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

0.2 Starting CafeOBJ interpreter
To enter CafeOBJ, just type its name: \texttt{cafeobj -help} will show you a summary of command options.

0.3 Leaving CafeOBJ
\texttt{q(uit)} exits CafeOBJ.

0.4 Getting Little Help
Typing \texttt{?} 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 \texttt{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 \texttt{esc} (escape key) and \texttt{return}, then it will immediately be back to you discarding preceding input and makes a fresh start. Alternatively, you can type in several \texttt{return} keys. This acts exactly the same as typing \texttt{esc} and \texttt{return}.

0.6 Rescue
Occasionally you may meet a strange prompt \texttt{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 \texttt{C-c} from keyboard, or if you are in Emacs, some key sequence specific to the mode you are in) forces the interpreter to break into underlying Lisp, and you will see the same prompt as the above. This might be useful when you feel the interpreter get confused. :q also works for returning to CafeOBJ interpreter from Lisp.

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

\texttt{set switch value}

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

```
switchvalue\texttt{what?} *** { switches for rewriting trace wholeon|otrace top-level rewrite step traceon|offtrace every rewrite step stepon|ostepwise 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\texttt{[term]} .stop rewriting when meets mel sorton|ocompute result sort with sort membership predicates reduce conditionon|oreduce conditional part in apply command verboseon|oset verbose mode exec traceon|offtrace concurrent execution exec limit number maximum number of concurrent execution exec normalizeon|offreduce term before and after each transition exec alloffind all solutions of \texttt{=*=>} *** - switches for system's behaviour include BOOLon|oimport BOOL implicitly include RWLon|oimport RWL implicitly include FOPL-CLAUSEon|oimport FOPL-CLAUSE implicitly auto contexton|ochange current context in automatic auto reconstron|operform automatic reconstruction of modules if it is inconsistent reg signatureon|oregularize module signature in automatic check regularityon|operform regularity check of signature in automatic check compatibilityon|operform compatibility check ofTRS in automatic check builtinon|operform operator overloaded check with built-in sorts select termon|osystem selects a term from ambiguously parsed terms quieton|osystem mostly says nothing - show/display options all axiomson|oprint all axioms in "\texttt{sh(ow)} modexp" command show mode:cafeobjset syntax of printed modules :-chao sor views show var sortson|offprint variables with sorts print mode:normalize term printing form :-fancy :-tree :-s-expr *** - miscellaneous settings libpath\texttt{pathname} set file search path print depth\texttt{number} maximum depth of terms to be printed accept \texttt{== proofon} -offace system's automatic
```

0.8 Setting Options
Options are for controlling the behaviour in several manner. The general form of setting top-level switch is:

\texttt{option value}

In the following, the default value of an option is shown underlined.

```
optionvalue\texttt{what?} *** { switches for rewriting trace wholeon|otrace top-level rewrite step traceon|offtrace every rewrite step stepon|ostepwise 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\texttt{[term]} .stop rewriting when meets mel sorton|ocompute result sort with sort membership predicates reduce conditionon|oreduce conditional part in apply command verboseon|oset verbose mode exec traceon|offtrace concurrent execution exec limit number maximum number of concurrent execution exec normalizeon|offreduce term before and after each transition exec alloffind all solutions of \texttt{=*=>} *** - switches for system's behaviour include BOOLon|oimport BOOL implicitly include RWLon|oimport RWL implicitly include FOPL-CLAUSEon|oimport FOPL-CLAUSE implicitly auto contexton|ochange current context in automatic auto reconstron|operform automatic reconstruction of modules if it is inconsistent reg signatureon|oregularize module signature in automatic check regularityon|operform regularity check of signature in automatic check compatibilityon|operform compatibility check ofTRS in automatic check builtinon|operform operator overloaded check with built-in sorts select termon|osystem selects a term from ambiguously parsed terms quieton|osystem mostly says nothing - show/display options all axiomson|oprint all axioms in "\texttt{sh(ow)} modexp" command show mode:cafeobjset syntax of printed modules :-chao sor views show var sortson|offprint variables with sorts print mode:normalize term printing form :-fancy :-tree :-s-expr *** - miscellaneous settings libpath\texttt{pathname} set file search path print depth\texttt{number} maximum depth of terms to be printed accept \texttt{== proofon} -offace system's automatic
```
0.8 Examining Values of Switches

show switch 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 ';' 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.


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.

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 bound 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 identifier. 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: $$term bound to the result term of reduce, execute, parse, or start commands. $$subterm bound 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 modexpopens module modexp close close 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.

\[ \text{start } \text{term} \]

This binds two special variables \( \$$term \) and \( \$$subterm \) to term.

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

\[ \text{apply action range selection} \]

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: reduce (action) reduce the selected term exec execute the selected term rule-spec applies specified rule to the selected term

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

\[ [+ | -] \text{[modexp] rule-name [substitutions]} \]

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 \([ \text{modexp} \text{ 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 appear in the selected rule before applying it. This has the form

\[ \text{with variable = term , } \]

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 available:

\[ ? \text{print out available commands. n(ext)go one step} \]

\[ g(o) number \] go number step

\[ c(ontinue) \] continue rewriting

\[ q uit(leave stepper) \] leave stepper

\[ r(ule)prints current rewrite rule \]

\[ s(ubst)prints substitution \]

\[ l(imit)prints rewrite limit counter \]

\[ p(attern)prints stop pattern \]

\[ r(wt) \text{set (unset) stop pattern} \]

\[ f(oo) \text{number} \text{stop pattern} \]

\[ s(ave) \text{image of} \]

\[ r(e) \text{save definitions of modules and views to} \]

\[ l(e) \text{read in CafeOBJ program from} \]

\[ r(e) \text{provide the feature require feature [file]} \]

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 filesave definitions of modules and views to file

require filerequire feature [file]
0.19 Protecting Your Modules

- `protect modexp` prevents the module from being redefined.
- `unprotect modexp` allows the module to be redefined.

0.20 Little Semantic Tools

- `check regularity [modexp]` reports the result of regularity check of module `check compatibility [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]` 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` lists contents of directories
- `cd pathname` changes working directory of the interpreter
- `pwd` prints working directory
- `! command` forks shell `command ev lisp` evaluate lisp expression `lisp` printing the result `evq lisp` evaluate lisp expression `lisp`