sented in the form like 'x(yz)' it means the keyword 'xyz' can be abbreviated to 'x'. '[something]' means 'something' is optional. | is used for listing alternatives. Slanted face, e.g., *variety* is used when it varies (a meta-variable) or is an expression of some language. For example, *modexp* is for module expressions and *term* is for terms (you should know what these are); others should easily be understood by their *name*s and/or from the context.

0.2 Starting CafeOBJ interpreter

To enter CafeOBJ, just type its name: `cafeobj`

'`cafeobj -help`' will show you a summary of command options.

0.3 Leaving CafeOBJ

`q(uit)` exits CafeOBJ.

0.4 Getting Little Help

Typing `?` at the top-level prompt will print out a list of whole top-level commands.

0.5 Escape

There would be a situation that you hit `return` expecting some feedback from the interpreter, but it does not respond. This occurs when the interpreter expects some more inputs from you thinking preceding input is not yet syntactically complete. If you encounter this situation, and you don't know what the interpreter expects, simply type in `esc`(escape key) and `return`, then it will immediately be back to you discarding preceding input and makes a fresh start. Alternatively, you can type in several `return` keys. This acts exactly the same as typing `esc` and `return`.

0.6 Rescue

Occasionally you may meet a strange prompt `CHAOS>>` after some error messages. This happens when the interpreter caused some internal errors and could not recover from it. Try typing `:q`, this may resume the session if you are lucky.

Sending interrupt signal (typing `C-c` from keyboard,

0.7 Setting Switches

Switches are for controlling the interpreter's behaviour in several manner. The general form of setting top-level switch is:

`set` *switch value*

In the following, the default value of a switch is shown underlined.

switchvaluewhat? *** – switches for rewriting trace wholeon—offtrace top-level rewrite step traceon—offtrace every rewrite step stepon—offstepwise rewriting process memoon—offenable term memoization clean memoon—offclean up term memo table before normalization statson—offshow statistics data after reduction rwt limit *number*maximum number of rewriting stop pattern[*term*] .stop rewriting when meets mel sorton—offcompute result sort with sort membership predicates reduce conditionson—offreduce conditional part in apply command verboseon—offset verbose mode exec traceon—offtrace concurrent execution exec limit*number*limit maximum number of concurrent execution exec normalizeon—offreduce term before and after each transition exec allon—offfind all solutions of `=(*)=>` *** – switches for system's behaviour include BOOLon—offimport BOOL implicitly incude RWLon—offimport RWL implicitly include FOPL-CLAUSEon—offimport FOPL-CLAUSE implicitly auto contexton—offchange current context in automatic auto reconstructon—offperform automatic reconstruction of modules if it is inconsistent reg signatureon—offregularize module signature in automatic check regularityon—offperform regularity check of signature in automatic check compatibilityon—offperform compatibility check of TRS in automatic check builtinon—offperform operator overloading check with built-in sorts select termon—offsystem selects a term from ambiguously parsed terms quieton—offsystem mostly says nothing – show/display options all axiomson—offprint all axioms in "`sh(ow)` ***modexp***" command show mode:cafeobjset syntax of printed modules —:chaosor views show var sortson—offprint variables with sorts print mode:normalset term priting form —:fancy —:tree —:s-expr *** – miscellaneous settings libpath*pathname*set file search path print depth*number*maximum depth of terms to be printed accept `=*=` proofon—offaccept system's automatic

The default value of *number* in `set lwt limit` command is 0 meaning no limit counter of rewriting is specified.

Omitting *term* in `set stop pattern` sets the stop pattern to empty, i.e., no term will match to the pattern.

0.8 Examining Values of Switches

show switchprint list of available switches with their values show switch *switch*print out the value of the specified *switch*

0.9 Setting Context

`select` *modexp*

This sets the context of the interpreter (current module) to the module specified by *modexp*. It must be written in single line. When you type in *modexp*, the '`;`<newline>' treated as a line continuation (that is, it is effectively ignored), so that you can type in multiple lines for long module expressions. Note that one or more blank characters are required before `;`.

0.10 Inspecting Module

`sh(ow)` and `desc(ribe)` commands print information on a module. In the sequel, we use a meta-variable *show* which stands for either `sh(ow)` or `desc(ribe)`. Most of the cases, giving `desc(ribe)` for *show* gives you more detailed information.

*show modexp*prints a module *modexp*. giving '." as *modexp* shows the current module *show* sorts [*modexp*]prints sorts of *modexp show* ops [*modexp*]prints operators of *modexp show* vars [*modexp*]prints variables of *modexp show* params [*modexp*]prints parameters of *modexp show* subs [*modexp*]prints direct submodules of *modexp show* sign [*modexp*]prints `sorts` and `ops` combined

modexp must be given in an one line. The same convention for long module expressions is used as that of `select` command (see Setting Context above.) If the optional [*modexp*] is omitted, it defaults to the current module. Optionally supplying `all` before `sorts`, `ops`, `axioms`, and `sign`, i.e., `desc all ops` for an instance) makes printed out information also include imported sorts, operators, etc. otherwise it only prints own constructs of the *modexp*.

*argname*prints information on the parameter *show* sub *n*prints information on the *n*th direct submodule *argname* can be given by position, not by name.

You can see the hierarchy of a module or a sort by the follwing `sh(ow)` commands: sh(ow) module tree *modexp*prints pictorial hierarchy of module. specifying `.` as *modexp* shows the hierarcy of the current module sh(ow) sort tree *sort*prints hierarchy of sort pictorially

0.11 Evaluating Terms

`red(uce)` [`in` *modexp* `:`] *term* `.`

`exec(ute)` [`in` *modexp* `:`] *term* `.`

`reduce` reduces a given term *term* in the term rewriting system derived from *modexp*. `execute` is similar to `reduce`, but it also considers axioms given by transition declarations. In both cases, omitted '`in` *modexp* `:`' defaults to the current module.

The result term of `reduce` and `execute` is bould to special variables `$$term` and `$$subterm` (see the next section).

0.12 Let Variables and Special Variables

`let` *let-variable* `=` *term* `.`

let-variable is an indentifier. Assuming the current module is set, `let` binds *let-variable* to the given term *term*. Once set, *let-variable* can be used wherever *term* can apper.

You can see the list of let bidings by:

`sh(ow) let`

There are two built-in special variables in the system: $$termbound to the result term of `reduce`, `execute`, `parse`, or `start` commands. $$subtermbound to the result of `choose` command

Let variales and special variables belongs to a context, i.e., each context has its own let variables and special variables.

0.13 Inspecting Terms

`parse` [`in` *modexp* `:`] *term* `.`

`parse` parses given term *term* in the module *modexp* (if omitted, parses in the current module) and prints the result. The result is bound to special variables `$$term` and `$$subterm`.

The following `sh(ow)` command assumes the current

0.14 Opening/Closing Module

open *modexp*opens module *modexp* closeclose the currently opening module Opening module can be modified, i.e., you can declare new sorts, operators, axioms. You can open only one module at a time.

0.15 Applying Rewrite Rules

Start The initial target (entire term) is set by `start` command.

`start` *term* .

This binds two special variables `$$term` and `$$subterm` to *term*.

Apply `apply` command applies actions to (subterm of) `$$term`.

`apply` *action range selection*

You specify an action by *action*, and it will be applied to the target (sub)term specified by *selection*.

range is either `within` or `at`: `within` means at or inside the (sub)term specified by the *selection*, and `at` means exactly at the *selection*.

Action *action* can be the followings: red(uction)reduce the selected term execexecute the selected term printprint the selected term *rule-spec*apply specified rule to the selected term

Rule-Spec *rule-spec* specifies the rule with possibly substitutions being applied, and given by

[`+ | -`][*modexp*].*rule-name* [*substitutions*]

The first optional '`+ | -`' specifies the direction of the rule; left to right(if `+` or omitted) or right to left (if `-`).

A rule itself is specified by '[*modexp*].*rule-name*]'. This means the rule with name *rule-name* of the module *modexp* (if omitted, the current module). *rule-name* is either a label of a rule or a number which shown by `sh(ow) rules` command (see Showing Available Rules below.)

substitution binds variables that apper in the selected rule before applying it. This has the form

`with` *variable* `=` *term* `,` ...

Showing Available Rules To see the list of the rewrite rules, use

`sh(ow)` [`all`] `rules`

selector { `of` *selector* } ⋯ selectordescription termthe entire term (`$$term`) topditto subtermselects `$$subterm` (*number* ⋯)selects by position `[` *number* `..` *number* `]`by range in flattened term structure { *number* `,` ⋯ }subset in flattened term structure

Step by Step Subterm Selection `choose` command selects a subterm of `$$subterm` and reset the `$$subterm` to the selected one.

`choose` *selector*

Matching Terms `match` *term_spec* `to` *pattern*

term_spec specifies the term to be matched with *pattern*: term_specdescription `term$$term` `top`ditto `subterm$$term` `it`ditto *term*ordinal term

patterndescription `[ all ][+ | -] rules`match with available rewrite rules *term*match with specified term

0.16 Stepper

If the switch `step` is set to `on`, invoking `reduce` or `execute` command runs into the term rewriting stepper. The stepper has its own command interpreter loop, where the following stepper commands are avilable:

?print out available commands. n(ext)go one step g(o) *number*go *number* step c(ontinue)continue rewriting without stepping q(uit)leave stepper continuing rewrite a(bort)abort rewriting r(rule)prints current rewrite rule s(ubst)prints substitution l(imit)prints rewrite limit counter p(attern)prints stop pattern stop [*term*]set (unset) stop pattern rwt [*number*]set (unset) rwrite limit counter You can also use families of `sh(ow)`(`desc(ribe)`) and `set` commands in stepper.

0.17 Reading In Files

input *file*read in CafeOBJ program from *file* provide *feature*provide the *feature* require *feature* [*file*]require *feature*

0.18 Save and Restore

save *files*save definitions of modules and views to *file* re-

unprotect *modexp*allow moudle to be redefined

0.20 Little Semantic Tools

check reg(ularity) [*modexp*]reports the result of regularity check of module check comat(ibility) [*modexp*reports the result of compatibility check of the module For both commands, omitted *modexp* will perform the check in the current module.

The following `check` command assumes the current module:

`check laziness` [*operator*]

This checks strictness of *operator*. If *operator* is omitted all of the operators declared in the current modules are checked.

0.21 TRAM Compiler Interface

`tram compile` [*modexp*]

This compiles module *modexp* to Term Rewriting Abstract Machine. If *modexp* is omitted, it defaults to the current module. *modexp* must be given in an line. You can supply multiple lines by using '`;`<newline>'.

To evaluate term in compiled module, use the following:

`tram exec` [`in` *modexp* :] *term*

Omitting '`in` *modexp* :' means the evaluation is performed in the current module. If the module *modexp* is not yet compiled, this compiles it implicitly, then perform the evaluation.

0.22 Miscellany

ls *pathname*list contents of directories cd *pathname*change working directory of the interpreter pwdprints working directory ! *command*fork shell *command* ev *lisp*evaluate lisp expression *lisp* printing the result evq *lisp*evaluate lisp expression *lisp*