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 "SMP Reference Manual". The "SMP Primer" offers a more pedagogical but incomplete treatment.

The information facility [1.2] in SMP provides on-line keyword-driven access to the contents of this document.

This summary describes all standard system-defined projections [2.3] in SMP. Many enhancements and additional facilities are provided in the "SMP Library". Contents of the library may be located and perused through the information facility. Complete documentation on the SMP Library is available separately.

The procedure for initiating an SMP job is 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 are 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. • [3.1, 3.6] • **Smp** [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 are indicated by •.

1. Basic system operation

1.1 Input and output

In standard mode the prompt for the *i*th input line 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]

In display input mode [10.4] a pointer may be used to locate and select parts of displayed expressions.

Printed output is given in a two-dimensional format based on standard mathematical notation. Specification of certain output characteristics [A.6] will cause printing of large expressions to pause periodically, issuing <**pause**> as a prompt. Printing will resume on entry of ⊂*newline*⊃.

• [2.12]

• **Lpr** [10.1]

Graphical output may also be obtained [10.2].

1.2 Information

(Implementation dependent)

?⊂*keywords*⊃

enters information mode and retrieves available information associated with ⊂*keywords*⊃. ⊂*keywords*⊃ may be a single word or a phrase of several words specifying the topic on which information is sought. Typographical case is ignored, and some words may be stripped to canonical roots.

??⊂*pattern*⊃

enters information mode and retrieves available information associated with the symbols, projectors or external files whose names match ⊂*pattern*⊃. ⊂*pattern*⊃ is a single "word" possibly containing the characters *****, which matches an arbitrary sequence of characters, and **.**, which matches an arbitrary single character. A literal ***** is entered by **^***, a literal **.** by **^.**.

• **Dsp** [10.6]

?

enters information mode and prompts for further input.

When information has been retrieved, a menu of items related to ⊂*keyword*⊃ or ⊂*pattern*⊃ is displayed; the user is prompted for a command to select an item or to initiate further searching. In information mode, the command **?** displays a summary of the commands available.

Information is provided on external files in the SMP Library as well as on all standard facilities in SMP. Facilities enabling users to add information on their own external files to the information mode database are described in the Implementation Notes.

• **Init** [10.6]

1.3 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 seconds [A.8]) required to generate **#O[*i*]**.
<Ldist>
- **Time** [10.8]
- **Clock** [10.9]

@i Equivalent to **#O[*i*]**.
<Ldist>

- **%%, %I, %O, %T** [6.3]

Pre Symbol whose value (if any) is taken as a template [2.7] and applied as a "preprocessor" on each input expression.

Post Symbol whose value (if any) is taken as a template and applied as a "postprocessor" on each output expression.

1.4 External files and job recording

External files are input by *<file*.
- [10.3]
- **Dsp, Dir** [10.6]
- **Load** [10.7]

All input and output expressions are entered into a record file usually named **smp.out**. (Implementation dependent)
- **Open, Put** [10.3]
- **Hard, Save** [10.6]

1.5 Termination and real-time interrupts

(Implementation dependent)

⊂*input termination character*⊃
signifies termination of the current procedure. In standard and edit modes, it causes termination of the present job; in subsidiary mode, it results in return to the previous mode.
- **Ret** [6.3]
- **Exit** [10.6]

⊂*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.
- **Get** [10.3]

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

1.6 Monitor escapes

(Implementation dependent)

!⊂*monitor command*⊃
given directly (with no prior characters typed) at an input prompt executes the specified monitor (shell) command; the original input prompt is then reissued.

- **Run** [10.6]

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*⊃. ⊂*space*⊃ ⊂*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 entire edit buffer.

Local editing

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

Global editing

	Print the text in the edit buffer and return to the present line.
720.if 240. Append the text in the edit buffer to the specified file.	
q	Exit from edit mode, discarding the edited expression.
\fIn	Move to the *n*th line of the edit buffer, leaving the present line unchanged. **\+***n* and **\-***n* move *n* lines forward and backward respectively; **\+** and **\-** move one line. Undo the most recent edit command which effected changes in the edit buffer. Undo all changes made during the current editing session, restoring the original contents of the edit buffer.
m	Display levels of parentheses, braces and brackets in the present line.

mg	Display levels of parentheses, braces and brackets throughout the 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 ⊂*e2*⊃ for the *n*th occurrence of character string ⊂*e1*⊃ in the present line.
\ g / ⊂*e1*⊃ **/** ⊂*e2*⊃	Textually substitute for ⊂*e1*⊃ throughout the text.
\ e	Invoke external editor [A.9] on the text in the edit buffer and return the modified text. (Implementation dependent)
\ ! ⊂*monitor command*⊃	Execute the specified monitor command [2.7].

The typographical case of the commands **d**, **p**, **q**, *etc*. is ignored.

- **Ed**, **Edh** [10.5]

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 $x \times 10^p$.

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 $x \times 10^p$. 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 $n0 + n1 \times 10^4 + n2 \times 10^8 + \cdots$ generated when integers with more than 10 digits are input.

F[*n1***,***n2***,...,(***expt***:0),(***dig***:6)]**
Floating point number $(n1 \times 10^{-4} + n2 \times 10^{-8} + \cdots) \times 10^{expt}$ 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, and are evaluated for all projections carrying property **Ldist** [4].

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.

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 **" "** serve only to delimit the symbol name and do not affect simplification of the symbol; they are printed on output only when necessary.
• [A.5]

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.

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 [6.3].

Some special system-defined symbols are:

Null Input and output as a blank [2.10]. Often used to indicate "default" filters [0, 2.3].

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** [1.3]
- **%%**, **%I**, **%O**, **%T** [6.3]
- **Pi**, **E**, *etc.* [8.4]
- **Terminal** [10.3]

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 to select a part in the value of **f**.

- **Proj** [7.3]

Unless the projector **f** carries the properties **Tier**, **Flat**, **Comm** or **Reor** [4], **f[x,y]** is equivalent to **f[x][y]**, and the filters **x** and **y** are used successively (from left to right) to select a part in the value of **f**. The value of **f[x]** is found first, and then the projection of the resulting expression with the filter **y** is obtained. When **f** does carry **Tier** properties, filters separated by commas are instead used together.

Projections from **f** whose values were assigned before the **Tier**, **Flat**, **Comm** or **Reor** properties were assigned may be converted to tiered format by the projection **Tier** [7.7].

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.

- **L** [10.4]

[*x1***,***x2***,...]** or **Np[***x1***,***x2***,...]**
: is a special system-defined null projection representing a sequence of expressions. A first step in the simplification of any projection is the replacement of null projections appearing as its filters by explicit sequences of filters. Sequences of values in contiguous lists may also be given as null projections [2.4]. The value of a list element whose index is a null projection is extracted by projection with a sequence of filters corresponding to all those of the null projection. Such list elements are generated by assignments for projections carrying the properties **Tier**, **Flat**, **Comm** or **Reor** [4], or containing explicit null projections as filters (the partial simplification in **Set** [3.2] does not replace such null projections). No values may be assigned [3.2] directly 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. Values corresponding to indices consisting of null projections [2.3] (as generated by assignments for projections with properties **Tier**, **Flat** or **Reor** [4]) are extracted by sequences of filters.

- **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]
- **L** [10.4]
- **At** [7.2]

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 **-1** through *-n*.
{±*n***}**	Level ±*n*.
{±*n1***,**±*n2***}**	Levels ±*n1* through ±*n2*.
{*-n1***,***n2***}**	The intersection of levels **-1** through *-n1* and levels 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). 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 taken 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*. Although this further projection may be considered as a single filter 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. If any of the *di* are multi-generic symbols, the first is paired with the maximal set of *si*. Any unpaired *di* are left unreplaced.
`*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** [10.5]

2.9 Commentary input

Any input (including ⊂*newline*⊃´s) 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	**#O[***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*	**Get[***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***]** **Prset[***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***]**	

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] =, ~=, >, >=, <, and <= may be input together; if # and ## represent input forms for two such operators, then *x1 # x2 ## x3* is treated as (*x1 # x2*) & (*x2 ## x3*). However, *x1* ~= *x2* ~= *x3* is treated as (*x1* ~= *x2*) & (*x2* ~= *x3*) & (*x1* ~= *x3*); in general, combinations of the form *expr1* ~= *expr2* ~= *expr3* ~= ... assert inequality of all the *expri*.

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.

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**
Div[x1,x2]	$\frac{x1}{x2}$
Pow[x1,x2]	$x1^{x2}$

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].

- **L** [10.4]

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.

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. Each filter is simplified in turn, unless the value (*k*) of any corresponding **Smp** property carried by the projector is **0**. The simplification of a filter is carried out until its value no longer changes, or until any projectors not carrying property **Rec** have appeared recursively at most *k* times (see below). If any filter is found to be extended [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**, **Comm** or **Reor** properties [4, 7.7] are cast into canonical form. If the projector carries the property **Sys** [4] built-in simplification routines are invoked for system-defined projections. External programs are used when applicable if the projector carries the property **Cons** [4, 10.7].
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*. The filters are used together if the projector carries the properties **Tier**, **Flat**, **Comm** or **Reor**. If the required part is present, any necessary replacements for generic symbols are performed (and values defined by generic assignments [3.2] are inserted); the projection is then 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. Sequences of filters in the projection are used together in matching null projections appearing as list indices. **Flat**, **Comm**, and **Reor** properties are accounted for in this matching process.

Lists — Immediate values [3.2] for entries of a list are simplified in turn. Indices are not simplified.

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.

Simplification of a projection may involve further projections from the same projector (recursion). Unless a projector *g* carries the property **Rec**, it may appear recursively at most *k* times in the simplification of a filter in a projection from *f* before steps 2 and 3 in the simplification of the *f* projection are performed. The value of *k* for a particular filter is determined by the value of any corresponding **Smp** property for **f**. The default is **Inf**, causing each filter to be simplified until it no longer changes, regardless of the number of recursive appearances of *g*. Other values of *k* allow steps 2 and 3 in the simplification of the projection from *f* to be performed even though the filter may not be "completely simplified". When *k* is **0**, the filter is left entirely unsimplified. The **Smp** properties of projectors such as **Mult** and **Plus** have value **1** since simplifications of their projections are valid regardless of the state of simplification of their filters.

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

- **Dep** [7.4]

Smp[*expr***,(***levspec***:1),(***rpt***:1),(***domcrit***:1),(***max***:1)]**
simplifies parts of *expr* in the domain specified by *levspec* and *domcrit*, simplifying at most *rpt* times per pass through the expression and making at most *max* passes.
- **Smp** [4]

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*.

- **Ev** [3.7]

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**, **Comm**, 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 **-1**, **0**, or a positive 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*.

*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).
- **Lcl** [6.3]
- **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 system 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* 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 [1.4] 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*: **0**)**]**
<Ldist>
yields the numerical value of *expr* in terms of real or complex decimal numbers, maintaining if possible an accuracy of at most *acc* significant figures, and setting all numerical coefficients smaller in magnitude 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. Complex numerical values are treated when possible.
- **Eq** [5]

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

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.

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 projections for which the **Smp** property [4] has the value **0**.

3.7 Partial simplification

Ev[*expr***]**
yields a held form obtained by simplifying *expr* but leaving unsimplified any delayed values which appear.

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].

The properties **Gen**, **Mgen**, **Cham**, **Smp**, **Tier**, **Flat**, **Reor** and **Comm**, 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 Symbol with system-defined values for projections.

Cons Symbol for which some projections are evaluated by binary code internally loaded by **Cons** or **Load** [10.7].

Inline When a program invoking a projection from ***s*** is constructed by **Cons** or **Prog** [10.7], the program will use directly the value of the projection rather than call an intermediate language function corresponding to ***s***.

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.

Smp The *i*th filter in a projection from ***s*** is simplified until any projector not carrying the property **Rec** has appeared recursively at most ***_s*[Smp][*i*]** times [3.1]. If the value of ***_s*[Smp][*i*]** is **0**, the filter is left unsimplified. If no value is given, it is taken by default as **Inf**, and the filter is simplified until it no longer changes. If the value of ***_s*[Smp]** is a number or symbol, it is taken to apply to all filters.

Ser Filters in projections from ***s*** are simplified from left to right even when parallel processing is possible.

Rec Arbitrary recursive appearances of ***s*** as a projector are permitted [3.1].

Pr The value of ***_s*[Pr]** is used as the printing format 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***.
• [10.10]

Tier The projections ***s*[x][y]** and ***s*[x,y]** are distinguished [2.3]; filters separated by commas are used together in the extraction of parts specified by projections. Values assigned to projections are represented by list entries with null projection indices containing the complete sequences of filters in the projections.

Flat All projections from ***s*** are flattened [7.7], so that ***s*** is treated as an "associative n-ary function". Assignments for projections are made as if ***s*** carried the property **Tier**.

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]. Assignments for projections are made as if ***s*** carried the property **Tier**.

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*,(*h1*:*g1*),(*k1*:*f*),(*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 *hi* appearing as their first filters, and yielding projections from *gi*. 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 for differentiation [9.4] and other purposes.

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.

File The value of *_s*[**File**] is used as the default *format* [A.6] entry in any *filespec* for the file *s* [10.3]. *_s*[**File**] is reassigned automatically when *format* is modified in a *filespec* or when the graphics code *gcode* [A.6] is modified in an **Open** [10.3] projection.

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 document.

Type extension is carried out before any **Flat**, **Comm**, or **Reor** properties of the original operator are used. Thus such properties must be specifically given to "relacement extensions" if they are desired.

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; however, **Uneq** is an exception, yielding the conjunction of results for all possible pairs 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** [3.4]

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

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]. However, combinations of the form *expr1* ~= *expr2* ~= *expr3* ~= ... are converted to **(***expr1* ~= *expr2***) & (***expr2* ~= *expr3***) & (***expr1* ~= *expr3***) &** ..., asserting inequality of all the *expri*.

~*expr* or **Not[*expr*]**
yields "true" 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** [3.4]

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

Ord[*expr1*,*expr2*]
yields **+1** if **expr1** is lexically [10.5] ordered before **expr2**, **-1** if *expr1* is lexically ordered after *expr2*, and **0** if they are literally equivalent [2.6].
- **Reor**, **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.

Sel[*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*;...*exprn* or **Proc[***expr1***,***expr2***,...,***exprn***]**
represents a procedure 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 unsimplified form of each *expri* is maintained throughout the execution of a **Proc**, so as to allow resimplification if required by a control transfer.

Unless specified otherwise by **Lcl** projections, all expressions may be accessed and affected in any procedure. Objects local to a procedure are (*cf.* [1.3]):

%% The last expression (other than **Null** [2.2]) generated by simplification of a segment in the 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 seconds [A.8]) required to generate **%O[***i***]**.
<**Ldist**>

Get[] initiates an interactive procedure, terminated by ⊂*input termination character*⊃ [1.5] or a **Ret** projection. A prompt **%I[***i***]::** is given for the *i*th input segment.

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.

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.

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**, **Loop**, **For** and **Do**. It may be used to exit interactive subsidiary procedures. **Ret [*expr*, Inf]** returns from any number of nested procedures to standard input mode.

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,...,***n***-1,***n***]**.
Similarly, *f***[***m1***,***m2***]..*f***[***n1***,***n2***]** yields
[*f***[***m1***,***m2***],***f***[***m1***+1,***m2***+***i***],f[***m1***+2,***m2***+2***i***],...,***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,...,***n*
{*s*,*e*,(*i*:**1**)}	*s*, *s* + *i*, *s* + **2***i*,..., *s* + *k***i* where *k* is the largest integer such that *s* + *k***i* is not greater than *e*.
{{*x1*,*x2*,...}}	*x1*, *x2*, ...

and collected into a complete specification {*spec1*,*spec2*,...}. 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 **0** are omitted. Entries whose values would yield **0** on application of the template *vcrit* are also omitted.
- **Outer** [9.6]
- **Dim** [7.4]

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

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

7.2 Template application

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

At[*temp***,***expr***,(***partspec***:L[***expr***])]**
yields the expression obtained from *expr* by applying (if possible) the template *temp* to parts specified by *partspec*. *partspec* may consist of a list giving the filters necessary to select a particular part of *expr*, or of a list of such lists, representing a sequence of parts. The template is applied in turn to each specified part; for overlapping part specifications it is applied first to the deepest parts. *partspec* is obtained by default by graphical part selection in *expr* using **L** [10.4].

Map[*temp***,***expr***,(***levspec***:1),(***domcrit***:1),(***ltemp***:Null),(***max***:Inf)]**
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].

Pos[⟨*form1*⟩**,*expr*,(*temp*:List),(*lev*:Inf),(*max*:Inf)]**
gives a list of the results obtained by applying the template *temp* to sets of filters specifying the positions of (at most *max*) occurrences of parts in *expr* (at or below level *lev* [2.5]) matching any of the *formi*.
• **In** [7.5]

Elem[*expr*,{*n1*,*n2*,...}]
extracts successively the *ni*th 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]). Yields **Inf** for a "static recursive" expression [3.1].

Dim[*list*,(*llev*:Inf)]
gives the ranges of contiguous 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 a held 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*^5)**,** (*acc*: **1*^-12)**]**
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***,** {**Elem[***v***,** {*i*}**]]** are in canonical order (for increasing *i*), or **Ap[***ord***,** {**Elem[***v***,** {*i*}**]**,**Elem[***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),(***max***:1)]**
"distributes" projections from *f* in the domain of *expr* specified by *levspec* and *domcrit* over lists appearing as their filters, making at most *max* passes through *expr*.
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 **List),(***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** [4] are automatically "flattened".

Tier[*list***]**
generates a single level list in which the complete sequence of indices specifying each expression in *list* appear in null projections.

Assignments for projections from symbols carrying the property **Tier** automatically generate lists with a single level in which the filters of the projections appear in indices as null projections [2.3].

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**, and their extensions [4], together with any replacements defined by **Dist** or **Powdist** properties [4].

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".

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 *fmult*. 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***[$$1],$n_?Natp[$n]]** -> *fmult***[Repl[***fplus***[$$1],$n]** followed by expansion of the result.

- **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**),(*rpt*:**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**, repeating the process until the result no longer changes, or at most *rpt* times.

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].

The SMP Library provides many additional relations and transformations 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 may be obtained by **N** projections [3.4]. Elementary arithmetic, numerical and transcendental functions, and mathematical constants may be evaluated to arbitrary numerical precision.

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.

Definitions of differentiation [9.4], integration [9.4], and power series expansion [9.5] for elementary arithmetic and transcendental functions are included; definitions for other functions are given in external files.

All projections representing mathematical functions carry the property **Ldist** [4], as well as the properties **Sys** and **Tier**.

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 functions

expr1* + *expr2* +** ... or ***expr1* - *expr2* +** ... or **Plus[expr1***,***expr2***,...]**
<Smp:1,Comm,Flat>

expr1* * *expr2* *** ... or **Mult[expr1***,***expr2***,...]**
<Smp:1,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

Floor[*x***]**
the greatest integer not larger than the real number *x* ("floor" of *x*).

Ceil[*x***]**
the least integer not smaller than the real number *x* ("ceiling" of *x*).

Mod[*n***,***m***]**
the real number *n* modulo the real number *m*.

Sign[*x***]**
1 or **-1** if *x* is a positive or negative real number, and **0** if *x* is zero.
- **P** [5]

Theta[*x***]**
Heavyside step function θ(*x*).

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

Integrals and derivatives involving **Theta** and **Delta** are treated.

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 $\pi = 3.14159...$

E $e = 2.71828...$

Euler Euler-Mascheroni constant $\gamma = 0.577216...$ [AS 6.1.3; GR 9.73; MOS 1.1]

Deg $\pi/180 = 0.0174533...$ (number of radians in one degree)

Phi Golden ratio $\phi = 1.61803...$

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

8.6 Combinatorial functions

n*!** or **Fctl[n***]**

Factorial [AS 6.1.6]. (**B** [2.1] projections are generated when necessary.)

• **Gamma** [8.7]

n*!!** or **Dfctl[n***]**

Double factorial [AS 6.1.49 (footnote); GR p.xliii]. (**B** [2.1] projections are generated when necessary.)

Comb[*n***,***m*[1],(*m*[2],...,*m*[*k*-1],...,(*m*[*k*]:*n*-*m*[1]-*m*[2]-...-*m*[*k*-1]))]**

Multinomial coefficient $(n;m[1],m[2],\ldots,m[k-1],m[k])$ [AS 24.1.2].

Comb[*n***,***m***]** gives binomial coefficient $\binom{n}{m}$ [AS 6.1.21; MOS 1.1].

Sti1[*n***,***m***]**

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

Sti2[*n***,***m***]**

Second kind Stirling numbers $\mathbf{S}_n^{(m)}$ [AS 24.1.4].

Part[*n***]**

Partition function [AS 24.2.1].

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 27.9.1].

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

8.7 Gamma, Zeta and related functions

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)].

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].

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

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

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

Ei[*z*]
Exponential integral Ei(z) [AS 5.1.2; GR 8.2; MOS 9.2.1; BMP 6.9.2.(25)].
- **Erf** [8.8]

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).

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

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].
- **Fctl** [8.6]

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

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)].

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

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

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].

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

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

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].

8.8 Confluent hypergeometric and related functions

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

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

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

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

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)].

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

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** [8.7]

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)].

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

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)].

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

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

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

Par[*p*,*z*]
Parabolic cylinder functions $D_p(z)$ [† AS 19.3.7; GR 9.240; MOS 8.1.1].

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

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

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

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

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

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].

8.9 Hypergeometric and related functions

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].

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

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

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].

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** [8.7]

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

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

8.10 Elliptic functions

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

JacSn[*x*,*m*] **JacCn[*x*,*m*]** **JacDn[*x*,*m*]**
JacCd[*x*,*m*] **JacSd[*x*,*m*]** **JacNd[*x*,*m*]**
JacDc[*x*,*m*] **JacNc[*x*,*m*]** **JacSc[*x*,*m*]**
JacNs[*x*,*m*] **JacDs[*x*,*m*]** **JacCs[*x*,*m*]**
JacAm[*x*,*m*]

Jacobian elliptic functions $\mathrm{Sn}(x|m)$ *etc.* [AS 16.1; GR 8.144; MOS 10.3].

Jacth[*i*,*z*,*m*]
Jacobi θ functions $\theta_i(z|m)$ [AS 16.27; GR 8.18; MOS 10.2].

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

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

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

8.11 Number theoretical functions

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

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

• **Mod** [8.3]

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.

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

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

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

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

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

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

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

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

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** [8.3]

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

- **Gcd** [8.11]

Rslt[*expr1***,***expr2***]**
polynomial resultant of *expr1* and *expr2* with respect to the "base" *form*.

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** [8.11]

- **Ex** [7.8]
- **Cb**, **Col** [7.9]

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***,⟨⟨***var1***,(***lower1***:0),(***upper1***:***var1***),(***step1***:1),(***test1***:1),(***endtest1***:0)⟩,...⟩]**
forms the sum of the values of *expr* when the symbols *vari* successively take on values from *loweri* to *upperi* with increments *stepi*. If the value of *testi* is **0**, the current value of *expr* is not included in the sum. The summation terminates if projection of *endtesti* onto the list {*vari*,*curexpr*,*cursum*}, where *curexpr* is the current value of *expr* and *cursum* is the current partial sum, yields true.

Prod[*expr***,⟨⟨***var1***,(***lower1***:0),(***upper1***:***var1***),(***step1***:1),(***test1***:1),(***endtest1***:0)⟩,...⟩]**
forms the product of the values of *expr* when the symbols *vari* successively take on values from *loweri* to *upperi* with increments *stepi*. If the value of *testi* is **0**, the current value of *expr* is not included in the product. The product terminates if projection of *endtesti* onto the list {*vari*,*curexpr*,*curprod*}, where *curexpr* is the current value of *expr* and *curprod* is the current partial product, yields true.

In multiple sums or products, *var1* is taken as the innermost summation or product variable.

The default *upperi* yield indefinite sums or products; if no *loweri* is specified, it is taken as an internally-generated symbol [2.2].

After each new term is added in a sum or product, any *endtesti* is applied as a template to a list consisting of the current value of *vari*, the last term added, and the current partial sum or product. If the result is determined to be true, no further terms are added, and the current partial sum or product is taken as the complete sum or product.

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

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

- **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*)}...}**,**({ *iterm1* }**)]**
forms the integral of *expr* successively with respect to the variables *var1*, ... between the limits *lower1*, ... and *upper1*, ... The default *upperi* yield indefinite integrals; if no *loweri* is specified, it is taken as an internally-generated symbol [2.2].

Any integral whose indefinite form involves only elementary arithmetic [8.2] and transcendental [8.5] functions is performed explicitly. Any *itermi* are taken as candidate additional terms in the integral.

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 at most order *ord1*, ... Terms proportional to *var1^j1 var2^j2* ... are given when *j1*, *j2*, ... lie within a simplex with vertices (*dji*:*sordi*), (*j1*:*ord1*, *ji*:*sordi*), (*j2*:*ord2*, *ji*:*sordi*), ... *ser***[***i***]** gives the coefficient of *var1^i* in the power series. Composition of a function *f* with a power series will be carried out only if *f* carries the property **Ldist**.

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 at most 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 is 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**)**,**(*temp*:**Eq**)**]**

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*

and applying the template *temp* to each entry of the resulting list.

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***:{1,2,3,...})]**
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***]**
yields the similarity transformation matrix necessary to place *list* in diagonal or Jordan canonical form.

10. Non-computational operations

10.1 Input and output operations

Lpr[*expr***,(***filespec***:Terminal)]**
prints *expr* to the file specified by *filespec* [10.3] in a direct linear format suitable for use as input and in which labelling of parts is manifest [2.10]. The default *filespec* is the standard input/output medium (usually the terminal). **Lpr** 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.

- **Pr** [4]
- **Fmt** [10.1]

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

Rd[(*prompt***:Null),(***filespec***:Terminal),(***eofval***:Null)]**
prints the expression *prompt*, then reads and simplifies one line of input (terminated by ⊂*newline*⊃) from the file specified by *filespec* [10.3]. The default is input from the standard input/output medium (usually the terminal). **Rd** yields the value *eofval* if it encounters the end of the file.

Rdh[(*prompt***:Null),(***filespec***:Terminal),(***eofval***:Null)]**
prints the expression *prompt*, then reads one line of input from the file *filespec* [10.3] and yields its "held" [3.5] form. The default is input from the standard input/output medium (usually the terminal). **Rdh** yields the value *eofval* if it encounters the end of the file.

Fmt[(*prspec***:Null),***expr1***,***expr2***,...]**
when output prints the *expri* in a format specified by *prspec*:

Null	*expr1* followed immediately by *expr2*, ...
positive integer *n*	*expr1* followed after *n* blank spaces by *expr2*, ...
negative integer *-n*	*expr1*, *expr2*, ... printed in tabular format, with tab positions every *n* 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.

Prsize[*expr***]**
yields a list of two elements, the number of horizontal character positions and the number of vertical character positions occupied by the printed form of *expr*.

Sx[*cc***,{***x1***,***x2***,...},(***class***:1),(***prec***:1)]**
when output prints the *xi* 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 printing format 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.

Plot[] clears the plotting area. (Implementation dependent)

The list *plist* (whose sublists are flattened [7.7]) contains:

Pt[{*x*,*y*,(*z*)},(*form*),({(*c1*:*z*)})**]**
: represents a point with coordinates *x*, *y* and *z*, to be printed as *form*. Additional coordinates *ci* may be used to define contours on surfaces. When *form* does not print as a single character, the coordinates are taken to specify its lower left 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.

Surf[*ptlist***,**(*form*)**]**
: represents a three-dimensional surface spanned by points specified by **Pt** projections in *ptlist*. The surface is ruled with contour lines at integer spacings in each of the additional coordinates *ci*.

Axes[{(*x*:**0**),(*y*:**0**),(*z*:**0**)},{(*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.

High-resolution graphics output is generated if a suitable device is available.

10.3 File input and output

In projections which perform input/output operations, a file is specified by a *filespec* filter of the form {*file*,(*lines*),(*format*)}. *lines* and *format* may be used to specify portions of *file* or to specify output format characteristics for *file*. If *lines* and *format* are omitted, *filespec* may be given simply as *file*, without braces; however, a list of such file specifications must be nested as {{*file1*},{*file2*},...}.

file is given as a single symbol, enclosed in " " if necessary [2.2].

lines may be given as *n*, identifying the *n*th line of the file; as *-n*, identifying the *n*th line from the end of the file; or as {*n1*,*n2*}, identifying lines *n1* through *n2*. In input operations, {*file*,*n*} represents the portion of *file* from line *n* to the end of the file. When *lines* is omitted, the *filespec* represents the entire file. In output operations, {*file*,*n*} specifies that output is to be appended after the *n*th line of *file*; contents of *file* from line *n* to the end of the file are overwritten. When *lines* is omitted, output is appended to *file*.

format, a list of the form {(*width*),(*length*),(*prstyle*),(*tabs*),(*gcode*)}, specifies output format characteristics for *file*. A detailed treatment of output characteristics may be found in [A.6].

Terminal
is a system-defined symbol used to denote the standard input/output medium (usually the terminal).
- **Open** [10.3]

<*filespec* or **Get[**(*filespec*:**Terminal**)**]**
enters subsidiary mode [6.3] and reads input up to the first ⊂*input termination character*⊃ from the specified file *filespec*. If ambiguous syntax [1.1, 2] is encountered, a message is printed, and no further input occurs. If *filespec* is successfully input, the projection yields the last output line generated. A blank line yields **Null**.
- **Rd**, **Rdh** [10.1]
- **Load** [10.7]
- **Dir** [10.6]

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

Open[(*filespec*:**Terminal**),(*gcode*)**]**
causes all subsequent standard and graphics [10.2] mode input and output expressions to be entered into the specified file *filespec* [10.3]. If the graphics code *gcode* [A.6] is given, its value overrides any *gcode* value in *filespec* and modifies the value of *_file***[File]** [4] accordingly. Terminal hardware characteristics [A.7] may be specified by **Init** [10.6].

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.

10.4 Display operations

(Implementation dependent)

In display input mode, the pointer (often a display cursor) of an interactive display terminal may be used to locate and select parts of displayed expressions. The pointer is positioned either through an analog device, or from the terminal keyboard: ⊂*up*⊃ ⊂*down*⊃ ⊂*left*⊃ and ⊂*right*⊃ move the pointer one position up, down, left or right respectively; ⊂*tab up*⊃ *etc*. move several positions. ⊂*select*⊃ selects the minimal expression covering the current pointer position and any previous pointer positions indicated by ⊂*mark*⊃. ⊂*scroll forward*⊃ and ⊂*scroll back*⊃ display subsequent and previous sections of output.

An **Init** projection [10.6] may be used to specify the character sequences typed from the terminal to indicate ⊂*up*⊃ *etc*. and the corresponding character sequences sent to the terminal to perform these operations.
• [A.7]

L[*expr*,(*temp*:**List**)**]**
enters display input mode displaying the expression *expr* and yields the result from application of the template *temp* to the set of filters corresponding to the subexpressions selected.
- **At** [7.2]

10.5 Textual operations

Ed[*expr*,(*filespec*)**]**
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. If *filespec* [10.3] is given, the specified file (or portion thereof), with the textual form of *expr* appended, is placed in the edit buffer; upon exit from **Ed**, **Null** is returned and the edited text is written back to the file.

Edh[*expr***,(***filespec***)]**
enters edit mode [1.7] with the text of a partially simplified form [3.5] of *expr* in the edit buffer. If *filespec* [10.3] is given, the specified file (or portion thereof), with the text of a partially simplified form [3.5] of *expr* appended, is placed in the edit buffer; upon exit from **Edh**, **Null** is returned and the edited text is written back to the file.
• **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***,(***n***)]**
gives a list of numerical codes for each of the characters appearing in the textual form of *expr* (as printed by **Lpr**). A second filter yields the *n*th code in the list. Characters are numbered from **0** to **94** in the order:
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
#%$⊂*space*⊃**!"&'()*+,-./:;<=>?@[\]^_‘{|}~**
Negative integers *-c* correspond to absolute character codes *c* in the native system character set.
• [A.5]

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

10.6 External operations

Init[(*par1***,***par2***,...)]**
assigns various external parameters as specified in [A.9], and yields a list of the values of all the assigned parameters. Any filter or part of a filter given as **Null** is taken to represent the previous value of that filter or part.

Exit[(*expr***)]**
terminates the current job, passing *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) command or program [A.3], using the textual forms of the *argi* as input; the text of any output generated is simplified and given as the image.
• [1.6]

Dsp[(*file***:smp.out),(***sect***:Inf)]**
prints the section of *file* specified by *sect*:

n	First *n* lines
-n	Last *n* lines
{*n1*,*n2*}	Lines *n1* through *n2*

Hard[(*expr***),(***code***),(***gcode***)]**
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.

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

Send[(*uname1***,***uname2***,...)]**
enters send mode: arbitrary text terminated by ⊂*input termination character*⊃ is sent to the locations or users identified by *unamei*. ⊂*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.2] to *dir*. **Dir[]** resets to the directory given at initialization.

10.7 Code file input and generation

(Implementation dependent)

Cons[{ ***proj1*** }**,**({{ ***progfile1*** }**,**(***lang***)}**)**,**(***codefile***),**
({ ***suppfile1*** }**)**,**({ ***symspec1*** }**)**,**(***lang***)]**
generates if possible definitions in intermediate language *lang* for the projectors *proji* which are defined in the current SMP session. These definitions are compiled together with the program files *progfilei*. The resulting machine code, together with the code files *suppfilei*, is either written into the code file *codefile* or, if *codefile* is not specified, used for the remainder of the SMP session to evaluate projections from the *proji*. *symspeci* [10.7] may be used to specify characteristics of the *proji* and of functions in the *progfilei*.

Cons returns a list of the *proji* for which functions were generated.

A **Cons** projection is equivalent to a sequence of **Prog**, **Code**, and (if no *codefile* is specified) **Load** projections.

- **Prog**, **Code**, **Load** [10.7]
- **Cons** [4]

Prog[{{ ***proj1*** }}**,**(***progfile***:**Terminal**)**,**({ ***symspec1*** }**)**,**(***lang***)]**
generates if possible definitions in intermediate language *lang* for the projectors *proji* which are defined in the current SMP session; these definitions are written to the file *progfile* (terminal as default). *symspeci* [10.7] may be used to specify characteristics of the *proji*.

Prog returns a list of the *proji* for which functions were generated.

Prog[*expr***,**(*progfile*:**Terminal**)**,**({ *symspec1* }**)**,**(***lang***)]**
generates if possible code in intermediate language *lang* corresponding to *expr*.

- **Cons**, **Code**, **Load** [10.7]

Code[{ ***proj1*** }**,**({{ ***progfile1*** }**,**(***lang***)}**)**,***codefile***,**({ ***suppfile1*** }**)**,**({ ***symspec1*** }**)]**
compiles intermediate language definitions for the projectors *proji* in the program files *progfilei*; the resulting machine code, together with the code files *suppfilei*, is written into *codefile*, a code file suitable for loading with **Load** or **Cons**. *symspeci* [10.7] may be used to specify characteristics of functions in the *progfilei*.

Code returns{ *proj1* }.

- **Cons**, **Prog**, **Load** [10.7]

Load[{ ***proj1*** }**,*****codefile*****]**
loads machine code definitions for the projectors *proji* in the code file *codefile*; these definitions are used for the remainder of the SMP session to evaluate projections from the *proji*.

Load returns{ *proj1* }.

- **Cons**, **Prog**, **Load** [10.7]
- **Cons** [4]

A program file [A.4] contains definitions in an intermediate compilable language (such as C or FORTRAN).

An object file contains definitions in the native machine language and is typically the output of a compiler and an assembler.

A code file [A.4] is a file generated by **Code** or **Cons** containing machine language definitions to evaluate SMP projections in a form suitable for loading with **Load** or **Cons**. When a code file is generated, characteristics of these definitions may be specified in *symspeci*; this information is entered into the file and used when the definitions are loaded into SMP. Code files may also contain instructions to operating system programs such as the linker. The contents and format of a code file are implementation-dependent.

Following is a detailed treatment of the filters taken by **Cons**, **Prog**, **Code**, and **Load**.

lang

lang is a numerical code specifying the language of any *progfilei* supplied to **Code** or **Cons**, the language of the definitions to be generated by **Prog** or **Cons**, and the default *objlang* for any *symspeci*. Available languages and their numerical codes are given in the Implementation Notes. A default *lang* is set by **Init** [10.6].

proj

In a **Prog** or **Cons** projection, ⦃ *proj1* ⦄ specifies the projectors for which intermediate language definitions are to be generated. A **Null** entry specifies that functions are to be generated for all projectors on which the previous list entry depends. (Entries in ⦃ *proj1* ⦄ may be lists: **{{f,g},}** and **{f,,g,}** are equivalent, both specifying the generation of functions for all projectors invoked by either **f** or **g**, as well as for **f** and **g**.) If no **Null** entries are present in ⦃ *proji* ⦄, **Prog** and **Cons** do not generate functions for such projectors, except when **Cons** is invoked without *codefile*, *progfile*, or *suppfile* arguments.

In a **Code** projection, ⦃ *proj1* ⦄ specifies projectors for which information is to be entered in *codefile* to permit calling of the machine language definitions from SMP when *codefile* is loaded. Such projectors may also be specified to **Cons**.

In a **Load** projection, ⦃ *proj1* ⦄ specifies projectors from which projections are to be evaluated by compiled functions in the *codefilei*. (The names of the projectors must be identical to those used by **Code** or **Cons** when the code files were generated.)

progfile

In ⦃ *progfile1* ⦄,(*lang*)⦄, *lang* specifies to **Code** or **Cons** the language of the files in ⦃ *progfile1* ⦄; *e.g.*, **{{{f1.c,f2.c},1},{src,2}}** specifies that the files **f1.c** and **f2.c** are in language **1** and that the file **src** is in language **2**.

0 is a special numerical code specifying the native machine language, used only to identify object files supplied to **Code** or **Cons**. A code of **0** may not be specified as the *lang* filter in a **Prog** or **Cons** projection.

codefile

codefilei are code files generated by **Code** or **Cons**. When a code file is generated, any information supplied to **Code** or **Cons** in *symspeci* or in *suppfilei* is entered into the file.

suppfile

suppfilei are code files supplied to **Load** or **Cons**. In a **Load** or **Cons** projection, it is not necessary to provide *symspeci* for any functions in the *suppfilei*.

symspec

symspec is a list of the form
{⦃ *symbol1* ⦄**,**(*object*)**,**(*objtype*)**,**(*narg*)**,**⦃ *argtype* ⦄**,**(*objlang*)**}**. *symspeci* are used to specify the characteristics of functions or variables in definitions to be generated by **Prog** or **Cons**, and in *progfilei* supplied to **Code** or **Cons**.

symbol1 is an SMP projector or symbol and *object* is the intermediate language function or variable corresponding to *symbol1*. Unless *object* is given, the name of the function or variable is derived from *symbol1*; the method of derivation is given in the Implementation Notes, and is dependent on *objlang*. If *object* is a variable, *objtype* specifies the data type of the variable. If *object* is a function, *objtype* specifies the data type of the value it returns, *narg* is the number of its arguments, ⦃ *argtype* ⦄ specifies the data types of the arguments, and *objlang* is the intermediate language in which the function is written.

Data types are given by numerical codes:

1 integer

2 single-precision floating-point

3 double-precision floating-point

The default type and any additional types available are given in the Implementation Notes.

A *symspec* may give a single *symbol*, a list of *symboli*, or the symbol **Null**. If a list of *symboli* is given, any values specified in the *symspec* are applied as characteristics for each of the *symboli*. If the symbol **Null** is given, values are applied to all objects for which those characteristics are not otherwise specified; this form of *symspec* may be used to override default values.

If the intermediate language function *object* is being supplied in a *progfile* as the definition for the projector *symbol* (where *symbol* is one of the *proji*), *narg* specifies the number of filters taken by *symbol*. In a **Cons** projection, if *narg* is not given in the *symspec* for *symbol*, SMP attempts to determine *narg* from an invocation of *symbol* in the SMP definition of another of the *proji*. If such an invocation does not exist, or if *object* is being supplied to **Code**, an unspecified *narg* causes projections from *symbol* to call *object* with two arguments, the number of filters in the projection and an array of the filters.

- **Inline** [4]

10.8 Resource management and analysis

Time[*expr***,(***n***:1)]**
simplifies *expr* *n* times, and yields an **Err** projection [2.1] giving the approximate average CPU time in seconds [A.8] used for each simplification.
- **#T** [1.3]
- **%T** [6.3]
- **Clock** [10.9]

Gc[(*frac***)]**
reclaims memory (store) not required for further processing (by "compacting garbage collection"). **Gc[]** reclaims memory as soon as possible, **Gc[***frac***]** whenever a fraction *frac* of the total available memory has been used. **Gc[Inf]** inhibits all further memory reclamation.
- **Init** [10.6]

Mem[]
yields a list giving the number of memory "blocks" used (after last memory reclamation, at present, and maximum so far). (When a finite absolute maximum memory is available [A.8], it is given as a fourth element of the list.) One block is the memory required to store a single symbol (usually 16 bytes [A.8]).
- **Init** [10.6]

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 subexpressions were stored separately.
- **Struct** [10.10]

Share[*expr***]**
minimizes memory used to store *expr* (by sharing memory for identical subexpressions). **Share[]** shares memory for all identical expressions.

10.9 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 CPU time and total elapsed real time (both in seconds [A.8]) since the initiation of the specified process (**0** if the process is not executing).

10.10 Development 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 is initiated.
- **Trace** [4]
- **Ev** [3.7]

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

Appendix. External interface

A.1 Introduction

This appendix describes in general terms features of SMP affected by its external environment. Details of these features vary between different implementations of SMP. Information for a particular implementation is given in the "Implementation Notes".

A.2 External files

If no explicit file directory is specified for an external file to be input by **Get**, a sequence of file directories defined by **Init** is searched in order. The first external file to be found with the required name is used. Typically one or several default "user" directories are searched first, followed by a central "library" directory.

External files for output generated by **Lpr**, **Put** or **Open** are taken to be in the current file directory, unless explicitly specified otherwise.

The record file for each SMP job, usually opened in the initialization file **smp.ini** [A.9], is typically placed in the "user" directory and named **smp.out**.

- **Dir** [10.5]

The names of external files in the SMP Library begin by convention with the letter **X**. Names of files containing syntax modifications [2.11] end with **SX**.

Comments in external files are taken by convention to begin with the following codes:

/**	File or section title
/*K:	Keywords
/*A:	Author
/*S:	Site
/*D:	Date
/*U:	Modification date
/*:	Projection or symbol description
/*R:	References
/*QV:	Related SMP objects or external files
/*E:	Example
/*	General commentary
/*P:	Prerequisite external files
/*W:	Warnings
/*I:	Implementation dependencies
/*F:	Future enhancements
/*B:	Bugs

A preprocessor described in the Implementation Notes may be used to place such external files in a form suitable for typesetting, as in this document. Three fonts are typically used: a text font, an **SMP expression** font, and an *italic* font to represent generic objects. Unless specified otherwise, commentary text is given in the text font. Words beginning with **$** are given in italics, while words beginning with **$#** are given in underlined italics. Text in nested comments is given in SMP expression font. Text not given as commentary (and thus intended as SMP input) is placed in SMP expression font.

A detailed treatment of the external file formatting conventions may be found in the SMP Library. The contents of external files following these conventions may be included in the information mode database using external programs described in the Implementation Notes.

A.3 External operations

Monitor (shell) commands specified by arbitrary text strings may be executed using **Run** [10.5]. When no explicit file directory is specified for external programs appearing in the commands, a sequence of file directories defined by **Init** is searched in order. The first program to be found with the correct name is used.

Input from SMP to external programs executed by **Run** is usually directed through a channel other than that used for standard terminal input (as specified in the Implementation Notes). Similarly, output to SMP from the external programs is through a channel other than that used for standard terminal output. External programs may thus receive input or generate output on the terminal independent of SMP.

Monitor programs are typically provided to redirect input and output from **Run** to the standard terminal input and output channels for the external program. Typically a program **fromsmp** takes a sequence or list of numbers from **Run**, converts them to floating point form (using *xx***E***ee* rather than *xx****^***ee* format if necessary), and passes them on the standard input channel. **tosmp** typically takes a sequence of numbers (separated by white space) on the standard output channel, converts them if necessary from *xx***E***ee* to *xx****^***ee* format, and passes them on the SMP input channel. The program **smpio** is typically provided to combine these operations; the monitor command to run a numerical external program from **Run** using standard input and output is then **smpio** *program*.

Most projections for external operations [10.6] execute monitor commands specified by **Init**. The monitor command executed by **Run** takes arguments *searchpath infile outfile comm*, where *comm* is the command to be executed, *searchpath* is a list of file directories to be searched for *comm*, *infile* is a file of input containing the textual forms of the second and later filters of the **Run** projection, and *outfile* is a file in which output from *comm* is to be placed. The monitor command executed by **Hard** takes arguments *file code* and prints *file* on the device specified by *code*, deleting *file* on completion. **Send** executes a monitor command with arguments *file uname* which sends text in *file* to the location or user identified by *uname*. **Save** executes a monitor command with arguments *old new* which copies the contents of the file *old* to the file *new*.

The command **\e** in edit mode [1.7] executes a monitor command with argument *file* specified in **Init**, which invokes an interactive text editor on *file* and leaves the edited text in the same named file.

A.4 Code and program files

The file directories searched for code files to be input by **Load** [10.7] are the same as for **Get** [A.2]. Code and program files generated by **Prog** and **Code** are placed in the current file directory, unless explicitly specified otherwise.

The names of code files in the SMP Library begin by convention with the letter **C**. Files giving additional definitions and commentary associated with code files have names beginning with **CX**.

The monitor command to be executed for compilation of intermediate language program files by **Code** and **Cons** is specified by **Init**, and typically takes arguments *outfile file1 file2* . . . . The *outfile* argument is implementation-dependent. On some systems, it may be a relocatable binary output from the compilation and linking of the *filei*. On others, it may be a set of instructions enabling the system linker to collect the files comprising the output from compilation of the *filei* and any *suppfilei* in future **Load** or **Cons** operations. Similarly, **Load** and **Cons** execute a monitor command to prepare code files for loading with arguments *infile outfile adr*. The code contained or referred to in *infile* (corresponding to the meaning of *outfile* above) is relocated to allow loading at hexadecimal address *adr*, and the resulting absolute binary code is placed in *outfile*. The code for the intermediate language function corresponding to the SMP projection to be defined may be required to appear at the beginning of *outfile*.

The default intermediate language for program files is specified in **Init**. The value **1** typically corresponds to the C language (B.W.Kernighan and D.M.Ritchie, "The C Programming Language", Prentice-Hall 1978). The names of variables or functions are if possible taken the same as those of the corresponding SMP symbols or projectors. Textual replacements such as **$→d_** , **#→h_** , and **%→p_** are performed if necessary. The Implementation Notes give details specific to particular installations.

Intermediate language libraries are often included in programs generated by **Prog**, as specified in the Implementation Notes.

A.5 Character codes

The following table correlates characters in the SMP character set with their SMP character codes and octal ASCII equivalents:

0	**0**	(060)
1	**1**	(061)
2	**2**	(062)
3	**3**	(063)
4	**4**	(064)
5	**5**	(065)
6	**6**	(066)
7	**7**	(067)
8	**8**	(070)
9	**9**	(071)
a	**10**	(141)
b	**11**	(142)
c	**12**	(143)
d	**13**	(144)
e	**14**	(145)
f	**15**	(146)
g	**16**	(147)
h	**17**	(150)
i	**18**	(151)
j	**19**	(152)
k	**20**	(153)
l	**21**	(154)
m	**22**	(155)
n	**23**	(156)
o	**24**	(157)
p	**25**	(160)
q	**26**	(161)
r	**27**	(162)
s	**28**	(163)
t	**29**	(164)
u	**30**	(165)
v	**31**	(166)
w	**32**	(167)
x	**33**	(170)
y	**34**	(171)
z	**35**	(172)
A	**36**	(101)
B	**37**	(102)
C	**38**	(103)
D	**39**	(104)
E	**40**	(105)
F	**41**	(106)
G	**42**	(107)
H	**43**	(110)
I	**44**	(111)
J	**45**	(112)
K	**46**	(113)
L	**47**	(114)
M	**48**	(115)
N	**49**	(116)
O	**50**	(117)
P	**51**	(120)
Q	**52**	(121)
R	**53**	(122)
S	**54**	(123)
T	**55**	(124)
U	**56**	(125)
V	**57**	(126)
W	**58**	(127)
X	**59**	(130)
Y	**60**	(131)
Z	**61**	(132)
#	**62**	(043)
%	**63**	(045)
$	**64**	(044)
⊂*space*⊃	**65**	(040)
!	**66**	(041)
"	**67**	(042)
&	**68**	(046)
´	**69**	(047)
(	**70**	(050)
)	**71**	(051)
*****	**72**	(052)
+	**73**	(053)
,	**74**	(054)
-	**75**	(055)
.	**76**	(056)
/	**77**	(057)
:	**78**	(072)
;	**79**	(073)
<	**80**	(074)
=	**81**	(075)
>	**82**	(076)
?	**83**	(077)
@	**84**	(100)
[	**85**	(133)
****	**86**	(134)
]	**87**	(135)
^	**88**	(136)
	89	(137)
`	**90**	(140)
{	**91**	(173)
|	**92**	(174)
}	**93**	(175)
~	**94**	(176)

The Implementation Notes provide a table of character codes for installations using other character sets. Any numerical code *c* in the native system character set may be represented with SMP character code *-c*.

- **Expl**, **Impl** [10.5]

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

A.6 Output characteristics

Characteristics of standard and graphics mode output are specified by codes in **Open** projections or by file specifications in *filespec* filters [10.3]. Output characteristics are often initialized in **smp.ini** [A.9].

- **Open** [10.3]
- **Init** [10.6]

In a *filespec* {*file*,(*lines*),(*format*)}, *format* is given as a list of the form {(*width*),(*length*),(*prstyle*),(*tabs*),(*gcode*)}. (*file* and *lines* are described in [10.3].)

width may be given either as a single number specifying the maximum width of a line or as a list {*leftmar*,*rightmar*} setting the left and right margins. The default value for *width* is {**1,80**}.

length is given as a single number specifying the number of lines to be printed before pausing [1.1]. The default value is **Inf**.

prstyle specifies the maximum length of expressions to be printed in standard two-dimensional format; expressions which would require more than the specified number of lines are to be printed in a direct linear format. *prstyle* may be given as a list {*max*,*lprmax*}, where *lprmax* applies to printing by **Lpr** [10.1] and *max* to all other printing, or as a single number to be taken as the value for both *max* and *lprmax*. The default value for *prstyle* is {**0,0**}; however, at the start of every SMP job, *prstyle* is initialized in the **File** property of the input/output medium **Terminal** to {**Inf,0**}.

- **Lpr** [10.1]

tabs may be given either as a single number *n*, indicating that terminal hardware tab stops are set every *n* columns starting at **1**, or as a list of the tab stop positions. The default value for *tabs* is **0**. *tabs* is used to inform SMP of the hardware tab stop positions, not to reset them.

gcode is a numerical code specifying graphics mode output characteristics. *gcode* may also be given as a filter in an **Open** projection [10.3]. Available numerical codes are described in the Implementation Notes and typically include:

0 Generate output suitable for devices without special graphics capabilities.

1 Generate device-independent codes suitable for input to an external plotting program specified in **Init**.

2 Generate output by executing an external plotting program specified in **Init** using device-independent codes.

3 Generate output suitable for Tektronix 4010 or equivalent.

4 Generate output suitable for DEC GIGI, VT125, or equivalent.

5 Generate output suitable for Tektronix 4025 or equivalent.

The default value of *gcode* is **0**.

Parts of *_file***[File]**, used as the default value for *format*, are automatically reassigned when *format* is modified in a *filespec* or when *gcode* is modified in an **Open** projection. Any part of *format* given as **Null** is taken to represent the previous value of that part.

- **File** [4]

A.7 Terminal characteristics

Terminal characteristics are typically given as a parameter to **Init** [10.6] of the form {*list1*, *list2*}.

list1 specifies the character sequences typed by the user in display mode [10.4] to indicate ⊂*up*⊃ ⊂*down*⊃ ⊂*left*⊃ ⊂*right*⊃ ⊂*tab up*⊃ ⊂*tab down*⊃ ⊂*tab left*⊃ ⊂*tab right*⊃ ⊂*scroll forward*⊃ ⊂*scroll back*⊃ ⊂*mark*⊃ and ⊂*select*⊃. These character sequences are represented as lists of SMP character codes [A.5] and are given in *list1* as values indexed respectively by the symbols **Up**, **Down**, **Left**, **Right**, **TabUp**, **TabDown**, **TabLeft**, **TabRight**, **Forward**, **Back**, **Mark**, and **Select**.

Default values for the character sequences to be given as display mode commands are typically:

⊂*up*⊃	**u**
⊂*down*⊃	**d**
⊂*left*⊃	**l**
⊂*right*⊃	**r**
⊂*tab up*⊃	**U**
⊂*tab down*⊃	**D**
⊂*tab left*⊃	**L**
⊂*tab right*⊃	**R**
⊂*scroll forward*⊃	**+**
⊂*scroll back*⊃	**-**
⊂*mark*⊃	**.**
⊂*select*⊃	⊂*return*⊃

These defaults may be overridden by execution of an external program described in the Implementation Notes which prompts the user for direct typing of the character sequences, generates an **Init** projection assigning appropriate values for *list1*, and places the **Init** projection in an initialization file.

If terminal capabilities have been specified in *list2* (see below), keys marked ↑, ↓, ←, and → are equivalent to ⊂*up*⊃ ⊂*down*⊃ ⊂*left*⊃ and ⊂*right*⊃.

list2 specifies various capabilities of the terminal, and is often generated on initialization. A monitor command executed in **smp.par** [A.9] attempts to ascertain the make and model of the user's terminal and searches for information on the terminal's capabilities in a database typically named **termcap** from a central system file directory. Any information retrieved is placed in a list indexed by symbols representing the terminal capabilities.

Details on the format of terminal descriptions in **termcap** and in *list2* may be found in the Implementation Notes.

A.8 System characteristics

The "block" is the basic unit of memory used in SMP. Its physical size is specified in the Implementation Notes, and is typically 16 bytes. SMP jobs usually allocate memory dynamically. If necessary or desirable, an absolute maximum on memory space may be specified in an **Init** projection.

- **Mem** [10.8]
- **Init** [10.6]

CPU time used by SMP is reported in seconds, to a resolution of one "tick". A tick is typically 1/60 second; its value at a particular installation is given in the Implementation Notes.

- **#T** [1.2]
- **Time** [10.8]
- **Clock** [10.10]

A.9 Initialization and termination

All SMP jobs first read a file typically named **smp.par** from a central system file directory. This file either contains an **Init** projection to define external parameters or generates such a projection by execution of a monitor command through **Run**.

A sequence of file directories specified in **Init** is searched for an initialization file usually named **smp.ini**. This file often includes **Open** projections to define further terminal characteristics and to initiate entry of input and output into a record file typically named **smp.out**. It may also include monitor escapes or **Dsp** projections to print pertinent information.

When an SMP job is terminated, it searches a sequence of file directories specified in **Init** for a termination file usually named **smp.end**. If found, the file is input before final termination. The SMP job passes by default a "successful completion" exit code to the monitor; other exit codes may be specified in **Exit**.

The external parameters to be given in **Init** are specified in the Implementation Notes, and are typically

1. List of file directories to be searched by **Get** and **Load**.
2. List of file directories to be searched by **Run**.
3. List of file directories to be searched in information mode for on-line documentation, interactive instruction scripts, tutorials, *etc*.
4. List of files to be read on initialization.
5. List of files to be read on termination.
6. List of files to be included in the information mode database.
7. List of monitor commands to be executed by **Run**, **Hard**, **Send**, **Save**, **Ed**, **Code**, **Load**, **Plot**.
8. Default code for **Hard**.
9. Default destination for **Send**.
10. Default intermediate language for **Prog** and **Cons**.
11. Default file directory for **Dir**.
12. Terminal characteristics [A.7].
13. Maximum number of memory blocks [A.8] to be allocated.
14. **1** for an interactive job; **{1,0}** for interactive input; **{0,1}** for interactive output.
15. **1** to disable external operations not directly controlled by the SMP job.

Index