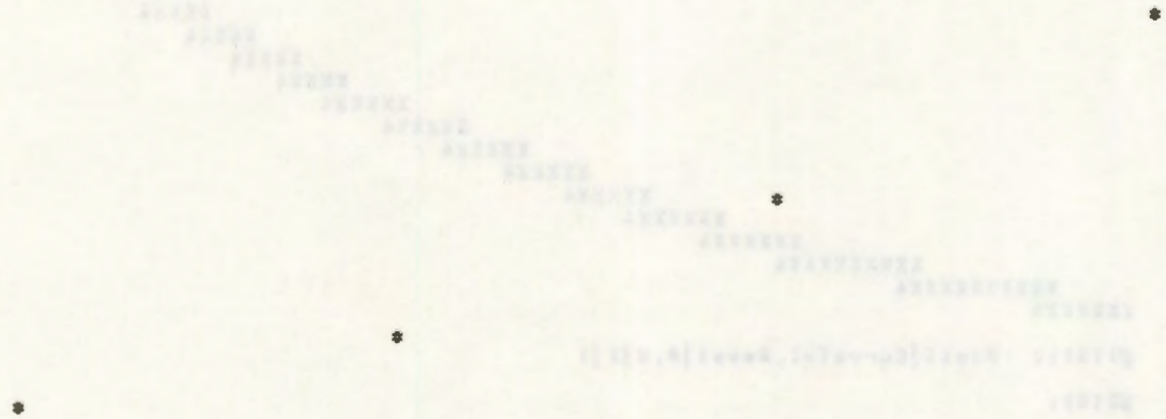


```
#I[3]::  Plot[{Line[t]}]
#O[3]:
```

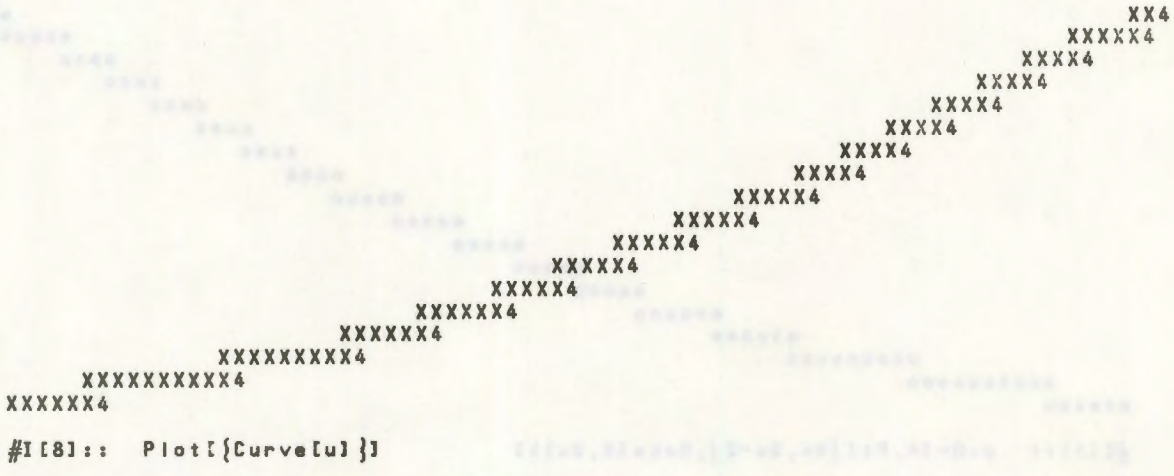
```
#I[4]::  Plot[{Curve[t]}]
#O[4]:
```



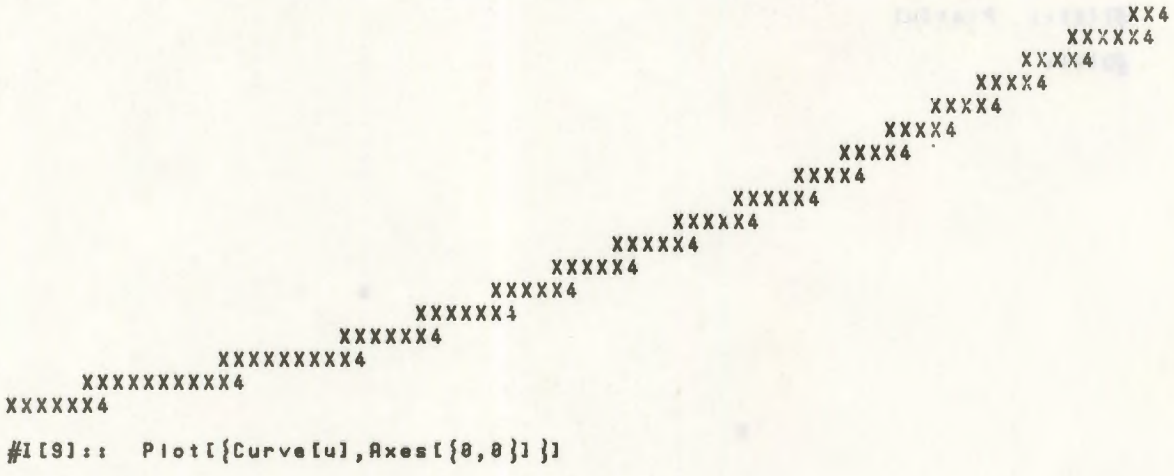
```
#I[5]::  u:Ar[4,Pt[{$x,$x^2},Make[X,$x]]]
#O[5]:   {Pt[{1,1},X1],Pt[{2,4},X2],Pt[{3,9},X3],Pt[{4,16},X4]}
#I[6]::  Plot[u]
#O[6]:
```



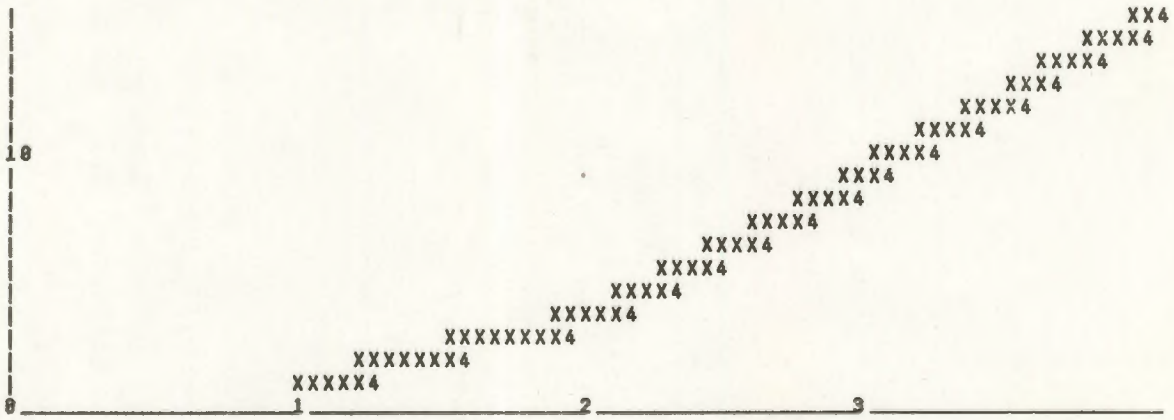
```
#I[7]::  Plot[{Line[u]}]

#O[7]:
```



```
#I[8]::  Plot[{Curve[u]}]

#O[8]:
```

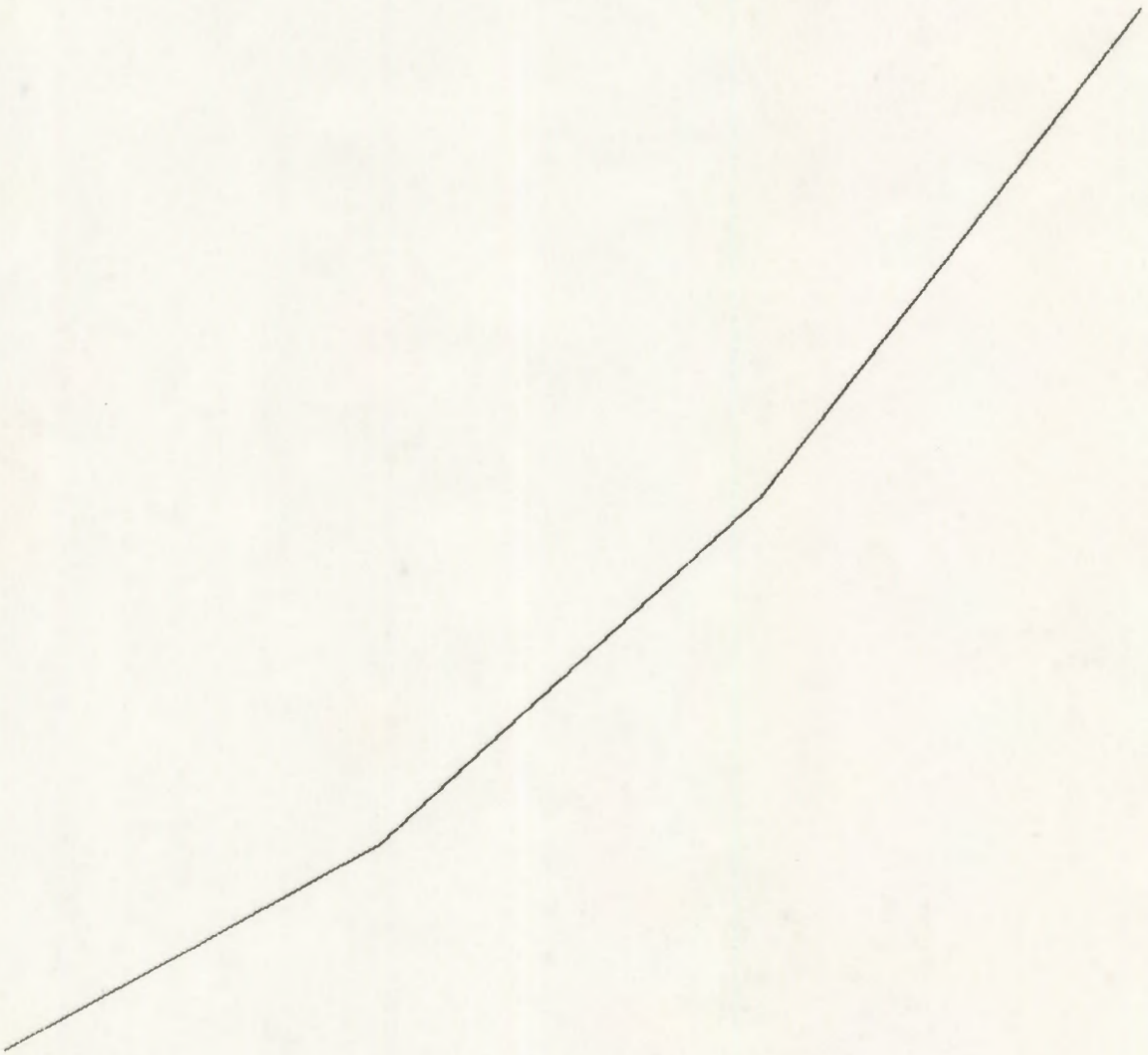


```
#I[9]::  Plot[{Curve[u],Axes[{0,0}]}]

#O[9]:
```

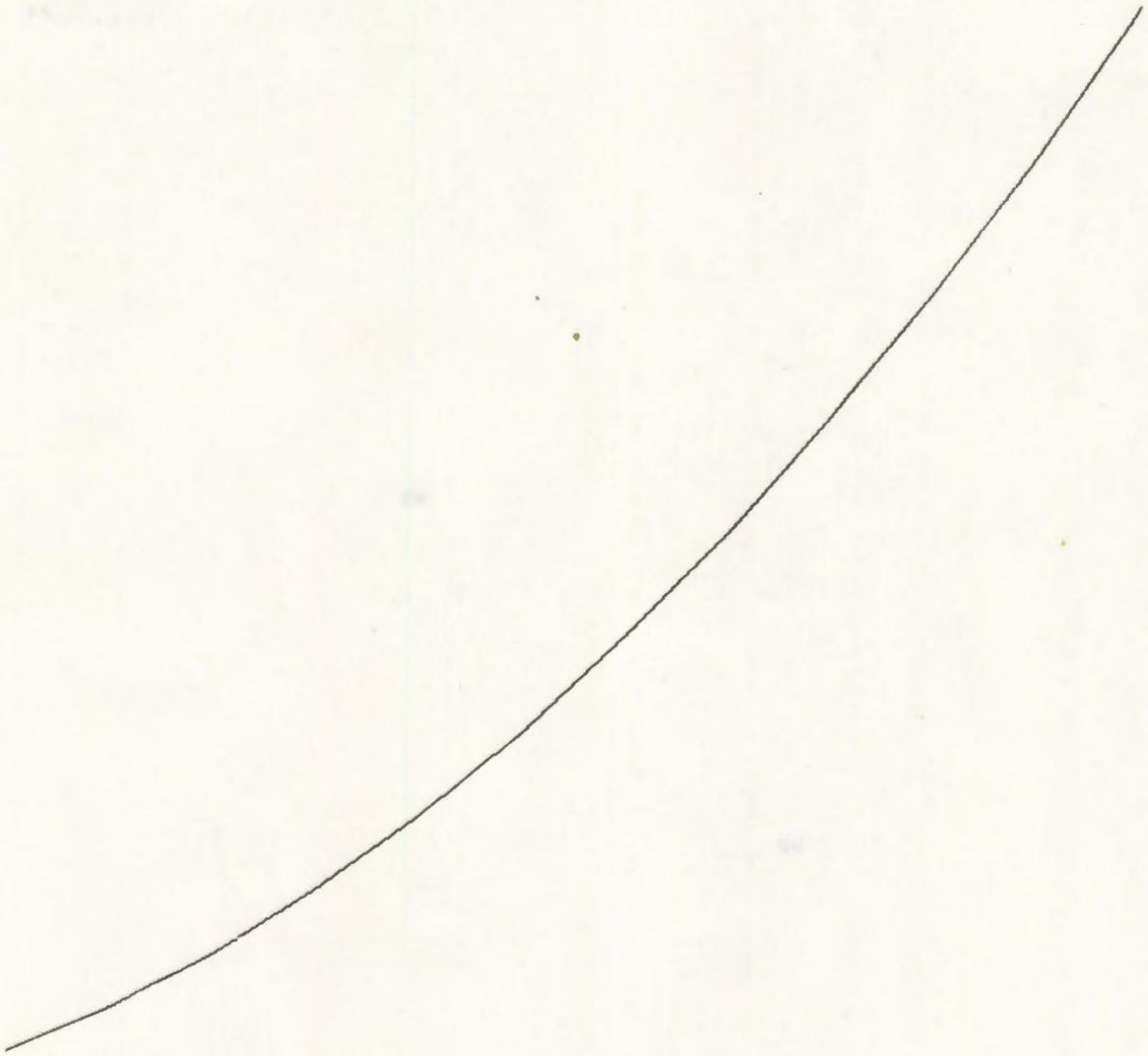


```
#I[1]:: t:Ar[4,Pt[{$x,$x^2}]]
#O[1]: {Pt[{1,1}],Pt[{2,4}],Pt[{3,9}],Pt[{4,16}]}
#I[2]:: Plot[t]
```

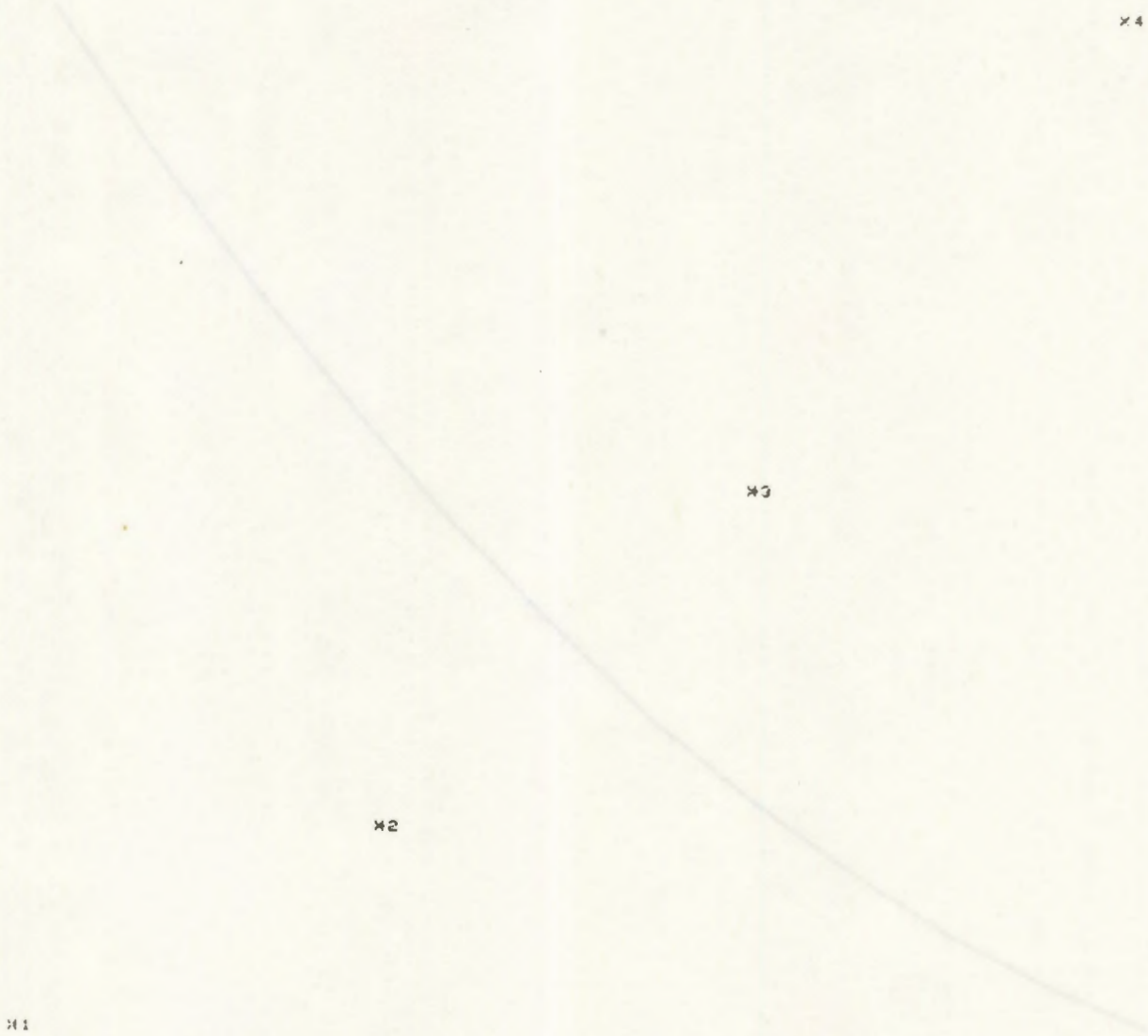
```
#I[3]:=  Plot[{Line[t]}]
```



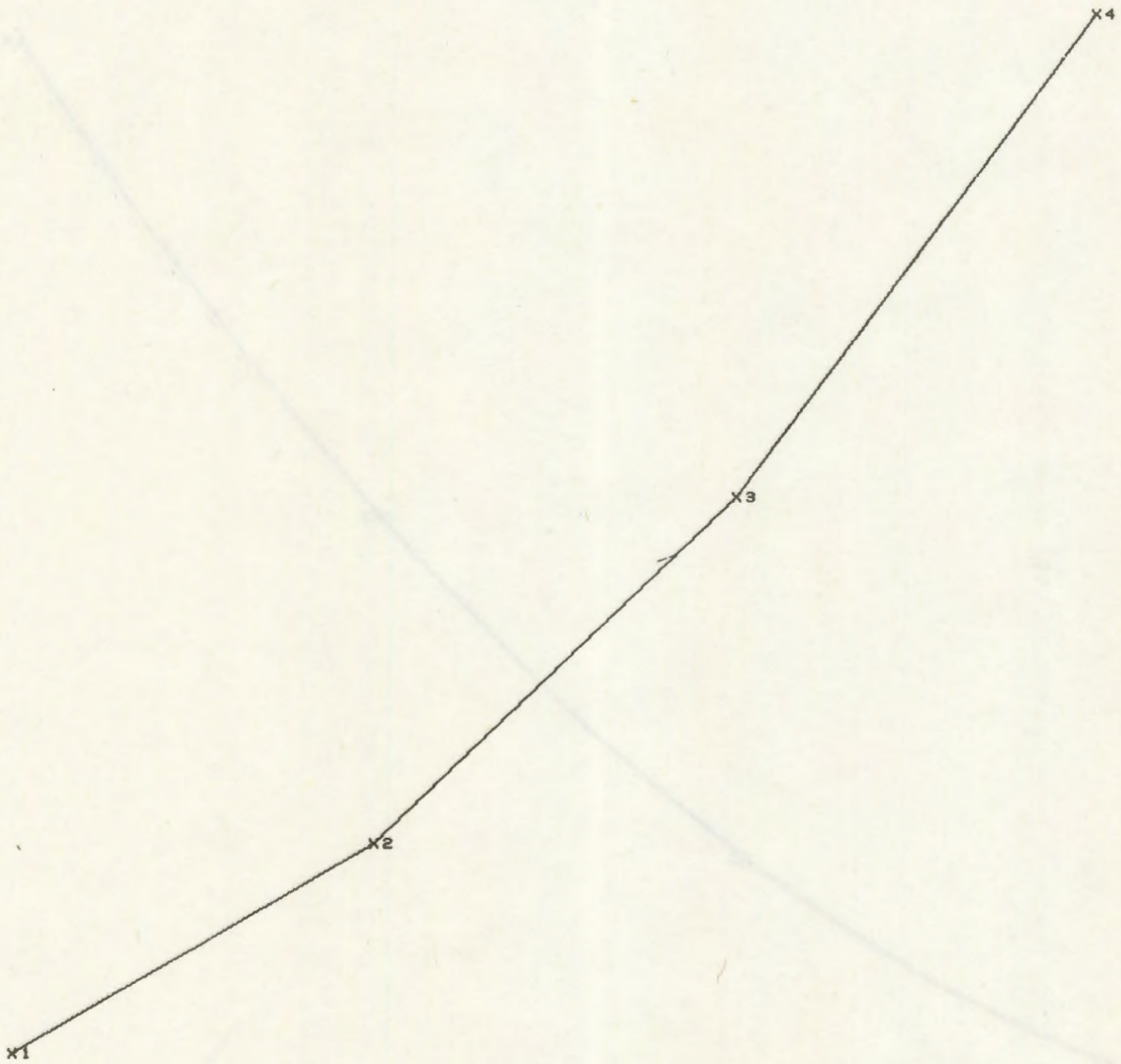
```
#I[4]:: Plot[{Curve[t]}]
```



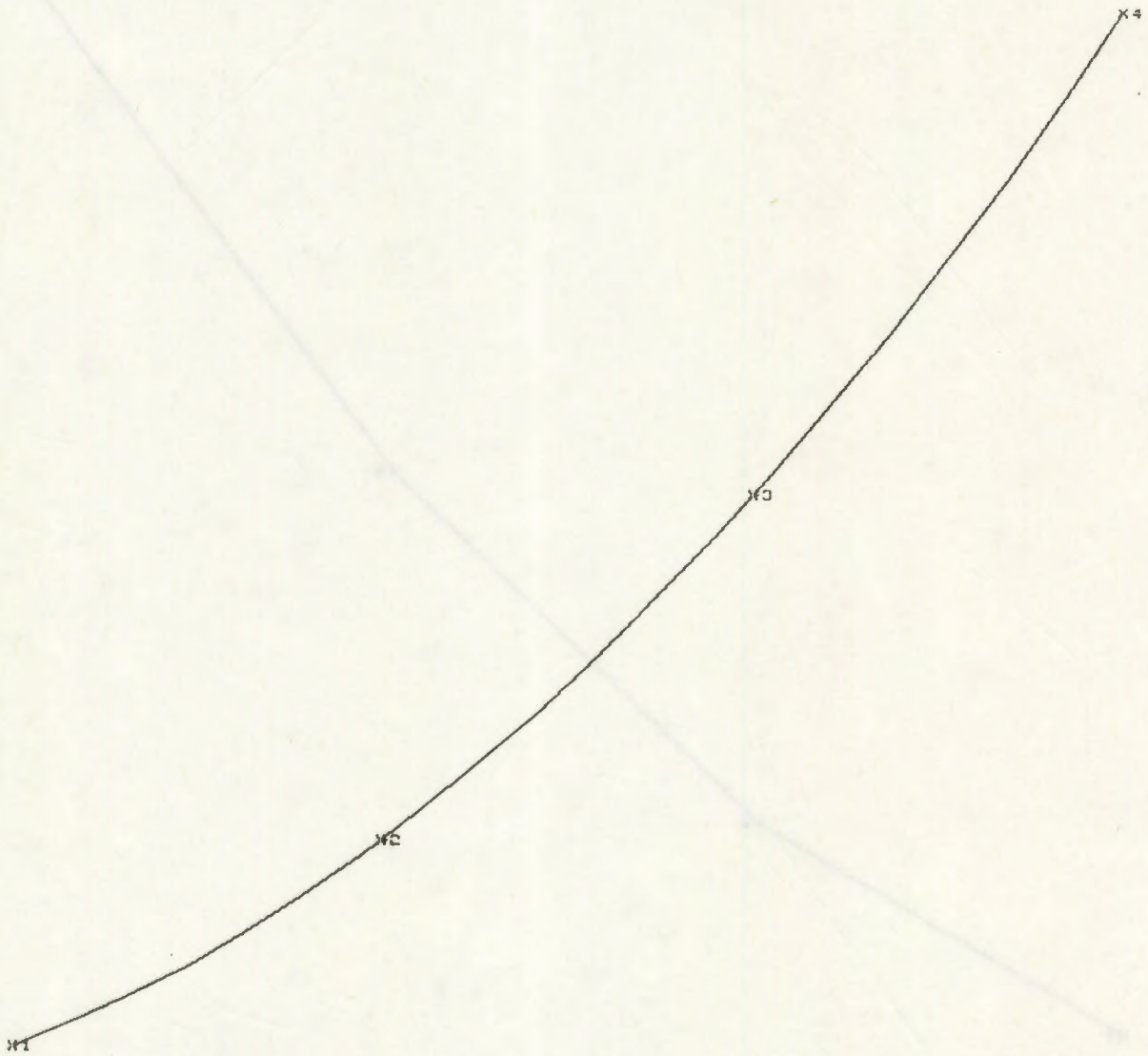
```
#I[5]::   u:Ar[4,Pt[{$x,$x^2},Make[X,$x]]]
#O[5]:    {Pt[{1,1},X1],Pt[{2,4},X2],Pt[{3,9},X3],Pt[{4,16},X4]}
#I[6]::   Plot[u]
```



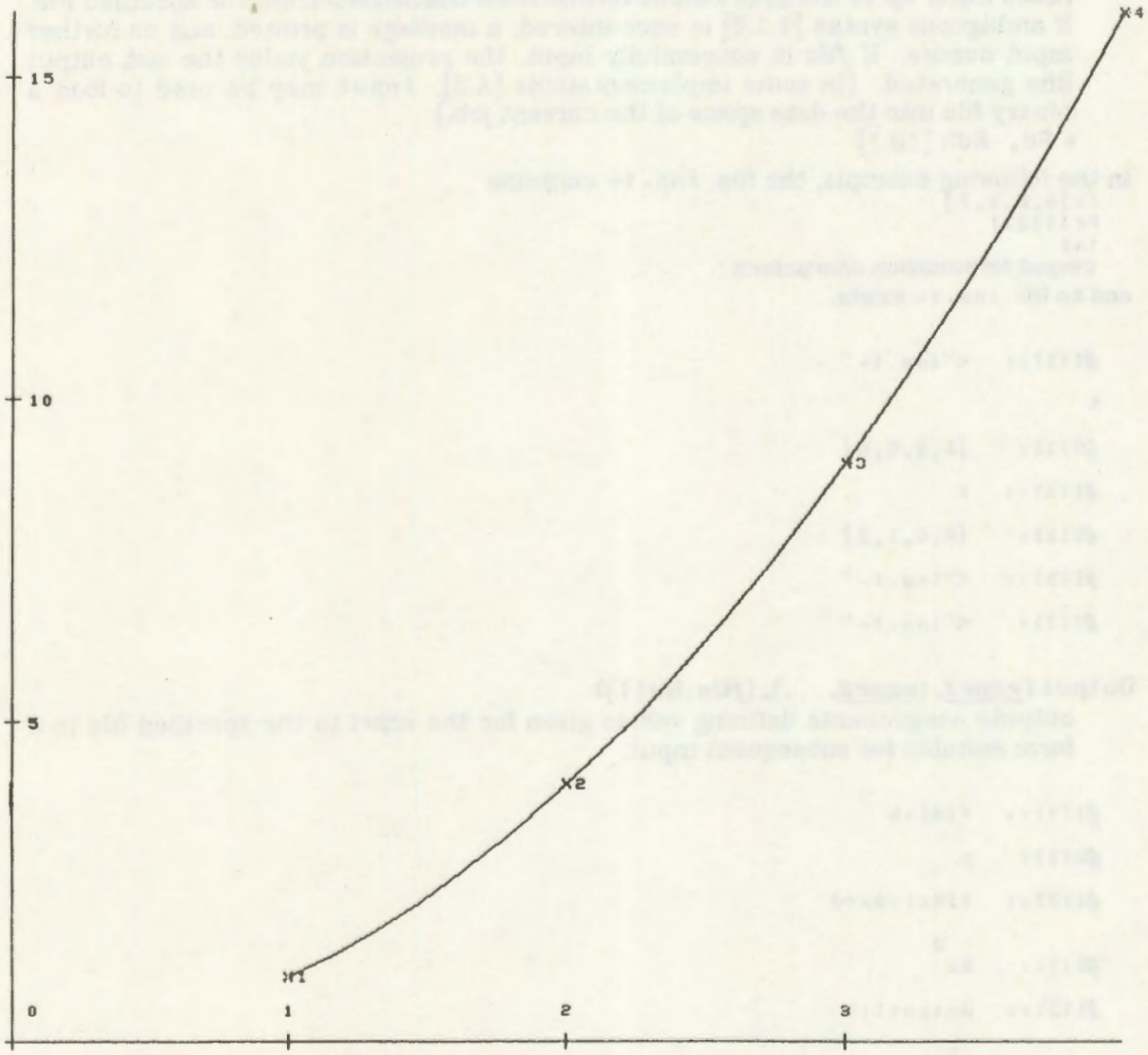
```
#I[7]:: Plot[{Line[u]}]
```



```
#I[8]::   Plot[{Curve[u]}
```



```
#I[9]::   Plot[{Curve[u],Axes[{0,0}]}]
```



10.3 File input and output

Files are specified by single symbols (with names enclosed in " " if necessary [2.2]). The portion of a file after the *n*th input line is labelled by {*filespec*,*n*}. {*filespec*,-*n*} specifies the portion following the *n*th line from the end of the file. Output is by default appended to files. Specification of {*file*,0} causes new output to overwrite existing contents of *file*. The standard input/output medium (usually terminal) is considered as a special file denoted by the symbol `Null`. Input and output characteristics of a file are specified by a numerical code (defined in [A.3]).

<*file* or **`Input[`*file*`]`**
reads input up to the first ⊂*input termination character*⊃ from the specified file. If ambiguous syntax [1.1,2] is encountered, a message is printed, and no further input occurs. If *file* is successfully input, the projection yields the last output line generated. (In some implementations [A.3], `Input` may be used to load a binary file into the data space of the current job.)
• `Rd`, `Rdh` [10.1]

In the following example, the file `inp.f+` contains

```
f:{4,5,1,2}
Pr[f[2]]
f+4
⊂input termination character⊃
```

and no file `inp.f-` exists.

```
#I[1]::  <"inp.f+"

5

#O[1]:   {8,9,5,6}

#I[2]::  f

#O[2]:   {4,5,1,2}

#I[3]::  <"inp.f-"

#O[3]:   <"inp.f-"
```

`Output[`*expr1*`,`(*expr2*`,...),(`*file*`:Null)]`
outputs assignments defining values given for the *expri* to the specified file in a form suitable for subsequent input.

```
#I[1]::  f[0]:b

#O[1]:   b

#I[2]::  f[$x]:$x^3

           3
#O[2]:   $x

#I[3]::  Output[f]

           3
f[$x] : $x
f[0] : b

                                  3
#O[3]:   {[0]: b, [$x]: $x }
```

```
#I[4]::  a:5

#0[4]:   5

#I[5]::  Output[f,a,]

          3
f[$x] : $x
f[0] : b
a : 5

#0[5]:   5

#I[6]::  Output[f,fadef]

                              3
#0[6]:   {[0]: b, [$x]: $x  }

#I[7]::  Output[a,f,fadef]

                              3
#0[7]:   {[0]: b, [$x]: $x  }
```

The file `fadef` then contains

```
f[$x] : $x^3
f[0] : b

a : 5
f[$x] : $x^3
f[0] : b
```

Open[*file*, (*code*:1)]
initiates entry of all subsequent input and output expressions into the specified file, in a mode defined by *code* [A.3].

Close[*file1*, *file2*, . . .]
terminates entry of input and output expressions into the specified files.

`Close[]` stops the printing of any output on the standard output medium until `Open[]` occurs.

- [1.3]

The file `rec` then contains

```
#0[3]:   {rec}

#I[4]::  b:2

#0[4]:   2

#I[5]::  Open[]

#0[5]:   {}

#I[6]::  a

#0[6]:   1

#I[7]::  Close[rec]

#0[3]:   {rec}

#I[4]::  b:2

#0[4]:   2

#I[5]::  Open[]
```

```
#O[5]:   {}
#I[6]::  a
                2
#O[6]:   x
#I[7]::  Close[rec]
```

10.4 Memory management

(Implementation dependent)

Gc[]

reclaims memory not required for further processing (by "compacting garbage collection").

```
#I[1]:: Pr[State[]];Rpt[Rex[],20];Pr[State[]];Gc[];Pr[State[]];

{0,12295,12405}
{0,21136,21237}
{12201,12212,24437}
```

10.5 External operations

Init[(*expr1, expr2, . . .*)]
defines various external parameters as specified in [A.3].

Exit[(*expr*)]
terminates the current job, passing the textual form of *expr* as an "exit code" [A.3] to the monitor (shell).

Run[*expr*, (*arg1, arg2, . . .*)]
executes the textual form of *expr* (printed by Lpr [10.1]) as a monitor (shell) program [A.2], using the textual forms of the *argi* as input [A.2]; the text of any output [A.2] generated is simplified and given as the image [1.10].

- [1.6]

In the following example, the monitor program `math.eta` reads two numbers as input, and yields a number as output.

```
#I[1]:  Run["math.eta",0.5,1]
#O[1]:  5.61124
#I[2]:  Run["math.eta",0.5]
#I[3]:  Run["math.eta",0.5,1,1.4]
#O[4]:  5.61124
```

10.6 Character string manipulation

Ed[*expr*]
enters edit mode [1.7], with the textual form of *expr*, as printed by Lpr, in the edit buffer, and yields as a result the edited expression.

Edh[*expr*]
enters edit mode with the text of a partially simplified form [3.5] of *expr* in the edit buffer.
• Ev [3.7]

```
#I[1]:: f:D[Log[Log[x]],x]

                1
#O[1]:  --------
        x Log[x]

#I[2]:: Ed[@1]

       1/(x*Log[x])
<edit>           ^+3
       1/(x*Log[x+3])
<edit>

                1
#O[2]:  ------------
        x Log[3 + x]

#I[3]:: Edh[#I[1]]

       f : D[Log[Log[x]],x]
<edit>                 ^1/
       f : D[Log[Log[1/x],x]
<edit>

               -1
#O[3]:  --------
                1
        x Log[--]
              x
```

Make[(*start*:#),(*expr*:*(next integer)*)]
generates a symbol with name obtained by concatenating the textual form of *start* with the textual form of *expr*, or, by default, with the smallest positive integer necessary to form a previously unused name.

```
#I[1]:: a:Make[]+Make[]^2

                  2
#O[1]:  #1 + #2

#I[2]:: {Make[x],Make[a+t],Make[h,2],Make[h]}

#O[2]:  {x1,"a + t1",h2,h1}
```

Expl[*expr*]
gives a list of numerical codes for each of the characters appearing in the textual form of *expr* (as printed by Lpr). Characters are numbered from 0 to 94 in the order:
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
⊂*space*⊃!"#$%&\'()*+,-./:;<=>?@[\]^_\`{|}~

Impl[{*n1*, *n2*, *n3*, ...}]
generates a symbol whose name consists of the characters specified by the numerical codes *n1*, *n2*, *n3*,...

```
#I[1]::  Expl[abc547D]

#O[1]:   {10,11,12,5,4,7,39}

#I[2]::  Imp[%]

#O[2]:   Imp[{10,11,12,5,4,7,39}]

#I[3]::  t:a^2+b+c

                    2
#O[3]:   b + c + a

#I[4]::  Exp[t]

                      2
#O[4]:   Exp[b + c + a ]

#I[5]::  Impl[%]

                           2
#O[5]:   Impl[Exp[b + c + a ]]

#I[6]::  Expl[SMP]

#O[6]:   {54,48,51}

#I[7]::  Cat[%,%,%]

#O[7]:   {54,48,51,54,48,51,54,48,51}

#I[8]::  Impl[%]

#O[8]:   SMPSMPSMP
```

10.7 Programming aids

Step [*expr*, (*nest*:1)]
steps through the simplification of *expr*. Each segment in procedures [6.3] or iteration structures [6.2] nested to depth less than *nest* is printed, and an interactive subsidiary procedure [1.8] is initiated.

```
#I[1]::  Step[a:b;b:2^4;Pr[a];a+2]

%I[1]:   a : b

%I[1]::  a

%O[1]:   b

%I[2]::  a:b+3

%O[2]:   3 + b

%I[3]::
⊂input termination character⊃

                  4
%I[2]:   b : 2

%I[1]::  b

%O[1]:   16

%I[2]::
⊂input termination character⊃

%I[3]:   Pr[a]

19

%I[1]::  %%

%O[1]:   %%

%I[2]::
⊂input termination character⊃

%I[4]:   a + 2

%I[1]::
⊂input termination character⊃

#O[1]:   21

#I[2]::  a

#O[2]:   19
```

- Trace [4]
- Ev [3.7]

Struct [*expr*]
prints a schematic picture of the internal representation of *expr*.

```
#I[1]::  t:5+a^(b+2d+c^2)+b^(3-x)

                              2
                 b + 2d + c       3 - x
#0[1]:   5 + a                 + b

#I[2]::  Struct[t]

lmb  -> proj: Plus
        num: 5
        lmb  -> proj: Pow
                sym: a
                lmb  -> proj: Plus
                        sym: b
                        sym: 2 d
                        lmb  -> proj: Pow
                                sym: c
                                num: 2
        lmb  -> proj: Pow
                sym: b
                lmb  -> proj: Plus
                        num: 3
                        sym: -1 x

                              2
                     b + 2d + c       3 - x
#0[2]:   Struct[5 + a                 + b     ]
```

10.8 Performance analysis

State[]

yields a list giving the number of memory "blocks" used (after last memory reclamation, at present, and maximum so far). One block is the memory required to store a single symbol (usually 16 bytes [A.5]).

- [1.4]

```
#I[1]::  State[]

#O[1]:   {8,12322,12337}
```

Size[*expr*]

yields a list whose first entry is the actual number of memory blocks occupied by *expr*, and whose second entry is the number which would be occupied if all common subexpressions were stored separately.

```
#I[1]::  Size[a]

#O[1]:   {1,1}

#I[2]::  Size[a+b]

#O[2]:   {3,3}

#I[3]::  r:x

#O[3]:   x

#I[4]::  Rpt[r:r/(1+r),4]

                                   x
#O[4]:   ------------------------------------------------------------
                             x
         (1 + x) (1 + --------------------)
                                      x
                      (1 + x) (1 + -----)
                                   1 + x

                                            x
                   * (1 + -------------------------------------------)
                                             x                   x
                          (1 + x) (1 + --------------------) (1 + -----)
                                                x            1 + x
                                       (1 + x) (1 + -----)
                                                    1 + x

                             x
                   * (1 + -----)
                          1 + x

#I[5]::  Size[r]

#O[5]:   {36,84}
```

Time[*expr*, (*n*:1)]

simplifies *expr* *n* times, and yields an Err projection [2.1] giving the approximate average CPU time in "clicks" [A.5] used for each simplification.

```
#I[1]:: t:S[x,$x->Log[$x],10]
#O[1]: Log[Log[Log[Log[Log[Log[Log[Log[Log[Log[x]]]]]]]]]]
#I[2]:: Time[D[t,x,x],5]
#O[2]: Err[.4833333,0]
```

10.9 Program construction

(Implementation dependent)

Cons[{*f1*, . . .}, {(*file1*:), . . .}, (*code*:0)]

constructs if possible an external program which obtains the values assigned to projections of the *fi*. Resulting programs are placed in the files *filei*. The language and treatment of the external programs is determined by *code* as specified in the implementation notes. With certain *code*, the external programs assume that all symbols take on numerical values. Non-local variables are assumed to have constant values. Projections whose evaluation may be carried out by external programs carry property **Cons**.

```
#I[1]::  f[$x_?Numbp[$x]]:Exp[$x]+3$x^2-1

                                  2
#O[1]:   -1 + Exp[$x] + 3 $x

#I[2]::  Time[f[5],100]

#O[2]:   Err[.007166667,.00858131]

#I[3]::  Cons[f]

#I[4]::  Time[f[6],100]

#O[4]:   Err[0.000854,0.000892]
```

The file `f.c` then contains

```
#include <smplib>

        /*
f[$x_?Numbp[$x]] : -1 + Exp[$x] + 3 $x^2
        */

double f(x)
double x;
{
double exp();
return(-1.0+exp(x)+3.0*x*x);
}
```

10.10 Asynchronous and parallel operations

(Implementation dependent)

Some implementations allow a set of independent processes to be performed in parallel, either as asynchronous jobs on a single computing unit, or as jobs in separate computing units.

Processes (including procedures within them) are specified by a unique expression used as a name: the basic process is `Null`. A particular expression may be modified by only one of a set of parallel processes. The order of operations in different processes is usually not determined.

`Fork[(`*expr*`:Null),(`*name*`:`*(next integer)*`),(`*pri*`:1)]`
initiates the named parallel process to simplify *expr* at priority *pri*, yielding *name*. If *name* is not specified, a unique integer name is assigned to the process. Any existing process *name* is terminated. Several processes competing for a single computing unit are executed at higher priorities for lower *pri*. Processes on separate computing units are executed when possible with instruction times in ratios given by *pri*.

`Wait[{`*name1*`,`*name2*`,...}]`
waits for completion of the processes *name1*, *name2*, ..., yielding the resulting {*expr1*, *expr2*, ...}

`Para[`*expr1*`,`*expr2*`,...]`
is equivalent to `Wait[{Fork[`*expr1*`],Fork[`*expr2*`],...}]` and simplifies the *expri* in parallel, yielding a list of the results.

`Fork[`*mess*`,`*code*`]` may be used to transfer *mess* to `Wait[`*code*`]` in another process.

`Fork[,`*name*`]` terminates the process *name*, possibly from within *name*.

`Clock[(`*name*`:`*(present process)*`)]`
yields a list of the total elapsed computing unit time (in clicks) and total elapsed real time (in seconds) since the initiation of the specified process (0 if the process is not executing).

`Rti[`*code*`]`
represents a real-time interrupt whose value is simplified immediately on receipt of the interrupt *code*.

- [1.4]

Appendix. External interface

A.1 Introduction

This appendix describes in general terms features of SMP affected by its external enviroment. Details of these features vary between different implementations of SMP. Information for a particular implementation should be given in the "Implementation Notes". Mechanisms for features under different operating systems will not be described; they are discussed in the "SMP Implementation Guide" (available separately).

A.2 External operations

Typical external operations provided in SMP implementations are:

Hard[(*expr*),(*code*)]
generates a hard copy of *expr* on the device specified by *code*. Hard[] yields a hard copy of all input and output expressions. Graphics output is given if possible.

Dsp[(*file*:smp.out)]
prints the specified file.

Save[(*rec*:smp.out),*file*]
creates a permanent copy *file* of the record file *rec*.

Send[(*uname*)]
enters send mode: arbitrary text terminated by ⊂*input termination character*⊃ is sent to the location or user identified by *uname*. ⊂*break interrupt*⊃ may be used to include SMP expressions. Send[] sends the text to a central SMP report file at each installation.

Dir[*dir*]
changes the default "user" file directory [A.4] to *dir*. Dir[] resets to the directory given at initialization.

A.3 Terminal characteristics

The following are ASCII equivalents for non-alphabetic characters used in this handbook: ! (041) 042 () # (043) $ (044) % (045) & (046) ' (047) ((050)) (051) * (052) + (053) , (054) - (055) . (056) / (057) : (072) ; (073) < (074) = (075) > (076) ? (077) @ (100) [(133) 134 ()] (135) ^ (136) _ (137) ` (140) { (173) | (174) } (175) ~ (176)

Replacements for input text may be specified using `Sxset` [2.11].

Characteristics of a terminal or file may be specified in `Open` or `Init` as a list whose entries usually include

Number of lines printed before pause for end of page. (Input of newline continues printing). (`Inf` for no pause).

Position of first character to be printed on left.

Position of last character to be printed on right.

Hardware tab spacing (0 if none).

Type of graphics mode (0 if none).

Graphics mode entry code.

Graphics mode return code.

Screen clear/form feed code.

Many other parameters may be provided in a particular implementation.

Common classes of terminals may be specified by a single integer code, as defined in the implementation notes.

A.4 External files

If no explicit file directory is specified, external files are first assumed to be in a default "user" directory, and failing that in a central "library" directory. The default directories are specified by `Init` [A.5].

The names of external files provided with releases of SMP all begin with the letter `X`. Names of new external files should begin with letters other than `X`. Files whose names end with `SX` contain syntax modifications [2.11].

Most external files contain SMP input lines. In some versions of SMP, external files may also contain direct binary forms of SMP expressions. Such files are recognized and treated appropriately by `Input` [10.3]. The names of binary files should usually end with `#B` or `.B`

The record file [1.3] for each SMP job is placed in the "user" directory, and usually named `smp.out`

A.5 Initialization

Init[(*udir*:),(*libdir*:),(*term*:)]
specifies default "user" and "library" directories, and defines characteristics of the terminal.

If provided, SMP jobs read an initialization file, usually named `smp.init`
Mechanisms for locating `smp.init` are described in "SMP implementation guide". A typical `smp.init` file is:

```
                    /**  SMP initialization file  **/
Init["$HOME","/u1/smp/X",1]

  /* Open smp.out record file */
       Open[{"smp.out",0}]
```

When an SMP job is initiated, it is often possible to pass "arguments" to the job, giving files to be read for initialization (after `smp.init`).

When an SMP job terminates, it passes by default a "successful completion" exit code to the monitor; other exit codes may be specified in `Exit`.

If provided, a termination file `smp.end` is read before termination of an SMP job.

A.6 System characteristics

The "block" is the basic unit of memory used by SMP. Its physical size in terms of bytes may vary from one implementation to another: in most cases, one block is 16 bytes.

- State [10.8]

The "click" is the basic unit of CPU time used in SMP. A "click" is defined by the time taken to execute certain initialization procedures; the time for operations in different installations should be roughly a fixed number of "clicks". On a VAX 11/780 one "click" corresponds to approximately one second.

- #T [1.2]
- Time [10.8]
- Clock [10.10]

A.7 External programs

External programs may be entered explicitly or may be constructed from SMP definitions using Cons [10.9]. They may be run explicitly using Run [1.10,10.9], or may be defined by Cons to be used automatically by SMP in simplifying particular projections. Each invocation of an external program receives only one set of parameters from Run or its associated projection. External programs may usually return any number of input lines to SMP. The simplification of a projection involving an external program is complete when the external program terminates.

External programs invoked by Run may communicate with SMP by one of several mechanisms:

1. Take input and output on the standard input and output media, but pass them through pre- and post-processors which direct them to and from SMP.
2. Use the SMPIO library functions to obtain input directly from SMP, and pass output directly to SMP. In this case, additional input and output may occur on the standard input/output medium.

The SMPIO library is usually included in external programs by an option for the compiler used.

For the C language the SMPIO library contains the functions `fromsmp` and `tosmp`, analogous to `scanf` and `printf` respectively. The following conversion characters may be used in the control string:

`%n` Decimal number (in SMP *^ format [2.1]).

`%s` Character string.

`fromsmp` and `tosmp` usually use the input-output channel 3.

External programs constructed with Cons are usually linked directly as the values of projections in a running SMP job.

INDEX

SMP Library

July 10, 1981

Introduction

The SMP library provides additional facilities relevant to particular applications of SMP. Most of the items in the library are based on external files consisting of SMP input expressions. In some implementations, the files actually loaded into SMP jobs are in a compiled binary form. Certain items in the library may be included as built-in facilities in some SMP implementations.

The names of external files may differ between implementations: rules for modifications of the names used below should be given when necessary in the Implementation Notes.

Two directories of external files are given below. The topic directory is based on a classification of application areas. The section directory is based on the sections of the Summary and Reference Manual. Finer subdivisions of the topics will be given when more items are available.

The complete text of the external files is reproduced below. Examples are included as commentary where appropriate.

As well as providing additional facilities, the files given below are intended to illustrate possible applications and methods. The library will be greatly expanded and improved in later versions. Existing files should be taken as models for future files.

Topic directory

A. Mathematics

1. Fundamentals

XAck Ackermann function
XDios Solution of Diophantine equations
XFib Fibonacci numbers
XGr Basic graph theory
XLCM Lowest common multiple
XLogic Elementary laws in propositional calculus
XLogic2 Elementary logic with quantifiers
XLogicPr Logical truth tables
XSets Elementary finite set theory
XSetsSX Set theory notation
XTup n-tuples

2. Algebra

XArperm Generation of permutations
XBase Conversion of integers in arbitrary number bases
XFrac Fractions
XMat1 Matrix input and generation
XMat2 Structural matrix operations
XMat3 Matrix character tests
XMat4 Algebraic matrix operations
XPerm0 Permutations
XPerm1 Elementary operations on permuations
XPermC Cycle decomposition of permutations
XRslt Polynomial resultants
XSympol Generate symmetric polynomials

3. Analysis

XAbs Extensions for Abs
XAir Airy and related functions
XAng Anger and Weber functions
XBer Bernoulli polynomials
XBes1 Functional relations for Bessel functions
XBes2 Recurrence relations for Bessel functions
XBes3 Bessel functions of integer order
XBes4 Bessel functions of half odd integer order
XBes5 Special cases for half odd integer order Bessel functions
XBeta Euler beta function
XChe Chebychef polynomials
XChg Confluent hypergeometric function
XDSol Series solution of differential equations

XTr27 Elementary transcendental functions - 2.7
XTr28 Elementary transcendental functions- 2.8
XTr29 Elementary trancendental functions - 2.9
XTr2a Elementary transcendental functions - 2.10
XTr311 Elementary transcendental functions - 3.1.1
XTr312 Elementary transcendental functions - 3.1.2
XTr32 Elementary transcendental functions - 3.2
XTr33 Elementary transcendental functions - 3.3
XTr4 Elementary transcendental functions - 4
XTr5 Elementary transcendental functions - 5
XVecan Three-dimensional vector analysis
XWhi Whittaker function
XWron Wronskian and Jacobian
XZeta Riemann zeta function

4. Geometry and topology

XRot2 Rotations in two dimensions
XRot3 Rotations in three dimensions
XPolar Polar graphs
XPlot Operations on plots

5. Applied mathematics

XFit Curve fitting
XHorn Horner representation
XInfo Basic information theory
XItp Lagrange interpolation of list values
XLItp Interpolation of contiguous list values
XRandC Generation of random numbers from continuous distributions
XRandD Generation of random numbers from discrete distributions
XRandL Random selection of list elements
XStat Statistical properties of univariate distributions
XTuring Turing machine simulation

B. Physical sciences

1. Classical mechanics

XLor Lorentz vectors

2. Fluid mechanics
3. Statistical mechanics
4. Properties of matter
5. Electrodynamics
6. Quantum theory

G Dirac gamma matrix manipulation
XFierz Fierz transformations
XPauli Representation of Pauli sigma matrices

7. Astrophysics and gravitation
   - XLevi Generate Levi-Civita tensor
8. Chemistry
9. Earth sciences
10. Physical quantities
    - XDim Dimensional analysis
    - XMKS MKS/SI units
    - XNAT Natural units
    - XPhys Fundamental physical constants

C. Life and social sciences

1. Biology
2. Medicine
3. Sociology
4. Economics

D. Technology

1. Mechanical engineering
2. Civil engineering
3. Hydraulic and aeronautical engineering
4. Electrical and optical engineering
5. Chemical engineering
6. Systems engineering

E. Miscellaneous

Section directory

3. Replacements and substitutions
4. Numerical evaluation
5. Deferred simplification
6. Pre-simplification
7. Partial simplification

4. Properties

5. Relational and logical operations
   - XLogic Elementary laws in propositional calculus

6. Control structures
   1. Conditional statements
   2. Iteration
   3. Procedures and flow control

7. Structural operations
   1. Projection and list generation
      - XArperm Generation of permutations
      - XTup n-tuples
   2. Template application
      - XDap Directional application
      - XNMap Multi-element generalization of Map
   3. Part extraction and removal
      - XLev Isolate single level
   4. Structure determination
      - XLenex Length of expanded expressions
      - XLPart List positions of all parts
   5. Content determination
      - XAny Test for any elements of list satisfying condition
   6. Character determination
      - XGenp Test for generic symbols
      - XIntp Additional rules for integer testing
   7. List and projection manipulation
      - XContig Make list contiguous
      - XDap Directional application
      - XInd Manipulation of indices in lists
      - XLArith Arithmetic operations on lists
      - XList0 Basic list manipulations
      - XList1 Operations on sublists
      - XMaxind Find maximal index
      - XPeel Peel away sublists
      - XProj Projection manipulation
      - XUnFlat List unflattening

8. Distribution and expansion
   - XLenex Length of expanded expressions
9. Rational expression manipulation and simplification
10. Statistical expression generation and analysis
    - XInfo Basic information theory
    - XRandL Random selection of list elements
    - XRpoly Random polynomial generation

8. Mathematical functions

1. Introduction
2. Elementary arithmetic operations
   - XExDot Expansion of dot products
3. Numerical functions
   - XAbs Extensions for Abs
   - XRandC Generation of random numbers from continuous distributions
   - XRandD Generation of random numbers from discrete distributions
4. Mathematical constants
5. Elementary transcendental functions
   - XTr1 Elementary transcendental functions - 1
   - XTr21 Elementary transcendental functions - 2.1
   - XTr22 Elementary transcendental functions - 2.2
   - XTr23 Elementary transcendental functions - 2.3
   - XTr24 Elementary transcendental functions - 2.4
   - XTr25 Elementary transcendental functions - 2.5
   - XTr26 Elementary transcendental functions - 2.6
   - XTr27 Elementary transcendental functions - 2.7
   - XTr28 Elementary transcendental functions- 2.8
   - XTr29 Elementary trancendental functions - 2.9
   - XTr2a Elementary transcendental functions - 2.10
   - XTr311 Elementary transcendental functions - 3.1.1
   - XTr312 Elementary transcendental functions - 3.1.2
   - XTr32 Elementary transcendental functions - 3.2
   - XTr33 Elementary transcendental functions - 3.3
   - XTr4 Elementary transcendental functions - 4
   - XTr5 Elementary transcendental functions - 5
6. Gamma, zeta and related functions
   - XBer Bernoulli polynomials
   - XBeta Euler beta function
   - XEul Euler polynomials and numbers
   - XGamma Gamma function
   - XLer Lerch transcendent
   - XZeta Riemann zeta function

2. Evaluation of sums and products
   - XIter General iterated forms
   - XSum Summation of series
3. Solution of equations
   - XDios Solution of Diophantine equations
   - XLdEq Lists of equations
   - XSol Inverses of elementary transcendental functions
4. Differentiation and integration
   - XInt Elementary definite integrals
   - XVecan Three-dimensional vector analysis
5. Series approximations and limits
6. Matrix and explicit tensor manipulation
   - XCon Tensor contraction
   - XDap Directional application
   - XLevi Generate Levi-Civita tensor
   - XMat1 Matrix input and generation
   - XMat2 Structural matrix operations
   - XMat3 Matrix character tests
   - XMat4 Algebraic matrix operations
   - XNorm Norm of a vector
   - XTEx Tensor expansion

10. Non-computational operations
    1. Input and output operations
    2. Graphical output
       - XPlhist Plot histogram
       - XRot2 Rotations in two dimensions
       - XRot3 Rotations in three dimensions
    3. File input and output
       - XWatch Watching external file input
    4. Memory management
       - XKillIO Kill Input/Output
    5. External operations
    6. Character string manipulation
       - XChar Character manipulation
       - XStr0 Basic character string manipulation
       - XStr1 Further character string manipulation
    7. Programming aids
    8. System performance analysis
    9. Program construction
    10. Asynchronous and parallel operations

G

G[*p1, p2, p3, . . .*]
<Flat>

represents a product of Dirac gamma matrices. G[] is the identity gamma matrix. The *pi* may have the types:

Gvec or Gvec[(*n*:4)]
: "Slashed" Lorentz vector of dimension *n* (default type).

Gscal
: Lorentz scalar.

Gind
: Gamma matrix with "dummy" index.

The implicit Dirac spinor indices carried by G projections are contracted by Dot projections. Conj forms the complex conjugate of a G projection.

The following additional objects may appear in G projections:

G5
: The pseudoscalar gamma matrix.

Usp[*p*, (*m*:0)]
: Spinor representing solution to free Dirac equation with momentum *p* such that *p*.*p*: *m*^2.

Uspb[*p*, (*m*:0)]
: Spinor conjugate to Usp[*p*, *m*].

Tr yields the trace of a product of gamma matrices represented by G projections.

```
#I[1]::  G[p,p,p]

#O[1]:   p.p G[p]

#I[2]::  p_q_k_Gvec

#O[2]:   Gvec

#I[3]::  x_m_Gscal

#O[3]:   Gscal

#I[4]::  G[p+m,p+x q]

#O[4]:   G[p,p + q x] + m G[p + q x]

#I[5]::  Ex[%]

#O[5]:   m G[p] + x G[p,q] + G[] p.p + m x G[q]

#I[6]::  G[p,q].G[k,q]

#O[6]:   G[p,q,k,q]

#I[7]::  Conj[%]

#O[7]:   G[q,k,q,p]

#I[8]::  Tr[%]

#O[8]:   -4 k.p q.q + 8 k.q p.q
```

XAbs

```
                    /**  Extensions for Abs  **/

/* Theta[x]
        represents the unit step function. */

/* Ramp[x]
        represents the unit ramp function. */
        Ramp[$x] : (Abs[$x] + $x)/2

/* Further simplification rules for Abs */
Abs[$x $$x] :: Abs[$x] Abs[$$x]
Abs[$x^($n_?Natp[$n])] : Abs[$x]^$n
D[Abs[$x],{$x,1,$y}] : Sign[$y]
Abs[Sign[$x_?($x!=0)]] : 1

/* Further simplification rules for Sign */
Sign[$x $$x] :: Sign[$x] Sign[$$x]
Sign[$x^($n_?Evenp[$n])] : 1
Sign[Abs[$x]] : 1

/*
#I[1]::  <<XAbs

#I[2]::  Abs[a b^2 c]

                      2
#O[2]:   Abs[a] Abs[b]  Abs[c]

*/
```

XAck

```
                    /**  Ackermann function  **/

/* Ack[x,y]
        Ackermann function. */
        Ack[0,$y] : Mod[$y+1,3]
        Ack[$x,0] : $x+1
        Ack[$x,1] : $x+2
        Ack[$x,2] : 2$x
        Ack[$x,3] : 2^$x
        Ack[$x,4] :: Arrow[2,$x+1]
        Ack[$x,$y] :: Ack[$x,$y] : Ack[Ack[$x-1,$y],$y-1]


/* Arrow[n,m]
        Knuth arrow function n^(n^(n^...))) with m powers. */
        Arrow[$n,0] : 1
        Arrow[$n,1] : $n
        Arrow[$n,$m] :: $n^Arrow[$n,$m-1]

/*

#I[1]::  <<XAck

#I[2]::  Ar[10,Arrow[$i,2]]

#O[2]:   {1,4,27,256,3125,46656,823543,16777220,387420500,1.*^10}

#I[3]::  Ack[2,5]

#O[3]:   16
```

```
#I[4]::  Ack[3,5]
*/
```

XAir

```
                /** Airy and related functions **/

SAir_:Ldist

SAir[1]: AirAi[$z] -> 1/Pi ($z/3)^(1/2) BesK[1/3,2/3 $z^(3/2)]
                                                    /* MOS p. 75 */

SAir[2]:          AirBi[$z] -> ($z/3)^(1/2) (BesI[-1/3,2/3 $z^(3/2)] + \
                        BesI[1/3,2/3 $z^(3/2)])
                                                    /* MOS p. 75 */

SAir[3]: AirAi[$z] -> 1/3 (-$z)^(1/2) (BesJ[1/3,2/3 (-$z)^(3/2)] + \
                        BesJ[-1/3,2/3 (-$z)^(3/2)])
                                                    /* MOS p. 75 */

SAir[4]: AirBi[$z] -> (-$z/3)^(1/2) (BesJ[-1/3,2/3 (-$z)^(3/2)] - \
                        BesJ[1/3,2/3 (-$z)^(3/2)])
                                                    /* MOS p. 75 */
```

XAng

```
                        /** Anger and Weber functions **/

SAng_:Ldist

SAng[1]: AngJ[$n,$z] -> Cos[Pi $n]/Sin[Pi $n] WebE[$n,$z] - WebE[-$n,$z]\
                        /Sin[Pi $n]
                                                    /* MOS p. 118 */

SAng[2]: WebE[$n,$z] -> AngJ[-$n,$z]/Sin[Pi $n] - Cos[Pi $n]/Sin[Pi $n]\
                        AngJ[$n,$z]
                                                    /* MOS p. 118 */

SAng[3]: AngJ[$n,$z] -> 2($n-1)$z^(-$n+1) AngJ[$n-1,$z] - 2/Pi/$z \
                        Sin[Pi($n-1)] - AngJ[$n-2]
                                                    /* MOS p. 118 */

SAng[4]: WebE[$n,$z] -> 2($n-1)/$z WebE[$n-1,$z] - 2/Pi/$z
                        (1 - Cos[Pi ($n - 1)]) - WebE[$n-2,$z]
                                                    /* MOS p. 118 */

SAng[5]: WebE[1/2,$z] -> AngJ[-1/2,$z]
                                                    /* MOS p. 119 */

SAng[6]: AngJ[-1/2,$z] -> WebE[1/2,$z]
                                                    /* MOS p. 119 */

SAng[7]: AngJ[-1/2,$z] -> (Pi $z/2)^(-1/2) (Cos[$z] (FreC[$z]\
                        + FreS[$z]) - Sin[$z] (FreC[$z] - FreS[$z]))
                                                    /* MOS p. 119 */

SAng[8]: WebE[1/2,$z] -> (Pi $z/2)^(-1/2) (Cos[$z] (FreC[$z]\
                        + FreS[$z]) - Sin[$z] (FreC[$z] - FreS[$z]))
```

```
                                                        /* MOS p. 119 */

SAng[9]: -WebE[-1/2,$z] -> (Pi $z/2)^(-1/2) (Cos[$z] (FreC[$z]\
                          - FreS[$z]) + Sin[$z] (FreC[$z] + FreS[$z]))
                                                        /* MOS p. 119 */

SAng[10]:         AngJ[1/2,$z] -> (Pi $z/2)^(-1/2) (Cos[$z] (FreC[$z]\
                          - FreS[$z]) + Sin[$z] (FreC[$z] + FreS[$z]))
                                                        /* MOS p. 119 */
```

XAny

```
        /** Test for any elements of list satisfying condition **/

/* Any[temp,list]
        tests whether any of the elements of list yield "true" on application
        of the template temp. */
        _Any[Nosmp] : {1,8}
        Any[$temp,$list] :: In[$1_=Rel[Ap[$temp,{$1}]],$list]

/*

#I[1]::  <XAny

#I[2]::  Any[Evenp,Ar[5,Prime]]

#O[2]:   1
*/
```

XArperm

```
                /** Generation of permutations **/

<XList8

/* Arperm[n,(spec:(all))]
        yields a list of the permuations of n elements
        which exhibit the symmetries spec. */
        /* Arperm[$n] :: Flat[Ar[Ar[$n,$n],List,Uneq],$n] */
        Arperm_:Tier
        Arperm[1]:{{1}}
        Arperm[$n_?Natp[$n]] :: Arperm[$n]: \
                Flat[Map[Ar[$n,Ins[$n,$%1,$%2]],Arperm[$n-1]],1]
        Arperm[$n,Cyclic] :: Ar[$n,Cyc[Ar[$n],$%1]]
        Arperm[$n,Even] :: Cat[Ar[$n!,Arperm[$n],Evenp]]
        Arperm[$n,Odd] :: Cat[Ar[$n!,Arperm[$n],Oddp]]

/*

#I[1]::  <XArperm

#I[2]::  Arperm[3]

#O[2]:   {{3,2,1},{2,3,1},{2,1,3},{3,1,2},{1,3,2},{1,2,3}}

#I[3]::  Arperm[3,Cyclic]
```

```
#O[3]:   {{2,3,1},{3,1,2},{1,2,3}}

#I[4]::  Arperm[3,Even]

#O[4]:   {{2,3,1},{3,1,2},{1,2,3}}

#I[5]::  Arperm[3,Odd]

#O[5]:   {{3,2,1},{2,1,3},{1,3,2}}

*/
```

XBase

```
        /** Conversion of integers in arbitrary number bases **/

/* To10[ccccc,n]
        converts the number ccccc from base n to base 10. */
        /* ccccc represents an integer whose digits are characters in
        the symbol name ccccc. The "digit" 10 is represented by a, 11 by b
        and so on. */
        To10[$s_:Symbp[$s],$b_:Natp[$b]] :: (Lc|[%|]; %|:Exp|[$s]; \
                        Sum[$b^(Len[%|]-%i)*%|[%i],%i,1,Len[%|]])

/* From10[x,n]
        converts the decimal integer x to a Base projection in base n. */
        From10[$n_:Natp[$n],$b_:Natp[$b-1]] :: (Lc|[%tot,%res,%i]; \
                For[%i:1; %res:$n, %res~=0, Inc[%i], %tot[%i]:Mod[%res,$b]; \
                %res:Gint[%res/$b]]; Imp|[Rev[%tot]])


/*
#I[1]::  <XBase

#I[2]::  From10[1452,2]

#O[2]:   "10110101100"

#I[3]::  To10[%,2]

#O[3]:   1452

#I[4]::  From10[%,16]

#O[4]:   "5ac"

#I[5]::  To10[%,16]

#O[5]:   1452

*/
```

XBer

```
                    /** Bernoulli polynomials **/

SBer_: Ldist

        Ber[0] : 1

        Ber[$n_?Natp[($n-1)/2],0] : 0

SBer[1]:        Ber[$n] -> -Sum[Comb[$n+1,k] Ber[k], k, 0, $n-1] / ($n + 1)

SBer[2]: Ber[$n,$x+$$y] --> Sum[Comb[$n,m] Ber[m,$x] $$y^($n-m),m,0,$n]
                                                    /* MOS p. 25 */

SBer[3]: Ber[$n_?Natp[$n],0] -> Ber[$n]

SBer[4]: Ber[$n_?Natp[$n],1/2] -> -(1-2^(1-$n))Ber[$n]

SBer[5]:        Ber[$n_?Natp[$n],1/4] -> (-1)^$n Ber[$n,3/4]

SBer[6]: Ber[$n_?Natp[$n],1/4] -> -2^(-$n) (1-2(1-$n)) Ber[$n]\
                    $n 4^(-$n) Eu![$n-1]

SBer[7]:        Ber[$n_?Natp[$n/2],5/6] -> Ber[$n,1/6]

SBer[8]: Ber[$n_?Natp[$n/2],1/6] -> 1/2 (1-2^(1-$n))(1-3^(1-$n))Ber[$n]

SBer[9]: Ber[$n_?Natp[$n],1-$x] -> (-1)^$n Ber[$n,$x]

SBer[9]:        Ber[$n_?Natp[$n],$x] -> (-1)^$n (Ber[$n,-$x]+$n (-$x)^($n-1))

SBer[10]:       Sum[$m^$n,$m,1,$N] -> 1/($n+1) (Ber[$n+1,$N+1] - Ber[$n+1])
                                                    /* MOS p. 26 */

SBer[11]:       Ber[$n_?Natp[$n],$x] -> Ber[$n,$x+1] - $n $x^($n-1)
                                                    /* MOS p. 26 */

SBer[12]:     Sum[Comb[$n_?Natp[$n],$m_?Natp[$m-1]] Ber[$m,$x],$m,0,$n-1] ->\
                    $n $x^($n-1)
                                                    /* MOS p. 26 */

SBer[13]:       Ber[$n_?Natp[$n],$x $m_?Natp[$m-1]] -> $m^($n-1)\
                    Sum[Ber[$n]($x + !/$m),!,0,$m-1]
                                                    /* MOS p. 26 */

                /* Bernoulli numbers */

SBer[14]:     Ber[$n,0] -> Ber[$n]

SBer[15]:       Ber[$n,1] -> Ber[$n]

SBer[16]:       Ber[$n_?Natp[$n/2-1/2]]:   0
                                                    /* MOS p. 27 */


SBer[17]:       Ber[$n_?Natp[$n/2]] --> 2(-1)^($n/2+1) $n!*\
                    Sum[(2 Pi m)^(-$n),m,1,Inf]
                                                    /* MOS p. 27 */

SBer[18]:       Ber[$n_?Natp[$n/2]] -> 2(-1)^($n/2+1)(2 Pi)^(-$n) $n!*\
                    Zeta[$n]
                                                    /* MOS p. 28 */

SBer[19]:       Ber[$n_?Natp[$n/2-1]] -> -$n Zeta[1-$n]
                                                    /* MOS p. 28 */
```

XBes1

```
            /**  Functional relations for Bessel functions  **/

SBes_Ldist

SBes[1,1]:        BesY[$n,$z] -> 1/Sin[Pi $n] (BesJ[$n,$z] Cos[Pi $n]-\
                        BesJ[-$n,$z])
                                                    /* MOS p. 66 */

SBes[1,2]:        BesH1[$n,$z] -> BesJ[$n,$z] + I BesY[$n,$z]
                                                    /* MOS p. 66 */

SBes[1,3]:        BesH1[$n,$z] -> 1/(I Sin[Pi $n]) (BesJ[-$n,$z] - \
                        BesJ[$n,$z] Exp[-I Pi $n])
                                                    /* MOS p. 66 */

SBes[1,4]:        BesH2[$n,$z] -> BesJ[$n,$z] - I BesY[$n,$z]
                                                    /* MOS p. 66 */

SBes[1,5]:        BesH2[$n,$z] -> 1/(I Sin[Pi $n]) (BesJ[$n,$z] Exp[I Pi $n] - \
                        BesJ[-$n,$z])
                                                    /* MOS p. 66 */

SBes[1,6]:        BesJ[$z,$z] -> BesJ[-$n,$z] Cos[-Pi $n] - BesY[-$n,$z]\
                        Sin[-Pi $n]
                                                    /* MOS p. 66 */

SBes[1,7]:        BesY[$n,$z] -> BesJ[-$n,$z] Sin[-Pi $n] + BesY[-$n,$z]\
                        Cos[-Pi $n]
                                                    /* MOS p. 66 */

SBes[1,8]:        BesH1[$n,$z] -> Exp[-I Pi $n] BesH1[-$n,$z]
                                                    /* MOS p. 66 */

SBes[1,9]:        BesH2[$n,$z] -> Exp[I Pi $n] BesH2[-$n,$z]
                                                    /* MOS p. 66 */

SBes[1,10]:       BesJ[$n,$z] -> Conj[BesJ[Conj[$n],Conj[$z]]]
                                                    /* MOS p. 66 */

SBes[1,11]:       BesY[$n,$z] -> Conj[BesY[Conj[$n],Conj[$z]]]
                                                    /* MOS p. 66 */

SBes[1,12]:       BesH1[$n,$z] -> Conj[BesH2[Conj[$n],Conj[$z]]]
                                                    /* MOS p. 66 */

SBes[1,13]:       BesH2[$n,$z] -> Conj[BesH1[Conj[$n],Conj[$z]]]
                                                    /* MOS p. 66 */
```

XBes2

```
        /** Recurrence relations for Bessel functions **/

SBes_Ldist

SBes[2,1]:      BesJ[$n,$z] -> $z/2/$n (BesJ[$n-1,$z] + BesJ[$n+1,$z])
                                                    /* MOS p. 67 */

SBes[2,2]:      BesY[$n,$z] -> $z/2/$n (BesY[$n-1,$z] + BesY[$n+1,$z])
                                                    /* MOS p. 67 */

SBes[2,3]:      BesH1[$n,$z] -> $z/2/$n (BesH1[$n-1,$z] + BesH1[$n+1,$z])
                                                    /* MOS p. 67 */

SBes[2,4]:      BesH2[$n,$z] -> $z/2/$n (BesH2[$n-1,$z] + BesH2[$n+1,$z])
                                                    /* MOS p. 67 */

SBes[2,5]:      BesJ[$n,$z] -> -BesJ[$n+2,$z] + 2($n+1)/$z BesJ[$n+1,$z]
                                                    /* MOS p. 67 */

SBes[2,6]:      BesY[$n,$z] -> -BesY[$n+2,$z] + 2($n+1)/$z BesY[$n+1,$z]
                                                    /* MOS p. 67 */

SBes[2,7]:      BesH1[$n,$z] -> -BesH1[$n+2,$z] + 2($n+1)/$z BesH1[$n+1,$z]
                                                    /* MOS p. 67 */

SBes[2,8]:      BesH2[$n,$z] -> -BesH2[$n+2,$z] + 2($n+1)/$z BesH2[$n+1,$z]
                                                    /* MOS p. 67 */

SBes[2,9]:      BesJ[$n,$z] -> -BesJ[$n-2,$z] + 2($n-1)/$z BesJ[$n-1,$z]
                                                    /* MOS p. 67 */

SBes[2,10]:     BesY[$n,$z] -> -BesY[$n-2,$z] + 2($n-1)/$z BesY[$n-1,$z]
                                                    /* MOS p. 67 */

SBes[2,11]:     BesH1[$n,$z] -> -BesH1[$n-2,$z] + 2($n-1)/$z BesH1[$n-1,$z]
                                                    /* MOS p. 67 */

SBes[2,12]:     BesH2[$n,$z] -> -BesH2[$n-2,$z] + 2($n-1)/$z BesH2[$n-1,$z]
                                                    /* MOS p. 67 */

SBes[2,13]:     BesI[$n,$z] -> $z/2/$n (BesI[$n-1,$z] - BesI[$n+1,$z])
                                                    /* MOS p. 67 */

SBes[2,14]:     BesK[$n,$z] -> $z/2/$n (BesK[$n-1,$z] - BesK[$n+1,$z])
                                                    /* MOS p. 67 */

SBes[2,15]:     BesK[$n,$z] -> BesI[$n+2,$z] + 2($n+1)/$z BesI[$n+1,$z]
                                                    /* MOS p. 67 */

SBes[2,16]:     BesK[$n,$z] -> BesK[$n+2,$z] - 2($n+1)/$z BesK[$n+1,$z]
                                                    /* MOS p. 67 */

SBes[2,17]:     BesI[$n,$z] -> BesI[$n-2,$z] - 2($n-1)/$z BesI[$n-1,$z]
                                                    /* MOS p. 67 */

SBes[2,18]:     BesI[$n,$z] -> BesK[$n-2,$z] + 2($n-1)/$z BesK[$n-1,$z]
                                                    /* MOS p. 67 */
```

XBes3

```
                /** Bessel functions of integer order **/

SBes_Ldist

SBes[3,1]:      BesJ[$n_?Intp[$n],$z] -> (-1)^(-$n) BesJ[-$n,$z]
                                                    /* MOS p. 70 */

SBes[3,2]:      BesY[$n_?Intp[$n],$z] -> (-1)^(-$n) BesY[-$n,$z]
                                                    /* MOS p. 70 */

SBes[3,3]:      BesI[$n_?Intp[$n],$z] -> (-1)^(-$n) BesI[-$n,$z]
                                                    /* MOS p. 70 */

SBes[3,4]:      BesJ[$n_?Evenp[$n],$z] -> (-1)^$n $n Sum[($n + i - 1)!/\
                  ($n - i)!*(-1)^i ($z/2)^(-i)/i!*BesJ[i,$z],i,0,$n]
                                                    /* MOS p. 71 */

SBes[3,5]:      BesY[$n_?Evenp[$n],$z] -> (-1)^($n/2) ($n/2) Sum[($n/2+i-1)!/\
                  ($n/2 - i)!*(-1)^i ($z/2)^(-i)/i!*BesY[i,$z],i,0,$n/2
                                                    /* MOS p. 71 */

SBes[3,6]:      BesI[$n_?Evenp[$n],$z] -> $n/2 Sum[($n/2 + i - 1)!/\
                  ($n/2 - i)!*(-1)^i ($z/2)^(-i)/i!*BesI[i,$z],i,0,$n/2
                                                    /* MOS p. 71 */

SBes[3,7]:      BesK[$n_?Evenp[$n],$z] -> $n/2 Sum[($n/2 + i - 1)!/\
                  ($n/2 - i)!*($z/2)^(-i)/i!*BesK[i,$z],i,0,$n/2]
                                                    /* MOS p. 71 */

Epsn[$x]: 1 + ($x>0)

SBes[3,8]:      BesJ[$n_?Intp[$n],$z] -> ($z/2)^$n $n!*Sum[Epsn[i]\
                  1/(($n + i)!*($n - i)!) BesJ[2i,$z],i,0,$n]
                                                    /* MOS p. 71 */

SBes[3,9]:      BesY[$n_?Intp[$n],$z] -> ($z/2)^$n $n!*Sum[Epsn[i]\
                  1/(($n + i)!*($n - i)!) BesY[2i,$z],i,0,$n]
                                                    /* MOS p. 71 */

SBes[3,10]:     BesI[$n_?Intp[$n],$z] -> ($z/2)^$n $n!*Sum[(-1)^i Epsn[i]\
                  1/(($n + i)!*($n - i)!) BesI[2i,$z],i,0,$n]
                                                    /* MOS p. 71 */

SBes[3,11]:     BesK[$n_?Intp[$n],$z] -> (-1)^$n($z/2)^$n $n!*Sum[(-1)^i\
                  Epsn[i] 1/(($n + i)!*($n - i)!) BesK[2i,$z],i,0,$n]
                                                    /* MOS p. 71 */
```

XBes4

```
                /**  Bessel functions of half odd integer order  **/

    SBes_Ldist

    SBes[4,5]:      BesJ[$n_?Natp[-$n-1/2],$z] -> (-1)^(-$n+1/2) BesY[-$n,$z]
                                                          /* MOS p. 72 */

    SBes[4,7]:       BesY[$n_?Natp[-$n-1/2],$z] -> (-1)^(-$n-1/2) BesJ[-$n,$z]
                                                          /* MOS p. 72 */

SBes[4,8]:        BesH1[$n_?Natp[-$n-1/2],$z] -> I (-1)^(-$n-1/2)      BesH1[-$n,$z]
                                                          /* MOS p. 72 */

    SBes[4,9]:       BesH2[$n_?Natp[-$n-1/2],$z] -> -I (-1)^(-$n-1/2) BesH2[-$n,$z]
                                                          /* MOS p. 72 */

  SBes[4,10]:       BesI[$n_?Natp[-$n-1/2],$z] -> (-1)^(-$n-1/2) 2/Pi BesK[-$n,$z]+\
                          BesI[-$n,$z]
                                                          /* MOS p. 72 */

    SBes[4,11]:       Besj[$n_?Intp[$n],$z] -> (Pi/2/$z)^(1/2) BesJ[$n+1/2,$z]
                                                          /* MOS p. 73 */

    SBes[4,12]:       Besh1[$n_?Intp[$n],$z] -> (Pi/2/$z)^(1/2) BesH1[$n+1/2,$z]
                                                          /* MOS p. 73 */

    SBes[4,13]:       Besh2[$n_?Intp[$n],$z] -> (Pi/2/$z)^(1/2) BesH2[$n+1/2,$z]
                                                          /* MOS p. 73 */
```

XBes5

```
          /**  Special cases for half odd integer order Bessel functions  **/

    SBes_Ldist

    SBes[5,1,1]:      BesJ[1/2,$z] -> BesY[-1/2,$z]
    SBes[5,1,2] :   BesY[-1/2,$z] -> BesJ[1/2,$z]
                                                          /* MOS p. 73 */

    SBes[5,2,1]:      BesJ[1/2,$z] -> (Pi/2 $z)^(-1/2) Sin[$z]
    SBes[5,2,2]:      BeY[-1/2,$z] -> Sin[$z] / Sqrt[Pi $z/2]
                                                          /* MOS p. 73 */

    SBes[5,3,1]:      BesY[1/2,$z] -> -BesJ[-1/2,$z]
    SBes[5,3,2]:      BesJ[-1/2,$z] -> -BesY[1/2,$z]
                                                          /* MOS p. 73 */

    SBes[5,4,1]:      BesY[1/2,$z] -> -(Pi/2 $z)^(-1/2) Cos[$z]
    SBes[5,4,2] :     BesJ[-1/2,$z] -> Cos[$z] / Sqrt[Pi $z/2]
                                                          /* MOS p. 73 */

    SBes[5,5,1]:      BesH1[1/2,$z] -> -I BesH1[-1/2,$z]
    SBes[5,5,2] :     BesH1[-1/2,$z] -> I BesH1[1/2,$z]
                                                          /* MOS p. 73 */

    SBes[5,6,1]:      BesH1[1/2,$z] -> -I (Pi/2 $z)^(-1/2) Cos[$z] Exp[I $z]
    SBes[5,6,2] :     BesH1[-1/2,$z] -> Cos[$z] Exp[I $z] / Sqrt[Pi $z/2]
                                                          /* MOS p. 73 */

    SBes[5,7,1]:      BesH2[1/2,$z] -> I BesH2[-1/2,$z]
    SBes[5,7,2] :     BesH2[-1/2,$z] -> -I BesH2[1/2,$z]
                                                          /* MOS p. 73 */
```

```
SBes[5,8,1]:      BesH2[1/2,$z] -> I (Pi/2 $z)^(-1/2) Cos[$z] Exp[-I $z]
SBes[5,8,2] :     BesH2[-1/2,$z] -> Cos[$z] Exp[-I $z] / Sqrt[Pi $z/2]
                                                    /* MOS p. 73 */

SBes[5,9]:        BesI[1/2,$z] -> (Pi/2 $z)^(-1/2) Sinh[$z]
                                                    /* MOS p. 73 */

SBes[5,10]:       BesI[-1/2,$z] -> (Pi/2 $z)^(-1/2) Cosh[$z]
                                                    /* MOS p. 73 */

SBes[5,11]:       BesK[1/2,$z] -> BesK[-1/2,$z]
                                                    /* MOS p. 73 */

SBes[5,12]:       BesK[-1/2,$z] -> BesK[1/2,$z]
                                                    /* MOS p. 73 */

SBes[5,13]:       BesK[1/2,$z] -> (Pi/2/$z)^(1/2) Exp[-$z]
                                                    /* MOS p. 73 */

SBes[5,14]:       BesK[-1/2,$z] -> (Pi/2/$z)^(1/2) Exp[-$z]
                                                    /* MOS p. 73 */

                  /* duplication formulae */

SBes[5,15] : BesJ[$m_?Natp[$m+1/2],$z] -> -Sqrt[Pi] ($m-1/2)! ($z/2)^$m \
               Sum[Comb[2$m-2k,k] BesJ[$m-k,$z/2] BesY[$m-k,$z/2],k,0,$m-1/2]

SBes[5,16] : BesY[$m_?Natp[$m+1/2],$z] -> Sqrt[Pi]/2 ($m-1/2)! ($z/2)^$m \
               Sum[Comb[2$m-2k,k] (BesJ[$m-k,$z/2]^2-BesY[$m-k,$z/2]^2), \
                  k,0,$m-1/2]

SBes[5,17] : BesK[$m_?Natp[$m+1/2],$z] -> ($m-1/2)!/Sqrt[Pi] ($z/2)^$m \
               Sum[(-1)^k Comb[2$m-2k,k] BesK[$m-k,$z/2]^2,k,0,$m-1/2]

SBes[5,18] : BesI[$m_?Natp[$m+1/2],$z] -> Sqrt[Pi] ($m-1/2)! ($z/2)^$m \
               Sum[(-1)^($m-1/2-k) Comb[2$m-2k,k] BesI[$m-k,$z/2] \
                   BesI[k-$m,$z/2], k,0,$m-1/2]

                                                    /* MOS 3.3.2 p74 */
```

XBeta

```
                  /** Euler beta function **/

SBeta_?Ldist

SBeta[1]:    Beta[$x,$y] -> ($y-1)/$x Beta[$x+1,$y-1]
                                                    /* MOS p. 7 */

SBeta[2]:    Beta[$x,$y] -> ($y-1)/($x+$y-1)Beta[$x,$y-1]
                                                    /* MOS p. 7 */

SBeta[3]:    Beta[$x,$y] -> ($x+$y)/$y Beta[$x,$y+1]
                                                    /* MOS p. 7 */

SBeta[4]:    Beta[$x,$y] Beta[$x+$y,$z] -> Beta[$y,$z] Beta[$y+$z,$x]
                                                    /* MOS p. 7 */

SBeta[5]:    Beta[$x,$y] Beta[$x+$y,$z] -> Gamma[$x] Gamma[$z] Gamma[$y]\
                  /  Gamma[$x+$y+$z]
                                                    /* MOS p. 7 */

SBeta[6] :   Beta[$x,$y] -> ($y-1)/($x+$y-1) Beta[$x,$y-1]
```

XBit

```
                        /** Bitwise operations **/

/* Bits[n]
        yields a list of binary bits corresponding to the integer n. */
        Bits[$n_:Natp[$n]] :: (Lcl[%tot,%res,%i]; %i:1;%res:$n; \
                Loop[%res~=0,%tot[%i]:Mod[%res,2];%res:Gint[%res/2];Inc[%i]]\
                Rev[%tot])

/* Intbit[list]
        finds the integer corresponding to a list of bits. */
        Intbit[$list_:Contp[$list]] :: \
                Sum[2^(Len[$list]-%i) $list[%i],%i,1,Len[$list]]

/* Bitand[n,m]
        yields the bitwise conjunction of n and m. */
        Bitand_:Comm
        Bitand[$n_:Natp[$n],$m_:(Natp[$m] & $m>$n)] :: \
                (Lcl[%n,%m]; %n:Bits[$n]; %m:Bits[$m]; \
                Intbit[Ldist[%n & Ar[Len[%n],%m]]])

/* Bitor[n,m]
        yields the bitwise disjunction of n and m. */
        Bitor_:Comm
        Bitor[$n_:Natp[$n],$m_:Natp[$m]] :: Intbit[Ldist[Bits[$n] | Bits[$m]]]
```

XBox

```
/* Box[x,y]
        represents a unit box centred at the point x,y. */
        Box[{$x,$y}] :: Line[{Pt[{$x-0.5,$y-0.5}],Pt[{$x+0.5,$y-0.5}], \
                        Pt[{$x+0.5,$y+0.5}],Pt[{$x-0.5,$y+0.5}], \
                                Pt[{$x-0.5,$y-0.5}]}]

/* Circle[{x,y}]
        represents a unit circle centred at the point x,y. */
        Circle[{$x,$y}] :: {Curve[Ar[20,Pt[{$x,$y}+{Sin[2Pi $1/20], \
                                Cos[2Pi $1/20]}]]]}
```

XChar

```
                    /** Character manipulation **/

/* Upperp[str]
        yields 1 if the first character of the string str
        is an upper case letter, and 0 otherwise. */
        Upperp[$str] :: (!Exp|[A][1]) <= Exp|[$str][1] <= (!Exp|[Z][1])

/* Lowerp[str]
        yields 1 if the first character of the string str
        is a lower case letter, and 0 otherwise. */
        Lowerp[$str] :: (!Exp|[a][1]) <= Exp|[$str][1] < (!Exp|[z][1])

/* Tolower[str]
        converts all upper case letters in str to lower case. */
        Tolower[$str] :: Imp|[Map[$1-(!Exp|[A][1]-Exp|[a][1]),Exp|[$str],1, \
                        (!Exp|[A][1]) <= $2 <= (!Exp|[Z][1])]]


/*
#I[1]::  <XChar

#I[2]::  t1:"a message"

#O[2]:   "a message"

#I[3]::  t2:"A MESSAGE"

#O[3]:   "A MESSAGE"

#I[4]::  Lowerp[t1]

#O[4]:   1

#I[5]::  Lowerp[t2]

#O[5]:   0

#I[6]::  Tolower[t2]

#O[6]:   "a message"

*/
```

XChe

```
                    /** Chebychef polynomials **/

SChe_:Ldist

SChe[1]:   CheT[$m,$x] -> 2$x CheT[$m-1,$x]-CheT[$m-2,$x]

SChe[2]:   CheT[$m,$x] -> (CheT[$m+1,$x]+CheT[$m-1,$x])/(2$x)

SChe[3]:   CheT[$m,$x] -> 2$x CheT[$m+1,$x]-CheT[$m+2,$x]

SChe[4]:   CheU[$m,$x] -> 2$x CheU[$m-1,$x]-CheU[$m-2,$x]

SChe[5]:   CheU[$m,$x] -> (CheU[$m+1,$x]+CheU[$m-1,$x])/(2$x)

SChe[6]:   CheU[$m,$x] -> 2$x CheU[$m+1,$x]-CheU[$m+2,$x]
```

XChg

```
                    /** Confluent hypergeometric function **/

SCh_Ldist

SCh[1]:   Chg[$a,$b,$z] -> Exp[$z]Chg[$b-$a,$b,-$z]

SCh[2]:   Chg[1,2,$z] -> 2 Exp[$z/2] Sinh[$z/2] / $z

SCh[3]:   Chg[1,2,$z] -> (Exp[$z] - 1) / $z
                                                        /* AS 13.6.14 */

SCh[4]:   Chg[$n,$n,$z] -> Exp[$z]
                                                        /* AS 13.6.12 */

SCh[5]:   Chg[$n,2$n,$z] -> Gamma[$n+1/2]Exp[$z/2](-I $z/4)^(-$n+1/2)
                            BesJ[$n-1/2,-I $z/2]
                                                        /* AS 13.6.1 */

SCh[6]:   Chg[$n,2$n,$z] -> Gamma[$n+1/2]Exp[$z/2](-I $z/4)^(-$n+1/2)\
                             (Sin[Pi $n]BesJ[-$n+1/2,-I $z/2] \
                             Cos[Pi $n]BesY[-$n+1/2,-I $z/2])
                                                        /* AS 13.6.2 */

SCh[7]:   Chg[$n,2$n,$z] -> Gamma[$n+1/2]Exp[$z/2] ($z/4)^(-$n+1/2)\
                            BesI[$n-1/2,$z/2]
                                                        /* AS 13.6.3 */

SCh[8]:   Chg[$a,2$a,$z] -> Gamma[$a+1/2] Exp[-I^1.5 Pi $z]\
                  (-Sqrt[I] Pi $z/4)^(1/2-$a)Kelbe[$a-1/2,I^1.5 $z/2]
                                                        /* AS 13.6.7 */

SCh[9]:   Chg[$n,1/2,$z] -> 2^(-1/2)Exp[$z/2]WebE[-2$n,2^0.5 $z^(1/2)]
                                                        /* AS 13.6.15 */

SCh[10] :  Chg[$a,1/2,$y] -> (-2)^(-$a)(-$a)!*Her[-2$a,Sqrt[2$y]]/(-2$a)!
                                                        /* AS 13.6.17 */

SCh[11]:   Chg[$n,3/2,$z] -> (-$n)!/(-2$n+1)!*(-1/2)^$n/\
                            (2$z)^0.5 Her[-2 $n+1,(2$z)^0.5]
                                                        /* AS 13.6.18 */

SCh[12]:   Chg[1/2,3/2,$z] -> Sqrt[-Pi/$z] Erf[Sqrt[-$z]]/2
                                                        /* AS 13.6.19 */

SCh[13]:   Chg[$m,$n,$z] -> ($n-1)!*$z^($m-$n)\
                            Exp[$z]Tor[2$m-1,$n-1,$z^0.5] / Gamma[$m]
                                                        /* AS 13.6.28 */

SCh[14]:   Chg[$a,$c,$z] -> Exp[-$z/2]$z^(-$c/2)WhiM[$c/2-$a,$c/2-1/2,$z]
                                                        /* MOS p.304 */
```

XCon

```
                    /** Tensor contraction **/

/* Con[list,ni,nj]
        forms the contraction of the tensor list over its
        nith and njth indices. */
        Con[$list,$ni_:Natp[$ni],$nj_:Natp[$nj]] :: \
                (Lc[%t]; %t:Trans[Trans[$list,{1,$ni}],{2,$nj}]; \
                        Ap[Plus,Ar[Len[%t],%t[$1,$1]]])

/*

#I[1]::  <<XCon

#I[2]::  w3:Ar[{2,2,2},f]

#O[2]:   {{{f[1,1,1],f[1,1,2]},{f[1,2,1],f[1,2,2]}},
                {{f[2,1,1],f[2,1,2]},{f[2,2,1],f[2,2,2]}}}

#I[3]::  Con[%,1,2]

#O[3]:   {f[1,1,1] + f[2,2,1],f[1,1,2] + f[2,2,2]}

#I[4]::  Con[w3,2,3]

#O[4]:   {f[1,1,1] + f[1,2,2],f[2,1,1] + f[2,2,2]}

#I[5]::  w4:Ar[{2,2,2,2},f];

#I[6]::  Con[w4,2,4]

#O[6]:   {{f[1,1,1,1] + f[1,2,1,2],f[2,1,1,1] + f[2,2,1,2]},
                {f[1,1,2,1] + f[1,2,2,2],f[2,1,2,1] + f[2,2,2,2]}}
*/
```

XContig

```
                    /** Make list contiguous **/

/* Contig[list,{n1,n2,..},elem]
        renders list contiguous with ni entries at level i by
        inserting elem where necessary. */
        Contig[$list_:Listp[$list],$n_:Listp[$list],$elem] :: \
                Contig0[$list,$n,$elem,1]
        Contig0[$list,$n,$elem,$lev_:$lev>Len[$n]] : $list
        Contig0[$list_:~Listp[$list],$n,$elem,$lev] : $list
        Contig0[$list_:Listp[$list],$n,$elem,$lev] :: \
                Ar[$n[$lev],If[P[Proj[$list[$1],{0}]=Proj], \
                        $elem,Contig0[$list[$1],$n,$elem,$lev+1]]]

/*
#I[1]::  <<XContig

#I[2]::  t:{[3]:a,[2]:b}

#O[2]:   {[3]: a, [2]: b}

#I[3]::  Contig[t,{4},0]

#O[3]:   {0,b,a,0}
```

```
#I[4]::  Contig[t,{8},x]

#O[4]:   {x,b,a,x,x,x,x,x}

#I[5]::  Ar[{{2,3},{3,4}},f]

#O[5]:   {[2]: {[3]: f[2,3], [4]: f[2,4]}, [3]: {[3]: f[3,3], [4]: f[3,4]}}

#I[6]::  Contig[%,{4,4},0]

#O[6]:   {0,{0,0,f[2,3],f[2,4]},{0,0,f[3,3],f[3,4]},0}
*/
```

XD

```
                /** Derivatives **/

D[Ber[$n,$z],{$z,$m,$x}]    :: $n!/($n-$m)!*Ber[$n-$m,$x]

D[Beta[$x,$y,1],{$x,1,$z}] :: Beta[$z,$y,1] (Psi[$z] - Psi[$z+$y])
D[Beta[$x,$y,1],{$y,1,$z}] :: Beta[$x,$z,1] (Psi[$z] - Psi[$x+$z])

D[CheT[$n,$z],{$z,1,$x}]    :: $n CheU[$n-1,$x]
D[CheT[$n,$z],{$z,$m,$x}]   :: 2^($m-1) Gamma[$m] $n Geg[$n-$m,$m,$x]

D[CheU[$n,$z],{$z,$m,$x}]   :: 2^$m $m!*Geg[$n-$m,$m+1,$x]

D[Chg[$a,$c,$z],{$z,$m,$x}]          :: Poch[$a,$m] Chg[$a+$m,$c+$m,$x] / Poch[$c,$m]

D[Coshi[$z],{$z,1,$x}]               :: Cosh[$x]/$x
D[Cosi[$z],{$z,1,$x}]                :: Cos[$x]/$x

D[Ei[$z],{$z,1,$x}]                  :: Exp[$x]/$x

D[EllK[$k,$t],{$t,1,$u}]    :: (1-$k^2 Sin[$u]^2) ^ (-1/2)
D[EllE[$k,$t],{$t,1,$u}]    :: Sqrt[1-$k Sin[$u]^2]

D[Erf[$z],{$z,1,$x}]                 :: 2 Exp[-$x^2] / Sqrt[Pi]
D[Erf[$z],{$z,$m,$x}]                :: -2 (-1)^$m Exp[-$x^2] Her[$m-1,$x]

D[Eui[$n,$z],{$z,$m,$x}]    :: $n!/($n-$m)!*Eui[$n-$m,$x]

D[Expi[1,$z],{$z,$m,$x}]    :: (-1)^$m Exp[-$x] Chg[1,1+$m,$x]
D[Expi[$n,$z],{$z,1,$x}]    :: -Expi[$n-1,$x]

D[FreC[$z],{$z,1,$x}]                :: Cos[Pi $x^2 / 2]
D[FreS[$z],{$z,1,$x}]                :: Sin[Pi $x^2 / 2]

D[Gamma[$z],{$z,1,$x}]               :: Gamma[$x] Psi[$x,1]
D[Gamma[$z,$a],{$z,1,$x}]   :: -$a^($x-1) Exp[-$a]

D[Geg[$n,$l,$z],{$z,$m,$x}]          :: 2^$m Poch[$l,$m] Geg[$n-$m,$l+$m,$x]

D[Her[$n,$z],{$z,$m,$x}]    :: 2^$m $n!/($n-$m)!*Her[$n-$m,$x]

D[Hg[$a,$b,$c,$z],{$z,$m,$x}]        ::\
        Poch[$a,$m] Poch[$b,$m] Hg[$a+$m,$b+$m,$c+$m,$x] / Poch[$c,$m]

D[JacP[$n,$a,$b,$x],{$z,$m,$x}]      :: \
        Poch[$a+$b+$c+1,$m] JacP[$n-$m,$a+$m,$b+$m,$x] / 2^$m

D[JacCd[$z,$m],{$z,1,$x}]   :: ($m-1) JacSd[$x,$m] JacNd[$x,$m]
D[JacCn[$z,$m],{$z,1,$x}]   :: -JacSn[$x,$m] JacDn[$x,$m]
D[JacCs[$z,$m],{$z,1,$x}]   :: -JacNs[$x,$m] JacDs[$x,$m]
D[JacDc[$z,$m],{$z,1,$x}]   :: (1-$m) JacSc[$x,$m] JacNc[$x,$m]
```

```
D[JacDn[$z,$m],{$z,1,$x}]  :: -$m JacSn[$x,$m] JacCn[$x,$m]
D[JacDs[$z,$m],{$z,1,$x}]  :: -JacCs[$x,$m] JacNs[$x,$m]
D[JacNc[$z,$m],{$z,1,$x}]  :: JacSc[$x,$m] JacDc[$x,$m]
D[JacNd[$z,$m],{$z,1,$x}]  :: $m JacSd[$x,$m] JacCd[$x,$m]
D[JacNs[$z,$m],{$z,1,$x}]  :: -JacDs[$x,$m] JacCs[$x,$m]
D[JacSc[$z,$m],{$z,1,$x}]  :: JacDc[$x,$m] JacNc[$x,$m]
D[JacSd[$z,$m],{$z,1,$x}]  :: JacCd[$x,$m] JacNd[$x,$m]
D[JacSn[$z,$m],{$z,1,$x}]  :: JacCn[$x,$m] JacDn[$x,$m]

D[JacZ[$z,$m],{$z,1,$x}]   :: JacDn[$x,$m]^2

D[KumU[$a,$c,$z],{$z,$m,$x}]        :: (-1)^$m Poch[$a,$m] KumU[$a+$m,$c+$m,$x]

D[Lag[$n,$a,$z],{$z,1,$x}] :: -Lag[1-$n,1+$a,$x]

D[LegP[$n,$z],{$z,1,$x}]   :: $n (LegP[$n-1,$x]-$x LegP[$n,$x]) / (1-$x^2)

D[Li[$n,$z],{$z,1,$x}]              :: Li[$n-1,$x]/$x

D[Lob[$z],{$z,1,$x}]                :: Log[Sec[$x]]

D[Logi[$z],{$z,1,$x}]               :: 1/Log[$x]

D[Psi[$z],{$z,1,$x}]                :: Psi[$x,2]
D[Psi[$z,$n],{$z,1,$x}]    :: Psi[$x,$n+1]

D[Sinhi[$z],{$z,1,$x}]              :: Sinh[$x]/$x
D[Sini[$z],{$z,1,$x}]               :: Sin[$x]/$x

D[Zeta[$z,$a],{$a,1,$b}]   :: -$z Zeta[$z+1,$b]
```

XDSol

```
        /** Series solution of differential equations **/

/* DSol[eqn,y,x,ord,{y[0],y'[0],...}]
        gives a series solution to the ordinary differential equation
        eqn with dependent variable y and independent variable x
        and specified boundary conditions, accurate to order x^ord. */
        DSol[$eqn1=$eqn2,$f,$x,$ord,$init] :: \
        (Lc[%a,%eqn,%list,%f]; %f:Sum[%a[i] $x^i,i,0,$ord];\
        %eqn:S[Ex[S[$eqn1-$eqn2,$f->%f]],$x^$i_=($i>$ord)->0];\
        %list:Union[Cat[Ar[$ord-1,Coef[$x^$1,%eqn]],{S[%eqn,$x->0]},\
                Ar[Len[$init],D[%f,{$x,$1-1,0}]-$init[$1]]]];\
        %list:Map[$2=0,%list];\
        %list:Sol[%list,Ar[{{0,$ord}},%a[$1]]];\
        S[%f,%list])

/*

#I[1]::  <XDSol

#I[2]::  eq:Dt[y,x]+a y=0

#O[2]:   Dt[y,x] + a y = 0

#I[3]::  DSol[%,y,x,2,{1}]

                        2  2
                       a  x
#O[3]:   1 - a x + -----
                       2

*/
```

XDap

```
                    /** Directional application **/

/* Dap[f,list,n]
        applies the template f to "columns" of elements at level
        n in list. */
        Dap[$f,$list_?Listp[$list],$n_?(Natp[$n] & $n>1)] :: \
                Trans[Map[Ap[$f,$%1],Trans[$list,$n]],$n-1]
        Dap[$f,$list_?Listp[$list],1] :: Ap[$f,$list]
```

XDiff

```
                          /** Finite differences **/

/* Diff[f,x,x0,n]
        yields the nth forward finite difference of f with respect to x
        at the point x=x0. */
        Diff[$f,$x,$x0,$n] :: Sum[S[$f,$x->$x0+$n-%r] \
                        (-1)^%r Comb[$n,%r],%r,0,$n]

/*
#I[1]::  <XDiff

#I[2]::  Diff[f[x],x,0,3]

#O[2]:   -f[0] + 3f[1] - 3f[2] + f[3]

#I[3]::  Ar[5,Ex[Diff[(x+a)^3,x,1,$1]]]

                          2
#O[3]:   {7 + 9a + 3 a ,12 + 6a,6,0,0}

*/
```

XDig

```
                          /** Digit manipulation **/

/* Dig[n,i]
        yields the coefficient of 10^i in the number n. */
        Dig[$n_?Numbp[$n],$i_?Intp[$i]] :: Mod[Gint[$n/10^$i],10]

/* LDig[n]
        yields a list of the digits in the integer n. */
        LDig[$n_?Natp[$n]] :: Expl[$n]

/* NMake[list]
        converts a list of digits to an integer. */
        NMake[$list] :: Sum[10^(Len[$list]-%i) $list[%i],%i,1,Len[$list]]

/*
#I[1]::  <XDig

#I[2]::  Dig[456123.321,4]
```

```
#O[2]:   5
#I[3]::  Dig[114.88425,-3]
#O[3]:   4
#I[4]::  LDig[14671280]
#O[4]:   {1,4,6,7,1,2,8}
#I[5]::  NMake[%]
#O[5]:   1467128
*/
```

XDim

```
                    /** Dimensional analysis **/
/* Fundamental dimensions:
        length
        mass
        time
        current
        temperature
        (luminous) intensity
        amount (of substance)      */
/* Derived dimensions */
SDim[1] :          area -> length^2
SDim[2] :          volume -> length^3
SDim[3] :          frequency -> time^-1
SDim[4] :          density -> mass/volume
SDim[5] :          concentration -> amount/volume
SDim[6] :          velocity -> length/time
SDim[7] :          acceleration -> length/time^2
SDim[8] :          force -> mass length time^-2
SDim[9] :          pressure -> force/area
SDim[10] :         stress -> force/area
SDim[11] :         viscosity -> pressure time          /* dynamic viscosity */
SDim[12] :         energy -> force length
SDim[13] :         work -> force length
SDim[14] :         heat -> force length
SDim[15] :         power -> energy/time
SDim[16] :         charge -> current time
SDim[17] :         voltage -> power/current
SDim[18] :         emf -> voltage
```

```
SDim[19] :        field -> voltage/length    /* electric field strength */

SDim[20] :        resistance -> voltage/current

SDim[21] :        conductance -> resistance^-1

SDim[22] :        capacitance -> charge/voltage

SDim[23] :        flux -> voltage time                  /* magnetic flux */

SDim[24] :        inductance -> resistance time

SDim[25] :        dose -> energy/mass

SDim[26] :        activity -> time^-1

SDim[27] :        illuminance -> intensity steradian/area

SDim[28] :        irradiance -> power/area

SDim[29] :        entropy -> energy/temperature

SDim[30] :        conductivity -> power temperature/length
                                        /* thermal conductivity */
```

XDios

```
          /**  Solution of Dicphantine equations  **/

/* PDios[eqn,{n1,|n1max,(n1min:0),(n1step:1)|},...]
          tests all specified values for the ni, printing those
          sets found to satisfy the equation or condition eqn. */
          PDios[$eqn,$$r] :: (Lcl[%v]; %v:Ar[Len[%v:List[$$r]],%v[$1,1]]; \
                    Flat[Ar[Ar[Len[%v],List[$$r][$1,2]],Pr[$$1];List[$$1], \
                    P[N[S[$eqn,Ldist[%v->List[$$2]]]]],Len[%v]-1])

/*
#I[1]::   <XDios

#I[2]::   PDios[x^2+y^2=z^2,{x,10},{y,10},{z,10}]

3         4         5
4         3         5
6         8         10
8         6         10

#O[2]:    {{3,4,5},{4,3,5},{6,8,10},{8,6,10}}


*/
```

XEul

```
                    /** Euler polynomials and numbers **/

SEul := Ldist

SEul[1]: Ber[$n_:Natp[$n],$x] -> 2^-$n Sum[Comb[$n,m] Ber[$n-m] Eul[m,2 $x],\
                    m,0,$n]
                                                    /* MOS p. 29 */

SEul[2]: Eul[$n_:Natp[$n],$x] -> 2^($n+1)/($n+1)(Ber[$n+1,($x+1)/2]-\
                    Ber[$n+1,$x/2])
                                                    /* MOS p. 29 */

SEul[3]: Eul[$n_:Natp[$n],$x] -> 2/($n+1)(Ber[$n+1,$x]-2^($n+1)Ber[$n+1,\
                    $x/2])
                                                    /* MOS p. 29 */

SEul[4]: Eul[$n_:Natp[$n],$x] -> 2/Comb[$n+1,2] Sum[Comb[$n+2,m] (2^($n-m+2)\
                    -1) Ber[m,$x] Ber[$n-m+2],m,0,$n]
                                                    /* MOS p. 29 */

SEul[5]: Eul[$n,$x+$y] -> Sum[Comb[$n,m] Eul[m,$x] $y^($n-m),m,0,$n]
                                                    /* MOS p. 29 */

SEul[6]: Eul[$n,$x] -> Sum[Comb[$n,m] 2^(-m)($x-1/2)^($n-m) Eul[m],m,0,$n]
                                                    /* MOS p. 29 */


            /* Euler polynomials and numbers, Special values */


        Eul[$n_:Oddp[$n]]: 0
                                                    /* MOS p. 29 */

SEul[7]:          Eul[$n_:Natp[$n],0] -> (-1)^$n Eul[$n,1]
                                                    /* MOS p. 29 */

SEul[8]: Eul[$n_:Natp[$n-1],0] -> 2/($n+1)(1-2^($n+1)) Ber[$n+1]
                                                    /* MOS p. 29 */

SEul[9]: Eul[$n_:Natp[$n],1/2] -> 2^(-$n) Eul[$n]
                                                    /* MOS p. 30 */

SEul[10]:         Eul[$n_:Natp[$n/2+1/2],1/3] -> -Eul[$n,2/3]
                                                    /* MOS p. 30 */

SEul[11]:         Eul[$n_:Natp[$n/2+1/2],1/3] -> -1/(2 $n) (1-3^(1-2 $n))\
                    (2^2 $n - 1) Ber[2 $n]
                                                    /* MOS p. 30 */

SEul[12]:         Eul[$n_:Natp[$n/2+1/2],2/3] -> 1/(2 $n)(1-3^(1-2 $n))\
                    (2^2 $n - 1) Ber[2 $n]
                                                    /* MOS p. 39 */

SEul[13]:         Eul[$n_:Natp[$n],$x] -> (-1)^$n Eul(1-$x)
                                                    /* MOS p. 30 */

SEul[14]:        Eul[$n_:Natp[$n],$x] -> (-1)^($n+1) (Eul[$n,-$x] - (-2 $x)^$n)
                                                    /* MOS p. 30 */

SEul[15]:         Eul[$n_:Natp[$n+1],$x] -> (-1)^$x Eul[$x,0] +\
                    Sum[(-1)^($x-m-1) m^$n,m,1,$x-1]
                                                    /* MOS p. 30 */

SEul[16]:         Eul[$n,$x] -> 2 $x^$n - Eul[$n,$x+1]
                                                    /* MOS p. 30 */

SEul[17]:         Eul[$n,$x] -> 2 ($x-1)^$n - Eul[$n,$x-1]
                                                    /* MOS p. 30 */

SEul[18]:         Eul[$n,$x] -> Sum[Comb[$n,$m] Eul[m,$x-1],m,0,$n]
```

```
                                                        /* MOS p. 30 */

SEul[19]:       Eul[$n,$x] -> 2 $x^$n - Sum[Comb[$n,m] Eul[m,$x],m,0,$n]
                                                        /* MOS p. 30 */

                        /* Euler's numbers */

SEul[21]:       Eul[$n_?Natp[$n]] -> 2^$n Eul[$n,1/2]
                                                        /* MOS p. 31 */
```

XExDot

```
                /** Expansion of dot products **/

/* x_Scal
        declares x to be a scalar. */

/* Scalp[expr]
        tests whether expr contains only scalar objects. */
        Scalp[$expr] :: Ap[And,Map[P[_$1[Type]=Scal],Cont[$expr]]]

/* ExDot[expr]
        factors all declared scalars out of dot products. */
        ExDot[$expr] :: S[$expr,$$x.($a_?Scalp[$a]).$$y-->$a $$x.$$y, \
                ($a_?Scalp[$a]).$$x-->$a $$x, $$x.($a_?Scalp[$a])-->$a $$x,\
                $$x.(($a_?Scalp[$a]) $$b).$$y-->$a $$x.$$b.$$y, \
                (($a_?Scalp[$a]) $$b).$$x-->$a $$b.$$x, \
                $$x.(($a_?Scalp[$a]) $$b)-->$a $$x.$$b, Inf]

/* ExMDot[expr]
        sets all dot products of a matrix with its inverse to the
        identity. */
        ExMDot[$expr] :: S[$expr,Minv[$m].$m->1,$m.Minv[$m]->1,Inf]

/*
#I[1]::   <XExDot

#I[2]::   x_y_Scal

#O[2]:    Scal

#I[3]::   Scalp[x+y^2+1]

#O[3]:    1

#I[4]::   Scalp[a+x]

#O[4]:    0

#I[5]::   ExDot[a.(x b).(1+y).c]

#O[5]:    x a.b.c (1 + y)

#I[6]::   ExMDot[a.Minv[b].b.Minv[a].c]

#O[6]:    c

*/
```

XFPow

```
                    /** Functionals **/

/* FPow[f,n,x]
        yields n nested applications of f to x. */
        _FPow[Nosmp]:{1,0,0}
        FPow[$f,$n_?Natp[$n],$x] :: Rel[Lcl[%e];%e:$x;Rpt[%e:Ap[$f,{%e}],$n]]
        FPow[$f,0,$x] : $x

/*
#I[1]::  <<FPow

#I[2]::  FPow[f,10,x^2]

                                        2
#O[2]:   f[f[f[f[f[f[f[f[f[f[x ]]]]]]]]]]

#I[3]::  Ex[FPow[a $x(1-$x),3,x]]

          3      3  2    4  2      4  3    4  4     5  2     5  3    5  4
#O[3]:  a  x - a  x  + a  x  - 2 a  x  + a  x  + a  x  - 2 a  x  + a  x

                   6  3      6  4      6  5      6  6     7  4     7  5
              + 2 a  x  - 6 a  x  + 6 a  x  - 2 a  x  + a  x  - 4 a  x

                   7  6      7  7    7  8
              + 6 a  x  - 4 a  x  + a  x

*/
```

XFib

```
                    /** Fibonacci numbers **/

/* Fib[n]
        yields the nth Fibonacci number. */
        Fib[$n_?Natp[$n]] :: Gint[N[!N[Phi^$n/Sqrt[5]+1/2]]]
        Fib[$n] :: N[(Phi^$n-(-Phi)^(-$n))/Sqrt[5]]
```

XFierz

```
                    /** Fierz transformations **/

/* DIM
        denotes number of dimensions (default 4). */
        DIM : 4

/* Fz[k,l]
        yields elements of the Fierz rearrangement matrix. */
        Fz[$k,$l] :: (-1)^$l (DIM-2$l)/$k Fz[$k-1,$l] - \
                                (DIM-$k+2)/$k Fz[$k-2,$l]
        Fz[0,$l] :: -1/2^Gint[DIM/2]
        Fz[1,$l] :: (-1)^($l+1) (DIM-2$l)/2^Gint[DIM/2]

/* Fierz[dim]
        yields the complete Fierz transformation matrix for any natural
```

```
        number of dimensions dim. */
        Fierz[$dim_,Natp[$dim]] :: (Lcl[DIM]; DIM:$dim; \
                If[Evenp[DIM],Ar[{{0,DIM},{0,DIM}},Fz],   \
                        Ar[{{0,(DIM+1)/2},{0,(DIM+1)/2}},Fz]])
```

XFit

```
                        /** Curve fitting **/

/* Fit[{{x1,y1},{x2,y2},...},{x,y}]
        obtains a linear fit for the relation between x and y. */
        Fit[$list,{$x,$y}] :: Ap[$y=$1 + $2 $x,Ap[Fit0,Trans[$list]]]

        Fit0[$x,$y] :: (Lcl[%b0,%b1]; %b1:(Len[$x] Ap[Plus,$x $y] - \
        Ap[Plus,$x] Ap[Plus,$y]) / (Len[$x] Ap[Plus,$x^2] - \
        Ap[Plus,$x]^2); %b0:(Ap[Plus,$y] - %b1 Ap[Plus,$x])/Len[$x]; \
                                {%b0,%b1} )

/* FitExp[{{x1,y1},{x2,y2},...},{x,y}]
        obtains an exponential fit of the form y = a b^x for the
        relation between x and y. */
        FitExp[$list,{$x,$y}] :: Ap[N[$y = Exp[$1] Exp[$2]^$x], \
                                Ap[Fit0[$%1,N[Log[$%2]]],Trans[$list]]]

/* FitPow[{{x1,y1},{x2,y2},...},{x,y}]
        obtains a power fit of the form y = a x^b for the relation
        between x and y. */
        FitPow[$list,{$x,$y}] :: Ap[N[$y = Exp[$1] $x^$2], \
                                Ap[Fit0,N[Log[Trans[$list]]]]]

/* Corr[{{x1,y1},{x2,y2},...}]
        yields the population correlation coefficient between the xi and yi. */
        _Corr[Init] :: <<Stat
        Corr[$list] :: N[Lcl[%x,%y]; {%x,%y}:Trans[$list]; \
        Ap[Plus,(%x - Mean[%x])(%y - Mean[%y]) / Sqrt[Var[%x] Var[%y]]]]

/*

#I[1]::  <<Fit

#I[2]::  t:Ar[5,N[{$,2Exp[$]}]]

#O[2]:   {{1,5.436564},{2,14.77811},{3,40.17107},{4,109.1963},{5,296.8263}}

#I[3]::  Fit[%,{x,y}]

#O[3]:   109.8776 + y = 67.71977x

#I[4]::  S[%,x->3]

#O[4]:   y = 93.28167

#I[5]::  FitPow[t,{x,y}]

                        2.421572
#O[5]:   y = 3.953084 x

#I[6]::  FitExp[t,{x,y}]

                       x
#O[6]:   y = 2 2.718282

#I[7]::  Corr[t]

#O[7]:   .3270786

*/
```

XFrac

```
                        /** Fractions **/

/* Fr[n,r]
        represents the fraction n+r, where n is an integer, and r is a rationa
        number less than one. */
        /* Define output form */
        _Fr[Prt][$x,$y] :: Fmt[,"(",$x,"+",$y,")"]

/* ToFr[expr]
        converts rational numbers other than integers in expr to Fr form. */
        ToFr[$expr] :: S[$expr,($x_?Ratp[$x] & ~Intp[$x]) -> \
                                        Fr[Gint[$x],$x-Gint[$x]]]

/* FromFr[expr]
        converts fractions in Fr form to rational numbers. */
        FromFr[$expr] :: S[$expr,Fr->'Plus]

/*
#I[1]::  <XFrac

#I[2]::  t:ToFr[17/6]

#O[2]:*  (2+5/6)

#I[3]::  FromFr[%]

#O[3]:   17/6

#I[4]::  t^2

                    2
#O[4]:*  (2+5/6)

#I[5]::  Lpr[%]

Fr[2,5/6]^2

#I[6]::  ToFr[FromFr[t^2]]

#O[6]:*  (8+1/36)

#I[7]::  Pre::FromFr

#O[7]:   ' FromFr

#I[8]::  Post::ToFr

#O[8]:   ' ToFr

#I[9]::  t^2

#O[9]:*  (8+1/36)

*/
```

XG

```
                    /**  Dirac algebra  **/

/* Ginds
        is a list of all symbols assigned type Gind. */
        Ginds :: Rel[Cont[,_$1[Type]=Gind]]

/* VCon[expr]
        contracts any repeated pairs of indices in dot products in expr. */
        Con[$expr] :: S[Ex[$expr,,,,,In['Vdot,$1]], \
                Vdot[($%mu_: _$%mu[Type]=Gind),$%mu] -> Ndim, \
                Vdot[$%mu_: _$%mu[Type]=Gind,$%v1] Vdot[$%mu,$%v2] -> \
                                Vdot[$%v1,$%v2], Inf]

/* Transformations of products of Dirac gamma matrices */

SG[1] : ('G[$$x,$p,$q_:(Ord[$p,$q]<0),$$y]) -> 2('G[$$x,$$y]) $p[1].$q[1] - \
                                                'G[$$x,$q,$p,$$y]

SG[2] : ('G[$p,$q_:(Ord[$q,$p]<0)]) --> 2 $p[1].$q[1] - G[$q,$p]

SG[3] : ('G[$$x,$p,$q_:(Ord[$q,$p]<0)]) --> 2 Sp[1].$q[1] - G[$$x,$q,$p]

SG[4] : ('G[$p,$q_:(Ord[$q,$p]<0),$$x]) --> 2 $p[1].$q[1] - G[$q,$p,$$x]

/* Chisholm's identity for Ndim:4 */

SG[5] : ('G[$mu_: _$mu[Type]=Gind,$$x_:Oddp[Len[$$x]],$mu]) --> -2 Rev[$$x]
```

XGQInt

```
        /**  Gaussian quadrature integration  **/

/* GQInt[f,{x,lo,hi},npt]
        forms the numerical integral of f with respect to x between lo
        and hi using npt-point Gaussian quadrature. */
        GQInt[$f,{$x,$lo,$hi},$npt_:In[$npt,{6,12,24}]] :: \
                        N[($hi-$lo)/2 Sum[S[$f,$x-> \
                        (($hi-$lo)GQx[$npt,i]+($hi+$lo))/2] GQw[$npt,i], \
                                        i,-$npt/2,$npt/2]]
        GQx[6,0]:GQw[6,0]:0
        GQx[6,1]:0.2386191860
        GQx[6,2]:0.6612093864
        GQx[6,3]:0.9324695142
        GQx[6,-1]:-GQx[6,1]
        GQx[6,-2]:-GQx[6,2]
        GQx[6,-3]:-GQx[6,3]
        GQw[6,1]:GQw[6,-1]:0.4679139345
        GQw[6,2]:GQw[6,-2]:0.3607615730
        GQw[6,3]:GQw[6,-3]:0.1713244923

/*
#I[1]::  <<GQInt;GQInt:;<<GQInt

#I[2]::  GQInt[f[x],{x,-1,1},6]

#O[2]:   .1713245f[-.9324695] + .3607616f[-.6612094] + .4679139f[-.2386192]

                + .4679139f[.2386192] + .3607616f[.6612094]

                + .1713245f[.9324695]
```

```
#I[3]:  GQInt[x^2,{x,0,1},6]
#O[3]:  .3333333
*/
```

XGamma

```
                        /** Gamma function **/
SGamma _: Ldist
SGamma[1]:  Gamma[$z] -> ($z-1)Gamma($z-1)
                                                    /* MOS p. 2 */
SGamma[2]:  Gamma[$z] -> Gamma[$z+1]/$z
                                                    /* MOS p. 2 */
SGamma[3]:  Gamma[$z] -> Pi/Gamma[1-$z]/Sin[Pi $z]
                                                    /* MOS p. 2 */
SGamma[4]:  Gamma[$z] -> -Pi/$z/Gamma[-$z]/Sin[Pi $z]
                                                    /* MOS p. 2 */
SGamma[5]:  Gamma[$n_?Natp[$n]] -> ($n-1)!
SGamma[6]:  Gamma[$n_?Natp[$n-1/2]] -> Pi^(1/2)2^(1/2-$n)(2$n-2)!!
SGamma[7]:  Gamma[$z] -> Pi^-0.5 2^($z-1) Gamma[$z/2] Gamma[1/2+$z/2]
                                                    /* MOS p. 3 */
SGamma[8]:  Gamma[$z] -> 2^(1-2$z)Pi^(1/2)Gamma[2$z]/Gamma[$z+1/2]
SGamma[9]:  Gamma[$z] -> 2^(-2$z)Pi^(1/2)Gamma[2$z-1]/Gamma[$z-1/2]
SGamma[10]:  Gamma[$z] -> 1/(2 Pi) 3^($z-1/2) Gamma[$z/3] Gamma[$z/3+1/3]\
                    Gamma[$z/3+2/3]
                                                    /* MOS p. 3 */
SGamma[11]:  Gamma[($n_?Natp[$n]) $z] -> (2Pi)^((1-$n)/2)$n^($n $z-1/2)\
                    Prod[Gamma[$z+k/$n],k,0,$n-1]
                /* Incomplete gamma function */
SGamma[12]:  Gamma[$x,$a] -> $x^$a Exp[-$x]Chg[$a,1+$a,$x]
SGamma[13]:  Gamma[$x,0] -> -Ei[-$x]
SGamma[14]:  Gamma[$x,1/2] -> Pi^(1/2)Erf[$x]
SGamma[15]:  Gamma[$x,1] -> Exp[-$x]
SGamma[16]:  Gamma[$x,$a] -> ($a-1)Gamma[$a-1,$x] +$x^($a-1)Exp[$x]
SGamma[17]:  Gamma[$x,$a] -> (Gamma[$x,$a+1]-$x^$a Exp[$x])/$a
                                                    /* MOS pp.337-9 */
```

XGeg

```
                    /**  Gegenbauer polynomials  **/

SGeg_Ldist

SGeg[1]:   Geg[$m,$a,$x] -> \
                (2($m+$a-1) $x Geg[$m-1,$a,$x]-($m+2$a-2) Geg[$m-2,$a,$x])/$m

SGeg[2]:   Geg[$m,$a,$x] -> (($m+1) Geg[$m+1,$a,$x]\
                +($m+2$a-1) Geg[$m-1,$a,$x])/(2($m+$a) $x)

SGeg[3]:   Geg[$m,$a,$x] -> (2($m+$a+1) $x Geg[$m+1,$a,$x]\
                -($m+2) Geg[$m+2,$a,$x])/($m+2$a)


SGeg[4]:   Geg[$m,$a,$x] -> ((2$a+$m-2) Geg[$m,$a-1,$x]\
                -($m+1) $x Geg[$m+1,$a-1,$x])/(2($a+1) (1-$x^2))

SGeg[5]:   Geg[$m,$a,$x] -> (2$a (1-$x^2) Geg[$m,$a+1,$x]\
                +($n+1) $x Geg[$m+1,$a,$x])/(2$a+$m)

SGeg[6]:   Geg[$m,$a,$x] -> ((2$a+$m-1) Geg[$m-1,$a,$x]\
                -2$a (1-$x^2) Geg[$m-1,$a+1,$x])/($m $x)


SGeg[7]:   Geg[$m,$a,$x] -> (($m+2$a-1) Geg[$m+1,$a-1,$x]\
                -($m+2) $x Geg[$m+2,$a-1,$x])/(2($a-1) (1-$x^2))

SGeg[8]:   Geg[$m,$a,$x] -> (2$a (1-$x^2) Geg[$m-1,$a+1,$x]\
                +($n+1) Geg[$m+1,$a,$x])/($x ($m+2$a))
```

XGenp

```
                    /**  Test for generic symbols  **/

/* Genp[expr]
        yields 1 if expr contains generic symbols, and 0 otherwise. */
        Genp[$expr] :: P[Len[Cont[$expr,_$%1[Gen]]]]
```

XGr

```
                    /**  Basic graph theory  **/

/* A non-directed graph is represented by its incidence matrix, which
specifies the number of arcs (edges) connecting each pair of nodes. */

/* Nodes[list]
        gives the number of nodes in the graph represented by list. */
        Nodes[$list] :: Len[$list]

/* Arcs[list]
        gives the number of arcs in the graph represented by list. */
        Arcs[$list] :: Ap[Plus,Flat[$list]]/2

/* Regs[list]
```

```
        gives the number of regions (faces) for planar graphs represented
        by list. */
        Regs[$list] :: Arcs[$list] - Nodes[$list] + 2

/* Degree[list,n]
        yields the degree of node n in the graph represented by list. */
        Degree[$list,$n] :: Ap[Plus,$list[$n]]

/* ToArc[list]
        converts the incidence matrix list to a list of Arc projections
        representing arcs. */
        ToArc[$list] :: Map[Arc,Map[Ap[Rep[:$2,$1],$0],Pos[$3_=$3,$list]]]

/* ToNode[list]
        converts a list of arcs to an incidence matrix. */
        ToNode[$list] :: Ar[Ar[2,`Ap[Max,Flat[S[$list,Arc->List]]]], \
                                Len[Pos[Arc[List[$$x]],$list]]]

/* Eulerp[list]
        tests whether the graph represented by the incidence matrix
        list is Euler traversable. */
        Eulerp[$list] :: Ap[Plus,Map[Oddp,Map[Ap[Plus,$1],$list]]] <= 2

/* Hamp[list]
        tests whether the graph represented by list is Hamilton traversable. */
        Hamp[$list] :: (Lc[%u]; %u:Ar[Len[$list],0]; %u[1]:1; Hamp0[$list,1])
        Hamp0[$list,$n] :: (Lc[%t,%i]; %t:%i:0; Loop[%i<=Len[$list], \
                If[%u[$list[$n,%i]],0,%u[%i]:1;If[~(%t:Hamp0[$list,%i]), \
                        %u[%i]:0]];Inc[%i],~%t]; %t)

/* Relab[list,perm]
        relabs the nodes in the graph represented by the incidence
        matrix list according to the permutation perm. */
        _Relab[Init] :: <XPerm0
        Relab[$list,$perm] :: Apper[$perm,Map[Apper[$perm,$1],$list]]

/* Isop[gr1,gr2]
        tests for isomorphism of the graphs represented by incidence
        matrices gr1 and gr2. */
        Isop[$gr1,$gr1] : 1
        Isop[$gr1,$gr2_=Nodes[$gr1]~=Nodes[$gr2]] : 0
        Isop[$gr1,$gr2_=Arcs[$gr1]~=Arcs[$gr2]] : 0
        Isop[$gr1,$gr2_=Isop1[$gr1]~=Isop1[$gr2]] : 0
                Isop1[$list] :: Sort[Map[Ap[Plus,$1],$list]]
        Isop[$gr1,$gr2_=Isop2[$gr1]~=Isop2[$gr2]] : 0
                Isop2[$list] :: Ex[Det[$list-%lam Ar[Dim[$list]]]]

/*

#I[1]::  <XGr

#I[2]::  g:{{0,2,0,0},{2,0,1,1},{0,1,0,1},{0,1,1,0}}

#O[2]:   {{0,2,0,0},{2,0,1,1},{0,1,0,1},{0,1,1,0}}

#I[3]::  Nodes[g]

#O[3]:   4

#I[4]::  Arcs[g]

#O[4]:   5

#I[5]::  Regs[g]

#O[5]:   3

#I[6]::  Degree[g,2]

#O[6]:   4

#I[7]::  ToArc[g]
```

```
#O[7]:   {Arc[{1,2}],Arc[{1,2}],Arc[{2,1}],Arc[{2,1}],Arc[{2,3}],Arc[{2,4}],
              Arc[{3,2}],Arc[{3,4}],Arc[{4,2}],Arc[{4,3}]}
#I[8]::  ToNode[%]
#O[8]:   {{0,2,0,0},{2,0,1,1},{0,1,0,1},{0,1,1,0}}
#I[9]::  Eulerp[g]
#O[9]:   1
#I[10]::  Hamp[g]
#O[10]:   0
#I[11]::  Relab[g,{3,1,2,4}]
#O[11]:   {{0,0,1,1},{0,0,2,0},{1,2,0,1},{1,0,1,0}}
#I[15]::  Isop[g,g]
#O[15]:   1
#I[16]::  Isop[g,Sort[g]]
#O[16]:   0
*/
```

XHarm

```
                /** Harmonic sequence **/

/* Harm[n]
        represents the nth partial sum of the harmonic sequence. */
        Harm[$n] :: Sum[1/%i,%i,1,$n]
```

XHer

```
                /** Hermite polynomials **/

SHer_Ldist

SHer[1]:   Her[$m,$x] -> 2$x Her[$m-1,$x]-2($m-1) Her[$m-2,$x]

SHer[2]:   Her[$m,$x] -> (Her[$m+1,$x]+2$m Her[$m-1,$x])/(2$m)

SHer[3]:   Her[$m,$x] -> (2$x Her[$m+1,$x]-Her[$m+2,$x])/(2$m+2)
```

XHg1

```
/** Hypergeometric functions - 1 **/

/* 1. Special Cases */

/* 1.1 Special Elementary Cases */

SHg_Ldist

SHg[1,1]:   Hg[1,1,2,$z]->Log[1-$z]/$z

SHg[1,2]:   Hg[1/2,1,3/2,$z] -> Log[(1+Sqrt[$z])/(1-Sqrt[$z])]/(2 Sqrt[$z

SHg[1,3]:   Hg[1/2,1/2,3/2,$z] -> Asin[Sqrt[$z]]/Sqrt[$z]

SHg[1,4]:   Hg[1,1,3/2,$z] -> Asin[$z]/$z

SHg[1,5]:   Hg[$a,$b,$b,$z] -> (1-$z)^(-$a)

SHg[1,6]:   Hg[$a,$a+1/2,3/2,$z] -> \
              ((1+$z)^(1-2 $a)-(1-$z)^(1-2 $a))/(2 $z (1-2 $a))

SHg[1,7]:   Hg[-$a,$a,1/2,$z] -> \
              ((1-$z+Sqrtz])^(2 $a)+(1-$z-Sqrt[-$z])^(2 $a))/2

SHg[1,8]:   Hg[$a,1-$a,1/2,-$z^2] -> \
              ((Sqrt$z^2]+$z)^(2 $a-1)\
              +(Sqrt$z^2]-$z)^(2 $a-1))/(2 Sqrt[1+$z^2])

SHg[1,9]:   Hg[$a,$a+1/2,2*$a,$z] -> ((1+Sqrt[1-$z])/2)^(1-2 $a)/Sqrt[1-$

SHg[1,10]:  Hg[$a,1-$a,3/2,Sin[$z]^2] -> Sin[(2 $a-1) $z]/((2 $a-1) Sin[$z])

Shg1,11]:   Hg[$a,2-$a,3/2,Sin[$z]^2] -> Sin[(2 $a-2) $z]/(($a-1) Sin[2 $

SHg[1,12]:  Hg[$a,-$a,1/2,Sin[$z]^2] -> Cos[2 $a $z]

SHg[1,13]:  Hg[$a,1-$a,1/2,Sin[$z]^2] -> Cos[(2 $a-1) $z]/Cos[$z]

SHg[1,14]:  Hg[$a,1/2+$a,1/2,-Tan[$z]^2] -> Cos[$z]^(2 $a) Cos[2 $a $z]
```

XHg2

```
/** Hypergeometric functions - 2 **/

/* 2 Special Values of the Argument */

SHg_Ldist

SHg[2,1]:   Hg[$a,$b,$c,1] ->
              Gamma[$c] Gamma[$c-$a-$b]/(Gamma[$c-$a] Gamma[$c-$b]

SHg[2,2]:   Hg[$a,$b,$a-$b+1,-1] -> Sqrt[Pi] Gamma[1+$a-$b] \
              /(2^$a Gamma[+$a/2-$b] Gamma[$a/2+1/2])

SHg[2,3]:   Hg[$a,$b,$a-$b+2,-1] -> Sqrt[Pi] Gamma[$a-$b+2]/(2^$a ($b-1))\
              (1/(Gamma[$a/2] Gamma[$a/2-$b+3/2])\
```

```
                        -1/(Gamma[$a/2+1/2] Gamma[$a/2-$b+1]))

SHg[2,4]:       Hg[1,$a,$a+1,-1] -> $a/2 (Psi[$a/2+1/2]-Psi[$a/2])

SHg[2,5]:       Hg[$a,$b,$a/2+$b/2+1/2,1/2] -> Sqrt[Pi] Gamma[$a/2+$b/2+1/2]\
                        /(Gamma[$a/2+1/2] Gamma[$b/2+1/2])

SHg[2,6]:       Hg[$a,$b,$a/2+$b/2+1,1/2] -> 2 Sqrt[Pi] Gamma[$a/2+$b/2+1]\
                        (1/(Gamma[$a/2] Gamma[$b/2+1/2])\
                        -1/(Gamma[$a/2+1/2] Gamma[$b/2]))/($a-$b)

SHg[2,7]:       Hg[$a,1-$a,$b,1/2] -> 2^(1-$b) Sqrt[Pi] Gamma[$b]\
                        /(Gamma[$a/2+$b/2] Gamma[$b/2-$a/2+1/2])

SHg[2,8]:       Hg[1,1,$a,1/2] -> ($a-1) (Psi[$a/2]-Psi[$a/2-1/2])

SHg[2,9]:       Hg[$a,$a,$a+1,1/2] -> 2^($a-1) $a (Psi[$a/2+1/2]-Psi[$a/2])

SHg[2,18]:      Hg[$a,$a+1/2,3/2-2 $a,-1/3] ->
                        (9/8)^(2 $a) Gamma[4/3] Gamma[3/2-2 $a]\
                        /(Gamma[3/2] Gamma[4/3-2 $a])

SHg[2,11]:      Hg[$a,$a+1/2,5/6+$a,1/9] -> (3/4)^$a Sqrt[Pi] Gamma[5/6+2/3 $a\
                        /(Gamma[$a/3+1/2] Gamma[$a/3+5/6])
```

XHg3

```
                /**  Hypergeometric functions - 3  **/

                /*  Special Cases: Polynomials  */
SHg_Ldist

SHg[3,1]:   Hg[-$n_Intp[$n],$n+1,1,$x] -> Legp[$n,1-2 $x]

SHg[3,2]:   Hg[-$n_Intp[$n],$n+2$b-1,$b,$x] -> \
                $n!/Poch[2$b-1,$n] Geg[$n,$b-1/2,1-2 $x]

SHg[3,3]:   Hg[-$n_Intp[$n],$c,$a,$x] -> \
                $n!/Poch[$a,$n] Jac[$n,1-2 $x,$a-1,$c-$a-$n]

SHg[3,4]:   Hg[-$n_Intp[$n],$n,1/2,$x] -> CheT[$n,1-2 $x]
```

XHg4

```
/** Hypergeometric functions - 4 **/

/* Special Cases: Legendre Functions */

SHg_Ldist

SHg[4,1]:  Hg[$a,$b,2 $b,$z] -> 2^(2 $b-1) Gamma[$b+1/2] $z^(1/2-$b)\
           (1-$z)^(($b-$a-1/2)/2) Legp[$a-$b-1/2,1/2-$b,(1-$z/2)/Sqrt[1-$z]]

SHg[4,2]:  Hg[$a,$b,2 $b,$z] -> 2^(2 $b) Gamma[$b+1/2] Exp[I Pi ($a-$b)]\
             (1-$z)^(($b-$a)/2)/(Sqrt[Pi] Gamma[2 $b-$a] $z^$b)\
           Legp[$b-1,$b-$a,2/$z-1]
```

XHg5

```
/** Functional relations for hypergeometric functions **/

/* Elementary Transformations */

SHg_Ldist

SHg[5,1]:        Hg[$a,$b,$c,$z] -> Hg[$b,$a,$c,$z]


/* Linear Transformations */

SHg[5,1]:        Hg[$a,$b,$c,$z] -> (1-$z)^($c-$b-$a) Hg[$c-$a,$c-$b,$c,$z]

SHg[5,2]:        Hg[$a,$b,$c,$z] -> Hg[$a,$c-$b,$c,$z/($z-1)]/(1-$z)^$a

SHg[5,3]:        Hg[$a,$b,$c,$z] -> Gamma[$c] Gamma[$c-$a-$b]\
                   /(Gamma[$c-$a] Gamma[$c-$b]) Hg[$a,$b,$a+$b-$c+1,1-$z]\
                     +(1-$z)^($c-$a-$b) Gamma[$c] Gamma[$a+$b-$c]\
                    /(Gamma[$a] Gamma[$b]) Hg[$c-$a,$c-$b,$c-$a-$b+1,1-$z]

SHg[5,4]:        Hg[$a,$b,$c,$z] -> 1/(-$z)^$a Gamma[$c] Gamma[$b-$a]\
                   /(Gamma[$b] Gamma[$c-$a]) Hg[$a,1-$c+$a,1-$b+$a,1/$z]\
                     +1/(-$z)^$b Gamma[$c] Gamma[$a-$b]\
                   /(Gamma[$a] Gamma[$c-$b]) Hg[$b,1-$c+$b,1-$a+$b,1/$z]

SHg[5,5]:        Hg[$a,$b,$c,$z] ->\
                   1/(1-$z)^$a Gamma[$c] Gamma[$b-$a]/(Gamma[$b]\
                    Gamma[$c-$a]) Hg[$a,$c-$b,$a-$b+1,1/(1-$z)]\
                   +1/(1-$z)^$b Gamma[$c] Gamma[$a-$b]/(Gamma[$a]\
                    Gamma[$c-$b]) Hg[$b,$c-$a,$b-$a+1,1/(1-$z)]


SHg[5,6]:        Hg[$a,$b,$c,$z] -> 1/$z^$a Gamma[$c] Gamma[$c-$a-$b]\
                 (Gamma[$c-$a] Gamma[$c-$b]) Hg[$a,$a-$c+1,$a+$b-$c+1,1-1/$z]\
                 +(1-$z)^($c-$a-$b) $z^($a-$c) Gamma[$c] Gamma[$a+$b-$c]\
                 /(Gamma[$a] Gamma[$b]) Hg[$c-$a,1-$a,$c-$a-$b+1,1-1/$z]


/* Quadratic Transformations */

SHg[5,7]:         Hg[$a,$b,2 $b,$z] -> \
                 (1-$z)^(-$a/2) Hg[$a/2,$b-$a/2,$b+1,$z^2/(4 $z-4)]
```

```
SHg[5,8]:     Hg[$a,$b,2 $b,$z] -> \
                (1-$z/2)^$a Hg[$a/2,$a/2+1/2,$b+1/2,$z^2/(2-$z)^2]

SHg[5,9]:      Hg[$a,$b,2 $b,$z] -> (1/2+Sqrt[1-$z]/2)^(-2 $a)\
               Hg[$a,$a-$b+1/2,$b+1/2,((1-Sqrt[1-$z])/(1+Sqrt[1-$z]))^2]

Shg[5,10]:    Hg[$a,$b,2 $b,$z] -> (1-$z)^(-$a/2) \
               Hg[$a,2 $b-$a,$b+1/2,-(1-Sqrt[1-$z])^2/(4 Sqrt[1-$z])]


SHg[5,11]:    Hg[$a,$a+1/2,$c,$z] -> (1/2+Sqrt[1-$z]/2)^(-2 $a)\
                Hg[2 $a,2 $a-$c+1,$c,(1-Sqrt[1-$z])/(1+Sqrt[1-$z])]

SHg[5,12]:    Hg[$a,$a+1/2,$c,$z] -> (1-Sqrt[$z])^(-2 $a)\

SHg[5,13]:    Hg[$a,$a+1/2,$c,$z] -> (1+Sqrt[$z])^(-2 $a)\
                Hg[2 $a,$c-1/2,2 $c-1,2 Sqrt[$z]/(1+Sqrt[$z])]

SHg[5,14]:    Hg[$a,$a+1/2,$c,$z] -> 1/(1-$z)^$a\
                 Hg[2 $a,2 $c-2 $a-1,$c,(Sqrt[1-$z]-1)/(2 Sqrt[1-$z])]

SHg[5,15]:    Hg[$a,$b,$a+$b+1/2,$z] -> Hg[2 $a,2 $b,$a+$b+1/2,1/2-Sqrt[1-$z]/2

SHg[5,16]:    Hg[$a,$b,$a+$b+1/2,$z] -> (1/2+Sqrt[1-$z]/2)^(-2 $a)\
                 Hg[2 $a,$a-$b+1/2,$a+$b+1/2,(Sqrt[1-$z]-1)/(Sqrt[1-$z]+1)]

SHg[5,17]:    Hg[$a,$b,$a+$b-1/2,$z] ->\
                1/Sqrt[1-$z] Hg[2 $a-1,2 $b-1,$a+$b-1/2,1/2-Sqrt[1-$z]/2]

SHg[5,18]:    Hg[$a,$b,$a+$b-1/2,$z] -> (1/2+Sqrt[1-$z])^(1-2 $a)/Sqrt[1+$z]\
                Hg[2 $a-1,$a-$b+1/2,$a+$b-1/2,(Sqrt[1-$z]-1)/(Sqrt[1-$z]+1)]


SHg[5,19]:    Hg[$a,$b,$a-$b+1,$z] ->\
                1/(1+$z)^(2 $a) Hg[$a/2,$a/2+1/2,$a-$b+1,4 $z/(1+$z)^2]

SHg[5,20]:    Hg[$a,$b,$a-$b+1,$z] -> (1+Sqrt[$z])^(-2 $a)\
                Hg[$a,$a-$b+1/2,2 $a-2 $b+1,4 Sqrt[$z]/(1+Sqrt[$z])^2]

SHg[5,21]:    Hg[$a,$b,$a-$b+1,$z] -> (1-Sqrt[$z])^(-2 $a)\
                Hg[$a,$a-$b+1/2,2 $a-2 $b+1,-4 Sqrt[$z]/(1-Sqrt[$z])^2]

SHg[5,22]:    Hg[$a,$b,$a-$b+1,$z] -> \
                1/(1-$z)^$a Hg[$a/2,$a/2-$b+1,-4 $z/(1-$z)^2]


SHg[5,23]:    Hg[$a,$b,($a+$b+1)/2,$z] -> \
                Hg[$a/2,$b/2,($a+$b+1)/2,4 $z($z-1)]

SHg[5,24]:    Hg[$a,$b,$a/2+$b/2+1/2,$z] -> 1/(1-2 $z)^$a \
                 Hg[$a/2,$a/2+1/2,$a/2+$b/2+1/2,4 $z ($z-1)/(1-2 $z)^2]


SHg[5,25]:    Hg[$a,1-$a,$c,$z] -> \
                (1-$z)^($c-1)Hg[$c/2-$a/2,$c/2+$a/2-1/2,$c,4 $z-4 $z^2]

SHg[5,26]:    Hg[$a,1-$a,$c,$z] -> (1-$z)^($c-1) (1-2 $z)^($a-$c)\
                 Hg[$c/2-$a/2,$c/2-$a/2+1/2,$c,4 $z($z-1)/(1-2 $z)^2]
```

XHist

```
                    /**  Form histogram  **/

/* Hist[list,nbin]
        forms a histogram of the contents of list with nbin bins. */
        Hist[$list_:Listp[$list],$nbin_:Natp[$nbin]] :: \
        (Lcl[%min,%max,%step,%hist,%ind]; %step:N[((%max:Ap[Max,$list])-\
        (%min:Ap[Min,$list]))/$nbin]; %hist:Ar[{{%min,%max,%step}},0];\
        Do[%i,Len[$list],%ind:%min+%step Gint[$list[%i]/%step]; \
                        %hist[%ind]:%hist[%ind]+1];%hist)

/*
#I[1]::  <XHist

#I[2]::  t:Ar[10,Prime]

#O[2]:   {2,3,5,7,11,13,17,19,23,29}

#I[3]::  Hist[t,3]

#O[3]:   {[2]: 4, [11]: 3, [20]: 2, [29]: 1}

*/
```

XHorn

```
                    /**  Horner representation  **/

/* Horn[poly,x]
        constructs a Horner representation of the polynomial poly
        with respect to x. */
        Horn[$poly,$x] :: (Lcl[%p,%n]; If[(%n:Expt[$x,$poly])<=1,Rst[$poly]],\
        %p:Coef[$x^%n,$poly]; Do[%1,%n-1,%p:%p $x + Coef[$x^(%n-%1),$poly]; \
        $x %p + S[$poly,$x->0])

/*
#I[1]::  <XHorn

#I[2]::  t:x^3+4a x^2+2x+c

                          2    3
#O[2]:   c + 2x + 4a x  + x

#I[3]::  Horn[%,x]

#O[3]:   c + x (2 + x (4a + x))

*/
```

XInd

```
          /** Manipulation of indices in lists **/

/* LInd[list]
          yields a list of the indices in list. **/
          LInd[$list] :: Ar[Len[$list],Ind[$list,$1]]

/* ToL[list]
          writes entries in list as {index,value}. */
          ToL[$list] :: Ar[Len[$list],{Ind[$list,$1],Elem[$list,{$1}]}]

/* ToInd[list]
          takes sublists {index,value} in list, and forms an
          indexed list [index]:value. */
          ToInd[$list] :: (Lc[%i]; Map[%i[$1[1]]:$1[2],$list]; %i)

/*
#I[1]::   <XInd

#I[2]::   t:{[a]:x^2,[b]:x^3,[c]:x+y}

                    2       3
#O[2]:    {[a]: x , [b]: x , [c]: x + y}

#I[3]::   LInd[%]

#O[3]:    {a,b,c}

#I[4]::   ToL[t]

                 2       3
#O[4]:    {{a,x },{b,x },{c,x + y}}

#I[5]::   ToInd[%]

                                3       2
#O[5]:    {[c]: x + y, [b]: x , [a]: x }

*/
```

XIndep

```
          /** Declaration of independent variables **/

/* Indep[{y1,y2,...},{x1,x2,...}]
          defines the yi to be independent of the xj so that Dt[yi,xj]:0. */
          Indep[$y,$x] :: Map[Map[Dt[$1,$2]:0,$x],$y]

/*

#I[1]::   <XIndep

#I[2]::   Dt[{y1,y2,y3},x]

#O[2]:    {Dt[y1,x],Dt[y2,x],Dt[y3,x]}

#I[3]::   Indep[{y1,y2},{x,xp}]

#O[3]:    {{0,0},{0,0}}

#I[4]::   Dt[{y1,y2,y3},x]

#O[4]:    {0,0,Dt[y3,x]}

*/
```

XInfo

```
                    /** Basic information theory **/

/* Shan[prob]
        represents the Shannon entropy for a language whose symbols have
        relative frequencies as given in the list prob. */
        Shan[$prob_?Listp[$prob]] :: -N[Ap[Plus,Map[$x Log[$x,2],$prob]]/  \
                                                    Ap[Plus,$prob]]

/* Freq[list]
        yields a list of the relative frequencies of symbols appearing
        as elements of list. */
        Freq[$list_?Listp[$list]] :: (Lc[%f]; Map[%f[$1[1]]:$1[2], \
                            Rex[$list][2]]; Re[%f])

/*
#I[1]::  <XInfo

#I[2]::  Ar[10,$x^2]

#O[2]:   {1,4,9,16,25,36,49,64,81,100}

#I[3]::  Shan[%]

#O[3]:   -5.817683

#I[4]::  {a,a,b,c,a,c,d,e,a,b,c,b,b,e}

#O[4]:   {a,a,b,c,a,c,d,e,a,b,c,b,b,e}

#I[5]::  Freq[%]

#O[5]:   {[e]: 2, [d]: 1, [c]: 3, [b]: 4, [a]: 4}

*/
```

XInt

```
        /** Elementary definite integrals **/

Int[$x^($n_?$n>-1) Exp[-$x + 1],{$x,0,Inf}]: Gamma[$n + 1]
                                                    /* CRC 470 */

Int[$t^($n_?Natp[$n+1]) ($p_?$p>0) ^(-$t),{$t,0,Inf}]: $n!/Log[$p]^($n+1)
                                                    /* CRC 471 */

Int[$t^($n_?($n+1)>0) Exp[($$a_?((~In[$a,$t])&(($a-1)>-1))) $t],\
                            {$x,0,Inf}] : Gamma[$n+1]/$a^$n
                                                    /* CRC 472 */

Int[$x^($m_?$m>-1) Log[$x]^($n_?$n>-1),{$x,0,1}] : \
                            (-1)^$n Gamma[$n+1]/($m+1)^($n+1)
                                                    /* CRC 473 */
```

```
Int[$x^$m (1-$x)^$n,{$x,0,1}] : Beta[$m+1,$n+1]
                                                        /* CRC 479 */

Int[$x^$m/(1+$x)^$n,{$x,0,Inf}] : Beta[$m+1,$n-$m-1]
                                                        /* CRC 481 */

Int[($x - ($a_~In[$a,$x]))^($m_$m > -1) (($b_~In[$b,$x]&($b > $a)) - $x)^($n\
        _$n > -1),{$x,$a,$b}] : Gamma[$n+1] Gamma[$n+1]/Gamma[$n + $m +2]\
        ($b - $a)^($m + $n + 1)
                                                        /* CRC 482 */

Int[1/(1 + $x)/$x^($p_$p < 1),{$x,0,Inf}] : Pi/Sin[$p Pi]
                                                        /* CRC 484 */

Int[1/(1 - $x)/$x^($p_$p < 1),{$x,0,Inf}] : -Pi Cot[$p Pi]
                                                        /* CRC 485 */

Int[$x^($p_$p > -1 & $p < 0)/(1 + $x),{$x,0,Inf}] : Pi/Sin[Pi ($p +1)]
                                                        /* CRC 486 */

Int[$x^($m_$m > -1 & ($m+1) < $n)/(1 + $x^$n),{$x,0,Inf}] : Pi/
        ($n Sin[($m+1)/$n Pi])
                                                        /* CRC 487 */

Int[$x^($a_$a > -1)/(($m_$m > 0) + $x^($b_$b > 0))^($c_$c > ($a+1)/$b),\
        {$x,0,Inf}] : $m ($a + 1)/($b - $c) (Gamma[($a +1)/$b]
        Gamma[$c - ($a + 1)/$b]/Gamma[$c])
                                                        /* CRC 488 */

Int[1/(1+$x)/$x^(1/2),{$x,0,Inf}] : Pi
                                                        /* CRC 489 */

Eps[$x]: -($x < 0) + ($x > 0)

Int[$a/(($b_$b = $a^2) + $x^2),{$x,0,Inf}] : Pi/2 Eps[$a]
                                                        /* CRC 490 */

Int[(($$a_~In[$$a,$x]) - $x^2)^(($n_Oddp[$n]) /2),{$x,0,$$a^(1/2)}] : \
        $n!!/($n+1)!! *  Pi/2 $$a^(($n+1)/2)
                                                        /* CRC 491 */

Int[$x^$m($a-$x^2)^$n,{$x,0,$a^(1/2)}] : $a^(($m + $n + 1)/2)/2
        Beta[($m + 1)/2,($n + 2)/2]
                                                        /* CRC 492 */

Int[Sin[$x]^$n,{$x,0,Pi/2}] : Pi^(1/2)/2 Gamma[($n+1)/2]/Gamma[$n/2 + 1]
                                                        /* CRC 493 */

Int[Sin[$m $x]/$x,{$x,0,Inf}] : Eps[$m] Pi/2
                                                        /* CRC 494 */

Int[Tan[$x]/$x,{$x,0,Inf}] : Pi/2
                                                        /* CRC 496 */

Int[Sin[($a_Intp[$a]) $x] Sin[($b_($b ~= $a) & Intp[$b]) $x],{$x,0,Pi}] : 0
                                                        /* CRC 497 */

Int[Cos[($a_Intp[$a]) $x] Cos[($b_($b ~= $a) & Intp[$b]) $x],{$x,0,Pi}] : 0
                                                        /* CRC 497 */

Int[Sin[$a $x] Cos[$a $x],{$x,0,Pi/$a}] : 0
                                                        /* CRC 498 */

Int[Sin[$a $x] Cos[$a $x],{$x,0,Pi}] : 0
                                                        /* CRC 498 */

Int[Sin[$a $x] Cos[($b_Intp[$a-$b]) $x],{$x,0,Pi}] : 2 $a/($a^2 - $b^2)\
        Oddp[$a - $b]
                                                        /* CRC 499 */

Int[Sin[$x] Cos[$m $x]/$x,{$x,0,Inf}] : Pi/2 ($m^2 < 1)       + Pi/4 ($m^2 = 1)
                                                        /* CRC 500 */
```

```
Int[Sin[$a $x] Sin[$b $x]/$x^2,{$x,0,Inf}] : Pi $a/2 (($a <= $b) + Pi $b/2 \
        ($b < $a)
                                                    /* CRC 501 */

Int[Sin[$m $x]^2,{$x,0,Pi}] : Pi/2
                                                    /* CRC 502 */

Int[Cos[$m $x]^2,{$x,0,Pi}] : Pi/2
                                                    /* CRC 502 */

Int[Sin[$x]^2/$x^2,{$x,0,Inf}] : Pi/2
                                                    /* CRC 503 */

Int[Sin[$x^2],{$x,0,Inf}] : 1/2 (Pi/2)^(1/2)
                                                    /* CRC 505 */

Int[Cos[$x^2],{$x,0,Inf}] : 1/2 (Pi/2)^(1/2)
                                                    /* CRC 505 */

Int[Sin[$x]/$x^(1/2),{$x,0,Inf}] : (Pi/2)^(1/2)
                                                    /* CRC 506 */

Int[Cos[$x]/$x^(1/2),{$x,0,Inf}] : (Pi/2)^(1/2)
                                                    /* CRC 506 */

Int[1/(1 + $a Cos[$x]),{$x,0,Pi/2}] : Acos[$a]/(1 - $a^2)^(1/2)
                                                    /* CRC 507 */

Int[1/($a + $b Cos[$x]),{$x,0,Inf}] : Pi/($a^2 - $b^2)^(1/2)
                                                    /* CRC 508 */

Int[1/(1 + $a Cos[$x]),{$x,0,2 Pi}] : 2 Pi/(1 - $a^2)^(1/2)
                                                    /* CRC 509 */

Int[(Cos[$a $x] - Cos[$b $x])/$x,{$x,0,Inf}] : Log[$b/$a]
                                                    /* CRC 510 */

Int[1/($a Sin[$x]^2 + $b Cos[$x]^2),{$x,0,Pi/2}] : Pi/(2 $a^(1/2) $b^(1/2))
                                                    /* CRC 511 */

Int[1/($a Sin[$x]^2 + $b Cos[$x]^2)^2,{$x,0,Pi/2}] : Pi($a + $b)/(4 $a^(3/2)\
        $b^(3/2))
                                                    /* CRC 512 */

Int[Sin[$x]^($n_>Natp[$n]) Cos[$x]^($m_>Natp[$m]),{$x,0,Pi/2}] : \
        Beta[($n+1)/2,($m+1)/2]/2
                                                    /* CRC 513 */

Int[Sin[$x]^$n,{$x,0,Pi/2}] : 2^($n-1) Beta[($n-1)/2,($n-1)/2]
                                                    /* GR 3.621 */

Int[Cos[$x]^$n,{$x,0,Pi/2}] : 2^($n-1) Beta[($n-1)/2,($n-1)/2]
                                                    /* GR 3.621 */

Int[Cos[$x]^(1/2),{$x,0,Pi/2}] : (2 Pi)^(3/2)/Gamma[1/4]^2
                                                    /* CRC 516 */

Int[Tan[$x]^$h,{$x,0,Pi/2}] : Pi/2/Cos[$h/2 Pi]
                                                    /* CRC 517 */

Int[(Atan[$a $x] - Atan[$b $x])/$x,{$x,0,Inf}] : Pi/2 Log[$a/$b]
                                                    /* CRC 518 */

Int[(Exp[($a_>$a < 0) $x] - Exp[($b_>$b < 0) $x])/$x,{$x,0,Inf}] : \
        Log[$b/$a]
                                                    /* CRC 522 */

Int[$x^$n Exp[($a_>$a < 0) $x],{$x,0,Inf}] : Gamma[$n+1]/(-$a)^($n+1)
                                                    /* CRC 523 */

Int[Exp[($a_>$a < 0) $x^2],{$x,0,Inf}] : 1/2/(-$a)^(1/2) Pi^(1/2)
                                                    /* CRC 524 */
```

```
   Int[$x Exp[-$x^2],{$x,0,Inf}] : 1/2
                                                        /* CRC 525 */

   Int[$x^2 Exp[-$x^2],{$x,0,Inf}] : Pi^(1/2)/4
                                                        /* CRC 526 */

Int[$x^$n Exp[($a_=$a < 0) $x^2],{$x,0,Inf}] : ($n-1)!!/2^($n/2+1)/(-$a)^($n/2)\
            (Pi/-$a)^(1/2)
                                                        /* CRC 527 */

   Int[$x^$n Exp[($a_=$a < 0) $x],{$x,0,1}] : $n!/(-$a)^($n+1) (1 - Exp[$a]\
           Sum[(-$a)^r/r!,r,0,$n])
                                                        /* CRC 528 */

   Int[Exp[-$x^2-($a_=$a >= 0)^2 /$x^2],{$x,0,Inf}] : Exp[-2 $a] Pi^(1/2)/2
                                                        /* CRC 529 */

   Int[Exp[$n $x] $x^(1/2),{$x,0,Inf}] : 1/(-2 $n) (Pi/(-$n))^(1/2)
                                                        /* CRC 530 */

   Int[Exp[$n $x]/$x^(1/2),{$x,0,Inf}] : (Pi/(-$n))^(1/2)
                                                        /* CRC 531 */

   Int[Exp[($a_=$a < 0) $x] Cos[$m $x],{$x,0,Inf}] : -$a/($a^2 + $m^2)
                                                        /* CRC 532 */

   Int[Exp[($a_=$a < 0) $x] Sin[$m $x],{$x,0,Inf}] : $m/($a^2 + $m^2)
                                                        /* CRC 533 */

   Int[$x Exp[($a_=$a < 0) $x] Sin[$m $x],{$x,0,Inf}] : -2 $a $m/($a^2 + $m^2)^2
                                                        /* CRC 534 */

Int[$x Exp[($a_=$a < 0) $x] Cos[$m $x],{$x,0,Inf}] : ($a^2-$m^2)/($a^2 + $m^2)^2
                                                        /* CRC 535 */

   Int[$x^$n Exp[($a_=$a < 0) $x] Sin[$b $x],{$x,0,Inf}] : $n!*((-$a-I $b)^($n+1)\
           - (-$a + I $b)^($n+1))/2 /($a^2 + $m^2)^($n+1)
                                                        /* CRC 536 */

   Int[$x^$n Exp[($a_=$a < 0) $x] Cos[$b $x],{$x,0,Inf}] : $n!*((-$a-I $b)^($n+1)\
           + (-$a + I $b)^($n+1))/2/($a^2 + $m^2)^($n+1)
                                                        /* CRC 537 */

   Int[Exp[($a_=$a < 0) $x] Sin[$x]/$x,{$x,0,Inf}] : Acot[-$a]
                                                        /* CRC 538 */

   Int[Exp[-$a^2 $x^2] Cos[$b $x],{$x,0,Inf}] : Pi^(1/2)/2/$a Exp[-$b^2/4/$a^2]
                                                        /* CRC 539 */

Int[Exp[-$t Cos[($x_=$x^2<Pi^2/4)]] $t^($b_=$b>-1) Sin[$t Sin[$x]],{$t,0,Inf}]:\
            Gamma[$b + 1] Sin[($b + 1) $x]
                                                        /* CRC 540 */

Int[Exp[$t Cos[($x_=$x^2 < Pi^2/4)]] $t^($b_=$b>0) Cos[$t Sin[$x]],{$t,0,Inf}]:\
            Gamma[$b + 1] Cos[($b + 1) $x]
                                                        /* CRC 541 */

   Int[$t^($b_=($b>-1 & $b<0))Cos[$t],{$t,0,Inf}] : Gamma[$b+1] Cos[($b + 1) Pi/2]
                                                        /* CRC 542 */

   Int[$t^($b_=($b>-1 & $b<0))Sin[$t],{$t,0,Inf}] : Gamma[$b+1] Sin[($b + 1) Pi/2]
                                                        /* CRC 543 */

   Int[Log[$x]^$n,{$x,0,1}] : (-1)^$n n!
                                                        /* CRC 544 */

   Int[Log[1/$x]^(1/2),{$x,0,1}] : Pi^(1/2)/2
                                                        /* CRC 545 */

   Int[Log[1/$x]^(-1/2),{$x,0,1}] : Pi^(1/2)
                                                        /* CRC 546 */
```

```
Int[Log[1/$x]^$n,{$x,0,1}] : n!
                                                        /* CRC 547 */

Int[$x Log[1 - $x],{$x,0,1}] : -3/4
                                                        /* CRC 548 */

Int[$x Log[1 + $x],{$x,0,1}] : 1/4
                                                        /* CRC 549 */

Int[Log[$x]/(1 + $x),{$x,0,1}] : -Pi^2/12
                                                        /* CRC 550 */

Int[Log[$x]/(1 - $x),{$x,0,1}] : -Pi^2/6
                                                        /* CRC 551 */

Int[Log[$x]/(1 - $x^2),{$x,0,1}] : -Pi^2/8
                                                        /* CRC 552 */

Int[Log[(1 + $x)/(1 - $x)]/$x,{$x,0,1}] : Pi^2/4
                                                        /* CRC 553 */

Int[Log[$x]/(1 - $x^2)^(1/2),{$x,0,1}] : -Pi/2 Log[2]
                                                        /* CRC 554 */

Int[$x^($m_?$m+1>0) Log[1/$x]^($n_?$n+1>0),{$x,0,1}] : Gamma[$n + 1]/\
        ($m + 1)^($n + 1)
                                                        /* CRC 555 */

Int[($x^$p - $x^$q)/Log[$x],{$x,0,1}] : Log[($p + 1)/($q + 1)]
                                                        /* CRC 556 */

Int[1/Log[1/$x]^(1/2),{$x,0,1}] : Pi^(1/2)
                                                        /* CRC 557 */

Int[Log[(Exp[$x] + 1)/(Exp[$x] - 1)],{$x,0,Inf}] : Pi^2/4
                                                        /* CRC 558 */

Int[Log[Sin[$x]],{$x,0,Pi/2}] : -Pi/2 Log[2]
                                                        /* CRC 559 */

Int[Log[Cos[$x]],{$x,0,Pi/2}] : -Pi/2 Log[2]
                                                        /* CRC 559 */

Int[Log[Sec[$x]],{$x,0,Pi/2}] : Pi/2 Log[2]
                                                        /* CRC 560 */

Int[Log[Csc[$x]],{$x,0,Pi/2}] : Pi/2 Log[2]
                                                        /* CRC 560 */

Int[$x Log[Sin[$x]],{$x,0,Pi}] : -Pi^2/2 Log[2]
                                                        /* CRC 561 */

Int[Sin[$x] Log[Sin[$x]],{$x,0,Pi/2}] : Log[2] - 1
                                                        /* CRC 562 */

Int[Log[Tan[$x]],{$x,0,Pi/2}] : 0
                                                        /* CRC 563 */

Int[Log[$a + ($b_?$b<=$a)Cos[$x]],{$x,0,Pi}] : Pi Log[($a+($a^2-$b^2)^(1/2))/2]
                                                        /* CRC 564 */

Int[Log[$a - ($b_?$b<=$a)Cos[$x]],{$x,0,Pi}] : Pi Log[($a+($a^2-$b^2)^(1/2))/2]
                                                        /* CRC 564 */

Int[1/Cosh[$a $x],{$x,0,Inf}] : Pi/2/$a
                                                        /* CRC 565 */

Int[$x/Sinh[$a $x],{$x,0,Inf}] : (Pi/2/$a)^2
                                                        /* CRC 566 */

Int[Exp[($a_?$a < 0) $x] Cosh[($b_?$b>=0 & $b < -$a) $x],{$x,0,Inf}] :\
        -$a/($a^2 - $b^2)
```

```
                                                        /* CRC 567 */

Int[Exp[($a_?$a < 0) $x] Sinh[($b_?$b>=00& $b < -$a) $x],{$x,0,Inf}] :\
        $b/($a^2 - $b^2)
                                                        /* CRC 568 */

Int[Exp[-$x] Log[$x],{$x,0,Inf}] : -Euler
                                                        /* CRC 572 */

Int[(1/(1 - Exp[-$x]) - 1/$x)Exp[-$x],{$x,0,Inf}] : Euler
                                                        /* CRC 573 */

Int[1/$x (1/(1 + $x) - Exp[-$x]),{$x,0,Inf}] : Euler
                                                        /* CRC 573 */
```

XIntp

```
        /** Additional rules for integer testing **/


Intp[($$x_?Intp[$$x])+($y_?Intp[$y])] : 1
Intp[($$x_?Intp[$$x])($y_?Intp[$y])] : 1

Intp[($n_?Intp[$n])^($m_?Natp[$m])] : 1
Intp[($n_?Abs[$n]>1)^($m_?Natp[-$m])] : 0

/*
#I[1]::  <<Intp

#I[2]::  Intp[x]:1

#O[2]:   1

#I[3]::  Intp[x^2-2]

#O[3]:   1

*/
```

XIter

```
                /** General iterated forms **/

/* Iter[f,expr,var,lo,hi]
        applies f to the set of values of expr attained when var
        takes on values lo, lo+1, lo+2, ... , hi. */
        _Iter[Nosmp]:{1,1,1,0,0}
        Iter[$f,$expr,$var,$lo,$hi_?Intp[$hi-$lo]] :: \
                Ap[$f,Ar[{{$lo,$hi}},Rel[S[$expr,$var->$1]]]]

/*
#I[1]::  <<Iter

#O[1]:   ' Ap[$f,Ar[{{$lo,$hi}},Rel[S[$expr,$var -> $1]]]]

#I[2]::  Iter[f,x+i^2,i,1,6]

#O[2]:   f[1 + x,4 + x,9 + x,16 + x,25 + x,36 + x]

 */
```

XItp

```
        /** Lagrange interpolation of list values **/

/* Itp[list,x]
        uses all the values given in list to yield an optimal estimate
        for the value corresponding to an index x. */
        Itp[$list,$x] :: (Lcl[%n]; %n:Len[$list]; \
                Sum[%x:Ind[$list,i];Prod[$x-Ind[$list,j],j,1,%n]/ \
                (($x-%x) Prod[%x-Ind[$list,j],j,1,%n,1,j~=i]) \
                        Elem[$list,{i}], i, 1, Len[$list]])

/*

#I[1]::  <XItp

#I[2]::  t:Ar[{{0,1,0.2}},N[Sin[$x]]]

#O[2]:   {[0]: 0, [.2]: .1986693, [.4]: .3894183, [.6]: .5646425,

              [.8]: .7173561, [1]: .841471}

#I[3]::  Itp[%,x]

#O[3]:   25.8684x (-1 + x) (-4/5 + x) (-3/5 + x) (-2/5 + x)

                - 101.411x (-1 + x) (-4/5 + x) (-3/5 + x) (-1/5 + x)

                + 147.8423x (-1 + x) (-4/5 + x) (-2/5 + x) (-1/5 + x)

                - 93.48574x (-1 + x) (-3/5 + x) (-2/5 + x) (-1/5 + x)

                + 21.91331x (-4/5 + x) (-3/5 + x) (-2/5 + x) (-1/5 + x)

#I[4]::  Ex[%]

                                    2              3              4
#O[4]:   .999978x + .0002439154 x  - .1676164 x  + .001612961 x

                              5
                + .007252478 x

#I[5]::  N[S[%,x->0.8]]

#O[5]:   .7173561

#I[6]::  N[Sin[0.8]]

#O[6]:   .7173561

*/
```

XJac

```
                    /** Jacobi polynomials **/

                     /* Recurrence relationships */

SJac_Ldist

SJac[1]:   JacP[$m,$a,$b,$x] -> (((2$m+$a+$b-1) ($a^2-$b^2)\
             +Poch[2$m+$a+$b-2,3] $x) JacP[$m-1,$a,$b,$x]\
             -2($m+$a-1) ($m+$b-1) (2$m+$a+$b)\
              JacP[$m-2,$a,$b,$x])/(2$m ($m+$a+$b) (2$m+$a+$b+2))

SJac[2]:   JacP[$m,$a,$b,$x] -> (2($m+1) ($m+$a+$b+1) (2$m+$a+$b)\
              JacP[$m+1,$a,$b,$x]+2($n+$a) ($n+$b) (2$n+$a+$b+2)\
              JacP[$m-1,$a,$b,$x])/((2$n+$a+$b+1) ($a^2-$b^2)\
             +Poch[2$n+$a+$b,3] $x)

SJac[3]:   JacP[$m,$a,$b,$x] -> (((2$m+$a+4b+3) ($a^2-$b^2)\
             +Poch[2$m+$a+$b+2,3] $x) JacP[$m+1,$a,$b,$x]\
             -2($m+2) ($m+$a+$b+3) (2$m+$a+$b+2) JacP[$m+2,$a,$b,$x])\
             /(2($m+$a+1) ($m+$b+1) (2$m+$a+$b+4))


SJac[4]:   JacP[$m,$a,$b,$x] -> (($m+$a) JacP[$m,$a-1,$b,$x]-($m+1)\
              JacP[$m+1,$a-1,$b,$x])/(($m+$a/2+$b/2+1/2) (1-$x))

SJac[5]:   JacP[$m,$a,$b,$x] -> (($m+$a/2+$b/2+1) (1-$x) JacP[$m,$a+1,$b,$x]\
             +($m+1) JacP[$m+1,$a,$b,$x])/($m+$a+1)

SJac[6]:   JacP[$m,$a,$b,$x] -> (($m+$a) JacP[$m-1,$a,$b,$x] \
             -($m+$a/2+$b/2) (1-$x) JacP[$m-1,$a+1,$b,$x])/$m


SJac[7]:   JacP[$m,$a,$b,$x] -> (($m+$b) JacP[$m,$a,$b-1,$x]+($m+1)\
              JacP[$m+1,$a,$b-1,$x])/(($m+$a/2+$b/2+1/2) (1+$x))

SJac[8]:   JacP[$m,$a,$b,$x] -> (($m+$a/2+$b/2+1) (1+$x) JacP[$m,$a,$b+1,$x]\
             -($m+1) JacP[$m+1,$a,$b,$x])/($m+$b+1)

SJac[9]:   JacP[$m,$a,$b,$x] -> (($m+$a/2+$b/2) (1+$x) JacP[$m-1,$a,$b+1,$x]\
             -($m+$b) JacP[$m-1,$a,$b,$x])/$m


SJac[10]:  JacP[$m,$a,$b,$x] -> \
             (2JacP[$m,$a-1,$b,$x]-(1+$x) JacP[$m,$a-1,$b+1,$x])/(1-$x)

SJac[11]:  JacP[$m,$a,$b,$x] -> \
             ((1-$x) JacP[$m,$a+1,$b,$x]+(1+$x) JacP[$m,$a,$b+1,$x])/2

SJac[12]:  JacP[$m,$a,$b,$x] -> \
             (2JacP[$m,$a,$b-1,$x]-(1-$x) JacP[$m,$a+1,$b-1,$x])/(1+$x)


SJac[13]:  JacP[$m,$a,$b,$x] -> (($m+$a+$b+1) JacP[$m,$a+1,$b,$x]\
             -($m+$b) JacP[$m-1,$a+1,$b,$x])/(2$m+$a+$b+1)

SJac[14]:  JacP[$m,$a,$b,$x] -> ((2$m+$a+$b) JacP[$m,$a-1,$b,$x]\
             +($m+$b) JacP[$m-1,$a,$b,$x])/($m+$a+$b)

SJac[15]:  JacP[$m,$a,$b,$x] -> (($m+$a+$b+1) JacP[$m+1,$a,$b,$x]\
             -(2$m+$a+$b+2) JacP[$m+1,$a-1,$b,$x])/(2$m+$a+$b+2)


SJac[16]:  JacP[$m,$a,$b,$x] -> (($m+$a+$b+1) JacP[$m,$a,$b+1,$x]\
             +($m+$a) JacP[$m-1,$a,$b+1,$x])/(2$m+$a+$b+1)

SJac[17]:  JacP[$m,$a,$b,$x] -> ((2$m+$a+$b) JacP[$m,$a,$b-1,$x]\
             -($m+$a) JacP[$m-1,$a,$b,$x])/($m+$a+$b)

SJac[18]:  JacP[$m,$a,$b,$x] -> ((2$m+$a+$b+2) JacP[$m+1,$a,$b-1,$x]\
             -($m+$a+$b+1) JacP[$m+1,$a,$b,$x])/(2$m+$a+$b+2)
```

```
SJac[19]:  JacP[$m,$a,$b,$x] -> JacP[$m-1,$a,$b+1,$x]-JacP[$m,$a-1,$b+1,$x]

SJac[20]:  JacP[$m,$a,$b,$x] -> JacP[$m+1,$a,$b-1,$x]-JacP[$m+1,$a-1,$b,$x]

SJac[21]:  JacP[$m,$a,$b,$x] -> JacP[$m,$a+1,$b-1,$x]-JacP[$m-1,$a+1,$b,$x]
```

XKillIO

```
        /**  Kill Input/Output  **/

/* KillIO[n1,n2]
        removes values assigned to #I and #0 lines numbered n1
        through n2. */
        KillIO[$n1_>Natp[$n1],$n2_>Natp[$n2]] ::  \
                        Ar[{{$n1,$n2}},#I[$%1]:#0[$%1]:Killed];

/*
#I[1]::  <XKillIO

#I[2]::  t:x

#0[2]:   x

#I[3]::  Rpt[t:t(1+t),2]

#0[3]:   x (1 + x) (1 + x (1 + x))

#I[4]::  Ex[%]

                  2      3      4
#0[4]:   x + 2 x    + 2 x  + x

#I[5]::  Fac[%]

                                 2
#0[5]:   x (1 + x) (1 + x + x )

#I[6]::  KillIO[2,4];

#I[7]::  #0

                                                  2
#0[7]:   {[6]: , [5]: x (1 + x) (1 + x + x ), [4]: Killed, [3]: Killed,

              [2]: Killed, [1]: 1Null, [0]: {"/u1/smp/SRC/smp.start"}}

#I[8]::  #I

#0[8]:   {[7]:: #0, [6]:: KillIO[2,4] ; , [5]:: Fac[%], [4]: Killed,

              [3]: Killed, [2]: Killed, [1]:: <XKillIO, [0]:: Init}

*/
```

XJac

```
                    /** Jacobi polynomials **/

                      /* Recurrence relationships */

SJac_LDist

SJac[1]:    JacP[$m,$a,$b,$x] -> (((2$m+$a+$b-1) ($a^2-$b^2)\
                  +Poch[2$m+$a+$b-2,3] $x) JacP[$m-1,$a,$b,$x]\
                  -2($m+$a-1) ($m+$b-1) (2$m+$a+$b)\
                   JacP[$m-2,$a,$b,$x])/(2$m ($m+$a+$b) (2$m+$a+$b+2))

SJac[2]:    JacP[$m,$a,$b,$x] -> (2($m+1) ($m+$a+$b+1) (2$m+$a+$b)\
                   JacP[$m+1,$a,$b,$x]+2($n+$a) ($n+$b) (2$n+$a+$b+2)\
                   JacP[$m-1,$a,$b,$x])/((2$n+$a+$b+1) ($a^2-$b^2)\
                  +Poch[2$n+$a+$b,3] $x)

SJac[3]:    JacP[$m,$a,$b,$x] -> (((2$m+$a+4b+3) ($a^2-$b^2)\
                  +Poch[2$m+$a+$b+2,3] $x) JacP[$m+1,$a,$b,$x]\
                  -2($m+2) ($m+$a+$b+3) (2$m+$a+$b+2) JacP[$m+2,$a,$b,$x])\
                  /(2($m+$a+1) ($m+$b+1) (2$m+$a+$b+4))


SJac[4]:    JacP[$m,$a,$b,$x] -> (($m+$a) JacP[$m,$a-1,$b,$x]-($m+1)\
                   JacP[$m+1,$a-1,$b,$x])/(($m+$a/2+$b/2+1/2) (1-$x))

SJac[5]:    JacP[$m,$a,$b,$x] -> (($m+$a/2+$b/2+1) (1-$x) JacP[$m,$a+1,$b,$x]\
                  +($m+1) JacP[$m+1,$a,$b,$x])/($m+$a+1)

SJac[8]:    JacP[$m,$a,$b,$x] -> (($m+$a) JacP[$m-1,$a,$b,$x] \
                  -($m+$a/2+$b/2) (1-$x) JacP[$m-1,$a+1,$b,$x])/$m


SJac[7]:    JacP[$m,$a,$b,$x] -> (($m+$b) JacP[$m,$a,$b-1,$x]+($m+1)\
                   JacP[$m+1,$a,$b-1,$x])/(($m+$a/2+$b/2+1/2) (1+$x))

SJac[8]:    JacP[$m,$a,$b,$x] -> (($m+$a/2+$b/2+1) (1+$x) JacP[$m,$a,$b+1,$x]\
                  -($m+1) JacP[$m+1,$a,$b,$x])/($m+$b+1)

SJac[9]:    JacP[$m,$a,$b,$x] -> (($m+$a/2+$b/2) (1+$x) JacP[$m-1,$a,$b+1,$x]\
                  -($m+$b) JacP[$m-1,$a,$b,$x])/$m


SJac[18]:   JacP[$m,$a,$b,$x] -> \
                  (2JacP[$m,$a-1,$b,$x]-(1+$x) JacP[$m,$a-1,$b+1,$x])/(1-$x)

SJac[11]:   JacP[$m,$a,$b,$x] -> \
                  ((1-$x) JacP[$m,$a+1,$b,$x]+(1+$x) JacP[$m,$a,$b+1,$x])/2

SJac[12]:   JacP[$m,$a,$b,$x] -> \
                  (2JacP[$m,$a,$b-1,$x]-(1-$x) JacP[$m,$a+1,$b-1,$x])/(1+$x)


SJac[13]:   JacP[$m,$a,$b,$x] -> (($m+$a+$b+1) JacP[$m,$a+1,$b,$x]\
                  -($m+$b) JacP[$m-1,$a+1,$b,$x])/(2$m+$a+$b+1)

SJac[14]:   JacP[$m,$a,$b,$x] -> ((2$m+$a+$b) JacP[$m,$a-1,$b,$x]\
                  +($m+$b) JacP[$m-1,$a,$b,$x])/($m+$a+$b)

SJac[15]:   JacP[$m,$a,$b,$x] -> (($m+$a+$b+1) JacP[$m+1,$a,$b,$x]\
                  -(2$m+$a+$b+2) JacP[$m+1,$a-1,$b,$x])/(2$m+$a+$b+2)


SJac[16]:   JacP[$m,$a,$b,$x] -> (($m+$a+$b+1) JacP[$m,$a,$b+1,$x]\
                  +($m+$a) JacP[$m-1,$a,$b+1,$x])/(2$m+$a+$b+1)

SJac[17]:   JacP[$m,$a,$b,$x] -> ((2$m+$a+$b) JacP[$m,$a,$b-1,$x]\
                  -($m+$a) JacP[$m-1,$a,$b,$x])/($m+$a+$b)

SJac[18]:   JacP[$m,$a,$b,$x] -> ((2$m+$a+$b+2) JacP[$m+1,$a,$b-1,$x]\
                  -($m+$a+$b+1) JacP[$m+1,$a,$b,$x])/(2$m+$a+$b+2)
```

```
SJac[19]:  JacP[$m,$a,$b,$x] -> JacP[$m-1,$a,$b+1,$x]-JacP[$m,$a-1,$b+1,$x]

SJac[20]:  JacP[$m,$a,$b,$x] -> JacP[$m+1,$a,$b-1,$x]-JacP[$m+1,$a-1,$b,$x]

SJac[21]:  JacP[$m,$a,$b,$x] -> JacP[$m,$a+1,$b-1,$x]-JacP[$m-1,$a+1,$b,$x]
```

XKillIO

```
        /** Kill Input/Output **/

/* KillIO[n1,n2]
        removes values assigned to #I and #O lines numbered n1
        through n2. */
        KillIO[$n1_:Natp[$n1],$n2_:Natp[$n2]] :: \
                        Ar[{{$n1,$n2}},#I[$%1]:#O[$%1]:Killed];

/*
#I[1]::  <XKillIO

#I[2]::  t:x

#O[2]:   x

#I[3]::  Rpt[t:t(1+t),2]

#O[3]:   x (1 + x) (1 + x (1 + x))

#I[4]::  Ex[%]

                  2      3    4
#O[4]:   x + 2 x   + 2 x   + x

#I[5]::  Fac[%]

                                 2
#O[5]:   x (1 + x) (1 + x + x )

#I[6]::  KillIO[2,4];

#I[7]::  #O

                                                  2
#O[7]:   {[6]: , [5]: x (1 + x) (1 + x + x ), [4]: Killed, [3]: Killed,

              [2]: Killed, [1]: 1Null, [0]: {"/u1/smp/SRC/smp.start"}}

#I[8]::  #I

#O[8]:   {[7]:: #O, [6]:: KillIO[2,4] ; , [5]:: Fac[%], [4]: Killed,

              [3]: Killed, [2]: Killed, [1]:: <XKillIO, [0]:: Init}

*/
```

XKumU

```
                    /** Kummer U function **/

SKumU_:Ldist

SKumU[1]:   KumU[$a,$b,$z] -> z^(1-$b)KumU[1+$a-$b,2-$b,$z]

SKumU[2]:   KumU[$a,$b,$z] -> Pi/Sin[Pi $b](Chg[$a,$b,$z]/(Gamma[1+$a-$b]
                  Gamma[$b])-$z^(1-$b)Kumg[1+$a-$b,2-$b,$z]/(Gamma[$a]\
                              Gamma[2-$b]))

SKumU[3] : KumU[$a,2$a,$z] -> $z^(1/2-$a) Exp[$z/2] BesK[$a-1/2,$z/2]/Sqrt[Pi]

SKumU[4] : KumU[$a,2$a,$z] -> Sqrt [Pi] (I $z)^(1/2-$a) Exp[$z/2+$a I Pi] \
                                    BesH1[$a-1/2,I $z/2] / 2

SKumU[5] : KumU[$a,2$a,$z] -> Sqrt [Pi] (I $z)^(1/2-$a) Exp[$z/2-$a I Pi] \
                                    BesH2[$a-1/2,-I $z/2] / 2

SKumU[6] : KumU[$a,2$a,$z] -> $z^(1/2-$a) Exp[$z/2] BesK[$a-1/2,$z/2]

SKumU[7] : KumU[5/6,5/3,$z] -> 3^(1/6) (2/z)^(2/3) Sqrt[Pi] Exp[$z/2] \
                                    AirAi[(3$z/4)^(2/3)]

SKumU[8] : KumU[$a,2$a,$z] -> I^($a-1/2) (2$z)^(1/2-$a) Exp[$z] \
                              Kelke[$a-1/2,Sqrt[2] Re[z]] / Sqrt[Pi]

SKumU[9] : KumU[$a,$b,$x] -> (-1)^$a (-$a)!*Lag[-$a,$b-1,$x]

SKumU[10] : KumU[$a,$a,$x] -> Exp[$x] Gamma[$x,1-$a]

SKumU[11] : KumU[1,1,$x] -> -Exp[$x] Ei[-$x]

SKumU[12] : KumU[1,1,$x] -> Exp[$x] Expi[$x]

SKumU[13] : KumU[1,1,$x] -> -Exp[$x] Logi[Exp[-$x]]

SKumU[14] : KumU[$a,$b,$x] -> \
                  Gamma[1-2$a] Exp[$x-2I $a Pi] Cuno[($b-2$a-1)/2,$b-1,$x]

SKumU[15] : KumU[$a,0,$x] -> Gamma[1-$a] Exp[$x] Batk[-2$a,$x]

SKumU[16] : KumU[1,1,$y] -> -Exp[$y] (Cosi[I $y]+I(Sini[I $y]-Pi/2))

SKumU[15] : KumU[$a,1/2,$z] -> 2^$a Exp[$z/2] Par[-2$a,Sqrt[2$z]]

SKumU[16] : KumU[$a,3/2,$z] -> 2^$a Exp[$z/2] Par[1-2$a,Sqrt[2$z]]/Sqrt[2z]

SKumU[17] : KumU[$a,3/2,$x] -> 2^(1-2$a) Her[1-2$a,Sqrt[$x]] / Sqrt[$x]

SKumU[18] : KumU[1/2,1/2,$x] -> Sqrt[Pi] Exp[$x] Erfc[Sqrt[$x]]

                                                        /* AS 13.6.21-39 */

SKumU[19] : KumU[$a,$c,$z] -> Exp[$z/2]$z^(-$c/2)WhiW[$c/2-$a,$c/2-1/2,$z]
                                                        /* MOS p.304 */
```

XLArith

```
                    /**  Arithmetic operations on lists  **/

/* LSum[list]
        yields the sum of all elements in the list or set of nested lists
        list. */
        LSum[$list] :: Ap[Plus,Flat[$list]]

/* LProd[list]
        yields the product of elements in list. */
        LProd[$list] :: Ap[Mult,Flat[$list]]

/*
#I[1]::  <XLArith

#I[2]::  {{a,b},{1,2,3},d}

#O[2]:   {{a,b},{1,2,3},d}

#I[3]::  LSum[%]

#O[3]:   6 + a + b + d

#I[4]::  LProd[@2]

#O[4]:   6a b d

*/
```

XLCM

```
                    /**  Lowest common multiple  **/

/* LCM[n1,n2,...]
        yields the lowest common multiple of n1, n2,.... */

        LCM_:Flat
        LCM[$n1,$n2] :: ($n1 $n2)/Gcd[$n1,$n2]
```

XLItp

```
        /**  Interpolation of contiguous list values  **/

/* LItp[list,x]
        uses all the values given in list to find an interpolated value
        for an element with index x. */
        LItp[$list_:Contp[$list],$x] :: \
                (Lc![%n]; %n:Len[$list];  \
                Sum[$list[k+Gint[(%n+1)/2]] (-1)^(Gint[%n/2]+k) \
              /((k+Gint[(%n-1)/2])!*(Gint[%n/2]-k)!*($x-Gint[(%n+1)/2]-k)) \
                        Prod[$x-Oddp[%n]-t,t,Evenp[%n],2Gint[%n/2]],\
                        k,-Gint[(%n-1)/2],Gint[%n/2]])

/* LItp2[list,x]
        uses two-point (linear) interpolation to yield an estimate
```

```
        for an element of list with index x. */
        LItp2[$list,$x_?(1 <= $x <= Len[$list])] :: \
                (Lc![%x]; %x:Gint[$x]; (1-$x+%x) $list[%x]+($x-%x) $list[%x+1])

/* LItp3[list,x]
        uses three-point Lagrange interpolation to estimate the
        value of an element of list with index x. */
        LItp3[$list,$x_?(2 <= $x <= Len[$list]-1)] :: \
                (Lc![%x,%p]; %p:$x-(%x:Gint[$x]); \
                %p(%p-1)/2 $list[%x-1] + (1-%p^2) $list[%x] + \
                        %p(%p+1)/2 $list[%x+1])

/*

#I[1]::  <XLItp

#I[2]::  t:Ar[6,$x^4]

#O[2]:   {1,16,81,256,625,1296}

#I[3]::  LItp[t,3.5]

#O[3]:   150.0625

#I[4]::  3.5^4

#O[4]:   150.0625

#I[5]::  LItp2[t,3.5]

#O[5]:   168.5

#I[6]::  LItp3[t,3.5]

#O[6]:   154.75

#I[7]::  LItp[Ar[3],x]

                (-3 + x) (-2 + x)                          3(-2 + x) (-1 + x)
#O[7]:   ----------------- - 2(-3 + x) (-1 + x) + -------------------
                    2                                        2

#I[8]::  Ex[%]

#O[8]:   x

*/
```

XLPart

```
                /** List positions of all parts **/

/* LPart[expr]
        yields a list of the positions of all subparts of expr. */
        LPart[$expr] :: Map[$1[2],Pos[$1,$expr]]

/*

#I[1]::  <XLPart

#I[2]::  Rex[]

                  2
#O[2]:   12 x  y z (5 + x) (3 + x + 2E y)

#I[3]::  LPart[%]
```

```
#0[3]:   {{1,1},{1,2},{1},{2},{3},{4,1},{4,2},{4},{5,1},{5,2},{5,3,1},{5,3,2},
          {5,3},{5},{0}}

*/
```

XLag

```
                    /** Laguerre polynomials **/

SLag_:Ldist

SLag[1]:   Lag[$m,$a,$x] -> \
           ((2$m+$a-1-$x) Lag[$m-1,$a,$x]-($m+$a-1) Lag[$m-2,$a,$x])/$m

SLag[2]:   Lag[$m,$a,$x] -> \
           (($m+1) Lag[$m+1,$a,$x]+($m+$a) Lag[$m-1,$a,$x])/(2$m+$a+1-$x

SLag[3]:   Lag[$m,$a,$x] -> \
           ((2$m+$a+3) Lag[$m+1,$a,$x]-($m+2) Lag[$m+2,$a,$x])/($m+$a+1)
```

XLap

```
            /** Laplace transforms **/

/* Lap[expr,t,s]
        represents the Laplace transform of expr
        from the t domain to s domain. */

        Lap[$n_:Numbp[$n],$t,$s] : $n/$s

        Lap[$t,$t,$s] : 1/$s^2

        Lap[$t^$p,$t,$s] : Gamma[$p+1]/$s^($p+1)

        Lap[$x + $$x,$t,$s] :: Lap[$x,$t,$s] + Lap[$$x,$t,$s]

        Lap[$x $$x,$t_:~In[$t,$x],$s] :: $x Lap[$$x,$t,$s]

        Lap[$x/$y,$t_:~In[$t,$y],$s] :: Lap[$x,$t,$s]/$y

        Lap[$x=$y,$t,$s] :: Lap[$x,$t,$s]=Lap[$y,$t,$s]

        Lap[D[$y,{$x,$n,$t}],$t,$s] :: \
                $s^$n Lap[$y,$t,$s] - Sum[D[$y,{$x,i,0}] $s^($n-i),i,1,$n]

        Lap[Exp[$$a $t] $$x,$t_:~In[$t,$$a],$s] :: Lap[$$x,$t,$s-$$a]

        Lap[($t+$$a)^$p,$t_:~In[$t,{$$a,$p}],$s] :: \
                        Gamma[$p+1,$$a $s] Exp[$$a $s]/$s^($p+1)
        Lap[Log[$t],$t,$s] :: -Log[Euler $s]/$s

        Lap[$t^$p Log[$t],$t,$s] :: Gamma[$p+1] (Psi[$p+1]-Log[$s])/$s^($p+1)

        Lap[Log[$t]^2,$t,$s] : (Zeta[2] + Log[Euler $s]^2)/$s

        Lap[Sin[$$a $t],$t_:~In[$t,$$a],$s] : $$a/($s^2 + $$a^2)
```

XLatsum

```
                    /**  Lattice sums  **/

/* Latsum[f,spec]
        sums the results of applying the template f to the sets of
        lattice points specified by spec. */
        _Latsum[Nosmp]:{1,0}
        Latsum[$f,$spec] :: Rel[Ap[Plus,Flat[Ar[$spec,$f]]]]

/*
#I[1]::   <XLatsum

#I[2]::   Latsum[1/($1^2+$2^3),{5,4}]

#O[2]:    1.428285

#I[3]::   Sum[Sum[1/(i^2+j^3),j,1,4],i,1,5]

#O[3]:    1.428285

*/
```

XLdEq

```
                    /**  Lists of equations  **/

/* LdEq[expr]
        distributes Eq over lists in expr, thus converting equations
        involving lists into lists of equations. */
        LdEq[$expr] :: Ldist[$expr,'Eq]

/*
#I[1]::   <XLdEq

#I[2]::   {{a,b},{c,d}}.{1,2}={5,-3}

#O[2]:    {a + 2b,c + 2d} = {5,-3}

#I[3]::   LdEq[%]

#O[3]:    {a + 2b = 5,3 + c + 2d = 0}

*/
```

XLegP

```
                    /** Legendre polynomials **/

SLegP_:Ldist

SLegP[1]:   LegP[$m,$x] -> ((2$m-1) $x LegP[$m-1,$x]-($m-1) LegP[$m-2,$x])/$m

SLegP[2]:   LegP[$m,$x] -> \
                  (($m+1) LegP[$m+1,$x]+$m LegP[$m-1,$x])/((2$m+1) $x)

SLegP[3]:   LegP[$m,$x] -> \
                  ((2$m+3) $x LegP[$m+1,$x]-($m+2) LegP[$m+2,$x])/($m+1)
```

XLenex

```
                    /** Length of expanded expressions **/

/* Lenex[expr]
        yields an upper limit on the length of the expression resulting
        from expansion of expr. */
        Lenex[$x+$$x] :: Lenex[$x]+Lenex[$$x]
        Lenex[$x $$x] :: Lenex[$x] Lenex[$$x]
        Lenex[$x^$n_?Natp[$n]] :: Comb[Lenex[$x]+$n-1,$n]
        Lenex[($x $$x)/$y] :: Lenex[$x] Lenex[$$x]
        Lenex[Log[$x $$x]] :: Lenex[$x]+Lenex[$$x]
        Lenex[$x] : 1

/*
#I[1]::  <XLenex

#I[2]::  t:(1+x+x^2)(a+b+1)^3

                      3           2
#O[2]:   (1 + a + b)  (1 + x + x )

#I[3]::  Lenex[%]

#O[3]:   30

#I[4]::  Ex[t]

                                                    2       2           2     2
#O[4]:   1 + 3a + 3b + x + 6a b + 3a x + 3a b  + 3a x  + 3b x + 3b x  + 3 a  b

           2       2  2    3      3  2      2        2  2    3      3  2
  + 3 a  x + 3 a  x  + a  x + a  x  + 3 b  x + 3 b  x  + b  x + b  x

                        2       2         2  2      2         2    2
  + 6a b x + 6a b x  + 3a b  x + 3a b  x  + 3 a  b x + 3 a  b x

       2    3      2    3    2
  + 3 a  + a  + 3 b  + b  + x

#I[5]::  Len[%]

#O[5]:   30

#I[6]::  Lenex[t^5]

#O[6]:   2856

*/
```

XLer

```
                    /** Lerch transcendent **/

SLer[1]:    Ler[1,$s,1] -> Zeta[$s]

SLer[2]:    Ler[1,$s,$a] -> Zeta[$s,$a]

SLer[3]:    Ler[$z,$s,1] -> Li[$s,$z]/$z

SLer[4]:    Ler[$z,$s,$a] -> I $z~$a Gamma[1-$s](2Pi)^($s-1)\
                  (Exp[-I Pi $s/2]Ler[Exp[-2 Pi I $a],1-$s,Log[$z]/(2Pi I)] \
                  -Exp[I Pi($s/2+2$a)]Ler[Exp[2Pi I $a],1-$s,1-Log[$z]/(2Pi I))
                                                    /* MOS pp. 33-4 */
```

XLev

```
                    /** Isolate single level **/

/* Lev[expr,n]
        yields a list of parts of expr on level n. */
        Lev[$expr,$n] :: (Lcl[%l]; %l:{}; Map[%l:Cat[%l,{$1}],$expr,{$n}]; %l)

/* Lenlev[expr,n]
        finds the number of parts of expr at level n. */
        Lenlev[$expr,$n] :: (Lcl[%n]; %n:0; Map[`Inc[%n],$expr,{$n}]; %n)

/*

#I[1]::  <XLev

#I[2]::  t:Rex[]

                 2
#O[2]:   56 x  (x + z)

#I[3]::  Lev[%,1]

           2
#O[3]:   {x ,x + z}

#I[4]::  Lev[@2,2]

#O[4]:   {x,2,x,z}

#I[5]::  Lenlev[@2,2]

#O[5]:   4

*/
```

XLevi

```
                    /** Generate Levi-Civita tensor **/

/* Levi[n]
        yields the Levi-Civita totally antisymmetric epsilon tensor
        in n dimensions. */
        Levi[$n_?Natp[$n]] :: Ar[Ar[$n,$n],Sig]
```

XList0

```
                    /** Basic list manipulations **/

/* First[list]
        yields the first entry in list. */
        First[$list_?Listp[$list]] :: Elem[$list,{1}]

/* Prep[elem,list]
        prepends elem to the beginning of list. */
        Prep[$elem,$list_?Listp[$list]] :: Cat[{$elem},$list]

/* App[elem,list]
        appends elem to the end of list. */
        App[$elem,$list_?Listp[$list]] :: Cat[$list,{$elem}]

/* Tk[n,list]
        takes the first n or last -n entries in list. */
        Tk[$n_?Natp[$n],$list_?Listp[$list]] :: Ar[$n,$list]
        Tk[$n_?Natp[-$n],$list_?Listp[$list]] :: \
                        Cat[Ar[{Len[$list]+{$n+1,0}},$list]]

/* Ins[elem,list,i]
        inserts the entry elem into list at position i. */
        Ins[$elem,$list_?Contp[$list],$n_?Natp[$n+1]] :: \
                Cat[Ar[$n-1,$list],{$elem},Ar[{{$n,Len[$list]}},$list]]

/* Rm[list,n]
        removes the n th entry from list. */
        Rm[$list_?Listp[$list],$n_?Natp[$n]] :: Cat[Ar[$n-1,$list], \
                Ar[{{$n+1,Len[$list]}},$list]]
        Rm[$list_?Contp[$list],$n_?Natp[$n]] :: \
                Cat[Ar[Len[$list],$list,$%1~=$n]]


/* Lrpt[list,n]
        yields a list consisting of n repetitions of list. */
        Lrpt[$list_?Contp[$list],$n_?Natp[$n]] :: \
                Cat[Rep![$list,$n]]

/*

#I[1]::  <XList0

#I[2]::  t:{a,b,c,d,e,f,g}

#O[2]:   {a,b,c,d,e,f,g}

#I[3]::  First[t]

#O[3]:   a

#I[4]::  Last[t]
```

```
#O[4]:   g

#I[5]::  Prep[1,t]

#O[5]:   {1,a,b,c,d,e,f,g}

#I[6]::  App[1,t]

#O[6]:   {a,b,c,d,e,f,g,1}

#I[7]::  Tk[4,t]

#O[7]:   {a,b,c,d}

#I[8]::  Tk[-4,t]

#O[8]:   {d,e,f,g}

#I[9]::  Ins[1,t,4]

#O[9]:   {a,b,c,1,d,e,f,g}

#I[10]::  Rm[t,4]

#O[10]:   {a,b,c,e,f,g}

#I[11]::  Lrpt[{a,b,c},4]

#O[11]:   {a,b,c,a,b,c,a,b,c,a,b,c}

*/
```

XList1

```
                    /** Operations on sublists **/

/* LPos[sub,list]
        find positions at which the sublist sub appears in list. */
        LPos[$sub_:Contp[$sub],$list_:Contp[$list]] :: \
                (Lc![%1,%2]; %1:{}; Do[%1,Len[$list]-Len[$sub]+1, \
                For[%2:0, P[$sub[%2+1]=$list[%1+%2]] & %2<Len[$sub], \
                Inc[%2],]; If[%2=Len[$sub],%1:Cat[%1,{%1}]]]; %1)
        LPos[$sub_:(Contp[$sub] & Len[$sub]=1),$list_:Contp[$list]] :: \
                Flat[Pos[$sub[1],$list,2]]

/* LSub[list2,list1,list]
        substitutes list2 for all occurrences of the sublist list1 in list. */
        LSub[$list2,$list1,$list] :: (Lc![%f]; %f_:Flat; \
        S[S[Ap[%f,$list],Ap[%f,$list1]->Ap[%f,$list2],1,Inf],%f->List])

/* LS[list,rep1,rep2,...]
        applies successively  the replacements repi specified for
        sublists in list. (The repi may contain patterns.) */
        LS[$list,$$reps] :: (Lc![%1]; %1:$list; \
        Map[%1:LSub[$1[2],$1[1],%1],List[$$reps],1,  \
                        ($2[0]='Rep) | ($2[0]='Repd)]; %1)

/* LIn[list1,list]
        yields 1 if list1 is a sublist of list, and 0 otherwise. */
        LIn[$list1,$list] :: (Lc![%f]; %f_:Flat; ~P[~Match[Ap[%f, \
                        Cat[{$$%1},$list1,{$$%2}]],Ap[%f,$list]]])

/*

#I[1]::  <XList1
```

```
#I[2]::  t:{a,b,c,a,b,d,e,a,c}
#O[2]:   {a,b,c,a,b,d,e,a,c}
#I[3]::  LPos[{a,b},t]
#O[3]:   {1,4}
#I[4]::  LSub[{g,h,i},{a,b},t]
#O[4]:   {g,h,i,c,g,h,i,d,e,a,c}
#I[5]::  LS[t,{a,b}->{g,h,i},{e}->{r,s}]
#O[5]:   {g,h,i,c,g,h,i,d,r,s,a,c}
#I[6]::  LIn[{a,b},t]
#O[6]:   1
#I[7]::  LIn[{a,d},t]
#O[7]:   0
*/
```

XLogic

```
        /** Elementary laws in propositional calculus **/

/* Note p=q represents the logical biconditional "if and only if p then q" */

/* Idempotent laws */
$p | $p : $p
$p & $p : $p

/* Commutative and associative laws built in */

/* Distributive laws */
$p | ($$q & $r) : ($p | $$q) & ($p | $r)

/* Identity laws built in */

/* Complement laws */
~~$p : $p
$p | ~$p : 1
$p & ~$p : 0

/* DeMorgan's laws */
~($$p | $q) : (~$$p) & (~$q)
~($$p & $q) : (~$$p) | (~$q)

/* Reflexive law */
$p => $p : 1

/* Antisymmetric law */
($p => $q) & ($q => $p) : $p=$q

/* Transitive law */
($p => $q) & ($q => $r) : $p=>$r
```

XLogic2

```
          /** Elementary logic with quantifiers **/

/* Quant[q1,q2,...,stat]
        represents the statement stat subject to the quantifiers
        q1, q2, ... */

/* All[x]
        represents the quantifier "for all x". */

/* Some[x]
        represents the quantifier "for some x" or "there exists an x
        such that". */

Quant[$s] : $s
Quant[$$q,Quant[$$r]] : Quant[$$q,$$r]

/* DeMorgan's law */
~Quant[All[$x],$$q,$s] : Quant[Some[$x],~Quant[$$q,$s]]
~Quant[Some[$x],$$q,$s] : Quant[All[$x],~Quant[$$q,$s]]
```

XLogicPr

```
          /** Logical truth tables **/

/* PrTF[expr]
        prints a table giving the values of the logical expression
        expr for all possible truth values of symbols appearing in it. */
        PrTF[$expr] :: (Lcl[%p]; Ap[Pr,Cat[%p:Cont[$expr],{$expr}]]; \
                        Ar[Ar[Len[%p],`{{0,1}}], \
                Ap[Pr,S[Cat[%p,{$expr}],Ldist[%p->List[$$1]]]]])

/*
#I[1]::  <XLogicPr

#I[2]::  PrTF[(p&r)=>(q|~p)]

p       q       r       p & r => q | ~ p
0       0       0       1
0       0       1       1
0       1       0       1
0       1       1       1
1       0       0       1
1       0       1       0
1       1       0       1
1       1       1       1

#O[2]:   {[0]: {[0]: {[0]: 1, [1]: 1}, [1]: {[0]: 1, [1]: 1}},
                {[1]: {[0]: {[0]: 1, [1]: 0}, [1]: {[0]: 1, [1]: 1}}}

*/
```

XLom

```
                                /** Lommel function **/

    SLom_Ldist

    SLom[1]: Lom[$m,$n,$z] -> $z^($m-1) - (($m-1)^2 - $n^2) Lom[$m-2,$n,$z]
                                                        /* MOS p. 109 */

    SLom[2]: Lom[$m,$n,$z] -> ($z^($m+1)-Lom[$m+2,$n,$z])/(($m+1)^2 - $n^2)
                                                        /* MOS p. 109 */

    SLom[3]: Lom[$m,$n,$z] -> $z/2/$n (($m + $v - 1) Lom[$m-1,Sn-1,$z] - \
                                ($m - $n - 1) Lom[$m-1,$n+1,$z])
                                                        /* MOS p. 109 */

    SLom[4]: Lom[$m,$n,$z] -> ($m+$n+1)^(-1) (2 $n/$z Lom[$m+1,$n+1,$z] - \
                                ($m - $n + 2) Lom[$m,$n+3,$z])
                                                        /* MOS p. 109 */

    SLom[5]: Lom[$m,$n,$z] -> (($m + $n -1) Lom[$m,$n-2,$z] -\
                                2($n-1)/$z Lom[$m+1,$n-1,$z])/($m-$n+1)
                                                        /* MOS p. 109 */

                        /* Special cases of Lommel's functions */

    SLom[6]: Lom[$n,$n,$z] -> Pi^(1/2) 2^($n-1) Gamma[1/2+$n] StrH[$n,$z]
                                                        /* MOS p. 109 */

    /* LomS[$m,$n,$z]
            Generalised Lommel function [MOS p. 109] */

    LomS[$m,$n,$z]: Lom[$m,$n,$z] + 2^($m-1) Gamma[($m-$n+1)/2] Gamma[($m+$n+1)/2]
                                                        /* MOS p. 109 */

    SLom[7]: LomS[$n,$n,$z] -> Pi^(1/2) 2^($n - 1) Gamma[1/2 + Sn]\
                                (StrH[$n,$z] - BesY[$n,$z])
                                                        /* MOS p. 110 */

    SLom[8]: Lom[$n,$n,$z] -> LomS[$n,$n,$z] + Pi^(1/2) 2^($n-1)
                                Gamma[1/2 + $n] BesY[$n,$z]
                                                        /* MOS p. 110 */

    SLom[9]: LomS[$n,$n,$z] -> Lom[$n,$n,$z] - Pi^(1/2) 2^($n-1)
                                Gamma[1/2 + $n] BesY[$n,$z]
                                                        /* MOS p. 110 */

    SLom[10]:           Lom[0,$n,$z] ->\
                                Pi/2 /Sin[Pi $n] (AngJ[$n,$z] - AngJ[-$n,$z])
                                                        /* MOS p. 109 */
    SLom[11]:           LomS[0,$n,$z] -> Pi/(2Sin[Pi $n])(AngJ[$n,$z]-AngJ[-$n,$z]\
                                -BesJ[$n,$z] + Bes[-$n,$z])
                                                        /* MOS p. 110 */

SLom[12]:         Lom[-1,$n,$z] -> -Pi/(2$n Sin[Pi $n])(AngJ[$n,$z]+AngJ[-$n,$z])
                                                        /* MOS p. 110 */

SLom[13]:         LomS[-1,$n,$z] -> Pi/(2$n Sin[Pi $n])(BesJ[$n,$z]+BesJ[-$n,$z]\
                                -AngJ[$n,$z] - AngJ[-$n,$z])
                                                        /* MOS p. 110 */

    SLom[14]:           Lom[1,$n,$z] -> 1 + $n^2 Lom[-1,$n,$z]
                                                        /* MOS p. 110 */

    SLom[15]:           LomS[1,$n,$z] -> 1 + $n^2 LomS[-1,$n,$z]
                                                        /* MOS p. 110 */

    SLom[16]:           Lom[0,1/2,$z] -> (2 Pi/$z)^(1/2) (Sin[$z] FreC[$z] - \
                                Cos[$z] FreS[$z])
                                                        /* MOS p. 110  */

    SLom[17]:           LomS[0,1/2,$z] -> (2 Pi/$z)^(1/2) (Cos[$z] (1/2 -\
```

```
                        FreS[$z]) - Sin[$z] (1/2 - FreC[$z]))
                                                    /* MOS p. 110 */

    SLom[18]:       Lom[-1,1/2,$z] -> 2 (2 Pi/$z)^(1/2) (Sin[$z] FreS[$z] +\
                        Cos[$z] FreC[$z])
                                                    /* MOS p. 110 */

SLom[19]:       LomS[-1,1/2,$z] -> 2 (2 Pi/$z)^(1/2) (Cos[$z](1/2 - FreC[$z])\
                        + Sin[$z] (1/2 - FreS[$z]))
                                                    /* MOS p. 110 */

    SLom[20]:       LomS[1/2,1/2,$z] -> z^(-1/2)
                                                    /* MOS p. 110 */

    SLom[21]:       LomS[3/2,1/2,$z] -> $z^(1/2)
                                                    /* MOS p. 110 */

    SLom[22]:       LomS[0,-1,$z] -> 1/$z
                                                    /* MOS p. 110 */

    SLom[23]:       LomS[-1/2,1/2,$z] -> $z^(-1/2) (Sin$z Cosi[$z] - \
                        Cos[$z] (-Pi/2 + Sini[$z]))
                                                    /* MOS p. 110 */

    SLom[24]:       LomS[-3/2,1/2,$z] -> -$z^(-1/2)(Sin[$z](-Pi/2 + Sini[$z])\
                        +Cos[$z] Cosi[$z])
                                                    /* MOS p. 110 */
```

XLor

```
                    /**  Lorentz vectors  **/

    /* Ldot[list1,list2]
            forms the contraction of two Lorentz vectors with a metric
            of signature ---+ */
            Ldot[$list1,$list2] :: {-1,-1,-1,1}.($list1 $list2)
```

XMKS

```
                    /**  MKS/SI units  **/

    <XDim

    /* Units for fundamental dimensions */

    SSI[0,1] :          length -> metre

    SSI[0,2] :          mass -> kilogram

    SSI[0,3] :          time -> second

    SSI[0,4] :          current -> ampere

    SSI[0,5] :          temperature -> kelvin
```

```
SSI[0,6] :        intensity -> candela
SSI[0,7] :        amount -> mole

/* fundamental units */
SSI[1,1] :        metre -> length
SSI[1,2] :        kilogram -> mass
SSI[1,3] :        second -> time
SSI[1,4] :        ampere -> current
SSI[1,5] :        kelvin -> temperature
SSI[1,6] :        candela -> intensity
SSI[1,7] :        mole -> amount

/* derived units */
SSI[2,1] :        hertz -> frequency
SSI[2,2] :        newton -> force
SSI[2,3] :        pascal -> pressure
SSI[2,4] :        joule -> energy
SSI[2,5] :        watt -> power
SSI[2,6] :        coulomb -> charge
SSI[2,7] :        volt -> voltage
SSI[2,8] :        ohm -> resistance
SSI[2,9] :        siemens -> conductance
SSI[2,10] :       mho -> conductance
SSI[2,11] :       farad -> capacitance
SSI[2,12] :       weber -> flux
SSI[2,13] :       tesla -> weber/area
SSI[2,14] :       henry -> inductance
SSI[2,15] :       gray -> dose
SSI[2,16] :       becquerel -> activity
SSI[2,17] :       lumen -> intensity steradian
SSI[2,18] :       lux -> illuminance
/* multipliers */
SSI[2,1] :        exa -> 1*^18
SSI[2,2] :        peta -> 1*^15
SSI[2,3] :        tera -> 1*^12
SSI[2,4] :        giga -> 1*^9
SSI[2,5] :        mega -> 1*^6
```

```
SSI[2,6] :      kilo -> 1*^3

SSI[2,7] :      hecto -> 1*^2

SSI[2,8] :      deca -> 10

SSI[2,9] :      deci -> 0.1

SSI[2,10] :     centi -> 1*^-2

SSI[2,11] :     milli -> 1*^-3

SSI[2,12] :     micro -> 1*^-6

SSI[2,13] :     nano -> 1*^-9

SSI[2,14] :     pico -> 1*^-12

SSI[2,15] :     femto -> 1*^-15

SSI[2,16] :     atto -> 1*^-18
```

XMSet

```
                /**  Automatic memo definition  **/

/* expr ::: val
        assigns the value val to the projection expr in such a way
        as to store explicitly each form of expr requested. */
        MSet_:Nosmp
        _MSet[Pr][$expr,$val] :: Sx[" ::: ",{$expr,$v},4]
        Sxset[":::",MSet,4]
        $expr ::: $val :: $expr .: $expr : Rel[$val]

/*
#I[1]::  <XMSet

#I[2]::  f[0]:f[1]:1

#O[2]:   1

#I[3]::  f[$x]:::$x f[$x-2]

#O[3]:   ' f[$x] : Rel[$x f[$x - 2]]

#I[4]::  f

#O[4]:   {[0]: 1, [1]: 1, [' $x]:: (' f[$x]) : Rel[' $x f[$x - 2]]}

#I[5]::  f[6]

#O[5]:   48

#I[6]::  f

#O[6]:   {[6]: 48, [4]: 8, [2]: 2, [0]: 1, [1]: 1,
                [' $x]:: (' f[$x]) : Rel[' $x f[$x - 2]]}

#I[7]::  f[11]

#O[7]:   10395

#I[8]::  f
```

```
#O[8]:   {[11]: 10395, [9]: 945, [7]: 105, [5]: 15, [3]: 3, [6]: 48, [4]: 8,
          [2]: 2, [0]: 1, [1]: 1,
          [' $x]:: (' f[$x]) : Re![' $x f[$x - 2]]}
*/
```

XMat1

```
                    /**  Matrix input and generation  **/

/* MRd[n,m]
        reads in turn each entry of an n * m matrix. */
        MRd[$n,$m] :: Ar[{$n,$m},Pr[{$1,$2},":"];Rd[]]

/* Diag[list]
        yields a matrix with diagonal list and all other entries zero. */
        Diag[$list_:Contp[$list]] :: Ar[Ar[2,`Len[$list]], \
                                If[$%1=$%2,$list[$%1],0]]

/*

#I[1]::  <XMat1

#I[2]::  MRd[2,2]

{1,1}   : a1
{1,2}   : a1
{2,1}   : b1
{2,2}   : b2

#O[2]:   {{a1,a2},{b1,b2}}

#I[3]::  Diag[{3,2,a}]

#O[3]:   {{3,0,0},{0,2,0},{0,0,a}}

*/
```

XMat2

```
                    /**  Structural matrix operations  **/

<XMat3

/* LDiag[mat]
        yields a list of the elements on the leading diagonal of mat. */
        LDiag[$mat_:Matp[$mat]] :: Ar[Min[Dim[$mat][1],Dim[$mat][2]], \
                                        $mat[$%1,$%1]]

/* TDiag[mat]
        yields a list of the elements on the trailing diagonal of mat. */
        TDiag[$mat_:Matp[$mat]] :: Ar[Min[Dim[$mat][1],Dim[$mat][2]], \
                                    $mat[$%1,Dim[$mat][2]-$%1+1]]

/* Minor[expr,i,j]
        forms the i,j th minor of the matrix expr. */
```

```
        _Minor[Init] :: <XList0
        Minor[$mat_,$i_,$j_] :: Natp[$mat],Natp[$i],Natp[$j]] :: \
                Map[Rm[$%1,$j],Rm[$mat,$i]]

/*
#I[1]::  <XMat2
#I[2]::  m:Ar[{3,3},f]
#O[2]:   {{f[1,1],f[1,2],f[1,3]},{f[2,1],f[2,2],f[2,3]},{f[3,1],f[3,2],f[3,3]}
#I[3]::  LDiag[m]
#O[3]:   {f[1,1],f[2,2],f[3,3]}
#I[4]::  TDiag[m]
#O[4]:   {f[1,3],f[2,2],f[3,1]}
#I[5]::  Minor[m,2,3]
#O[5]:   {{f[1,1],f[1,2]},{f[3,1],f[3,2]}}
*/
```

XMat3

```
                /**  Matrix character tests  **/

/* Matp[expr]
        yields 1 if expr represents a matrix (rank 2 tensor). */
        Matp[$expr] :: Fullp[$expr,2]

/* Sqmatp[expr]
        yields 1 if expr represents a square matrix. */
        Sqmatp[$expr] :: Matp[$expr] & P[Dim[$expr][1] = Dim[$expr][2]]

/* Symp[expr]
        yields 1 if expr represents a symmetric matrix. */
        Symp[$expr] :: Matp[$expr] & P[$expr = Trans[$expr]]

/* Asymp[expr]
        yields 1 if expr represents an antisymmetric matrix. */
        Asymp[$expr] :: Matp[$expr] & P[$expr = -Trans[$expr]]


/* Diagp[expr]
        yields 1 if expr represents a diagonal matrix. */
        Diagp[$expr] :: Matp[$expr] &          \
                                P[$expr-Diag[LDiag[$expr]]=Ar[Dim[$expr,2]]

/* Projecp[expr]
        yields 1 if expr represents a projection matrix. */
        Projecp[$expr] :: Matp[$expr] & Ex[$expr.$expr=$expr]
```

XMat4

```
                    /**  Algebraic matrix operations  **/

/* Mpow[mat,n]
        yields the n th power of the matrix mat (for integer n). */
        Mpow[$mat_?Sqmatp[$mat],0] :: Ar[Ar[2,Len[$mat]]]
        Mpow[$mat_?Sqmatp[$mat],$n_?Natp[$n]] :: Dot[Repl[$mat,$n]]
        Mpow[$mat_?Sqmatp[$mat],$n_?Natp[-$n]] :: Dot[Repl[Minv[$mat],-$n]]

/* Adj[expr]
        forms the Hermitean adjoint of expr. */
        Adj[$expr] :: Conj[Trans[$expr]]

/* Cof[expr,i,j]
        forms the i,j th cofactor of the matrix expr. */
        _Cof[Init] :: <XMat2
        Cof[$expr_?Matp[$mat],$i_?Natp[$i],$j_?Natp[$j]] :: \
                Det[Minor[$expr,$i,$j]]

/* Charpol[expr,var]
        forms the characteristic polynomial for the matrix expr
        with respect to var. */
        Charpol[$expr_?Matp[$expr],$var] :: Det[$expr - $var Ar[Dim[$expr]]]

/* Gentr[expr,k]
        forms the k th order trace of the matrix expr. */
        Gentr[$expr_?Matp[$expr],$k_?Natp[$k]] :: (Lcl[%lam]; \
                        Coef[%lam^$k,Ex[Charpol[$expr,%lam]]])

/*

#I[1]::  <XMat4

#I[2]::  m:{{a1,a2},{b1,b2}}

#O[2]:   {{a1,a2},{b1,b2}}

#I[3]::  Cof[m,1,1]

#O[3]:   b2

#I[4]::  Charpol[m,x]

#O[4]:   -a2 b1 + (a1 - x) (b2 - x)

#I[5]::  Ex[%]

                                        2
#O[5]:   a1 b2 - a1 x - a2 b1 - b2 x + x

#I[6]::  Gentr[m,2]

#O[6]:   1

#I[7]::  Gentr[m,1]

#O[7]:   -a1 - b2

*/
```

XMaxind

```
                    /** Find maximal index **/

/* Maxind[list]
        yields the maximal index in list. */
        Maxind[$list_?Contp[$list]] :: Len[$list]
        Maxind[$list] :: Ap[Max,Ar[Len[$list],Ind[$list,$1]]]

/*
#I[1]::  <XMaxind

#I[2]::  t:{[4]:a,[5]:b,[2]:c,[1]:d}

#O[2]:   {[4]: a, [5]: b, [2]: c, [1]: d}

#I[3]::  Maxind[t]

#O[3]:   5

*/
```

XMorse

```
        /** Morse code (international version) translator **/

" ":=""
".-":="a";"-...":="b";"-.-.":="c";"-..":="d";".":="e";"..-.":="f";
"--.":="g";"....":="h";"..":="i";".---":="j";"-.-":="k";".-..":="l";
"--":="m";"-.":="n";"---":="o";".--.":="p";"--.-":="q";".-.":="r";
"...":="s";"-":="t";"..-":="u";"...-":="v";".--":="w";"-..-":="x";
"-.--":="y";"--..":="z";

/*
#I[1]::  <XMorse

#I[2]::  ... -- .---.

#O[2]:   smp

*/
```

XNAT

```
                    /** Natural units **/

/* Basic constants :
          Qc        velocity of light in vacuo
          Qhbar     Planck's constant
          Qk        Boltzmann's constant
          Qalpha    Dimensionless fine structure constant
          QG        Newton's constant of gravitation
          QN        Avogadro's number. */

<XDim

/* 1. Conversion from system with Qhbar : Qc : Qk : 1 */

SNAT[1,1] :       length -> Qhbar/(Qc mass)

SNAT[1,2] :       time -> Qhbar/(Qc^2 mass)

SNAT[1,3] :       temperature -> (Qk mass Qc^2)^-1

SNAT[1,4] :       current -> mass (Qhbar^-1 Qc^5 alpha)^(1/2)

SNAT[1,5] :       amount -> QN

/*

#I[1]::  <XNAT

#I[2]::  current energy/length

          current energy
#O[2]:    --------------
              length

#I[3]::  Si[%,SDim]

          current length mass
#O[3]:    -------------------
                     2
                time

#I[4]::  Si[%,SNAT[1]]

            11/2       1/2     3
          Qc     alpha     mass
#O[4]:    ---------------------
                    3/2
               Qhbar

*/
```

XNMap

```
          /**  Multi-element generalization of Map  **/

/* NMap[f,list,n]
          applies f to groups of n elements in list. */
          _NMap[Nosmp]:=1,0,0
          _NMap[Init] :: <<UnFlat
          NMap[$f,$list_:Contp[$list],$n_:Natp[$n]] :: \
                    Rel[Map[Ap[$f,$%1],UnFlat[$list,$n]]]

/*
#I[1]::   <<NMap

#I[2]::   t:Ar[10]

#O[2]:    {1,2,3,4,5,6,7,8,9,10}

#I[3]::   NMap[f,t,2]

#O[3]:    {f[1,2],f[3,4],f[5,6],f[7,8],f[9,10]}

#I[4]::   NMap[f,t,3]

#O[4]:    {f[1,2,3],f[4,5,6],f[7,8,9],f[10]}

*/
```

XNSol

```
      /**  Numerical solution of equations by Newton's method **/

/* NSol[eqn,x,x0,acc]
          attempts to find a solution for x in the equation eqn
          using Newton's method starting at the point x=x0,
          with accuracy acc. */
          NSol[$a=$b,$x_:Symbp[$x],$x0_:Numbp[N[$x0]],$acc_:Numbp[$acc]] ::  \
                    (Lc[%f,%df,%x,%x0];%x:$x0;%f:$a-$b;%df:D[%f,$x]; \
                    Loop[,%x:N[(%x0:%x)-S[%f/%df,$x->%x]],Abs[%x-%x0]>$acc];%x)

/*
#I[1]::   <<NSol

#I[2]::   NSol[Sin[x]=x,x,1,1/1000]

#O[2]:    .001476886

#I[3]::   N[Sin[%]]

#O[3]:    .001476886

*/
```

XNorm

```
                    /**  Norm of a vector  **/

/* Norm[list]
        yields the norm of the vector represented by list. */
        Norm[$list] :: Sqrt[Ap[Plus,$list^2]]
```

XOp1

```
                    /**  Orthogonal polynomials - 1  **/


        /* 1.  Special Cases  */

        /* 1.1 Special Values */

    JacP[$n,$a,$b,1] : Comb[$n+$a,$n]
    JacP[0,$a,$b,$x] : 1
                                                    /* AS 22.4.1 */

    Geg[$n,$a_:($a!=0),1] : Comb[$n+2 $a-1,$n]
    Geg[$n_:Oddp[$n],$a_:($a!=0),0] : 0
    Geg[$n_:Evenp[$n],$a_:($a!=0),0] : (-1)^($n/2) Comb[$a+$n/2-1,$n/2]
    Geg[0,$a_:($a!=0),$x] : 1
                                                    /* AS 22.4.2 */
    Geg[$n_:($n!=0),0,1] : 2/$n
    Geg[$n_:Oddp[$n],0,0] : 0
    Geg[$n_:(Evenp[$n] & $n!=0),0,0] : 2(-1)^($n/2)/$n
    Geg[0,0,$x] : 1
                                                    /* AS 22.4.3 */

    CheT[$n,1] : 1
    CheT[$n_:Oddp[$n],0] : 0
    CheT[$n_:Evenp[$n],0] : (-1)^($n/2)
    CheT[0,$x] : 1
                                                    /* AS 22.4.4 */

    CheU[$n,1] : $n+1
    CheU[$n_:Oddp[$n],0] : 0
    CheU[$n_:Evenp[$n],0] : (-1)^($n/2)
    CheU[0,$n] : 1
                                                    /* AS 22.4.5 */
```

```
LegP[$n,1] : 1

LegP[$n_?Oddp[$n],0] : 0

LegP[$n_?Evenp[$n],0] : (-1)^($n/2) Comb[$n,$n/2]/2^$n

LegP[0,$x] : 1
                                                    /* AS 22.4.6 */


Lag[$n,$a,0] : Comb[$n+$a,$n]

Lag[0,$a,$x] : 1
                                                    /* AS 22.4.7 */

Her[$n_?Oddp[$n],0] : 0

Her[$n_?Evenp[$n],0] : (-1)^($n/2) $n!/($n/2)!
$n!/($n/2)!

Her[0,$x] : 1
                                                    /* AS 22.4.8 */
```

XOp2

```
        /** Orthogonal polynomials - 2 **/

        /* 1.2 Special Cases */


JacP[$n,$b,$b,$x] : Poch[$b+1,$n] / (Poch[2$b+1,$n] Geg[$n,$b+1/2,$x])

JacP[$n,$a,1/2,$x] : Poch[1/2,$n]/(Sqrt[(1+$x)/2] Poch[$a+1/2,$n]\
      Geg[2$n,$a+1/2,Sqrt[(1+$x)/2]])

JacP[$n,$a,-1/2,$x] : \
     Poch[1/2,$n]/Poch[$a+1/2,$n] Geg[2$n,$a+1/2,Sqrt[(1+$x)/2]]

JacP[$n,-1/2,-1/2,$x] : Comb[2$n,$n]/4^$n CheT[$x]

JacP[$n,0,0,$x] : LegP[$n,$x]

Geg[$n,1,$x] : CheU[$n,$x]

Geg[$n,0,$x] : 2/$n CheT[$n,$x]

Geg[$n,1/2,$x] : LegP[$n,$x]

Lag[$n,1/2,$x] : Her[2$n+1,Sqrt[$x]] / (2 $n! (-4)^$n  Sqrt[$x])

Lag[$n,-1/2,$x] : Her[2$n,Sqrt[$x]] / ($n! (-4)^$n)
```

XOp3

```
        /** Orthogonal polynomials - 3 **/

        /* Functional Relations */

        /* With more general functions */

SOr_:Ldist

SOr[1]:  JacP[$n,$a,$b,$x] -> \
           Comb[$n+$a,$n] Hg[-$n,$n+$a+$b+1,$a+1,(1-$x)/2]

SOr[2]:  JacP[$n,$a,$b,$x] -> Comb[2$n+$a+$b,$n] ((($x-1)/2)^$n\
           Hg[-$n,-$n-$a,-2$n-$a-$b,2/(1-$x)]

SOr[3]:  JacP[$n,$a,$b,$x] -> Comb[$n+$a,$n] ((1+$x)/2)^$n\
           Hg[-$n,-$n-$b,$a+1,($x-1)/($x+1)]

SOr[4]:  JacP[$n,$a,$b,$x] -> Comb[$n+$b,$n] ((($x-1)/2)^$n\
           Hg[-$n,-$n-$a,$b+1,($x+1)/($x-1)]
                                                    /* AS 22.5.42-45 */

SOr[5]:  Geg[$n,$a,$x] -> Comb[$n+2$a-1,$n] Hg[-$n,$n+2$a,$a+1/2,(1-$x)/2]
                                                    /* AS 22.4.46 */

SOr[6]:  Geg[$n,$a,$x] -> Gamma[$a+1/2] Gamma[2$a+$n]/($n!*Gamma[2$a])\
           (($x^2-1)/4)^(1/4-$a/2) LegP[$n+$a-1/2,1/2-$a,$x]
                                                    /* AS 22.5.60 */

SOr[7]:  CheT[$n,$x] -> Hg[-$n,$n,1/2,(1-$x)/2]

SOr[8]:  CheU[$n,$x] -> ($n+1) Hg[-$n,$n+2,3/2,(1-$x)/2]
                                                    /* AS 22.4.47-48 */

SOr[10]:  LegP[$n,$x] -> Hg[-$n,$n+1,1,(1-$x)/2]

SOr[11]:  LegP[$n,$x] -> \
           Comb[2$n,$n] ((($x-1)/2)^$n Hg[-$n,-$n,-2$n,2/(1-$x)]

SOr[12]: LegP[$n,$x] -> \
           Comb[2$n,$n] ($x/2)^$n Hg[-$n/2,(1-$n)/2,1/2-$n,1/$x^2]

SOr[13]: LegP[$n_:Oddp[$n],$x] -> \
           (-1)^$n (2$n-1)!/(2^$n $n!)^2 $x Hg[-$n,$n+3/2,3/2,$x^2]

SOr[14]: LegP[$n_:Evenp[$n],$x] -> \
           Comb[2$n,$n] Hg[-$n,$n+1/2,1/2,$x^2] / (-4)^$n

                                                    /* AS 22.4.49-53 */

SOr[15]: Lag[$n,$a,$x] -> Comb[$n+$a,$n] Chg[-$n,$a+1,$x]
                                                    /* AS 22.5.54 */

SOr[16]: Her[$n_:Oddp[$n],$x] -> \
           (-1)^((($n-1)/2) $n!/((($n-1)/2)!*2$x Chg[-($n-1)/2,1/2,$x^2]

SOr[17]: Her[$n_:Evenp[$n],$x] -> \
           (-1)^($n/2) $n!/($n/2)! Chg[-$n/2,1/2,$x^2]
                                                    /* AS 22.5.55-56 */

SOr[18]: Her[$n,$x] -> 2^$n Par[1/2-$n/2,3/2,$x^2]

SOr[19]: Her[$n,$x] -> 2^($n/2) Exp[$x^2/2] Par[-$n-1/2,Sqrt[2] $x]

             /* With Orthogonal Polynomials */

SOr[20]:  Geg[$n_:Oddp[$n],$a_:($a!=0),$x] -> Poch[$a,$n+1] $n! \
           2^(2$n+1) $x JacP[$n,$a-1/2,1/2,2$x^2-1] / (2$n+1)!
```

```
SOr[21]:   Geg[$n_?Evenp[$n],$a_?($a!=0),$x] -> Poch[$a,$n] $n!\
             2^(2$n) JacP[$n,$a-1/2,1/2,2$x^2-1] / (2$n)!

SOr[22]:   Geg[$n,$a!=0,$x] ->          \
             Poch[2$a,$n] JacP[$n,$a-1/2,$a-1/2,$x] / Poch[$a+1/2,$n]

SOr[23]:   Geg[$n,0,$x] -> \
             2($n-1)! JacP[$n,-1/2,-1/2,$x] / Poch[1/2,$n]

                                                        /* AS 22.5.25-28 */

SOr[24]:   CheT[$n_?Oddp[$n],$x] -> \
             $n! $x JacP[$n,-1/2,1/2,2$x^2-1] / Poch[1/2,$n]

SOr[25]:   CheT[$n,$x] -> $n! JacP[$n,-1/2,-1/2,$x] / Poch[1/2,$n]

SOr[26]:   CheU[$n_?Evenp[$n],$x] -> \
             $n! JacP[$n,1/2,-1/2,2$x^2-1] / Poch[1/2,$n]

SOr[27]:   CheT[$n,$x] -> $n/2 Geg[$n,0,$x]

SOr[28]:   CheU[$n,$x] -> Geg[$n,1,$x]

                                                        /* AS 22.5.29-34 */

SOr[29]:   LegP[$n,$x] -> JacP[$n,0,0,$x]

SOr[30]:   LegP[$n,$x] -> Geg[$n,1/2,$x]
                                                        /* AS 22.5.35-36 */

SOr[31]:   Her[$n_?Oddp[$n],$x] -> \
             (-1)^(($n-1)/2) 2^$n (($n-1)/2)! Lag[($n-1)/2,1/2,$x^2]

SOr[13]:   Her[$n_?Evenp[$n],$x] -> \
             (-1)^($n/2) 2^$n (n/2)! Lag[$n/2,-1/2,$x^2]
                                                        /* AS 22.5.40-41 */
```

XOpR

```
          /** Rodrigues formulae for orthogonal polynomials **/

Sopr[1] : JacP[$n,$a,$b,$z] -> D[(1-%x)^$a (1+%x)^$b (1-%x^2)^$n,{%x,n,$z}] \
                                 / (-2)^$n / $n! / (1-$z)^$a / (1+$z)^$b

Sopr[2] : Geg[$n,$a,$z] -> \
            Poch[2$a,$n] (1-$z)^(1/2-$a) D[(1-%x^2)^($a+$n+1/2),{%x,$n,$z
             / (-2)^$n / $n! / Poch[$a+1/2,$n]

Sopr[3] : CheT[$n,$z] -> Sqrt[Pi(1-$z^2)] D[(1-%x^2)^($n-1/2),{%x,$n,$z}]\
                           / (-2)^$n / Gamma[$n+1/2]

Sopr[4] : CheU[$n,$z] -> ($n+1) Sqrt[Pi/(1-$z^2)] D[(1-%x^2)^($n+1/2),{%x,$n,$z
                         / 2 / (-2)^$n / Gamma[$n+1/2]

Sopr[5] : LegP[$n,$z] -> D[(1-%x^2)^$n,{%x,$n,$z}] / (-2)^$n / $n!

Sopr[6] : Lag[$n,$a,$z] -> D[%x^($n+$a) Exp[-%x], {%x,$n,$z}] Exp[$z] / $n! / $

Sopr[7] : Her[$n,$z] -> (-1)^$n Exp[$z^2] D[Exp[-%x^2],{%x,$n,$z}]

                                                        /* AS 22.11 */
```

XPR

```
                    /**  Special output forms  **/

/* PSig
        prints as   ---
                     \
                     /
                    ---
                                                          */
_PSig[Pr] : Fmt[{{3,2},{2,2},{1,2},{1,1},{1,0},{1,-1},{2,-1},{3,-1}},\
        "_","_","_","_",

/* PPi
        prints as
                    ____
                    | |
                    | |
                                                          */
_PPi[Pr] : Fmt[{{1,-1},{1,0},{1,1},{2,1},{3,1},{3,0},{3,-1}},\
        "|","|","_","_","_","|","|"]

/* Plam
        prints as
                     \
                     /\
                                                          */
_Plam[Pr] : Fmt[{{1,-1},{1,0}},"/

/* PLam
        prints as   /\
                    /  \
                                                          */
_PLam[Pr] : Fmt[{{1,-1},{2,0},{3,-1}},"/","/

/* PDelta
        prints as   /\
                    /__\
                                                          */
_PDelta[Pr] : Fmt[{{1,-1},{2,-1},{3,-1},{2,0}},"/","__","

/* PGamma
        prints as   _____
                     |
                     |
                                                          */
_PGamma[Pr] : Fmt[{{1,-1},{1,0},{1,1}},"|","|","_____"]

/* PXi
        prints as   ----
                     --
                    ----
                                                          */
_PXi[Pr] : Fmt[{{1,-1},{2,-1},{3,-1},{2,0},{1,1},{2,1},{3,1}},"-","--","-",  \
                                "__","_","__","_"]

/* PInt
        prints as   /
                    |
                    /
                                                          */
_PInt[Pr] : Fmt[{{1,1},{1,0},{1,-1}},"/","|","/"]

/* PSqrt
        prints as  --
                   /
                  /
                                                          */
_PSqrt[Pr] : Fmt[{{1,-1},{2,0},{3,1}},"/","/","--"]

/*
```

```
#I[1]::  <XPR

#I[2]::  PPi^2
          __2
#O[2]:*  | |
         | |

#I[3]::  _∫[Pr]:PSig

         ---
          \
#O[3]:*  /
         ---

#I[4]::  f+1/f

         ---
               1
#O[4]:*  /   + ---
         ---   ---
                \
                /
                ---

*/
```

XPar

```
                    /** Parabolic cylinder function **/
SPar_⊥Ldist

SPar[1]:    Par[$n,$z] -> 2^(($n+1/2)/2)$z^(-1/2)WhiW[$n/2+1/4,1/4,$z^2/2]

SPar[2]:    Par[$n,$z] -> 2^(($n-1)/2)Exp[-$z^2/4]$z KumU[(1-$n)/2,3/2,$z^2/2]
                                                        /* MOS p.324 */

                              /* Special values */

SPar[3]:    Par[0,$z] -> Exp[-$z^2/2]

SPar[4]:    Par[1/2,$z] -> Pi^(-1/2)($z/2)^(3/2)\
                  (BesK[1/4,$z^2/4]+BesK[3/4,$z^2/4])

SPar[5]:    Par[3/2,$z] -> Pi^(-1/2)($z/2)^(5/2)(2BesK[1/4,$z^2/4]\
                  +3BesK[3/4,$z^2/4]-BesK[5/4,$z^2/4])

SPar[6]:    Par[5/2,$z] -> ($z/2)^(7/2)/Pi(5BesK[1/4,$z^2/4]+9BesK[3/4,$z^2/4]\
                  -5BesK[5/4,$z^2/4]-BesK[7/4,$z^2/4])

SPar[7]:    Par[-1/2,$z] -> ($z/(2Pi))^(1/2)BesK[1/4,$z^2/4]

SPar[8]:    Par[-3/2,$z] -> ($z)^(3/2)/(2Pi)^(1/2)\
                  (BesK[3/4,$z^2/4]-BesK[1/4,$z^2/4])

SPar[9]:    Par[-5/2,$z] -> $z^(5/2)/(3(2Pi)^(1/2))(BesK[5/4,$z^2/4]\
                  -3BesK[3/4,$z^2/4]+2BesK[1/4,$z^2/4])

SPar[10]:   Par[$n,$z] -> 2^(-$n/2)Exp[-$z^2/4]Her[$n,$z/2^(1/2)]
                                                        /* MOS p.325 */
                  /* Recurrence relations */

SPar[11]:    Par[$n,$z] -> $z Par[$n-1,$z]-($n-1)Par[$n-2,$z]
```

```
SPar[12]:   Par[$n,$z] -> (Par[$n+1,$z]+$n Par[$n-1,$z])/$z

SPar[13]:   Par[$n,$z] -> ($z Par[$n+1,$z]-Par[$n+2,$z])/($n+1)
                                                    /* MOS p.327 */
```

XPauli

```
                    /**  Representation of Pauli sigma matrices  **/

/* Sigma[i]
        gives a representation of the the ith Pauli sigma matrix. */
        Sigma[0] : {{1,0},{0,1}}
        Sigma[1] : {{0,1},{1,0}}
        Sigma[2] : {{0,-I},{I,0}}
        Sigma[3] : {{1,0},{0,-1}}
```

XPeel

```
                    /**  Peel away sublists  **/

/* Peel[{expr1,expr2,...}]
        represents the sequence expr1,expr2,... in a list.
        ("Peels" away lists). */
        Peel[$list_?Listp[$list]] :: Ap[Np,$list]

/*

#I[1]::  <<XPeel

#I[2]::  Peel[{a,b},{c,d}]

#O[2]:   {a,b,{c,d}}
*/
```

XPerm0

```
                    /** Permutations **/

/* A permutation is represented by a contiguous list of sequential integers
in any order */

/* Permp[expr]
        yields 1 if expr represents a permutation. */
        Permp[$expr] :: P[Sort[$expr]=Ar[Len[$expr]]]
        Permp[$expr_:~Contp[$expr]] : 0

/* Fiper[list1,list2]
        finds if possible a permutation which places the elements of
        list2 in the order of list1. */
        Fiper[$list1_:Contp[$list1],          \
                        $list2_:(Contp[$list2]&Len[$list1]=Len[$list2])] :\
                Flat[Ar[Len[$list1],Pos[$list2[$%1],$list1]]]

/* Apper[perm,list]
        applies the permutation perm to list. */
        Apper[$perm_:Permp[$perm],$list_:Contp[$list]] :: \
                Ar[Len[$list],$list[$perm[$%1]]]

/*

#I[1]::  <XPerm0

#I[2]::  Permp[{1,3,2,4}]

#O[2]:   1

#I[3]::  Permp[{1,3,1,4}]

#O[3]:   0

#I[4]::  t1:{a,c,d,b}

#O[4]:   {a,c,d,b}

#I[5]::  Fiper[{a,b,c,d},t1]

#O[5]:   {1,3,4,2}

#I[6]::  Apper[%,{a,b,c,d}]

#O[6]:   {a,c,d,b}

*/
```

XPerm1

```
          /**  Elementary operations on permuations  **/

<XPerm0

/* Pcomp[perm1,perm2]
          forms the compostion (product) of the two permutations
          perm1 and perm2. */
          Pcomp[$p1_:Permp[$p1],$p2_:Permp[$p2]] :: Ar[Len[$p1],$p2[$p1[$%1]]]

/* Ppow[perm,n]
          forms the nth power of the permutation perm. */
          Ppow[$p_:Permp[$p],0] :: Ar[Len[$p]]
          Ppow[$p_:Permp[$p],$n_:Natp[$n]] :: \
                              $[$p,$%1-->Ar[Len[$p],$%1[$p[$%2]]],$n-1]
          Ppow[$p_:Permp[$p],$n_:Natp[-$n]] :: Ppow[Pinv[$p],-$n]

/* Pinv[perm]
          yields the inverse of the permutation perm. */
          Pinv[$p_:Permp[$p]] :: Ar[Len[$p],Pos[$%1,$p][1,1]]


/*

#I[1]::   <XPerm1

#I[2]::   p1:{1,5,4,2,3}

#O[2]:    {1,5,4,2,3}

#I[3]::   p2:{5,1,3,2,4}

#O[3]:    {5,1,3,2,4}

#I[4]::   Pcomp[p1,p2]

#O[4]:    {5,4,2,1,3}

#I[5]::   Pcomp[p2,p1]

#O[5]:    {3,1,4,5,2}

#I[6]::   Ppow[p2,6]

#O[6]:    {4,5,3,1,2}

#I[7]::   Pinv[p1]

#O[7]:    {1,4,5,3,2}

#I[8]::   Pcomp[%,p1]

#O[8]:    {1,2,3,4,5}

*/
```

XPermC

```
        /**  Cycle decomposition of permutations  **/

/* Cycles are represented by projections of the form C[i1,i2,i3,...] */

<XPerm0

/* ToC[perm]
        yields a list of cycles whose composition is perm
        (cycle decomposition). */
        ToC[$p_;Permp[$p]] :: (Lcl[%a,%t,%n,%l]; \
                %a:{};%t:Ar[Len[$p],1];Do[%i,Len[$p],If[%t[%i], \
                                For[%n:$p[%i];%i:{}, \
                %t[%n], %n:$p[%n], %t[%n]:0;%i:Cat[%i,{%n}]]; \
                        %a:Cat[%a,{Ap[C,%i]}]]];%a)


/* ToP[cycs]
        yields the permutation represented by the list of cycles cycs. */
        ToP[$cycs_;Listp[$cycs]] :: (Lcl[%i]; Map[Ar[Len[$c], \
                %i[Cyc[$c,-1][$i]]:$c[$i]],$cycs,1]; %i)


/*

#I[1]::   <XPermC

#I[2]::   ToC[{1,5,7,2,4,3,6}]

#O[2]:    {C[1],C[5,4,2],C[7,6,3]}

#I[3]::   ToP[%]

#O[3]:    {1,5,7,2,4,3,6}

*/
```

XPhys

```
                /**  Fundamental physical constants  **/


/* 1. Principal constants */

/* speed of light in vacuo */
SPhys[1,1] :        Qc -> 2.997924580*^8 metre second^-1

/* Planck's constant */
SPhys[1,2] :        Qh -> 6.626179*^-34 joule second

/* Dirac's constant */
SPhys[1,3] :        Qhbar -> 1.0545887*^-34 joule second

/* charge on electron */
SPhys[1,4] :        Qe -> 1.6021892*^-19 coulomb

/* mass of electron */
SPhys[1,5] :        Qme -> 9.109534*^-31 kilogram

/* Avogadro's number */
SPhys[1,6] :        QN -> 6.022045*^23 mole^-1

/* constant of gravitation */
```

```
SPhys[1,8] :      QG -> 6.6720e~11 newton metre^2 kilogram~-2


/* 2. Atomic constants */

/* fine structure constant */
SPhys[2,1] :      Qalpha -> 7.2973506e~3

/* Rydberg constant (infinite nuclear mass) */
SPhys[2,2] :      QRinf -> 1.097373177e^7 metre~-1

/* Bohr radius */
SPhys[2,3] :      Qa0 -> 5.2917706e~11 metre

/* electron Compton wavelength */
SPhys[2,4] :      QlambdaC -> 2.4263089e~12 metre

/* classical "electron radius" */
SPhys[2,5] :      Qre -> 2.817938e~15 metre

/* Thomson cross-section */
SPhys[2,6] :      QsigmaT -> 6.6522448e~29 metre^2

/* Bohr magneton */
SPhys[2,7] :      QmuB -> 9.274078e~24 joule tesla~-1

/* nuclear magneton */
SPhys[2,8] :      QmuN -> 5.5050824e~27 joule tesla~-1

/* gyromagnetic ratio of free proton */
SPhys[2,9] :      Qgamma -> 2.6751987e^8 second~-1 tesla~-1

/* electron volt */
SPhys[2,10] :     QeV -> 1.6021892e~19 joule


/* 3. Thermal constants */

/* molar gas constant */
SPhys[3,1] :      QR -> 8.31441 joule kelvin~-1 mole~-1

/* Loschmidt number */
SPhys[3,2] :      QL -> 2.686754e^25 metre~3

/* Boltzmann constant */
SPhys[3,3] :      Qk -> 1.380662e~23 joule kelvin~-1

/* Stefan-Boltzmann constant */
SPhys[3,4] :      Qsigma -> 5.67032e~8 watt metre~-2 kelvin~-4
```

XPlhist

```
                    /**  Plot histogram  **/

/* Plhist[list]
        plot list as a histogram. */
        Plhist[$list] :: Plot[{Ar[Len[$list],  \
        Line[{Pt[{Ind[$list,{$1}],0}],Pt[{Ind[$list,{$1}],Elem[$list,{$1}]}]\
      Pt[{Ind[$list,{$1+1}],Elem[$list,{$i}]}],Pt[{Ind[$list,{$1+1}],0}]}], \
                        Axes[0,0]}]
```

XPlot

```
                    /**  Operations on plots  **/

/* PCat[p1,p2,...]
        combines the plots p1, p2, ... into a single plot. */
        PCat[Plot[$p1,$$p1],Plot[$p2,$$p2]] :: Plot[Union[$p1,$p2]]

/* PAp[trx,try,pl]
        applies the templates trx and try respectively to each point
        in the plot pl. */
        PAp[$trx,$try,$pl] :: S[$pl,Pt[{$x,$y},$$f]--> \
                                Pt[{Ap[$trx,{$x}],Ap[$try,{$y}]},$$f]]
```

XPolar

```
                    /**  Polar graphs  **/

/* Polar[expr,theta,npt]
        yields a polar plot of expr obtained by evaluation at npt
        points in the angle theta. */
        Polar[$expr,$theta,$npt] :: \
        Graph[{$expr Cos[$theta],$expr Sin[$theta]},$theta,0,2Pi,,,$npt]
```

XPoly

```
                    /** Information on polynomials **/

/* LCoef[poly,var]
        yields a list of the coefficients of powers of var in poly. */
        LCoef[$poly,$var] :: Ar[Expt[$var,$poly],Coef[$var^$%1,$poly]]

/* LExpt[poly,var]
        yields a list of the exponents with which var appears in poly. */
        LExpt[$poly,$var] :: Union[Expt[$var,$poly,List]]

/*
#I[1]::  <XPoly

#I[2]::  t:Ex[(a+x)^3 (1-x)^2]

                 2       3       4       2       2  2       2  3       3       3  2
#O[2]:   3a x  - 6a x  + 3a x  + 3 a  x - 6 a  x  + 3 a  x  - 2 a  x + a  x
                   3    3       4    5
               + a  + x  - 2 x  + x

#I[3]::  LCoef[%,x]

                 2       3       2    3       2
#O[3]:   {3 a  - 2 a ,3a - 6 a  + a ,1 - 6a + 3 a ,-2 + 3a,1}

#I[4]::  LExpt[t,x]

#O[4]:   {1,2,3,4,5}
*/
```

XPrimep

```
/* Primep[expr]
        yields 1 if expr is a prime number, and 0 otherwise. */
        Primep[$expr] :: Natp[$expr] & P[Len[Nfac[$expr]]=1]
```

XProj

```
                    /** Projection manipulation **/

/* PCat[proj1,proj2]
        yields a projection whose filters are the concatenation of
        those in the projections proj1, proj2 (which must have
        the same projector). */
        PCat[$x_:Projp[$x],$y_:(Projp[$y] & $y[0]=$x[0])] :: \
                Proj[$x[0],{S[$x,$x[0]->Np],S[$y,$y[0]->Np]}]

/* Pap[temp,proj1,proj2,...]
        treats the filters of the projections proj1, proj2, ... as lists,
        and applies the template temp to them, yielding a projection with
        the resulting list as its filters. */
        _Pap[Nosmp]:{1}
```

```
Pap[$temp,$$proj_=(Ap[And,Map[Projp,{$$proj}]] & \
        Ap[Eq,Map[$%[0],{$$proj}]])] :: \
        Proj[$$proj[1][0],Ap[temp,Map[S[$%1[0]->Np,{$$proj}]]]]
```

XPsi

```
        /** Digamma and Polygamma (Psi) functions **/

SPsi _= Ldist

SPsi[1]:   Psi[$z] -> Psi[$z+1] -1/$z
                                                    /* MOS p. 14 */

SPsi[2]:   Psi[$z] -> Psi[1-$z]-Pi Cot[Pi $z]
                                                    /* MOS p. 14 */

SPsi[3]:   Psi[$z] -> Psi[-$z]-1/$z-Pi Cot[Pi $z]
                                                    /* MOS p. 14 */

SPsi[4]:   Psi[$z] -> Psi[-$z+2]+1/($z-1)-Pi Cot[Pi($z-1)]
                                                    /* MOS p. 14 */

SPsi[5]:   Psi[$z] -> Psi[-$z+1]+Pi Tan[Pi($z-1/2)]
                                                    /* MOS p. 14 */

SPsi[6] :  Psi[$z] -> Psi[$z-1] + 1/($z-1)

SPsi[7]:   Psi[$n_=Natp[$n],1] -> (-1)^($n+1)$n!*Zata[$n+1]
                                                    /* AS 6.4.2 */

SPsi[8]:   Psi[$m_=Natp[$m],$n] -> (-1)^$m $m!*(-Zeta[$m+1]+\
           Sum[k^(-$m-1),k,1,$n-1]
                                                    /* AS 6.4.3 */

SPsi[9]:   Psi[$n_=Natp[$n],1/2] -> (-1)^($n+1) $n!*(2^($n+1)-1)Zeta[$n+1]
                                                    /* AS 6.4.4 */

SPsi[10]:  Psi[$n,$z] -> Psi[$n,$z]+(-1)^$n $n!*$z^(-$n-1)
                                                    /* AS 6.4.6 */

SPsi[11]:  Psi[$n,$z] --> (-1)^($n+1)Psi[$n,1-$z]-Pi D[Cot[Pi $q],{$q,$n,$z}]
                                                    /* AS 6.4.7 */

SPsi[12]:  Psi[$n,($m_=Natp[$m]) $z] -> ($n=0)Log[$m]+$m^(-$n-1)\
           Sum[Psi[$n,$z+k/$m],k,0,$m-1]
                                                    /* AS 6.4.8 */

<<Harm

Psi[1]: -Euler

Psi[1/2]: -Euler-2Log[2]

Psi[-1/2]: -Euler-2Log[2]+2

Psi[$n_=Natp[$n-1]] : Harm[$n-1]-Euler
```

XRandC

```
   /**  Generation of random numbers from continuous distributions  **/

/* NRand[(x:0),(sd:1)]
        generates a random number from a normal (Gaussian) distribution
        with mean x and standard deviation sd. */
        NRand_:Tier
        NRand[]::NRand[0,1]
        NRand[$mean]::NRand[$mean,1]
        NRand[$mean,$sd] :: \
                N[$mean + $sd Sqrt[-2 Log[Rand[]]] Cos[(!N[2Pi]) Rand[]]]

/* BRand[rho]
        generates a pair of random numbers from a bivariate normal distributio
        with zero mean, unit variance and correlation coefficient rho. */
        BRand[$rho] :: (Lc![%x1]; \
                        {%x1:NRand[], N[$rho %x1 + Sqrt[1-$rho^2] NRand[]]})

/* ERand[theta]
        generates a random number from the exponential distribution
        Exp[-x/theta]. */
        ERand[$theta] :: N[-$theta Log[Rand[]]]

/* ARRand[a,b,dist,maxdist]
        uses an acceptance-rejection method to generate a random number
        between a and b from the distribution defined by the template
        dist whose maximum value is not less than maxdist.
        Number of attempts necessary is proportional to
        (b-a) maxdist. */
        ARRand[$a,$b,$dist,$maxdist] :: (Lc![%y];  \
                        Loop[,%y:N[$a + ($b-$a) Rand[]], \
                        Rand[] >= N[Ap[$dist,{%y}]/$maxdist]]; %y)
```

XRandD

```
   /**  Generation of random numbers from discrete distributions  **/

/* IRand[n]
        generates a random integer from a uniform distribution between
        0 and n-1. */
        IRand[$n] :: Gint[Rand[$n]]

/* PNorm[list]
        yields a normalized list of probabilities from a list of relative
        frequencies. */
        PNorm[$list] :: N[$list/Ap[Plus,$list]]

/* PCum[list]
        yields a list of cumulative probabilities. */
        PCum[$list] :: (Lc![%t]; %t:0; Map[%t:%t+$%1;%t,PNorm[$list]])

/* DRand[{cp1,cp2,...}]
        yields a random position in the list with distribution determined
        by the cumulative probabilities cpi. */
        /* Not optimal algorithm */
        DRand[$list] :: (Lc![%x]; %x:Rand[]; Pos[$1_=$1>%x,$list,2,1][1,2,1])
```

XRandL

```
                    /**  Random selection of list elements  **/

/* LRand[list]
        yields one of the elements of list, randomly chosen with equal
        probabilities. */
        ⊥LRand[Init] :: <XRandD
        LRand[$list_:Listp[$list]] :: Ent[$list,{1+IRand[Len[$list]]}]

/* LDRand[list,prob]
        yields one of the elements of list, randomly chosen with
        relative frequencies given by prob. */
        ⊥LDRand[Init] :: <XRandD
        LDRand[$list,$prob_:(Contp[$prob] & Len[$prob]=Len[$list])] :: \
                $list[DRand[PCum[$prob]]]

/* ORand[list]
        yields a random reordering of list. */
        _DRand[Init] :: <XPerm
        ORand[$list] :: (Lcl[%I]; %I:Ar[Len[$list],`Rand[]]; \
                Apper[Fiper[%I,Sort[%I]],$list])
```

XRot2

```
                    /**  Rotations in two dimensions  **/

/* Vec2p[expr]
        tests whether expr represents a two-dimensional vector. */
        Vec2p[$expr] :: Contp[$expr] & Len[$expr]=2

/* RotM2[theta]
        yields a matrix representing a two-dimensional rotation through
        an angle of theta radians. */
        RotM2[$theta] : {{Cos[$theta],Sin[$theta]},{-Sin[$theta],Cos[$theta]}}

/* Rot2[vec,theta,(pt:{0,0})]
        rotates the two-dimensional vector vec about the point pt
        through an angle of theta radians. */
        Rot2_:Tier
        Rot2[$vec_:Vec2p[$vec],$theta] : RotM2[$theta].$vec
        Rot2[$vec_:Vec2p[$vec],$theta,$pt_:Vec2p[$pt]] :: \
                                $pt + RotM2[$theta].($vec-$pt)
```

XRot3

```
                /**  Rotations in three dimensions  **/

/* Vec3p[expr]
        tests whether expr represents a three-dimensional vector. */
        Vec3p[$expr] :: Contp[$expr] & Len[$expr]=3

/* RotM3[theta]
        yields a matrix representing a three-dimensional rotation specified
        by the Euler angles phi, theta, psi. */
        RotM3[$phi,$theta,$psi] : \
                {{Cos[$psi] Cos[$phi] - Cos[$theta] Sin[$phi] Sin[$psi], \
                 Cos[$psi] Sin[$phi] + Cos[$theta] Cos[$phi] Sin[$psi], \
                 Sin[$psi] Sin[$theta]}, \
                 {-Sin[$psi] Cos[$phi] - Cos[$theta] Sin[$phi] Cos[$psi], \
                 -Sin[$psi] Sin[$phi] + Cos[$theta] Cos[$phi] Cos[$psi], \
                 Cos[$psi] Sin[$theta]}, \
                 {Sin[$theta] Sin[$phi], -Sin[$theta] Cos[$phi], Cos[$theta]}


/* Rot3[vec,phi,theta,psi,(pt:{0,0,0})]
        rotates the three-dimensional vector vec about the point pt
        through the Euler angles phi, theta, psi. */
        Rot3_:Tier
        Rot3[$vec_?Vec3p[$vec],$theta] : RotM3[$theta].$vec
        Rot3[$vec_?Vec3p[$vec],$theta,$pt_?Vec3p[$pt]] :: \
                                $pt + RotM3[$theta].($vec-$pt)
```

XRpoly

```
                /**  Random polynomial generation  **/

/* Ranup[n]
        generates a random univariate polynomial of size n. */
        Ranup[$n_?Natp[$n]] :: \
                Ex[Rex[$n,{{x,1},{2,1},{-1,1}},{{Plus,1,-7},{Mult,1,-5}}]]

/* Ranmp[n,nx]
        generates a random polynomial of size n in an average of nx
        variables. */
        Ranmp[$n_?Natp[$n],$nx_?Natp[$nx]] :: \
                Ex[Rex[$n,Cat[Ar[{{10,10+$nx-1}},Impl[{$%1}],{{2,1},{-1,1}}],\
                        {{Plus,1,-7},{Mult,1,-5}}]]

/*

#I[1]::  <XRpoly

#I[2]::  Ranup[10]

                  3         4         5
#O[2]:   168 x  + 240 x  + 78 x

#I[3]::  Ranup[10]

                       2
#O[3]:   2 + 5x + 8 x

#I[4]::  Ranmp[10,3]

                                  2
#O[4]:   12 + 3a + 2b + 4c + 8a c
```

```
#I[5]::  Ranmp[18,3]

                                        2  5       2  6       3  5       2  5
#O[5]:   12b + 18a b + 4b c + 96 a  c  + 32 a  c  + 80 a  c  + 16 a  b c

                  2
            + 2 b

*/
```

XRslt

```
                    /**  Polynomial resultants  **/

/* Rslt[a,b,x]
        forms the resultant of the polynomials a and b with respect
        to the unit x. */
        Rslt[$a,$b,$x] :: (Lc![%eq,%a,%b]; %eq:Sol[{$a=%a,$b=%b},,{$x}]; \
                  S[Num[Rat[Ex[%eq[1,1]-%eq[1,2]]]],{%a->0,%b->0}])


/* Disc[a,x]
        forms the discriminant of the polynomial a in x,
        which must be zero if a has multiple roots.  */
        Disc[$a,$x_?Symbp[$x]] :: Rslt[Expt[$x,$a]$a-$x D[$a,$x],D[$a,$x],$x]

/*

#I[1]::  <XRslt

#I[2]::  Rslt[x^2+a x y,y^2-a x^3 y,x]

          4  4     2
#O[2]:   a  y  + y

#I[3]::  t:(1+a x)(1-b x^2)

                          2
#O[3]:   (1 + a x) (1 - b x )

#I[4]::  Ex[%]

                       2        3
#O[4]:   1 + a x - b x  - a b x

#I[5]::  Disc[%,x]

             2  2        4          3
#O[5]:   24 a  b  - 12 a  b - 12 b

#I[6]::  Disc[Ex[(1-x)^2*@4]]

#O[6]:   0

*/
```

XSets

```
        /** Elementary finite set theory **/

/* Sets are represented as ordered lists with no repeated elements.
A list may be placed in this canonical form by Union[list].
Once in this form, the equivalence of two sets is determined by Eq. */

/* Union[set1,set2,...]
        forms the union of the sets set1, set2, ... */

/* Inter[set1,set2,...]
        forms the intersection of the sets set1, set2, ... */

/* Cmpl[set,uset]
        yields the complement of set with respect to the universal
        set uset. */
        Cmpl[$list_;Listp[$list],$ulist_;Listp[$ulist]] :: \
                        Cat[Ar[Len[$ulist],$ulist,,~In[$%1,$list]]]


/* Sub[set,crit]
        yields a subset of set all of whose elements satisfy the condition
        crit. */
        Sub[$set,$crit] :: Flat[Ar[Len[$set],,,$crit]]

/* Subp[sub,set]
        yields 1 if sub is a subset of set and 0 if it is not. */
        Subp[$sub,$set] :: Len[Union[$sub,$set]]=Len[Union[$set]]

/* Disjp[$set1,$set2]
        yields 1 if the sets set1 and set2 are disjoint (have no elements
        in common). */
        Disjp[$set1,$set2] :: P[Inter[$set1,$set2]={}]

/* Multset[set1,set2,...]
        forms the product set of the set1 (set of all possible ordered
        n-tuples of elements from n sets). */
        Multset[$$set] :: Flat[Outer[List,$$set],Len[List[$$set]]-1]

/* Powset[set]
        forms the set of all possible subsets of set (power set). */
        Powset[$1_;Contp[$1]] :: Flat[Lc1[%f,%g]; S[Dist[Ap[%g,Ar[Len[$1], \
                %f[{},{$1[$%1]}]]],{%g,%f,List,%g}],%g->Cat],Len[$1]-1]

/*

#I[1]::  <XSets

#I[2]::  s1:Union[{a,b,a,c,d}]

#O[2]:   {a,b,c,d}

#I[3]::  s2:{c,e,f}

#O[3]:   {c,e,f}

#I[4]::  s3:Ar[5]

#O[4]:   {1,2,3,4,5}

#I[5]::  Union[s1,s2,s3]

#O[5]:   {1,2,3,4,5,a,b,c,d,e,f}

#I[6]::  Inter[s1,s2]

#O[6]:   {c}

#I[7]::  Cmpl[s2,s1]

#O[7]:   {a,b,d}
```

```
#I[8]::  Sub[s3,Evenp]

#O[8]:   {2,4}

#I[9]::  Subp[{a,c},s1]

#O[9]:   1

#I[10]::  Disjp[s1,s3]

#O[10]:   1

#I[11]::  Multset[s1,s2]

#O[11]:   {{{a,c},{a,e},{a,f}},{{b,c},{b,e},{b,f}},{{c,c},{c,e},{c,f}},
                {{d,c},{d,e},{d,f}}}

#I[12]::  Powset[s2]

#O[12]:   {{},{f},{e},{e,f},{c},{c,f},{c,e},{c,e,f}}

*/
```

XSetsSX

```
                    /**  Set theory notation  **/

/* aUbUc...  or  a U b U c ...
        stands for Union[a,b,c,...]. */
        Sxset["U",Union,3]

/* aIbIc...  or  a I b I c ...
        stands for Inter[a,b,c,...]. */
        Sxset["I",Inter,3]

/* aor  a  b
        stands for Cmpl[b,a]. */
        Sxset["

/* aCb or a C b
        stands for Subp[a,b]. */
        Sxset["C",Subp,4]

/* a~*b~*c... or a ~* b ~* c ...
        stands for Multset[a,b,c...]. */
        Sxset["~*",Multset,3]

/* ~~a  or  ~~ a
        stands for Powset[a]. */
        SxsetI"~~",Powset,1]
```

XSol

```
          /**  Inverses of elementary transcendental functions  **/

Sol[Exp[$x]=$y,$x]  :: Sol[$x=Log[$y],$x]
Sol[Log[$x]=$y,$x]  :: Sol[$x=Exp[$y],$x]
Sol[Sin[$x]=$y,$x]  :: Sol[$x=Asin[$y],$x]
Sol[Cos[$x]=$y,$x]  :: Sol[$x=Acos[$y],$x]
Sol[Tan[$x]=$y,$x]  :: Sol[$x=Atan[$y],$x]
Sol[Asin[$x]=$y,$x]  :: Sol[$x=Sin[$y],$x]
Sol[Acos[$x]=$y,$x]  :: Sol[$x=Cos[$y],$x]
Sol[Atan[$x]=$y,$x]  :: Sol[$x=Tan[$y],$x]
Sol[Sinh[$x]=$y,$x]  :: Sol[$x=Asinh[$y],$x]
Sol[Cosh[$x]=$y,$x]  :: Sol[$x=Acosh[$y],$x]
Sol[Tanh[$x]=$y,$x]  :: Sol[$x=Atanh[$y],$x]
Sol[Asinh[$x]=$y,$x]  :: Sol[$x=Sinh[$y],$x]
Sol[Acosh[$x]=$y,$x]  :: Sol[$x=Cosh[$y],$x]
Sol[Atanh[$x]=$y,$x]  :: Sol[$x=Tanh[$y],$x]
Sol[Gd[$x]=$y,$x]  :: Sol[$x=Agd[$y],$x]
Sol[Agd[$x]=$y,$x]  :: Sol[$x=Gd[$y],$x]
```

XSpare

```
                    /**  Remove almost all values  **/

/* Spare[v1,v2,...]
          removes all values except those of v1, v2, ... */
          Spare_:Nosmp
          Spare[$$x] :: (Lc![’Spare,$$x]; Set[])

/*
#I[1]::   <XSpare

#I[2]::   a:b:c:1

#O[2]:    1

#I[3]::   Spare[a]

#O[3]:    Spare[’ a]

#I[4]::   {a,b,c}

#O[4]:    {1,b,c}

*/
```

XStat

```
        /** Statistical properties of univariate distributions **/

/* Mean[list]
        gives the mean of the values in list. */
        Mean[$list] :: Ap[Plus,$list]/Len[$list]

/* GMean[list]
        gives the geometric mean of the values in list. */
        GMean[$list] :: N[Ap[Mult,$list^(1/Len[$list])]]

/* HMean[list]
        gives the harmonic mean of the values in list. */
        HMean[$list] :: N[Len[$list]/Ap[Plus,Map[1/$1,$list]]]

/* Med[list]
        gives the median of the values in list. */
        Med[$list_?Oddp[Len[$list]]] :: Sort[$list][(Len[$list]+1)/2]
        Med[$list_?Evenp[Len[$list]]] :: \
                (Lc[%]; %:Sort[$list]; (%[Len[$list]/2] + \
                        %[Len[$list]/2+1])/2 )

/* MD[list]
        gives the mean deviation of the values in list. */
        MD[$list] :: Mean[Abs[$list-Mean[$list]]]

/* Var[list]
        gives the variance of the values in list. */
        Var[$list] :: Mean[($list-Mean[$list])^2]

/* SD[list]
        gives the standard deviation of the values in list. */
        SD[$list] :: N[Sqrt[Var[$list] Len[$list]/(Len[$list]-1)]]

/* Range[list]
        gives the range of the values in list. */
        Range[$list] :: Ap[Max,$list]-Ap[Min,$list]

/* RMS[list]
        gives the root mean square of the values in list. */
        RMS[$list] :: N[Sqrt[Mean[$list^2]]]

/* Fract[list,r]
        yields the r fractile of the values in list. */
        _Fract[Init] :: <<LItp
        Fract[$list,$r] :: LItp3[Sort[$list],$r Len[$list]]

/* Q1[list]
        yields the first quartile of the values in list. */
        Q1[$list] :: Fract[$list,1/4]

/* Q3[list]
        yields the third quartile of the values in list. */
        Q3[$list] :: Fract[$list,3/4]

/* QD[list]
        yields the quartile deviation of the values in list. */
        QD[$list] :: (Q3[$list] - Q1[$list])/2

/* Mom[list,n]
        yields the nth moment of the values in list. */
        Mom[$list,$n] :: Mean[$list^$n]

/* Cmom[list,n]
        yields the nth central moment of the values in list. */
        Cmom[$list,$n] :: Mean[($list-Mean[$list])^$n]

/* Cum[list,n]
        yields the nth cumulant of the values in list. */
        Cum[$list,1] :: Mean[$list]
        Cum[$list,2] :: Var[$list]
```

```
        Cum[$list,3] :: Cmom[$list,3]
        Cum[$list,4] :: Cmom[$list,4] - 3 Cmom[$list,2]^2
        Cum[$list,5] :: Cmom[$list,5] - 10 Cmom[$list,3] Cmom[$list,2]

/* Skew[list]
        gives the coefficient of skewness of the values in list. */
        Skew[$list] :: Cmom[$list,3]/SD[$list]^3

/* Kurt[list]
        gives the coefficient of kurtosis of the values in list. */
        Kurt[$list] :: Cmom[$list,4]/SD[$list]^4

/* Excess[list]
        gives the coefficient of excess of the values in list. */
        Excess[$list] :: Cmom[$list,4]/SD[$list]^4 - 3

/* Qskew[list]
        gives the quartile coefficient of skewness of the values in list. */
        Qskew[$list] :: (Q3[$list] - 2 Med[$list] + Q1[$list]) /  \
                                (Q3[$list] - Q1[$list])

/* Expc[op,list]
        yields the expectation value of the operator represented
        by the template op on the values in list. */
        _Expc[Nosmp]:{1,0}
        Expc[$op,$list] :: Rel[Ap[Plus,Map[$op,$list]]]/Len[$list]

/* Char[list,x]
        yields the characteristic function of the values in list. */
        Char[$list,$x] :: Expc[Exp[I $1 x],$list]

/*

#I[1]::  <XStat

#I[2]::  t:Ar[20,`Rand[]]

#O[2]:   {.7512801,.9799641,.3078826,.1427415,.8638817,.1446294,.4414654,

              .2309524,.664948,.1285033,.1148291,.243175,.5178353,.4259514,

              .6094183,.9818086,.02981599,.4496743,.86934847,.4439974}

#I[3]::  Post:N

#O[3]:   N

#I[4]::  Mean[t]

#O[4]:   .4230211

#I[5]::  GMean[t]

#O[5]:   .3851757

#I[6]::  HMean[t]

#O[6]:   .1837451

#I[7]::  Med[t]

#O[7]:   .4337884

#I[8]::  SD[t]

#O[8]:   .2941784

#I[9]::  Range[t]

#O[9]:   .9581481

#I[10]::  RMS[t]
```

```
#O[10]:   .5110389
#I[11]::  Q1[t]
#O[11]:   .1427415
#I[12]::  Q3[t]
#O[12]:   .6094183
#I[13]::  Skew[t]
#O[13]:   .4057557
#I[14]::  Kurt[t]
#O[14]:   1.824871
#I[15]::  Expc[Exp,t]
#O[15]:   1.592872
*/
```

XStr

```
                              /** Struve functions **/

SStr_Ldist

SStr[1]: StrH[$n,$z] -> BesY[$n,$z] + Pi^(-1/2) Sum[($z/2)^(-2m+$n-1)\
                        (2m)!*2^(-2m)/m!/($n-m-1/2)!,m,0,$n-1/2]
                                                    /* MOS p. 115 */

SStr[2]: StrL[$n,$z] -> BesI[$n,$z] + 2/Pi (-1)^($n-1/2) BesK[$n,$z]\
                        -Pi^(-1/2) Sum[(-1)^m ($z/2)^(-2m+$n-1) (2m)!*2^(-2%\
                        (m!*($n-m)!),m,0,$n-1/2]
                                                    /* MOS p. 115 */

SStr[3]: StrH[$n_Natp[-$n+1/2],$z] -> (-1)^(-$n-1/2) BesJ[-$n,$z]
                                                    /* MOS p. 115 */

SStr[4]: StrH[1/2,$z] -> ($z/2 Pi)^(-1/2)(1-Cos[$z])
                                                    /* MOS p. 115 */

SStr[5]: StrL[-1/2,$z] -> ($z/2 Pi)^(-1/2)(1-Cosh[$z])
                                                    /* MOS p. 115 */
```

XStr0

```
          /** Basic character string manipulation **/

/* Char[str,i]
          yields the ith character in the string str. */
          Char[$str,$i_:(0<=$i<CLen[$str])] :: Impl[Expl[$str][$i]]

/* CLen[str]
          gives the number of characters in the string str. */
          CLen[$str] :: Len[Expl[$str]]

/* CRev[str]
          reverses the string str. */
          CRev[$str] :: Impl[Rev[Expl[$str]]]

/* CJoin[str1,str2,...]
          concatenates the strings str1, str2, ... */
          CJoin[$$s] :: Impl[Ap[Cat,Map[Expl,List[$$s]]]]

/*
#I[1]::   <XString

#I[2]::   t:"A character string"

#O[2]:    "A character string"

#I[3]::   Char[t,4]

#O[3]:    h

#I[4]::   Char[t,2]

#O[4]:    " "

#I[5]::   CLen[t]

#O[5]:    18

#I[6]::   CRev[t]

#O[6]:    "gnirts retcarahc A"

#I[7]::   CJoin[%,t,t]

#O[7]:    "gnirts retcarahc AA character stringA character string"

*/
```

XStr1

```
        /**  Further character string manipulation  **/

<XList1

/* CRep[str1,str,i]
        replaces the character at position i in str by the string
        str1. */
        CRep[$str1,$str,$i] :: (Lc|[%i]; %i:Exp|[$str]; %i[$i]:Exp|[$str1];  \
                                    Imp|[Flat[%i]])

/* CIns[str1,str,i]
        inserts the character string str1 at position i in str. */
        _CIns[Init] :: <XList0
        CIns[$str1,$str,$i] :: Imp|[Ins[Exp|[$str1],Exp|[$str],$i]]

/* CPos[form,str]
        yields a list of the positions of the substring form
        in the string str.  */
        CPos[$form,$str] :: LPos[Exp|[$form],Exp|[$str]]

/* CS[str,rep1,rep2,...]
        applies successively the replacements repi for substrings of the
        string str. Each replacement is used until it is no longer applicable.
        The special "generic character" $ represents any single character. */
        CS[$str,$$reps] :: Imp|[Ap[LS,Cat[{Exp|[$str]}, \
                S[Map[Exp|[$1[1]]->Exp|[$1[2]],List[$$reps]], \
                        (|Exp|["$"][1])->$%1]]]]

/*

#I[1]::  <XStr1

#I[2]::  s:"the cat in the hat"

#O[2]:   "the cat in the hat"

#I[3]::  CRep[RRRR,s,5]

#O[3]:   "the RRRRat in the hat"

#I[4]::  CIns[" not",s,8]

#O[4]:   "the cat not in the hat"

#I[5]::  CPos[the,s]

#O[5]:   {1,12}

#I[6]::  CS[s,the->a,$at->XXXXX]

#O[6]:   "a XXXXX in a XXXXX"

*/
```

XSum

```
                    /** Summation of series **/

                    /* Canonical forms for sums */

/* Canonical form for limits */
SSum[1] : Sum[$e,$i,$1,$2] --> Sum[S[$e,$i->$i-$1+1,-1,1],$i,1,$2-$1+1]

/* Sums of rational functions */
SSum[2] : Sum[$e,$i,$1,$2] --> Sum[Pf[$e,$i],$i,$1,$2]

                    /* Simplification of sums */

/* Remove Nosmp property for Sum */
_Sum[Nosmp]:

Sum[$x+$$x,$i,$1,$2] :: Sum[$x,$i,$1,$2] + Sum[$$x,$i,$1,$2]

Sum[$x,$i_;(~In[$i,$x]),$1,$2] :: ($2-$1+1) $x

Sum[$$n $x,$i_;~In[$i,$$n],$1,$2] :: $$n Sum[$x,$i,$1,$2]

Sum[$x/($$n $y),$i_;~In[$i,$$n],$1,$2] :: Sum[$x/$y,$i,$1,$2]/$$n

                    /* Sums of positive powers */

Sum[$i,$i,1,$n] : $n($n+1)/2

Sum[$i^($m_;Natp[$m]),$i,1,$n] : (Ber[$m+1,$n+1]-Ber[$m+1])/($m+1)

Sum[(-$a)^$i $i^($m_;Natp[$m]),$i_;~In[$i,$a],1,$n] : \
        ((-1)^$n Eul[$m,$n+1] - Eul[$m,0])/2

                    /* Geometric progression */

Sum[$a^$i,$i_;~In[$i,$a],1,$n] : ($a^$n - 1)/($a - 1)
```

XSumPR

```
                    /** Special output form for Sum **/

          <XPR
          _Sum[Pr][$expr,$var,$start,$end] :: \
          Fmt[{{1,0},{3,0},{1,-1},{1,1}},PSig,$expr,$var=$start,$end]

/*

#I[1]::  <XSum#PR

#I[2]::  1+Sum[1/i^a,i,1,Inf]

             Inf
             ---
                 -a
#O[2]:*  1 + /     i
             ---
             i = 1

*/
```

XSympol

```
                    /**  Generate symmetric polynomials  **/

/* Sympol[n,x]
        generates a list of all symmetric polynomials in n variables
        x[i]. */
        Sympol[$n_;Natp[$n],$x] :: (Lcl[%x]; \
                                Ps[Ex[Prod[(1+$x[%i] %x),%i,1,$n]],%x,0,$n][5])
```

XTEx

```
                    /**  Tensor expansion  **/

/* TEx[f[x1,x2,...],reor]
        constructs a sum of tensor obtained by reordering the xi
        and assigning the factors specified by the permutation
        symmetries reor. */
        <XPerm0
        <XArperm
        TEx[$f[$$x],$reor] :: (Lcl[%i,%o,%t,%c,%f]; _%f[Reor]:$reor; \
        %o:Ap[%f,List[$$x]]; %t:0; Ap[Plus,Map[%i:Apper[$1,List[$$x]]; \
                If[In[%f,%c:Ap[%f,%i]/%o],0,Inc[%t,%c];%c Ap[$f,%i]], \
                        Arperm[Len[List[$$x]]]]]/%t)

/*

#I[1]::  <XTEx

#I[2]::  TEx[f[a,b],Sym]

         f[a,b] + f[b,a]
#O[2]:   ---------------
                2

#I[3]::  TEx[f[a,b,c],Sym]

         f[a,b,c] + f[a,c,b] + f[b,a,c] + f[b,c,a] + f[c,a,b] + f[c,b,a]
#O[3]:   ------------------------------------------------------------------
                                        6

#I[4]::  TEx[f[a,b,c],Cyclic]

         f[a,b,c] + f[b,c,a] + f[c,a,b]
#O[4]:   ------------------------------
                        3

*/
```

XTr1

```
        /** Elementary transcendental functions - 1 **/

STr_Ldist

        /* 1.  Exponential and Logarithm */

        /* 1.1  Functional Relations */

        /* 1.2 Relations to Other Functions */

STr[1,2,1]:   Exp[$x] -> Cos[I $x]-I Sin[I $x]

STr[1,2,2]:   Exp[$x] -> Cosh[$x]+Sinh[$x]
```

XTr21

```
            /** Elementary transcendental functions - 2.1 **/

                /* 2.1  Circular Functions */

                    /* 2.1  Special Cases */

STr_Ldist

    Sin[Pi/60] : ((Sqrt[6]+Sqrt[2]) (Sqrt[5]-1)\
        -2(Sqrt[3]-1) (Sqrt[5+Sqrt[5]]))/16

    Sin[Pi/30] : (Sqrt[30-6 Sqrt[5]]-Sqrt[5]-1)/8

    Sin[Pi/20] : (Sqrt[10]+Sqrt[2]-2Sqrt[5-Sqrt[5]])/8

    Sin[Pi/15] : (Sqrt[10+2Sqrt[5]]-Sqrt[15]+Sqrt[3])/8

    Sin[Pi/12] : (Sqrt[6]-Sqrt[2])/4

    Sin[Pi/10] : (Sqrt[5]-1)/4
```

XTr22

```
/** Elementary transcendental functions - 2.2 **/

/* 2.2 Pythagorean Relations */

STr_Ldist

STr[2,2,1]:  Sin[$x] -> Sqrt[1-Cos[$x]^2]

STr[2,2,2]:  Cos[$x] -> Sqrt[1-Sin[$x]^2]

STr[2,2,3]:  Tan[$x] -> Sqrt[Sec[$x]^2-1]

STr[2,2,4]:  Cos[$x] -> 1/Sqrt[1+Tan[$x]^2]

STr[2,2,5]:  Sin[$x] -> 1/Sqrt[1+Cot[$x]^2]

STr[2,2,6]:  Tan[$x] -> 1/Sqrt[Csc[$x]^2-1]
```

XTr23

```
/** Elementary transcendental functions - 2.3 **/

/* 2.3 Angle-sum Relations */

STr_Ldist

STr[2,3,1]:  Sin[$x+$$y] -> Sin[$x] Cos[$$y]+Cos[$x] Sin[$$y]

STr[2,3,2]:  Cos[$x+$$y] -> Cos[$x] Cos[$$y]-Sin[$x] Sin[$$y]

STr[2,3,3]:  Tan[$x+$$y] -> (Tan[$x]+Tan[$$y])/(1-Tan[$x] Tan[$$y])
```

XTr24

```
/** Elementary transcendental functions - 2.4 **/

/* 2.4 Multiple Angle Relations */

Sin[$n_?Evenp[$n] $x]::Sin[$x] (Sum[(-1)^k Comb[$n-k-1,k]\
    (2Cos[$x])^($n-2k-1),k,0,$n/2-1])

Sin[$n_?Oddp[$n] $x] :: Sin[$x] (Sum[(-1)^k Comb[$n-k-1,k]\
    (2Cos[$x])^($n-2k-1),k,0,$n/2-1/2])

Cos[$n_?Evenp[$n] $x] :: 2^($n-1) Cos[$x]^$n+\
  $n/2Sum[(-1)^k Comb[$n-k-1,k-1] (2Cos[$x])^($n-2k),k,1,$n/2]

Cos[$n_?Oddp[$n] $x] :: 2^($n-1) Cos[$x]^$n+\
  $n/2Sum[(-1)^k Comb[$n-k-1,k-1] (2Cos[$x])^($n-2k),k,1,$n/2-1/2]
```

XTr25

```
/** Elementary transcendental functions - 2.5 **/

/* 2.5 Power Relations */

Sin[$x_,Evenp[$n]]$n :: 1/2^($n) (Sum[(-1)^($n/2-k) 2Comb[$n,k]\
      Cos[2($n/2-k) $x],k,0,$n/2-1]+Comb[$n,$n/2])

Sin[$x_,Oddp[$n]]$n :: 1/2^($n-1) (Sum[(-1)^($n/2+k-1/2) Comb[$n,k] \
      Sin[($n-2k) $x],k,0,$n/2-1/2])


Cos[$x_,Evenp[$n]]$n :: 1/2^($n) (Sum[2Comb[$n,k]\
      Cos[2($n/2-k) $x],k,0,$n/2-1]+Comb[$n,$n/2])

Cos[$x_,Oddp[$n]]$n :: 1/2^($n-1) (Sum[Comb[$n,k] \
      Cos[($n-2k) $x],k,0,($n-1)/2])
```

XTr26

```
/** Elementary transcendental functions - 2.6 **/

/* 2.6 Fractional Angle Relations */

STr_,Ldist

STr[2,6,1]:  Sin[$x] -> Sqrt[(1-Cos[2 $x])/2]

STr[2,6,2]:  Cos[$x] -> Sqrt[(1+Cos[2 $x])/2]

STr[2,6,3]:  Tan[$x] -> Sqrt[(1-Cos[2 $x])/(1+Cos[2 $x])]

STr[2,6,4]:  Tan[$x] -> (1-Cos[2 $x])/Sin[2 $x]

STr[2,6,5]:  Tan[$x] -> Sin[2 $x]/(1+Cos[2 $x])
```

XTr27

```
/** Elementary transcendental functions - 2.7 **/

/* 2.7 Products of Sines and Cosines */

STr_,Ldist

STr[2,7,1]:  Sin[$x] Sin[$y] -> Cos[$x-$y]/2-Cos[$x+$y]/2

STr[2,7,2]:  Cos[$x] Cos[$y] -> Cos[$x+$y]/2+Cos[$x-$y]/2

STr[2,7,3]:  Sin[$x] Cos[$y] -> Sin[$x+$y]/2+Sin[$x-$y]/2
```

XTr28

```
        /** Elementary transcendental functions      - 2.8  **/
                /* 2.8  Sums of Functions  */

STr_:Ldist
STr[2,8,1]:   Sin[$x]+Sin[$y] -> 2Sin[($x+$y)/2] Cos[($x-$y)/2]
STr[2,8,2]:   Sin[$x]-Sin[$y] -> 2Cos[($x+$y)/2] Sin[($x-$y)/2]
STr[2,8,3]:   Cos[$x]+Cos[$y] -> 2Cos[($x+$y)/2] Cos[($x-$y)/2]
STr[2,8,4]:   Cos[$x]-Cos[$y] -> -2Sin[($x+$y)/2] Cos[($x-$y)/2]
STr[2,8,5]:   Tan[$x]+Tan[$y] -> Sin[$x+$y]/(Cos[$x] Cos[$y])
STr[2,8,6]:   Tan[$x]-Tan[$y] -> Sin[$x-$y]/(Cos[$x] Cos[$y])
```

XTr29

```
              /** Elementary trancendental functions - 2.9  **/
        /* 2.9  Squares of Sines and Cosines  */

STr_:Ldist
STr[2,9,1]:   Sin[$x]^2-Sin[$y]^2 -> Sin[$x+$y] Sin[$x-$y]
STr[2,9,2]:   Cos[$x]^2-Cos[$y]^2 -> -Sin[$x+$y] Sin[$x-$y]
STr[2,9,3]:   Cos[$x]^2-Sin[$y]^2 -> Cos[$x+$y] Cos[$x-$y]
```

XTr2a

```
           /** Elementary transcendental functions - 2.10  **/
                /* 2.10  Relations to Other Functions  */

STr_:Ldist
STr[2,10,1]:  Sin[$x] -> -I (Exp[I $x]-Exp[-I $x])/2
STr[2,10,2]:  Sin[I $x] -> I Sinh[$x]
STr[2,10,3]:  Cos[$x] -> (Exp[I $x]+Exp[-I $x])/2
STr[2,10,4]:  Cos[I $x] -> Cosh[$x]
STr[2,10,5]:  Tan[$x] -> I (1-Exp[I $x])/(1+Exp[I $x])
STr[2,10,6]:  Tan[I $x] -> I Tanh[$x]
```

XTr311

```
/**  Elementary transcendental functions - 3.1.1  **/
/* 3.  Inverse Circular Functions  */
/* 3.1  Functional Relations  */
/* 3.1.1  Complex Argument  */
STr_Ldist
STr[3,1,1]:  Asin[$z] -> Pi/2-Acos[$z]
STr[3,1,2]:  Acos[$z] -> Pi/2-Asin[$z]
Str[3,1,3]:  Atan[$z] -> Pi/2-Atan[1/$z]
STr[3,1,4]:  Atan[$z] -> -Pi/2-Atan[1/$z]
```

XTr312

```
/**  Elementary transcendental functions - 3.1.2  **/
/* 3.1.2  Real Argument between -1 and 1  */
STr_Ldist
STr[3,1,5]:  Asin[$x] -> Pi/2-Asin[Sqrt[1-$x^2]]
STr[3,1,6]:  Asin[$x] -> Asin[Sqrt[1-$x^2]]-Pi/2
STr[3,1,7]:  Asin[$x] -> Acos[Sqrt[1-$x^2]]
STr[3,1,8]:  Asin[$x] -> -Acos[Sqrt[1-$x^2]]
STr[3,1,9]:  Asin[$x] -> Atan[$x/Sqrt[1-$x^2]]

STr[3,1,10]:  Acos[$x] -> Pi/2-Acos[Sqrt[1-$x^2]]
STr[3,1,11]:  Acos[$x] -> Pi/2+Acos[Sqrt[1-$x^2]]
STr[3,1,12]:  Acos[$x] -> Asin[Sqrt[1-$x^2]]
STr[3,1,13]:  Acos[$x] -> Pi-Asin[Sqrt[1-$x^2]]
STr[3,1,14]:  Acos[$x] -> Pi/2-Atan[$x/Sqrt[1-$x^2]]

STr[3,1,15]:  Atan[$x] -> Pi/2-Atan[Sqrt[1-$x^2]]
STr[3,1,16]:  Atan[$x] -> -Pi/2-Atan[Sqrt[1-$x^2]]
STr[3,1,17]:  Atan[$x] -> Asin[$x/Sqrt[1+$x^2]]
STr[3,1,18]:  Atan[$x] -> Pi/2-Acos[$x/Sqrt[1+$x^2]]
```

XTr32

```
        /**  Elementary transcendental functions - 3.2  **/
             /* 3.2  Addition Relations  */

STr_Ldist

STr[3,2,1]:   Asin[$x]+Asin[$y] -> Asin[$x Sqrt[1-$y^2]+$y Sqrt[1-$x^2]]
STr[3,2,2]:   Asin[$x]-Asin[$y] -> Asin[$x Sqrt[1-$y^2]-$y Sqrt[1-$x^2]]
STr[3,2,3]:   Acos[$x]+Acos[$y] -> Acos[$x $y+Sqrt[(1-$x^2) (1-$y^2)]]
STr[3,2,4]:   Acos[$x]-Acos[$y] -> Acos[$x $y-Sqrt[(1-$x^2) (1-$y^2)]]
STr[3,2,5]:   Atan[$x]+Atan[$y] -> Atan[($x+$y)/(1-$x $y)]
STr[3,2,6]:   Atan[$x]-Atan[$y] -> Atan[($x-$y)/(1+$x $y)]
STr[3,2,7]:   Asin[$x]+Acos[$y] -> Asin[$x $y+Sqrt[(1-$x^2) (1-$y^2)]]
STr[3,2,8]:   Asin[$x]+Acos[$y] -> Acos[$y Sqrt[1-$x^2]-$x Sqrt[1-$y^2]]
STr[3,2,9]:   Asin[$x]-Acos[$y] -> Asin[$x $y-Sqrt[(1-$x^2) (1-$y^2)]]
STr[3,2,10]:  Asin[$x]-Acos[$y] -> Acos[$y Sqrt[1-$x^2]+$x Sqrt[1-$y^2]]
```

XTr33

```
        /**  Elementary transcendental functions - 3.3  **/
             /* 3.3  Relations to Other Functions  */

STr_Ldist

STr[3,3,1]:   Asin[$x] -> -I Asinh[I $x]
STr[3,3,2]:   Acos[$x] -> I Acosh[I $x]
STr[3,3,3]:   Atan[$x] -> -I Atanh[I $x]
STr[3,1,4]:   Asin[$x] -> -I Log[(1-$x^2)+I $x]
STr[3,1,5]:   Acos[$x] -> -I Log[$x+I (1-$x^2)]
STr[3,1,6]:   Atan[$x] -> I/2Log[(1-I $x)/(1+I $x)]
```

XTr4

```
/**  Elementary transcendental functions - 4  **/

/* 4.1  Relations to Other Functions  */

STr_Ldist

STr[4,1,1]:   Sinh[$x] -> (Exp[$x]-Exp[-$x])/2

STr[4,1,2]:   Sinh[$x] -> -I Sin[I $x]

STr[4,1,3]:   Cosh[$x] -> (Exp[$x]+Exp[-$x])/2

STr[4,1,4]:   Cosh[$x] -> Cos[I $x]

STr[4,1,5]:   Tanh[$x] -> (Exp[$x]-Exp[-$x])/(Exp[$x]+Exp[-$x])

STr[4,1,6]:   Tanh[$x] -> -I Tan[I $x]

/* All other relations may be obtained by using [4,1,1] - [4,1,6] to convert
to circular functions and using the relations in Sec 2. */
```

XTr5

```
/**  Elementary transcendental functions - 5  **/

/* 5.  Inverse Hyperbolic Functions  */

/* 5.1  Relations to Other Functions  */

STr_Ldist

STr[5,1,1]:   Asinh[$z] -> -I Asin[I $z]

STr[5,1,2]:   Acosh[$z] -> I Acos[I $z]

STr[5,1,3]:   Atanh[$z] -> -I Atan[I $z]

STr[5,1,4]:   Asinh[$x] -> Log[$x+Sqrt[1+$x^2]]

STr[5,1,5]:   Acosh[$x] -> Log[$x+Sqrt[$x^2-1]]

STr[5,1,6]:   Atanh[$x] -> 1/2Log[(1+$x)/(1-$x)]

/* All other relations may be obtained by using [5,1,1] - [5,1,6] to convert
to circular functions and using the relations in Sec 3. */
```

XTup

```
                    /**  n-tuples  **/

/* Tupo[tot,n]
        yields a list of all possible ordered n-tuples of tot elements. */
        Tupo[$tot_:Natp[$tot],$n_:Natp[$n]] :: Tupo[$tot,$n] : \
        Cat[Tupo[$tot-1,$n],Map[Cat[$1,{$tot}],Tupo[$tot-1,$n-1]]]
        Tupo[$tot,0] : {{}}
        Tupo[$tot,$n_:($n>$tot)] : {}

/* Tup[tot,n]
        yields a list of all possible unordered n-tuples of tot elements. */
        _Tup[Init] :: <XList0
        Tup[$tot_:Natp[$tot],$n_:Natp[$n]] :: Tup[$tot,$n] : \
                Cat[Tup[$tot-1,$n],Flat[Map[Ar[$n,Ins[$tot,$1,$2]], \
                                        Tup[$tot-1,$n-1]],1]]
        Tup[$tot,0] : {{}}
        Tup[$tot,$n_:($n>$tot)] : {}

/* Tupa[tot,n]
        yields a list of unordered n-tuples of tot elements, allowing
        repetitions of an element. */
        Tupa[$tot,$n] :: Flat[Ar[Ar[$n,$tot],List],$n-1]

/* Pairs[tot]
        yields a list of all possible partitionings of tot elements
        into pairs. */
        _Pairs[Init] :: <XSets
        Pairs[$n_:Natp[$n/2]] :: \
                Union[Map[Sort,Trans[{Tupo[$n,$n/2], \
                Map[Cmpl[$1,Ar[$n]],Tupo[$n,$n/2]]},$n/2],2]]

/*

#I[1]::  <XTup

#I[2]::  Tupo[3,2]

#O[2]:   {{1,2},{1,3},{2,3}}

#I[3]::  Tup[3,2]

#O[3]:   {{2,1},{1,2},{3,1},{1,3},{3,2},{2,3}}

#I[4]::  Tupa[3,2]

#O[4]:   {{1,1},{1,2},{1,3},{2,1},{2,2},{2,3},{3,1},{3,2},{3,3}}

#I[5]::  Pairs[4]

#O[5]:   {{{1,2},{3,4}},{{1,3},{2,4}},{{1,4},{2,3}}}

*/
```

XTuring

```
                    /**  Turing machine simulation  **/

/* Tape[i]
        is the ith symbol on the data tape.
        (Tape[$i]:1 defines a tape consisting solely of 1's) */

/* Spec[st1,symb1]
        is of the form {symb2,st2} and specifies that when the machine
        is in state st1 and reads the symbol symb1 from the tape, it
        writes symb2 in place of symb1 on the tape, and makes a transition
        to state st2. The machine halts when no applicable Spec exists. */

/* Left
        is a special symbol which when read causes the tape to advance under
        the reading head one symbol to the left. */

/* Right
        is a special symbol causing the tape to advance to the right. */

/* The symbol Null represents a blank position on the tape. */

/* Start[pos,st0]
        starts the Turing machine in state st0 and at position pos on
        the tape; if the operation of the machine terminates, it yields
        the final position and state. */
        Start[$pos,$st0] :: (Lc|[%pos,%st]; %pos:$pos; %st:$st0; \
                 Rpt[Next[Spec[%st,Tape[%pos]]],Inf]; {%pos,%st})
        Next[$x_=~Valp[$x]] :: Ret[]
        Next[{Left,$st}] :: (%st:$st; Dec[%pos])
        Next[{Right,$st}] :: (%st:$st; Inc[%pos])
        Next[{$symb,$st}] :: (Tape[%pos]:$symb; %st:$st)

/*

#I[1]::  <XTuring

#I[2]::  Tape[$i_=$i<5]:1

#O[2]:   1

#I[3]::  Spec[1,0]:{Right,1}

#O[3]:   {Right,1}

#I[4]::  Spec[1,1]:{0,2}

#O[4]:   {0,2}

#I[5]::  Spec[2,0]:{Right,2}

#O[5]:   {Right,2}

#I[6]::  Spec[2,1]:{Right,1}

#O[6]:   {Right,1}

#I[7]::  Start[1,1]

#O[7]:   {5,1}

#I[8]::  Tape

#O[8]:   {[3]: 0, [1]: 0, [$i _= (5 > $i)]: 1}

*/
```

XUnFlat

```
                        /**  List unflattening  **/

/* UnFlat[list,n]
        collects successive sets of n entries in list into sublists. */
        UnFlat[$list_=Contp[$list],$n_=Natp[Len[$list]/$n]] :: \
                Cat[Ar[{{1,Len[$list],$n}},Ar[$n,$list[$1+$2-1]]]]
        UnFlat[$list_=Contp[$list],$n_=Natp[$n]] :: \
                Cat[Ar[{{1,Len[$list],$n}},Cat[Ar[$n,$list[$1+$2-1], \
                $1+$2<=Len[$list]+1]]]]

/*
#I[1]::  <XUnFlat

#I[2]::  t:Ar[10]

#O[2]:   {1,2,3,4,5,6,7,8,9,10}

#I[3]::  UnFlat[t,2]

#O[3]:   {{1,2},{3,4},{5,6},{7,8},{9,10}}

#I[4]::  UnFlat[t,3]

#O[4]:   {{1,2,3},{4,5,6},{7,8,9},{10}}

*/
```

XUnmark

```
                /**  Remove Marks  **/

/* Unmark[expr]
        removes all marks in expr. */
        Unmark[$expr] :: S[$expr,`$->$]
```

XVecan

```
        /**  Three-dimensional vector analysis  **/

/* C0  :  list of orthonormal coordinates
   SF  :  list of scale factors associated with each coordinate
                (diagonal components of metric). */

/* Vecp[v]
        yields 1 if v is a contiguous list of three elements, representing
        a 3-dimensional vector, and 0 otherwise. */
        Vecp[$v] :: Contp[$v] & Len[$v]=3

/* VAbs[vec]
        yields the norm (modulus) of the vector vec. */
        VAbs[$list_=Contp[$list]] :: Sqrt[Ap[Plus,$list^2]]

/* Default Cartesian coordinate system: */
        C0 : {x,y,z}
```

```
        SF : {1,1,1}

/* Assignments for other coordinate systems: */
        /* Cartesian: */    CAR :: {CO:{x,y,z},SF:{1,1,1}}
        /* Circular cylindrical: */  CYL :: {CO:{r,phi,z},SF:{1,r,1}}
        /* Spherical: */    SPH :: {CO:{r,th,phi},SF:{1,r,r Sin[th]}}
        /* Further coordinate systems given in Morse & Feshbach, sect. 5 */

/* Scf[cart]
        computes the scale factors SF for the coordinates CO
        from the list cart of values for the Cartesian coordinates
        x,y,z in terms of the curvilinear coordinates CO[1],CO[2],CO[3]. */
        Scf[$cart] :: (SF:Ar[3,VAbs[D[$cart,CO[$1]]]])

/* Arc
        arc length ds^2 with respect to coordinates CO. */
        Arc :: Sum[SF[%1]^2 Dt[CO[%1]]^2,%1,1,3]

/* Vol
        volume element dV in coordinates CO. */
        Vol :: Prod[SF[%1] Dt[CO[%1]],%1,1,3]

/* Grad[f]
        gradient of scalar expression f with respect to coordinates CO. */
        Grad[$f] :: Ar[3,D[$f,CO[$1]]/SF[$1]]

/* Dvg[f]
        divergence of vector (list) f with respect to CO. */
        Dvg[$f_?Vecp[$f]] :: Sum[D[$f[%1] (SF[1] SF[2] SF[3])/SF[%1]^2, \
                        CO[%1]], %1, 1, 3]/(SF[1] SF[2] SF[3])

/* Curl[f]
        curl of vector expression f with respect to CO. */
        Curl[$f_?Vecp[$f]] :: \
                {(D[SF[3] $f[3], CO[2]] - D[SF[2] $f[2], CO[3]])/(SF[2]SF[3]),\
               (D[SF[1] $f[1], CO[3]] - D[SF[3] $f[3], CO[1]])/(SF[3]SF[1]), \
                (D[SF[2] $f[2], CO[1]] - D[SF[1] $f[1], CO[2]])/(SF[1]SF[2])}

/* Lap[f]
        Laplacian of scalar expression f with respect to CO. */
        Lap[$f] :: Sum[D[(SF[1] SF[2] SF[3])/SF[%1]^2 D[$f,CO[%1]], \
                        CO[%1]], %1, 1, 3]/(SF[1] SF[2] SF[3])

/*

#I[1]::  <<Vecan

#I[2]::  t:{x^2+a y^2,x y^3,x y z}

                 2     2     3
#O[2]:   {a y  + x ,x y ,x y z}

#I[3]::  Dvg[%]

                          2
#O[3]:   2x + x y + 3x y

#I[4]::  Grad[%]

                      2
#O[4]:   {2 + y + 3 y ,x + 6x y,0}

#I[5]::  Vol

#O[5]:   Dt[x] Dt[y] Dt[z]

#I[6]::  SPH

#O[6]:   {{r,th,phi},{1,r,r Sin[th]}}

#I[7]::  Vol
```

```
                2
#O[7]:    r  Dt[phi] Dt[r] Dt[th] Sin[th]

#I[8]::   r^2 Sin[th]^3 Cos[phi]

                2                3
#O[8]:    r  Cos[phi] Sin[th]

#I[9]::   Lap[%]

                2                    2      2                    4
          - r  Cos[phi] Sin[th]  + 3 r  Cos[phi] Sin[th]

                                2                    2        2
                          + 9 r  Cos[phi] Cos[th]  Sin[th]
#O[9]:    ------------------------------------------------
                               2
                              r  Sin[th]

#I[10]::  Ex[%]

                                3                                    2
#O[10]:   3Cos[phi] Sin[th]  - Cos[phi] Sin[th] + 9Cos[phi] Cos[th]  Sin[th]

*/
```

XWarn

```
                    /** Warning messages **/

$x/0 :: (Prh[$x/0,"generated"];)
0^$x_;($x<0) :: (Prh[0^$x,"generated"];)
Log[0] :: (Prh["Log[0] generated"];)
Gamma[$n_;Natp[-$n]] :: (Prh[Gamma[$n],"generated"];)

<$f :: (Prh[$f,"file unavailable"];)

($a : $b) :: (Prh[$a:$b,"impossible assignment"];)
($a :: $b) :: (Prh[$a::$b,"impossible assignment"];)

Minv[$m_;Listp[$m]] :: (Prh[$m,"inversion impossible"];)

/*

#I[1]::   <XWarn

#I[2]::   2/0

2
-       generated
0

          2
#O[2]:    -
          0

*/
```

XWatch

```
                    /** Watching external file input **/
/* Watch[file]
        inputs the specified external file, printing each assignment
        in the file before it is made. */
        Watch[$file] :: (Set_;Setd_;Trace; <$file; _Set[Trace]:_Setd[Trace]:)
```

XWhi

```
                    /** Whittaker function **/

SWhi _: Ldist

SWhi[1]:   WhiM[$l,$m,$z] -> Exp[-$z/2]$z^(1/2+$m)WhiM[1/2+$m-$l,1+2$m,$z]

SWhi[2]:   WhiW[$l,$m,$z] -> Gamma[-2$m]/Gamma[1/2-$m-$l]WhiM[$z]\
                            +Gamma[2$m]/Gamma[1/2+$m-$l]WhiM[$l,-$m,$z]

SWhi[3]:   WhiM[$k,$m,$z] -> Exp[-$z/2]$z^($m+1/2)Chg[1/2+$m-$k,1+2$m,$z]

SWhi[4]:   WhiW[$l,$m,$z] -> Exp[-$z/2]$z^(1/2+$m)KumU[1/2+$m-$l,1+2$m,$z]

                    /* for special cases, see XChg and XKumU */
```

XWron

```
                    /** Wronskian and Jacobian **/

/* Wron[{y1,y2,...},x]
        forms the Wronskian of {y1,y2,...} with respect to x, which
        must vanish if the yi are linearly dependent. */
        Wron[$list,$x] :: Det[Ar[Len[$list],D[$list,{$x,$1-1}]]]

/* Jac[{x1,x2,..},{u1,u2,..}]
        forms the Jacobian of the transformation from {x1,x2,...}
        to {u1,u2,...}. */
        Jac[$x,$u_;Len[$u]=Len[$x]] :: Det[Ar[Len[$x],D[$x,$u[$1]]]]

/*

#I[1]::  <XWron

#I[2]::  Wron[{x^2,Exp[x]},x]

                        2
#O[2]:   -2x Exp[x] + x  Exp[x]

#I[3]::  Jac[{x^2+y^2,x(y-1)},{x,y}]

                                 2
#O[3]:   -2y (-1 + y) + 2 x

*/
```

XYear

```
/* Year[month,day]
        gives the number of each day in a non-leap year */

Month : {[Jan]:31, [Feb]:28, [Mar]:31, [Apr]:30, [May]:31, [Jun]:30, \
                [Jul]:31, [Aug]:31, [Sep]:30, [Oct]:31, [Nov]:30, [Dec]:31}
Year : (Lc![%t];%t:0;Ar[{{{Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec}}},\
                %t+Ar[Inc[%t,Month[$%1]]]])


/*

#I[1]::   Year[Aug,29]

#O[1]:   241
*/
```

XYoung

```
                /**  Young graphs  **/

/* {i1,i2,...} represents a Young graph with i1 boxes in the top row,
        i2 in the second, ..... */

/* PDim[list]
        yields the dimensionality of the representation of the symmetric
        group corresponding to the young graph specified by list. */
        PDim[$list] :: Ap[Plus,$list]!*Prod[Prod[$list[i]-$list[j]+j-i, \
                        j,i+1,Len[$list]],i,1,Len[$list]] /  \
                        Prod[($list[i]+Len[$list]-i)!,i,1,Len[$list]]

/* YGMult[yg1,yg2]
        yields a list of Young graphs occuring in the product of the
        representations of the symmetric group corresponding to yg1
        and yg2. */
        YGMult[$!1,$!2] :: (Lc![%!1,%!2]; \
        %!1:{Ar[Len[$!1],Ar[$!1[$1],0]]};\
        %!2:Flat[Ar[Len[$!2],Ar[$!2[$1],$1]]]; \
        Do[%i,Len[%!2],%!1:Flat[Map[YGadd[$%1,%!2[%i]],%!1],1];\
        %!1:Cat[Ar[Len[%!1],%!1[$1],,YGt0[$1]&YGt1[$1,%!2[%i]]& \
                        YGt2[$1,%!2[%i]]]]]; Map[Len,%!1,{2}])

        YGadd[$list,$n] :: \
        Cat[Ar[Len[$list],YGadd1[$list,$1,$n]],{Cat[$list,{{$n}}]}]
        YGadd1[$list,$pos,$n] :: \
        Cat[Ar[$pos + -1,$list],{Cat[$list[$pos],{$n}]},\
         Ar[{{$pos + 1,Len[$list]}},$list]]

        /* Test for legal graphs */
        YGt0[$list] :: Ap[Ge,Map[Len,$list]]
        YGt1[$list,$n]::Ap[Uneq,Map[$1[2],Pos[$n,$list]]]
        YGt2[$list,$n] :: YGt2p[Del[0,Flat[Map[Rev,$list]]],$n]
        YGt2p[$list,$n] :: \
        (Lc![%t,%r]; %r:Ar[$n,0]; For[%t:1;%c:1,%c<=Len[$list]&Ap[Ge,%r], \
        Inc[%c],%![$list[%c]]:%![$list[%c]]+1]; If[%c>Len[$list],1,0])
```

```
        YGt3[$list,$sun] :: $sun >= Len[$list]

/*
#I[1]::  <<Young

#I[2]::  YGMult[{1,1},{1,1}]

#O[2]:   {{2,2},{2,1,1},{2,1,1},{1,1,1,1}}

#I[3]::  Map[PDim,%]

#O[3]:   {3,2,3,3,1}

#I[4]::  PDim[{5,3,2,2,1}]

#O[4]:   21450

*/
```

XZeta

```
                    /** Riemann zeta function **/

SZeta _: Ldist

SZeta[1]:       Zeta[$z] -> 2(2 Pi)^($z-1) Sin[Pi/2 $z] Gamma[1-$z] Zeta[1-$z
                                                        /* MOS p. 19 */

                    /* Special values */

Zeta[0]: -1/2

SZeta[2]:       Zeta[$m _? Natp[-$m]] -> -Ber[-$m+1]/(-$m+1)
                                                        /* MOS p. 19 */

SZeta[3]:       Zeta[$m _? Natp[-$m/2]] -> 0
                                                        /* MOS p. 19 */

SZeta[4]:       Zeta[$m _? Natp[-$m/2+1/2]] -> Ber[-$m+1]/($m-1)
                                                        /* MOS p. 19 */

SZeta[5]:       Zeta[$m _? Natp[$m/2]] -> (-1)^($m/2+1) (2 Pi)^$m/2/(2 $m)!*\
                 Ber[$m]
                                                        /* MOS p. 19 */

SZeta[6]:       Zeta[$m _? Natp[$m/2]] -> (2 Pi)^$m /2/(2 $m)!*Abs[Ber[$m]]
                                                        /* MOS p. 19 */


                /* The generalized zeta function */

          /* Special values for the argument and parameter */

SZeta[7]:       Zeta[$z,1] -> Zeta[$z]
                                                        /* MOS p. 23 */

SZeta[8]:       Zeta[$z,1] -> 1/(2^$z-1) Zeta[$z,1/2]
                                                        /* MOS p. 23 */

SZeta[9]:       Zeta[0,$a] -> 1/2 - $a
                                                        /* MOS p. 23 */

SZeta[10]:       Zeta[$m_?Natp[1-$m],$a] -> Ber[-$m+1,$a]/($m-1)
                                                        /* MOS p. 23 */
```