Kac Manual

1. Getting started

This manual is for the terminal version of *Kac*. This program is started by typing the command "`kac`" in a terminal window. Then a line

```
Kac, version 5.0, compiled on Wed Jul 19 2000, at 13:06:00 MET
```

should appear, followed by the prompt

```
>
```

indicating that input is expected. In the following user input is always display in typewriter font, showing a prompt followed by the actual input. Details of the command syntax will be given later. The basic syntax has the form

```
> Command Keyword ...  Keyword [parameter]...(parameter)
```

Command and keywords (if any) must be typed as shown, but the parameters can take various values, depending on the command. Commands and keywords are case insensitive, and may be abbreviated as long as this does not lead to ambiguities. Parameters in square brackets are obligatory, those in round brackets optional. Output will be shown as it appears on the screen, without the prompt.

To find out which commands are available, type

```
> Commands
```

To find out what a command does, use `Help`. For example

```
> Help Commands
Command syntax:  Commands
List all commands
Keywords:
All Detail
Enter Keyword >
```

The `Help` command always gives the correct syntax of a command, and in particular the optional and mandatory parameters. In addition it will list the available keywords, in this case `All` and `Detail`. To find out what `Commands Detail` does one may type `Detail` at the `Enter Keyword >` prompt, or type directly at the command prompt

```
> Help Commands Detail
Command syntax:  Commands Detail
List all commands,keywords and descriptions
```

The output of `Command Detail` is in fact a concise manual.

The first thing one always does after starting up the program is to create a conformal field theory (CFT). The simplest non-trivial examples are provided by the untwisted affine Lie algebras. They are created by means a command

```
> Algebra Type [rank r][level k]
```

where the keyword "type" must be equal to A,B,..., or G, the usual allowed types of simple Lie algebras; r is the rank of the simple Lie algebra and k the level of the affine representations to be considered.

After typing the `Algebra` command the spectrum is computed (*i.e.* the Virasoro eigenvalues of the ground states of the representations), as well as various other properties. However, since the number of representations can be huge, the spectrum is not automatically displayed.

Only the number of primaries is shown, for example

```
> Algebra E 8 30
Number of primaries:  20956
```

To display the spectrum one uses the `Display` command, for example

```
> Algebra A 1 2
Number of primaries:  3
> Display
L. Grnd.  Weights
0 (0,2) 0.0000000
1 (2,0) 0.5000000
2 (1,1) 0.1875000
```

The properties of the representations are presented in a number of columns. In this example there are just three: an integer label assigned to each representation, the extended Dynkin labels, and the conformal weight. There can be more columns. How to activate them and what they represent will be discussed later.

The `Display` command has optional parameters. To see them type `Help Display`

```
> Help Display
Command syntax:  Display (field min)(field max)
Display spectrum
(Only for field ‘‘min" or from ‘‘min" to ‘‘max")
Keywords:
Entry Massless
```

The line marked `Command syntax` shows that there are two optional parameters. These parameters have the names “min” and “max” and are preceded by their type, in this case “field”. Parameters of type “field” must be one of the non-negative integers appearing in the column marked “L.” (for “Label”) of the spectrum table. The parameters of the `Algebra A ... G` commands also have types, namely “rank” and “level”. Not all parameters of all commands have definite types. One can see the obligatory and optional parameters and their types using the `Help` command, as shown above. In the case of `Display`, if there is one parameter only the representation with the corresponding label is displayed, and if there are two parameters the corresponding range of representations is displayed.

MOVE TO LATER The help text also reveals the existence of two keywords, “Entry” and “Massless”. This case is an exception to the rule that keywords can be abbreviated. In order to prevent confusion with the parameters “min” and “max” (which should be expressions that are converted to integers), any keywords that compete with parameters must be enterred without abbreviation. If one types `Display Mass`, then `Mass` is assumed to be the name of a variable

The main use of *Kac* is the computation of “modular properties” of CFT’s. A rather basic quantity is the modular transformation matrix S. The command

```
> S
```

computes and displays the entire matrix S_{ij} with $i \leq j$ (S is symmetric). With one parameter i the command `S` displays the i^{th} row of S_{ij}, with two parameters i and j it gives just one matrix element.

The fusion rules of the CFT are shown using the command

```
> Fusion
```

Without parameters all fusion rules are displayed in a self-explanatory format, e.g.

```
(2) x (2) = (0) + (1)
```

With one parameter i all the fusion rules for field "i" are given, with two parameters i and j the fusion rule $(i) \times (j)$ is shown, and finally with three parameters one obtains a single fusion coefficient, for example

```
> Fusion 2 2 1
N_2_2^1=1
```

In general the fusion rules have the form

$$(i) \times (j) = \sum_k N_{ij}{}^k (k)$$

The command

```
> Fusion i j k
```

yields the value of $N_{ij}{}^k$, written as `N_i_j^k`.

1.1. Tensor mode

Tensor products of algebras can be constructed by going first to tensor mode:

```
> Tensor
```

and then typing the factors of the desired tensor product. For example

```
> G A 1 3
> G E 6 2
> G F 4 2
```

builds $A_{1,3} \otimes E_{6,2} \otimes F_{4,2}$. The command `G` has the same keywords and parameters as the `Algebra` command.

The notation comes from a common notation for coset CFT's as G/H. Indeed, there is also a command `H`, which has the effect of tensoring with the "complement" of an algebra. The complement of an algebra $\mathcal{A}$ is defined as having complex conjugate matrices T and S. As a result the formal tensor product

```
> Tensor
> G F 4 1
> H G 2 1
```

produces an algebraic structure that has the same matrices S, T and fusion rules as the coset CFT $F_{4,1}/G_{2,1}$. The spectrum matches that of the actual coset CFT only up to integers. The computation of the exact spectrum is not built in; in fact it does not even matter if "H" can actually be embedded in "G". In general the construction of coset CFT's also requires field identification, see below. Any number of factors G and H can be tensored.

To show the spectrum of a tensor product (or the spectrum modulo integers of a coset CFT) type `Display`. The matrix S and the fusion rules can be computed in tensor mode exactly

as in single algebra mode. However, before computing any modular quantity (such as S or fusion rules) a command must be typed to indicate that the tensor product is completed. This can be `Display`, or, if one does not want to display the spectrum,

```
> Compute
```

.

To return to single algebra mode, one types

```
> Single
```

1.2. Simple currents

To see the simple currents of a CFT type

```
> Browse Simple_currents
> Total simple current order:  4
> Basis decomposition of currents:
> Group:  Z_2 *Z_2
> Nr.  0:  Weight:  0/1 0 0
> Nr.  1:  Weight:  1/1 1 0 (fixed points)
> Nr.  2:  Weight:  1/1 0 1 (fixed points)
> Nr.  3:  Weight:  1/1 1 1 (fixed points)
```

This shows the simple currents of $D_{4,2}$, and the decomposition in terms of $\mathbf{Z}_2 \times \mathbf{Z}_2$ uses internally. The "Nr." refers to the labelling as shown by `Display`. The last column indicates that the current has at least one fixed point (This information is not shown in tensor mode).

1.3. Extended algebras

One may extend the chiral algebra of any CFT by simple currents using the command

```
> Current (fcur current_1)...(fcur current_n)
```

where `current_i` is the component of the current in the i^{th} factor. Any number of mutually local currents can be added (identical currents are not added to the list; dependent ones are added, but if a new current is a product of previous ones it has no effect on the result). After completing the addition of currents use `Compute` (or `Display`) to compute (and display) the spectrum and enable the modular properties.

For example, one can tensor $B_{3,1} \otimes B_{3,1}$ and extend the algebra to obtain $D_{7,1}$ in the following way

```
> Tensor
> G B 3 1
> G B 3 1
> Current 1 1
> Compute
> Number of primaries:  4
```

Field identification in coset CFT's may be thought of as a formal extension by a spin-0 current. For example, the following construction yields the Ising model

```
> Tensor
> G A 1 1
> G A 1 1
```

```
> H A 1 2
> Current 1 1 1
> Display
> Number of primaries:  3
Lbl Comb.  Weights
0 {0,0,0} 0.0000000
1 {0,0,1} 0.5000000
2 {0,1,2} 0.0625000
```

In this case the spectrum happens to be correct, but in general it is only correct up to integers. The second column shows a representative of the field identification orbit of the coset primary.

1.4. Creating new algebras

The results of a tensor procedure with or without additional extensions can be made into a new CFT that is accessible with the `Algebra`, `G` or `H` commands. To do this type after completing the input `Create [CFT_name]`
where "`CFT_name`" is any character string. To reload this algebra type

```
> Algebra Cft [CFT_name]
> G Cft [CFT_name]
> H Cft [CFT_name]
```

in single algebra mode resp. tensor mode. These "CFT"'s can now be used on equal footing with any other algebras.

This several advantages: tensoring can be speeded up by first tensoring subsets, the limits on the number of factors and currents can be evaded, and furthermore there is more information available in single algebra mode than in tensor mode.

2. Built in algebras

2.1. (Untwisted) Affine Algebras

Command:

```
> Algebra Type [rank r][level k]
```

where the keyword "type" must be equal to A,B,..., or G, the usual allowed types of simple Lie algebras; r is the rank of the simple Lie algebra and k the level of the affine representations to be considered. The level must be an integer ≥ 1; the rank is a positive integer ≥ 1 for type A; ≥ 2 for types B and C; ≥ 4 for type D, and the usual values for the exceptional algebras. For the classical algebras lower values of the rank correspond to different types, e.g. $D_3 \sim A_3$.

The labels in the "ground state" column are defined as follows: the first r labels are the ground state Dynkin labels, according to the following labelling; the last entry is the extended Dynkin label.

2.2. Twisted Affine Algebras

Command:

```
> Algebra Type [rank r][level k]
```

where "Type" can be At, Bt, Ct, Ft, Gt or Bc. The latter algebra plays a rôle in the fixed point resolution of certain algebras of type B and C, which is the main reason why the twisted algebras were built in. They are not conformal field theories, and most features are not available for these algebras. The notation is based on the book by J. Fuchs.

Further details are not documented.

2.3. $U(1)$ Algebras

Command:

```
> Algebra U [radius R]
```

This yields the $c = 1$ circle theory with R primaries. R should be even, but odd R is tolerated. Using odd values of R leads to inconsistencies in certain commands. The self-dual theory is obtained for $R = 2$.

2.4. $c = 1$ ORBIFOLDS

The command

```
Algebra Orbifold [pos_int N]
```

gives the $\mathbf{Z}_2$ reflection orbifold of the $R = 2N$ circle theories. The resulting orbifold CFT has $N + 7$ primaries. For more details about these theories see [1](the errors in the formula for S in this paper have been corrected in *Kac*).

Notation

The labelling is as follows:

Label	Representation	Weight
(0,0)	identity	0
(0,2)	"∂X"	1
(0,1)	J	$\frac{N}{4}$
(0,3)	J^c	$\frac{N}{4}$
(1,k)	ϕ_k	$\frac{k^2}{4N}$
(2,1)	σ	$\frac{1}{16}$
(2,2)	σ'	$\frac{1}{16}$
(3,1)	τ	$\frac{9}{16}$
(3,2)	τ'	$\frac{9}{16}$

2.5. MINIMAL MODELS

The command

```
> Algebra Minimal [N_susy N][level k]
```

gives the minimal model with N supersymmetries and "level" k. The number N may take the values 0, 1 or 2, and k must be a positive integer. For example, $N = 0$, $k = 1$ gives the Ising model.

Notation

To be written.

2.6. Displaying some CFT properties

The central charge can be displayed as follows

```
> Browse Central_charge
```

```
> Browse CFT
```

Display several CFT properties.

2.7. Accessing primaries

Each primary (highest weight representation) is assigned an internal label, which is shown (if enabled) in the first column of the `display` command. It is possible to access fields directly by enterring their Dynkin labels using the command

```
> Dynkin (int i_1)...(int i_n)
```

The number of arguments must equal the number of ground state labels (shown in the second column of `display`, if enabled), except for untwisted affine algebras, where it must be one less. This is because in that case the last label is superflous (given the level), and each affine Lie algebra ground state is completely specified by a set of horizontal Lie algebra Dynkin labels.

The `Dynkin` may be called as a function to allow direct specification of primaries in terms of their labels. For example

```
> Display Dynkin(0,1)
```

shows the relevant data for a representation with labels (0,1), for example of the affine Lie algebra A_2.

The inverse of the `Dynkin` command is

```
> Expand [field lbl]
```

which expands the argument into Dynkin labels.

2.8. Examining stored algebras

All algebras in storage can be displayed using

```
> Browse Algebras
```

The columns in the output table give the name of the algebra, the amount of storage used (in Bytes) and the number of primaries. The output format is controlled by the `Table` command (discussed later in this manual), applied to the table named "Algebras".

3. Lie algebra properties

The commands in this chapter are available for untwisted affine Lie algebras, the chiral algebras of WZW-models; some of them also work in other cases.

The command

```
> Browse Lie
```

shows the name and adjoint dimension of horizontal Lie algebra (the number of generators) and the dual Coxeter number. This command works for any algebra and tensor product, except that in the latter case only the number of generators is shown.

```
> Dimension [field lbl]
```

Get dimension of ground state "lbl".

```
> Browse Cartan
```

Show Cartan matrix and its inverse.

3.1. Weyl group properties

```
> Weyl Order
```

Display order of the Weyl group

```
> Weyl Group
```

Display decomposition of Weyl group elements in terms of simple roots. Notation in output: 1.2.3. denotes successive reflection w.r.t. root 3, 2 and 1.

```
> Weyl Multiply [non_neg_int i][non_neg_int j]
```

Multiply Weyl group elements "i" and "j", with labels referring to the first column of the output of the Weyl Group command.

```
> Weyl Reflection [field i](simple_root j)
```

Compute the horizontal Weyl reflection of the horizontal algebra representation of field "i" in the plane orthogonal to simple root "j"; Without second parameter: compute action of all Weyl group elements. The output is in Dynkin labels.

```
> Weyl Orbit [field i]
```

Compute the complete Weyl orbit of field "i". Same as "Weyl reflection i", . except that identical weights are shown only once.

```
> Weyl Parity Orbits (field i)
```

Compute the complete Weyl orbit modulo 2 of field "i" Without parameter:compute all distinct parity orbits.

3.2. Gauss sums

```
> Compute Gauss_sum (field x)
```

Compute Gauss sum:

$$\sum_w e^{\frac{2pii}{k}(w-x/2)^2}$$

Here x is a weight, and the summation is over all weights w in the full Weyl chamber at level k, with a boundary compensation factor. The latter is equal to $\frac{1}{N}$ if a weight is located at the common boundary of N Weyl chambers. In other words, this factor is such that if we were to tile the entire weight space with Weyl chambers, and sum over all tiles, then the total contribution of any weight is the same. The sum may also be written as a sum over the weight-lattice modulo k times the co-root lattice (were the translations by k time the coroots are generated by affine Weyl transformations). The result depends only on the Weyl parity orbit of the Dynkin labels of x modulo 2. If no parameter x is specified the computation is done for all parity orbits. To display the orbits use the command "Weyl Parity Orbits"

The result of this computation is known to be equal to

$$\sqrt{|L_w/L_c|}(\frac{ik}{2})^{r/2}\sum_u(-1)^{\frac{k}{2}(Cu,Cu)+(x,Cu)} \tag{3.1}$$

where L_w and L_c are the weight and co-root lattices respectively. The u's are all 2^r vectors of zeroes and ones, and C is the symmetrized Cartan matrix, formed by the dot-products of the simple co-roots. The vector Cu is the a co-root written in Dynkin basis, a representative of the co-root lattice modulo twice the co-root lattice.

In the output the factor before the sum in (3.1) is omitted.

```
> Compute Gauss_sum Orbits (current i)
```

Compute the complete Weyl orbits modulo 2 in conjugacy class"i" Without parameter: use conjugacy class 0.

3.3. Indices of representations

```
> Indices (field lbl)
```

Compute indices (for rep. "lbl").

4. Simple currents

```
> Browse Simple_currents
```
Show simple current data.
```
> Get Charge [current J][field a]
```
Get the charge of "a" w.r.t. "J".
```
> Simple_fusion [current J](field j)
```
Compute simple current fusion rules.
```
> Get Order [current J][field a]
```
Get the order of the simple current "J" on "a".
```
> Orbit [current J][field i]
```
Compute J-orbit of i.
```
> Check Classes
```
Check that conjugacy class definitions match simple current charges for WZW models.

5. Tensor Products and Extensions

Out of the built in (or user-created) CFT one may construct new ones by tensoring and/or extending the chiral algebra with simple currents. It is also possible to build coset conformal field theories, although the spectrum of such theories can only be computed up to integers.

To use these features one has to switch to Tensor mode by typing
```
> Tensor
```
In this mode the `Algebra` command is not available, but is replaced by two commands
```
> G
```
and
```
> H
```
which accept (almost) all the parameters of `Algebra`. The command `G` adds the algebra enterred as argument to the tensor product, and the command `H` adds it to the "denominator" of a coset CFT.

Note that *Kac* does not know about the embedding $H \subset G$, and will accept most combinations as input. Instead, H is treated as a tensor factor with complex conjugate matrices S and T. Therefore it is in principle legitimate to use factors H that cannot even be embedded in G.

The chiral algebra of a tensor product or coset can be extended using the command
```
> Current (fcur current_1)...(fcur current_n)
```
Here `current_i` must be a simple current of factor i. The number of arguments of the `Current` command must equal the number of factors G plus H in the tensor product or coset. The the chiral algebra is automatically closed under fusion of these currents, and multiple occurrences are ignored. One may also use simple currents to impose field identification in coset CFT's.

```
> Reset Currents
```

Reset currents.

```
> Reset Tensor
```

Reset tensor mode.

Before any modular properties can be used, the spectrum must be computed. This is in contrast to single algebra mode, where spectrum computation occurs automatically. To compute the spectrum without displaying it, type

```
> Compute
```

To display the spectrum type

```
> Display
```

```
> Browse Chiral_algebra
```

Display chiral algebra.

```
> Browse CFT
```

Display several CFT properties.

```
> Expand [field lbl]
```

```
> Browse Decomposition [field i](col_1)...(col_n)
```

Display decomposition of representation "i".

```
> Browse Groundstate [field i](col_1)...(col_n)
```

Display decomposition of ground state of representation "i".

```
> Identify (fprim lbl_1)...(fprim lbl_n)
```

Find orbit in which primary combination occurs (if any).

5.1. Graded tensor products

By a graded tensor product we mean a tensor product with implicit extension of the chiral algebra by certain currents of order 2 and without fixed points. The canonical example is a tensor product of $N = 2$ superconformal field theories. To produce a new such theory one may tensor them as ordinary CFT's, and the extend the chiral algebra with all combinations of the supercurrents in each factor. Although one may do this using the `Current` command, it is much faster to use graded tensor products.

To use this one must first of all switch to the right mode using

```
> Set Graded (switch ON/OFF)
```

Furthermore one of the simple currents in each factor of interest must be selected as a "grade current". This is done by typing

```
> Grade [current J]
```

in single algebra mode. Here J must be a simple current of order 2 without fixed points. For minimal $N = 2$ CFT's the supercurrent is automatically designated as a grade current. When `Set Graded` is ON, the tensor product is automatically extended by all integer spin

combinations of all grade currents. This extension is on top of any other simple current that is selected using the `Current` command. It is allowed to have ungraded factors in the tensor product; they are simply tensored normally with the rest.

5.2. Creating CFT's from tensor products

A (simple current extended) tensor product or coset CFT may be stored by means of the command

```
> Create [CFT_name]
```

This creates a new CFT, which may be loaded by means of the command

```
> Algebra Cft [CFT_name]
```

It may also be used in tensor products using the `G` command, or even in coset-like construbctions using the `H` command, although there are not many meaningful applications of the latter option. In other words, CFT's made using `Create` can be used on equal footing with the built-in algebras.

After storing all the data, the program returns to single algebra mode, with the newly created CFT selected.

Note that when a CFT is created all entries of its modular matrices S are computed and stored, as well as all entries of the fixed point resolution matrices S^J for all simple currents J that have fixed points. Therefore the create command should be used with care.

To view what is happening during creation of a CFT one may switch on the display of additional information:

```
> Set Create Show_info (switch ON/OFF)
```

One may directly access the fixed point resolution matrices using

```
> Algebra FCFT [CFT_name][pos_int current]
```

Here "`CFT_name`" must be one of the character strings shown by the `Browse Algebras` command, and current must be a simple current that has fixed points. One may also select the "fixed point CFT" when the CFT itself is selected in single algebra mode: type

```
> Get FCFT [pos_int current]
```

The difference with the `Algebra FCFT` command is thus that the latter can be used at any moment in single algebra mode.

Note that a fixed point CFT is in general not a CFT, except for certain simple currents of certain WZW-models.

6. Spectra

To display the spectrum of a CFT in single algebra or tensor mode, type

```
> Display (field min)(field max)
```

Without parameters, this display all primaries; with one parameter it displays only that primary field, and with two parameters it displays a range of fields.

The properties of each primary fields are displayed in a table with several columns. To list the available columns use the `Table Info` command, in one of the two forms

```
> Table Info Spectrum
> Table Info Tensorspectrum
```

This gives the available columns in single algebra and coset mode respectively. Not all columns are available in all situations. If a column does not make sense it is usually omitted automatically and without warning.

Columns may be enabled or disabled; this is shown in the last column of the `Tables Info` command. Columns may be suppressed or restored using the generic `Tables` subcommands (see the section "Tables" in the chapter on syntax) but in this case a shortcut is available in the form of the command `List` or `Unlist`.

```
> List (column_1)...(column_n)
```

Restore printing of all columns of spectrum tables with name that starts with "column_i"

```
> Unlist (column_1)...(column_n)
```

Suppress printing of all columns of spectrum tables with name that starts with "column_i".

The content of the various columns is as follows:

— Label: Integer used to refer to a primary, for example in fusion computations.

— Ground state: Indicates ground state representation; these are extended Dynkin labels for affine algebras. For other algebras, see the section on built in algebras

— Tensor label: The label the representation had in tensor mode, before using the `Create` command.

— Weights: Conformal weights in real form.

— Rational weights: Conformal weight in rational form.

— Dimensions: Ground state dimensions.

— Conjugate: Charge conjugate.

— Bulk invariant: Bulk Invariant. This gives, for a primary i, the list of primaries j with $Z_{ij} \neq 0$.

— Contents: Ground state representation content in terms of the first horizontal Lie algebra factor. Available only in superstring mode.

— Charges: Charges w.r.t basis simple currents, as shown by the `Browse simple_currents` command.

— Grade sector: Indicates if field is NS or R with respect to the grade current.

— Grade multiplet: Indicates grade multiplet partner, obtained by the action of the grade current.

— Orbit base: First field on the simple curent orbit of a field.

— Orbit position: Current that generates the field from its orbit base.

— Orbit label: Label of the orbit to which the field belongs.

— Stabilizer: Set of generators (not necessarily minimal) of the stabilizer of a field.

— Q_dims: Quantum dimensions.

— Casimirs: Ground state Casimir eigenvalues (available only for untwisted Kac-Moody algebras).

— Modular anomaly: Value of h-c/24.

— Combination: Combination of G and H fields that forms the ground state of a tensor or coset representation.

— FP label: Fixed point labels (elements of the stabilizer that distinguish fixed points).

— FP mult: Fixed point multiplicity due to untwisted stabilizer.

The latter three are only available in tensor mode, whereas many other items are only available in single algebra mode.

In addition to these built-in columns, the user made add custom-made columns by means of the command

```
> New Column [name][expression]
```

Here "name" is the name of the column, used also as a header. It should not clash with existing column names. The output of "expression" is shown in the extra column. Normally one should use an expression that returns a single line of output, and that depends on a primary field label. The latter is indicated by a '#'. For example

```
> New Column triple_fusion fusion(#,#,#)
```

adds a column named "triple_fusion" containing the fusion coefficients $N_{ii}{}^{i}$. Note that the last argument is an *expression*, and not a command! Any occurence of '#' is replaced by the row number.

The display command has two keywords:

```
> Display Massless [name][expression]
```

displays only the conformal fields of weight 0 and 1, which yield the massless fields in a string spectrum.

```
> Display Entry [col][field i]
```

displays only the indicated entry in the spectrum table.

These keywords cannot be abbreviated.

7. Fixed point resolution

```
> Browse Stabilizer [field lbl]
```
Show information on the stabilizer of field "lbl".
```
> Browse Twists
```
Show simple current twists.
```
> Get Twist [field ][current K][current J]
```
Get the twist F(a,K,J).

```
> Browse Eta
```
Show charge conjugation twist and its field dependence.
```
> Browse Etatwists
```
Compare current twists and eta twists.
```
> Get FCFT [pos_int current]
```
Load fixed point cft for current "current".
```
> Get FCFT_shift [current J]
```
Get the shift parameter "m" that determines the phase between the fixed point resolution matrix $S\hat{J}$ and the orbit Lie algebra matrix O^J: $S^J = e^{-i\pi m/4} O^J$.
```
> Get Fp_fusion [current J][field i][field g][field h]
```
Compute fixed point fusion rules This command computes $\sum_n S_{in} O^J_{gn} O^J_{hn} / S_{0n}$ where g and h are fixed points of J and the sum is over all fixed points. The matrix O^J is the matrix S of the orbit lie algebra, which differs from the fixed point resolution matrix S^J by a phase.
```
> Set Trace_resolution (switch ON/OFF)
```
Switch display of FCFT contributions to resolved S If this switch is ON, the contributions to S_ab are shown for each current J that fixes a and b.

8. Modular properties

```
> S (field lbl1)(field lbl2)
```
Compute matrix elements of S.
```
> Fusion (field i)(field j)(field k)
```
Compute fusion rules.
```
> Set Fusion Sumcheck (switch ON/OFF)
```
Switch dimension sum rule check in fusion.
```
> Compute S_sum (field lbl1)...(field lbln)
```
Compute sum of indicated rows of S.
```
> Get Conjugate [field a]
```
Get the conjugate of field "a".
```
> Get Weight [field a]
```
Get the weight of field "a" in rational form.
```
> Pfusion (field i)(field j)(field k)
```

Compute P-fusion rules.

```
> Pmatrix (field lbl1)(field lbl2)
```

Compute matrix elements of P.

```
> Check P-gauss [field i][field j]
```

Check gauss-sum P-matrix formula for labels "i" and "j".

```
> Get Schur (field a)
```

Compute Frobenius-Schur indicator of field "a". Without arguments: all fields.

```
> Get G_schur [field p][field q]
```

Compute generalized Frobenius-Schur indicator of fields "p" and "q".

```
> Set High_precision (switch ON/OFF)
```

Switch high or low precision in modular data display.

```
> Set S_matrix Maxsize [non_neg_int value]
```

Set maximal size for S_matrix storage.

```
> Set S_store None
```

Direct computation of S.

```
> Set S_store Orbit
```

Keep orbit/orbit matrix elements in memory.

```
> Set S_store Save
```

Write orbit/orbit matrix elements to disk.

```
> Compute S_analytic (factor)(lbl_1)...(lbl_2r)
```

Compute the Kac-Peterson formula as an analytic function on weight space. "factor" is a real factor multiplying the exponents in the Weyl sum. (factor=1 gives the KP-formula). "lbl_i" are analytically continued Dynkin labels, 2r in total, where "r" is the rank; "labl_i" may be any expression that evaluates to a real number.

9. Modular invariant partition functions

One may select a modular invariant partition that defines a "bulk" CFT using the `Select` command. This bulk CFT affects the computation of superstring spectra and the computation of boundary coefficients and crosscap coefficients. In the latter case (boundaries and crosscaps) any invariant is implicitly conjugated. This implies that if no bulk invariant is selected, the diagonal invariant will be used for "closed" string spectrum computations, and the charge conjugation invariant for "open" string computations.

The following possibilities are available

```
> Select Simple_current_invariant (current J_1)(current J_2)...(current J_n)
> Select Invariant [invariant]
> Select Conjugation_invariant
> Select Diagonal_invariant
```

The first selects one of the simple current invariants that belong to the subgroup generated by the currents $J, J_2, \ldots, J_n$. This is the subgroup that is obtained by closing the set of currents under fusion. The set of currents may be overcomplete; superfluous currents are removed. The set of invariants grows rapidly with the number of discrete factors in this group. The user is prompted after each invariant to select it or continue the search. If no arguments are given *all* subgroups of the effective center are checked.

The second version of the select command allows the selection of exceptional invariants (in fact any modular invariant). Here the input must be a character string of one of the two allowed forms. The first type of expression is a sum of terms, each of the form

$$(\text{mult}_1) * (\text{orbit}_1) * (\text{orbit}_2) \quad \text{or} \quad (\text{mult}_2) * (\text{orbit}_3)^2$$

where "mult" are integer multiplicites and the "orbits" have the form

$$(n_a a + n_b b + \ldots n_k k)$$

where $n_a \ldots n_k$ are integer multiplicies and $a \ldots k$ labels of primary fields. This gives the partition function explicitly. The first character of an invariant should be '`(`', in other words the multiplicity of the first orbits must be 1. The second allowed type of expression is

$$\langle a, b\rangle\langle c, d\rangle \ldots$$

where $a, b, c, d \ldots$ are primary field labels. This form represents an automorphism with interchange of a with b, c with d etc. Also higher order cyclic permutations are allowed, e.g. $\langle a, b, c\rangle$.

The third version of the `Select` command selects the charge conjugation invariant, and the fourth results in the trivial, diagonal invariant, which is the default if nothing is selected.

The selection process of simple current invariants can be made more efficient in one of the following ways

```
> Set Selection Symmetric (switch ON/OFF)
```

restricts the selection process to symmetric invariants.

```
> Set Selection Automorphism (pair_1)(pair_2)...(pair_n)
```

restricts the selection process to automorphism invariants. The optional arguments are of the form `i:j` and require a pairing of primary field labels for the selected invariant. This criterium is switched off by typing

```
> Set Selection Automorphism Off
```

Furthermore the pairings are switched off automatically whenever the CFT is changed, because then they will usually not be sensible anymore. The command

```
> Set Selection Extension
```

requires the invariant to have a non-trivial chiral algebra.

```
> Set Selection Left [current J_1](current J_2)...(current J_n)
> Set Selection Right [current J_1](current J_2)...(current J_n)
```

require the indicated currents to be present in the left (right) chiral algebra. The latter two criteria can be turned off by typing the commands without arguments; they are automatically turned off if the CFT is changed. Some of the selection criteria are evidently

incompatible with each other.
If the order in which the invariants are generated is known, the n^{th} invariant can be selected by typing

```
> Set Autoselect [pos_int choice]
```

(where n ="choice"). When this setting is chosen, the user is not prompted, but the n^{th} invariant is selected automatically. Here "choice"=1 corresponds to the first invariant. This automatic selection feature can be switched off by typing

```
> Set Autoselect Off
```

Each invariant that is generated is displayed. Since this can produce a lot of output one can limit the number of orbits that is shown by means of the command

```
> Set Selection Max_display [non_neg_int value]
```

Then only the orbits of the first "`Max_display`" fields are shown.

The complete set of selection criteria can be displayed using

```
> Show Selection Criteria
```

The `Show Selection` command has three other keywords to display the values of the `Max_display`, `Symmetric` and `Extension` parameters. To switch off all selection criteria type

```
> Reset Selection
```

Two further commands are available to simply list the simple current invariants belonging to a subgroup, or to check a candidate modular invariant, without selecting them as bulk invariants

```
> Compute Simple_current_invariant (current J_)(current J_2)...(current J_n)
> Check Invariant [invariant]
```

The argument syntax is as explained above for the `Select` command. The selection criteria described above apply also to the `Compute` command.

Examples

```
> Algebra D 4 1
Number of primaries:  4
> Compute Simple 1 2

(0)^2+(1)*(3)+(2)*(1)+(3)*(2)

(0)^2+(1)*(2)+(2)*(3)+(3)*(1)

Total number:  2
> Select Invariant <1,2,3>
> Select Invariant <1,3,2>
> Algebra D 4 2
Number of primaries:  11
> Compute Simple 1 2

(0+1+2+3)^2 + 4*(10)^2
```

```
(0)^2+(1)^2+(2)^2+(3)^2+(4)^2+(5)^2+(6)^2+(7)^2
+(8)^2+(9)^2+(10)^2

Total number:  2
> Select Invariant (0+1+2+3)^2+4*(10)^2
```

9.1. Manipulating modular invariants

A selected bulk invariant can be stored by typing

```
> Store (name)
```

If a name is given, it is stored under that name; if no argument is present, a name is generated automatically. This name is `Invariant_n`, where n is an integer, starting with 0. Note that an invariant will not be stored if an identical one is already present. In this way one may collect distinct modular invariants.

Stored invariants can be selected as bulk invariants by typing

```
> Select Invariant [name]
```

Here "name" must be identical to the name specified when the invariant was stored. If the name was automatically generated as `Invariant_n`, then `Select Invariant n` suffices. Note that names should not start with the characters '`(`' or '`<`' or with digits, to avoid ambiguities in the interpretation of the command.

Stored invariants can be multiplied using

```
> Multiply [name_1][name_2](name_3)
```

This multiplies the invariants specified by the first two arguments as matrices, storing the result as "`name_3`", or as an automatically generated name if this argument is absent. The first two arguments can be integers to indicate `Invariant_n`, as above for the select command.

Any stored invariant can be conjugated by means of the

```
> Conjugate (name_1)(name_2)
```

command. The effect is that all fields in the anti-holomorphic orbits are replaced by their conjugates. The result of a conjugation is always stored. If both arguments are absent, the current bulk invariant is conjugated and stored as `Invariant_n`, with an automatic value for "n"; if the first argument is present, the named invariant is conjugated; as above, the name may be an integer. If both arguments are present the conjugate of invariant "`name_1`" is stored as "`name_2`".

Finally one may transpose a modular matrix by typing

```
> Transpose (name_1)(name_2)
```

The syntax is as for the `Conjugate` command.

One may store all modular invariants automatically by means of

```
> Set Autostore (switch ON/OFF)
```

When this is switched on (the default setting) all invariants computed by the "`compute simple_current_invariants`" are automatically stored, as well as all invariants that are selected using the `Select` command. The identity is stored as "`Invariant_0`". Invariants checked using the `Check` command, and all invariants generated during a selection with-

out being actually selected are however *not* stored. Conjugates, transposes and results of multiplications are in any case store automatically.

Note that the selection criteria specified using `Set selection` apply also to the selection of invariants from storage. If, for example, automorphisms are required, then an extension modular invariant cannot be loaded as the current bulk invariant. However, explicitly specified exceptional invariants as well as the conjugation invariant can still be selected even if they violate the selection criteria, because this violation is obvious for the user. to simple current invariants, not to storage of invariants. Note also that the "Autostore" feature can easily overflow available memory in the case of CFT's with many primaries and many simple current invariants. The value of Autostore can be set in the Defaults file.

```
> Reset Invariants
```

Reset modular invariant storage.

9.2. Modular invariant loops

It is possible to run through all simple current invariants (or a selected subset) of a given theory and perform operations on them. To do this use

```
> Set Loop_command [commandline]
```

with any valid command line. This commandline is executed for any invariant that is found using the `Compute Simple_current_invariant` command, given the chosen selection criteria. If execution results in an error, the loop is terminated. Note that if a loop command is set, each invariant that is found is selected as a bulk invariant, so that in this case the bulk invariant is changed. Running `Compute Simple_current_invariant` without a loop command simply displays all invariants without selecting them, so that the original bulk invariant is maintained. The loop command can be switched off by typing

```
> Set Loop_command Off
```

Note that the command is performed on *all* invariants. If `Autostore` is on, some invariants may turn out to be the same as invariants already in storage. These may either have been generated in previous runs, or in the present one. Since it cannot be decided if they did or did not occur in the same run, this information is not used.

In principle all simple current invariants generated by `Compute Simple_current_invariant` are distinct, but degeneracies may occur in certain cases with atypical simplicity. To apply a command only to the distinct invariants, proceed as follows. First reset the invariant storage:

```
> Reset Invariants
```

Then generate all invariants of interest using `Compute Simple_current_invariant`, with selection criteria and parameters as desired. In this case identical invariants are stored only once. Finally, one can make an ordinary "`for`" loop trough all stored invariants by using the function `total(invariants)`:

```
> For i=0 i<tot(inv) i=i+1 [commandline]
```

where "commandline" begins with `Select invariant i;` , after which any other commands may follow.

9.3. Inspecting invariants

The names of all store invariants can be shown with the command

```
> Browse Invariants
```

Each invariants can be inspected separately by typing

```
> Browse Bulk_invariant (name)
```

Here "`name`" is any stored invariant or and integer for automatically generated names. If the argument is absent the current bulk invariant is shown.

The standard output gives the explicit form of the invariant and the number of Ishibashi labels resulting from it (implicitly conjugated). Further information can be switched on by

```
> Set Invariant Show_info (switch ON/OFF)
```

which shows in particular details about the way the modular invariant is generated from simple currents. If this switch is ON, the same information is also shown during the generation of simple current invariants. The command

```
> Set Invariant Statistics (switch ON/OFF)
```

enables displaying, for each T-eigenspace, the number of diagonally coupled fields, the number coupled *off-diagonally* to their charge conjugate, as well as the number of self-conjugate fields. These numbers appear in a table with columns marked 'D', 'C' and 'S' respectively. This information can sometimes be useful to identify invariants that are related in straightforward ways, e.g. by permutation of factors.

```
> Reset Invariants
```

Reset modular invariant storage.

9.4. Galois symmetry

```
> Search Galois_invariants
```

Search for galois invariants.

```
> Compute Galois_orbits
```

Compute all galois orbits.

```
> Compute Scale_orbits
```

Compute scale orbits.

9.5. Finding exceptional extensions

```
> Browse Integral_spin
```

Display all integral spin fields.

```
> Check Algebra (int mult1)...(int mult2)
```

Check if a potential chiral algebra has non-negative projections on all fields. The parameters are the multiplicities of the integer spin fields in the order given by "Browse integral_spin".

```
> Search Extension
```

Find potential chiral algebra extensions.

10. Boundaries and Crosscaps

The boundary and crosscap coefficients depend on the choice of bulk modular invariant and Klein bottle projection. If no selection is made, the simplest case is assumed: the so-called "Cardy-case", corresponding to the charge conjugation invariant in the bulk, combined with a Frobenius-Schur Klein bottle. Note that in boundary coefficient computations all selected bulk invariants are implicitly multiplied with the charge conjugation invariant.

An algorithm is built in for the computation of all boundary and crosscap coefficients for all simple current modifications of the Cardy case. This algorithm requires the fixed point resolution matrices of simple currents and hence it cannot work, in general, in tensor mode. Use "`create`" to make a new CFT that can be used in single algebra mode.

There is no simple algorithm for exceptional invariants, but there is a search procedure built in which can be tried if the invariant is a pure automorphism (this includes the charge conjugation invariant, implicitly multiplied with another charge conjugation to yield the diagonal invariant).

Once the MIPF has been selected, one may compute the Ishibashi labels by typing

```
Browse Ishibashi (Ishibashi k)
```

This works for all modular invariants. Without parameter is lists all Ishibashi labels, with parameter just the k^{th} one. The Ishibashi labels are denoted (m, i) where m refers to a bulk field, and i is a resolution label. This just enumerates the degenerate Ishibashi labels in the exceptional case, and corresponds to currents in the stabilizer of m in the simple current case.

10.1. Selecting Klein bottles

The data needed to compute crosscaps and boundary coefficients are the bulk MIPF and a set of Klein bottle coefficients. The bulk selection was explained in the previous section. The selection of Klein bottles can be done in one of the following two ways

— If the bulk invariant is a simple current invariant: The canonical Klein bottle coefficient is given by the Frobenius-Schur indicator. Modifications may be made as follows

```
Set Kleinbottle [current J]
```

This defines a Klein bottle modification related to the simple current: the Klein bottle coefficient of field i is multiplied with $e^{2\pi i Q_J(i)}$. A second class of modifications that is always allowed consists of the crosscap sign choices. The simple current formula for boundaries and crosscaps depends on a sign choice for each factor $\mathbf{Z}_k$, k even, in the simple current group that is involved in the construction of the modular invariant. These signs can be set using

```
Set Crosscap_signs (sign s_1)...(sign s_n)
```

This command has a parameter for every *even* factor in the simple current group. The Klein bottle is reset to zero, and all crosscap signs are reset to 1 each time a new modular invariant is selected.

— If the bulk invariant is an exceptional invariant: The canonical Klein bottle coefficient is chosen to be Z_{ii} for fields i that appear diagonally and 0 for fields that do not. Here

Z_{ij} is the modular invariant. Any other Klein bottle can be chosen as follows

```
Select Kleinbottle (entry_1)...(entry_n)
```

Here `entry_i` must be of the form `label:value` where "label" is a primary field label and "value" is a valid Klein bottle coefficient (smaller in absolute value than Z_{ii} and of equal parity). Only the labels that are given non-canonical values have to be enterred. If no arguments are given, the Klein bottle coefficients are reset to their canonical values. This can also be done using the command

```
Reset Kleinbottle
```

The latter command also resets the Kleinbottle current and the crosscap signs.

If a non-trivial Klein bottle current is chosen in combination with an exceptional invariant, it is ignored. A non-standard Klein bottle cannot be selected in combination with a simple current invariant (apart from the Klein bottle current and crosscap signs mentioned above). However, it is possible to convert a simple current invariant to an exceptional one using

```
> Make Bulk Exceptional
```

After this is done the invariant is no longer treated as a simple current invariant; one may select Klein bottles but not set Klein bottle currents or crosscap signs.

All choices affecting crosscaps and Klein bottles are reset to their canonical values when a bulk invariant is selected, conjugated, or turned into an exceptional invariant.

10.2. Searching exceptional Klein bottles

One may perform a systematic loop search for valid Klein bottles satisfying the positivity and integrality condition in the closed sector. The requirement they must satisfy is that the corresponding crosscap coefficients are non-zero only on Ishibashi states.

In addition two other constraints can be imposed: the Klein bottle constraint ($K_iK_jK_\ell \geq 0$ if i, j and ℓ are coupled) and the requirement that

$$\sum_\ell \frac{S_{i\ell}}{S_{0\ell}} K_\ell$$

be non-negative integers. The latter follows from the so-called orientation sensitive trace formula. It should be mentioned here that the Klein bottle constraint is violated in certain cases, and that the trace formula has not been proved.

The following commands are available

```
> Search Klein_bottle
```

Search for Klein bottle signs consistent with Ishibashi states.

```
> Search Klein_bottle Kbc
```

Search for Klein bottle signs consistent with Ishibashi states and the Klein bottle constraint.

```
> Search Klein_bottle Trace
```

Search for Klein bottle signs consistent with Ishibashi states and satisfying the trace formula.

```
> Search Klein_bottle All
```

Search for Klein bottle signs consistent with Ishibashi states and satisfying the trace formula and the Klein bottle constraint.

10.3. Simple current bulk invariants

After the bulk invariant, Klein bottle current and crosscap signs are chosen the various coefficients and the boundary or Ishibashi labels can be computed.

To see the boundary labels, type

```
Browse Boundary (bound a)
```

The boundary labels are denoted $[m, i]$ where m refers to a bulk field (in fact a simple current orbit representative), and i is a resolution label.

The commands to display boundaries and crosscap coefficients are

```
Get Reflection (Ishibashi i)(bound a)
Get Crosscap (Ishibashi i)
```

What is listed is the coefficient multiplied with a factor $\sqrt{S_{0i}}$. This implies that in the Cardy case the result that is displayed is S_{ia} and P_{0i}.

The signs set by `Set Crosscap_signs` (see previous sub-section) are the ones that are allowed in the generic case. In special cases (if certain potentially allowed fixed points do in fact not occur) more sign freedom may exist. This may be explored by means of the command

```
Flip Crosscap_signs (current s_1)...(current s_n)
```

This flips the signs belonging to a certain current, not a cyclic factor of the simple current group. This will only have an effect if the current is involved in the construction of the invariant. These signs are multiplied with the generic signs defined with the "`Set Crosscap_signs`" command. Using the "`Flip Crosscap_signs`" option may result in inconsistencies. On the other hand sign choices made with "`Set Crosscap_signs`" are always allowed.

Resetting

The command

```
> Reset Kleinbottle
```

may be used to reset all the signs affecting Klein bottles (and crosscaps). In particular this undoes the commands `Set Kleinbottle`, `Set Crosscap_signs`, `Flip Crosscap_signs`, and, in the case of exceptional invariants, `Select Kleinbottle`. The selection of any other MIPF resets all these signs as well.

The choice of bulk invariant and Klein bottle can be reset to the canonical Cardy values by typing

```
> Reset Bulk
```

(This has therefore the same effect as `Select Diagonal`).

Inspecting the settings

To see the current setting of these sign choices use

```
> Browse Signs
```

The output shows five columns, containing respectively the current label, its weight, the sign used in the simple current formula for the crosscap, the corresponding phase shift used in the formula for the boundary, and the phase shift needed to go from the "Orbit Lie algebra matrix" to the "Fixed point resolution matrix". In formulas:

— Crosscap signs $\rho(J)$: They enter in the formula for the crosscap as $\Gamma_i \propto \sum_J \rho(J) P_{Ji}$.

— Boundary phase shifts $\rho(J)$: They enter in the formula for the boundaries as $B_{i,a} \propto \sum_J \alpha(J) S^J_{ia}$. The phases $\alpha(J)$ are in general eight roots of unity, and are written as $e^{-i\pi x/4}$. The value of x appears in column 4.

— FCFT shifts: They are defined as $S^J = e^{-i\pi m/4} O^J$, where S^J is the fixed point resolution matrix and $O(J)$ the orbit Lie algebra matrix. The value of m is listed in column 5. In those cases where O^J is not defined (CFT's that are not WZW-models) it can be defined in terms of S^J, up to a phase ambiguity. Internally this ambiguity is fixed by making an arbitrary choice among the valid possibilities. The other valid possibilities are obtained by choosing different crosscap signs, which affects $\alpha(J)$.

Generating all valid choices

It is possible to combine Klein bottle and crosscap sign choices, provided that some rules are satisfied. Some Klein bottle choice are inconsistent with certain modular invariants, and some others are redundant. All valid distinct choices can be generated using the command

```
> Loop Crosscap (commandline)
```

If a commandline is given it is executed for each of the valid choices.

10.4. Exceptional bulk invariants

A search procedure is built in to look for boundaries and/or crosscaps of exceptional automorphism invariants. After choosing an algebra in single algebra or tensor mode, one selects the in invariant using "`select invariant`" or "`select conjugate`", as explained earlier. The conjugation invariant is always treated as an exceptional invariant.[⋆] The conjugate of any invariant (obtained using the `conjugate` command) is also treated as an exceptional invariant (even if the `conjugate` command is typed in twice). If you wish to treat the diagonal or simple current invariants as exceptional ones (for example to test this algorithm), type

```
> Make Bulk Exceptional
```

In some cases this conversion is made automatically, and a warning message is given. The conversion to an exceptional invariant (automatic or manual) is irreversible, but of course one may select a new invariant.

There are two problems that can now be addressed: the first is to look for boundaries and crosscaps, and the second to look only for boundaries. The former is default. To search only for boundaries, type

```
> Set Orientifold off
```

In the former case the required input data are the modular invariant and a Klein bottle. The latter is by default taken to be $1^{\dagger}$ on fields appearing diagonally and 0 on fields appearing off-diagonally (in the modular invariant times implicit charge conjugation). Using the "`Flip Klein_bottle`" command the entries "1" can be flipped to "−1".

⋆ Reminder: all invariants are implicitly conjugated when constructing boundaries. Therefore if the conjugation invariant is selected, the boundaries that one obtains belong to the *diagonal* combination of characters.

† Note that the default in the exceptional case is *not* the Frobenius-Schur indicator.

The search algorithm is explained in chapter 12. The boundary search is started with the command

```
> Search Boundaries (pos_int Maxloop) (non_neg_int Method)
```

The parameter "Maxloop" specifies the maximal size of each annulus coefficient A^i. The parameter "Method" selects the search method: either a full set of loops $0 \leq A^i \leq$ Maxloop (Method=0) or a search restricted to the annuli satisfying $S_{im}A^i = 0$, if m is not an Ishibashi label. To select the latter option, choose the value 1 for the "Method" parameter. The default values are Maxloop=2, Method=1. With this command all steps in the algorithm are executed successively.

Another way to start the search is to enter annuli manually, using

```
> Try Boundary [limits]
```

Here limits is a string indicating a new annulus, or exact limits between which should be searched for annuli. For example

```
> Try Boundary 1,1,1,1,0:3,1,1
```

specifies that all annuli $A^i = 1$, $0 \leq A^4 \leq 3$ will be tried. If there are N primaries the string "limits" must have as many entries, separated by comma's. This method stops after step 6 of the algorithm, since typically one may want to supply several candidate annuli before combining them into sets. To start step 7 and the subsequent ones type

```
> Assemble
```

The amount of information provided during the process can be controlled using

```
> Set Boundary_search Info_level [non_neg_int level]
```

The default for "level" is 0. For larger values more information is shown. The information that is provided is roughly the following.

0. Overall success or failure information.
1. Klein bottle; basic statistics on boundary combination analysis.
2. Crosscap coefficients; list of boundary combinations (step 9); complete boundary sets;
3. Unsigned boundaries (found during steps 6 and 11); All annuli and moebius amplitudes for complete boundary sets; Maximal (not complete) sets for each combination.
4. All annuli and moebius amplitudes for maximal incomplete boundary sets.
5. Completion statistics per boundary set (step 11).
6. Orthogonal basis vectors, completion equations, basis decomposition of new boundary vectors (step 11).
7. Construction of boundary sets (step 10): signed boundary vectors and the set numbers to which they are added.
8. Construction of boundary sets: reasons for rejection of boundary vectors.

```
> Reset Boundary_search
```

Reset exceptional boundary coefficients to start new search.

```
> Get Class_alg [Ishibashi i][Ishibashi j][Ishibashi k]
```

Compute classifying algebra coefficients.

11. Permutations and outer automorphisms

The modular properties of conformal field theories are invariant under a group of outer automorphisms. This includes charge conjugation, Dynkin diagram automorphisms of WZW models and permutations of identical factors in a tensor product. Modular invariants that are different, but that are related by such an automorphism may yield equivalent results. Whether the results are really equivalent depends on the problem under consideration. For example, two identical CFT's in a tensor product may play a totally different rôle in a problem, in which case the interchange of these factors would not lead to equivalent results. The outer automorphisms will be referred to as "permutations" in the following, since that is the most common case.

If, however, there is a true equivalence one may wish to eliminate superfluous solutions and restrict the computations to one representative of the equivalence class. This can be done for simple current invariants with the command

```
> Set Reject Permutations (switch ON/OFF)
```

When this is ON, the following outer automorphism are take into account

— In single algebra mode, the Dynkin diagram automorphisms of WZW models, except charge conjugation. These automorphisms are therefore spinor conjugation for D_{2n} and triality for D_4.

— In coset mode, the group generated by all permutations of identical factors plus all outer automorphisms of the factors themselves, that leave the chiral algebra – as specified with the `Current` command – invariant.

When a new CFT is created using the `create` command this group is stored as the outer automorphism group of that CFT, and will be taken into account if the CFT is used in further tensor products.

Note that outer automorphisms of a CFT are taken into account only if `Reject Permutations` is ON at the moment that the CFT was created; they cannot be computed afterwards. This holds also for WZW models, which are created by the `Algebra` (or `G` or `H` command).

The automorphisms act on all primaries of the CFT, but what matters is only the restriction of this action to the set of simple currents. This can be displayed using

```
> Browse Permutations
```

These permutations affect the following choices

— The selection of a simple current subgroup of the center.

— The matrix $X(J,K)$ defined on that subgroup, that determines the modular invariant partition function.

— The crosscap sign choices.

— The Klein bottle choice.

When `Reject Permutations` is ON, this affects the commands `Compute Compute Simple_current_invariant` and `Select Simple_current_invariant` (both commands without parameters). First of all only one subgroup out of each equivalence class is chosen. The subgroup of the permutations that respect the simple current subgroup is determined; this

permutation subgroup is then applied to the various choices of $X(J,K)$, which are also restricted to class representatives. Finally the permutation subgroup that respects $X(J,K)$ is determined.

The latter can then be used to restrict the scope of the command `Loop Crosscap`. The remaining permutations are applied to the crosscap sign choices and the Klein bottle current, and again only one representative of each equivalence class is chosen.

To view the restriction process one may type

```
> Set Reject Info (switch ON/OFF)
```

This displays all matrices X, crosscap signs and Klein bottle currents, and the reason they are rejected (no information is given regarding the simple current subgroup selection).

To control which permutations are used and which are not use the `create` command with `Reject Permutations ON` or OFF. For example, one may create two copies of the algebra D_4, one with permutation info and one without, each with a different name. Note that that the setting should be `ON` at the moment the algebra is computed for the first time (with the `Algebra`, `G` or `H` command), since otherwise it is stored without outer automorphism information. To use only a subset of the permutations of N identical factors, first tensor part of the factors, create a new CFT, and tensor tensor it with the rest in a second step.

12. String Theory

12.1. Closed superstrings

Any unitary CFT with an integer central charge c smaller than 26 can be viewed as bosonic string compactification in $26-c$ dimensions. No special commands are provided (or required) to deal with this case.

Certain special CFT's can be mapped to heterotic or type-II strings. This requires one or both chiral sectors respectively to be mapped to an N=1 superconformal field theory. These sectors should satisfy the following requirements for superstrings in D space-time dimensions

1. The chiral algebra must contain the affine algebra $SO(D{+}6)$, level 1.
2. The chiral algebra must contain the affine algebra E_8, level 1.
3. The total central charge of the CFT must be $26 - D$.
4. The fermionic sector must have N=1 world-sheet supersymmetry.
5. The bosonic CFT must be modular invariant.

The program checks points 1,3 and 5. If the central charge is $18 - D$ an implicit E_8 factor is assumed. Replacing the E_8 factor by another $c = 8$ CFT, or violating superconformal invariance leads to nonsensical results, but the user is allowed to experiment with this. The correct structure of a chiral CFT that can be mapped to a fermionic string is thus as follows

$$SO(D{+}6)_1 \times E_{8,1} \times [N{=}1 \text{ CFT}]$$

The $N = 1$ CFT is most easily built by tensoring $N = 1$ (or $N = 2$) CFT's, using a graded

tensor product (see the chapter on tensor mode) to maintain worldsheet supersymmetry. The tensor product with $SO(D+6)_1$ must also be graded. The grade currents must be the vector of $SO(D+6)_1$ and the supercurrents of the superconformal CFT's. Instead of $SO(D+6)$ one may use other algebras that contain $SO(D+6)$, level 1. They can be $SO(N)$, $N > D+6$ or, depending on D, the exceptional algebras F_4, E_6, E_7 and E_8, all at level 1.

Superstring mode

In order for superstring spectrum analysis to take place, one must switch to superstring mode:

```
> Set Super (switch ON/OFF)
```

Without parameter this command has the same effect as with parameter "ON".

This setting affects spectrum computations in the following way. If the first factor in the tensor product is either $SO(M)$, F_4, E_6, E_7 or E_8 (level 1), then the ground states of all representations are decomposed in terms of representations of these algebras. This decomposition is computed simultaneously with the computation of ground state dimensions. It can only be done if the latter are computable: there should therefore be no factors "H" in the tensor product.

Computations of dimensions and decompositions are only done when needed. This is the case if the columns "Dimensions" and/or "Contents" in the tensor spectrum table are enabled. They can be enabled by typing

```
> List dimensions
> List contents
```

This results in extra columns seen with the "Display" command. These columns are shown only if they can be computed; if not, they can be omitted without warning.

Extended algebras

If the tensor product is extended by simple currents of spin 1, it may happen that the first tensor factor gets enlarged, resulting in a larger horizonal Lie algebra. The possible embeddings are chains built from[⋆]

$$\begin{aligned} SO(M) &\subset SO(M+1) \\ SO(9) &\subset F_4 \\ SO(10) &\subset E_6 \\ SO(12) &\subset E_7 \\ SO(16) &\subset E_8 \end{aligned}$$

These extensions are automatically taken into account, and the ground states of all representations are decomposed w.r.t. these extended algebras. Extensions to exceptional algebras imply space-time supersymmetry.

⋆ The second case cannot be obtained using simple currents.

String spectra

To see the massless spectrum that is obtained if the CFT is interpreted as a heterotic string type

```
> Browse heterotic
```

The space-time dimension is the one inferred from the first tensor factor, with the maximal value for D that is allowed. The simple current extensions discussed above are *not* taken into account in computing the dimension.

If a different dimension is preferred, the automatic choice can be overruled by

```
> Set Dimension [dim value]
```

where "value" must be at least 3 and at most 10.

This command "browse heterotic" only works if superstring mode is on. The decompositions of all representations are computed, if this was not done previously. Only the representations with conformal weight 1 contribute to the massless spectrum.

The selected bulk invariant is taken into account in the spectrum computation. If no invariant was selected, the diagonal invariant is used. Chiral algebra extensions in this bulk invariant are (at present) not taken into account in the sense that if the bulk invariant extends H to G, all results are presented in terms of representation of H.

```
> Browse Heterotic Decomposition
```

Same as "browse heterotic" but in addition to the totals, the decomposition of the spectrum in terms of terms in the partition function is shown.

```
> Browse Heterotic Flipped
```

Same as "browse heterotic" but interprets the partition function with left and right chiral sector flipped. Normally the left sector is used as the gauge sector. The left sector is the one appearing as the left factor in asymmetric terms in the modular invariant, as shown by the command `Browse Bulk_invariant`. An additional keyword `Decomposition` is allowed after `Flipped`.

```
> Browse Type-II
```

Display type-II string spectrum (using the bosonic string map)

```
> Browse Type-II Decomposition
```

Same as "browse type-II" but in addition to the totals, the decomposition of the spectrum in terms of terms in the partition function is shown.

```
> Browse Type-II Flipped
```

Same as "browse heterotic" but with a relative space-time chirality flip between left and chiral sectors (maps type-IIA to type IIB). An additional keyword `Decomposition` is allowed after `Flipped`.

Creating superstrings

If the command

```
> Create [CFT_name]
```

is used with superstring mode on, then the decompositions of all representations are kept in storage with the newly created algebra. They are shown in the "Contents" column of the spectrum table, if this column is enabled. After creation of an algebra, it is not possible anymore to compute the decomposition of representations, because the relevant information is lost. Hence `Set super` has no effect in this case.

Using created superstrings

Superstring CFT's created by this procedure can be re-used as factors of a tensor product or be extended by additional currents. This will happen if they are the first factor of the tensor product, and if superstring mode is on.

Examples

The following example gives the CFT underlying the "quintic" Calabi-Yau manifold with $h_{21} = 101$ and $h_{11} = 1$.

```
> Algebra Minimal 2 3
```

This creates the basic building block, the $k = 3$, $N = 2$ minimal super-CFT.

```
> Algebra D 5 1
> Grade 2
```

This creates a copy of $SO(10)$ level 1 with the vector as grade current.

```
> Set Graded
> Tensor
```

Graded tensor product are to be used.

```
> G d 5 1
> G min 2 3
> G min 2 3
> G min 2 3
> G min 2 3
> G min 2 3
> Current 1 1 1 1 1 1
```

This is the chiral part gravitino vertex operator. Its presence ensures space-time supersymmetry.

```
> Set Super
```

Switch to superstring mode.

```
> Create quintic
```

This creates a CFT named "quintic". It has 4000 primaries, and its heterotically interpreted massless spectrum can be shown by typing

```
> Browse heterotic
```

The result is

```
--------------- Spectrum for diagonal invariant ---------------
Space-time dimension 4
Gauge group E6 x E8 x [Unknown (Dim=4)]
There are 4 additional spin-1 currents,
which form an unidentified gauge group factor.
Massless spectrum (Representations of first factor)
```

```
N=1 chiral multiplets:  330 x [1] + 101 x [27] + [27*]
```

This shows that the gauge group is $E_6 \times E_8$, and that there are four additional vector bosons belonging to an unidentified gauge group (which does not extend $E_6 \times E_8$). There are 101 chiral families, 1 anti-chiral family, and 330 singlets; the latter three are part of $N = 1$ chiral multiplets. Having created this CFT we can go back to tensor mode:

```
> Tensor
```

and load the quintic CFT, and type "`Compute`" to indicate that there are no further factors an currents.

```
> G Cft quintic
> Compute
```

Now we may select a bulk invariant, for example

```
> Select Simple 5
> Browse heterotic
...
N=1 chiral multiplets:  210 x [1] + 21 x [27] + [27*]
```

Of course this could have been done just as well in single algebra mode. However, since simple current 5 has integer spin, we may also use it to extend the chiral algebra:

```
> Current 5
```

The previously selected bulk invariant is removed. The new CFT has 160 primaries

```
> Compute
Number of primaries:  160
```

and its heterotic spectrum is of course exactly the one we found above, when this invariant was used as a bulk invariant. The extended theory has 5 simple currents of fractional spin, that can be used to build a simple current automorphism invariant.

```
> Select Simple 1
> Browse heterotic
...
N=1 chiral multiplets:  210 x [1] + [27] + 21 x [27*]
```

This corresponds to the mirror manifold.

12.2. Supermultiplets

```
> Browse Susy (dim D)(L_susy)(R_susy)(m1)(m2)(int sym)
```

Show spin content of susy representation.

```
> Browse Lorentz [dim D]
```

Explain notation for Lorentz representations appearing in D-dimensional super multiplets.

12.3. Open Strings

Open string partition functions are specified by Klein bottle, Annulus and Moebius coefficients. These can be computed if the boundaries and crosscaps for the underlying CFT's are known. If this is the case, the partition functions can be displayed by means of the following commands

```
> Annulus (field i)(bound a)(bound b)
> Moebius (field i)(bound a)
> Klein (field i)
```

Here i is a character (primary field) label and a, b any valid boundary label. All parameters are optional. If they are absent all values are shown.

Physical open string partition functions are obtained by multiplying these quantities with Chan-Paton multiplicities, which are obtained by solving the tadpole conditions. Solving these conditions is the subject of the next chapter.

13. Tadpole cancellation

Given a CFT with a space-time interpretation and a bulk invariant for which the reflection and crosscap coefficients are known one may try to cancel the massless tadpoles. The equations are

$$\sum_a N_a B_{ia} = \epsilon(-1)^{h_i} 2^{D/2} \Gamma_i$$

where h_i is the conformal weight of the representation. Only representations that yield massless states contribute. This means that either $h_i = 0$ (the vacuum) or $h_i = 1$. Furthermore, in the case of superstrings, we restrict i to representations that contain physical states (characters that contain among their massless states a vector representation or a definite spinor representation of the covariant lattice group). The space-time dimension, D, must be integer. If it is not integer, the actual non-integer value is used for bosonic strings, and one may try to solve the equations anyway. For superstrings non-integer dimensions make even less sense, and superstring mode is automatically turned off if an inconsistency is encountered.

It is assumed that there is one special Ishibashi label, 0, which has the property that all B_{0a} are real and non-negative. The parameter ϵ is an overall crosscap sign, that is chosen in such a way that the right hand side is positive for $i = 0$. Positivity of generic annuli requires all N_a to have the same sign, and hence they are non-negative with these conventions. If $\Gamma_0 = 0$ all Chan-Paton multiplicities are zero. The tadpole equations may be displayed by typing

```
> Browse Tadpole Equations (non_neg_int nr.)
```

Each equation is shown as a vector, the last entry being the negative of the right hand side. For example if an equation is displayed as

```
1.000 1.000 -512.000
```

Then there are two CP-labels that satisfy $N_1 + N_2 = 512$. First all non-trivial equations are shown in their original form, and then all independent equations are shown. If a parameter is present only the chosen equation is shown.

The special tadpole equation mentioned above is in fact the dilaton tadpole. The tadpole equations can only be solved automatically if this condition is imposed, since otherwise the CP-labels are in general not bounded.

Once ϵ is fixed, the Chan-Paton gauge groups are also fixed. They are given by the value of the Moebius amplitude M^0_a times ϵ. This can have the values $+1$ (gauge group $SO(N_a)$), -1 (gauge group $Sp(N_a)$, valid only if N_a is even) or 0. In the latter case the gauge group is $U(N_a)$, and there are two boundaries contributing to it. To see the relation between boundaries and CP-gauge groups type

```
> Browse CP
```

In the tadpole equations one simplication is made automatically: CP-labels at mutually complex boundaries are set equal to each other. This reduces the loop search significantly. The equations shown by `> Browse Tadpole Equations` already include these reductions, and hence the variables are not directly equal to the CP labels.

The command to start solving the tadpole equations is

```
> Solve Tadpole
```

The procedure consists of three steps:

— Finding kernel vectors

— Reducing bounds

— Loop search

Obviously one can find all solutions by means of a finite loop search, since all N_a are bounded from above by the first equation. In practice, this is however impossible. In many cases the equations have a non-trivial kernel containing integer vectors. Suppose such a vector is $(-1, 1, 0, \ldots, 0)$. If there is a solution with $N_0 > 0$ we can always reduce it to a solution with $N_0 = 0$ by adding this vector, and thereby increasing N_1. Hence we may restrict the loop search to $N_0 = 0$ without missing any solutions. While there are obvious bounds on the CP multiplicities from the first equation, there are less obvious bounds from (linear combinations of) the others. Hence we may improve the bounds by combining equations. If all goes well, these two methods are sufficient to make a loop search possible.

These three methods can be invoked manually by means of the following commands

```
> Solve Tadpole Kernel (pos_int max_loop)
> Solve Tadpole Bound (pos_int interrupt)
> Solve Tadpole Loops (pos_int interrupt)
```

The arguments, if present, limit the scope or duration of the computations. The value of "max_loop" is the maximal number of candidate kernel vectors that is tested for each CP label. The default value is 10000. The value of "interrupt" is the number of seconds after which the user is asked if the computation should be continued. The default value is 60. The `Solve Tadpole` command performs these three operations successively with default paramater values.

The result of boundary reductions can be examined using

```
> Browse Tadpole Bounds
```

The set of kernel vectors obtained so far is shown by

```
> Browse Tadpole Kernel
```

A candidate solution can be be checked using the command

`> Check Tadpole Solution (entry_1)...(entry_n)`

Here "entry_i" denotes the value of a variable, which must be enterred as `label:value`. Here "label" is a valid variable label, corresponding to a column in the list of equations displayed by the command `Browse Tadpole Equations`. As remarked above, "value" is not directly a CP label: there is an implicit factor of 2 for symplectic groups, and an automatic pairing for unitary groups,

It is also possible to specify directly CP multiplicities for boundaries (as displayed by the `Browse Boundary` command). The command that does this is

`> Check CP (entry_1)...(entry_n)`

and has the same syntax as `> Check Tadpole Solution`. For example `Check CP 0:1 17:2` sets N_0 to 1, N_{17} to 2, and all others to zero, and then checks the tadpole equations for this set of labels.

`> Check Tadpole Parameters (entry_1)...(entry_n)`

Check solution to the tadpole conditions that is obtained by setting the parameters to selected values Entry_i is of the form label:value. Here "label" refers to the parameter "N_label", and "value" is the integer value assigned to it.

`> Reset Tadpole`

Description: Reset tadpole conditions.

`> Set Tadpole Info_level`

Description: Set amount of information printed while solving tadpole conditions.

`> Set Tadpole Show_spectrum`

Description: Switch automatic spectrum computation for basic solutions to tadpole cancellation.

14. Algorithms

14.1. Boundaries and crosscaps for exceptional automorphims

The formulas that are used are

$$\begin{aligned} K^i &= \frac{\sum_m S^i{}_m U_m U_m}{S_{0m}} \\ M^i{}_a &= \frac{\sum_m P^i{}_m U_m R_{ma}}{S_{0m}} \\ A^i{}_{ab} &= \frac{\sum_m S^i{}_m R_{ma} R_{mb}}{S_{0m}} \end{aligned} \tag{14.1}$$

The coefficients U_m will be referred to as the crosscap coefficients, and R_{ma} as the reflection coefficients. For fixed a the vector R_{ma} will be called a "boundary vector". The label m

refers to Ishibashi states, and is in one-to-one correspondence with the primaries that appear combined with their charge conjugate in the modular invariant. We impose the following requirements

1. $A^i{}_{ab} \geq 0$
2. $|K^i| \leq Z_{ii}$
3. $|M^i{}_a| \leq A^i{}_{aa}$
4. $M^i{}_a = A^i{}_{aa} \bmod 2$
5. $U_m = 0; R_{ma} = 0$ if m is not and Ishibashi label
6. $\sum_m R_{ma} R^*_{mb} = \delta_{ab}$
7. $A^0{}_{ab} = C^B_{ab}$

where C^B_{ab} is the boundary conjugation matrix, which must be an involution. The objective is to find a maximal set of boundary vectors satisfying these conditions. The maximal number one can get is equal to the number of Ishibashi states. Sets with the maximal number are called "complete".

Any solution to this problem is subject to the ambiguities

$$\begin{aligned} U_m &\to \epsilon_m U_m \\ R_{ma} &\to \epsilon_m R_{ma} \end{aligned} \tag{14.2}$$

and

$$R_{ma} \to -R_{ma} \tag{14.3}$$

These are fixed by choosing an ordering in the label a of the reflection coefficients, and a function $P(z)$ on the complex plane, satisfying $P(z) = \pm 1$ such that $P(z)P(-z) = -1$, $P(0) = 0$. This function generalizes the notion of "positivity" to the complex plane. For any m, the first occurrence of a non-zero entry in the ordered set $U_m, R_{m1}, \ldots, R_{m\ell}$ is then chosen "positive". Furthermore the first non-zero coefficient $R_{ma} \neq 0$, with $U_m \neq 0$, of the first boundary is chosen positive. Because of 6, $|R_{ma}|^2 \leq 1$. On the other hand

$$R^2_{ma} = S_{0m} \sum S_{im} A^i{}_{aa} \ .$$

Since $S_{i0} > 0$ for all i in an RCFT, we have

$$A^i_{aa} < [S_{0m} S_{i0}]^{-1} \tag{14.4}$$

The basic steps in the search procedure are as follows (if only boundaries are searched,

all steps involving crosscaps, Klein bottles or Moebius amplitudes are skipped; the sign ambiguity (14.3) is then already contained in (14.2)).

1. Compute the crosscap coefficients from the Klein bottle, using the first equation in (14.1).
2. Reject the result if requirement 5 is violated; Then there is no solution to the problem.
3. Search for diagonal annuli. This is done by generating non-negative integer vectors with as many components as there are primaries. These components are restricted by (14.4) and by $A^0_{aa} \leq 1$, so that this is in principle a finite search. In practice a complete search is impossible.
4. Solve for R^2_{ma}, and impose requirement 5.
5. Impose 6, *i.e.* $\sum_m |R^2_{ma}| = 1$.
6. Consider all relevant sign choices for $\sqrt{R^2_{ma}}$ and and compute the Moebius amplitude; then impose 3 and 4. Relevant sign choices are the signs for all m with $U_m \neq 0, R_{ma} \neq 0$ except the first m with that property (this fixes the $R_{ma} \to -R_{ma}$ ambiguity). Reflection coefficients surviving until this point are stored. In the following we will call these stored coefficients *unsigned boundary vectors*. This name is most appropriate if the crosscap requirements are *not* imposed, because then all stored R_{ma} are positive.
7. Consider all pairs of unsigned boundary vectors. Check if for any choice of the unrestricted signs (*i.e.* those for which $U_m = 0$) these coefficients can appear in one set, *i.e.* check requirements 1 and 6 for $a \neq b$. (Condition 7. then appears to hold automatically).
8. Construct all combinations of reflection coefficients that are pairwise consistent for some sign choice.
9. Consider all maximal combinations (*i.e.* those that are not contained in another combination). Consider for each boundary in that combination all remaining sign choices (*i.e.* all sign choices of $R_{ma} \neq 0$ if crosscaps are *not* considered, all sign choices on labels m with $U_m = 0$, $R_{ma} \neq 0$, plus the overall sign if crosscaps *are* considered. The signed boundary vectors in the combination are put in some definite order.
10. Form consistent sets by adding the boundary vectors one by one. A new vector is added to a set if it satisfies the positivity constraints fixing the sign ambiguities (14.2) and (14.3), and if it is orthogonal and yields non-negative off-diagonal annuli. Each allowed vector is added to all existing sets, and as the first member of a new set. When a boundary vector is complex, both the vector and its conjugate are added.
11. A set is complete if it has as many boundaries as there are Ishibashi states. Incomplete sets may be completed by choosing a complete set of vectors v_i orthogonal to the boundaries in the set. A "missing" boundary must be a linear combination of those vectors: $B_{m,\text{new}} = \sum \alpha_i v_i$. By computing the Moebius amplitudes $M^i{}_{\text{new}}$ and the off-diagonal annuli $A^i{}_{a,\text{new}}$, where a is an already present boundary, one gets many constraints on the coefficients, of the form $\sum \alpha_i \lambda_i = N_\lambda$, where λ_i is a known vector and N_λ an unknown integer. Since $|\vec{\alpha}| = 1$, the integers are bounded: $0 \leq N_\lambda \leq |\vec{\lambda}|$ for annuli, and $|N_\lambda| \leq |\vec{\lambda}|$ for Moebius amplitudes. This can be solved by standard linear

equation solving methods, if there exists a complete set of independent equations. To do so, all allowed N_λ must be checked. To optimize this, we choose an independent set of λ's of minimal integer range. For any solution to these equations the diagonal annuli $A^i_{\text{new,new}}$ are computed, and if they are non-negative and satisfy conditions 3. and 4. the solution is accepted, and stored as an unsigned boundary vector.

12. After step 11 is completed for all boundary sets of all combinations, and if this resulted in new unsigned boundary vectors, the process is repeated starting at step 7.

14.2. Solving the tadpole conditions

Remarks:

A shortcut is possible in steps 3 and 4. Requirement 5. implies that the diagonal annulus coefficients must satisfy

$$S_{mi}A^i{}_{aa} = 0$$

if m is *not* and Ishibashi state. These can be used as equations to solve for some $A^i{}_{aa}$ in terms of the other annuli. This reduces the loop search in step 3. This is usually faster, but due to additional overhead it may be slower in special cases.

To prevent runaway computations, several accumulations are limited: There is a maximal number of boundary combinations (step 8), and of boundary sets per boundary combination (step 10). By default both limits are set to 200. Finally the maximal number of equations considered in step 11 is ten times the number of undetermined variables. Exceeding any of these limits has no catastrophic effect, but results in an incomplete analysis of the possible boundaries. If the boundary combination limit is reached, no new boundary combinations will be considered, but the existing ones are extended by any allowed boundary vector; if the number of boundary sets reaches its limit, this is handled in the same way. All three limits can be simultaneously multiplied by n by means of the command

```
> Set Boundary_search Capacity [pos_int n]
```

The default is $n = 1$.

Completion of boundary sets (step 11) is only possible if for any Ishibashi label there is at least one boundary (or crosscap) that is non-vanishing for that label.

Step 11 may be done for *all* sets ot just for the sets of maximal size within a given combination. This option is chosen by the command

```
> Set Boundary_search Use_all_sets (switch ON/OFF)
```

The default value is OFF.

14.3. Solving the tadpole conditions

In general, the full set of tadpole conditions reduces to a set of equations of the form

$$N_a B_{ia} = \epsilon(-1)^{h_i} 2^{D/2} \Gamma_i$$

where B_{ai} are the boundary coefficients and Γ_i the crosscap coefficients, and D the space-time dimension. The equations must be solved for the Chan-Paton multiplicies N_a. The precise choice of labels i depends on the kind of string theory (super or bosonic), and is dealt with automatically. For a given string theory, the tadpole equations for *all* physical massless particles are taken into account. These are either descendants of the vacuum ($h_i = 0$, *i.e.* the dilaton) or ground states with $h_i = 1$. Although it is possible to compute spectra for CP-multiplicities that do not satisfy all equations, one cannot find such solutions automatically.

The CP-multiplicies N_a must all have the same sign; if not, there will inevitably be annuli $N_a N_b A^i{}_{ab}$ that are negative. The dilaton tadpole coefficients B_{0a} are always positive, at least in all known cases. If Γ_0 is non-zero and real we can fix ϵ in such a way that all N_a are non-negative. [In all known cases Γ_i is real; if Γ_0 vanishes, all CP-multiplicities are zero.]

The $i = 0$ equation bounds all CP-multiplicities from above (since $B_{0a} > 0$ for all a), and hence a systematic nested loop search for all solutions is possible in principle. In practice, this is usually not feasible.

A reduction of the loop length can be achieved in the following ways

— Identify kernel vectors. We look for a set of solutions to the homogeneous equations with one entry equal to -1 (the "tag") and all other entries non-negative. Tags of different vectors should be distinct, and no tag should overlap with a non-zero entry of another kernel vector.

Suppose we have a solution that is non-zero on one of the tags. We can make that entry vanish by adding an integer multiple of the corresponding kernel vector. This may increase some of the other CP-multiplicities, but in any case we obtain another valid solution. Since tags do not overlap with non-zero entries of other vectors, we can repeat this for all other kernel vectors. In this way we can make all tagged entries vanish. It is therefore not necessary to look for solutions that are non-zero on the tagged entries: such solutions can be reconstructed at the end. Hence the loop search on the tagged entries reduces to zero.

To implement this four kinds of searches for such vectors are performed: A search for vectors with one, two and three entries equal to 1, and the others (apart from the tag) equal to -1, plus an attempt to solve the homogeneous equations directly with a value -1 for one of the entries that is not tagged yet. The latter method has no constraint on the other entries, except that they should be non-negative and vanish on previously obtained tags.

— Reduction of bounds. So far only the dilaton tadpole equation was used to limit the range of the variables. Using the other equations or linear combinations of equations

may lead to further constraints. Any such equation has the form

$$\alpha N + \beta_i N_i + \gamma_j N_j = \delta$$

where we have organized the equation in such a way that $\alpha > 0, \beta_i > 0$ and $\gamma_i < 0$, and N is the CP-multiplicity singled out for analysis. The second term is minimized if N_i is at its minimum, the third term if N_j is at its maximum. This minimum determines a maximal value for N, which becomes the new maximum if it is smaller than the previous one. Similarly a new minimum is determined. The reduced range for N may in its turn lead to a reduced range for the other variables. This method is applied to suitably chosen linear combinations of the original equations, until a time limit is reached, or until no changes occur after a certain number of steps.

— Double tag vectors. These are kernel vectors with two entries equal to -1. These tags should not coincide with the single tags discussed above, nor with positive values of other kernel vectors. Furthermore one of the two tags of vector A should not coincide with the two tags of any vector B. If we know about the existence of such a vector with tags i and j we may demand that either i or j vanishes. This does not reduce the maxima and minima of the variables, but it can be used to reduce the loop search dynamically: the value of j is not increased if i is non-zero or vice versa. Obviously this can be generalized to multiple tags, but this has little practicle advantages.

— Using the equations of motion. If there are N_e independent equations, as many variables can be determined by solving an equation. The corresponding variables can be removed from the search. For each equation, the variable with the largest range is the one that is selected, in order to get the maximal reduction.

— Secondary kernel vectors If after all these reductions the loop search still has multiple solutions, these solutions evidently differ by kernel vectors that were not detected yet. These vectors are collected. If all goes well, one obtains an independent set of integer vectors such that any solution found in the loop search can be obtained from the first solution by adding integer linear combinations of these vectors. If this is the case we speak of a unique solution.

Obviously the solution would always be unique if we work over the rational numbers, but not necessarily over the integers. Solutions that cannot be obtained by integer linear combinations are stored as separate solutions.

Finally we must determine which of the primary kernel vectors can actually be used. The constraint here is the requirement of positivity of the CP labels. Given a solution of the loop search, any single or double tag kernel vector can only be subtracted from it.

15. Checks

This chapter contains a variety of commands to check the internal consistency of the program, or to check certain conjectures.

`> Check Branching [pos_int G_type][pos_int G_rank][pos_int H_type][pos_int H_rank]`
Check the internal branching tables. These tables specify the decomposition of the adjoint and the fundamental representations for the Lie-algebra embedding $G \supset H$, where G and H are of types B, D or E. These tables are used in superstring spectrum computations

`> Check Closed (pos_int max_err)`
Check positivity and integrality of Torus + Klein bottle. ("max_err" is the maximum number of error messages; Defaults to no limit).

`> Check Conjugates`
Check $S^2 = C$.

`> Check Eta`
Check $Y(f,0,i)/Y(f,0,0) = e^{i\pi h_J}\eta^J$.

`> Check Fp_conj`
Check $(S^J)^2 = \eta^J C$.

`> Check Full_open (pos_int max_err)`
Check all open string consistency conditions ("max_err" is the maximum number of error messages; Defaults to no limit).

`> Check Homomorphism (field lbl1)...(field lbln)`
Check homomorphism generated by lbl1...lbln.

`> Check Hopf`
Check rational Hopf algebra relations.

`> Check Klein (pos_int max_err)`
Check positivity of $N_{ijk}K_iK_jK_k$ ("max_err" is the maximum number of error messages; Defaults to no limit).

`> Check Modular`
Check unitarity and $(ST)^3 = S^2$.

`> Check Open (pos_int max_err)`
Check positivity and integrality of Annulus + Moebius strip ("max_err" is the maximum number of error messages; Defaults to no limit).

`> Check Orbits`
Check that the algebraic simple current action agrees with the fusion rules.

`> Check Simple [field a]`
Check if "a" is a simple current.

`> Check Symmetry`
Check symmetry of S.

`> Check Trace_formula`
Check the boundary trace formula for the annulus

`> Check Twist_prod`
Check product rule in second argument of F(a,K,J).

`> Check Twists`

Compare current twists and eta twists (only shows errors).

```
> Check Unitarity (field a)(field b)
```

Check unitarity.

16. Functions

The following functions are provided to get easier access to internal parameters, especially for programming purposes.

```
> Total Boundaries
```

Return total number of boundary labels

```
> Total Invariants
```

Return total number of modular invariants stored as "Invariant_n"

```
> Total Ishibashi
```

Return total number of Ishibashi labels

```
> Total Primaries
```

Return total number of primaries

```
> Total Simple_currents
```

Return total number of simple currents

17. Syntax

17.1. General command line structure

Execution starts after each completed command line. A "command line" may be built out of several "input lines", which are either lines in a file, or lines enterred on a terminal.

A command line is 'completed' if does not end with a continuation character '\' or has unbalanced curly brackets "{ }". If a line is not complete a next line is read and added to the previous one. In interactive mode the prompt for a continuation line is "... " instead of the standard prompt "> ". The continuation character '\' is only treated as such if is the last character of an input line.

Blanks at the beginning of each input line, including continuation lines, are ignored. However, when a line is continued because curly brackets are not closed, a blank is inserted at the end of the incomplete line.

Examples:

```
> Command\n            → Command
> Comm\\n
...  and\n             → Command
> Comm \\n
...  and\n             → Comm and
> This {is a\n
...  continuation}     → This is a continuation
```

Here '\n' denotes the end-of-line character ("return" on a terminal).

17.2. Command Blocks

A command line consists in general of several command blocks, separated by ";". The end of each command line has an implied ";".

Each command block in its turn may consist of more than one commands. The separate commands in a block are not indicated by the user, but are determined by the interpreter.

Example:

```
> Command1 Command2; Command3 Command4 Command5;
```

is a command line consisting of two command blocks, which in their turn consist respectively of two and three commands. The closing ';' is optional.

17.3. Command Interpretation

Each command has the following structure: "Command (Keywords) (Parameters)"

Any string bounded by blanks (a "word") is a command, keyword or a parameter depending on its position. Any string bounded by opening and closing curly brackets is read as a single word. Within the curly brackets, blanks, all other brackets, ";" and "\" are ignored (i.e "\" and ";" are treated as ordinary characters).

Keywords are character strings that can be recognized by the command interpreter, whereas parameters are unrestricted at the interpreter level (e.g. file names). A command may be executable with or without keywords, and the number of parameters may be fixed or variable. This information can be obtained using "help". Round brackets indicate optional parameters, square brackets indicate mandatory ones.

Examples:

```
> File (filename)
> File on (filename)
> File off
```

The command "File" can have keywords "on" or "off" but is also executable without any keywords. In the first two cases a parameter "filename" may be supplied, but is not obligatory.

```
> If [cond][comm1][comm2]
```

means that the command "If" must have three parameters.

```
> Command [par1][par2](par3)(par4)
```

means that "Command" has at least two and at most four parameters.

```
> System [command...]
```

means that "system" must have at least one, but may have any larger number of parameters. The syntax could also have been shown as

```
> System [command]...
> System [command_1](command_2)...
> System [command_1](command_2)...(command_n)
```

which all mean exactly the same. Finally,

```
> Command (parameters...)
```

would indicate that any number of parameters is allowed.

Commands and keywords are case insensitive and may be abbreviated in any unambiguous way unless a keyword could be mistaken for a parameter.

Example: "File of" opens an output file named "of" and is not equivalent to "file off".

Keywords must be enterred in the correct order, as specified in the "help" command.

17.4. Command Block Interpretation

The command interpreter will always try to use the maximal number of words of the command block. The first word is the command itself. If first level keywords exists, they are compared with the second word. If this matches, and the "command keyword" combination may have a second level keyword, the third word is checked against the second level keyword list, etc.

If the number of remaining words exceeds the maximum number "p" of parameters for the "command keyword_1 ... keyword_k" combination, then "p" of the remaining words are passed on to the command for execution. Depending on the command, some of the parameters may be rejected (for instance if a number is expected, but a string is supplied). If the command can be executed with fewer parameters, it is. Each command returns either an error code or the number of parameters it has been able to use. In the former case execution for the entire command line is stopped. In the latter case execution continues with the first word that has not been used (even if it was passed on to a command, but was rejected).

Example: Suppose `Command (par1)` as well as `Command Key1 (par1)` are valid. Suppose

```
> Command Key1 AAA;
```

is enterred. This will then be interpreted as `Command Key1` with parameter par1=AAA. The command

```
> Command AAA
```

will be interpreted as "`Command`" with parameter par1=AAA, since AAA cannot be recognized as a keyword. Note that in this example it would not be possible to pass "`Key1`" as a parameter to "`Command`". (This can be evaded using macro substitutions, see below).

The next word in the command line is first tried as a keyword of the previous command at the last level (i.e. keyword_k) If it doesn't match, the next lower level is checked until the command level is reached.

Example:

```
> Status Errors Memory
```

performs first `Status Errors` and then `Status Memory` (even if both keywords are followed by parameters, which in this example is not possible).

```
> Command1 Command2
```

performs first `Command1` and then `Command2`, although in this case

```
> Command1; Command2;
```

is preferable.

The main purpose of allowing more than one command in a command block is to allow

```
> Command keyword1 keyword2 keyword3
```

as a shortcut for `Command keyword1; command keyword2; command keyword3;`
Other uses of this feature are not recommended, since the possibility of ambiguities is obvious. In particular commands with an indefinite number of parameters require termination by ";".

Exceptions: The `Help` command can have any valid command and keyword combination as parameters, but anything remaining in the command block is ignored

17.5. Comments

Any text following a "%" is ignored. This can be used for comments. The "%" is treated as a "newline". This is even true within brackets.

17.6. Macro definitions

The command

```
> Define X Y
```

sets X equal to Y. This means that if the string X appears in any command block, it is replaced by Y. No substitutions are made within curly brackets.

Replacements are done recursively, applying all macro defintions until no changes take place. The string Y is not restricted in any way, except that it is not allowed to contain X as a substring, thus producing an infinite recursion. It is not difficult to produce infinite recursion by means of two or more `Define` commands. Avoiding this is the responsability of the user.

The command

```
> Undefine {X}
```

removes the macro definition for X. Note that the brackets are essential for this to work properly, since otherwise "X" would be replaced by its alias before `undefine` is executed.

Preferably macros should be clearly distinghuished from ordinary strings, for example by choosing as the first character a '$'. For example

```
> Define p l
```

would make the execution of the "help" command impossible, except by enclosing it in curly brackets. Avoiding such problems is also the responsability of the user.

```
> Free Macros (macro)
```

17.7. Variables

In addition to macros one can also assign strings to variables. This can be done with the "command":

```
> X=Y
```

Just like `Define` this assigns the string Y to the variable X (which is created if it doesn't exist). However, the replacement of X by Y only takes place if X appears as a parameter of a command, or if X appears in an "expression" (see below). Replacement is non-recursive, unlike `Define`. Curly bracket are irrelevant, since replacement takes place only after curly brackets have been resolved.

One may use the same variable names for assignments and macros, but this is not useful, since macro substitutions are done before variable substitutions.

Example

```
> X=1
> Evaluate X
1
```

The "`Evaluate`" command evaluates an algebraic expression (see below). Here it evaluates the expression "1", obtained after substituting the variable X.

Variables are only replaced in parameters of a command, not in keywords. Choosing a variable name that coincides with a keyword is allowed, but may lead to conflicts.

Variable names are always stripped of all blanks. Hence the statement

```
> { A B = 100 }
```

assigns the value 100 to the variable "AB". Variable names may not begin with digits. See also "Expression syntax" below.

17.8. Variable substitution

If a parameter matches a variable name, the value of that variable is automatically substituted. For example

```
> fn=file|1
> file fn
```

directs output to "file1" (the symbol | concatenates two strings, see below). This may be used to direct output to distinct files depending on the result of a computation.

If automatic substitution is not desired it can be switched of for a given command by typing

```
> Set Disable_sub [command](key_1)...(key_n)
```

and switched on again by

```
> Set Enable_sub [command](key_1)...(key_n)
```

Note that each keyword is a separate parameter. Hence the correct syntax is as above, and *not* to use curly brackets as in

```
> Set Enable_sub {command key_1}
```

Most commands have substitution disabled by default (e.g. "help" and "evaluate"). This is in particular true for all commands that have parameters of definite type (see below under "parameter types"), since in that case the substitution is anyway done during the type evaluation. If automatic substitution is enabled this is shown in the "help" text. Otherwise it is disabled.

17.9. Algebraic instructions

Three types of special command words are recognized: expressions, assignments and conditions.

— Expressions: An "expression" consists of "objects", round brackets "(" and ")", which must be closed, and operators, which act on two objects and replace them by a another. The simplest way an expression can occur is as the parameter of the "`evaluate`" command.

Example:

```
> Evaluate 1+1;
2
```

For more information, see the section on expression syntax below.

— Assignments: They have the form X=expression, where X is a variable.

The assignment statement is the only exception to the rule that every command must be of the form "Command keywords parameters". It must be a single word, or be enclosed in curly brackets.

— Conditions: Conditions are special kinds of expressions that return either TRUE or FALSE. Basic conditions have the form "Expression1 * Expression2",where "*" is one of the following:

==	Equal to
!=	Not equal to
<=	Smaller than or equal to
>=	Larger than or equal to
<	Smaller than
>	Larger than
=	Equal (for strings)
>	Larger than (for strings)
<	Smaller than (for strings)

In the first six conditionals expressions 1 and 2 must evaluate to integers or rationals The string comparisons are defined as for the C-function 'strcmp'. Conditions may be combined using "and" (&&) and "or" (||).

Conditions may be tested using the evaluate command.

17.10. Expression syntax

An expression may contain blanks, but in that case it must be enclosed by curly brackets, so that it is read as a single word.

Example:

```
> evaluate {10 * (4 + 5)};
90
```

Valid operators are:

Nr.	Symbol	Operation	Left Arg.	Right Arg.	Return
0	/	Division	i,l,r	i,l,r	i,l,r
1	*	Multiplication	i,l,r	i,l,r	i,l,r
2	−	Subtraction	i,l,r	i,l,r	i,l,r
3	+	Addition	i,l,r	i,l,r	i,l,r
4	==	Equality	i,l,r	i,l,r	c
5	!=	Inequality	i,l,r	i,l,r	c
6	>	Greater than	i,l,r	i,l,r	c
7	<	Less than	i,l,r	i,l,r	c
8	>=	Greater than or equal	i,l,r	i,l,r	c
9	<=	Less than or equal	i,l,r	i,l,r	c
10	<∼	Less than (using strcmp)	s, (c,i,l,r)	s, (c,i,l,r)	c
11	=∼	Equal (using strcmp)	s, (c,i,l,r)	s, (c,i,l,r)	c
12	>∼	Greater than (using strcmp)	s, (c,i,l,r)	s, (c,i,l,r)	c
13	&&	Logical AND	c	c	c
14	||	Logical OR	c	c	c
15	|	String concatenation	s, (c,i,l,r)	s, (c,i,l,r)	c
16	=	Assignment	s	s,c,i,l,r	s,c,i,l,r
17	∼	String assignment	s	s,(c,i,l,r)	s

Here “strcmp” refers to the return value of the C-function “strcmp”. This list also defines the priority of these operations. Types between parenthesis are converted to the first type, ’s’, since all types can be converted to strings. The left arguments of operators 16 and 17 are strings that are used as variable names to which the right-hand sides are assigned. Any valid variable name is allowed.

Blanks Trailing and leading blanks are removed from all operator arguments before evaluation, with the exception of the string operations 10, 11, 12, 15 and 17 (second argument only). For example

```
> Evaluate {7== 7}
TRUE
> Evaluate {A=∼ A}
FALSE
```

Since the second operation compares the strings “A” and “ A”.

Variable substitution

All operator arguments except the left arguments of operators 16 and 17 ar replaced by their value if they are variable names.

All arguments are strings, but they can be converted to various types, depending on the operation. The types are

s:	String	
i:	Integer	(< 10000)
l:	Large Integer	(Unlimited)
r:	Rational	(Ratio of two Large Integers)
c:	Boolean	(TRUE or FALSE)
e:	Error	

Columns 4 and 5 of the table of operators indicate which types are acceptable for the left and right argument of each operator.

Operator 17 is identical to 16 except that the latter removes double, leading and trailing blanks from the string on the right hand side. Operators 10,11,12 and 15 act on all types, after converting them to strings.

If an operation results in an error (e.g. 1/0) the operation return type 'e'. Any subsequent operation involving type 'e' yields again type 'e'. The distinct types are only used internally, and are not seen by the user.

An expression may contain variables, which must be enclosed by blanks, brackets, or operators. They are non-recursively replaced. In integer operations, the final replacement must yield an integer, or an error will occur.

Replacement takes place at the latest possible moment, namely just before one of the operations is about to be performed. In (sub)-expressions of the form "A x B", where "x" is an operator from the table, variable substitution always takes place for "B", but not for "A" is the operation is "=" or "∼". Therefore sub-expressions like (Y=2) will work correctly even if Y already has a value. Also Y=Y+1 works as one might expect: the value of Y+1 is assigned to Y.

If no operation is performed, strings are simply returned as strings. For example

```
> Evaluate (3)(4)
3)(4
```

returns the string between the outer parentheses, without any attempt at evaluation.

Floating point numbers

In all arithmetic operations floating point numbers in non-exponential form (e.g. 3.1459) are acceptable as input. They are converted to rational numbers with the full precision of the input number, and are processed as rational numbers, *i.e.* without loss of precision.

The answer can be converted back to floating point format by means of the "`float`" command. This command is exactly the same as "`evaluate`" except that the output is assumed to be an integer or a rational number, which is printed as a floating point number. For example

```
> Float 3.1459*3.1459
9.896686810000000
```

17.11. The `Evaluate` command

By default, no automatic variable substitution is done fore the "evaluate" command. However, if an expression contains no operators a substitution is attempted. There is no substitution in the result of an expression evaluation.

Example:

```
> AB=9
> Evaluate AB
9
```

because "AB" is an expression without operators;

```
> Evaluate A|B
AB
> Evaluate N=A|B
9
```

In the last case conversion is enforced by the "=" operation. Note however

```
> Evaluate (A|B)
9
```

Here the parentheses are evaluated first, yielding "AB". At the next lower parenthesis level, this yields the single-term expression "AB". Its evaluation yields 9. Finally consider the following example of non-recursive variable substitution

```
> A=B
> B=C
> C=5
> Evaluate A
B
> Evaluate (A)
C
> Evaluate ((A))
5
```

Here each brackets enforces another expression evaluation.

In expressions with identical operators and no brackets evaluation takes place from left to right

```
> Evaluate 1/2/3
1/6
```

and also

```
> AB=8
> ABCD=9
> Evaluate A|B|C|D
8CD
```

Retrieving the last evaluation result

The result of each evaluation is stored in the internal variable "$last", which can be used as input in expressions. This is not only true for the result of "`evaluate`", but also for "`float`" and "`execute`" (which is the same as "`evaluate`", but does not print standard output).

(Two other related commands are "`print`" and "`return`", see below; they do not store their results in "$last".)

When "`evaluate`" (or "`float`" and "`execute`") are called as functions the return value is *not* stored in "$last".

Quotes and escapes

Quotes (") and/or the escape character \ may be used to employ special characters in strings. Any character preceded by \ is treated as an ordinary character with no special significance. This includes \" and \\. For example

```
> Evaluate "7*8="|7*8
7*8=56
```

The same result is obtained by means of

```
> Evaluate 7\*8\=|7*8
```

All unescaped quotes are removed just before printing or returning output, in the commands 'evaluate", "execute", "print" and "return". After removing quotes, all unescaped escape characters are removed. For example

```
> Evaluate "\*"
*
> Evaluate \"string\"
"string"
> Evaluate \\a
\a
```

Note that quotes escape and escapes are *not* removed after expression evaluations unless these evaluations are done by invoking one of the four commands mentioned above. For example

```
> m="a"
```

assigns the value ""a"" (including quotes) to "m". If the variable "m" is used subsequently in the evaluate command, the quotes are removed in the output.

A second point where quotes and escapes are removed is in variable names. Hence

```
> "b"=1
```

assigns 1 to *b* (unquoted).

17.12. Control flow

The following commands are available

```
> Repeat [integer][CommandLine]
```

This repeats "CommandLine" "integer" times

```
> If [Condition][True_CommandLine][False_CommandLine]
```

If "Condition" returns TRUE, perform "True_CommandLine", else perform "False_CommandLine".

```
> For [Assigment1][Condition][Assignment2][CommandLine]
```

Perform "Assignment1". Then, as long as "Condition" returns TRUE, perform "CommandLine", followed by "Assignment2".

```
> While [Condition][CommandLine]
```

"CommandLine" is executed as long as "Condition" is TRUE.

```
> Break (integer)
```

Jump out of "integer" nested loops. Without parameter: return to terminal or file input level. "integer" is lowered by 1 each time a "for", "while" or "repeat" loop is broken off (unless this is due to errors).

Example:

```
> For X=1 X<10 X=X+1 {Evaluate X*X}
```

17.13. Procedures

The command

```
> Procedure [name][commandline]
```

defines a procedure "name" which executes the commandline each time it is called.

The syntax of "name" is "procname(a1,...,an|b1,...,bn)", where "a1,...,an" are obligatory parameters and "b1,...,bn" are optional.*

A procedure is called exactly like a command, namely as

```
> procname v1 v2 ...  vn w1 ...  wn
```

where "vi" are the values of "ai" and "wi' those of "bi". The parameter values "wi" may be omitted, either completely or for i>j, for some j. Just as command names, procedure names may be abbreviated. The names are case insensitive, so that for example defining a procedure "Sqrt" as well as a procedure "sqrt" is not possible.

A procedure appears as a command in the command list, and basic help info is available:

```
> Proc procname(a1,a2,a3|b1,b2) commandline
> Help procname
> Command syntax:  Procname [a1][a2][a3](b1)(b2)
```

Example:

```
> Proc squares {For X=1 X<10 X=X+1 {Evaluate X*X } }
```

defines a procedure that prints the first nine squares each time it is called.

```
> Proc squares(N) {For X=1 X<N+1 X=X+1 {Evaluate X*X} }
```

does the same as the previous one, but N rather than 9 times. This procedure is called in the following way:

```
> squares 3
1
4
9
```

Arguments of procedures are treated as dummy variables. Example:

```
> N=3
> proc test(N) {eval N}
```

* At present optional parameters have no practical use, since there is no method available to detect them.

```
> test 5
5
```

After these commands the value of N is 3, and not 5.

Dummy variables are internally represented by names starting with '%'. Since this is also used for comments, it is impossible for the user to use such internal variables.

Procedures can be removed using the command

```
> Free Procedures (procedure)
```

If no argument is given, all user-defined procedures are removed.

17.14. Functions

Any command or user-defined procedure may also be called as a function. Functions are called in expressions. There are two differences between a function call and a procedure call

- The arguments of a function are evaluated as expressions; this is in general not true for command parameters.
- The output is returned to the expression in which the function is called.

The main purpose of functions is to make the return value available for further computations.

Examples

```
> procedure fac(n) {
> m=1;
> for l=1 l<=n l=l+1 m=m*l;
> evaluate m;
> }
```

This procedure computes n!. When used as a command one types

```
> fac 10
3628800
```

But one may also type

```
> evaluate fac(10)
3628800
```

In the latter case, the argument may itself be an expression

```
> evaluate fac(2*5+7)
355687428096000
```

In this case the command line call

```
> fac 2*5+7
```

would have the same effect, but for a different reason: the string "2*5+7" is substituted for "n" in "l¡=n" and evaluated as soon as that expression is evaluated. Erroneous use of expressions in command parameters may lead to bizarre results, such as

```
> Procedure square(N) {evaluate N*N}
> Square 10+8
98
```

(note that 98=10+8*10+8). This can be avoided by using brackets either in the procedure definition or around the parameter. If a parameter has a definite type (as defined below)

the type conversion may involve evaluation of expressions. In that case "10+8" would be replaced by 18 before passing it on to the command. However, at present user-defined procedures cannot have type definitions, so that the foregoing remark only refers to built-in commands.

The return value of a function may be used in expressions

```
> evaluate fac(4)+6*fac(11)
239500824
```

Functions may be called recursively to arbitary depth

```
> eval fac(fac(fac(3)))
260121894......0000000000000000
```

(with many digits of 720! omitted)

Any command may be used as a function, but of course not all commands have useful return values.

Procedure names may contain blanks, and hence keywords can be used in function names. In fact keywords may also be specified as function arguments, and procedure parameters may appear already before the brackets. The function call must be terminated by a closing bracket. For example, the following calls are equivalent

```
> file on filename
> execute file(on,filename)
> execute {file on(filename)}
> execute {file on filename()}
```

Note the need for curly brackets in the last two statements. If a keyword is given as a function argument it too is subject to evaluation, as in

```
> eval status(mem|ory)
```

Return values

Not all output is returned by a function as a "return value". There are two exceptions: error messages and output tables (see below).

In addition to the "evaluate" command there are three other commands that differ only in the way output is directed. The four commands are:

— `Evaluate`: Prints and returns output.

— `Execute`: No output (printed nor returned).

— `Return`: Only returns output, does not print it.

— `Print`: Only prints output, does not return it.

Extracting data

Numbers in the return value of a function can be extracted from the output string using

```
> Extract [pos_int n][string]
```

This extracts the n^{th} number from "string" (counting starts with 1). In command line use of "`extract`" the string is used literally, except if it is a variable name (automatic variable substitution is enabled by default for "`extract`"). For example

```
> Extract 3 {The 3^ th number in this string is not 1, but 5}
```

```
5
```

If "extract" is called as a function, "string" is evaluated as an expression, for example

```
> eval extract(1,time())
0.16
```

Note that the empty brackets following "time" are necessary to indicate that it is a function. This example also shows that floating point numbers are correctly extracted. However, exponential forms like $0.1E-20$ are not recognized as a single number.

The "extract" command treats any uninterrupted string of digits (including a preceding − sign and at most one dot) as a number, regardless of the characters immediately in front or behind that string.

17.15. Global and local variables

All variables that are not explicitly declared are local to a function or to the command level. For example in the factorial function defined above, the values of l and m do not overwrite the values any parameters l or m that might exist at the command level, or in functions from which "fac" is called. Conversely, variables defined at command level are not known to a function unless they are passed on as parameters.

Global parameters can be defined by means of the "global" command.

```
> Global X=500
```

the value of X is now know to all functions.

If a function defines its own variable X that has the same name as a global variable, the local definition takes precedence. In parameter substitutions local variables also take precedence.

Parameters are passed to functions by value; the variables of the calling procedure are not changed. Functions only have access to global variables, not to local ones except its own.

Variables can be freed (removed, in order to save memory) using one of the commands

```
> Free Glubal (variable)
> Free Local (variable)
> Free Variable (variable)
```

Without parameter, these commands free respectively all global variables, all local variables at command level, and both of the above. Local variables at deeper levels (*i.e.* those used in functions) cannot be freed manually; they are automatically freed when the function is completed.

17.16. Parameter types

Parameters of commands can have datatypes that are shown in the "help" output. For example for the "repeat" command we find

```
Repeat [pos_int count][commandline]
```

The type of "count" is "pos_int". This means that the character string "count" is converted to a positive integer, if possible. If "count" is an expression, it is evaluated, and if it is a variable, it is substituted. If the result is not a positive integer the command will return an error. For other datatypes there will be other conversion methods, or no conversion at all. In general conversion to integers involves the evaluation of expressions.

The parameter types can be inspected with the command

```
> Browse Datatypes
Type                Description
id                  String
switch              ON or OFF
sign                Sign (+1 or -1)
non_neg_int         Non-negative integer
pos_int             Positive integer
int                 Integer
```

Here only the types are shown that are built into the interpreter; the actual set of types depends on the program. The last for types convert to integer values. The type "switch" accepts the strings "ON" or "OFF", where each character may be upper or lower case.

Between the typing of a command line and the execution of the command the following expression evaluations and variable substitutions may take place, and in the following order

1. Automatic expression evaluation for function arguments.
2. Local variable substitution in command parameters.
3. Global variable substitution in command parameters.
4. Argument evaluation due to type conversion of command parameters.

Furthermore parameter substitution also takes place during expression evaluation, and for operatorless expressions. To avoid undesirable multiple evaluations, parameter substitution (local and global) is by default switched off for any command with one or more parameters of definite type (it can be switched on if desired). Furthermore expression evaluation in type conversions is switched off when a command is called as a function. This ensures that in all situations only a single evaluation takes place. If mutiple evaluations are needed, this can be achievedby multiple nested calls of "`Evaluate`", or by using brackets.
Consider the following example, involving the "`repeat`" command, whose first argument is a positive integer (type "tt pos_int"). `> A=B`

```
> B=C
> C=2
> Evaluate repeat(A,time)
Non-numerical parameter:  B
RETURN CODE 1
> Evaluate repeat((A),time)
Non-numerical parameter:  C
RETURN CODE 1
> Evaluate repeat(((A)),time)
CPU time used 0.02 seconds
CPU time used 0.02 seconds
> repeat A time
Non-numerical parameter:  B
```

17.17. Math mode

In math mode all lines that are enterred are treated as parameters of the "`Evaluate`" command. Hence typing "1+1" leads directly to the answer "2". Math mode is turned on by the comand

```
> Set Math (switch ON/OFF)
```

The default is ON.

To turn math mode off it should be remembered that only expressions are accepted as input. Hence one should type `> Set Math(OFF)` to leave math mode.

17.18. Finding commands

Several commands are available to find commands.

```
> Find [string]
```

finds all commands with "string" as a substring of the command name or one of the keywords.

```
> Apropos [string]
```

finds all commands with "string" as a substring of the command name or one of the keywords, or of the primary help text.

```
> Commands
```

lists all commands

```
> Commands All
```

lists all commands and keywords, and

```
> Commands Detail
```

lists all commands keywords and primary help texts. Finally

```
> Help (command)(key_1)...(key_n)
```

gives the command syntax and the help text for the command.

The help text may consist of two parts. The second part may be shared by several commands, The first part is what is referred to as "primary help text" above.

17.19. Input and output

```
> File (fn)
> File Off
> Terminal
> Terminal Off
> Terminal On
> Load (filename)


> Prompt
```

read further input from terminal

```
> Set Echo (switch ON/OFF)
```

Switch command echoing

```
> Set Interactive (switch ON/OFF)
```

Switch terminal input

17.20. Printing

```
> Print [string]
```

17.21. System commands

```
> System [cmd...]
> Time
```

17.22. Tables

```
> Table Header [table]
> Table Info [table]
> Table Left_adjust [table][column]
> Table List
> Table Noheader [table]
> Table Restore [table][column]
> Table Right_adjust [table][column]
> Table Set_space [table][column][pos_int s]
> Table Suppress [table][column]
```

17.23. Memory management

```
> Free All
> Status Memory
```

17.24. Status commands

There are several commands to check the status of various parameters of the program. The command

```
> Status Definitions
```

displays all current macro definitions, global and local variables, and user-defined procedures.

```
> Status Errors
```

Displays all error messages that have occurred (this may include messages that were not displayed. Calls to the error routine can also occur during normal operation, and hence if messages are displayed this does not necessarily mean that there is a problem). The command

```
> Status Memory
```

shows the total amount of dynamically allocated memory, in bytes. There are two keywords

```
> Status Memory Detail
> Status Memory More_detail
```

which are self-explanatory. The value of internal parameters can be displayed using

```
> Status Parameters
```

Some of these parameter values (indicated by "(*)") can be saved in a default file for future runs. To browse the current defaults type

```
> Status Preferences
```

The values that are shown by the latter command may not correctly reflect changes in the values. This is discussed in the next subsection.

17.25. Quiting

```
> Quit
```

Exit program.

17.26. Defaults and preferences

There are in general several parameters that affect the program. This includes the ones listed by the "`Status Parameters`" command, the ones listed by the "`Status Preferences`" commands, and others. All these parameters have default values, and some of these defaults can be changed permanently to the values preferred by the user. The defaults are stored in a file. The name of that file can be shown using

```
> Show Default_file
```

Each line in this file has two entries, and is of the form
"Parameter" "Value"
Some of the lines in the defaults file may refer to output tables. These are lines of the form
"Table:Name:Parameter" "Value"

There is no generic command for changing default values. Parameters that appear with "`Status Parameters`" can be changed with the "tt Set" command. The values of the these parameters can be inspected individually by means of the "`Show`" command, with the same keywords as "`Set`". Table defaults can be changed with the "`Table`" command. For other parameters special commands may be available. The changes are saved in the defaults file by typing

```
> Save Preferences
```

The defaults file can be edited with a standard text editor. Mistakes in the parameter names are ignored (*i.e.* the "value" is not assigned to any parameter). On the other hand, mistakes in the values may have important consequences. It is better not to change the table defaults by hand, but by means of the "*Table*" command. Mistyping a table name leads to an error message (which, however, has no consequences other than ignoring the line in question).

The command `> Status Preferences`
shows the preferences as they were read in, but may not show some changes made after starting the program. This is because some default parameters may be mapped internally to other variables, and only the latter are updated. The internal variables are mapped back to the parameters in the defaults file when the command "`Save Preferences`" is enterred. Hereafter the updated values are shown correctly by the " `Status Preferences`" command.

17.27. Error messages

```
> Explain
> Status Errors
> Reset Errors
```

```
> Set Error_log (switch ON/OFF)
```

Switch error log file use

```
> Set Error_trace (switch ON/OFF)
```

Switch error trace

17.28. Debugging

```
> Set Debug Level [pos_int level]
> Set Debug Alloc_trace
> Set Debug Off
```

```
> Set Log (switch ON/OFF)
```

Switch log file use

17.29. Built-in commands

The following commands (and keywords) are known to the interpreter itself. Each program it is used for may add its own commands.

Apropos	Find string in help texts
Break	Jump out of loop
Browse Datatypes	Display data types
Commands	List all commands
All	List all commands and keywords
Detail	List all commands,keywords and help info.
Continue	Do nothing
Define	Macro definition
Evaluate	Evaluate expression and print/return result
Explain	Explain errors
Extract	Extract integer from string
Execute	Evaluate expression but do not print/return result
File	Switch file output on/off
Find	Find a command
Free All	Remove macros,procedures and variables
Local	Remove local variables
Macros	Remove macro definitions
Global	Remove global variables
Procedures	Remove procedure definitions
Variables	Remaove local and global variables
For	Start loop
Global	Define a global variable
Help	Command information
If	Conditional switch
Info syntax	Print this list
Load	Read command or procedure file
Math	Switch math mode on or off
Procedure	Define procedures
Print	Print strings directly (not returned in functions)
Quit	Terminate execution
Repeat	Repeat instruction
Reset Errors	Set error messages to “do not ignore”
Return	Evaluate expression and return result
Set Debug(*)	Debug settings
Echo	Display input lines
Error_trace	Display origin of errors
Enable_substitution	Enable automatic parameter substitution
Disable_substitution	Disable automatic parameter substitution
Status Aliases	Show all macros, variables and procedures
Errors	Show all error messages that have occurred
Memory	Show allocated workspace
System	Perform system call
Table	Table commands
Terminal	Output to terminal on/off
Time	Display CPU time used
Undefine	Macro un-definition
Verify	Evaluate with error output only
While	Conditional loop

Built-in commands and application-dependent commands are indistinguishable for the user. Commands listed with a (*) are only known internally, and are hidden by the `Commands` command. Not all keywords are listed here; see the corresponding sections for `Set debug` and `Table`.

17.30. Special characters

(blank)	Separates a command block into “words”
;	Command block separator
%	Comment; internal variable
\	Continuation character, escape character
{	Opening bracket for command words
}	Closing bracket for command words
(	Opening bracket in expressions or procedure definitions
)	Closing bracket in expressions or procedure definitions
+	Integer addition
−	Integer subtraction
*	Integer multiplication
=	Assignment statement; also used in conditions
!	Used as “not” in conditions
>	Used as “larger than” in conditions
<	Used as “smaller than” in conditions
&	Used in logical AND
|	Used in logical OR, as separator in proc. defs. and in string concat.
∼	Used in string conditions and assignment
,	Used as separator in procedure definitions
@	Used internally as quotes
#	Used as dummy variable in tables

18. Kac defaults and parameters

18.1. Parameters

Name	In defaults file
Autostore	YES
Invariant statistics	NO
Boundary_search info_level	NO
Boundary_search use_all_sets	NO
Boundary_search capacity	NO
Orientifold	NO
Selection extension	NO
Selection symmetric	NO
Selection automorphism	NO
Create show_info	NO
Invariant show_info	NO
Tadpole show_spectrum	NO
Fusion sumcheck	NO
Selection max_display	YES
S_matrix maxsize	YES
Trace_resolution	NO
Super	NO
Graded	NO
Dimension	NO
High_precision	YES
Maxrank	YES
Tadpole info_level	YES
interactive	NO
error_log	YES
log	YES
error_trace	NO
math	NO
echo	NO

18.2. The defaults file

18.3. Datatypes

The parameters of commands can have one of the following types

Name	description	Auto conj.	Auto range
Ishibashi	Ishibashi label	NO	YES
bound	Boundary label	NO	YES
dim	Space-time dimension in superstring mode	NO	NO
rank	Rank of horizontal algebra	NO	NO
radius	Circle radius (defined as number of primaries)	NO	NO
level	Level (central charge)	NO	NO
N_susy	Number of supersymmetries	NO	NO
field	Primary field label	YES	YES
fprim	CFT primary in tensor factor	NO	NO
fcur	Simple current in tensor factor	NO	NO
current	Simple current	YES	YES
id	String	NO	NO
switch	ON or OFF	NO	NO
sign	Sign (+1 or -1)	NO	NO
non_neg_int	Non-negative integer	NO	NO
pos_int	Positive integer	NO	NO
int	Integer	NO	NO

Type "id" is the default, if no type is shown in the `Help` output. If a parameter has the "Auto-conjugation" property, as indicated in column 3, then it can be conjugated by preceding it by a − sign. For example `Fusion 1 -2` computes the fusion between field 1 and the (charge)-conjugate of 2. If the "Auto-range" property is available, one can perform a command for a range of parameter values by typing either "`min:max`" or "`*`" instead of the parameter value. For example `Display 3:8` displays all fields from 3 up to and including 8; `Get Conjugate *` displays all conjugates.

18.4. Limitations

19. Glossary

This is a complete list of all commands in alfabetic order. It is a slightly edited version of the entire internal "help" system of the program.

```
    > Account Heterotic
```
INTERNAL
```
    > Algebra A [rank r][level k]
    > Algebra At [rank r][level k]
    > Algebra B [rank r][level k]
    > Algebra Bc [rank r][level k]
    > Algebra Bt [rank r][level k]
    > Algebra C [rank r][level k]
    > Algebra Cft [CFT_name]
```
Load cft "CFT_name".

```
> Algebra Ct [rank r][level k]
> Algebra D [rank r][level k]
> Algebra E [rank r][level k]
> Algebra F [rank r][level k]
> Algebra Fcft [CFT_name][pos_int current]
```

Load fixed point cft of cft "CFT_name" and current "current".

```
> Algebra Ft [rank r][level k]
> Algebra G [rank r][level k]
> Algebra Gt [rank r][level k]
> Algebra Minimal [N_susy N][level k]
```

minimal model "N" supersymmetries and level "k".

```
> Algebra Orbifold [pos_int N]
```

Z_2 orbifold of U_2N (CFT with 2"N" primaries).

```
> Algebra Test [pos_int k]
```

minimal N=2 level "k".

```
> Algebra U [radius R]
> Annulus (field i)(bound a)(bound b)
```

Compute Annulus coefficient $A^i_{ab} = A(i, a, b)$.

```
> Apropos [string]
```

List all commands or keywords containing "string" as a substring or having "string" in the command description.

```
> Assemble
```

Try to assemble a complete set of boundaries using previous results of "try boundary".

```
> Benchmark Fusion
```

Display time and memory allocation for full fusion computation.

```
> Break (pos_int count)
```

Stop execution of current command line and go up "count" steps in the command line hierarchy. Without parameters: return to terminal/file level.

```
> Browse Algebras
```

Show names of algebras in storage.

```
> Browse Boundary (bound a)
```

Show boundary labels. Without arguments: All boundaries are shown. With argument "a": Boundary "a" is shown.

```
> Browse Bulk_invariant (name)
```

Show current bulk invariant, or bulk invariant "name".

```
> Browse Cartan
```

Show Cartan matrix and its inverse.

```
> Browse Central_charge
```

Show central charge.

```
> Browse Cft
```

Display several CFT properties.

```
> Browse Chiral_algebra
```

Display chiral algebra.

```
> Browse Conjugation Boundary [bound a]
```

Show boundary conjugate of a.

```
> Browse Cp
```

Show Chan-Paton labels and gauge groups

```
> Browse Datatypes
```

Display data types.

```
> Browse Decomposition [field i](col_1)...(col_n)
```

Display decomposition of representation “i”.

```
> Browse Eta
```

Show charge conjugation twist and its field dependence.

```
> Browse Etatwists
```

Compare current twists and eta twists.

```
> Browse Generic [int nr]
```

Display spectrum of generic standard model braneconfiguration “nr” from “Models.list”

```
> Browse Groundstate [field i](col_1)...(col_n)
```

Display decomposition of ground state of representation “i”.

```
> Browse Heterotic
```

Display heterotic string spectrum (using the bosonic string map). Works only in “super” mode and if the conditions for a superstring interpretation are satisfied (see “help set super” for more details).

```
> Browse Heterotic Decomposition
```

Same as “browse heterotic” but in addition to the totals, the decomposition of the spectrum in terms of terms in the partition function is shown.

```
> Browse Heterotic Flipped
```

Same as “browse heterotic” but with left and right sector reversed.

```
> Browse Heterotic Flipped Decomposition
```

Combines keywords “flipped” and “decomposition”.

```
> Browse Integral_spin
```

Display all integral spin fields.

```
> Browse Invariants
```

List names of all stored invariants.

```
> Browse Ishibashi (Ishibashi a)
```

Show Ishibashi labels. Without arguments: All Ishibashi labels are shown. With argument “a”: Ishibashi label “a” is shown.

```
> Browse Lie
```

Display horizontal Lie algebra properties.

```
> Browse Lorentz [dim D]
```

Explain notation for Lorentz representations appearing in D-dimensional super multiplets.

```
> Browse Metric (Ishibashi i)
```

Show the diagonal Ishibahsi metric “i”.

```
> Browse Orbit Boundary [bound a]
```

Show boundary orbit.

```
> Browse Orbit Ishibashi [Ishibashi m]
```

Show Ishibashi orbit.

```
> Browse Orientifold
```

Show current orientifold properties.

```
> Browse Permutations
```

Display permutation/outer automorphism symmetries among simple currents

```
> Browse Signs
```

Show phase shifts used in reflection coefficients and signs used in crosscap computation. Phase shifts x: Refl. coeffs are $\exp[-i\pi x/4]S^J$ Crosscap signs e: Crosscaps are $\sum_J e(J)P_{Jm}$ Here S^J is the fixed point resolution matrix: $S^J = \exp[-i\pi m/4]O^J$, the orbit Lie algebra matrix. the phase "m" is listed under "FCFT shift." A "*" indicates that there are no fixed points.

```
> Browse Simple_currents
```

Show simple current data.

```
> Browse Stabilizer [field lbl]
```

Show information on the stabilizer of field "lbl".

```
> Browse Susy (dim D)(L_susy)(R_susy)(m1)(m2)(int sym)
```

Show spin content of susy representation.

```
> Browse Tadpole Bounds
```

Display tadpole upper and lower boundaries.

```
> Browse Tadpole Equations (non_neg_int nr.)
```

Display tadpole cancellation conditions for massless fields With parameter: display eqn "nr.".

```
> Browse Tadpole Kernel
```

Display tadpole kernel vectors.

```
> Browse Twists
```

Show simple current twists.

```
> Browse Type-i
```

Display type-I closed string spectrum (using the bosonic string map) Works only in "super" mode and if the conditions for a superstring interpretation are satisfied (see "help set super" for more details).

```
> Browse Type-ii
```

Display type-II string spectrum (using the bosonic string map) Works only in "super" mode and if the conditions for a superstring interpretation are satisfied (see "help set super" for more details).

```
> Browse Type-ii Decomposition
```

Same as "browse type-II" but in addition to the totals, the decomposition of the spectrum in terms of terms in the partition function is shown.

```
> Browse Type-ii Flipped
```

Same as "browse type-II" but with relative chirality reversed.

```
> Browse Type-ii Flipped Decomposition
```

Combines keywords "flipped" and "decomposition".

```
> Build Cy [int code]
```

Build minimal model tensor product identified by "code" For example 441010 yields the supersymmetrized tensor product of two k=4 and two k=10 minimal models.

```
> Check Algebra (int mult1)...(int mult2)
```

Check if a potential chiral algebra has non-negative projections on all fields. The parameters are the multiplicities of the integer spin fields in the order given by "Browse integral_spin".

`> Check Annulus [field i][bound a][bound b]`

Check that the direct computation of the annulus gives the same answer as the orbit method.

`> Check Branching [pos_int G_type][pos_int G_rank][pos_int H_type][pos_int H_rank]`

Check the internal branching tables.

`> Check Classes`

Check that conjugacy class definitions match simple current charges for WZW models.

`> Check Closed (pos_int max_err)`

Check positivity and integrality of Torus + Klein bottle. ("max_err" is the maximum number of error messages; Defaults to no limit).

`> Check Compare_algebras [name1][name2]`

Compare algebras "name1" and "name2".

`> Check Completeness`

Check that number of Ishibashi labels equals number of boundaries.

`> Check Conjugates`

Check $\hat{S2}$=C.

`> Check Cp (entry_1)...(entry_n)`

Check if CP labels satisfy the tadpole conditions. Entry_i is of the form label:value. Here "label" refers to the Chan-Paton label "n_label", and "value" is the integer value assigned to it. The labelling starts with n_0, and unassigned labels are set to zero. For example "check tadpole 0:1 17:2" sets n_0 to 1, n_17 to 2, and all others to zero.

`> Check Eta`

Check Y(f,0,i)/Y(f,0,0)=exp(i pi h_J) eta(J).

`> Check Fp_conj`

Check $(\hat{SJ})\hat{2}$=eta$\hat{J}$ C.

`> Check Full_open (pos_int max_err)`

Check all open string consistency conditions ("max_err" is the maximum number of error messages; Defaults to no limit).

`> Check Fusion [field i][field j][field k]`

Check integrality of the naive stabilizer-based fusion coefficients

`> Check Homomorphism (field lbl1)...(field lbln)`

Check homomorphism generated by lbl1...lbln.

`> Check Hopf`

Check rational Hopf algebra relations.

`> Check Invariant [invariant]`

Check the modular invariant "invariant" Syntax of "invariant" (example): n*(a+b+c)*(d+m*e+f)+(g+
where n,m,k are integers and a-i valid labels.

`> Check Klein (pos_int max_err)`

Check positivity of $N_{ijk}K_iK_jK_k$ ("max_err" is the maximum number of error messages; Defaults to no limit).

`> Check Modular`

Check unitarity and $ST\hat{3}$=$S\hat{2}$.

`> Check Nimrep [field i][field j][bound a][bound b]`

Check NIMrep condition: $\sum_c A^c_{ia}A^b_{jc} = \sum_k N^k_{ij}A^b_{ka}$

`> Check Open (pos_int max_err)`

Check positivity and integrality of Annulus + Moebius strip ("max_err" is the maximum number of error messages; Defaults to no limit).

```
> Check Orbits
```

Check that the algebraic simple current action agrees with the fusion rules.

```
> Check Oriented_annulus
```

Check positivity and integrality of oriented annulus.

```
> Check P-gauss [field i][field j]
```

Check gauss-sum P-matrix formula for labels "i" and "j".

```
> Check Pmat
```

Check pmatrix efficiency

```
> Check Reflection
```

Check that the direct computation of the reflection coefficients gives the same answer as the orbit method.

```
> Check Simple [field a]
```

Check if "a" is a simple current.

```
> Check Smlist
```

UNDOCUMENTED

```
> Check Symmetry
```

Check symmetry of S.

```
> Check Tadpole Parameters (entry_1)...(entry_n)
```

Check solution to the tadpole conditions that is obtained by setting the parameters to selected values Entry_i is of the form label:value. Here "label" refers to the parameter "N_label", and "value" is the integer value assigned to it.

```
> Check Tadpole Solution (entry_1)...(entry_n)
```

Check solution to the tadpole conditions. Syntax as for "check CP", but the labels refer to "raw" variables in the tadpole equations (as shown by the command "browse tadpole equations"). *** These labels are NOT equal to the CP-labels.***

```
> Check Trace_formula
```

Check the boundary trace formula for the annulus

```
> Check Twist_prod
```

Check product rule in second argument of F(a,K,J).

```
> Check Twists
```

Compare current twists and eta twists (only shows errors).

```
> Check Unimrep Klein [field i][field j]
```

Check U-NIMrep condition for moebius/klein: $\sum_b M_i^b M_{jb} = \sum_k Y_{ij}^k K_k$

```
> Check Unimrep Moebius [field i][field j][bound a]
```

Check U-NIMrep condition for moebius/annulus: $\sum_b A_{ia}^b M_{ib} = \sum_k Y_{ij}^k M_{ka}$

```
> Check Unitarity (field a)(field b)
```

Check unitarity.

```
> Collect Heterotic
```

collect heterotic string spectra and number of boundaries Works only in "super" mode and if the conditions for a superstring interpretation are satisfied (see "help set super" for more details).

```
> Collect Orientifolds
```

load all distinct FHSSW orientifolds

`> Collect Simple_current_invariants`

Load all simple current invariants that respect the selection criteria currently in effect.

`> Commands`

List all commands.

`> Commands All`

List all commands and keywords.

`> Commands Detail`

List all commands, keywords and descriptions.

`> Commands Tex`

List all commands, keywords and descriptions in TeX format.

`> Compute`

Compute spectrum

`> Compute Galois_orbits`

Compute all galois orbits.

`> Compute Gauss_sum (field x)`

[This command is only available for WZW models.] Compute Gauss sum: Sum_w [exp 2pi i (w-x/2)$\hat{2}$/h]. Here x is a weight, and the summation is over all weights w in the weight-lattice modulo h times the co-root lattice, or in other words, over all primaries at level=h plus their horizontal Weyl images. The result depends only on the Weyl orbit of the Dynkin labels of x modulo 2. If no parameter x is specified the computation is done for all orbits. To display the orbits use the command "Compute Gauss_sum Orbits"

NORMALIZATION: a factor (ih/2)$\hat{}$rank/2 times the square root of the volume of weight over coroot lattice is omitted.

NOTE: h=k+g, where g is the dual Coxeter number. However, the computation is done for h equal to the "level".

`> Compute Gauss_sum Orbits (current i)`

Compute the complete Weyl orbits modulo 2 in conjugacy class"i" Without parameter: use conjugacy class 0.

`> Compute S_analytic (factor)(lbl_1)...(lbl_2r)`

Compute the Kac-Peterson formula as an analytic function on weight space. "factor" is a real factor multiplying the exponents in the Weyl sum. (factor=1 gives the KP-formula). "lbl_i" are analytically continued Dynkin labels, 2r in total, where "r" is the rank; "labl_i" may be any expression that evaluates to a real number.

`> Compute S_sum (field lbl1)...(field lbln)`

Compute sum of indicated rows of S.

`> Compute Scale_orbits`

Compute scale orbits.

`> Compute Simple_current_invariant (current J_1)(current J_2)...(current J_n)`

Compute all invariants for the subgroup generated by all currents under fusion. Without parameters: consider all subgroups.

`> Conjugate (name_1)(name_2)`

Without parameters: conjugate the current bulk invariant. With one parameter: conjugate

the named invariant and store the result under an automatically generated name. With two parameters: conjugate the invariant “name_1” and store the result as “name_2”.

```
> Continue
```

Do nothing.

```
> Create [CFT_name]
```

Create a new CFT.

```
> Current (fcur current_1)...(fcur current_n)
```

Add simple current combination.

```
> Define [string][replacement]
```

Macro definition for “string”.

```
> Dimension [field lbl]
```

Get dimension of ground state “lbl”.

```
> Display (field min)(field max)
```

Display spectrum (Only for field “min” or from “min” to “max”).

```
> Display Entry [col][field i]
```

Display entry in column “col” and row i of the spectrum table.

```
> Display Massless
```

Display fields contributing to the massless spectrum.

```
> Dynkin (int i1)...(int in)
```

Get the label for representation (i1,...,in) For untwisted affine Lie algebras i1,...,in are Dynkin labels, not including the last (extended) label. Hence “n” equals the rank For all other algebras the labels are those listed by the “display” command.

```
> Evaluate [expression]
```

Evaluate algebraic expression.

```
> Execute [expression]
```

Execute algebraic expression without printing output.

```
> Expand [field lbl]
```

Expand coset field into Dynkin labels.

```
> Extract [pos_int label][string]
```

Extract integer nr. “label” from “string”.(Count starts at 1) “string” is taken literally in command line use of “extract”. When “extract” is used as a function, e.g “extract(1,string)”, “string” is evaluated as an expression.

```
> File (fn)
```

Output to file; file name = fn.

```
> File Append (fn)
```

Append output to file; file name = fn.

```
> File Off
```

Close output file.

```
> File On (fn)
```

Output to file; file name = fn.

```
> Find [string]
```

List all commands or keywords containing “string” as a substring.

```
> Fix Cpmult (entry_1)...(entry_n)
```

fix Chan-Paton multiplicity. Entries have the form label:value.Here label is any valid bound-

ary label. When the tadpoleequations are solved these labels are kept fixed.For boundary conjugates the fixed values are set automaticaly.

```
> Float [expression]
```

Evaluate algebraic expression and convert to floating point number.

```
> For [command1][condition][command2][commandline]
```

Execute commandline while "condition" holds. before execution of "commandline","command1" is executed once. After each execution of "commandline" "command2" is executed.

```
> Free All
```

Remove all macros, procedures and variables.

```
> Free Global (variable)
```

Remove "variable" (global); without parameters: remove all global variables

```
> Free Local (variable)
```

Remove "variable" (local); without parameters: remove all local variables.

```
> Free Macros (macro)
```

Remove "macro"; without parameters: remove all macro definitions.

```
> Free Procedures (procedure)
```

Remove "procedure"; without parameters: remove all procedures.

```
> Free Variable (variable)
```

Remove "variable" (local and/or global); without parameters: remove all variables.

```
> Fusion (field i)(field j)(field k)
```

Compute fusion rules.

```
> G A [rank r][level k]
> G At [rank r][level k]
> G B [rank r][level k]
> G Bc [rank r][level k]
> G Bt [rank r][level k]
> G C [rank r][level k]
> G Cft [CFT_name]
```

Load cft "CFT_name".

```
> G Ct [rank r][level k]
> G D [rank r][level k]
> G E [rank r][level k]
> G F [rank r][level k]
> G Ft [rank r][level k]
> G G [rank r][level k]
> G Gt [rank r][level k]
> G Minimal [N_susy N][level k]
```

Minimal model with "N" Supersymmetries and level k.

```
> G Orbifold [pos_int N]
```

Z_2 orbifold of U_{2N}.

```
> G Test [pos_int k]
```

minimal N=2 level k

```
> G U [radius R]
> Get Bfusion [field i][bound a]
```

compute boundary fusion

```
> Get Charge [current J][field a]
```

Get the charge of “a” w.r.t. “J”.

```
> Get Class_alg [Ishibashi i][Ishibashi j][Ishibashi k]
```

Compute classifying algebra coefficients.

```
> Get Conjugate [field a]
```

Get the conjugate of field “a”.

```
> Get Crosscap (Ishibashi i)
```

Get the crosscap coefficient for Ishibashi field “i”.

```
> Get Fcft [pos_int current]
```

Load fixed point cft for current “current”.

```
> Get Fcft_shift [current J]
```

Get the shift parameter “m” that determines the phase between the fixed point resolution matrix $S\hat{J}$ and the orbit Lie algebra matrix $O\hat{J}$: $S\hat{J}$ = exp[-i pi m/4] $O\hat{J}$.

```
> Get Fp_fusion [current J][field i][field g][field h]
```

Compute fixed point fusion rules This command computes Sum_n S_in $O\hat{J}$_gn $O\hat{J}$_hn/S_0n where “g” and “h” are fixed points of “J” and the sum is over all fixed points. The matrix $O\hat{J}$ is the matrix S of the orbit lie algebra, which differs from the fixed point resolution matrix $S\hat{J}$ by a phase.

```
> Get G_schur [field p][field q]
```

Compute generalized Frobenius-Schur indicator of fields “p” and “q”.

```
> Get Nimrep (field i)(bound a)(bound b)
```

Compute NIMrep matrix $(A^i)^b_a$. Without arguments: all matrices. One argument: Matrix A^i. Two arguments: $(A^i)^b_a$ (all b).Three arguments: $(A^i)^b_a$.

```
> Get Order [current J][field a]
```

Get the order of the simple current “J” on “a”.

```
> Get Reflection (Ishibashi i)(bound a)
```

Get the reflection coefficient for Ishibashi field “i” on boundary “a”.

```
> Get S_fix [current J][field i][field j]
```

Compute the matrix element S^J_{ij} of the fixed point resolution matrix of current J.

```
> Get S_ola [current J][field i][field j]
```

Compute the matrix element O^J_{ij} of the orbit lie algebra matrix of current J.

```
> Get Schur (field a)
```

Compute Frobenius-Schur indicator of field “a”. Without arguments: all fields.

```
> Get Twist [field ][current K][current J]
```

Get the twist F(a,K,J).

```
> Get Weight [field a]
```

Get the weight of field “a” in rational form.

```
> Global [assignment]
```

Declare global variable local and assign a value.

```
> Grade [current J]
```

Add an NSR-type grading according to the charges of the current “J”. These charges must be (half)-integer.

```
> H A [rank r][level k]
```

```
> H At [rank r][level k]
> H B [rank r][level k]
> H Bc [rank r][level k]
> H Bt [rank r][level k]
> H C [rank r][level k]
> H Cft [CFT_name]
```
Load cft "CFT_name".
```
> H Ct [rank r][level k]
> H D [rank r][level k]
> H E [rank r][level k]
> H F [rank r][level k]
> H Ft [rank r][level k]
> H G [rank r][level k]
> H Gt [rank r][level k]
> H U [radius R]
> Help (command)(key_1)...(key_n)
```
General help or help for commands.
```
> Identify (fprim lbl_1)...(fprim lbl_n)
```
Find orbit in which primary combination occurs (if any).
```
> Identify Orientifold
```
Find equivalence class of the current orientifold.
```
> If [condition][true_commandline](false_commandline)
```
Execute one of the two commandlines depending on "condition".
```
> Indices (field lbl)
```
Compute indices (for rep. "lbl").
```
> Initialize
```
TEMPORARY
```
> Intersect [bound a][bound b]
```
Compute the oriented chiral intersection matrix of boundaries "a" and "b". This is defined as $I_{ab} = sum_(i, massless, lefthanded) A^i_{ab}$ Character(i) Also shown is the chirality, $I_{ab} - I_{a^c b^c}$ (Implemented only for 4-D CY compactifactions)
```
> Klein (field i)
```
Compute Klein bottle coefficient K^i.
```
> List (column_1)...(column_n)
```
Restore printing of all columns of spectrum tables with name that starts with "column_i"
```
> Load (filename)
```
Read command or procedure file.
```
> Localize Boundary [bound a]
```
Find location of D-brane (U(1) and orbifold only)
```
> Localize Crosscap
```
Find location of O-plane (U(1) and orbifold only)
```
> Make Bulk Exceptional
```
Regard the current bulk invariant as an exceptional invariant.
```
> Modelscan [filename]
```

Scan models in “filename” for tadpole solutions

```
> Modelscan2 [filename]
```

UNDOCUMENTED

```
> Moebius (field i)(bound a)
```

Compute Moebius strip coefficient $M_a^i = M(i, a)$.

```
> Multiply [name1][name2](name3)
```

Multiply bulk invariants “name1” and “name2” to yield “name3”. If not specified, “name3” is automatically assigned.

```
> Multiply Close
```

Close set of bulk invariants under multiplication.

```
> Multiply Close Display
```

Close set of bulk invariants under multiplication and display new items.

```
> New Column [name][expression]
```

Add new column to current output table.

```
> Orbit [current J][field i]
```

Compute J-orbit of i.

```
> Pfusion (field i)(field j)(field k)
```

Compute P-fusion rules.

```
> Pmatrix (field lbl1)(field lbl2)
```

Compute matrix elements of P.

```
> Print [string]
```

Print string.

```
> Procedure [name][commandline]
```

Define a procedure.

```
> Prompt
```

read further input from terminal

```
> Quit
```

Exit program.

```
> Repeat [pos_int count][commandline]
```

Execute “commandline” “count” times.

```
> Reset Boundary_search
```

Reset exceptional boundary coefficients to start new search.

```
> Reset Bulk
```

Reset bulk invariant and Klein bottle.

```
> Reset Currents
```

Reset currents.

```
> Reset Errors
```

Restore all error messages previously disabled with “ignore”.

```
> Reset Invariants
```

Reset modular invariant storage.

```
> Reset Orientifold
```

Resets all crosscap_signs.

```
> Reset Selection
```

Reset criteria for modular invariant selection.

`> Reset Tadpole`
Reset tadpole conditions.
`> Reset Tensor`
Reset tensor mode.
`> Return [expression]`
Execute algebraic expression and return value, but do not print it.
`> S (field lbl1)(field lbl2)`
Compute matrix elements of S.
`> Save Preferences`
Save preferences.
`> Search Boundaries (pos_int Maxloop)`
Attempt to determine boundary coefficients given a bulk invariant. "Maxloop" is the maximal loop size. "Methods" chooses between full loop search (value 0) and the shortcut (value 1).
`> Search Crosscap`
Attempt to find crosscap coefficients given a complete set of boundaries.
`> Search Galois_invariants`
Search for galois invariants.
`> Search Klein_bottle`
Search for Klein bottle signs consistent with Ishibashi states.
`> Search Klein_bottle All`
Search for Klein bottle signs consistent with Ishibashi states and satisfying the trace formula and the Klein bottle constraint.
`> Search Klein_bottle Kbc`
Search for Klein bottle signs consistent with Ishibashi states and the Klein bottle constraint.
`> Search Klein_bottle Trace`
Search for Klein bottle signs consistent with Ishibashi states and satisfying the trace formula.
`> Search Mipf`
search all Modular Invariant Partition Functions that can be obtained combining all available methods.
`> Search Mipf Automorphism`
search for Modular Invariant Partition Functions of automorphism type.
`> Search Mipf Built_in`
list Modular Invariant Partition Functions that are explicitly built in (currently only for c=1 orbifolds).
`> Search Mipf Extension`
search for Modular Invariant Partition Functions of extension type (non-exhaustive algorithm).
`> Search Mipf Simple_current`
list all Modular Invariant Partition Functions of simple current type.
`> Search Moebius`
Attempt to find a Moebius and Klein bottle given a complete set of boundaries.
`> Search Nimrep (pos_int Maxloop)`
Attempt to determine NIMrep given a bulk invariant. "Maxloop" is the maximal loop size.

`> Select Boundary_set [non_neg_int nr]`

Select boundary set "nr" from the list. obtained with "search boundaries".

`> Select Conjugation_invariant`

Set bulk partition function to the conjugation invariant.

`> Select Diagonal_invariant`

Set bulk partition function to the diagonal invariant.

`> Select Invariant [invariant]`

Set bulk partition function to an explicitly given modular invariant. Format as for "check invariant". If "invariant" starts with a character it is interpreted as a name of a stored invariant.

`> Select Klein (entry_1)...(entry_n)`

Set exceptional Klein bottles. Entries are of the form "lbl:value". Missing entries are set equal Z_{ii}.

`> Select Klein Nr [non_neg_int nr]`

Select exceptional Klein bottle "nr" from the list. obtained with "search klein_bottle".

`> Select Orientifold [non_neg_int i]`

Select one of the allowed orientifolds

`> Select Simple_current_invariant (current J_1)(current J_2)...(current J_n)`

Compute all invariants for the subgroup generated by all currents under fusion, and prompt for selection. Without arguments: consider all subgroups.

`> Set Autoselect [pos_int choice]`

Set modular invariant selection to automatic mode, with automatic selection of invariant "choice".

`> Set Autoselect Off`

Set modular invariant selection to interactive mode.

`> Set Autostore (switch ON/OFF)`

Switch Automatic storage of modular invariants.

`> Set Boundary_search Capacity [pos_int value]`

Set storage space scaling factor used in boundary search.

`> Set Boundary_search Info_level [non_neg_int value]`

Set amount of information printed while searching for boundaries and crosscaps.

`> Set Boundary_search Use_all_sets (switch ON/OFF)`

Switch usage of all boundary sets versus maximal size ones only.

`> Set Create Show_info (switch ON/OFF)`

Switch display of information during CFT creation.

`> Set Crosscap_signs (sign s_1)...(sign s_n)`

Set signs used in crosscap formula. There is one sign choice for each cyclic factor in the simple current group that defines the bulk invariant. Signs for odd cyclic factors are ignored. Missing signs are set to 1.

`> Set Debug Alloc_trace Off`

Stop showing all (re-)allocations.

`> Set Debug Alloc_trace On`

Show all (re-)allocations.

`> Set Debug Level [pos_int level]`

Set debug level to “level” (maximum value 4).

```
> Set Debug Off
```

Switch debug off.

```
> Set Dimension [dim value]
```

Set space time dimension for superstring mode.

```
> Set Disable_sub [command](key_1)...(key_n)
```

Disable automatic substitution for parameters of “command key_1 ... key_n”.

```
> Set Echo (switch ON/OFF)
```

Switch command echoing

```
> Set Enable_sub [command](key_1)...(key_n)
```

Enable automatic substitution for parameters of “command key_1 ... key_n”.

```
> Set Error_log (switch ON/OFF)
```

Switch error log file use

```
> Set Error_trace (switch ON/OFF)
```

Switch error trace

```
> Set Fusion Sumcheck (switch ON/OFF)
```

Switch dimension sum rule check in fusion.

```
> Set Graded (switch ON/OFF)
```

Switch graded/ungraded tensor product evaluation.

```
> Set High_precision (switch ON/OFF)
```

Switch high or low precision in modular data display.

```
> Set Init_debug (switch ON/OFF)
```

Switch display initialization information

```
> Set Interactive (switch ON/OFF)
```

Switch terminal input

```
> Set Invariant Show_info (switch ON/OFF)
```

Switch display of information during simple current invariant computation.

```
> Set Invariant Statistics (switch ON/OFF)
```

Switch display of statistics of bulk invariants.

```
> Set Kleinbottle [current J]
```

Specify simple current “J” to be used to modify Klein bottle projection.

```
> Set Log (switch ON/OFF)
```

Switch log file use

```
> Set Loop_command [commandline]
```

Set “commandline” to be applied to each modular invariant. Choose “commandline” equal to “off” to turn off.

```
> Set Math (switch ON/OFF)
```

Switch math mode

```
> Set Maxrank [non_neg_int value]
```

Set maximal rank.

```
> Set Moebius Security_level [non_neg_int value]
```

Set Level of reliability required for solving moebius polynomial equations.

```
> Set Nimrep Max_row [non_neg_int value]
```

Set maximal number of NIMrep row candidates

`> Set Orientifold (switch ON/OFF)`
Switch requirement of Crosscap/Moebius/Klein consistency during boundary search.
`> Set Outputdir [string]`
Directory for output files
`> Set Outputfilepostfix [string]`
postfix for output file name
`> Set Outputfileprefix [string]`
prefix for output file name
`> Set Reject Info (switch ON/OFF)`
Switch display of permutation rejection criteria
`> Set Reject Permutations (switch ON/OFF)`
Switch automatic suppression of invariants related by permutations and outer automorphisms
`> Set S_matrix Maxsize [non_neg_int value]`
Set maximal size for S_matrix storage.
`> Set Selection Automorphism (pair_1)(pair_2)...(pair_n)`
Require simple current invariant to be of automorphism type. Parameters of the form i:j require pairing of field "i" with field "j" Turn off with "set selection automorphism off."
`> Set Selection Extension (switch ON/OFF)`
Switch restriction to extension invariants during modular invariant selection
`> Set Selection Left (current J_1)(current J_2)...(current J_n)`
Require presence of indicated currents in the left chiral algebra
`> Set Selection Max_display [non_neg_int value]`
Set Maximal number of orbits shown during selection of simple current invariants.
`> Set Selection Right (current J_1)(current J_2)...(current J_n)`
Require presence of indicated currents in the right chiral algebra
`> Set Selection Symmetric (switch ON/OFF)`
Switch restriction to symmetric invariants during modular invariant selection
`> Set Sm_search Check_moebius (switch ON/OFF)`
Switch check Moebius chirality for (anti)-symmetric tensors
`> Set Sm_search Global Show_all (switch ON/OFF)`
Switch Display all independent global anomalies
`> Set Sm_search Global Show_new (switch ON/OFF)`
Switch Display all new independent global anomalies
`> Set Sm_search New_search (switch ON/OFF)`
Switch updating of existing msol/stat files
`> Set Sm_search Nfamily [pos_int value]`
Set Number of families searched for in "Search Standard_model"
`> Set Sm_search No_abort (switch ON/OFF)`
Switch fast abort during standard model search
`> Set Sm_search No_frac (switch ON/OFF)`
Switch Require absence of fractionally charged massless states
`> Set Sm_search Pati-salam (switch ON/OFF)`
Switch Include Pati-Salam models in search

```
> Set Sm_search Print_numbers (switch ON/OFF)
```
Switch Print invariant numbers in output
```
> Set Sm_search Reject_massive_y (switch ON/OFF)
```
Switch Reject triplets with massive Y-bosons
```
> Set Sm_search Reject_repeated [non_neg_int value]
```
Set Reject brane configurations similar to previous ones Values: 0: no skip; 1: Skip previous solutions 2: skip previous occurrence
```
> Set Sm_search Zero-tension (switch ON/OFF)
```
Switch Include zero tension orientifolds in search
```
> Set Store_s (switch ON/OFF)
```
Switch storage of orbit-orbit S-matrix elements for single algebras
```
> Set Super (switch ON/OFF)
```
Switch superstring interpretation of spectra.
```
> Set Tadpole Check_solution (switch ON/OFF)
```
Switch automatic equation check for tadpole solutions.
```
> Set Tadpole Info_level [non_neg_int value]
```
Set amount of information printed while solving tadpole conditions.
```
> Set Tadpole Remove_identical (switch ON/OFF)
```
Switch remove variables with identical tadpole coefficients
```
> Set Tadpole Show_solution (switch ON/OFF)
```
Switch automatic display of CP multiplicities for each solution.
```
> Set Tadpole Show_spectrum (switch ON/OFF)
```
Switch automatic spectrum computation for tadpole solutions.
```
> Set Tadpole Simplest_only (switch ON/OFF)
```
Switch Only find simplest Chan-Paton groups
```
> Set Tadpole Single_solution (switch ON/OFF)
```
Switch interrupt loop search after a single solution is found
```
> Set Tadpole Unitary_groups_only (switch ON/OFF)
```
Switch Allow only unitary CP groups
```
> Set Trace_resolution (switch ON/OFF)
```
Switch display of FCFT contributions to resolved S If this switch is ON, the contributions to S_ab are shown for each current J that fixes a and b.
```
> Set Use_orbits [non_neg_int value]
```
Set level of orbit usage for fusion and annulus computations
```
> Show Autostore
```
Show setting of Automatic storage of modular invariants.
```
> Show Boundary_search Capacity
```
Show storage space scaling factor used in boundary search.
```
> Show Boundary_search Info_level
```
Show amount of information printed while searching for boundaries and crosscaps.
```
> Show Boundary_search Use_all_sets
```
Show setting of usage of all boundary sets versus maximal size ones only.
```
> Show Create Show_info
```
Show setting of display of information during CFT creation.

`> Show Default_file`
show filename of default file
`> Show Dimension`
Show space time dimension for superstring mode.
`> Show Echo`
Show setting of command echoing
`> Show Error_log`
Show setting of error log file use
`> Show Error_trace`
Show setting of error trace
`> Show Fusion Sumcheck`
Show setting of dimension sum rule check in fusion.
`> Show Graded`
Show setting of graded/ungraded tensor product evaluation.
`> Show High_precision`
Show setting of high or low precision in modular data display.
`> Show Init_debug`
Show setting of display initialization information
`> Show Interactive`
Show setting of terminal input
`> Show Invariant Show_info`
Show setting of display of information during simple current invariant computation.
`> Show Invariant Statistics`
Show setting of display of statistics of bulk invariants.
`> Show Log`
Show setting of log file use
`> Show Math`
Show setting of math mode
`> Show Maxrank`
Show maximal rank.
`> Show Moebius Security_level`
Show Level of reliability required for solving moebius polynomial equations.
`> Show Nimrep Max_row`
Show maximal number of NIMrep row candidates
`> Show Orientifold`
Show setting of requirement of Crosscap/Moebius/Klein consistency during boundary search.
`> Show Outputdir`
Show Directory for output files
`> Show Outputfilepostfix`
Show postfix for output file name
`> Show Outputfileprefix`
Show prefix for output file name
`> Show Reject Info`
Show setting of display of permutation rejection criteria

`> Show Reject Permutations`
Show setting of automatic suppression of invariants related by permutations and outer automorphisms
`> Show S_matrix Maxsize`
Show maximal size for S_matrix storage.
`> Show Selection Criteria`
Show current settings of criteria used for invariant selection.
`> Show Selection Extension`
Show setting of restriction to extension invariants during modular invariant selection
`> Show Selection Max_display`
Show Maximal number of orbits shown during selection of simple current invariants.
`> Show Selection Symmetric`
Show setting of restriction to symmetric invariants during modular invariant selection
`> Show Sm_search Check_moebius`
Show setting of check Moebius chirality for (anti)-symmetric tensors
`> Show Sm_search Global Show_all`
Show setting of Display all independent global anomalies
`> Show Sm_search Global Show_new`
Show setting of Display all new independent global anomalies
`> Show Sm_search New_search`
Show setting of updating of existing msol/stat files
`> Show Sm_search Nfamily`
Show Number of families searched for in "Search Standard_model"
`> Show Sm_search No_abort`
Show setting of fast abort during standard model search
`> Show Sm_search No_frac`
Show setting of Require absence of fractionally charged massless states
`> Show Sm_search Pati-salam`
Show setting of Include Pati-Salam models in search
`> Show Sm_search Print_numbers`
Show setting of Print invariant numbers in output
`> Show Sm_search Reject_massive_y`
Show setting of Reject triplets with massive Y-bosons
`> Show Sm_search Reject_repeated`
Show Reject brane configurations similar to previous ones Values: 0: no skip; 1: Skip previous solutions 2: skip previous occurrence
`> Show Sm_search Zero-tension`
Show setting of Include zero tension orientifolds in search
`> Show Store_s`
Show setting of storage of orbit-orbit S-matrix elements for single algebras
`> Show Super`
Show setting of superstring interpretation of spectra.
`> Show Tadpole Check_solution`
Show setting of automatic equation check for tadpole solutions.

`> Show Tadpole Info_level`
Show amount of information printed while solving tadpole conditions.
`> Show Tadpole Remove_identical`
Show setting of remove variables with identical tadpole coefficients
`> Show Tadpole Show_solution`
Show setting of automatic display of CP multiplicities for each solution.
`> Show Tadpole Show_spectrum`
Show setting of automatic spectrum computation for tadpole solutions.
`> Show Tadpole Simplest_only`
Show setting of Only find simplest Chan-Paton groups
`> Show Tadpole Single_solution`
Show setting of interrupt loop search after a single solution is found
`> Show Tadpole Unitary_groups_only`
Show setting of Allow only unitary CP groups
`> Show Trace_resolution`
Show setting of display of FCFT contributions to resolved S If this switch is ON, the contributions to S_ab are shown for each current J that fixes a and b.
`> Show Use_orbits`
Show level of orbit usage for fusion and annulus computations
`> Simple_fusion [current J](field j)`
Compute simple current fusion rules.
`> Single`
Switch to single algebra mode; Display presently available algebra
`> Solve Tadpole (switch parm)`
Attempt to solve tadpole conditions. parm=ON: find parametrized solution (default). parm=OFF: find all solutions.
`> Solve Tadpole Bounds (switch parm)(pos_int interrupt)`
Attempt to solve tadpole conditions by reducing bounds parm=ON: find parametrized solution (default). parm=OFF: find all solutions. Pause after "interrupt" seconds (default 60).
`> Solve Tadpole Kernel (switch parm)(pos_int max_loop)`
Attempt to solve tadpole conditions by determining kernel parm=ON: find parametrized solution (default). parm=OFF: find all solutions. Give up after "max_loop" candidates per CP label (default 10000).
`> Solve Tadpole Loops (switch parm)(pos_int interrupt)`
Attempt to solve tadpole conditions using diophantine loops parm=ON: find parametrized solution (default). parm=OFF: find all solutions. Pause after "interrupt" seconds (default 60).
`> Solve Tadpole Simple`
Attempt to solve tadpole conditions using only simple current boundaries.
`> Status`
Show mode
`> Status Definitions`
Show defined macros, variables and procedures.
`> Status Errors`

Show error statistics.

```
> Status Memory
```

Show total memory use.

```
> Status Memory Detail
```

Show details of memory use for various parts of the program.

```
> Status Memory More_detail
```

Show memory use for arrays in the program.

```
> Status Parameters
```

Show values of internal parameters.

```
> Status Preferences
```

Show values of parameters in default data base.

```
> Store (name)
```

Store current bulk invariant as “name”. If no argument is present a name is automatically assigned.

```
> System [cmd...]
```

Perform Unix command [cmd...].

```
> Table Header [table]
```

Switch on header for table “table”.

```
> Table Info [table]
```

Info on table “table”.

```
> Table Left_adjust [table][column]
```

left adjust “column” in “table”.

```
> Table List
```

List output tables.

```
> Table Noheader [table]
```

Switch off header for table “table”.

```
> Table Restore [table][column]
```

Restore displaying of “column” in “table”.

```
> Table Right_adjust [table][column]
```

right adjust “column” in “table”.

```
> Table Set_space [table][column][pos_int s]
```

set spacing of “column” in “table” to “s”.

```
> Table Suppress [table][column]
```

Suppress displaying of “column” in “table”.

```
> Tensor
```

Switch to tensor mode; Display factors and simple currents

```
> Terminal
```

Switch for terminal output (without keyword equivalent to ‘Terminal On’).

```
> Terminal Off
```

Switch output to terminal off.

```
> Terminal On
```

Switch output to terminal on.

```
> Time
```

Print CPU time used so far.

`> Total Boundaries`

Return total number of boundary labels

`> Total Invariants`

Return total number of modular invariants stored as “Invariant_n”

`> Total Ishibashi`

Return total number of Ishibashi labels

`> Total Primaries`

Return total number of primaries

`> Total Simple_currents`

Return total number of simple currents

`> Transpose (name_1)(name_2)`

Without parameters: transpose the current bulk invariant. With one parameter: transpose the named invariant and store the result under an automatically generated name. With two parameters: transpose the invariant “name_1” and store the result as “name_2”.

`> Try Boundary [limits]`

Try a boundary “a”, specified by a set of annuli A^i_{aa} Input of the form 1,2,0:2,3 means $A^0 = 1$, $A^1 = 2$, A^2 ranges from 0 to 2 (inclusive), $A^3 = 3$.

`> Undefine [string]`

Remove macro definition for “string”.

`> Unlist (column_1)...(column_n)`

Suppress printing of all columns of spectrum tables with name that starts with “column_i”.

`> Verify [commandline]`

Execute “commandline” showing no output, but only errors.

`> Weyl Group`

Display decomposition of Weyl group elements in terms of simple roots.

`> Weyl Multiply [non_neg_int i][non_neg_int j]`

Multiply Weyl group elements “i” and “j”

`> Weyl Orbit [field i]`

Compute the complete Weyl orbit of field “i”. Same as “Weyl reflection i” except that identical weights are shown only once.

`> Weyl Order`

Display order of the Weyl group

`> Weyl Parity Orbits`

Compute the complete Weyl orbits modulo 2.

`> Weyl Reflection [field i](simple_root j)`

Compute the horizontal Weyl reflection of field “i” in the plane orthogonal to simple root “j”; Without second parameter: compute action of all Weyl group elements.

`> While [condition][commandline]`

Execute “commandline” while “condition” holds.

REFERENCES

[1] R. Dijkgraaf, C. Vafa, E. Verlinde and H. Verlinde, Commun. Math. Phys. 123 (1989) 16.