ProofPower

Z REFERENCE MANUAL
Information on the current status of ProofPower is available on the World-Wide Web, at URL:

http://www.lemma-one.demon.co.uk/ProofPower/index.html

This document is published by:

Lemma 1 Ltd.
2nd Floor
31A Chain Street
Reading
Berkshire
UK
RG1 2HX
e-mail: pp@lemma-one.com
KEYWORD INDEX 569
ABOUT THIS PUBLICATION

0.1 Purpose

This document, one of several making up the user documentation for the ProofPower system, is the reference manual for the system.

0.2 Readership

This document is intended to be consulted by users already acquainted with the basic principles behind ProofPower who need detailed information on the behaviour of specific facilities provided by the system. It is not a tutorial for learning the basic use of the system. A ‘keyword in context’ index is supplied, which is useful for identifying the full range of facilities of a particular kind, provided that the reader is familiar with the naming conventions adopted in the development of ProofPower.

0.3 Related Publications

A bibliography is given at the end of this document. Publications relating specifically to ProofPower are:

1. ProofPower Tutorial [4], tutorial covering the basic ProofPower system.
2. ProofPower Z Tutorial [6], tutorial covering ProofPower Z support option.
3. ProofPower Installation and Operation [5];

0.4 Assumptions

It is assumed that the reader has some prior acquaintance with ProofPower either by attending a course on ProofPower or by reading the tutorial.

0.5 Acknowledgements

ICL gratefully acknowledges its debt to the many researchers (both academic and industrial) who have provided intellectual capital on which ICL has drawn in the development of ProofPower.
We are particularly indebted to Mike Gordon of The University of Cambridge, for his leading role in some of the research on which the development of ProofPower has built, and for his positive attitude towards industrial exploitation of his work.

The ProofPower system is a proof tool for Higher Order Logic which builds upon ideas arising from research carried out at the Universities of Cambridge and Edinburgh, and elsewhere. In particular the logic supported by the system is (at an abstract level) identical to that implemented in the Cambridge HOL system [1], and the paradigm adopted for implementation of proof support for the language follows that adopted by Cambridge HOL, originating with the LCF system developed at Edinburgh [2]. The functional language ‘Standard ML’ used both for the implementation and as an interactive metalanguage for proof development, originates in work at Edinburgh, and has been developed to its present state by an international group of academic and industrial researchers. The implementation of Standard ML on which ProofPower is based was itself originally implemented by David Matthews at the University of Cambridge, and is now commercially marketed by Abstract Hardware Limited.

The ProofPower system also supports specification and proof in the Z language, developed at the University of Oxford. We are therefore also indebted to the research at Oxford (and elsewhere) which has contributed to the development of the Z language.
UNIX INTERFACES

**hol_list**

```sml
hol_list [-c] [-d database[#theoryname]] [-i scripts] [-v] theory ...
hol_list [-d database[#theoryname]] [-i scripts] [-v]
hol_list [-c] [-d database] [-i scripts] [-v] -a
```

**Description**

*hol_list* is used to obtain selected information from a ProofPower-HOL database. It functions in the same manner as *zed_list* except that it uses defaults appropriate to the ProofPower-HOL, and a HOL theory lister.

In the first form of use, where a list of one or more theory names is specified, *hol_list* uses ProofPower-HOL to generate on its standard output listings (in the HOL language using the function `output_theory`) of the indicated theories in a form suitable for processing by doctex. Any cache theory (i.e., the theory name is in the list returned by `get_cache_theories`) will be printed with most of the theory detail elided, unless the `-c` option is given.

In the second form, with no list of theory names, *hol_list* lists the names of all the theories in the database whose language is “HOL”, in a sorted order, one per line on its standard output channel. The third form, with `-a`, is like the first but causes all of the theories in the database whose language is “HOL” to be listed in a sorted order.

In any of the three forms the program will start a session as if by command *hol* with the supplied `-d` and `-i` arguments (if any), and it is in this environment that the theory listing is done. The output of this startup will be suppressed, including any indication of failure to load the initialisation scripts.

Each theory is, if possible, made current, or at least in scope, when it is listed.

In any form `-v` indicates the log of the preprocessing should also be output.

**Errors**

*hol_list* prints a message and exits (with the value 1) if the database or any of the theories does not exist. The log of the failure is sent to the standard output, the message to the error output.

**See Also**

*pp_list*, *zed_list*, *pp*, *pp_make_database*

**hol**

```sml
hol [-d database[#theoryname]] [-i files] [-f files [-n|-s] [-v]] [-- ml_flags]
zed [-d database[#theoryname]] [-i files] [-f files [-n|-s] [-v]] [-- ml_flags]
```

**Description**

*hol* and *zed* are identical to *pp*, q.v., except that they use default databases *hol* and *zed* respectively, and hence `-d database` is optional.
SMIL

\[ \text{pp_list} \ [-c] \ -d \ \text{database}[^{\#\text{theory}}] \ [-i \ \text{scripts}] \ [-l \ \text{lang}] \ [-v] \ \text{theory} \ ... \n\]

\[ \text{pp_list} \ -d \ \text{database}[^{\#\text{theory}}] \ [-i \ \text{scripts}] \ [-l \ \text{lang}1 \ [-l \ \text{lang}2 \ ...]] \ [-v] \]

\[ \text{pp_list} \ [-c] \ -d \ \text{database} \ [-i \ \text{scripts}] \ [-l \ \text{lang}1 \ [-l \ \text{lang}2 \ ...]] \ [-v] \ -a \]

**Description**  
\text{pp_list} is used to obtain selected information from a \text{ProofPower} database.

In the first form of use, where a list of one or more theory names is specified, \text{pp_list} uses \text{ProofPower} to generate on its standard output listings of the indicated theories held in the database given by the \(-d\) option in a form suitable for processing by \text{doctex}.

If there is no \(-l\) option then the theory lister used will depend on the language of the theory. If the language is “HOL” then \text{output\_theory} is used. Otherwise it will attempt to use a function named:

\[ <\text{language in lower case}>\_\text{output\_theory} \]

and only if that doesn’t exist will it use \text{output\_theory}. All but the first language will be ignored.

If the \(-l \ \text{lang}\) option is given then it will take the language code of all theories given to be \text{lang}, and then work as above.

If no \(-d\) option is given then the function fails.

Any cache theory (i.e. the theory name is in the list returned by \text{get\_cache\_theories}) will be printed with most of the theory detail elided, unless the \(-c\) option is given.

In the second form, with no list of theory names, \text{pp_list} lists the names of all the theories in the database one per line on its standard output channel in a sorted order. If any \(-l\) options are given then only theories whose language is one of those listed will be noted.

The third form, with \(-a\), is like the first but causes all of the theories in the database to be listed in a sorted order. If any \(-l\) options are given then only theories whose language is one of those given will be listed, and they will be individually printed according to their own language.

In any of the three forms, the program will start a session as if by command \text{pp} with the supplied \(-d\) and \(-i\) arguments (if any), and it is in this environment that the theory listing is done. The output of this startup will be suppressed, including any indication of failure to load the initialisation scripts.

Each theory is, if possible, made current, or at least in scope, when it is listed.

In any form \(-v\) indicates the log of the preprocessing should also be output.

**Errors**  
\text{pp_list} prints a message and exits (with the value 1) if the database or any of the theories does not exist. The log of the failure is sent to the standard output, the message to the error output.

**See Also**  
\text{zed\_list}, \text{hol\_list}, \text{pp}, \text{pp\_make\_database}
**Description**  
`pp_make_database` makes a new child database to contain **ProofPower** theories.  
The new database initially contains a single theory, called the *cache theory* for the database,  
with name given by `cachetheory` (which is used by certain system functions to cache various  
definitions and theorems and which is used as the initial current theory when the database is used  
by the `pp`, `hol` and `zed` commands). If `cachetheory` is omitted then the database name, prefixed  
by “cache’ ” is taken to be the same as the name of the new cache theory.  

The `-p` option may be used to indicate the database which is to be the parent of the new database  
and to indicate which theory in it is to be the parent of the theory `cachetheory`. The parent  
theory is taken to be the cache theory for the parent database if it is not given explicitly.  

For portability, the parent database name should normally be given without any architecture-  
or compiler-specific prefixes or suffixes. Any such prefixes or suffixes will be added automatically  
by `pp_make_database`. If the resulting file name is an absolute path name (i.e., starts with a ‘/’  
character), then that is used as the parent database file name. If the resulting file name is not  
an absolute path name, `pp_make_database` looks for the parent database file first in the current  
directory and then in the user’s search path (given in the environment variable `$PATH`).  

If the `-p` option is not supplied then the database `hol` supplied with the system is used as the  
parent database, and the parent theory is the theory `hol`. This is an appropriate default for a  
**ProofPower-**HOL child database. An appropriate value for **ProofPower-**Z might be the database  
`zed` supplied with the system.  

In interactive use, `pp_make_database` will normally ask for confirmation before overwriting the  
database if it already exists. The `-f` (force) option may be used to suppress the request for  
confirmation before overwriting an existing database.  

The `-v` option produces more output which may be useful for diagnostic purposes.  

Under Poly/ML, databases are subject to an adjustable size limit. By default, `pp_make_database`  
will adjust the size limit of the parent database to the minimum possible and adjust the size limit  
of the child database to the maximum allowed. The `-c` option suppresses these adjustments.  

The supplied child database name will be used to create the child database file name which is  
derived using an algorithm specific to the Standard ML compiler being used.  

**Errors**  
`pp_make_database` prints a message and exits (with value 1) if the parent database or  
theory does not exist, if the new database cannot be created or if the name of the cache theory  
collides with the name of a theory in the parent database.  

Some systems impose a limit on the depth of nesting of the database hierarchy and the command  
will print an error message and exit (with value 1) if this limit would be exceeded.  

The environment variable `PPCOMPILER` may be used to select between the Poly/ML or SML/NJ  
compiler if **ProofPower** has been installed for both compilers. If it is set, the value of this variable  
must be either “POLYML” or “SMLNJ”.

**See Also**  
hol, zed, pp.
SML

\[ \text{pp} -d \text{database}[\#\text{theoryname}] [-i files] [-f files [-n][-s] [-v]] [- - ml\_flags] \]

**Description**  
pp runs ProofPower on the indicated database. If no \(-d\) database is provided to pp, the function fails. For portability, the database name should be given without any architecture- or compiler-specific prefixes or suffixes. Any such prefixes or suffixes will be added automatically by pp. If the resulting file name is an absolute path name (i.e., starts with a ‘/’ character), then that is used as the database file name. If the resulting file name is not an absolute path name, pp searches for the database file using the search path given in the environment variable $PPDATABASEPATH, if set. If $PPDATABASEPATH is not set, pp searches for the database in the current directory, then in the subdirectory db of the user’s home directory and then in the subdirectory db of the ProofPower installation directory.

If specified, \textit{theoryname} gives the name of a theory to be made the current theory at the start of the session. If \textit{theoryname} is not specified, then current theory will be set to the theory current when the database was last saved by \texttt{save_and_quit} or, if just created, to the cache theory for the database. The files identified by any [-i files] options are then executed in turn. files is a comma-separated list of files.

If \(-f\) files is provided, then the files specified in the list \textit{files} are loaded in batch mode. Once loading is complete the database is saved and the batch session is terminated. The saving of the database can be suppress by providing the \(-n\) flag. The default action if any of the files fails to load is for the session to terminate at that point and the database is not saved. By providing the \(-s\) flag, the user can indicate to the system to save the database in batch mode upon failure. The \(-n\) and \(-s\) flags are mutually exclusive. If they are both provided, a warning message is issued and the \(-s\) flag is ignored.

By default, the production of subgoal package output in a batch load is as determined by the value of the flag \texttt{subgoal\_package\_quiet} stored in the database. If the \(-v\) flag is specified to pp, the subgoal package output is produced whereas if the \(-q\) flag is specified, it is suppressed.

If \(-f\) files is not provided, then the system then issues a prompt for user input.

Flags which appear after \(--\) are passed directly onto the Standard ML system for processing. This mechanism can be used to tailor the heap size under SML/NJ: e.g., \texttt{pp -d hol -- -h 32000}.

The environment variable \texttt{PPCOMPILER} may be used to select between the Poly/ML or SML/NJ compiler if ProofPower has been installed for both compilers. If it is set, the value of this variable must be either “POLYML” or “SMLNJ”.

The environment variable \texttt{PPLINELENGTH}, if set, determines the initial value of the string control \texttt{line\_length}. This gives the line length used by various listing facilities, e.g., \texttt{print\_theory} and \texttt{output\_theory}. In interactive use, the xpp interface will set \texttt{PPLINELENGTH} automatically if it has not been set explicitly by the user.

**Errors**  
pp prints a message and exits (with status 1) if the database cannot be accessed or if the theory name specified as part of the \(-d\) argument does not exist in the database.

**See Also**  
pp\_make\_database, pp\_list, pp\_read, hol, zed
**SML**

`xpp [Standard X Toolkit options] [xpp options]`

**Description**  
The program `xpp` provides a convenient way to prepare, check and execute ProofPower scripts under the X Windows System. `xpp` combines a general purpose text editor with a command interface for operating the ProofPower specification and proof facilities. Consult the `xpp` help menu or the `xpp` User Guide for information on how to use it.

'Standard X toolkit options' refers to common options which are automatically supported by most X Windows applications. An example is the option '-display', which may be used to specify the X server on which you wish `xpp` output to be displayed.

The `xpp` option `-f file` may be used to specify a file to be loaded into the editor when `xpp` starts. If you omit this option, `xpp` will start off editing an empty file.

If you specify the `xpp` option `-d database`, `xpp` will run an interactive command session working on the specified ProofPower database. If you omit this option, `xpp` will just run as an editor.

The command line options mentioned above are the most common ones. The program has a number of other options you may wish to use. Consult the `xpp` User Guide for further details.

**See Also**  
USR031: ProofPower - Xpp User Guide

---

**SML**

`zed_list [-c] [-d database[#theory]] [-i scripts] [-v] theory ...
zed_list [-d database[#theory]] [-i scripts] [-v]
zed_list [-c] [-d database] [-i scripts] [-v] -a`

**Description**  
`zed_list` is used to obtain selected information from a ProofPower-Z database. It functions in the same manner as `hol_list` except that it uses defaults appropriate to the ProofPower-Z, and a Z theory lister.

In the first form of use, where a list of one or more theory names is specified, `zed_list` uses ProofPower-Z to generate on its standard output listings (in the Z language using the function `z_output_theory`) of the indicated theories, in a form suitable for processing by doctex. Any cache theory (i.e. the theory name is in the list returned by `get_cache_theories`) will be printed with most of the theory detail elided, unless the `-c` option is given.

In the second form, with no list of theory names, `zed_list` lists the names of all the theories whose language is Z in the database one per line on its standard output channel, in a sorted order.

The third form, with `-a`, is like the first but causes all of the theories in the database whose language is "Z" to be listed in a sorted order.

In any of the three forms the program will start a session as if by command `zed` with the supplied `-d` and `-i` arguments (if any), and it is in this environment that the theory listing is done. The output of this startup will be suppressed, including any indication of failure to load the initialisation scripts.

Each theory is, if possible, made current, or at least in scope, when it is listed.

In any form `-v` indicates the log of the preprocessing should also be output.

**Errors**  
`zed_list` prints a message and exits (with the value 1) if the database or any of the theories does not exist. The log of the failure is sent to the standard output, the message to the error output.

**See Also**  
`pp_list`, `hol_list`, `zed`, `pp_make_database`

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
**conv ascii**
```
conv ascii [−r] [−K] [−k keyword_file_name] <filename> ...
```

**Description**
conv ascii converts ProofPower documents using the extended character set into ASCII keyword format. conv extended performs the opposite conversion.

The *filename* arguments may be just the base-name, perhaps with a directory name prefix, or may include the `.doc` suffix. By default, the result of the conversion is checked by converting in the opposite direction and comparing with the input. If the check is successful, the `.doc` file is then replaced by the result of the conversion. If the conversion appears to be unsuccessful the output of the conversion is placed in a file with suffix `.asc` or `.ext` in the current directory, and the `.doc` file is left unchanged.

If −r is specified no check is made and the output of the conversion is placed in a file with suffix `.asc` or `.ext`.

Note that the check will always fail on a file containing a mixture of extended characters and ASCII keywords. Use −r and then, if all is well, overwrite the `.doc` file with the `.asc` or `.ext` file using *mv* or *cp* to convert a such file into a homogeneous one.

The check will also fail if the file is already in the desired format, in which case there is no need to run the conversion program.

The −K and −k options indicate the keyword files to be used as for doctex and docsml (and are only needed if fonts other than those supplied with ProofPower are being used.)

**See Also**
docpr

---

**docdvi**
```
docdvi [−v] [−f view_file_name] [−K] [−k keyword_file_name] 
[−e edit_file_name] [−p TeX_program_name] [−N] <filename> ...
```

**Description**
Shell script that combines the actions of doctex, bibtex (which is part of the basic TEX distribution) and texdvi with the intention of fully processing a simple document from its `.doc` form to a printable `.dvi` file.

The option −N controls how many times LATEX should be invoked, the default is three (i.e., ‘−3’), the values of *N* may be in the range one to four inclusive. The other options are as for doctex and texdvi.

LATEX and bibtex are run so that if they detect errors and prompt for input they will read an end of file and thus stop immediately.

In some cases an extra run of LATEX may be required. In these cases LATEX will output the message: ‘LaTeX Warning: Label(s) may have changed. Rerun to get cross-references right.’

**See Also**
doctex, texdvi

---

**docpr**
```
docpr [−n] [−p] [−s] [−v] [−w width] <filename> ...
```

**Description**
Shell script that prints out files that may contain extended characters in a verbatim-like manner. Lines may be numbered in the output by using the −n option. Lines are folded at at 80 characters wide, or at the width given by the −w *width* option. The output may be viewed on screen with the −s option, the default is to print the output. By default all intermediate files are deleted, with the −p option the `.dvi` file will be preserved. With the −v option details of the files processed are listed on the standard output.

**See Also**
doctex, texdvi

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
SML
doctex [-v] [-f view_file_name] [-e edit_script]
[-K] [-k keyword_file_name] <filename> ...
docsml [-v] [-f view_file_name] [-K] [-k keyword_file_name] <filename> ...

Description Shell scripts that sieve each of their filename arguments to produce various output files. These arguments may be given just as the base-name, perhaps with a directory name prefix, or may include the .doc suffix. When the −v option is set details of the files read and written are shown on the standard output. The default steering files are named sieveview and sievekeyword and looked for first in the current directory, second on the callers execution path (from the UNIX environment variable $PATH). The default viewfile may be changed with the −f option. The default keyword file may be suppressed with the −K option. Additional keyword files may be given with the −k option which may be used several times. The −e option identifies the name of a script of ex commands which are used to edit the .tex file.

The output file from doctex has suffix .tex and is intended for processing with texdvi. The output file from docsml has suffix .sml and is typically processed by loading it into a ProofPower database.

See Also texdvi, docdvi

SML
texdvi [-v] [-b] [-p TeX_program_name] <filename> ...

Description Shell script that runs \LaTeX\ on each of the filename arguments to produce the corresponding .dvi file. These arguments may be just the base-name, perhaps with a directory name prefix, or may include the .tex suffix. When the −v option is set details of the .tex and .dvi files read and written are shown on the standard output. To support indexing this script ensures that a .sid file exists before \LaTeX\ is called; when \LaTeX\ completes any .idx file is sorted to create a .sid file ready for the next time texdvi is used. When initially producing a .dvi file texdvi will need to be run up to four times so that the derived information such as tables of contents and inter-page references stabilise.

The \LaTeX\ program is latex by default but a different program may be specified with the −p option.

If the −b option is specified, bibtex is run after running latex.

See Also texdvi, docdvi
Chapter 2

PROGRAMMING UTILITIES
## 2.1 Error Management

**SML**

```
signature BasicError = sig
```

**Description** This is the signature of the structure `BasicError`.

**SML**

```
exception Fail of MESSAGE
exception Error of MESSAGE
```

**Description** These exceptions are raised to report error conditions. `Fail` is for errors which may be trapped (so that the associated message is suppressed). `Error` is intended to ensure that the message will be reported and, by convention, should not be trapped.

**Uses** Obscure debugging situations.

**SML**

```
type MESSAGE
```

**Description** This type is used to pass error and other messages around in the system.

**Uses** Obscure debugging situations.

**SML**

```
val area_of : exn -> string
```

**Description** This returns the name of the function which raised an exception (provided the exception was raised with `fail` following the usual conventions). If the exception was not the one raised by `fail` then it is raised again.

**Uses** For use when coding new facilities to add to the system.

**SML**

```
val divert : exn -> string -> string -> int -> (unit -> string) list -> 'a
val list_divert : exn -> string -> ((string * int * (unit -> string) list)) list -> 'a
val elaborate : exn -> int -> string -> int -> (unit -> string) list -> 'a
```

**Description** These functions support a style of error handling in which, if an error is reported during evaluation of an expression, the source of the error may be checked and the error report modified if needed to give a more meaningful report to the user. Sources of errors are identified by the string passed as the first argument to the function `fail` which is used to flag trappable errors. By convention, this string gives the name of the top level function which has raised the error.

In the call `divert X from new new_msg inserters`, `X` is the exception which has been raised and `from` identifies a possible source for an error report. `inserters` is a list of functions to be used to generate insertions for the error message (as with `fail` q.v.). If an error has been reported by `from`, the call will have the same effect as if `fail new new_msg inserters` had been called.

`list_divert X new triples` handles the more general case in which errors from several sources are expected. `X` and `new` are as for `divert`. `triples` gives a list of triples giving possible sources of error and the corresponding new messages and insertion functions.

`elaborate` is similar to `divert` but makes it possible to expand on the information provided by the function that has raised the exception. In the call `elaborate X old_msg new new_msg inserters`, `old_msg` identifies an error message text. If `X` results from a call of `fail` (or equivalent) with that error message text, the effect is as if `fail new new_msg (inserters'@inserters)` had been called, with `inserters'` the list of string-valued functions associated with `X`.

**Uses** For use when coding new facilities to add to the system.
2.1. Error Management

SML

```sml
val fail : string -> int -> (unit -> string) list -> 'a
val error : string -> int -> (unit -> string) list -> 'a
```

**Description** These functions report a message of the corresponding class with text determined by an integer parameter and a list of string valued functions. The string parameter is intended to give the name of the top level function which has invoked the error message.

The error messages are stored in a database maintained by `new_error_message` and the integer parameter gives the key for the desired entry in the database. The list of string-valued functions allow the messages to be parameterised. When the error is printed, the functions are evaluated to produce a list of strings. Substrings of the database entry of the form “?i” where i is a decimal digit are replaced by the corresponding entries in the list (with “?0” corresponding to the head of the list). (If there are more than ten entries in the list, entries after the tenth are evaluated but the result of the evaluation is ignored).

`fail` is for unrecoverable errors which may, however, be trapped. It causes exception `Fail` to be raised.

`error` is for unrecoverable errors which must be reported to the user. It causes exception `Error` to be raised. As for `set_flag` etc.

**Uses** For use when coding new facilities to add to the system.

SML

```sml
val get_error_message : int -> (string list) -> string
```

**Description** This function returns the entry in the error message database associated with the given integer key. The second parameter gives a list of strings to be inserted into the text of the message. Substrings of the message text of the form “?i”, where i is a decimal digit, indicate positions where these insertions are to be made. “?0” identifies the string at the head of the list etc.

**Errors**

2002 The error number ?0 does not identify an entry in the error message database

SML

```sml
val get_error_messages : unit -> {id:int, text:string} list
val set_error_messages : {id:int, text:string} list -> unit
```

**Description** `get_error_messages` returns the contents of the error message database as a list.

`set_error_messages` uses `new_error_message` to add any new error messages in a list of such into the database of error messages. It will issue a message on the standard output (and change nothing) for any messages that do not match those already present.
**val get_message_text**: `MESSAGE -> string`  
**Description**  This returns a printable form of an error message text. The message text is given without the header information which is inserted by `get_message`, q.v.

**Uses**  In constructing extensions to the system.

The error message data structure includes functions passed as arguments to `fail` or `error` that are called to generate parts of the message. If any of these functions raises `Fail`, the exception is caught and the string returned is a report on the failure.

**See Also**  `fail`, `error`, `get_message`  

**Errors**

- 2004 Failure detected formatting message: ?0
- 2005 * failure ?0.?1 reported *

**val get_message**: `MESSAGE -> string`  
**Description**  This returns a printable form of an error message value. The message text is followed by a trailer of the form “<#nnnnn area>”, where #nnnnn is the number of the message in the error database and area typically gives the name of the function which gave rise to the error message.

**Uses**  In constructing extensions to the system.

**See Also**  `get_message_text`  

**val new_error_message**: `{id:int, text:string} -> unit`  
**Description**  This function adds a new entry to the database of error messages. Note that substrings of the message of the form “?i” where i is a decimal digit have special significance (see `fail` for details). “??” may be used to insert a single “?” character in a message.

If the `id` and the `text` are identical to an existing entry, then `new_error_message` has no effect. If there is an existing entry with the same `id` but a different `text` then a message is reported on the standard output and the existing entry is left unchanged.

**Errors**

- 2001 The error number ?0 is already in use for a different message

**Uses**  For use when adding facilities to the system.

**val pass_on**: `exn -> string -> string -> 'a`  
**Description**  `pass_on exn from to` is similar to `reraise`, q.v., but the function name associated with the exception is only modified if it is equal to `from`, in which case it is changed to `to`.

**val pending_reset_error_messages**: `unit -> unit -> unit`  
**Description**  This function is intended for use in system initialisation and shutdown. The binding `val p = pending_reset_error_messages()`, defines `p` as a function which will set the internal state of the `BasicError` module to the value it had at the time the binding for `prcs` was made. This is used to remember the set-up for error messages introduced in a child database.
2.2 Data Types

SML

|val pp′change_error_message : {id:int, text:string} −→ unit

Description  This function changes an entry in the database of error messages. If the number does not identify an existing entry a new entry is made.

Uses  ICL Use only.

SML

|val pp′error_init : unit −→ unit

Description  This function is used to initialise certain aspects of the error reporting system. It is called automatically at the start of each session. It is harmless, but unnecessary, to call it within a session.

SML

|val reraise : exn −→ string −→ 'a

Description  This re-raises an exception. If the exception is the exception Fail (as raised by fail, q.v.) then the function name associated with the exception is changed to the name given by the second argument.

Uses  For use when coding new facilities to add to the system.

2.2 Data Types

SML

|signature UtilitySharedTypes = sig

Description  Any new types in the Utility structures mentioned in more than one signature will be declared in this signature.

SML

|datatype 'a OPT = Nil | Value of 'a;

Description  A type of “optional” values.

Uses  A typical use for the datatype 'a OPT is in implementing partial functions for which raising an exception is not an appropriate action for undefined cases.

See Also  force_value, is Nil

SML

|type 'a S_DICT;

Description  The type of simple dictionaries: (string * 'a) list.

See Also  Signature SimpleDictionary.
2.3 Lists

SML

signature ListUtilities = sig

Description Holds a variety of utility Standard ML list functions.

SML

val all_different : "a list -> bool;

Description all_different determines whether a list has any repeated entries.

See Also all_distinct

SML

val all_distinct : ('a * 'a -> bool) -> 'a list -> bool;

Description all_distinct eq list determines whether list has any repeated entries using eq to test for equality. Each member, x of the list is tested against all the subsequent members of the list, with x being the first argument to eq.

See Also all_different

SML

val all : 'a list -> ('a -> bool) -> bool;

Description all list cond is true iff. all elements of list satisfy cond.

SML

val any : 'a list -> ('a -> bool) -> bool;

Description any list cond is true iff. some element of list satisfies cond.

SML

val app : ('a -> unit) -> 'a list -> unit;

Description Apply a function to each element of a list in turn for the side-effect.

SML

val combine : 'a list -> 'b list -> ('a * 'b) list;

Description combine combines a pair of lists into a list of pairs. It is the left inverse of split.

Errors

1007 Cannot combine unequal length lists

See Also split, zip

SML

val contains : "a list -> "a -> bool;

Description contains list x searches for a member of list equal to x and returns true iff. it finds one.

See Also present, mem

SML

val cup : "a list * "a list -> "a list;

Description An infix binary union operation for lists, with Standard ML equality test. It has the same result ordering as union(q.v.).

See Also list_cup, union
SML
val diff : "a list * "a list -> "a list;

Description  diff is the set difference operator for lists.

SML
val drop : 'a list * ('a -> bool) -> 'a list;

Description  list drop cond is the list obtained by deleting all members of list for which the boolean function cond is true.
See Also  less

SML
val filter : ('a -> bool) -> 'a list -> 'a list;

Description  filter pred list returns a list that is list, except that elements of the list that don’t satisfy pred are dropped.

Definition

| filter pred [] = [] |
| filter pred (a :: x) = ( |
|       if pred a |
|       then (a :: filter pred x) |
|       else filter pred x); |

SML
val find : 'a list -> ('a -> bool) -> 'a;

Description  find list cond searches for the first member of list satisfying cond, and returns such a member if there is one.
Errors
1004  Element cannot be found in list

SML
val flat : 'a list list -> 'a list;

Description  flat takes a list of lists and returns the result of concatenating them all.

SML
val fold : ('a * 'b -> 'b) -> 'a list -> 'b -> 'b;

Description  Fold a list into a single value:

definition  fold f [x1, x2, ..., xk] b = f(x1, f(x2, ... f(xk, b))...)

See Also  revfold

SML
val force_value : 'a OPT -> 'a;

Description  Force an object of type 'a OPT (q.v) into one of type 'a:

Definition

| force_value (Value x) = x |

Errors
1001  Argument may not be Nil
Chapter 2. PROGRAMMING UTILITIES

|val from : 'a list * int -> 'a list;

Description  list from n takes the trailing slice of list. It uses 0-based indexing. If n is 0 or negative then entire list is returned, and if n indexes past the other end of the list then the empty list is returned.

Example  \([0,1,2,3]\) from 2 = [2,3]

See Also  to

|val grab : "a list * "a -> "a list;

Description  list grab what is the list obtained by inserting what at the head of list if it is not a member of it already, in which case list is returned.

See Also  insert

|val hd : 'a list -> 'a;
|val tl : 'a list -> 'a list;

Description  hd returns first element of a list, tl returns all but the first element of a list.

Definition  

\[
\begin{align*}
hd (a :: x) &= a \\
\text{tl} (a :: x) &= x
\end{align*}
\]

Errors  

|1002  An empty list has no head |
|1003  An empty list has no tail |

|val insert : ('a * 'a -> bool) -> 'a list -> 'a -> 'a list;

Description  insert eq list what is the list obtained by inserting what at the head of list if it is not a member, by equality test eq, of it already, in which case list is returned.

See Also  grab

|val interval : int -> int -> int list;

Description  interval a b is the list \( [a, a+1, a+2 \ldots, b] \). This is taken to be [] if \( a > b \) and to be \([a]\) if \( a = b \).

|val is Nil : 'a OPT -> bool

Description  Is the argument equal to Nil (q.v).

Definition  

\[
\begin{align*}
is\_Nil Nil &= true \\
is\_Nil _ &= false
\end{align*}
\]

|val is nil : 'a list -> bool;

Description  is nil tests whether a list is empty([]). It can be used for lists of types which do not admit equality.
2.3. Lists

SML

\textbf{val lassoc1} : ("a + "a) list -> "a -> "a;

\textbf{Description} lassoc1 \textit{alist arg} is \textit{x}, where \textit{(arg, x)} is the first element of \textit{alist} with \textit{arg} as its left item. The function is made total by taking \textit{arg} as the result if there is no appropriate member of the list.

\textbf{See Also} lassoc\textit{N} and rassoc\textit{N}, where \textit{N} = 1 \ldots 5.

SML

\textbf{val lassoc2} : ("a + 'b) list -> ("a -> 'b) -> "a -> 'b;

\textbf{Description} lassoc2 \textit{alist f arg} is \textit{x}, where \textit{(arg, x)} is the first element of \textit{alist} with \textit{arg} as its left item. The function is made total by returning \textit{f arg} if there is no appropriate member of the list.

\textbf{See Also} lassoc\textit{N} and rassoc\textit{N}, where \textit{N} = 1 \ldots 5.

SML

\textbf{val lassoc3} : ("a + 'b) list -> "a -> 'b;

\textbf{Description} lassoc3 \textit{alist arg} is \textit{x}, where \textit{(arg, x)} is the first element of \textit{alist} with \textit{arg} as its left item.

\textbf{Errors} 1005 No such value in association list

\textbf{See Also} lassoc\textit{N} and rassoc\textit{N}, where \textit{N} = 1 \ldots 5.

SML

\textbf{val lassoc4} : ("a + 'b) list -> 'b -> "a -> 'b;

\textbf{Description} lassoc4 \textit{alist default arg} is \textit{x}, where \textit{(arg, x)} is the first element of \textit{alist} with \textit{arg} as its left item. The function is made total by returning \textit{default} if there is no appropriate member of the list.

\textbf{See Also} lassoc\textit{N} and rassoc\textit{N}, where \textit{N} = 1 \ldots 5.

SML

\textbf{val lassoc5} : ("a + 'b) list -> "a -> 'b OPT;

\textbf{Description} lassoc5 \textit{alist arg} is Value \textit{x}, where \textit{(arg, x)} is the first element of \textit{alist} with \textit{arg} as its left item. The function is made total by returning \textit{Nil} if there is no appropriate member of the list.

\textbf{See Also} lassoc\textit{N} and rassoc\textit{N}, where \textit{N} = 1 \ldots 5.

SML

\textbf{val length} : 'a list -> int;

\textbf{Description} length returns the length of a list. Note that the Standard ML function \textit{size} can be used to find the length of strings.

SML

\textbf{val less} : "a list * "a -> "a list;

\textbf{Description} list less \textit{what} is the list obtained by deleting all members of \textit{list} which are equal to \textit{what}.

\textbf{See Also} drop
SML

val list_cup : "a list list -> "a list;

Description A distributed union operation for lists, with Standard ML equality test.

Definition

\[
\text{list}_\text{cup} \ [\text{list}_0, \text{list}_1, ..., \text{list}_n] = \\
\text{list}\_0 \ \text{cup} \ (\text{list}_1 \ \text{cup} \ ...(\text{list}_n \ \text{cup} \ [])...)
\]

See Also cup, list_union


SML

val list_overwrite : ("a * 'b) list * ("a * 'b) list -> ("a * 'b) list;

Description alist list_overwrite olist overwrites alist with each element of olist, using overwrite(q.v).

Definition

fun list_overwrite olist = (  
  fold (fn (l1, l2) => l2 overwrite l1) olist alist
)

See Also overwrite, list_roverwrite.


SML

val list_roverwrite : ('a * 'b) list * ('a * 'b) list -> ('a * 'b) list;

Description alist list_roverwrite olist overwrites alist with each element of olist, using roverwrite (q.v.).

Definition

fun list_roverwrite olist = (  
  fold (fn (l1, l2) => l2 roverwrite l1) olist alist
)

See Also roverwrite, list_overwrite.


SML

val list_union : ('a * 'a -> bool) -> 'a list -> 'a list;

Description A distributed union operation for lists, with parameterised equality test:

Definition

\[
\text{list}_\text{union} \ eq \ [\text{list}_0, \text{list}_1, ..., \text{list}_n] = \\
\text{union} \ eq \ \text{list}_0 \ ((\text{union} \ eq \ \text{list}_1 \ ...(\text{union} \ eq \ \text{list}_n \ [])...))
\]

See Also union, list_cup.


SML

val mapfilter : ('a -> 'b) -> 'a list -> 'b list;

Description Map a function over a list. If, when evaluating

\[
\text{mapfilter} \ f \ (x_1 :: \ldots \ x_k - 1 :: x_k :: x_k + 1 :: \ldots)
\]

the evaluation of \( f \ x_k \) raises a Fail exception, then the result will be

\[
(f \ x_1 :: \ldots \ f \ x_k - 1 :: f \ x_k + 1 :: \ldots)
\]


SML

val mem : "a * "a list -> bool;

Description \( x \ \text{mem} \ \text{list} \) searches for a member of list equal to \( x \) and returns true iff. it finds one.

See Also contains, present

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
2.3. Lists

SML
\[ \text{val} \ nth : \text{int} \to \text{'a list} \to \text{'a}; \]

Description Return the \( n \)-th element of a list. The head of the list is the \( 0 \)-th element.

Errors
\[ 1009 \quad \text{Index past ends of list} \]

SML
\[ \text{val} \ overwrite : (\text{''a} \to \text{''b}) \text{ list} \times (\text{''a} \to \text{''b}) \to (\text{''a} \to \text{''b}) \text{ list}; \]

Description \( \text{alist overwrite} (a, b) \) gives the list in which the first pair in \( \text{alist} \) that has the left item \( a \) is replaced with the pair \( (a, b) \). If no such pair is found in \( \text{alist} \) then it returns the list of \( (a, b) \) appended to the tail of \( \text{alist} \).

See Also \ overwrite, list overwrite

SML
\[ \text{val} \ present : (\text{''a} \to \text{bool}) \to \text{'a} \to \text{'a list} \to \text{bool}; \]

Description \( \text{present eq x list} \) searches for a member, \( y \), of \( \text{list} \) that satisfies \( \text{eq}(x, y) \) and returns \text{true} iff. it finds one.

See Also \ contains, mem

SML
\[ \text{val} \ rassoc1 : (\text{''a} \times \text{''a}) \text{ list} \to \text{''a} \to \text{''a}; \]

Description \( \text{rassoc1 alist arg} \) is \( x \), where \( (x, \text{arg}) \) is the first element of \( \text{alist} \) with \( \text{arg} \) as its right item. The function is made total by taking \( \text{arg} \) as the result if there is no appropriate member of the list.

See Also \ lassoc\( N \) and rassoc\( N \), where \( N = 1 \ldots 5 \).

SML
\[ \text{val} \ rassoc2 : (\text{''a} \times \text{''b}) \text{ list} \to (\text{''b} \to \text{'a}) \to \text{''b} \to \text{'a}; \]

Description \( \text{rassoc2 alist f arg} \) is \( x \), where \( (x, \text{arg}) \) is the first element of \( \text{alist} \) with \( \text{arg} \) as its left item. The function is made total by returning \( f \ \text{arg} \) if there is no appropriate member of the list.

See Also \ lassoc\( N \) and rassoc\( N \), where \( N = 1 \ldots 5 \).

SML
\[ \text{val} \ rassoc3 : (\text{''a} \times \text{''b}) \text{ list} \to \text{''b} \to \text{'a}; \]

Description \( \text{rassoc3 alist arg} \) is \( x \), where \( (x, \text{arg}) \) is the first element of \( \text{alist} \) with \( \text{arg} \) as its right item.

Errors
\[ 1005 \quad \text{No such value in association list} \]

See Also \ lassoc\( N \) and rassoc\( N \), where \( N = 1 \ldots 5 \).

SML
\[ \text{val} \ rassoc4 : (\text{''a} \times \text{''b}) \text{ list} \to \text{'a} \to \text{''b} \to \text{'a}; \]

Description \( \text{rassoc4 alist default arg} \) is \( x \), where \( (x, \text{arg}) \) is the first element of \( \text{alist} \) with \( \text{arg} \) as its right item. The function is made total by returning \( \text{default} \) if there is no appropriate member of the list.

See Also \ lassoc\( N \) and rassoc\( N \), where \( N = 1 \ldots 5 \).
val rassoc5 : ('a * "b) list -> 'b -> 'a OPT;

Description  rassoc5  alist  arg  is  Value  x,  where  (x,  arg)  is  the  first  element  of  alist  with  arg  as  its  right  item.  The  function  is  made  total  by  returning  Nil  if  there  is  no  appropriate  member  of  the  list.

See Also  lassocN  and  rassocN,  where  N = 1 . . . 5.

val revfold : ('a * 'b -> 'b) -> 'a list -> 'b;

Description  Fold  a  list  into  a  single  value:

Definition  revfold f [x1, x2, ..., xk] b = f(xk, ..., f(x2, f(x1, b))...)

See Also  fold

val roverwrite : ('a * 'b) list * ('a * "b) -> ('a * "b) list;

Description  alist  roverwrite  (a, b)  gives  the  list  that  in  which  the  first  pair  in  alist  that  has  the  right  item  b  is  replaced  with  the  pair  (a, b).  If  no  such  pair  is  found  in  alist  then  it  returns  the  list  of  (a, b)  appended  to  the  end  of  alist.

See Also  overwrite, list_roverwrite

val split3 : ('a * 'b * 'c) list -> 'a list * 'b list * 'c list;

Description  Split  a  list  of  triples  into  a  triple  of  lists.  split3  is  the  analogue  of  split  for  lists  of  triples.

See Also  split

val split : ('a * 'b) list -> 'a list * 'b list;

Description  Split  a  list  of  pairs  into  a  pair  of  lists.

Definition  split [(x0, y0), (x1, y1), ... (xk, yk)] = [x0, x1, ..., xk],  [y0, y1, ..., yk]

See Also  split3, combine

val subset : "a list * "a list -> bool;

Description  l1  subset  l2  is  true  iff.  all  the  elements  of  l1  are  also  elements  of  l2

See Also  

val to : 'a list * int -> 'a list;

Description  list  to  n  takes  the  initial  slice  of  list.  It  uses  0-based  indexing.  If  n  is  0  or  negative  an  empty  list  is  returned,  and  if  n  indexes  past  the  other  end  of  list  then  the  entire  list  is  returned.

Example  [0,1,2,3] to 2 = [0,1,2]

See Also  from
2.3. Lists

SML

\[
\text{val union} : ('a * 'a -> bool) -> 'a list -> 'a list -> 'a list;
\]

Synopsis  A prefix binary union operation for lists, with parameterised equality test.

Description  \(\text{union}\) is essentially a binary union operation for lists. Since we need it to work on types which are not equality types, it has a parameter giving the relation to be used to determine equality of members of the lists. In some cases it may be important for the order of members of the union to be known. The rule is that \(\text{union eq list1 list2}\) is the list obtained by prepending those elements of \(\text{list1}\) not already present in \(\text{list2}\), to the list \(\text{list2}\). Presence for \(x\) in the list being created being that there is a member, \(y\), of the list being created with \(\text{eq}(x, y) = \text{true}\). If \(\text{list1}\) contains duplicates then all but the rightmost will be eliminated, but those in \(\text{list2}\) will not be. Note also that if one of the lists is small it is better supplied as the first list argument if efficiency is of the essence.

Definition

\[
\text{union eq \( (\text{list1 @} [a])\)} \text{ list2} = \text{union eq list1 } \text{(}
\begin{align*}
& \text{if present eq a list2} \\
& \text{then list2} \\
& \text{else } (a :: \text{list2})
\end{align*}
\text{)} \text{ } | \text{ union eq } [] \text{ list2} = \text{list2}
\]

See Also  \(\text{cup, list\_union}\)

SML

\[
\text{val which} : (('a * 'a) -> bool) -> 'a -> 'a list -> int OPT;
\]

Description  \(\text{which eq x list}\) returns Value of the position of first element, \(y\), in \(\text{list}\) for which \(\text{eq}\ x\ y\) is true. It uses 0-based indexing. If no such \(y\) is found, then it returns \(\text{Nil}\).

SML

\[
\text{val zip} : ('a -> 'b)\text{list} -> 'a \text{ list} -> 'b \text{ list};
\]

Description  Given a list of functions, and a list of arguments, of the same length, apply each function to its corresponding argument. For the cases when the list of functions induce side effects, note that the functions are applied from the head of their list to the tail, and will be applied until there are insufficient elements of either list to continue. If there lists are not of equal length then at that point a failure will be raised.

See Also  \(\text{combine}\)

Errors

\(1008\)  List lengths differ

SML

\[
\text{val } \sim=< : "a list \* "a list -> bool;}
\]

\[
\text{val } \sim= : "a list \* "a list -> bool;
\]

Description  \(l1 \sim=< l2\) is true iff. every member of \(l1\) is also a member of \(l2\). \(l1 \sim= l2\) is true iff. the set of members of \(l1\) is equal to the set of members of \(l2\).

See Also  \(\text{subset}\)
2.4 Functions

SML
signature FunctionUtilities = sig
Description Holds a variety of utility Standard ML functions concerned with functions.

SML
val ** : ('a -> 'b) * ('c -> 'd) -> 'a * 'c -> 'b * 'd;
Description The infix operator **, with precedence 4 (higher than "o"), applies the first of a pair of functions to the first of a pair, and the second of the pair of functions to the second of the pair, returning the pairing of the results.
Definition
(f ** g) x = (f x, g x)

SML
val curry : ('a * 'b -> 'c) -> 'a -> 'b -> 'c;
Description curry f a b gives f (a, b).
See Also uncurry

SML
val fst : 'a * 'b -> 'a;
Description fst is the left projection function for pairs: fst(a, b) = a.
See Also snd

SML
val fun_and : (('a -> bool) * ('a -> bool)) -> 'a -> bool;
val fun_or : (('a -> bool) * ('a -> bool)) -> 'a -> bool;
val fun_not : ('a -> bool) -> 'a -> bool;
val fun_true : 'a -> bool;
val fun_false : 'a -> bool;
Description These functions allow a style of programming that handles predicates rather than booleans.
Definition
(f fun_and g) x = f x andalso g x
(f fun_or g) x = f x orelse g x
(fun_not f) x = not(f x)
fun_true x = true
fun_false x = false

SML
val fun_pow : int -> ('a -> 'a) -> 'a -> 'a;
Description For non-negative n, fun_pow n f is f^n, i.e. the function λx•f(...f(fx)....)
where f appears n times.
Errors
1010 First argument must not be negative
2.4. Functions

**SML**

<table>
<thead>
<tr>
<th>val repeat : (unit -&gt; 'a) -&gt; unit;</th>
</tr>
</thead>
<tbody>
<tr>
<td>val iterate : ('a -&gt; 'a) -&gt; 'a -&gt; 'a;</td>
</tr>
</tbody>
</table>

**Description** repeat applies its argument to () until it fails (with an error generated by fail, q.v.), whereupon it returns (). iterate f a applies f to a. If this causes no failure it then calls iterate f on the result. If it fails (with an error generated by fail, q.v.) it returns a. Failures other than those caused by fail are not handled.

**Definition**

\[
\begin{align*}
\text{fun repeat } f & = (f ()); \text{ repeat } f \text{ handle (Fail _) } => ()
\end{align*}
\]

\[
\begin{align*}
\text{fun iterate } f \ a & = (\text{iterate } f (f \ a)) \text{ handle (Fail _) } => a
\end{align*}
\]

**SML**

| val snd : 'a * 'b -> 'b;  |

**Description** snd is the right projection function for pairs: \(\text{snd}(a, b) = b\).

**See Also** fst

**SML**

| val swap : 'a * 'b -> 'b * 'a;  |

**Description** swap interchanges the elements of a pair: \(\text{swap}(a, b) = (b, a)\).

**SML**

| val switch : ('a -> 'b -> 'c) -> 'b -> 'a -> 'c;  |

**Description** switch \(f\ a\ b\) gives \(f b a\).

**SML**

| val uncurry : ('a -> 'b -> 'c) -> 'a * 'b -> 'c;  |

**Description** uncurry \(f\ (a, b)\) gives \(f a\ b\).

**See Also** curry
### 2.5 Combinators

<table>
<thead>
<tr>
<th>SML</th>
<th>signature Combinators = sig</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Holds the three combinators $S$, $K$, $I$.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>val I : $'a$ -&gt; $'a$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>The identity combinator: $I x = x$.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>val K : $'a$ -&gt; $'b$ -&gt; $'a$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>The deletion combinator: $K x y$ is $x$.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>val S : ($'a$ -&gt; $'b$ -&gt; $'c$) -&gt; ($'a$ -&gt; $'b$) -&gt; $'a$ -&gt; $'c$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>The duplication combinator: $S f g a$ is $(f a)(g a)$.</td>
</tr>
</tbody>
</table>
2.6 Characters

**Signature**

```
signature CharacterUtilities = sig
```

**Description**
Holds a variety of utility Standard ML functions concerned with character handling.

```
val is_all_decimal : string -> bool;
```

**Description**
`is_all_decimal` checks whether a string consists of one or more decimal digits.

```
val nat_of_string : string -> int;
```

**Description**
`nat_of_string` converts a string into non-negative integer (using decimal notation).

**See Also**
`string_of_int`

**Errors**

- `1012`: ?0 is not a decimal string
- `1013`: String is empty

```
val string_of_int : int -> string;
```

**Description**
`string_of_int` converts an integer into a decimal string.

**See Also**
`nat_of_string`
2.7 Simple Dictionary

SML

signature SimpleDictionary = sig

Description Holds a set of Standard ML functions concerned with a linear search dictionary.

Uses For handling small dictionaries.

See Also EfficientDictionary.

SML

val initial_s_dict : 'a S_DICT;

Description The empty dictionary, which gives a starting point for the use of the simple dictionary functions. It does not associate a value with any name.

SML

val s_delete : string -> 'a S_DICT -> 'a S_DICT;

Description s_delete deletes an element of the domain of a dictionary. If the element is not in the domain it returns the dictionary unchanged. s_delete name dict returns a dictionary that does not associate anything with name, but otherwise associates as dict.

SML

val s_enter : string -> 'a -> 'a S_DICT -> 'a S_DICT;

Description s_enter implements overwriting by a singleton function. s_enter name value dict returns the dictionary that associates name with value, and otherwise associates as dict. Overwriting is done “in place”, entries not previously present will be placed at the end of the dictionary viewed as a list.

SML

val s_extend : string -> 'a -> 'a S_DICT -> 'a S_DICT;

Description s_extend implements extension by a singleton function, that is to say it is like s_enter. s_extend name value dict returns the dictionary that associates name with value, and otherwise associates as dict. It fails if name is already in the domain of dict. Entries not previously present will be placed at the head of the dictionary viewed as a list.

Errors

1014 ?0 is already in dictionary

SML

val s_lookup : string -> 'a S_DICT -> 'a OPT;

Description s_lookup implements application (of the dictionary viewed as a partial function). s_lookup name dict returns the value that dict associates with name.

SML

val s_merge : 'a S_DICT -> 'a S_DICT -> 'a S_DICT;

Description s_merge extends one dictionary by another. The dictionary s_merge dict1 dict2 will associate a name with the value that either dict1 or dict2 associates it with.

Failure Will get the s_extend failure message if any element is common to the domains of both dictionaries (dict1 and dict2). Duplicate keys in the first list will also cause an s_extend error, but will be replicated in the result if found in the second list.
## 2.8 Efficient Dictionary

**SML**

\[
\text{signature EfficientDictionary} = \text{sig}
\]

**Description**  
This is the signature of a structure implementing dictionaries (lookup-up tables) based on hash-search techniques.

**Uses**  
For handling large dictionaries.

**See Also**  
SimpleDictionary.

**SML**

\[
\text{type } \alpha \text{ E_DICT};
\]

**Description**  
The type of efficient dictionaries.

**SML**

\[
\text{type } \text{E_KEY};
\]

**val**  
\text{e_get_key} : string \to \text{E_KEY};

**val**  
\text{e_key_lookup} : \text{E_KEY} \to 'a \text{ E_DICT} \to 'a \text{ OPT}

**val**  
\text{e_key_enter} : \text{E_KEY} \to 'a \to 'a \text{ E_DICT} \to 'a \text{ E_DICT};

**val**  
\text{e_key_extend} : \text{E_KEY} \to 'a \to 'a \text{ E_DICT} \to 'a \text{ E_DICT};

**val**  
\text{e_key_delete} : \text{E_KEY} \to 'a \text{ E_DICT} \to 'a \text{ E_DICT};

**val**  
\text{string_of_e_key} : \text{E_KEY} \to \text{string};

**Description**  
The abstract data type \text{E_KEY} represents the hash-keys used in the internals of the efficient dictionary \text{(E_DICT)} access functions. \text{e_get_key} computes the hash-key for a given string. This may then be used as an argument to the functions \text{e_key_lookup}, \text{e_key_enter}, \text{e_key_extend} and \text{e_key_delete} which perform the same functions as the corresponding functions without “key,” in the name. This approach may be used if the same string is to be used to access several efficient dictionaries to avoid the computational cost of recalculating the hash-key. \text{string_of_key} is the left inverse of \text{e_get_key}.

**Failure**  
The failures are exactly as for the corresponding string access functions. In particular, the area names in error messages are, e.g., “e_lookup” rather than “e_key_lookup” etc.

**SML**

\[
\text{val } \text{e_delete} : \text{string} \to 'a \text{ E_DICT} \to 'a \text{ E_DICT};
\]

**Description**  
\text{e_delete} deletes an element of the domain of a dictionary. If the element is not in the domain it returns the dictionary unchanged. \text{e_delete name dict} returns a dictionary that does not associate anything with \text{name}, but otherwise associates as \text{dict}.

**SML**

\[
\text{val } \text{e_enter} : \text{string} \to 'a \to 'a \text{ E_DICT} \to 'a \text{ E_DICT};
\]

**Description**  
\text{e_enter} implements overwriting by a singleton function. \text{e_enter name value dict} returns the dictionary that associates \text{name} with \text{value}, and otherwise associates as \text{dict}.

**SML**

\[
\text{val } \text{e_extend} : \text{string} \to 'a \to 'a \text{ E_DICT} \to 'a \text{ E_DICT};
\]

**Description**  
\text{e_extend} implements extension by a singleton function, that is to say it is like \text{e_enter}. \text{e_extend name value dict} returns the dictionary that associates \text{name} with \text{value}, and otherwise associates as \text{dict}. It fails if \text{name} is already in the domain of \text{dict}.

**Errors**  
1014 0 is already in dictionary

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
SML

val e_flatten : 'a E.DICT → 'a S.DICT;

Description e_flatten converts an efficient dictionary into a simple one. The result will contain no duplicates, but will be in no useful order.

SML

val e_lookup : string → 'a E.DICT → 'a OPT

Description e_lookup implements application (of the dictionary viewed as a partial function). e_lookup name dict returns the value that dict associates with name.

SML

val e_merge : 'a E.DICT → 'a E.DICT → 'a E.DICT;

Description e_merge extends one efficient dictionary by another. The dictionary e_merge dict1 dict2 will associate a name with the value that either dict1 or dict2 associates it with.

Failure Will get the e_extend failure message if an element is common to the domains of both dictionaries.

SML

val e_stats : 'a E.DICT → {height : int, nentries : int, nnodes : int, sumweights : int};

Description e_stats dict returns statistics about the internals of the efficient dictionary dict. Efficient dictionaries are currently represented as binary trees whose non-leaf nodes each carry a simple dictionary of entries (in case of collision of hash values). The statistics currently returned are the height of the tree, the number of entries, the number of nodes and the sum over all entries of the depth of the entries (i.e., the sum of the number of entries per node weighted by node-depth).

SML

val initial_e_dict : 'a E.DICT;

Description The empty dictionary, which gives a starting point for the use of the efficient dictionary functions. It does not associate a value with any name.

SML

val list_e_enter : 'a E.DICT → 'a S.DICT → 'a E.DICT;

Description list_e_enter extends an efficient dictionary by overwriting with entries from a simple one. That is, for each association in the simple dictionary an e_enter is executed on the efficient dictionary.

SML

val list_e_merge : 'a E.DICT → 'a S.DICT → 'a E.DICT;

Description list_e_merge extends an efficient dictionary by merging with entries from a simple one. That is, for each association in the simple dictionary an e_extend is executed on the efficient dictionary.

Failure Will get the e_extend failure message if an element is common to the domains of both dictionaries.
2.9 Sorting

\begin{verbatim}
signature Sort = sig

include Order;

Description This provides an efficient sort utility package. For historical reasons it includes
the structure Order.
\end{verbatim}

\begin{verbatim}
val sort : 'a ORDER -> 'a list -> 'a list
val merge : 'a ORDER -> 'a list -> 'a list

Description sort sorts a list and merge merges two lists assumed already to be sorted. Both
functions are parametrised by an ordering function of type 'a ORDER, i.e., 'a -> 'a -> int.
The integer, say n returned by an application of this function, say f a_1 a_2, is interpreted as
follows:
n < 0 a_2 is to come after a_1 (i.e. the arguments are in order).
n > 0 a_2 is to come before a_1 (i.e. the arguments are out of order).
n = 0 a_2 is to be taken as equal to a_1

Sorting eliminates duplicate elements in the sense of the equality test given by the ordering.
Merging includes just one copy of an element that occurs once in each of its arguments in the
result. The result of merging unsorted lists is unspecified; in particular, the result is unspecified
if there is duplication within one of the lists.

Example
To sort a list of integers, ilist in ascending order:

\texttt{sort (curry (op \-)) ilist}

or

\texttt{sort int\_order ilist}

See Also For convenient ways of constructing orderings, see, e.g. string\_order and list\_order.
\end{verbatim}
2.10 Sparse Arrays

SML
signature SparseArray = sig

Description  This is the signature of a structure implementing sparse arrays (i.e. imperative data structures representing finite partial functions on the integers). The sparse arrays also give an efficient means for handling dense (i.e. contiguous) arrays whose size varies. To facilitate their use for such dynamically sized arrays, the sparse arrays have lower and upper bound attributes which gives the smallest and largest indices into the array which identify an occupied cell.

The design of the structure is an adaptation of the library structure Array implementing fixed length arrays.

SML
type 'a SPARSE_ARRAY;

Description  This is the type of a sparse array with entries of type 'a.

SML
val array : int -> 'a SPARSE_ARRAY;

Description  This function creates an empty sparse array. The parameter indicates the length of an internal data structure used to represent the array. For a contiguous array or for a sparsely filled array with a random distribution of occupied cells, the average access time for an element will be proportional to \( n/l \) where \( n \) is the number of occupied cells and \( l \) is this length.

Errors
1102 The length parameter must be positive

SML
val lindex : 'a SPARSE_ARRAY -> int
val uindex : 'a SPARSE_ARRAY -> int

Description  lbound(array) (resp. ubound(array)) returns the smallest (resp. largest) index of an occupied cell in the sparse array array. An exception is raised if the array is empty.

Errors
1103 the array is empty

SML
val scratch : 'a SPARSE_ARRAY -> unit;

Description  scratch array empties all cells in the sparse array array.

SML
val sub_opt : ('a SPARSE_ARRAY * int) -> 'a OPT

Description  sub(array, i) returns Value a, where a is the occupant of the i-th cell of the sparse array array. If the cell is unoccupied it returns Nil.

SML
val sub : ('a SPARSE_ARRAY * int) -> 'a

Description  sub(array, i) returns the occupant of the i-th cell of the sparse array array. An exception is raised if the cell is not occupied.

Errors
1101 Cell with index ?0 is empty
val update : ('_a SPARSE_ARRAY * int * '_a) -> unit;

Description  update(array, i, a) makes a the occupant of the i-th cell of the sparse array array. The cell need not previously have been occupied (indeed, update is the only means by which cells become occupied).
2.11 Dynamic Arrays

SML

signature DynamicArray = sig

Description This is the signature of a structure implementing dynamic arrays with 0-based indexing (i.e. imperative data structures representing finite partial functions on the integers, whose range is an interval \(1 \ldots n\)). The implementation gives constant access time. The design of the structure is an adaptation of the library structure Array implementing fixed length arrays.

SML

type 'a DYNAMIC ARRAY;

Description This is the type of a dynamic array with entries of type 'a.

SML

val array : int -> 'a DYNAMIC ARRAY;

Description This function creates an empty dynamic array. The parameter indicates the length of an internal data structure used to represent the initial size of the array. The average access time for an element will be constant — the underlying array structure is grown as necessary.

Errors
1301 The initial size parameter must be positive

SML

val scratch : 'a DYNAMIC ARRAY -> unit;

Description scratch array empties all cells in the sparse array array and reduces the underlying data structure to the initial length specified when the array was first created.

SML

val sub_opt : ('a DYNAMIC ARRAY * int) -> 'a OPT

Description sub(array, i) returns Value a, where a is the occupant of the \(i\)-th cell of the dynamic array array. If the cell is unoccupied it returns Nil.

SML

val sub : ('a DYNAMIC ARRAY * int) -> 'a

Description sub(array, i) returns the occupant of the \(i\)-th cell of the dynamic array array. An exception is raised if the cell is not occupied.

Errors
1101 Cell with index \(?0\) is empty
1303 Index \(?0\) is out of range

SML

val uindex : 'a DYNAMIC ARRAY -> int

Description lbound(array) the largest index of an occupied cell in the dynamic array array or \(\sim 1\) if no cells in the array are occupied.

SML

val update : ('a DYNAMIC ARRAY * int * 'a) -> unit;

Description update(array, i, a) makes a the occupant of the \(i\)-th cell of the sparse array array. The cell need not previously have been occupied (indeed, update is the only means by which cells become occupied).
2.12 Arbitrary Magnitude Integer Arithmetic

SML

signature Integer = sig
  eqtype INTEGER;
  val idiv : INTEGER * INTEGER -> INTEGER;
  val imod : INTEGER * INTEGER -> INTEGER;
  val @* : INTEGER * INTEGER -> INTEGER;
  val @+ : INTEGER * INTEGER -> INTEGER;
  val @- : INTEGER * INTEGER -> INTEGER;
  val @~ : INTEGER -> INTEGER;
  val @s : INTEGER -> INTEGER;
  val @< : INTEGER * INTEGER -> bool;
  val @> : INTEGER * INTEGER -> bool;
  val @= : INTEGER * INTEGER -> bool;
  val integer_of_string : string -> INTEGER;
  val string_of_integer : INTEGER -> string;
  val int_of_integer : INTEGER -> int;
  val integer_of_int : int -> INTEGER;
  val natural_of_string : string -> INTEGER;
  val zero : INTEGER;
  val one : INTEGER;
  val string_of_float : INTEGER * INTEGER * INTEGER -> string;
  val integer_order : INTEGER -> INTEGER -> int;

Description  This is the signature of an open structure providing arithmetic on integers of arbitrary magnitude. It is used to support HOL natural numbers and other object language numeric types. The names of the usual arithmetic operators are decorated with an initial i or @ as appropriate. The string conversions work with signed decimal string representations. Either '\-' or '\~' may be used for unary negation and a leading '+' is also allowed. @@ is an abbreviation for integer_of_string. natural_of_string is a converter for non-negative numbers (it has the same error cases as nat_of_string).

string_of_float interprets a triple \((x, p, e)\) as a floating point number with value \(x \times 10^{e-p}\) and converts the triple into its string representation.

integer_order implements the ordering of the integers in the form used by sort, q.v.

Errors

1201 the divisor is zero
1202 an empty string is not a valid decimal number
1203 the string '\?0' is not a valid decimal number
1204 the conversion would overflow
2.13 Order-preserving Efficient Dictionary

SML

```sml
val initial_oe_dict : 'a OE.DICT;
val oe_lookup : string -> 'a OE.DICT -> 'a OPT;
val oe_enter : string -> 'a OE.DICT -> 'a OE.DICT;
val oe_extend : string -> 'a OE.DICT -> 'a OE.DICT;
val oe_delete : string -> 'a OE.DICT -> 'a OE.DICT;
val oe_key_lookup : E.KEY -> 'a OE.DICT -> 'a OPT;
val oe_key_enter : E.KEY -> 'a OE.DICT -> 'a OE.DICT;
val oe_key_extend : E.KEY -> 'a OE.DICT -> 'a OE.DICT;
val oe_key_delete : E.KEY -> 'a OE.DICT -> 'a OE.DICT;
val oe_flatten : 'a OE.DICT -> 'a S.DICT;
val oe_key_flatten : 'a OE.DICT -> ('a * E.KEY) S.DICT;
val e_dict_of_oe_dict : 'a OE.DICT -> 'a E.DICT;
val list_oe_merge : 'a OE.DICT -> 'a S.DICT -> 'a OE.DICT;
val oe_merge : 'a OE.DICT -> 'a OE.DICT -> 'a OE.DICT;
```

**Description**  
This type and associated access functions implement order-preserving efficient dictionaries. The functions have exactly the same effect as the corresponding functions `e_lookup`, `e_enter` etc., qv., for the type `E.DICT` except that `se_flatten` returns a list which preserves the order in which entries were made (last-in, first-out). If an entry is updated (rather than added) by `oe_key_enter` or `oe_enter`, the updated entry appears in its original position.

`list_oe_merge` enters the list of items in the second argument into the dictionary given as its first argument in tail-first (right-to-left) order.

**Failure**  
The failures are exactly as for the corresponding `E.DICT` functions. In particular, the area names in error messages are, e.g., “`e_lookup`” rather than “`oe_lookup`” etc.
## 2.14 Compatibility with SML’90

**SML**

```sml
signature BasicIO = sig
  type instream;
  type outstream;
  exception Io of {cause:exn, function:string, name:string}
  val close_in : instream -> unit
  val close_out : outstream -> unit
  val end_of_stream : instream -> bool
  val input : instream * int -> string
  val lookahead : instream -> string
  val open_in : string -> instream
  val open_out : string -> outstream
  val output : outstream * string -> unit
  val std_in : instream
  val std_out : outstream
end;

signature ExtendedIO = sig
  type instream;
  type outstream;
  val can_input : instream * int -> bool
  val flush_out : outstream -> unit
  val open_append : string -> outstream
  val is_term_in : instream -> bool
  val input_line : instream -> string
  val is_term_out : outstream -> bool
  val system : string -> bool
  val get_env : string -> string
  val std_err : outstream
end;
```

**Description**  These are the signatures of the structures that implement SML’90-style I/O. BasicIO is open. ExtendedIO is not.

ExtendedIO differs from the the original SML’90 in several respects:

- It provides `system` instead of `execute` (which cannot be implemented cleanly on UNIX implementation sof the SML’97 standard basis library, since the SML’90 signature does not give an interface for the caller to reap the executed process).

- It provides `std.err`, which was not in the SML’90 library at all (and is the same as TextIO.stdErr in the SML’97 standard basis library).

- It provides `get.env` which is the UNIX `get.env` with non-existent environment variables returning an empty string.
signature PPArray = sig
    exception Subscript
    type 'a array
    val arrayoflist : 'a list -> 'a array
    val array : int * 'a array
    val length : 'a array -> int
    val sub : 'a array * int -> 'a
    val tabulate : int * (int -> 'a) -> 'a array
    val update : 'a array * int * 'a -> unit
end;

Description  This is the signature of a structure that provides an array datatype compatible with the ProofPower code (independent of the underlying compiler).

signature PPString = sig
    val implode : string list -> string;
    val explode : string -> string list;
    exception Ord;
    val ord : string -> int;
    val chr : int -> string;
    val string_of_exn : exn -> string;
end;

Description  This is the signature of an open structure that provides string functions compatible with the ProofPower code (independent of the underlying compiler).

signature PPVector = sig
    exception Subscript
    exception Size
    type 'a vector
    val vector : 'a list -> 'a vector
    val length : 'a vector -> int
    val sub : 'a vector * int -> 'a
    val tabulate : int * (int -> 'a) -> 'a vector
end;

Description  This is the signature of a structure that provides a vector (read-only array) datatype compatible with the ProofPower code (independent of the underlying compiler).
SML

(*
structure SML97BasisLibrary = struct
  val explode : string -> char list; ...
  structure Array = Array; ...
end;*)

Description This is a structure containing the required structures of the Standard ML '97 Basis Library together with some functions from the '97 standard for the language that are redefined by ProofPower.

It is provided so that these structures can still be accessed when ProofPower defines a structure of the same name as a basis library structure (e.g., “Char”).

The structure SML97BasicLibrary.Prelude contains the functions from the '97 standard for the language that are redefined by ProofPower. If you open this structure and later wish to revert to the ProofPower versions of explode, hd, etc., open the structures PPString and ListUtilities.
SYSTEM FACILITIES
3.1 System Control

**SML**

```sml
signature SystemControl = sig

Description  This is the signature of the structure SystemControl.
```

**SML**

```sml
val get_flags : unit -> (string * bool) list
val get_int_controls : unit -> (string * int) list
val get_string_controls : unit -> (string * string) list
val get_controls : unit ->
  ((string * bool) list * (string * int) list * (string * string) list)

Description  These functions return the names and current values of the system flags or controls.
```

**SML**

```sml
val get_flag : string -> bool
val get_int_control : string -> int
val get_string_control : string -> string

Description  These functions are used to get the values of named control variables of the corresponding types. The parameter gives the name of the control variable.
```

**Errors**

- 2011 The name ?0 is not in use as a control variable name

**Uses**  This function is for use when adding new facilities to the HOL system which require global control variables.

**SML**

```sml
val new_flag :
  {name:string, control:bool ref, default:unit->bool, check:bool -> bool} -> unit
val new_int_control :
  {name:string, control:int ref, default:unit->int, check:int -> bool} -> unit
val new_string_control :
  {name:string, control:string ref, default:unit->string, check:string -> bool} -> unit

Description  These functions are used to introduce new named control variables of the corresponding types. The name parameter gives the name of the new control variable. The control component of the parameter gives the variable itself. The default component of the parameter is a function which is used by reset_flag, reset_int_control or reset_string_control to reset the value.

After the introduction, users may update the control using one of set_flag, set_int_control or set_string_control.

The check component of the parameter is a function to check the validity of the control values, and, if desired, to notify other code of the change in the value. When one of the control setting functions, is called, an error is reported if the check function for the control returns false when applied to the new value supplied by the caller.

The following message is raised as a warning if the control variable name is already in use. If the user elects to continue, the old control variable is renamed (by decorating it with one or more prime characters) and a new control variable is introduced with the specified name.

**Errors**

- 2010 The name ?0 is already in use as a control variable name

**Uses**  This function is for use when adding new facilities to the HOL system which require global control variables.
### System Control

| val pending_reset_control_state : unit -> unit -> unit |

**Description** This function is intended for use in system initialisation and shutdown. The binding `val prcs = pending_reset_control_state()`, defines `prcs` as a function which will set the internal state of the `SystemControl` module to the value it had at the time the binding for `prcs` was made. This is used to remember the set-up for controls introduced in a child database. Note that, to avoid problems with stateful user-defined check functions, this function does not attempt to set the values of the controls. The values are, after all, not part of the `SystemControl` module’s internal state.

| val reset_flags : unit -> unit |
| val reset_int Controls : unit -> unit |
| val reset_string Controls : unit -> unit |
| val reset Controls : unit -> unit |

**Description** These functions reset the current values of all the system flags or controls in the system, as by `reset_flag`, etc.

| val reset_flag : string -> bool |
| val reset_int_control : string -> int |
| val reset_string_control : string -> string |

**Description** These functions are used to reset the values of named control variables of the corresponding types. The parameter gives the name of the control variable. They return the previous value of the control variable.

#### Errors

- 2011 The name ?0 is not in use as a control variable name
- 2012 Value out of range for control variable ?0

**Uses** This function is for use when adding new facilities to the HOL system which require global control variables.

| val set_flags : (string * bool) list -> unit |
| val set_int Controls : (string * int) list -> unit |
| val set_string Controls : (string * string) list -> unit |
| val set Controls : (((string * bool) list * (string * int) list * (string * string) list) |

**Description** These functions set the current values of the system flags or controls named in the lists. Items that are not mentioned keep their previous values.

| val set_flag : (string * bool) -> bool |
| val set_int_control : (string * int) -> int |
| val set_string_control : (string * string) -> string |

**Description** These functions are used to change the values of named control variables of the corresponding types. The first parameter gives the name of the control variable. The second parameter gives the desired new value. They return the previous value of the control variable.

#### Errors

- 2011 The name ?0 is not in use as a control variable name
- 2012 Value out of range for control variable ?0

**Uses** This function is the standard means of changing global control variables.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
3.2 System Initialisation

```
SML
signature HOLSystem = sig

Description This is the signature of the structure HOLSystem which contains functions used
to end a HOL session and to save the results of a HOL session, as well as two access routes to the
UNIX environment to the Standard ML session.
```

```
SML
signature Initialisation = sig

Description This is the signature of the structure HOLInitialisation which contains functions
which may be used to add and test new start of session functions. These functions are for use by
those extending the system.
```

```
SML
(∗ flag: gc_messages; default false ∗)

Description The flag gc_messages can be used to turn the Standard ML compiler garbage
collector messages on and off (true meaning on) providing that facility is supported by the compiler
being used. By default, garbage collection messages are turned off.
```

```
SML
type ICL/DATABASE_INFO_TYPE;
val pp'database_info : ICL/DATABASE_INFO_TYPE;

Description Private ProofPower database information, that neither contains information useful
to the user, nor should be overwritten by the user. Note that it is not an assignable variable. It
is set by pp'set_database_info.
```

```
SML
val get_init_funs : unit -> (unit -> unit) list;
val get_save_funs : unit -> (unit -> unit) list;

Description These functions returns the list of functions that have been registered with
new_init_fun and new_save_fun. They are made visible because they are needed to save the
state in a child database.
```

```
SML
val get_shell_var : string -> string;

Description get_shell_var shvar will extract the value (as a string), if any, bound to shell
environment variable shvar. If the variable is not set the empty string will be returned.
```

```
SML
val init : unit -> unit;

Description init causes the initialisation functions in the table maintained by new_init_fun to
be executed, as they would be at the start of a session. The failure of any individual initialisation
function will not affect the attempted execution of the others.

Uses Mainly for use in testing extensions to the system.

See Also new_init_fun.

Errors
36014 Exception caught by init: ?0 (?1)
```
3.2. System Initialisation

\begin{verbatim}
val load_files : string list -> bool

Description load_files takes a list of files and compiles each file (using use_file). A message
indicating the success or failure is output as each file is processed and a summary is output when
all files have been processed. If all the files loaded without any error, load_files returns true else
it returns false.
\end{verbatim}

\begin{verbatim}
val new_init_fun : (unit -> unit) -> unit;

Description new_init_fun adds a new entry to a table of functions which are invoked at the
start of each session. At the beginning of each session, these functions are executed in turn, with
the function stored by the most recent use of new_init_fun executed last.
\end{verbatim}

\begin{verbatim}
val new_save_fun : (unit -> unit) -> unit;

Description new_save_fun adds a new entry to a table of functions which are invoked when the
state of a session is saved with save, save_and_quit or save_and_exit. The functions are executed
in turn, with the function stored by the most recent use of new_save_fun executed last.
\end{verbatim}

\begin{verbatim}
val pp\'reset_database_info : bool -> ICL\'DATABASE_INFO_TYPE -> unit;

Description This function resets the current system state to a given stored value (which will
generally be given by the variable pp\'database\_info), optionally setting the current theory. It is
not intended to be called other than in the system start-up code.
\end{verbatim}

\begin{verbatim}
val pp\'set\_database_info : unit -> unit;

Description This function sets the value of pp\'database\_info so that it describes the current
system state. The function is used by save_and_quit, and elsewhere, but should not be directly
invoked by the user.
\end{verbatim}

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
### Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030

| SML val pp'threeory_hierarchy : pp'Kernel.pp'HIERARCHY 'OPT; |
| Description  Private ProofPower database information, that neither contains information useful to the user, nor should be overwritten by the user. Note that it is not an assignable variable. |

| SML val print_banner : unit -> unit; |
| Description  Output the system startup banner. |

| SML val print_status : unit -> unit; |
| Description  This command will list: |
| 1. Current theory name; |
| 2. Current proof context name(s); |
| 3. Number of distinct goals to be achieved; |
| 4. Current subgoal label; |

| SML val quit : unit -> unit |
| val exit : int -> unit |
| Description  quit() is used to end a session with the HOL system. In interactive use, the user is warned that the database will not be saved, and asked whether they still wish to quit. The session will be quit if the response is “y”, and otherwise the user is returned to the HOL session. If it is used non-interactively, or use_terminal (q.v.) is not active, then the session will end without the database being saved. |

_exit ends the current session of the HOL system with an exit status that is the argument to exit. The exit status is available to the calling environment (e.g., as documented in the UNIX manual page for sh(1)). This facility enables the user to flag errors to the outside environment from within ProofPower. |

| See Also  save_and_quit, save_and_exit to save the database. |
3.2. System Initialisation

**SML**

```sml
val save : unit -> unit;
val save_as : string -> unit;
val save_and_quit : unit -> unit;
val save_and_exit : int -> unit;
```

**Description**

`save()` saves the user’s current work to disk using the current database name (which is initially derived from the name supplied on the command line when `ProofPower` is invoked using the supplied shell scripts). `save_as name` saves the user’s current work to disk under a new name (which becomes the current name used in subsequent calls of `save()`).

`save_and_quit()` saves the user’s current work to disk and then ends the current `ProofPower` session.

`save_and_exit` saves the user’s current work and then ends the current `ProofPower` session with an exit status that is the argument to `save_and_exit`. The exit status is available to the calling environment (e.g., as documented in the UNIX manual page for `sh(1)`). This facility enables the user to flag errors to the outside environment from within `ProofPower`.

If these function are called from another function rather than at the top-level then the function should be the last side-effecting function call before returning to the top-level, otherwise the behaviour when a new session is started on the saved state will be compiler-dependent.

The state of subsystems such as the subgoal package is preserved between sessions by system-dependent means. The compactification cache is cleared at the end of each session in order to reduce the size of the saved database.

**See Also**

`quit`, `exit`, `clear_compactification_cache`

**Errors**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>36010</td>
<td>The database name has not been set</td>
</tr>
<tr>
<td>36017</td>
<td>STATE WAS FOUND TO BE INCONSISTENT: state should not be saved</td>
</tr>
</tbody>
</table>

**Errors**

If the database cannot be saved then depending on the Standard ML compiler, the function may exit anyway, with a compiler-specific raised error message. The only warning of this is that the start of session text informs the user of the database is read-only at that point in time. This does not happen with Standard ML of New Jersey, which reports the error and then continues the session.
3.3 Warnings

SML

\[ \text{signature Warning} = \text{sig} \]

**Description**  This is the signature of the structure containing the function `warn` which is used to report recoverable error conditions. It also contains the function `comment` which is used to pass comments from the system to the user.

SML

\[
\text{val comment} : \text{string} \to \text{int} \to (\text{unit} \to \text{string}) \text{ list} \to \text{unit}
\]

**Description**  `comment` is used to report messages to the user. The parameters are exactly as for `fail` and `error` (q.v.).

**Errors**  
10010  *** COMMENT ?0 raised by ?1:

SML

\[
\text{val warn} : \text{string} \to \text{int} \to (\text{unit} \to \text{string}) \text{ list} \to \text{unit}
\]

**Description**  `warn` is used to report on recoverable error conditions. The parameters are exactly as for `fail` and `error` (q.v.). The behaviour of `warn` depends on the system control flag `ignore_warnings` and on whether or not the system is running interactively, as shown in the following table:

<table>
<thead>
<tr>
<th>interactive</th>
<th>ignore_warnings</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>false</td>
<td>the message is reported; the system asks the user whether to continue; if the answer is ‘yes’ then control returns to the caller of <code>warn</code> otherwise an exception is raised.</td>
</tr>
<tr>
<td>yes</td>
<td>true</td>
<td>the message is reported and control returns to the caller of <code>warn</code></td>
</tr>
<tr>
<td>no</td>
<td>false</td>
<td>the message is reported and an exception is raised</td>
</tr>
<tr>
<td>no</td>
<td>true</td>
<td>the message is reported and control returns to the caller of <code>warn</code></td>
</tr>
</tbody>
</table>

**Errors**  
10001  *** WARNING ?0 raised by ?1:  
10002  Do you wish to continue (y/n)?  
10003  execution of ?0 abandoned
3.4 Profiling

**SML**

```sml
signature Profiling = sig

Description The signature contains definitions that may be used to record statistics, e.g., on the number of times certain functions have been called.
```

**SML**

```sml
(* profiling – boolean flag declared by new_flag *)

Description Turns profiling on (if true) or off (if false). Default is false, but flag is true during build of ProofPower-HOL. This should be maintained via the functions of structure SystemControl.
```

**SML**

```sml
val prof : string -> unit;
val counts : string -> int OPT;
val get_stats : unit -> int S_DICT;
val set_stats : int S_DICT -> unit;
val print_stats : int S_DICT -> unit;
val init_stats : unit -> unit;

Description These five functions provide a simple database facility, associating each name with a count. A call to prof name increments, if the flag “profiling” is true, the count for name. A call to counts name returns the value of the current count for name. A call to get_stats provides the counting database as an integer dictionary, in order of first name entry into database being first in the dictionary viewed as a list. The function print_stats will provide a one line - one entry display of an integer dictionary, in particular the kind of dictionary provided by get_stats. A call to init_stats initialises all the counts to 0 (which is also the state in which the database starts). A call to set_stats will restore a statistics database to a given set of values (such as those given by get_stats). The input list must contain no duplicated names.

It is likely that the output of get_stats would be best sorted before being printed by print_stats.

Uses The intended use of this database is to profile function calls, with the implementer making one call to prof per profiled function.

Errors

1020 Input list is ill-formed
3.5 Timing

SML

|signature Timing = sig

Description The signature contains definitions that can be used to measure execution time of ML code.

SML

|datatype TIMER_UNITS = Microseconds | Milliseconds | Seconds;
type 'b TIMED = {result : 'b, time : int, units : TIMER_UNITS};
val time_app : TIMER_UNITS --> ('a --> 'b) --> 'a --> 'b TIMED;

val reset_stopwatch : unit --> unit;
val read_stopwatch : TIMER_UNITS --> int;

Description The function time_app and the associated data types TIMER_UNITS and 'a TIMED may be used to measure the execution time of a function.

In the call time_app u f x, u specifies the units in which the timing is to be measured, f is the function to be timed and x gives the argument to be passed to f. The return value gives the result of the application f x together with the time taken measured in the specified units and a reminder of what the units were.

reset_stopwatch_time and read_stopwatch_time give a way of timing sequences of ML commands. read_stopwatch_time u returns the elapsed time measured in the units specified by u since the last call of reset_stopwatch_time. read_stopwatch_time will either return a meaningless value or result in arithmetic overflow if reset_stopwatch_time has not been called in the current session.

The following points should be born in mind when using these functions:

- The times are “wall-clock” times. Garbage-collection and other overheads will be included.
- Depending on the underlying Standard ML compiler, arithmetic overflow may occur if the units are chosen inappropriately for the time period being measured.
- The functions will themselves introduce a time overhead, which may vary depending on system load and other system-dependent factors.

Errors

1021 Arithmetic overflow in time conversion
Chapter 4

INPUT AND OUTPUT

4.1 The Reader/Writer

SML

signature HOLReaderWriter = sig

Description This structure holds the HOL specific reader writer code. All the errors that the basic reader writer may raise may also be raised by the HOL reader writer.

SML

signature ReaderWriter = sig

Description File and terminal reading and writing functions.

Errors

5001 End of file found in comment
5002 End of file found in string
5003 Unknown keyword ‘?0’ after ‘?1’
5004 Unknown keyword ‘?0’
5005 Unknown extended character ‘?0’ (decimal ?1) after ‘?2’
5006 Unknown extended character ‘?0’ (decimal ?1)
5007 Unexpected symbol ‘?0’ (a symbol of type $Invalid$ has been read)
5008 Bracket mismatch, ‘?0’ found after an opening ‘?1’
5010 Unknown language requested by symbol ‘?0’ with language name ‘?1’
5011 Unknown language requested
5014 Newline found in string after ‘?0’
5030 End of file in quotation
5032 End of file found in Standard ML quotation
5036 Unknown language ‘?0’ requested

Several error messages are provided to report faults in the user’s textual input to the ICL HOL system, they may be produced from all of the routines use_file, use_string and use_terminal. Some error messages might be associated with particular routines in the ReaderWriterSupport structure but that is incidental to most users, so they are all gathered here.

SML

signature ReaderWriterSupport = sig

Description A set of declarations that allows the addition of new embedded languages. The HOL language is an example of a language embedded into a basic system that understands Standard ML with extended characters and percent keywords.

SML

(* prompt1 − boolean flag declared by new_flag, default: ";:" *)
(* prompt2 − boolean flag declared by new_flag, default: ";:#" *)

Description Prompt strings for use_terminal. String prompt1 is used when the reader writer is expecting the first line of a top-level expression, prompt2 is used for subsequent lines. The strings used here must comprise characters whose decimal codes are in the range 32 to 126 inclusive, but excluding the characters ‘Q’ (i.e., code 81) and ‘%’ (37).
### SML

\[
\begin{align*}
\text{SML} & \quad (\ast \text{RW\_diagnostics} \quad - \quad \text{integer control declared by new\_int\_control, default: 0 } \ast) \\
\text{Description} & \quad \text{For reader writer diagnostic purposes.}
\end{align*}
\]

\[
\begin{align*}
\text{SML} & \quad (\ast \text{use\_extended\_chars} \quad - \quad \text{boolean flag declared by new\_flag, default: true } \ast) \\
\text{Description} & \quad \text{Controls how the writer changes the text output from the PolyML compiler. When true extended characters are written, when false the corresponding keywords are written.}
\end{align*}
\]

\[
\begin{align*}
\text{SML} & \quad (\ast \text{use\_file\_non\_stop\_mode} \quad - \quad \text{boolean flag declared by new\_flag, default: false } \ast) \\
\text{Description} & \quad \text{Makes use\_file continue reading text (if the flag is true) or stop reading (if false) from the file after an error is reported by PolyML, this includes both syntax and execution errors. Default is to stop reading.}
\end{align*}
\]

### SML

\[
\begin{align*}
\text{datatype NAME\_CLASS} & \quad = \quad \text{Simple} \\
& \quad | \quad \text{Starting} \quad \text{of} \quad (\text{READER\_ENV} \quad \rightarrow \quad (\text{string} \quad \times \quad \text{bool}) \\
& \quad \quad \rightarrow \quad \text{string} \quad \rightarrow \quad \text{bool} \quad \rightarrow \quad \text{string list} \\
& \quad \quad \rightarrow \quad \text{string list} \quad \times \quad \text{string} \\
& \quad | \quad \text{Middle} \quad \text{of} \quad \text{string} \\
& \quad | \quad \text{Ending} \quad \text{of} \quad \text{string} \\
& \quad | \quad \text{Ignore} \\
& \quad | \quad \text{Invalid};
\end{align*}
\]

**Description** These detail the characteristics of a symbol. *Simple* is used for symbols that may be part of identifiers. *Starting, Middle* and *Ending* relate to the symbols position when embedding text of other languages. The function with *Starting* is the reader routine for the particular embedded language. Details of how this function should be written (and of it arguments) are given in the implementation document corresponding to this design. *Ignore* is used for characters that are completely ignored in the input, the extended characters for indexing come in this category. *Invalid* will cause an error message.

**See Also** Error 5007
4.1. The Reader/Writer

```sml
datatype SYMBOL = SymKnown of string * bool * PrettyNames.PRETTY_NAME
| SymUnknownChar of string
| SymUnknownKw of string * bool
| SymDoublePercent
| SymWhite of string list
| SymCharacter of string
| SymEndOfInput
;
```

**Description**  
`SymKnown` indicates a symbol declared via `add_new_symbols`, if a keyword was read the string holds the characters without the enclosing percents and the boolean is `true`. Otherwise, when an extended character is read the string holds the character and the boolean is `false`.

`SymUnknownChar` indicates an extended character not declared via `add_new_symbols`.

`SymUnknownKw` indicates a keyword not declared via `add_new_symbols` or a badly formed keyword with no closing percent sign. The boolean is `true` for a well-formed keyword.

`SymDoublePercent` indicates an empty keyword, i.e., two adjacent percent signs.

`SymWhite` indicates a non-empty sequence of formatting characters (space, tab, newline, and formfeed) which are passed as individual characters in reverse order in the string list.

`SymEndOfInput` indicates an empty string was seen.

All other cases are passed back as a single character in `SymCharacter`.

```sml
exception TooManyReadEmpties;

**Description**  
Associated with the reader functions is the exception `TooManyReadEmpties` which is raised when the parser has read the end of the file and has passed the end of file character at least 100 times to the compiler. Raising this exception signifies something has gone wrong in a reader.

```sml
structure PrettyNames : sig

**Description**  
A structure within `ReaderWriterSupport` that gathers all the information relating the extended characters and percent keywords understood by the system, together with the interfaces for interrogating and extending the information.
```
**SML**

```sml
type PRETTY_NAME (* = (string list * string OPT * NAME_CLASS) *) ;
```

**Description** Each symbol is defined in a three-element tuple of this type. Elements of the tuple are as follows. First, a non-empty list of the keywords that may be used for this symbol. These keywords exclude the enclosing percent signs. Second, an optional character for the symbol. Third, a value of datatype `NAME_CLASS` indicating the characteristics of the symbol.

The extended character field, when used, contains a single character. It may be the letter “Q” or any character with decimal code greater than 127.

**See Also** Function `add_new_symbols`, for details of the validation of values of this type.

**Example**

```sml```
``` |
(['"fn", "lambda"], Value "\lambda", Simple),
```
```

**SML**

```sml
type READER_ENV (* = {advance: unit -> string,
look_at_next: unit -> string,
push_back: string -> unit
}) *) ;
```

```sml```
```val skip_and_look_at_next : READER_ENV -> unit -> string;
```
```

**Description** All of the parsing functions in the reader writer support use the functions provided in this record type to read characters from the current input stream. Attempting to read characters by any other method will have unpredictable results. The utility function `skip_and_look_at_next` combines `advance` and `look_at_next` discarding the result of `advance`. Some applications will want to use instances of this data type to count line numbers, so pushing back newlines that have not been read is not advisable.

**SML**

```sml```
```type READER_ENV;
type READER_FUNCTION;
```
```

**Description** These types are used for reader functions for embedded languages, they are identical to the types of the same name in signature `ReaderWriterSupport`.

**See Also** Signature `ReaderWriterSupport`.
4.1. The Reader/Writer

SML

type READER_FUNCTION (*
    = READER_ENV
    -> (string * bool) (* Starting symbol *)
    -> string (* Language name *)
    -> string (* Opening text *)
    -> string list (* Left hand context *)
    -> string list *);

Description The type of the reader functions for embedded languages. The first string argument gives the symbol that started the quotation. For a keyword enclosing percent signs are omitted and the boolean is true. For an extended character the boolean is false. The second string holds the language name without the leading “%dntext%” or trailing “%cantext%”, the default language and type are expanded to give their full names, namely “HOL” or “HOL:” for the colon form. The third string is text to be included at the start of the quoted text, in the case of a HOL quotation it is the first characters that are to be read by the HOL recogniser. The string list is the left hand context of the call and must be returned with the text of the quotation added to its head.

SML

val abandon_reader_writer : unit -> unit;

Description Only meaningfully used after use_terminal has been called, when it forces an exit from that routine.

SML

val add_error_code : int * string list -> string list;
val add_error_codes : int list * string list -> string list;

Description For each error number “nn” given as the first argument an entry of the form “ERROR nn” is added to the head of the second argument. (Note that “\n” denotes a space character.)

SML

val add_general_reader : string * string * string * READER_FUNCTION -> unit;
val add_specific_reader : string * string * READER_FUNCTION -> unit;
val add_named_reader : string * string * string * READER_FUNCTION -> unit;

Description Adds reader functions to the database of known readers. The first strings give the language name, the last string holds the name of a Standard ML constructor which is to be written before the quotation when it occurs in within languages other than Standard ML. Typical values of the last string are “Lex.Term” and “Lex.Type”.

Errors

5033 Reader already present for language ‘?0’
5034 Improper reader name ‘?0’
5035 Improper reader name ‘?0’ and ‘?1’
val add_new_symbols : PRETTY_NAME list -> unit;

Description Adds details of new symbols to the data structures characterising all known symbols. There is some validation of the symbols added, the list of names should not be empty, the individual names should not contain two adjacent "Q"s and the character field should have a single character which is either a "Q" or has decimal code greater than 127.

Errors
5100 Keyword '0' has adjacent 'Q's
5101 Empty keyword list
5102 Invalid extended character '0' with keyword '1'
5103 Keyword '0' duplicated
5104 Character '0' duplicated

Errors 5100, 5101 and 5102 are issued as warnings against particular parts of the argument value, they do not prevent the other parts from being added to the data structures.

val ask_at_terminal : string -> string;

Description Asks a question at the terminal by writing out the given string then reading a single line of text which is returned. Characters are read until a newline or end of file is reached, in the first case the the returned string will end with a newline.

Any characters in the type ahead buffer of the terminal input stream before ask_at_terminal is called are read and saved (for later analysis by the normal reading functions) before the prompt is output and the response is read.

Errors
5012 Function 'use_terminal' is not active
5013 Input stream is not a terminal, nothing read

val diag_line : string -> unit;

Description diag_line outputs a string to the standard output stream followed by a new line, after translating it with translate_for_output(q.v.). It is intended for use in printing formatted terms, theorems and the like (for which the pretty printer will have included new lines within the string if necessary).

See Also diag_string, raw_diag_line.

val diag_string : string -> unit;

Description diag_string outputs a string on the standard output stream, after translating it with translate_for_output(q.v.). If the string exceeds the value of get_line_length it attempts to split the string into tokens, to fit within the line length. A token is taken to be an initial string of spaces, followed by exclusively non-space characters.

See Also list_diag_string, diag_line, raw_diag_string.

val expand_symbol : SYMBOL -> string;

Description A value of type SYMBOL is expanded into the corresponding character string.
4.1. The Reader/Writer

SML

```sml
val find_name : string -> PRETTY_NAME OPT
val find_char : string -> PRETTY_NAME OPT
```

**Description**  Finds the characteristics of a symbol based on its keyword or character. Both functions return *Nil* if the symbol is not known. They return the tuple given to `add_new_symbols` for known symbols.

SML

```sml
val general_quotation : READER_ENV
  -> (string * bool) (* Start of quotation symbol *)
  -> string   (* Opening characters *)
  -> bool    (* Context, true => in Standard ML *)
  -> string list (* Left hand context *)
  -> string list;
val specific_quotation : READER_ENV
  -> (string * bool) (* Start of quotation symbol *)
  -> string   (* Opening characters *)
  -> bool    (* Context, true => in Standard ML *)
  -> string list (* Left hand context *)
  -> string list;
val named_quotation : READER_ENV
  -> (string * bool) (* Start of quotation symbol *)
  -> string   (* Opening characters *)
  -> bool    (* Context, true => in Standard ML *)
  -> string list (* Left hand context *)
  -> string list;
```

**Description**  Process the text of a quotation and add it to the left hand context given. The opening quotation symbol has been read and is passed as the first string argument, a keyword is passed without its enclosing percent signs and the boolean is true, for an extended character the boolean is false. For general and named quotations the next characters to be read denote the language of the quotation. The boolean argument indicates whether the left hand context is in Standard ML text or in a quotation of another language.

**Errors**

- 5004 Unknown keyword ‘?0’
- 5006 Unknown extended character ‘?0’ (decimal ?1)
- 5010 Unknown language requested by symbol ‘?0’ with language name ‘?1’
- 5011 Unknown language requested
- 5030 End of file in quotation
- 5031 End of file in language name of general quotation
```sml
val get_box_braces : (READER_ENV -> string list -> string list) -> READER_ENV -> string list -> string list;
val get_curly_braces : (READER_ENV -> string list -> string list) -> READER_ENV -> string list -> string list;
val get_round_braces : (READER_ENV -> string list -> string list) -> READER_ENV -> string list -> string list;
```

**Description**  
These functions assemble a section of bracketed text. The opening bracket has been read, the first unread character is the first character within the brackets. Each routine reads text up to and including the matching closing bracket. The first argument is the parsing routine for reading items of text within the brackets. The third argument is the left hand context, which is returned with the bracketed text read by these functions, and the enclosing braces. The three pairs of brackets: "[ ]", "{ }", and "( )" are handled by functions `get_box_braces`, `get_curly_braces` and `get_round_braces` respectively.

**Errors**  
- 5008 Bracket mismatch, '?'0' found after an opening '?'1'

```sml
val get_HOL_any : READER_ENV -> string list -> string list
```

**Description**  
Assemble a section of HOL text starting with the first unread character. Text is read up to and including the first unmatched symbol of value *Ending_.* The second argument gives the left hand context, the new text read is added to that context and returned. All the errors that the basic reader writer may raise may also be raised by the HOL reader writer.

**See Also**  
Type `READER_ENV`.

```sml
val get_ML_any : READER_ENV -> string list -> string list
```

**Description**  
Assemble a section of Standard ML text starting with the first unread character. Text is read up to the first semi colon ';', unmatched closing bracket or ending keyword. A semi colon will be read and added to the returned text, a closing bracket or ending keyword is left unread for the calling routine. The syntax error where too many closing bracket are presented must be resolved by the outermost routine that calls function. The second argument gives the left hand context, the new text read is added to that context and returned.

**Errors**  
- 5003 Unknown keyword '?'0' after '?'1'
- 5005 Unknown extended character '?'0' (decimal ?1) after '?'2'
- 5007 Unexpected symbol '?'0' (a symbol of type $Invalid$ has been read)

**See Also**  
Type `READER_ENV`.
4.1. The Reader/Writer

SML
val get_ML_string : READER_ENV -> string list -> string list * int list;
val get_primed_string : READER_ENV -> string list -> string list * int list;

Description Assemble a string literal and add it to the left hand context given in the second argument. On entry the opening string quote has been read, exit when the closing string quote has been read. The goal of this routine is to form an equivalent string that can be read by a Standard ML compiler, and to defer as much validation of the string as possible to that compiler. Minimal validation is performed on escape sequences. Well-formed layout sequences (i.e., the sequence “\f.\n”) are removed, characters not recognised as formatting ones are retained and wrapped between “\ ” and “ \” for later checking by the Standard ML compiler. Extended characters are translated to their three digit decimal form.

Function get_ML_string reads a Standard ML string.

Function get_primed_string reads a string enclosed with single left-hand primes (‘). These are similar to Standard ML strings but with the meanings of the single (‘) and double (“”) prime characters interchanged.

An end of file found in the string indicates that there is no more input available, and so an immediate failure (error 5002) is raised. Error code 5014 is included to aid in understanding where errors occur, this error is not actually generated until the first non white-space character after the newline is processed. All other errors detected in strings are reported when found, additionally their numbers passed back in the result.

Errors
5002 End of file found in string
5014 Newline found in string after ‘0’

See Also Type READER_ENV.

SML
val get_percent_name : READER_ENV
-> string * PrettyNames.PRETTY_NAME OPT * bool;

Description Assemble a percent keyword and look it up in the list of known keywords. On entry the opening percent (%) is the first unread character.

The tuple returned contains: (1) the keyword read, but without the percent characters; (2) the symbols entry as given to add_new_symbols or Nil for an unknown keyword; (3) a flag set true if the keyword had a closing percent character, false otherwise, error reporting is left to the calling functions. Non-alphanumeric keywords may contain the characters “!* & $ # + - / ; : < = > ? @ \ ^ | *”

See Also Type PRETTY_NAME. Type READER_ENV. Function is_special_char.

SML
val get_use_extended_chars_flag : unit -> bool;

Description This function gives the value of the flag use_extended_chars.

SML
val HOL_lab_prod_reader : READER_FUNCTION;

Description This is the reader function for HOL labeled products. It is provided to allow specialised versions of the HOL language to be read, it is not intended to be called directly called by any user code. All the errors that the basic reader writer may raise may also be raised by the HOL reader writer.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
val HOL_reader : string -> bool -> READER_FUNCTION;

**Description**  This is the HOL reader function, its first argument is the name of the recogniser for the particular aspect of HOL to be recognised. Its second argument indicates whether this reader is considered to be used only at outermost (i.e., Standard ML’s top-level): true is used for outermost usage, false for HOL text that may be used within other expressions. This function is provided to allow specialised versions of the HOL language to be read, it is not intended to be called directly called by any user code. All the errors that the basic reader writer may raise may also be raised by the HOL reader writer.

val is_same_symbol : (string * string) -> bool

**Description**  Compare two symbols return true if they are identical, i.e., the same string. Otherwise, look up both with find_char and find_name then if they are the same symbol return true, if either is not a known symbol or they are not the same symbol return false.

val is_special_char : string -> bool;

**Description**  Checks whether the string contains a single non-alphanumeric character that is allowed in a keyword. Returns true if the argument contains exactly one of the characters listed in the description of function get_percent_name, otherwise false is returned.

See Also  Function get_percent_name.

val is_white : string -> bool

**Description**  Returns true if the string is a single white-space character, false otherwise.

val list_diag_string : string list -> unit;

**Description**  list_diag_string outputs a list of strings onto the standard output stream, after translating them with translate_for_output(q.v.). The strings in the list are concatenated (with spaces to separate them) and then output with diag_string (q.v).

See Also  diag_string, diag_line, list_raw_diag_string.

val local_error : string -> int -> (unit -> string) list -> unit;
val local_warn : string -> int -> (unit -> string) list -> unit;

**Description**  An error or warning message is written to the standard output, then the function returns. The arguments are identical in form to functions error and fail of DS/FMU/IED/DTD002.

See Also  Functions error and fail.

val look_up_general_reader : string * string -> (READER_FUNCTION * string) OPT;
val look_up_specific_reader : string -> (READER_FUNCTION * string) OPT;
val look_up_named_reader : string * string -> (READER_FUNCTION * string) OPT;

**Description**  Looks up readers in the database of known readers. The argument strings are matched against the first string given in the call of the add_..._reader, if the reader is known then the corresponding constructor string and reader function are returned. The value Nil is returned for an unknown reader.
4.1. The Reader/Writer

```sml
val read_symbol : READER_ENV -> SYMBOL;
```

**Description** Reads one or more characters and returns a value of type SYMBOL. No errors are reported by this routine. The routine reads as many characters as necessary to form a symbol. End of file is returned as a SymEndOfInput.

```sml
val reset_use_terminal : unit -> unit;
```

**Description** Restores the state that controls use_terminal to its default values. N.b., this bypasses the check that use_terminal makes on recursive calls (and so could cause a small memory leak if not used with care).

```sml
val skip_comment : READER_ENV -> unit;
```

**Description** Skip over a comment which comprises a sequence of characters within which the comment braces ‘(*’ and ‘*)’ are properly balanced. This routine is entered when the opening round bracket of the comment has been read, the opening asterisk is the first unread character. Note that Standard ML comments separate lexical items thus the calling routine should not simply discard the comment, it might replace the comment with a space character to ensure the lexical items remain separated.

**Errors**
5001 End of file found in comment

**See Also** Type READER_ENV.

```sml
val SML_recogniser : string * string * 'a * string -> 'a;
```

**Description** This routine is not intended to be directly called by any user code, it is provided to allow the quotation of Standard ML text. The context of use of this routine is that the “macro processing” of the Standard ML quotation “%<%%dntext%SML%cantext% 42 %>%” yields the text “(ReaderWriterSupport.SML_recogniser (%<%", "SML", 42 , "%>%")” which is read by the Standard ML compiler.

**Errors**
5032 End of file found in Standard ML quotation
5050 Incorrect symbols starting or ending of Standard ML quotation: ‘?0’, ‘?1’, ‘?2’

```sml
val string_of_int3 : int -> string
```

**Description** The string representation of small positive integers is needed in various places, particularly within Standard ML strings where some characters are denoted by their decimal code in three digits, preceded by a backslash. Function string_of_int3 gives a three character with leading zeros representation of small positive numbers. In general the routine PolyML.makestring cannot be used, if the value last passed to PolyML.print_depth is zero then PolyML.makestring converts numbers into three dots. The intended use of this function is in building reader writer extensions for other languages. In such places it is intended that the caller only supply suitable arguments, getting this wrong indicates something wrong in the design of the caller. The text of the message anticipates this usage.

**Errors**
5040 DESIGN ERROR: Number ?0 is too big or is negative
**val to_ML_string : string -> string**

**Description** Converts characters which are to form part of a string literal into another string which may be read by a Standard ML compiler and which has the same meaning. This is intended to form the string representation of extended characters for passing them through to a Standard ML compiler. Characters other than space, tab and newline which are outside the range 32 to 126 (decimal) inclusive are converted to their four character equivalent of a backslash followed by a three digit decimal number with leading zeroes.

**val translate_for_output : string -> string;**

**Description** Translates a string according to the macro processing rules used when outputting text. The output produced depends on the setting of the control flag `use_extended_chars`, when false the result will have no extended characters, the keyword forms will be used.

**val use_file : string -> unit;**
**val use_file1 : string -> unit;**

**Description** Both of these functions compile and execute ProofPower-ML (i.e., Standard ML extended to allow mathematical symbols) from the named file. If the file does not exist then it will read the file with the given name and suffix “.ML”, if that file does not exist it will try the suffix “.sml”.

`use_file` passes the file name string through `translate_for_output` before using it as an operating system file name which is appropriate for file names given as ProofPower-ML strings. The variant `use_file1` uses the string exactly as given.

**See Also** Error messages given with signature for `ReaderWriter`. Flag `use_file.non_stop_mode`.

**Errors**

5009 Cannot read file ‘?0‘ or ‘?0.ML‘ or ‘?0.sml‘

**val use_string : string -> unit;**

**Description** Read Standard ML with extended characters allowed, from the given string.

**See Also** Error messages given with signature for `ReaderWriter`.

**val use_terminal : unit -> unit;**

**Description** Read Standard ML with extended characters allowed, from the terminal. This routine takes over the terminal, it handles all exceptions as the outermost level of the ML system. To return to the default PolyML terminal reader use `abandon_reader_writer`.

This routine prompts to the conventions of PolyML but uses the strings “:> ” and “: # “, the PolyML prompts do not have the colon. These strings are held as the string controls ‘prompt1‘ and ‘prompt2‘ and thus may be altered.

Typing two control-D characters to the terminal prompt, or reading the end-of-file, causes the function `PolyML.quit` to be called.

**See Also** Error messages given with signature for `ReaderWriter`. Control strings‘prompt1‘ and ‘prompt2‘.
4.2 Output

```
signature SimpleOutput = sig

Description  Holds a variety of utility Standard ML functions concerned with simple output. Related facilities may be found in structure ReaderWriter. Function ask_at_terminal (q.v) provides for prompted input of text from the terminal.

Strings containing extended characters and strings derived from HOL types and terms should be passed through the ReaderWriter function translate_for_output (q.v) before being output. This allows the proper output of keywords and extended characters on both graphic and simple ASCII terminals.
```

```
(* line_length — integer control declared by new_int_control *)

Description  An integer control dictating the output’s length of line available for printing.

See Also  set_line_length, get_line_length
```

```
val format_list : ('a -> string) -> 'a list -> string -> string;

Description  format_list formatter items seperator is used to format a list of items for printing as a string, perhaps for printing. Given formatter, a function to format a single item, items, a list of items, and seperator, a string to separate elements of a multi-element list, the resulting string is the contatenation of the formatted items with interposed separators. The formatted head element of the list becomes the left hand end of the result string.

Example

format_list string_of_term [⌜T\neg,\neg x\neg,\neg a \& b\neg\rceil] "", "
--->
val it = "T\neg,\neg x\neg,\neg a \& b\neg": string
```

```
val get_line_length : unit -> int

Description  Returns current output line length.

See Also  set_line_length
```

```
val list_raw_diag_string : string list -> unit;

Description  list_raw_diag_string outputs a list of strings onto the standard output stream. The strings in the list are concatenated (with spaces to separate them) and then output with raw_diag_string (q.v).

See Also  raw_diag_string, raw_diag_line, list_diag_string.
```

```
val raw_diag_line : string -> unit;

Description  raw_diag_line outputs a string to the standard output stream followed by a new line. It is intended for use in printing formatted terms, theorems and the like (for which the pretty printer will have included new lines within the string if necessary).

See Also  raw_diag_string, diag_line.
```
val raw_diag_string : string -> unit;

Description  raw_diag_string outputs a string on the standard output stream. If the string exceeds the value of get_line_length it attempts to split the string into tokens, to fit within the line length. A token is taken to be an initial string of spaces, followed by exclusively non-space characters.

See Also  list_raw_diag_string, raw_diag_line, diag_string.

val set_line_length : int -> int

Description  Set the output line length, returning the previous line length. Default length is 80, minimum length 20.

See Also  get_line_length

Errors

1015  line length must be at least 20
4.3 HOL Lexical Analysis

**SML**

```sml
signature Lex = sig
```

**Description**
This is the signature of the structure which contains the lexical analyser for ICL HOL.

**Uses**
For use by those who wish to extend the system to handle languages other than HOL which have a similar lexical structure.

**SML**

```sml
datatype ASSOC = LeftAssoc
               | RightAssoc;
datatype FIXITY = Nonfix
                 | Binder
                 | Infix of ASSOC * int
                 | Prefix of int
                 | Postfix of int;
```

**Description**
These data types are used in the symbol table and elsewhere to give the syntactic status of a name. Nonfix means no special status. The integer components are the precedences for infix, prefix or postfix status.
Chapter 4. INPUT AND OUTPUT

SML
datatype HOL_TOKEN = HTAqTm of TERM
  | HTAqTy of TYPE
  | HTName of string
  | HTNumLit of string
  | HTString of string
  | HTChar of string
  | HTBinder of string
  | HTInOp of {name:string, is_type_op:bool,
               is_term_op:bool, prec : ASSOC * int}
  | HTPostOp of {name:string, prec : int}
  | HTPreOp of {name:string, prec : int}
  | HTAnd
  | HTBlob
  | HTColon
  | HTElse
  | HTIf
  | HTIn
  | HTLbrace
  | HTLbrack
  | HTLet
  | HTLsqbrack
  | HTRbrace
  | HTRbrack
  | HTRsqbrack
  | HTSemi
  | HTThen
  | HTVert
  | HTEos;

Description  This is the data type of the output from the HOL lexical analyser.

Uses  For use by those who wish to extend the system to handle languages other than HOL
which have a similar lexical structure.

SML
datatype INPUT = Text of string
  | String of string
  | Char of string
  | Type of TYPE
  | Term of TERM
  | Separator of string
  | Error of int;

Description  This is the data type of the input to the HOL lexical analyser.

Uses  For use by those who wish to extend the system to handle languages other than HOL
which have a similar lexical structure.
4.3. HOL Lexical Analysis

**SML**

```sml
val is_alnum : string → bool
val is_copula : string → bool
val is_digit : string → bool
val is_macro : string → bool
val is_punctuation : string → bool
val is_space : string → bool
val is_symbolic : string → bool
```

**Description**  These functions classify character strings according to their first character. They all return false if the argument is an empty string. The characters for which the various functions return true are shown in the following table.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_alnum</td>
<td>A letter or a number or the prime character ‘!’</td>
</tr>
<tr>
<td>is_copula</td>
<td>An underscore or the subscription, or superscription characters</td>
</tr>
<tr>
<td>is_digit</td>
<td>A decimal digit</td>
</tr>
<tr>
<td>is_macro</td>
<td>The character ‘%’ which introduces preprocessor macros</td>
</tr>
<tr>
<td>is_punctuation</td>
<td>‘(’, ‘), ‘{’, ‘}’, ‘[’, ‘]’, ‘;’, ‘:’, ‘,’</td>
</tr>
<tr>
<td>is_space</td>
<td>A formatting character, i.e., space, tab, newline etc.</td>
</tr>
<tr>
<td>is_symbolic</td>
<td>Any character which is not does not fall into any of the above classes</td>
</tr>
</tbody>
</table>

**SML**

```sml
val lex : (string list list) → (string → FIXITY) →
        INPUT list → HOL_TOKEN list
```

**Description**  This is the HOL lexical analyser.

The first parameter is the list of (exploded) strings which are to be taken as terminator symbols. Terminators are recognised by looking for the first match in the list, so that if one terminator is a leading substring of another the longer one must come first. No punctuation symbol should appear in a terminator. For HOL this parameter is always obtained by calling the symbol table function `get_terminators`, which maintains the list of terminators sorted in order of decreasing length.

The second parameter is used to classify names as binder, infix, prefix, postfix or nonfix.

The third parameter is the input to be lexically analysed.

**Uses**  For use by those who wish to extend the system to handle languages other than HOL which have a similar lexical structure.

**Errors**

- 15001  antiquotation not allowed after ‘$’
- 15002  ‘$’ not allowed at end of quotation
- 15003  lexical analyser or reader/writer error detected (?0)
- 15004  ill-formed keyword symbol
- 15005  ?0 is not a valid character literal (must contain exactly one character)
- 15006  error code ?0 reported by reader/writer

The last of these error messages occurs, e.g., when a keyword symbol has been entered incorrectly and is preceded by a more comprehensive error message from the reader/writer.
val num_lit_of_string : string -> (INTEGER * (INTEGER * INTEGER) OPT) OPT;

**Description**  The argument to this function should be a string representing a numeric literal (either a natural number, $N$, or a floating point number with optional, optionally signed, exponent part, $X.Y$ or $X.YeZ$. The result value is $Nil$ if the string cannot be interpreted as a numeric literal. Otherwise, the result value is $N$, or $(XY, P, 0)$ or $(XY, PZ)$, where $XY$ stands for the natural number obtained by concatenating the digit sequences $X$ and $Y$ and $P$ is the number of digits in $Y$. 
4.4 Pretty Printing

SML
signature PrettyPrinter = sig

SML
| (* Flag pp_top_level_depth : integer control, default -1 *)
| (* Flag pp_format_depth : integer control, default -1 *)

Description These control the depth to which HOL types and terms are printed. Control pp_top_level_depth applies to values printed as part of Standard ML top-level expressions. Control pp_format_depth applies to values printed by the "format..." routines. When these controls are negative, types and terms are fully printed, otherwise the value indicates how deeply the expression is printed where zero indicates suppressing the whole type or term. Suppressed types and terms, or parts thereof, are shown as three dots.

See Also Functions format_term, format_term1, format_thm, format_thm1, format_type and format_type1.

SML
| (* Flag pp_print_assumptions : boolean control, default true *)

Description This controls whether the assumptions of values of type THM are printed. The default is to print assumptions. If assumptions are not printed then each is shown as three dots.

See Also Functions format_thm and format_thm1.

SML
| end (* of signature PrettyPrinter *);

SML
| val format_term : bool -> TERM -> string list;
| val format_term1 : bool -> int -> TERM -> string list;

Description Produce a number of lines, one string per line, containing a pretty printing of the given HOL Term. The text is suitable for directly outputting via the diag_line and diag_string routines BasicIO.output. If the boolean argument is set false then the strings produced from terms whose language is the same as that of the current theory will not include the term quotation symbols, in all other cases the term quotation symbols will be included. Line width is given by the integer in format_term1, or for format_term the current line width (as maintained by set_line_length, q.v.) is used.

See Also Pretty printer controls: pp_add_brackets, pp_show_HOL_types, pp_types_on_binders and pp_let_as_lambda.

SML
| val format_thm : THM -> string list;
| val format_thm1 : int -> THM -> string list;

Description Produce a number of lines, one string per line, containing a pretty printing of the given HOL theorem. The text is suitable for directly outputting via the diag_line and diag_string routines. The theorem is printed with a comma separated list of terms for the assumptions, a turnstile and finally the term representing the conclusion. Assumptions in the same language as the conclusion are not enclosed with the term quotation symbols. Other assumptions have term quotation symbols. Line width is given by the integer in format_term1, or for format_term the current line width (as maintained by set_line_length, q.v.) is used.

See Also Pretty printer controls: pp_add_brackets, pp_show_HOL_types, pp_types_on_binders and pp_let_as_lambda.
val format_type : bool -> TYPE -> string list;
val format_type1 : bool -> int -> TYPE -> string list;

Description Produce a number of lines, one string per line, containing a pretty printing of the
given HOL type. The text is suitable for directly outputting via the diag_line and diag_string
routines If the boolean argument is set true then type quotation symbols will be included in the
returned strings, when false they are excluded. Line width is given by the integer in format_term1,
or for format_term the current line width (as maintained by set_line_length, q.v.) is used.

See Also Pretty printer control: pp_add_brackets.

val pp_init : unit -> unit;

Description Initialise the pretty printing system so that values of types TERM, TYPE and
THM will be prettily printed out as “top level” Standard ML values.

val show_type : bool -> int OPT -> OppenFormatting.OPPEN_FUNS
  -> TYPE -> unit;
val show_term : bool -> int OPT -> OppenFormatting.OPPEN_FUNS
  -> TERM -> unit;
val show_thm : int OPT -> OppenFormatting.OPPEN_FUNS
  -> THM -> unit;

Description These functions enable programming of Oppen-style pretty-printing for data types
that contain embedded types, terms and theorems.
4.5 Theory Lister

**SML**
```sml
signature Lister = sig

Description This is the signature of the structure Lister which contains functions for listing theories.
```

**SML**
```sml
signature ListerSupport = sig
data type LISTER SECTION =
  LSBanner | LSParents | LSChildren
  LConsts  | LAliases  | LUnDeclaredAliases
  LTypes   | LTypeAbbrevs | LUnDeclaredTypeAbbrevs
  LFixity  | LTerminators | LUnDeclaredTerminators
  LAXioms  | LDefs      | LSThms

LSTrailer
  | LSADString of string -> (string list * string)
  | LSADStrings of string -> (string list * string list)
  | LSADThms of string -> (string list * THM list) list
  | LSADTerms of string -> (string list * TERM list)
  | LSADTypes of string -> (string list * TYPE list)
  | LSADTables of string -> (string list * string list) list
  | LSADSection of string -> string
  | LSADNestedStructure of string -> (string * LISTER SECTION list);

Description ListerSupport is the signature of a structure containing a functions, gen_theory_lister and gen_theory_lister1 for creating variant theory listers, e.g. for languages other than ProofPower-HOL. The data type ListerSupport.LISTER SECTION controls what is listed. Each constructor of this type determines an element of the listing. The first block of constructors for the type LISTER SECTION cause sections of the listing like those produced by the HOL theory lister to be included (except that LSBanner uses the first argument to print, output, or output1 to compute the contents of the banner heading.) The second block of constructors are for creating application-defined sections of the listing and in each case the constructor takes as its operand a function which is passed the name of the theory being listed as argument. LSADSection produces a section header containing the result of applying the argument function to the theory name unless that result is an empty string, in which case it has no effect. The others are for printing (labelled) individual strings (LSADString) or columns of strings (LSADStrings), or (labelled) lists of theorems, terms, types or rows of strings (LSADTables). In each case the first component of (each element of) the result is used as a list of labels for the elements and is printed in the left margin and the second component is indented.
```

**SML**
```sml
(* sorted_listings  - flag - default false *)
(* listing_indent   - integer - control default 2 *)

Description These two system control variables influence the behaviour of the functions list_theory and output_theory which are used to generate theory listings. If sorted_listings is false (the default) then items are unsorted, otherwise they are sorted according to string_order (q.v.). listing_indent sets the indent level of the listings in terms of a number of tabstops, and its default is 2.

Errors
33052 integer control ‘0' must be greater than zero

See Also output_theory
```
val gen_theory_lister : LISTER_SECTION list ->
    { print: (string -> string) -> string -> unit,
      out: (string -> string) -> {theory: string, out_file: string} -> unit,
      out1: (string -> string) -> {theory: string, out_file: string} -> unit};
val gen_theory_lister1 : LISTER_SECTION list ->
    { print: (string -> string) -> string -> unit,
      out: (string -> string) -> {theory: string, out_file: string} -> unit,
      out1: (string -> string) -> {theory: string, out_file: string} -> unit};
end (* of structure ListerSupport *) (* of structure ListerSupport *);

Description  The functions ListerSupport.gen_theory_lister and
ListerSupport.gen_theory_lister1 are used to create customised theory listers and can also be
used to create formatted listings of other kinds.

They return a triple of functions each of which has as its first argument a function to compute
the contents of the banner line in the listing from the name of the theory name. Given such an
argument, the three components, print, out, and out1 deliver results which behave very much
like print_theory, output_theory and output_theory1, respectively, as regards where they send the
listing and whether or not they insert \LaTeX formatting controls in it, but what they put in the
listing is determined by the argument to gen_theory_lister. This argument is a list of elements of
type LISTER_SECTION, q.v.

The integer control listing_indentand the flag sorted_listings control the print of labelled lists of
theorems, terms etc. listing_indent gives the number of spaces of indent from the left margin of
the lists. If sorted_listings is true, the lists will be sorted using the concatenation of the labels as
the sort key otherwise they are printed in the order supplied.

gen_theory_lister1 is just like gen_theory_lister except that it does not check whether the theory
exists or whether it is in scope.

val output_theory1 : {theory:string, out_file:string} -> unit

Description  output_theory1{theory = thy, out_file = file} causes a listing of the theory thy to
be output to the file file. The listing is in a format suited for display on the screen or for viewing
with a text editor. The theory must be in scope, i.e. it must be the current theory or one of its
ancestors.

See Also  output_theory print_theory

Errors
33050  The theory ?0 is not in scope
33051  There is no theory called ?0
33101  i/o failure on file ?0 (?1)
33102  the theory ?0 does not exist
4.5. Theory Lister

|val output_theory : {theory:string, out_file:string} -> unit

**Description**  
$output_theory\{theory = thy, out_file = file\}$ causes a listing of the theory $thy$ to be output to the file $file$. The listing is in a format suited for printing using the ICL HOL document preparation system. The theory must be in scope, i.e. it must be the current theory or one of its ancestors.

**See Also**  
$output_theory1$, $print_theory$

**Errors**
- 33050 The theory $?0$ is not in scope
- 33051 There is no theory called $?0$
- 33101 i/o failure on file $?0$ (?)
- 33102 the theory $?0$ does not exist

|val print_theory : string -> unit

**Description**  
$print_theory\ thy$ causes a listing of the theory $thy$ to be written to the standard output. The listing is in a format suited for display on the screen. The theory must be in scope, i.e. it must be the current theory or one of its ancestors.

**Errors**
- 33050 The theory $?0$ is not in scope
- 33051 There is no theory called $?0$

**See Also**  
$output_theory$, $output_theory1$
4.6 Z Theory Lister

```sml
signature ZLister = sig

Description  This is the signature of the structure ZLister which contains functions for listing ProofPower-Z theories.

```

```sml
val z_output_theory : {theory:string, out_file:string} -> unit

Description  z_output_theory{theory = thy, out_file = file} causes a listing of the theory thy to be output to the file file. The listing is in a format suited for printing using the ProofPower document preparation system. The theory must be in scope, i.e. it must be the current theory or one of its ancestors.

See Also  output_theory z_print_theory z_output_theory1

Errors  As for output_theory.
```

```sml
val z_output_theory1 : {theory:string, out_file:string} -> unit

Description  z_output_theory1{theory = thy, out_file = file} causes a listing of the theory thy to be output to the file file. The listing is in a format suited for display on the screen or for viewing with a text editor. The theory must be in scope, i.e. it must be the current theory or one of its ancestors.

See Also  output_theory1 z_print_theory z_output_theory

Errors  As for output_theory1.
```

```sml
val z_print_fixity : string -> unit

Description  If id has been defined as an infix operator, or other kind of fancy-fix symbol, z_print_fixity id prints out a Z fixity paragraph showing the template or templates in which id appears.

Errors  65100 there are no fixity paragraphs in scope containing ?0
```

```sml
val z_print_theory : string -> unit

Description  z_print_theory thy causes a listing of the ProofPower-Z theory thy to be written to the standard output. The listing is in a format suited for display on the screen. The theory must be in scope, i.e. it must be the current theory or one of its ancestors.

Errors  As for print_theory.

See Also  print_theory z_output_theory z_output_theory1
```
5.1 Syntactic Manipulations

It should be noted that the functions documented in this section are drawn from two signatures, TypesAndTerms and icl’ TypesAndTerms.

Since the former includes the latter, an object available under the name xxx, say, or in full TypesAndTerms.xxx, is also available under the name icl’ TypesAndTerms.xxx. It is the intention that users should not access objects by names of the form icl’ TypesAndTerms.xxx. In practice, since the structure TypesAndTerms is open, the unqualified name xxx will do unless you have redefined the name xxx.

**SML**

| signature pp’TypesAndTerms = sig
| Description This provides the type of HOL types: TYPE, of HOL terms: TERM, and some functions upon them. A user should access all the elements of this signature through signature DerivedTerms (q.v).

**SML**

| signature TypesAndTerms = sig
| Description This provides various functions on derived TERMS, which are not considered necessary to create the abstract data type THM. It also contains, by inclusion, the types, and functions on the types TERM and TYPE from structure pp’TypesAndTerms(q.v.).

**SML**

| datatype DEST_SIMPLE_TYPE =
| Var_type of string
| C_type of (string * TYPE list);
| Description This is the type of simple destroyed types, related to the type TYPE by dest_simple_type(q.v) and mk_simple_type(q.v.). The value constructors correspond to type variables and compound types.

**SML**

| datatype DEST_SIMPLE_TERM =
| Var of string * TYPE
| Const of string * TYPE
| App of TERM * TERM
| Simple of TERM * TERM;
| Description This is the simple type of destroyed terms, related to the type TERM by dest_simple_term(q.v) and mk_simple_term(q.v.). The four value constructors represented destroyed variables, constants, applications and simple λ-abstractions respectively.

**Uses** In writing pattern-matching functions upon HOL terms.

**See Also** DEST_TERM.
**datatype** DEST_TERM = DVar of string * TYPE |
   DConst of string * TYPE |
   DApp of TERM * TERM |
   Dλ of TERM * TERM |
   DEq of TERM * TERM |
   D⇒ of TERM * TERM |
   DT |
   DF |
   D¬ of TERM |
   DPair of TERM * TERM |
   D∧ of TERM * TERM |
   D∨ of TERM * TERM |
   D⇔ of TERM * TERM |
   DLet of ((TERM * TERM) list * TERM) |
   DEnumSet of TERM list |
   D∅ of TYPE |
   DSetComp of TERM * TERM |
   DList of TERM list |
   DEmptyList of TYPE |
   D∀ of TERM * TERM |
   D∃ of TERM * TERM |
   D∃_1 of TERM * TERM |
   Dε of TERM * TERM |
   DIf of (TERM * TERM * TERM) |
   DN of INTEGER |
   DFloat of INTEGER * INTEGER * INTEGER |
   DChar of string |
   DString of string;

**Description**  This type is that of a term destroyed using the appropriate derived destructor functions (e.g. dest_eq) as well as the primitive ones. The type given to D∅ and DEmptyList is the type of an element of the associated set or list. The type is related to TERM by mk_term (q.v.) and dest_term (q.v)

**See Also**  DEST_SIMPLE_TERM

**eqtype** TERM;

**Description**  This is the type of well-formed HOL terms. Objects of this type are manipulated by term constructor, destructor and recogniser functions, such as mk_app, dest_λ and is_var.

**eqtype** TYPE;

**Description**  All HOL terms will be “typed”, by associating them with an object of type TYPE.

A type may either be a type variable or a compound type.

This is not an equality type (i.e. = cannot be used in tests for equality - see =: instead.).

**val** =§ : (TERM * TERM) -> bool;

**Description**  This is the (infix) equality test for HOL terms. It is retained for backwards compatibility — the type of HOL terms is now an equality type.

Instead of equality it is often preferable to test for α-convertibility, using ~=$
5.1. Syntactic Manipulations

SML
\[ \text{val } \doteq : (\text{TYPE} \times \text{TYPE}) \rightarrow \text{bool} \]

**Description** This is the (infix) equality test for HOL types. It is retained for backwards compatibility — the type of HOL types is now an equality type.

SML
\[ \text{val } \text{bin}_\text{bool}_\text{op} : \text{string} \rightarrow \text{TYPE} \rightarrow \text{TYPE} \rightarrow \text{TERM}; \]

**Description** Returns a constant with the given name, and type
\[ \&: \text{BOOL} \rightarrow \text{BOOL} \rightarrow \text{BOOL} \]
The type arguments are dummies, present only to make the function have an acceptable signature for certain other functions.

SML
\[ \text{val } \text{BOOL} : \text{TYPE}; \]

**Description** The HOL type of truth values:

**Definition**
\[ \text{val } \text{BOOL} = \&: \text{BOOL} \]

**See Also** Theory “min”.

SML
\[ \text{val } \text{CHAR} : \text{TYPE}; \]

**Description** This is the HOL type of single characters.

**Definition**
\[ \text{val } \text{CHAR} = \&: \text{CHAR} \]

**See Also** Theory “char”.

SML
\[ \text{val } \text{dest}_\text{app} : \text{TERM} \rightarrow (\text{TERM} \times \text{TERM}); \]

**Description** Destroys a function application into the function and argument. Note that many derived term constructs, e.g. all quantifications, are also applications.

**Definition**
\[
\begin{align*}
\text{dest}_\text{app} \; \&(f \; t) &= (\&f, \&t) \\
\text{dest}_\text{app} \; \&(\forall x \cdot t) &= (\&\forall x, \lambda x \cdot t)
\end{align*}
\]

**Errors**
\[ 3010 \quad \text{?0 is not of form: } \&(t1 \; t2) \]

SML
\[ \text{val } \text{dest}_\text{binder} : \text{string} \rightarrow \text{int} \rightarrow \text{string} \rightarrow \text{TERM} \rightarrow \text{TERM} \rightarrow \text{TERM} \times \text{TERM}; \]

**Description** A generic method of implementing binder destructor functions:

**Definition**
\[
\begin{align*}
\text{dest}_\text{binder} \; \text{area} \; \text{msg} \; \text{binder}_\text{nm} \; \&(\lambda \text{varstruct} \cdot \text{body}) &= \\
(\&\text{varstruct}, \&\text{body})
\end{align*}
\]

where \( \text{binder} \) is a constant whose name is \( \text{binder}_\text{nm} \). The \( \text{varstruct} \) may be any allowed variable structure.

**See Also** \text{dest}_\text{simple}_\text{binder}

**Failure** If the term cannot be destroyed, then the error will be from \( \text{area} \), with a message indexed by \( \text{msg} \).
### dest_bin_op: string \to int \to string \to TERM \to (TERM \& TERM);

**Description** dest_bin_op area msg rator_nrm term first assumes that term is of the form `rator t1 t2`, where rator is a constant with name rator_nrm, and attempts to return the pair \((t1, t2)\).

**Example**

```
dest_bin_op "dest_\&" 4032 "\&a \& b\&" = ("a\&", "b\&")
```

If the function fails it will fail with message msg, area area and with the string form of term.

### dest_char: TERM \to string;

**Description** Destroy a character literal.

**Example**

```
dest_char "'a" = "a"
```

**Errors**

3024 \?0 is not a character literal

### dest_const: TERM \to (string \& TYPE);

**Description** This destroys a constant into its name and type.

**Errors**

3009 \?0 is not a constant

### dest_ctype: TYPE \to string \& TYPE list;

**Description** Extract the components of a compound type.

**Definition**

```
dest_ctype ::(ty1,ty2,...)tc\& = ("tc\&", \[tc1\&;tc2\&;\ldots\])
dest_ctype ::ty tc\& = ("tc\&", \[tc\&;\])
dest_ctype ::tc\& = ("tc\&", \[\])
```

**Errors**

3001 \?0 is not a compound type

### dest_empty_list: TERM \to TYPE;

**Description** A derived term destructor function for empty lists.

**Definition**

```
dest_empty_list ::[]:ty LIST\& = ::ty\&
```

**Errors**

4034 \?0 is not of form: ::[]\&

### dest_enum_set: TERM \to (TERM list);

**Description** A derived term destructor function for enumerated sets.

**Definition**

```
dest_enum_set ::\{a; b; \ldots\}\& = ::a\&; ::b\&; \ldots
```

**Errors**

4011 \?0 is not of form: ::\{t1, \ldots\}\&
5.1. Syntactic Manipulations

SML

```sml
val dest_eq : TERM -> (TERM * TERM);
```

Description  A derived term destructor function for equations.

Definition

\[
\begin{align*}
\text{dest_eq } & (a = b) = (a, b) \\
\text{dest_eq } & (a \Leftrightarrow b) = (a, b)
\end{align*}
\]

Errors

3014  ?0 is not of form: t = u

SML

```sml
val dest_float : TERM -> INTEGER * INTEGER * INTEGER;
```

Description  Destroy a floating point literal.

Definition

\[
\begin{align*}
\text{dest_float } & (XX.YY) = (x, 0, 0) \\
\text{dest_float } & (XX.YYeZZ) = (x, p, 0) \\
\text{dest_float } & (XX.YYeZZ) = (x, p, z)
\end{align*}
\]

where \(x\) is the natural number with decimal representation \(XX.YY\), \(p\) is the number of digits after the point in \(XX.YY\) and \(z\) is the integer represented by \(ZZ\) (with \(p = z = 0\) in the first case and \(z = 0\) in the second).

Errors

4042  ?0 is not a floating point literal

SML

```sml
val dest_f : TERM -> unit;
```

Description  This will return () if given the term \(F\), and otherwise fail.

Errors

4037  ?0 is not: F

SML

```sml
val dest_if : TERM -> (TERM * TERM * TERM);
```

Description  Destroy a conditional.

Definition

\[
\begin{align*}
\text{dest_if } & (c \text{ then } y \text{ else } n) = (c, y, n)
\end{align*}
\]

Errors

4006  ?0 is not of form: if c then y else n

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
SML
\texttt{val dest let} : \texttt{TERM} \rightarrow ((\texttt{TERM} \ast \texttt{TERM})\texttt{list} \ast \texttt{TERM});

\textbf{Description} A derived term destructor function for \textit{let}-terms. See \texttt{mk let} for details of format. The distinction between a local function definition, and a variable structure bound to an abstraction is lost, with both being destroyed to the second form.

\textbf{Example}
\[
\begin{align*}
\text{dest let}(\text{mk let}([([f \, x], \, y^\circ]])_\text{bdy}) &= (([f^\circ], \, \lambda \, x \bullet \, y^\circ)_\text{bdy}) \\
\text{dest let}(\text{mk let}([([f \, x \bullet \, y^\circ])_\text{bdy}]) &= (([f^\circ], \, \lambda \, x \bullet \, y^\circ)_\text{bdy})
\end{align*}
\]

\textbf{Errors}
4009 \, ?0 is not of form: \textit{let} ... in ...

\textit{dest let} \, (\text{mk let}([], \text{term})) will actually fail (unless \text{term} is already a \textit{let}-term), as apply \texttt{mk let} to ([], \text{term}) will just return \text{term}.

SML
\texttt{val dest list} : \texttt{TERM} \rightarrow (\texttt{TERM} \texttt{list});

\textbf{Description} A derived term destructor function for list-terms.

\textbf{Definition}
\[
\begin{align*}
\text{dest list} \, [\text{a; b; ...}] &= [\, \text{a}\,; \, \text{b}\,; \, ...] \\
\text{dest list} \, [] &= []
\end{align*}
\]

\textbf{Errors}
4015 \, ?0 is not of form: \textit{[t1,...]}

SML
\texttt{val dest mon op} : \texttt{string} \rightarrow \texttt{int} \rightarrow \texttt{string} \rightarrow \texttt{TERM} \rightarrow \texttt{TERM};

\textbf{Description} \texttt{dest mon op area msg rator nm term} assumes that \texttt{term} is of the form \text{rator} \, \texttt{t}, where \text{rator} is a constant with name \text{rator nm}, and the function attempts to return \texttt{t}.

\textbf{Example}
\[
\text{dest mon op} \, "\text{dest}^\circ" \text{ 4029} \, "\text{¬}" \, \text{¬} \, \text{t}^\circ = \, \text{¬} \, \text{t}^\circ
\]

\textbf{Failure} The failure message for failing to destroy the term will be from \texttt{area}, and will have the text indexed by \texttt{msg}, and will have as argument the string form of \texttt{term}.

SML
\texttt{val dest multi \,¬} : \texttt{TERM} \rightarrow (\texttt{int} \ast \texttt{TERM});

\textbf{Description} \texttt{dest multi \,¬} \, \texttt{t} will strip \,\texttt{¬} from \,\texttt{t}, returning the number of times, as well as the result. It will return \,(0, \, \texttt{t}) if \,\texttt{t} is either not boolean, or has no negations.

\textbf{Example}
\[
\text{dest multi \,¬} \, \text{¬}(\,\text{¬} \, \texttt{T}^\circ) = (\, \text{2,} \, \texttt{T}^\circ)
\]

SML
\texttt{val dest pair} : \texttt{TERM} \rightarrow (\texttt{TERM} \ast \texttt{TERM});

\textbf{Description} A derived term destructor function for pairs.

\textbf{Definition}
\[
\text{dest pair} \, ([t1, \, t2])^\circ = ([t1^\circ, \, \texttt{t2}^\circ]
\]

\textbf{Errors}
4003 \, ?0 is not of form: \textit{([t1,t2])}

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
5.1. Syntactic Manipulations

**SML**
```
val dest_set_comp : TERM -> (TERM * TERM);
```

**Description**  A derived term destructor function for set comprehensions.

**Example**
```
dest_set_comp "\{ x \mid x > 5\} = (\langle x \rangle, \langle x > 5\rangle)
```

**Errors**
```
4013  ?0 is not of form: "\{ v \mid p\}
```

**SML**
```
val dest_simple_binder : string -> int -> string -> TERM -> TERM * TERM;
```

**Description**  Executing `dest_simple_binder area msg binder_nm \(\lambda\ var \bullet\ body\)` will give \(\langle var\rangle, \langle body\rangle\).

**Example**
```
dest_simple_binder "dest_simple_\forall" 3032 "\forall x \bullet\ t = (\langle x\rangle, \langle t\rangle)
```

**See Also**  `dest_binder`

**Failure**  If the term cannot be destroyed, then the error will be from `area`, with a message indexed by `msg`, and argument the string form of `term`.

**SML**
```
val dest_simple_term : TERM -> DEST_SIMPLE_TERM;
```

**Description**  An injective function, that destroys a term, returning its top-level structure, and the associated constituent parts.

**See Also**  `DEST_SIMPLE_TERM`

**SML**
```
val dest_simple_type : TYPE -> DEST_SIMPLE_TYPE;
```

**Description**  This function destroys a HOL type into something of type `SIMPLE_DEST_TYPE` (q.v).

**SML**
```
val dest_simple_\forall : TERM -> (TERM * TERM);
```

**Description**  A derived term destructor function for \(\forall\)-terms. It cannot destroy paired abstraction \(\forall\)-terms, being the inverse of `mk_simple_\forall`.

**Definition**
```
dest_simple_\forall \forall var \bullet body = (\langle var\rangle, \langle body\rangle)
```

**See Also**  `dest_\forall`

**Errors**
```
3032  ?0 is not of form: \forall var \bullet body
```

**SML**
```
val dest_simple_\exists_1 : TERM -> (TERM * TERM);
```

**Description**  A derived term destructor function for simply abstracted \(\exists_1\)-terms. It may destroy only simple abstraction \(\exists_1\)-terms, being the inverse of `mk_simple_\exists_1`.

**Definition**
```
dest_simple_\exists_1 \exists_1 var \bullet body = (\langle var\rangle, \langle body\rangle)
```

**Errors**
```
4019  ?0 is not of form: \exists_1 var \bullet t
```

**See Also**  `dest_\exists_1`
| val dest_simple_∃ : TERM -> (TERM * TERM); |
| Description | A derived term destructor function for ∃-terms. It cannot destroy paired abstraction ∃-terms, being the inverse of mk_simple_∃. |
| Definition | dest_simple_∃ (∃ var • body) = (∀ var, body) |
| See Also | dest_∃ |
| Errors | 3034 ?0 is not of form: ∀ var • body |

| val dest_simple_λ : TERM -> (TERM * TERM); |
| Description | Destroys a simple λ-abstraction. It cannot destroy paired λ-abstractions, being a inverse of mk_simple_λ. |
| Definition | dest_simple_λ (∀ λ v • t) = (∀ v, t) |
| See Also | dest_λ |
| Errors | 3011 ?0 is not of form: ∀ λ var • t |

| val dest_string : TERM -> string; |
| Description | Destroy a string literal. |
| Example | dest_string (∀ "abc") = "abc" |
| Errors | 3025 ?0 is not a string literal |

| val dest_term : TERM -> DEST_TERM |
| Description | This function returns the “best” interpretation of a term in the form of an object of type DEST_TERM. E.g. it will return DEq(1 2) rather than DComb(1 2). It will also use the paired abstraction forms of functions in preference to the simple forms, e.g., it uses dest_λ not dest_simple_λ. |
| The function assumes that the name of a constant is sufficient to identify it without checking the type, as with, e.g., dest_bin_op(q,v). |
| See Also | mk_term |

| val dest_t : TERM -> unit; |
| Description | This will return () if given the term ∀ T, and otherwise fail. |
| Errors | 4036 ?0 is not: ∀ T |
### 5.1. Syntactic Manipulations

<table>
<thead>
<tr>
<th>SML</th>
<th><code>val dest_vartype : TYPE -&gt; string;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Extract the name of a type variable:</td>
</tr>
<tr>
<td><strong>Definition</strong></td>
<td>( \text{dest_vartype} ; \vdash ; \text{’tv} \Rightarrow \text{”tv”} )</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>3019 ?0 is not a type variable</td>
</tr>
<tr>
<td>3027 STRING STORE ERROR: cannot translate internal id (?0) to string</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th><code>val dest_var : TERM -&gt; (string * TYPE);</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This destroys a term variable into its name and type.</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>3007 ?0 is not a term variable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th><code>val dest_∅ : TERM -&gt; TYPE;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>A derived term destructor function for empty enumerated sets.</td>
</tr>
<tr>
<td><strong>Definition</strong></td>
<td>( \text{dest_∅} ; \vdash \text{∅ SET} \Rightarrow \text{∅} )</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>4035 ?0 is not of form: ∅</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th><code>val dest_⇔ : TERM -&gt; (TERM * TERM);</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>A derived term destructor function for bi-implications. N.B. this may be successfully applied to boolean equalities.</td>
</tr>
<tr>
<td><strong>Definition</strong></td>
<td>( \text{dest_⇔} ; \vdash ; t1 \iff t2 \Rightarrow (\vdash t1, \vdash t2) )</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>4031 ?0 is not of form: ∪</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th><code>val dest→type : TYPE -&gt; (TYPE * TYPE);</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Extract the two constituent types of a function type.</td>
</tr>
<tr>
<td><strong>Definition</strong></td>
<td>( \text{dest→type} ; \vdash ; \text{ty1 \rightarrow ty2} \Rightarrow (\vdash ty1, \vdash ty2) )</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>3022 ?0 is not of form: ∪</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th><code>val dest∧ : TERM -&gt; (TERM * TERM);</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>A derived term destructor function for conjunctions.</td>
</tr>
<tr>
<td><strong>Definition</strong></td>
<td>( \text{dest∧} ; \vdash ; t1 \land t2 \Rightarrow (\vdash t1, \vdash t2) )</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>4032 ?0 is not of form: ∪</td>
</tr>
</tbody>
</table>
SML

val dest_∨ : TERM -> (TERM * TERM);

Description  A derived term destructor function for disjunctions.

Definition  \[ dest_\lor \; \Gamma t1 \lor t2 \; \gamma = (\Gamma t1 \; \gamma, \Gamma t2 \; \gamma) \]

Errors 4027  ?0 is not of form: \( \Gamma t1 \lor t2 \; \gamma \)

SML

val dest_¬ : TERM -> TERM;

Description  A derived term destructor function for negations.

Definition  \[ dest_\neg \; \Gamma \neg t \; \gamma = \Gamma t \; \gamma \]

Errors 4029  ?0 is not of form: \( \Gamma \neg t \; \gamma \)

SML

val dest_⇒ : TERM -> (TERM * TERM);

Description  A derived term destructor function for implications, returning the antecedent and consequent.

Definition  \[ dest_\Rightarrow \; \Gamma a \Rightarrow b \; \gamma = (\Gamma a \; \gamma, \Gamma b \; \gamma) \]

Errors 3016  ?0 is not of form: \( \Gamma t \Rightarrow u \; \gamma \)

SML

val dest_∀ : TERM -> (TERM * TERM);

Description  A derived term destructor function for \( \forall \)-terms. It may destroy a paired abstraction \( \forall \)-term, being the inverse of mk_∀.

Definition  \[ dest_\forall \; \Gamma \forall \; \text{varstruct} \bullet \; \text{body} \; \gamma = (\Gamma \; \text{varstruct} \; \gamma, \Gamma \; \text{body} \; \gamma) \]

Errors 4017  ?0 is not of form: \( \Gamma \forall \; \text{vs} \bullet \; t \; \gamma \)

SML

val dest_∃_1 : TERM -> (TERM * TERM);

Description  A derived term destructor function for \( \exists_1 \)-terms. It may destroy paired abstraction \( \exists_1 \)-terms, being the inverse of mk_∃_1.

Definition  \[ dest_\exists_1 \; \Gamma \exists_1 \; \text{varstruct} \bullet \; \text{body} \; \gamma = (\Gamma \; \text{varstruct} \; \gamma, \Gamma \; \text{body} \; \gamma) \]

Errors 4021  ?0 is not of form: \( \Gamma \exists_1 \; \text{vs} \bullet \; t \; \gamma \)

See Also  dest_simple_∃_1
5.1. Syntactic Manipulations

SML
\[ \text{val dest}_\exists : \text{TERM} \to (\text{TERM} \times \text{TERM}); \]

**Description**  
A derived term destructor function for $\exists$-terms. It may destroy paired abstraction $\exists$-terms, being the inverse of \text{mk}_\exists.

**Definition**  
\[ \text{dest}_\exists \quad \frac{\exists \ \text{varstruct} \cdot \text{body}}{\Rightarrow (\exists \ \text{varstruct}, \ \text{body})} \]

**Errors**  
4020  ?0 is not of form: $\exists \ vs \cdot t$

**See Also**  
\text{dest_simple}_\exists

---

SML
\[ \text{val dest}_{\times \_type} : \text{TYPE} \to (\text{TYPE} \times \text{TYPE}) \]

**Description**  
\[ \text{dest}_{\times \_type} \quad \frac{\gamma : \text{ty}_1 \times \text{ty}_2}{\Rightarrow (\gamma : \text{ty}_1, \gamma : \text{ty}_2)} \]

**Errors**  
4018  ?0 is not of the form: $\gamma : \text{ty}_1 \times \text{ty}_2$

---

SML
\[ \text{val dest}_\epsilon : \text{TERM} \to (\text{TERM} \times \text{TERM}); \]

**Description**  
A derived term destructor function for $\epsilon$-terms.

**Definition**  
\[ \text{dest}_\epsilon \quad \frac{\epsilon \ \text{varstruct} \cdot \text{body}}{\Rightarrow (\epsilon \ \text{varstruct}, \ \text{body})} \]

**Errors**  
4023  ?0 is not of form: $\epsilon \ vs \cdot t$

---

SML
\[ \text{val dest}_\lambda : \text{TERM} \to (\text{TERM} \times \text{TERM}); \]

**Description**  
Destroys a $\lambda$-abstraction. It can destroy paired $\lambda$-abstractions, being an inverse of \text{mk}_\lambda.

**Definition**  
\[ \text{dest}_\lambda \quad \frac{\lambda \ \text{vs} \cdot t}{\Rightarrow (\lambda \ \text{vs}, \ \gamma : \text{t})} \]

**See Also**  
\text{dest_simple}_\lambda

**Errors**  
4002  ?0 is not of form: $\gamma : \text{t}$

Further details of the errors will be given, before the above exceptions are raised.

---

SML
\[ \text{val dest}_N : \text{TERM} \to \text{INTEGER}; \]

**Description**  
Destroy a numeric literal.

**Example**  
\[ \text{dest}_N \quad \frac{5}{5} = 5; \]

**Errors**  
3026  ?0 is not a numeric literal

---

SML
\[ \text{val equality} : \text{TYPE} \to \text{TYPE} \to \text{TERM}; \]

**Description**  
Returns the constant $\gamma \$ = $ \gamma$ upon terms with the first type argument. The second type is a dummy argument, present only to make the function have an acceptable signature for certain other functions.
SML

val frees : TERM -> TERM list;

Description  Extract the free term variables within the term argument. The resulting variables will be in reverse order of first occurrence (for a term viewed without fixity properties, such as infix variables).

See Also  dest_frees

SML

val gen_vars : TYPE list -> TERM list -> TERM list;

Description  gen_vars ty1 tml generates a list of differently named term variables, with the types in ty1, whose names are not present within any of the terms in tml as variable names.

It will be much faster to make one call to this function with a list of types, than to make the equivalent number of individual calls.

SML

val get_variant_suffix : unit -> string;

Description  Returns the string control variant_suffix used to create variant names in string_variant (q.v.) and its relatives. The string is set by set_variant_suffix (q.v.).

SML

val inst_type : ((TYPE * TYPE) list) -> TYPE -> TYPE;

Description  inst_type alist type recursively descends through type, replacing any type variables by whatever the association list alist associates with them. If the association list does not contain a type variable found in type, then that type variable will not be changed. Replaced types are not recursively processed by this function.

Errors
3019  ?0 is not a type variable

SML

val inst : TERM list -> (TYPE * TYPE) list -> TERM -> TERM;

Description  inst avlist slist term instantiates the type variables of term with the associated types found in slist. An element of slist will be (return, tv), where tv is a type variable that is to be instantiated to return. It will rename bound variables as necessary to prevent name capture problems. It will also not allow free variables to become the same as those in the avoidance list, avlist, or to become bound.

It partially evaluates with two arguments.

Errors
3007  ?0 is not a term variable
3019  ?0 is not a type variable
3020  Internal error in type instantiation (?0 would become bound)

SML

val is_app : TERM -> bool;

Description  Return true only when the term is a function application (i.e. of form "f x"), and false otherwise: no exceptions can be raised. Note that many derived term constructs, e.g. all quantifications, are also applications. Thus is_app "∀ x • t" will return true.
5.1. Syntactic Manipulations

SML

|val is_binder : string -> TERM -> bool;

Description   is_binder binder_n m tm is true only when tm is of the form \(\langle binder(\lambda \text{vs} \bullet \text{body})\rangle\), where binder is a constant whose name is binder_n m, and vs an allowed variable structure, and false otherwise. It cannot raise an exception.

See Also    is_simple_binder

SML

|val is_bin_op : string -> TERM -> bool;

Description   is_bin_op rator_n m term returns true iff. term is of the form \(\langle rator t_1 t_2 \rangle\), and rator is a constant with name rator_n m. It cannot raise an exception.

Example      is_bin_op "\&" \(\langle a \& b \rangle\) = true

SML

|val is_char : TERM -> bool;

Description   Return true only when the term is a character literal (e.g. \(\langle 'a' \rangle\)), and false otherwise: no exceptions can be raised.

SML

|val is_const : TERM -> bool;

Description   Return true only when the term is a constant, and false otherwise: no exceptions can be raised. Note that even if the constant has not been declared, or has an inappropriate type it will still satisfy this predicate.

SML

|val is ctype : TYPE -> bool;

Description   Return true only when the term is a compound type, and false otherwise: no exceptions can be raised. If the argument isn’t a compound type then it must be a type variable.

SML

|val is_empty_list : TERM -> bool;

Description   Return true only when the term is an empty list-term, \(\langle [] \rangle\), and false otherwise: no exceptions can be raised.

SML

|val is_enum_set : TERM -> bool;

Description   Return true only when the term is an enumerated set (i.e. of form \(\langle \{a; b; \ldots\} \rangle\)), and false otherwise: no exceptions can be raised.

SML

|val is_eq : TERM -> bool;

Description   Return true only when the term is an equation (i.e. of form \(\langle a = b \rangle\) or \(\langle a \equiv b \rangle\)), and false otherwise: no exceptions can be raised.

SML

|val is_float : TERM -> bool;

Description   Return true when the term is a floating point literal, and false otherwise: no exceptions are raised.
<table>
<thead>
<tr>
<th>SML</th>
<th>val is_free_in : TERM → TERM → bool;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Description</strong> is_free_in (v) (\text{term}) returns true iff. there is a free occurrence of (v) in (\text{term}). It will raise an exception if the first argument is not a term variable.</td>
</tr>
</tbody>
</table>

| Errors | 3007 ?0 is not a term variable |

<table>
<thead>
<tr>
<th>SML</th>
<th>val is_free_var_in : (string * TYPE) → TERM → bool;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Description</strong> Given a destroyed term variable, return true only when it is free within the term supplied as a second argument, and false otherwise: no exceptions can be raised.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>val is_f : TERM → bool;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Description</strong> Return true only when the term is (⌜F : BOOL\⌝), and false otherwise: no exceptions can be raised.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>val is_if : TERM → bool;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Description</strong> Return true only when the term is a conditional (i.e. of form (⌜if\ a \ then\ b \ else\ c\⌝)), and false otherwise: no exceptions can be raised.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>val is_let : TERM → bool;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Description</strong> Return true only when the term is a let-term (i.e. of form (⌜let\ x = y\ in\ z\⌝)), and false otherwise: no exceptions can be raised.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>val is_list : TERM → bool;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Description</strong> Return true only when the term is a list-term (i.e. of form (⌜[a; b; ...]\⌝)), and false otherwise: no exceptions can be raised.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>val is_mon_op : string → TERM → bool;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Description</strong> is_mon_op (\text{rator}<em>\text{nm}) (\text{term}) returns true iff. (\text{term}) is of the form (\text{rator} \ t), where (\text{rator}) is a constant with name (\text{rator}</em>\text{nm}). It cannot raise an exception.</td>
</tr>
<tr>
<td></td>
<td><strong>Example</strong> is_mon_op &quot;¬&quot; (⌜¬\ t\⌝) = (⌜t\⌝)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>val is_pair : TERM → bool;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Description</strong> Return true only when the term is a pair (i.e. of the form (⌜(a, b)\⌝)), and false otherwise: no exceptions can be raised.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>val is_set_comp : TERM → bool;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Description</strong> Return true only when the term is a set comprehension (i.e. of form (⌜{v \mid p}\⌝)), and false otherwise: no exceptions can be raised.</td>
</tr>
</tbody>
</table>
5.1. Syntactic Manipulations

```sml
val is_simple_binder : string -> TERM -> bool;

Description  is_simple_binder binder_nm term returns true iff. argument term is of the form \( \text{\texttt{\textbackslash l}} \text{\texttt{\textbackslash b}} \text{\texttt{\textbackslash r}} \text{\texttt{\textbackslash n}} \text{\texttt{\textbackslash m}} (\lambda \text{\texttt{\textbackslash v}} \text{\texttt{\textbackslash a}} \text{\texttt{\textbackslash r}} \text{\texttt{\textbackslash v}} \text{\texttt{\textbackslash b}} \text{\texttt{\textbackslash o}} \text{\texttt{\textbackslash d}} \text{\texttt{\textbackslash y})} \text{\texttt{\textbackslash n}}} \text{\texttt{\textbackslash m}}, where binder is a constant with the name binder_nm.

See Also  is_binder
```

```sml
val is_simple_\forall : TERM -> bool;

Description  A derived term test for simple \( \forall \)-terms (i.e. of form \( \forall x \cdot t \)), not formed with paired abstractions.

See Also  is_\forall
```

```sml
val is_simple_\exists_1 : TERM -> bool;

Description  Return true only when the term is a \( \exists_1 \)-term (i.e. of form \( \exists_1 x \cdot t \)), formed only by simple abstraction, and false otherwise: no exceptions can be raised.

See Also  is_\exists_1
```

```sml
val is_simple_\exists : TERM -> bool;

Description  A derived term test for \( \exists \)-terms (i.e. of form \( \exists x \cdot t \)), not formed with paired abstractions.

See Also  is_\exists
```

```sml
val is_simple_\lambda : TERM -> bool;

Description  Is the term a simple \( \lambda \)-abstraction (i.e. of form \( \lambda x \cdot t \)).

See Also  is_\lambda
```

```sml
val is_string : TERM -> bool;

Description  Return true only when the term is a string literal (e.g. \( "abc" \)), and false otherwise: no exceptions can be raised.
```

```sml
val is_type_instance : TYPE -> TYPE -> bool;

Description  is_type_instance ty_1 ty_2 returns true iff ty_1 is an instance of ty_2. It cannot raise an exception.
```

```sml
val is_t : TERM -> bool;

Description  Return true only when the term is \( T : BOOL \), and false otherwise: no exceptions can be raised.
```

```sml
val is_vartype : TYPE -> bool;

Description  Return true only when the type is a type variable, and false otherwise: no exceptions can be raised. If the argument isn’t a type variable then it must be a compound type.
```
val is_var : TERM -> bool;

Description Return true only when the term is a variable, and false otherwise: no exceptions can be raised.

val is_∅ : TERM -> bool;

Description Return true only when the term is an empty enumerated set, \( \{\} \), and false otherwise: no exceptions can be raised.

val is_\(\leftrightarrow\) : TERM -> bool;

Description Return true only when the term is a bi-implication (i.e. of form \( a \leftrightarrow b \)), and false otherwise: no exceptions can be raised. N.B. this may be successfully applied to boolean equations.

val is_\(\to\)type : TYPE -> bool;

Description Return true only when the type is a function type, i.e. of form \( \tau_1 \to \tau_2 \), and false otherwise: no exceptions can be raised.

val is_\(\land\) : TERM -> bool;

Description Return true only when the term is a conjunction (i.e. of form \( a \land b \)), and false otherwise: no exceptions can be raised.

val is_\(\lor\) : TERM -> bool;

Description Return true only when the term is a disjunction (i.e. of form \( a \lor b \)), and false otherwise: no exceptions can be raised.

val is_\(\neg\) : TERM -> bool;

Description Return true only when the term is a negation (i.e. of form \( \neg \, x \)), and false otherwise: no exceptions can be raised.

val is_\(\Rightarrow\) : TERM -> bool;

Description Return true only when the term is an implication (i.e. of form \( a \Rightarrow b \)), and false otherwise: no exceptions can be raised.

val is_\(\forall\) : TERM -> bool;

Description Return true only when the term is a \( \forall \)-term (i.e. of form \( \forall \, v \, t \)), possibly formed with paired abstraction, and false otherwise: no exceptions can be raised.

val is_\(\exists\)1 : TERM -> bool;

Description Return true only when the term is a \( \exists_1 \)-term (i.e. of form \( \exists_1 \, v \, t \)), possibly formed with paired abstraction, and false otherwise: no exceptions can be raised.

See Also is_simple_\(\forall\)

See Also is_simple_\(\exists\)1

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
5.1. Syntactic Manipulations

SML

val is_∃ : TERM -> bool;

Description Return true only when the term is a ∃-term (i.e. of form \( \exists vs \cdot t \)) , possibly formed with paired abstraction, and false otherwise: no exceptions can be raised.

See Also is_simple_∃

SML

val is_×_type : TYPE -> bool;

Description Return true only when the type is a pair type, i.e. of the form: \( \gamma : ty_1 \times ty_2 \), and false otherwise: no exceptions can be raised.

SML

val is_ϵ : TERM -> bool;

Description Return true only when the term is a ϵ-term (i.e. of form \( \epsilon vs \cdot t \)), possibly formed with paired abstraction, and false otherwise: no exceptions can be raised.

SML

val is_λ : TERM -> bool;

Description This function returns true iff. the term is of the form \( \lambda vs \cdot t \). It cannot raise exceptions.

See Also is_simple_λ

SML

val is_N : TERM -> bool;

Description Return true only when the term is a numeric literal (e.g. \( 5 \)), and false otherwise: no exceptions can be raised.

SML

val key_mk_const : (E_KEY * TYPE) -> TERM;
val key_dest_const : TERM -> E_KEY * TYPE;

Description Internally, the names of constants are represented using efficient dictionary keys. These functions allow the creation and destruction of constants by key rather than by name.

SML

val key_mk ctype : E_KEY * TYPE list -> TYPE;
val key_dest ctype : TYPE -> E.KEY * TYPE list;

Description Internally, the names of type constructors are represented using efficient dictionary keys. These functions allow the creation and destruction of compound types by key rather than by name.

SML

val list_mk_app : (TERM * TERM list) -> TERM;

Description Applies a function to multiple arguments.

Definition \[ \text{list_mk_app } (\gamma t \gamma, [\gamma t1 \gamma, \gamma t2 \gamma, \gamma t3 \gamma, ...]) = \gamma t t1 t2 t3 ... \]

Failure May give rise to the error message from \text{mk_app}.
val list_mk_binder : (TERM * TERM -> TERM) -> (TERM list * TERM) -> TERM;

Description  If maker (⌜vs⌜, ⌜b⌜) makes an abstraction ⌜bind vs • b⌜, then

list_mk_binder maker ([⌜vs_1⌜, ⌜vs_2⌜, ...],⌜body⌜)

returns ⌜bind vs_1 • bind vs_2 • ... • body⌜. Notice that this can be used for implementing both simple and paired abstractions, with the vs_i being variable structures when so allowed, and otherwise variables.

val list_mk_bin_op : string -> int -> int ->
    (TYPE -> TYPE -> TERM) -> TERM list -> TERM;

Description  This function combines a list of terms using the given operator, as if by mk_bin_op (q.v). Notice the bracketing in the example.

Example

list_mk_bin_op area msg ∧ fun [⌜a⌜, ⌜b ∧ c⌜, ⌜d⌜] = ⌜a ∧ ((b ∧ c) ∧ d)⌜

where ∧ fun takes two (dummy) arguments and returns ⌜$∧$⌜.

Errors

3017 An empty list argument is not allowed

Failure  The failure message for failing to combine its arguments will be as mk_bin_op for the offending two arguments. If given an empty list the error will be from area area, but with message 3017.

val list_mk_let : (((TERM * TERM)list)list * TERM) -> TERM

Description  This generates a nested let-term.

Example

list_mk_let ([][⌜x↓, ⌜1↓]],[⌜y↓, ⌜2↓]])⌜let x = 1 in let y = 2 in x+y⌜

val list_mk_simple_λ : (TERM list * TERM) -> TERM;

Description  λ-abstract a list of variables from a term.

Definition

list_mk_simple_λ ([⌜x1↓, ⌜x2↓,...],⌜t↓] = ⌜λ x1 x2 ... • t↓

This function will be implemented using mk_simple_λ(q.v), not mk_λ.

See Also  list_mk_λ

Failure  May give rise to the error message from mk_simple_λ.
### 5.1. Syntactic Manipulations

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>val list_mk_simple_\forall</code> : <code>TERM list -&gt; TERM</code>;</td>
<td>Universally quantify a term with a list of variables.</td>
<td><code>\( \forall [x_1, x_2, \ldots] \cdot \text{body} \)</code></td>
</tr>
<tr>
<td><code>val list_mk_simple_\exists</code> : <code>TERM list -&gt; TERM</code>;</td>
<td>Existentially quantify a term with a list of variables.</td>
<td><code>\( \exists [x_1, x_2, \ldots] \cdot \text{body} \)</code></td>
</tr>
<tr>
<td><code>val list_mk_type</code> : <code>TYPE list -&gt; TYPE</code>;</td>
<td>Create the type of a multi-argument function.</td>
<td><code>\( \text{type} [\text{ty}_1, \ldots, \text{ty}_n] = \text{ty}_1 \rightarrow \ldots \rightarrow \text{ty}_n \)</code></td>
</tr>
<tr>
<td><code>val list_mk_\land</code> : <code>TERM list -&gt; TERM</code>;</td>
<td>Conjoin a list of terms:</td>
<td><code>\( \land [a, b, c, \ldots] = a \land b \land c \ldots \)</code></td>
</tr>
<tr>
<td><code>val list_mk_\lor</code> : <code>TERM list -&gt; TERM</code>;</td>
<td>A function to make a disjunction of a list of terms.</td>
<td><code>\( \lor [a, b, c, \ldots] = a \lor b \lor c \ldots \)</code></td>
</tr>
</tbody>
</table>

**Errors**
- 3017 An empty list argument is not allowed
- 3031 `?0` is not of type `\:BOOL`
### list.mk ⇒ : TERM list −> TERM;

**Description** Makes a multiple implication term, using $mk ⇒ (q.v.)$.

**Definition**

\[
list.mk ⇒ \left[ \begin{array}{c}
\vdash t_1, \vdash t_2, \ldots, \vdash t_n
\end{array} \right] = \vdash t_1 ⇒ \vdash t_2 ⇒ \ldots ⇒ \vdash t_n
\]

Note that giving a singleton list containing a non-boolean will return that term, rather than fail.

**Errors**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>3015</td>
<td>'?1 is not of type $\vdash BOOL$'</td>
</tr>
<tr>
<td>3017</td>
<td>'An empty list argument is not allowed'</td>
</tr>
<tr>
<td>3031</td>
<td>'?0 is not of type $\vdash BOOL$'</td>
</tr>
</tbody>
</table>

### list.mk ∀ : TERM list * TERM −> TERM;

**Description** Repeatedly universally quantify a term.

**Definition**

\[
list.mk ∀ \left[ \begin{array}{c}
\vdash a, \vdash b, \vdash c, \ldots
\end{array} \right], \\vdash body = \vdash ∀ a b c \ldots \vdash body
\]

This uses $mk ∀$ to generate its result.

**Failure** This may give the errors of $mk ∀$.

### list.mk ∃ : TERM list * TERM −> TERM;

**Description** Repeatedly existentially quantify a term.

**Definition**

\[
list.mk ∃ \left[ \begin{array}{c}
\vdash a, \vdash b, \vdash c, \ldots
\end{array} \right], \\vdash body = \vdash ∃ a b c \ldots \vdash body
\]

This uses $mk ∃$ to generate its result.

**Failure** This may give the errors of $mk ∃$.

### list.mk ϵ : TERM list * TERM −> TERM;

**Description** Repeatedly apply $ϵ$ to a term.

**Definition**

\[
list.mk ϵ \left[ \begin{array}{c}
\vdash a, \vdash b, \vdash c, \ldots
\end{array} \right], \\vdash body = \vdash ϵ a b c \ldots \vdash body
\]

**Failure** This may give the errors of $mk ϵ$.

### list.mk λ : ( TERM list * TERM ) −> TERM;

**Description** Repeatedly $λ$-abstract from a term.

**Definition**

\[
list.mk λ \left[ \begin{array}{c}
\vdash a, \vdash b, \vdash c, \ldots
\end{array} \right], \\vdash body = \vdash λ a b c \ldots \vdash body
\]

This function is implemented using $mk λ$, not $mk.simple λ$.

**See Also** list.mk_simple λ

**Failure** May give rise to the error message from $mk λ$. 

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
5.1. Syntactic Manipulations

```
SML
val list_term_union : (TERM list list) -> TERM list;

Description  Take the union of a number of lists of terms viewed as sets, removing any α-
convertible duplicates.

See Also  list_union for precise ordering of result.
```

```
SML
val list_variant : TERM list -> TERM list -> TERM list;

Description  list_variant stoplist vlist returns a list of variants of the list of variables vlist,
whose names are not present in the stoplist, which is also a list of term variables. No names are
duplicated, the function returning one new variable for each member of vlist. The variants are
generated by sufficient appending of the variant string (see set_variant_string).

Errors
3007  ?0 is not a term variable
```

```
SML
val mk_app : (TERM * TERM) -> TERM;

Description  This produces a function application.

Definition
\[ mk\_app (\gamma f, \gamma t) = \gamma f \ t \]

Note that many derived term constructs, e.g. all quantifications, are also applications. Thus

Example
\[ mk\_app (\forall x, \lambda t) = \forall x \cdot t \]

Errors
3005  Cannot apply ?0 to ?1 as types are incompatible
3006  Type of ?0 not of form \[ : ty1 \rightarrow ty2 \]
```

```
SML
val mk_binder : string -> int -> (TYPE -> TYPE -> TERM) ->
(TERM * TERM) -> TERM;

Description  A generic method of implementing binder constructor functions:

Definition
\[ mk\_binder\ area\ msg\ binder\ nm (\gamma varstruct, \gamma body) = \gamma binder'(\lambda varstruct\bullet body) \]

\[ binder'\ is\ formed\ by\ applying\ binder\ to\ the\ types\ of\ the\ varstruct\ and\ body.\ varstruct\ may\ be\ any\ allowed\ variable\ structure.\]

See Also  mk_simple_binder

Errors
4016  ?0 is not an allowed variable structure

Failure  If the term cannot be made, then the error will be from area, with a message indexed
by msg. If the first term argument is not an allowed variable structure then failure 4016 is raised
from area area.
```
val mk_bin_op : string -> int -> int -> (TYPE -> TYPE -> TERM) ->
            (TERM * TERM) -> TERM;

Description  mk_bin_op area msg1 msg2 rator_fn (t_1, t_2) attempts to form \( \Gamma t_1 \text{ rator} t_2 \). rator' is gained by applying rator_fn to the types of \( t_1 \) and \( t_2 \).

Example

\[
\text{mk_bin_op } "\text{mk} \&" \text{ 3031 3015 (fn } \_ \Rightarrow \text{ fn } _ \_ \Rightarrow \Gamma \text{ \&} (\Gamma a, \Gamma b) = \Gamma a \land b
\]

Failure  The failure message for failing to apply rator to the first term will be from area area, and will have the text indexed by msg1, with the two terms as strings for arguments. If the failure is from applying the rators plus first term to the second term the error message will be from area area, and will have the text indexed by msg2, with the two terms as strings for arguments. It is not unusual for one of these strings of terms to be thrown away by the message msg2 provided by the caller of this function.

val mk_char : string -> TERM;

Description  Construct a character literal.

Example

\[
\text{mk_char } "a" = \Gamma c \rightarrow a
\]

Errors 3023  String ?0 is not a single character

val mk_const : (string * TYPE) -> TERM;

Description  This produces a constant.

Definition

\[
\text{mk_const}("c", \Gamma ty) = \Gamma c : ty
\]

The function makes no checks against the declaration of the constant, the declaration of the type constructors of the type supplied, or the appropriateness of the type supplied: see get_const_info (q.v.). However it will not form constants whose types clash with those constants required by the implementation of the abstract data type THM (q.v.). These are \( = \), \( \Rightarrow \), \( \forall \), and \( \exists \).

Errors

3002  Type of constant with name "=" must be of form \( \Gamma ty1 \rightarrow ty1 \rightarrow BOOL \)
3003  Type of constant with name "\Rightarrow" must be of form \( \Gamma BOOL \rightarrow BOOL \rightarrow BOOL \)
3004  Type of constant with name ?0 must be of form \( \Gamma (ty1 \rightarrow BOOL) \rightarrow BOOL \)

val mk ctype : string * TYPE list -> TYPE;

Description  Create a compound type from a type constructor and sufficient arguments. The function makes no checks against the declaration or arity of the type constructor or the type arguments: see get_type_info (q.v.).

Definition

\[
\text{mk ctype } ("tc", [\Gamma ty1, \Gamma ty2, ...]) = \Gamma (ty1, ty2, ...)tc
\]

\[
\text{mk ctype } ("tc", []) = \Gamma tc
\]
5.1. Syntactic Manipulations

SML
\begin{verbatim}
val mk_empty_list : TYPE -> TERM

Description  A derived term constructor function for generating an empty list term with elements of a given type.

Definition  \[ mk_empty_list \mapsto \gamma : ty \mapsto \gamma[] : ty\ \text{LIST} \mapsto \gamma \]

See Also  \( \text{mk_list} \)
\end{verbatim}

SML
\begin{verbatim}
val mk_enum_set : TERM list -> TERM

Description  A derived term constructor function for generating enumerated sets. The argument is a list of the members of the set. The type of a set of elements of type \( \gamma:TY \mapsto \gamma \mapsto \gamma \) is \( \gamma:TY\ \text{SET} \mapsto \gamma \). If the term list is empty the function will fail (see \( \text{mk}_\emptyset \)). The set must be of terms with the same HOL type.

Definition  \[ mk_enum_set \mapsto \gamma : a \mapsto b \mapsto ... \mapsto \gamma \{ a; b; ... \} \mapsto \gamma \]

Errors
\begin{itemize}
\item \text{3012}  ?0 and ?1 do not have the same types
\item \text{3017}  An empty list argument is not allowed
\end{itemize}
\end{verbatim}

SML
\begin{verbatim}
val mk_eq : (TERM * TERM) -> TERM;

Description  A derived term constructor function for generating equations.

Definition  \[ mk_eq \mapsto \gamma : \gamma = \gamma \mapsto \gamma \]
\[ mk_eq \mapsto \gamma : a:BOOL \mapsto b:BOOL \mapsto \gamma a \Leftrightarrow b \mapsto \gamma \]

Errors
\begin{itemize}
\item \text{3012}  ?0 and ?1 do not have the same types
\end{itemize}
\end{verbatim}

SML
\begin{verbatim}
val mk_float : INTEGER * INTEGER * INTEGER -> TERM;

Description  Make a floating point literal.

Definition  \[ mk_float \mapsto \gamma : x \mapsto 0 \mapsto \gamma \mapsto \gamma XX \mapsto \gamma \]
\[ mk_float \mapsto \gamma : x \mapsto p \mapsto 0 \mapsto \gamma \mapsto \gamma XX.YY \mapsto \gamma \]
\[ mk_float \mapsto \gamma : x \mapsto p \mapsto z \mapsto \gamma \mapsto \gamma XX.YYeZZ \mapsto \gamma \]

where \( XX.YY \) is the decimal representation of \( x \times 10^{-p} \) and \( ZZ \) is the decimal representation of \( z \) (with \( p = z = 0 \) in the first case and \( z = 0 \) in the second).

Errors
\begin{itemize}
\item \text{4041}  the mantissa of a HOL floating point literal must be non–negative
\end{itemize}
\end{verbatim}

SML
\begin{verbatim}
val mk_f : TERM;

Description  The term \( \gamma F : BOOL \mapsto \gamma \).
\end{verbatim}
SML
val mk_if : (TERM * TERM * TERM) -> TERM;

**Description** Make a conditional.

**Definition**
\[ \text{mk}_\text{if} \ (\Gamma \ c, \Gamma \ y, \Gamma \ n) = \Gamma \text{if } c \text{ then } y \text{ else } n \]

**Errors**
3012 \text{?0 and ?1 do not have the same types}
3031 \text{?0 is not of type \text{\texttt{:BOOL}}}

SML
val mk_list : ((TERM * TERM) list * TERM) -> TERM

**Description** A derived term constructor function for generating let-terms. The arguments may have any form allowed by ICL HOL Concrete Syntax. Thus they may be variable structures formed by pairing, or single clause, non-recursive functions, whose arguments may only be variable structures formed by pairing.

**Example**
\[
\text{mk}_\text{let} \ (\Gamma \ (\text{x}), \Gamma \ (\text{1}, \text{2})), \Gamma \text{ x + y}) = \\
\text{let } (\text{x}, \text{y}) = (\text{1}, \text{2}) \text{ in } \text{x + y}
\]

**Errors**
3012 \text{?0 and ?1 do not have the same types}
4007 \text{?0 is not a well-formed LHS for \text{mk}_\text{let}}

SML
val mk_list : TERM list -> TERM

**Description** A derived term constructor function for generating list-terms. The argument is a list of the members of the list. If the term list is empty the function will fail (see \text{mk}_{\text{empty list}}). The list must be of terms with the same HOL type.

**Definition**
\[ \text{mk}_\text{list} \ (\Gamma \ \text{a}, \Gamma \ \text{b}, \ldots) = \Gamma [\text{a}; \text{b}; \ldots] \]

**Errors**
3012 \text{?0 and ?1 do not have the same types}
3017 \text{An empty list argument is not allowed}
5.1. Syntactic Manipulations

```ml
val mk_mon_op : string -> int -> (TYPE -> TERM) -> TERM -> TERM;
```

**Description**  
`mk_mon_op area msg rator_fn "rand"` attempts to form the term `"rator rand"`.  
`"rator"` is gained by applying `rator_fn` to the type of `"rand"`.

**Example**  
`mk_mon_op "mk¬" 3031 (fn _ => "$¬") t:BOOL = ¬ t`

**Failure**  
The failure message for failing to apply `rator` to its arguments will be from area `area`,  
and will have the text indexed by `msg`.

```ml
val mk_multi¬ : (int * TERM) -> TERM;
```

**Description**  
`mk_multi¬ (n, t)` will apply the constructor `mk¬` n times to `t`.

**Example**  
`mk_multi¬ (2, "T") = ¬(¬ T)`

**Errors**  
3031 0 is not of type `¬`BOOL
4030 0 is negative

```ml
val mk_pair : (TERM * TERM) -> TERM;
```

**Description**  
A derived term constructor function for generating pairs.

**Definition**  
`mk_pair("t1", "t2") = (t1, t2)`

```ml
val mk_set_comp : (TERM * TERM) -> TERM
```

**Description**  
A derived term constructor function for generating set comprehensions.

**Example**  
`mk_set_comp ("x", "x > 5") = \{ x | x > 5\}`

**Errors**  
3015 1 is not of type `\{x : BOOL\}`
4016 0 is not an allowed variable structure

```ml
val mk_simple_binder : string -> int -> (TYPE -> TYPE -> TERM) ->  
                       (TERM * TERM) -> TERM;
```

**Description**  
`mk_simple_binder area msg binder_fn (var, body)` generates the term:  
`\binder(\var = body)`

where `binder` is `binder_fn` applied to the types of `var` and `body`.  
`var` must be a term variable.

**See Also**  
`mk_binder`

**Errors**  
3007 0 is not a term variable

**Failure**  
If the term cannot be made, then the error will be from area `area`, with a message indexed  
by `msg`, and the two terms as string arguments.  
If the first of the pair of terms is not a variable  
then error 3007 will be given from area `area`.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
val mk_simple_term : DEST_SIMPLE_TERM -> TERM;

Description  Create a well-formed TERM from a statement of a top-level structure, and the associated constituent parts.

It makes the same checks as mk_const, mk_app, etc(q.v.), and gives the same error messages as these if there is a failure.

See Also  DEST_SIMPLE_TERM

Errors
3005  Cannot apply ?0 to ?1 as types are incompatible
3006  Type of ?0 not of form ":ty1 -> ty2"
3007  ?0 is not a term variable

val mk_simple_type : DEST_SIMPLE_TYPE -> TYPE;

Description  This function constructs a HOL type from something of type SIMPLE_DEST_TYPE (q.v).

val mk_simple_\forall : (TERM * TERM) -> TERM;

Description  A derived term constructor function for generating simple \forall-terms.

Definition
\[
\text{mk_simple}_\forall (\text{"var"}, \text{"body"}) = \forall \text{ var} \cdot \text{body}
\]

\text{var} must be a term variable.

See Also  \text{mk}_\forall

Errors
3007  ?0 is not a term variable
3015  ?1 is not of type ":BOOL"

val mk_simple_\exists_1 : (TERM * TERM) -> TERM;

Description  A derived term constructor function for generating simply abstracted \exists_1-terms.

Definition
\[
\text{mk_simple}_\exists_1 (\text{"var"}, \text{"body"}) = \exists_1 \text{ var} \cdot \text{body}
\]

\text{var} must be a variable.

Errors
3007  ?0 is not a term variable
3015  ?1 is not of type ":BOOL"

See Also  \text{mk}_\exists_1
5.1. Syntactic Manipulations

SML

\texttt{val mk\_simple\_∃ : (TERM} ∗ TERM\texttt{) −} \to \texttt{TERM};

\textbf{Description} A derived term constructor function for generating simple ∃-terms.

\textbf{Definition} \(\text{mk\_simple\_∃ (⌜var⌝, ⌜body⌝)} = \exists \; \text{var} \cdot \text{body\)}

\textit{var} must be a term variable.

\textbf{See Also} \(\text{mk\_∃}\)

\textbf{Errors}

\begin{itemize}
\item \texttt{3007} \(\exists 0\) is not a term variable
\item \texttt{3015} \(\exists 1\) is not of type \(\exists:\text{BOOL}\)
\end{itemize}

SML

\texttt{val mk\_simple\_λ : (TERM} ∗ TERM\texttt{) −} \to \texttt{TERM};

\textbf{Description} This produces a simple λ-abstraction. It may only abstract variables.

\textbf{Definition} \(\text{mk\_simple\_λ (⌜v⌝, ⌜t⌝)} = \lambda \; \text{v} \cdot \text{t\)}

\textbf{See Also} \(\text{mk\_λ}\)

\textbf{Errors}

\begin{itemize}
\item \texttt{3007} \(\exists 0\) is not a term variable
\end{itemize}

SML

\texttt{val mk\_string : string −} \to \texttt{TERM;}

\textbf{Description} Construct a string literal.

\textbf{Example} \(\text{mk\_string "abc" =⌜"abc\"\)}

SML

\texttt{val mk\_term : DEST\_TERM −} \to \texttt{TERM}

\textbf{Description} Create a term from a derived term. It is an inverse to \texttt{dest\_term (q.v)}, and therefore understands how to handle paired abstractions.

The function is implemented using the individual primitive and derived term constructors (e.g. \texttt{mk\_const} and \texttt{mk\_∀}), with what checks they use.

\textbf{Failure} This function will fail with the same messages as the appropriate term constructor functions.

SML

\texttt{val mk\_t : TERM;}

\textbf{Description} The term \(⌜T : BOOL\\)}.

SML

\texttt{val mk\_vartype : string −} \to \texttt{TYPE;}

\textbf{Description} Create a HOL type variable from a string:

\textbf{Definition} \(\text{mk\_vartype "tv" =⌜:tv\\)}

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
val mk_var : (string * TYPE) -> TERM;

Description  This produces a term variable.
Definition  \[ \text{mk}_\text{var}("v", \vdash ty) = \Gamma v : ty \]  

val mk_∅ : TYPE -> TERM;

Description  A derived term constructor function for generating an empty (enumerated) set with elements of a given type.
Definition  \[ \text{mk}_\emptyset \vdash \Gamma : ty \]  

See Also  mk_enum_set

val mk⇔ : (TERM * TERM) -> TERM;

Description  A derived term constructor function for generating bi-implications.
Definition  \[ \text{mk}_\iff (\vdash t_1, \vdash t_2) = \vdash t_1 \iff t_2 \]  

Errors  3015  ?1 is not of type \vdash \text{BOOL}
3031  ?0 is not of type \vdash \text{BOOL}

val mk→_type : (TYPE * TYPE) -> TYPE;

Description  Create a function type from two types. A function type is just a kind of compound type.
Definition  \[ \text{mk}_\rightarrow\text{type} (\vdash ty_1, \vdash ty_2) = \text{mk}_\text{ctype}(\rightarrow, [\vdash ty_1, \vdash ty_2]) = \vdash ty_1 \to ty_2 \]  

val mk∧ : (TERM * TERM) -> TERM;

Description  A derived term constructor function for generating conjunctions.
Definition  \[ \text{mk}_\land (\vdash t_1, \vdash t_2) = \vdash t_1 \land t_2 \]  

Errors  3015  ?1 is not of type \vdash \text{BOOL}
3031  ?0 is not of type \vdash \text{BOOL}

val mk∨ : (TERM * TERM) -> TERM;

Description  A derived term constructor function for generating disjunctions.
Definition  \[ \text{mk}_\lor (\vdash t_1, \vdash t_2) = \vdash t_1 \lor t_2 \]  

Errors  3015  ?1 is not of type \vdash \text{BOOL}
3031  ?0 is not of type \vdash \text{BOOL}
5.1. Syntactic Manipulations

SML

\texttt{val \texttt{mk}_\neg : \texttt{TERM} \rightarrow \texttt{TERM};}

\textbf{Description} A derived term constructor function for generating negations.

\textbf{Definition}

\[ \texttt{mk}_\neg \Gamma \neg t = \Gamma t \neg \]

\textbf{Errors}

\begin{align*}
3031 & \quad \text{"0 is not of type \neg \text{BOOL}\"} \\
3031 & \quad \text{"0 is not of type \neg \text{BOOL}\"}
\end{align*}

SML

\texttt{val \texttt{mk}_\Rightarrow : (\texttt{TERM} \times \texttt{TERM}) \rightarrow \texttt{TERM};}

\textbf{Description} A derived term constructor function for generating implications. It takes two arguments: the antecedent and the consequent.

\textbf{Definition}

\[ \texttt{mk}_\Rightarrow (\Gamma a \Gamma, \Gamma b \Gamma) = \Gamma a \Rightarrow b \]

\textbf{Errors}

\begin{align*}
3015 & \quad \text{"1 is not of type \\Rightarrow \text{BOOL}\"} \\
3031 & \quad \text{"0 is not of type \\Rightarrow \text{BOOL}\"}
\end{align*}

SML

\texttt{val \texttt{mk}_\forall : (\texttt{TERM} \times \texttt{TERM}) \rightarrow \texttt{TERM};}

\textbf{Description} A derived term constructor function for generating \forall-terms.

\textbf{Definition}

\[ \texttt{mk}_\forall (\Gamma \text{varstruct} \Gamma, \Gamma \text{body} \Gamma) = \Gamma \forall \text{varstruct} \bullet \text{body} \Gamma \]

varstruct may be any allowed variable structure.

\textbf{Errors}

\begin{align*}
3015 & \quad \text{"1 is not of type \forall \text{BOOL}\"} \\
4016 & \quad \text{"0 is not an allowed variable structure}\]
\end{align*}

\textbf{See Also} \texttt{mk\_simple\_}\forall

SML

\texttt{val \texttt{mk}_\exists_1 : (\texttt{TERM} \times \texttt{TERM}) \rightarrow \texttt{TERM};}

\textbf{Description} A derived term constructor function for generating \exists_1-terms.

\textbf{Definition}

\[ \texttt{mk}_\exists_1 (\Gamma \text{varstruct} \Gamma, \Gamma \text{body} \Gamma) = \Gamma \exists_1 \text{varstruct} \bullet \text{body} \]

varstruct may be any allowed variable structure.

\textbf{Errors}

\begin{align*}
3015 & \quad \text{"1 is not of type \exists_1 \text{BOOL}\"} \\
4016 & \quad \text{"0 is not an allowed variable structure}\]
\end{align*}

\textbf{See Also} \texttt{mk\_\exists_1}
val mk_∃ : (TERM * TERM) -> TERM;

Description  A derived term constructor function for generating ∃-terms.

Definition  \[ \text{mk}_\exists (\text{"varstruct"}, \text{"body"}) = \text{"∃ varstruct\• body"} \]

varstruct may be any allowed variable structure.

Errors  
\[ 3015 \ ?1 \text{ is not of type } \text{⌜BOOL⌝} \]
\[ 4016 \ ?0 \text{ is not an allowed variable structure} \]

See Also  mk_simple_∃

val mk_×_type : (TYPE * TYPE) -> TYPE

Description  mk_×_type (⌜:ty_1⌝,⌜:ty_2⌝) returns a pair type: ⌜:ty_1 × ty_2⌝.

val mk_ϵ : (TERM * TERM) -> TERM;

Description  A derived term constructor function for generating ϵ-terms.

Definition  \[ \text{mk}_\epsilon (\text{"varstruct"}, \text{"body"}) = \text{"ϵ varstruct\• body"} \]

varstruct may be any allowed variable structure.

Errors  
\[ 3015 \ ?1 \text{ is not of type } \text{⌜BOOL⌝} \]
\[ 4016 \ ?0 \text{ is not an allowed variable structure} \]

val mk_λ : TERM * TERM -> TERM

Description  This creates a λ-abstraction of an allowed variable structure from a term.

Example  
\[ \text{mk}_\lambda (\text{⌜x⌝}, \text{⌜x + y⌝}) = \text{⌜λ x\• x + y⌝} \]
\[ \text{mk}_\lambda (\text{⌜(x, y)⌝}, \text{⌜x + y⌝}) = \text{⌜λ (x, y)\• x + y⌝} \]
\[ \text{mk}_\lambda (\text{⌜((x1,x2), (y1,y2))⌝}, \text{⌜x2 + y2⌝}) = \text{⌜λ ((x1,x2), (y1,y2))\• x2 + y2⌝} \]

See Also  mk_simple_λ

Errors  
\[ 4016 \ ?0 \text{ is not an allowed variable structure} \]

val mk_ℕ : INTEGER -> TERM;

Description  Construct a numeric literal: the argument may not be negative.

Example  
\[ \text{mk}_\mathbb{N} 5 = \text{⌜5⌝} \]

Errors  
\[ 3021 \ ?0 \text{ should be 0 or positive} \]

val quantifier : string -> TYPE -> TYPE -> TERM;

Description  quantifier name type dummy returns a constant, with the given name, and type ⌜:(type→BOOL)→BOOL⌝, This is an appropriate type for binders. The dummy is present only to make the function have an acceptable signature for certain other functions.
5.1. Syntactic Manipulations

SML

\[ \text{val rename : } (\text{string } \ast \text{TYPE}) \rightarrow \text{string} \rightarrow \text{TERM} \rightarrow \text{TERM}; \]

Description  rename (oname, type) cname term returns a term based on term, but with any free variables with name oname, and type type renamed to cname.

SML

\[ \text{val set_variant_suffix : string } \rightarrow \text{string}; \]

Description  Sets the string control variant_suffix used to create variant names in string_variant (q.v.) and its relatives. The string is initially a single prime character. The function returns the previous setting of the control.

Errors  \[3028\] string may not be empty

SML

\[ \text{val string_of_term : TERM } \rightarrow \text{string}; \]

Description  This returns a display of a term in the form of a string, with no inserted new lines, suitable for use with \text{diag_string} and \text{fail}.

See Also  \text{format_term} is a formatted string display of a term.

SML

\[ \text{val string_of_type : TYPE } \rightarrow \text{string}; \]

Description  This returns a display of a type in the form of a string, with no inserted new lines, suitable for use with \text{diag_string} and \text{fail}.

See Also  \text{format_type} is a formatted string display of a type.

SML

\[ \text{val string_variant : string list } \rightarrow \text{string } \rightarrow \text{string}; \]

Description  string_variant vlist name returns a string that is different from any name in vlist. Variants are formed by repeatedly appending the variant string(see set_variant_string) to the name. Note that string_variant [] name gives name.

Uses  Somewhat faster than \text{variant} if term variables are already destroyed, and their names and types are directly accessible.

See Also  \text{variant}

SML

\[ \text{val STRING : TYPE}; \]

Description  This is the HOL type of strings, a type abbreviation for lists of objects of type \text{CHAR}.

Definition  \[ \text{val STRING} = \langle \text{CHAR LIST} \rangle; \]

See Also  Theory “char”.

SML

\[ \text{val strip_app : TERM } \rightarrow \text{TERM } \ast \text{TERM list}; \]

Description  Splits a term into a head term, that is not an application, and the list of argument terms, if any, to which that head term was applied.

Example  \[
\begin{align*}
\text{strip_app } \langle \text{t t1 t2 t3 ...} \rangle &= \langle \text{t}, [\langle \text{t1}, \langle \text{t2}, \langle \text{t3}, [... \rangle \rangle \rangle \rangle \rangle \\
\text{strip_app } \langle \text{T} \rangle &= \langle \text{T}, [] \rangle
\end{align*}
\]
val strip_binder : string -> TERM -> TERM list * TERM;

Description  
strip_binder binder applied to

\[ \langle \lambda v_1 \cdot \langle \lambda v_2 \cdot \ldots \cdot \text{body} \ldots \rangle \rangle^\gamma \]

will return

\[ \langle v_1^\gamma, v_2^\gamma, \ldots \rangle, \text{body}^\gamma \]

where the \( v_i \) are allowed variable structures. The function acts as dest_binder (q.v), and will handle paired abstraction terms. It will return an empty list and the original term if the supplied term is not formed using the binder.

See Also  
strip_simple_binder

val strip_bin_op : string -> TERM -> TERM list

Description  
This function strips a binary operator, attempting to destroy its term argument, and recursively stripping to the right, as if by dest_bin_op. A term not formed from the operator is returned unchanged, as a singleton list.

Example

\[ \text{strip_bin_op } "\land" \langle a \land (b \land c) \land d \rangle = \langle a^\gamma, b^\gamma, c^\gamma, d^\gamma \rangle \]

val strip_leaves : (\'<a\'> \rightarrow \'<a\'>) \rightarrow \'<a\'> \rightarrow \'<a\'> list;

Description  
Given a function that destroys an object into a pair of objects (and here we are thinking of, for example, dest_\&), recursively descend the results of destruction down both branches, destroying until failure.

Example

\[ \text{strip_leaves dest_\& } \langle a \& b \rangle \& c \& d \rangle = \langle a^\gamma, b^\gamma, c^\gamma, d^\gamma \rangle \]

val strip_let : TERM -> ((TERM * TERM) list) list * TERM

Description  
This destroys a sequence of nested let constructs.

Example

\[ \text{strip_let } \langle \text{let } x = 1 \ \text{in let } y = 2 \ \text{in } x+y \rangle = \langle \langle \langle x^\gamma, 1^\gamma \rangle, \langle y^\gamma, 2^\gamma \rangle \rangle, x^\gamma+y^\gamma \rangle \]

val strip_simple_binder : string -> TERM -> TERM list * TERM;

Description  
strip_simple_binder binder applied to

\[ \langle \lambda v_1 \cdot \langle \lambda v_2 \cdot \ldots \cdot \text{body} \ldots \rangle \rangle^\gamma \]

will return

\[ \langle v_1^\gamma, v_2^\gamma, \ldots \rangle, \text{body}^\gamma \]

where the \( v_i \) are simple variables. The function acts as dest_simple_binder (q.v), and will not handle paired abstraction terms. It will return an empty list and the original term if the supplied term is not formed using the binder.

See Also  
strip_binder
### 5.1. Syntactic Manipulations

#### SML

```sml
val strip_simple_∀ : TERM -> (TERM list * TERM);
```

**Description**  Strip a multiply universally simply quantified term.

**Definition**

\[
\text{strip\_simple\_∀ } \forall a \ b \ c \ldots \cdot \text{body} = \left[ \text{\textasciitilde a\textasciitilde, \textasciitilde b\textasciitilde, \textasciitilde c\textasciitilde, \ldots} \right], \text{\textasciitilde body}\]

#### SML

```sml
val strip_simple_∃ : TERM -> (TERM list * TERM);
```

**Description**  Strip a repeatedly existentially simply quantified term.

**Definition**

\[
\text{strip\_simple\_∃ } \exists a \ b \ c \ldots \cdot \text{body} = \left[ \text{\textasciitilde a\textasciitilde, \textasciitilde b\textasciitilde, \textasciitilde c\textasciitilde, \ldots} \right], \text{\textasciitilde body}\]

#### SML

```sml
val strip_spine_left : (\'a -> \'a * \'a) -> \'a -> \'a list;
```

**Description**  Given a function that destroys an object into a pair of objects (and here we are thinking of, for example, \(\text{dest\_∧}\)), recursively descend the left results of destruction, destroying until failure.

**Example**

\[
\text{strip\_spine\_left dest\_∧ \text{\textasciitilde (a \& b) \& c \& d\textasciitilde} = \left[ \text{\textasciitilde a\textasciitilde, \textasciitilde b\textasciitilde, \textasciitilde c\textasciitilde, \textasciitilde d\textasciitilde} \right]}
\]

#### SML

```sml
val strip_spine_right : (\'a -> \'a * \'a) -> \'a -> \'a list;
```

**Description**  Given a function that destroys an object into a pair of objects (and here we are thinking of, for example, \(\text{dest\_∧}\)), recursively descend the right results of destruction, destroying until failure.

**See Also**  \(\text{strip\_bin\_op}\) for stripping terms formed by binary (constant\_op) term operators.

**Example**

\[
\text{strip\_spine\_left dest\_∧ \text{\textasciitilde (a \& b) \& c \& d\textasciitilde} = \left[ \text{\textasciitilde a\textasciitilde, \textasciitilde b\textasciitilde, \textasciitilde c\textasciitilde, \textasciitilde d\textasciitilde} \right]}
\]

#### SML

```sml
val strip_→_type : TYPE -> TYPE list;
```

**Description**  Strip the type of a multi-argument function into its constituent types, only descending into the right hand result of \(\text{dest\_→\_type}\).

**Definition**

\[
\text{strip\_→\_type } \text{\textasciitilde ty1 \& \ldots \& tyn} = \left[ \text{\textasciitilde ty1\textasciitilde, \textasciitilde tyn\textasciitilde} \right]
\]

#### SML

```sml
val strip_∧ : TERM -> TERM list
```

**Description**  Break a term into its constituent conjuncts, descending recursively only to the right.

**Example**

\[
\text{strip\_∧ \text{\textasciitilde a \& (b \& c) \& d\textasciitilde} = \left[ \text{\textasciitilde a\textasciitilde, \textasciitilde b\textasciitilde, \textasciitilde c\textasciitilde, \textasciitilde d\textasciitilde} \right]}
\]
<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>strip_∨ : TERM → TERM list</code></td>
<td></td>
<td>Break a term into its constituent disjuncts, descending recursively only to the right.</td>
</tr>
<tr>
<td>Example</td>
<td></td>
<td><code>strip_∨ a ∨ (b ∨ c) ∨ d = [⌜a⌜,⌜b⌜,⌜c⌜,⌜d⌜]</code></td>
</tr>
<tr>
<td><code>strip_⇒ : TERM → TERM list;</code></td>
<td></td>
<td>Strip a multiple implication into a list of antecedents appended to the singleton list of the innermost consequent.</td>
</tr>
<tr>
<td>Definition</td>
<td></td>
<td><code>strip_⇒ t1 ⇒ t2 ⇒ ... ⇒ tn = [⌜t1⌜,⌜t2⌜,...,⌜tn⌜]</code></td>
</tr>
<tr>
<td>Note</td>
<td></td>
<td>Note that stripping a non-boolean will result in a singleton list containing that term, not a fail.</td>
</tr>
<tr>
<td><code>strip_∀ : TERM → (TERM list × TERM);</code></td>
<td></td>
<td>Strip a multiply universally quantified term (perhaps with paired abstractions).</td>
</tr>
<tr>
<td>Definition</td>
<td></td>
<td><code>strip_∀ ∀ a b c ... • body = [⌜a⌜,⌜b⌜,⌜c⌜,...],⌜body⌜</code></td>
</tr>
<tr>
<td><code>strip_∃ : TERM → (TERM list × TERM);</code></td>
<td></td>
<td>Strip a repeatedly existentially quantified term, possibly formed with paired abstractions.</td>
</tr>
<tr>
<td>Definition</td>
<td></td>
<td><code>strip_∃ ∃ a b c ... • body = [⌜a⌜,⌜b⌜,⌜c⌜,...],⌜body⌜</code></td>
</tr>
<tr>
<td><code>strip_ε : TERM → (TERM list × TERM);</code></td>
<td></td>
<td>Strip multiple ε’s.</td>
</tr>
<tr>
<td>Definition</td>
<td></td>
<td><code>strip_ε ε a b c ... • body = [⌜a⌜,⌜b⌜,⌜c⌜,...],⌜body⌜</code></td>
</tr>
<tr>
<td><code>strip_λ : TERM → (TERM list × TERM);</code></td>
<td></td>
<td>Strip a multiple λ-abstraction.</td>
</tr>
<tr>
<td>Definition</td>
<td></td>
<td><code>strip_λ λ a b c ... • body = [⌜a⌜,⌜b⌜,⌜c⌜,...],⌜body⌜</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>This uses <code>dest_λ</code> (q.v.) rather than <code>dest_simple_λ</code>.</td>
</tr>
</tbody>
</table>
5.1. Syntactic Manipulations

SML

```sml
val subst : (TERM * TERM) list -> TERM -> TERM;
```

**Description** subst \([\langle t_1, u_1 \rangle, \langle t_2, u_2 \rangle, \ldots \rangle \) returns the term formed from \(t\) by parallel substitution of the \(t_i\) for the \(u_i\). The \(u_i\) can be variables or arbitrary terms but only “free” occurrences of a \(u_i\) will be changed (i.e., only occurrences in which no free variable of \(u_i\) becomes a bound variable in \(t\)). Bound variables in \(t\) are renamed as necessary to prevent bound variable capture.

If some \(u_i\) appears more than once in the substitution list, say \(u_i = u_j\) for \(i < j\), then the later pair \(\langle t_j, u_j \rangle\) is ignored.

**Definition**

\[
\text{subst} \left[ \left[ \langle t_1, u_1 \rangle, \langle t_2, u_2 \rangle, \ldots \rangle \right] \right] t = \left[ t[\langle t_1/u_1 \rangle, \langle t_2/u_2 \rangle, \ldots \rangle] \right]
\]

**See Also** var_subst

**Errors** 3012 ?0 and ?1 do not have the same types

---

SML

```sml
val term_any : (TERM -> bool) -> TERM -> bool;
```

**Description** Given a predicate on terms, tests to see if any sub-term of some term (or the term itself) satisfies the predicate. The search ceases on the first satisfaction, rather than all the tests being done and the results combined.

---

SML

```sml
val term_consts : TERM -> (string * TYPE) list;
```

**Description** This function extracts the subterms of a term which are constants, giving destroyed constants in each case (duplicates are eliminated)

---

SML

```sml
val term_diff : (TERM list * TERM list) -> TERM list;
```

**Description** Remove any terms in the first list that are \(\alpha\)-convertible to any in the second. An infix function.

---

SML

```sml
val term_fail : string -> int -> TERM list -> 'a;
```

**Description** term_fail area msg tml first creates a list of functions from unit to string, using string_of_term (q.v.) providing displays of the list of terms. It then calls fail with the area, msg and this list of functions. This allows terms to be presented in error messages.

---

SML

```sml
val term_fold : ((TERM list) -> (TERM * 'a) -> 'a) -> (TERM * 'a) -> 'a;
```

**Description** term_fold tmfun \((tm, e)\) traverses \(tm\) (depth first) and folds \(tmfun\) on the subterms for which it does not fail. term_fold does not traverse a subterm on which \(tmfun\) did not fail. \(tmfun\) has as its first argument a list giving the bound variables which are in scope at the point of use. It does not attempt to apply \(tmfun\) to a bound variable of an abstraction.

---

SML

```sml
val term_grab : (TERM list * TERM) -> TERM list;
```

**Description** If the given term is not \(\alpha\)-convertible to any member of the list, then add it to the list. An infix function.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
SML

val term_less : (TERM list * TERM) -> TERM list;

Description Remove any terms in the list that are α-convertible to the given term. An infix function.

SML

val term_map : ((TERM list) -> TERM -> TERM) -> TERM -> TERM;

Description term_map tmfun tm traverses tm (breadth first) looking for subterms for which the application tmfun tm does not fail and replaces such subterms with tmfun tm. It does not traverse the resulting subterms. tmfun has as its first argument a list giving the bound variables which are in scope at the point of use. It does not attempt to apply tmfun to a bound variable of an abstraction.

SML

val term_match : TERM -> TERM -> (TYPE * TYPE) list * (TERM * TERM) list;

Description term_match tm_1 tm_2 attempts to find if tm_1 is an instance of tm_2, up to α-convertibility. If so, then it returns two lists. The first gives the correspondence between types in tm_1 with type variables in tm_2. The second gives the correspondence between (type instantiated) terms in tm_1 with free variables in tm_2. Trivial (i.e. (x, x)) correspondences are not noted.

Errors 3054 ?0 is not a term instance of ?1

SML

val term_mem : (TERM * TERM list) -> bool;

Description Is the given term α-convertible to any term in the list? An infix function.

SML

val term_tycons : TERM -> (string * int) list;

Description Returns the set of type constructors and their arity present in types present within a term (represented as a list).

SML

val term_types : TERM -> TYPE list;

Description Gives a list of all the types of constants, variables or λ-abstraction variables within the term argument.

SML

val term_tyvars : TERM -> string list;

Description Returns the list of type variable names present in types present within a term.

SML

val term_union : (TERM list * TERM list) -> TERM list;

Description Take the union of two term lists viewed as sets, removing any α-convertible duplicates. An infix function.

See Also union for precise ordering of result.

SML

val term_vars : TERM -> (string * TYPE) list;

Description This function extracts the subterms of a term which are variables (including abstraction variables), giving destroyed variables in each case.
5.1. Syntactic Manipulations

val type_any : (TYPE -> bool) -> TYPE -> bool;

Description  Given a predicate on types, tests to see if any sub-type of some type (or the type itself) satisfies the predicate. The search ceases on the first satisfaction, rather than all the tests being done and the results combined.

val type_fail : string -> int -> TYPE list -> 'a;

Description  type_fail area msg tyl first creates a list of functions from unit to string, using string_of_type (q.v.) providing displays of the list of types. It then calls fail with the area, msg and this list of functions. This allows types to be presented in error messages.

val type_map : (TYPE -> TYPE) -> TYPE -> TYPE;

Description  type_map tyfun ty traverses ty (breadth first) looking for subtypes, st, for which the application tyfun st does not fail and replaces such subtypes with tyfun st. It does not traverse the resulting subtypes.

val type_match1 : (TYPE * TYPE) list -> TYPE -> TYPE -> (TYPE * TYPE) list;

Description  type_match1 is similar to type_match, q.v., but has an additional context parameter representing an instantiation; type_match1 will fail unless the supplied context can be extended to give the required match. For example, the first line below evaluates true, but the second fails.

\[
\text{type_match1}[(\ell::'b \to \mathbb{N}), (\ell::'a \to \mathbb{N})] \rightarrow 'b \ominus 'a \rightarrow 'b \ominus \ell::'a \rightarrow 'b \ominus (\ell::'b \to \mathbb{N}), (\ell::'a \to \mathbb{N})];
\]

Trivial associations are included in the result so that they can be passed as the context in subsequent calls. The second element of each pair in the context must be a type variable.

See Also  type_match

Errors

3055  ?0 is not a type instance of ?1 in the supplied context

3019  ?0 is not a type variable

val type_match : TYPE -> TYPE -> (TYPE * TYPE) list;

Description  type_match ty_1 ty_2 attempts to match ty_1 with ty_2, i.e., to determine if ty_1 can be obtained from ty_2 by instantiating type variables. If so, it returns a representation of the type instantiation as an association list suitable for use as an argument to inst_type q.v. Trivial (i.e. (x, x)) associations are not included. For example:

\[
\text{type_match} (\ell::'a \to \mathbb{N}) \rightarrow 'b \ominus 'a \rightarrow 'b \ominus \ell::'a \rightarrow 'b \ominus (\ell::'b \to \mathbb{N}), (\ell::'a \to \mathbb{N})];
\]

See Also  type_match1, inst_type

Errors

3053  ?0 is not a type instance of ?1

val type_of : TERM -> TYPE;

Description  This gives the HOL type of a term.
val type_tycons : TYPE -> (string * int) list;

Description  This returns a list of names of type constructors, and the arity of their use, within a type.

val type_tyvars : TYPE -> string list;

Description  Returns the list of type variable names present in a type.

val variant : TERM list -> TERM -> TERM;

Description  variant stoplist v returns a variant of variable v whose name is not used for any variable in stoplist (which must be only variables). The variants are generated by sufficient appending of the variant string (see set_variant_string).

Errors  3007  ?0 is not a term variable

See Also  string_variant, list_variant

val var_subst : (TERM * TERM) list -> TERM -> TERM;

Description  var_subst alist term returns the term formed by, for each pair in alist, substituting in term all free instances of the term variable which is the second of the pair with the first of the pair. The pair of the first matching term variable in the list will be used, duplicates later in the list will be ignored. Renaming may occur to prevent bound variable capture.

Note that the term variables must have the same types as the terms that are to replace them.

Definition  var_subst [(r t1 /", r x1 "), (r t2 /", r x2 "), ..., ] r t = r t[x1/t1, t2/x2, ...]

Errors  3007  ?0 is not a term variable
      3012  ?0 and ?1 do not have the same types

See Also  subst

val ~=$: (TERM * TERM) -> bool;

Description  An infix equality test that returns true only when its two term arguments are \(\alpha\)-convertible, and false otherwise: no exceptions can be raised. Equality of terms is gained by using \(=\$\)

val N : TYPE;

Description  This is the HOL type of the natural numbers, \(0, 1, \ldots\).

Definition  val N = "\N";

See Also  Theory "N".
5.2 Discrimination Nets

SML

\[
\text{signature NetTools = sig}
\]

**Description** This provides the discrimination net tools that will be used to maintain and use databases of values indexed by term form.

SML

\[
\text{type } \tau \text{ NET};
\]

**Description** This is the type of a discrimination net, its type parameter being the type of values that are handled by the net.

SML

\[
\text{val empty_net : } \tau \text{ NET};
\]

**Description** This is the starting discrimination net, which returns an empty list of values, regardless of term form.

SML

\[
\text{val list_net_enter : } (\text{TERM } \ast \tau) \text{ list } \rightarrow (\tau \text{ NET}) \rightarrow (\tau \text{ NET});
\]

**Description** This enters a list of values and indexing terms into a discrimination net, returning the resulting net.

SML

\[
\text{val make_net : } (\text{TERM } \ast \tau) \text{ list } \rightarrow (\tau \text{ NET});
\]

**Description** This enters a list of values and indexing terms into an empty discrimination net, returning the resulting net.

SML

\[
\text{val net_enter : } (\text{TERM } \ast \tau) \rightarrow (\tau \text{ NET}) \rightarrow (\tau \text{ NET});
\]

**Description** This enters a value and its indexing term into a discrimination net, returning the resulting net.

SML

\[
\text{val net_lookup : } (\tau \text{ NET}) \rightarrow \text{TERM } \rightarrow (\tau \text{ list});
\]

**Description** \text{net_lookup net term} will return a list of at least all the values entered into net that were indexed by terms which can be matched (by \text{term\_match}, q.v.) to term. I.e. term can be produced by type and term variable instantiation from the indexing term.

A principal purpose of \text{net\_lookup} is to make the process of rewriting a term using a list of equations and conversions more efficient by quickly filtering out items which are not applicable. Consequently speed is more important than accuracy: to use the wrong metaphor, it is not important if some inapplicable equations “slip through the net” provided all the applicable ones do as well.

The discrimination net actually returns all values whose indexing terms have the same structure as the term matched, ignoring types and variables. Thus only the pattern of constant names, combinations and abstractions will be considered, with variables in the indexing term being presumed to match any term form, regardless of type.

If \text{net\_lookup} returns more than one value, then the only ordering on the resulting values specified is that if two entries are made into the net with the same index term, then if the \text{net\_lookup} term matches the index term then the second entered value will be returned before the first in the list of matches.
THE MANAGEMENT OF THEORIES AND THEOREMS

6.1 Standard ML Type Definitions

SML
datatype THEORY_STATUS =
    TSNormal | TSLocked | TSAncestor | TSDeleted;

Description Objects of this datatype indicate the status of a theory within a hierarchy, being:

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSNormal</td>
<td>Theory is present and may be written to.</td>
</tr>
<tr>
<td>TSLocked</td>
<td>Theory is present, and cannot be written to as it is locked.</td>
</tr>
<tr>
<td>TSAncestor</td>
<td>Theory is present, and cannot be written to as it is in an ancestor for some hierarchy.</td>
</tr>
<tr>
<td>TSDeleted</td>
<td>Theory has been deleted: the theory name may be reused for a new theory.</td>
</tr>
</tbody>
</table>

SML
datatype USER_DATUM =
    UD_Term of TERM * (USER_DATUM list)
| UD_Type of TYPE * (USER_DATUM list)
| UD_String of string * (USER_DATUM list)
| UD_Int of int * (USER_DATUM list);

Description This provides a monomorphic type of trees whose nodes are labelled by terms, types, strings or integers.

Uses This type is used in the type USER_DATA, and may be used elsewhere, as a means of storing data that may be represented in a “reasonably general” structure for ProofPower related purposes, which also is not polymorphic.

SML
type CONV;

Description This is the type name conventionally used for conversions, that is, inference rules whose last argument is a term, and whose result is an equation whose LHS is precisely that term (no α-conversion). Though it would be type correct, we conventionally do not use this type name for other functions of type \( \text{TERM} \rightarrow \text{THM} \).

Definition
\[
\text{type CONV} = \text{TERM} \rightarrow \text{THM};
\]

SML
type SEQ;

Description This is the type of sequents, consisting of a list of assumptions and a conclusion.

Definition
\[
\text{type SEQ} = (\text{TERM list}) * \text{TERM};
\]

\(=\#\) provides a strict equality test on sequents, \(\sim\#\) provides an equality test on the sequents up to α-convertibility and order of assumptions.
**SML**

type THEORY_INFO;

**Description**  This is a labelled record type containing certain information associated with a theory.

<table>
<thead>
<tr>
<th>Label</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>THEORY_STATUS</td>
<td>Current status of the theory.</td>
</tr>
<tr>
<td>inscope</td>
<td>bool</td>
<td>True if the theory is currently in scope (i.e. can its theorems, types and constants be usefully referred to).</td>
</tr>
<tr>
<td>contents</td>
<td>THEORY</td>
<td>The theory contents.</td>
</tr>
<tr>
<td>children</td>
<td>int list</td>
<td>List of the immediate children of the theory.</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>The name of the theory, as a string.</td>
</tr>
</tbody>
</table>

**SML**

type THEORY;

**Description**  A theory is a named collection of type names, constant names, axioms, definitions and theorems. In the abstract data type of theorems, the “names” of theories are represented as integers. For each type name the arity of the type is recorded and for each constant name its type is recorded. In order to allow deletion of types, constants, axioms and definitions. So-called level numbers are used to enables theorems that may depend on deleted material to be identified and rejected. In order for non-critical information such as operator fixity to be stored, a theory also includes a user-data slot which may be used to encode such information.

A theory is represented as a labelled record type, as follows:

<table>
<thead>
<tr>
<th>Label</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>int</td>
<td>Internal representation of theory name.</td>
</tr>
<tr>
<td>ty_env</td>
<td>{arity : int, level : int} OE_DICT</td>
<td>A dictionary indexed by type constructor names, returning arity, and definition level.</td>
</tr>
<tr>
<td>con_env</td>
<td>{ty : TYPE, level : int} OE_DICT</td>
<td>A dictionary indexed by constant name, returning the type and definition level.</td>
</tr>
<tr>
<td>parents</td>
<td>int list</td>
<td>Internal representations of names of parents of theory.</td>
</tr>
<tr>
<td>del_levels</td>
<td>(int * int) list</td>
<td>A list of ranges of deleted definition levels — if empty then no levels have been deleted.</td>
</tr>
<tr>
<td>axiom_dict</td>
<td>THM OE_DICT</td>
<td>A dictionary of axioms.</td>
</tr>
<tr>
<td>defn_dict</td>
<td>THM OE_DICT</td>
<td>A dictionary of definitions.</td>
</tr>
<tr>
<td>thm_dict</td>
<td>THM OE_DICT</td>
<td>A dictionary of theorems.</td>
</tr>
<tr>
<td>current_level</td>
<td>int</td>
<td>The current definition level.</td>
</tr>
<tr>
<td>user_data</td>
<td>USER_DATA ref</td>
<td>The user data stored in the theory.</td>
</tr>
</tbody>
</table>

**SML**

type THM;

**Description**  This is the abstract data type of theorems in **ProofPower**, whose primitive constructors are the inference rules and extensional mechanisms of the abstract data type. $|=|$ provides a strict equality test on the conclusion and assumptions of theorems, $\sim|=|$ provides an equality test on the conclusion and assumptions of theorems up to $\alpha$-convertibility and order of assumptions.
SML

\texttt{type USER\_DATA;}

\textbf{Description}  This is the type of a store for objects of type \textit{USER\_DATUM}. It is implemented as:

\texttt{ML \texttt{type USER\_DATA = USER\_DATUM S\_DICT;}}

\textbf{Uses}  Within the type \textit{THEORY} it is used to include such details as the fixity of types and constants.
6.2 Symbol Table

```sml
signature SymbolTable = sig

Description This is the signature for the structure which contains the symbol table and its access functions. This structure contains private functions which are invoked as one navigates around the theory database. These private functions may give rise to error 20001 if the theory database user data has been corrupted (e.g. by explicit and incorrect use of the lower level interfaces).

Any of the functions in the structure which update the current theory may give rise to error 20002

Errors
20001 A symbol table entry in theory ?0 is corrupt (use restore_defaults to clear)
20002 The current theory, ?0, is not open for writing
20003 Internal error: ?0

val declare_alias : (string * TERM) -> unit;

Description declare_alias (s, c) declares s as an alias for the constant c. s must comply with the HOL lexical rules for an identifier.

Errors
20301 The term ?0 is not a constant
20302 The string ?0 is already in use as an alias for ?1
20305 The constant ?0 is not in scope
20306 The string ?0 is not an identifier

val declare_binder : string -> unit;

Description declare_binder s declares s to have the syntactic status of a binder in the current context. s must comply with the HOL lexical rules for an identifier and must not be the string ",`.

See Also undeclare_fixity

Errors
20201 A fixity declaration is not allowed for ?0 (which is not an identifier)
20202 Cannot change the fixity of `,`

val declare_const_language : string * string -> unit;

Description declare_const_language (s, l) adds the language indicator l to those associated with the name s when used as a constant in the current context.

Errors
20501 There is no constant called ?0 in the current context
```
6.2. Symbol Table

SML

val declare_left_infix : (int * string) -> unit;
val declare_right_infix : (int * string) -> unit;
val declare_infix : (int * string) -> unit;

Description  declare_left_infix (p, s) declares s to have the syntactic status of an left associative infix operator with precedence p in the current context.  s must comply with the HOL lexical rules for an identifier.

Similarly, declare_right_infix is used to declare right associative operators.  declare_infix is provided for compatibility with earlier versions of the system and is the same as declare_right_infix.

See Also  undeclare_fixity

Errors

20201  A fixity declaration is not allowed for ?0 (which is not an identifier)

SML

val declare_nonfix : string -> unit;

Description  declare_nonfix s undoes the effect of a declaration of s to have special syntactic status (using declare_binder, declare_infix, declare_prefix or declare_postfix).

The effect of declare_nonfix s depends on the theory in which the special status for s was declared: if it was declared in the current theory, then the declaration is just removed; if in an ancestor theory then a declaration for s as a nonfix is inserted in the current theory.  (Thus in the first case, the syntactic status for s reverts to what it was before the earlier declaration, whereas in the second case the syntactic status will be suppressed.)

s must must not be the string “;”.

See Also  undeclare_fixity

Errors

20201  A fixity declaration is not allowed for ?0 (which is not an identifier)
20202  Cannot change the fixity of ‘;’
20203  There is no fixity declaration for ?0 in the current context

SML

val declare_postfix : (int * string) -> unit;

Description  declare_postfix (p, s) declares s to have the syntactic status of a postfix operator with precedence p in the current context.  s must comply with the HOL lexical rules for an identifier and must not be the string “;”.

See Also  undeclare_fixity

Errors

20201  A fixity declaration is not allowed for ?0 (which is not an identifier)
20202  Cannot change the fixity of ‘;’
<table>
<thead>
<tr>
<th>Description</th>
<th>SML</th>
</tr>
</thead>
<tbody>
<tr>
<td>declare_prefix (p, s)</td>
<td>val declare_prefix : (int * string) -&gt; unit;</td>
</tr>
<tr>
<td>(s) to have the syntactic status of a prefix operator with precedence (p) in the current context. (s) must comply with the HOL lexical rules for an identifier and must not be the string &quot;&lt;&quot;.</td>
<td></td>
</tr>
<tr>
<td>See Also</td>
<td>undeclare_fixity</td>
</tr>
<tr>
<td>Errors</td>
<td>20201 A fixity declaration is not allowed for ?0 (which is not an identifier)</td>
</tr>
<tr>
<td></td>
<td>20202 Cannot change the fixity of ','</td>
</tr>
<tr>
<td>declare_terminator (s)</td>
<td>val declare_terminator : string -&gt; unit</td>
</tr>
<tr>
<td>(s) checks that (s) is a valid terminator, and if so declares that (s) is to be used as a lexical terminator in the current context.</td>
<td></td>
</tr>
<tr>
<td>Errors</td>
<td>20101 The string ?0 is not a valid terminator. Terminators must start</td>
</tr>
<tr>
<td></td>
<td>with a symbolic character, must not contain spaces, and must not end with underscore, (\lambda) or (\gamma).</td>
</tr>
<tr>
<td></td>
<td>20102 The string ?0 is already declared as a terminator</td>
</tr>
<tr>
<td>declare_type_abbrev (s, [\alpha_1, \ldots, \alpha_k], \tau)</td>
<td>val declare_type_abbrev : (string * string list * TYPE) -&gt; unit;</td>
</tr>
<tr>
<td>(s) as a type abbreviation for the type (\tau). The identifier (s) may not already have been declared as a type abbreviation or be the name of a type constructor defined in the present context, in which cases a warning message is issued. (s) must comply with the HOL lexical rules for an identifier.</td>
<td></td>
</tr>
<tr>
<td>Errors</td>
<td>20401 The identifier ?0 is already declared as a type abbreviation</td>
</tr>
<tr>
<td></td>
<td>20402 The identifier ?0 is already declared as a type constructor</td>
</tr>
<tr>
<td></td>
<td>20407 The formal parameter list ?0 contains duplicate type variable names</td>
</tr>
<tr>
<td></td>
<td>20408 The string ?0 is not an identifier</td>
</tr>
<tr>
<td>expand_type_abbrev (s, [\tau_1, \ldots, \tau_k])</td>
<td>val expand_type_abbrev : (string * TYPE list) -&gt; TYPE;</td>
</tr>
<tr>
<td>(s) is the expansion of the type abbreviation (s) with respect to the arguments ([\tau_1, \ldots, \tau_k]).</td>
<td></td>
</tr>
<tr>
<td>Errors</td>
<td>20404 The identifier ?0 is not declared as a type abbreviation</td>
</tr>
<tr>
<td></td>
<td>20405 The type abbreviation ?0 should have ?1 argument not ?2</td>
</tr>
<tr>
<td></td>
<td>20406 The type abbreviation ?0 should have ?1 arguments not ?2</td>
</tr>
<tr>
<td>get_aliases (thy)</td>
<td>val get_aliases : string -&gt; (string * TERM) list;</td>
</tr>
<tr>
<td>(thy) returns information about identifiers which have been declared as aliases in the theory (thy). The return value is a list of pairs. Each pair contains a name and a constant for which that name is an alias. The same name may be used as an alias for several different constants, and if this happens there will be multiple entries for that alias in the list.</td>
<td></td>
</tr>
<tr>
<td>Errors</td>
<td>20601 There is no theory called ?0</td>
</tr>
</tbody>
</table>
6.2. Symbol Table

**SML**

val get_alias_info : string -> (string * TYPE)list OPT;

**Description**

get_alias_info c returns the list of aliases for the constant with name c, or Nil if c is not the name of a constant. For each pair (a, τ) in the result, a is an alias for c at instances of the type τ.

**SML**

val get_alias : (string * TYPE) -> string;

**Description**

get_alias(c, τ) returns the most appropriate alias for the constant with name c at the type τ. If no aliases for the name c have been declared then c is returned otherwise the most recent alias s associated with a type τ′ which can be instantiated to τ is returned.

**SML**

val get_binders : string -> string list;

**Description**

get_binders thy returns the list of identifiers which have been declared as binders in the theory thy.

**Errors**

20601 There is no theory called ?0

**SML**

val get_const_info : string -> (TYPE * ((string * TYPE)list)) OPT;

**Description**

get_const_info a returns the information, (τ, cs), associated with the name a used as a constant name or an alias for a constant, if any. cs is the list of names and types of constants to which a might refer (as an alias or as the actual constant name). τ is the type to use for this name during type inference, namely, the antinifier of the types in cs.

**SML**

val get_const_language : string -> string list;

**Description**

get_const_language s returns the language indicators associated with the name s when used as a constant in the current context. If there is no constant called s, then get_const_language s returns the language indicator associated with the current theory. The language indicator is “HOL” for all identifiers supplied as part of the ICL HOL system. The head element of the list returned is the language indicator associated with the constant’s declaring theory.

**SML**

val get_current_language : unit -> string;

**Description**

get_current_language () returns the language indicator associated with the current theory.

**SML**

val get_current_terminators : unit -> string list list;

**Description**

get_current_terminators() returns the list of identifiers which have been declared as terminators in the current context using new_terminator. The names are returned in exploded form, i.e. as a list of strings each containing one character.

**SML**

val get_fixity : string -> Lex.FIXITY;

**Description**

get_fixity s returns the syntactic status of s in the current context.
val get_language : string -> string;

Description get_language thy returns the language indicator associated with the theory thy.

Errors
20601 There is no theory called ?0

val get_left_infixes : string -> (int * string) list;
val get_right_infixes : string -> (int * string) list;

Description get_left_infixes thy (resp. get_right_infixes thy) returns the list of identifiers (and associated precedences) which have been declared as left (resp. right) associative infix operators in the theory thy.

Errors
20601 There is no theory called ?0

val get_nonfixes : string -> string list;

Description get_nonfixes thy returns the list of identifiers which are declared as binder, infix, prefix or postfix in an ancestor of the theory thy, but have had that special status suppressed (using declare_nonfix) in the theory thy itself.

Errors
20601 There is no theory called ?0

val get_postfixes : string -> (int * string) list;

Description get_postfixes thy returns the list of identifiers (and associated precedences) which have been declared as postfix operators in the theory thy.

Errors
20601 There is no theory called ?0

val get_prefixes : string -> (int * string) list;

Description get_prefixes thy returns the list of identifiers (and associated precedences) which have been declared as prefix operators in the theory thy.

Errors
20601 There is no theory called ?0

val get_terminators : string -> string list;

Description get_terminators thy returns the list of identifiers which have been declared as terminators in the theory thy.

Errors
20601 There is no theory called ?0

val get_type_abbrev : string -> (string list * TYPE);

Description get_type_abbrev s returns the formal argument list and type associated with the type abbreviation s.

Errors
20404 The identifier ?0 is not declared as a type abbreviation
val get_type_abbrevs : string -> (string * (string list * TYPE)) list;

Description  get_type_abbrevs thy returns information about the type abbreviation declarations which have been made in the theory thy. The return value is a list of pairs. Each pair contains the name of the corresponding type abbreviation together with its formal arguments and the type for which it is an abbreviation.

Errors
20601  There is no theory called \?

val get_type_info : string -> (int * (string list * TYPE) OPT) OPT;

Description  get_type_info s returns the type information, if any, associated with s. See DS/FMU/IED/DTD020 for more information.

val get_undeclared_terminators : string -> string list;

Description  get_undeclared_terminators thy returns the list of identifiers whose status as terminators has been suppressed (with undeclare_terminator) in the theory thy.

Errors
20601  There is no theory called \?

val get_undeclared_type_abbrevs : string -> string list;

Description  get_undeclared_type_abbrevs thy returns the list of identifiers which have had their status as type abbreviations suppressed in the theory thy.

Errors
20601  There is no theory called \?

val get_undeclared_aliases : string -> (string * TERM) list;

Description  get_undeclared_aliases thy returns information about aliases which have been suppressed (with undeclare_alias) in the theory thy. The return value is a list of pairs. Each pair contains a name and a constant for which that name is no longer to be used as an alias. There may be more than one entry for a given name in the list (since several undeclare_alias commands may apply to one name).

Errors
20601  There is no theory called \?

val is_type_abbrev : string -> bool;

Description  is_type_abbrev s returns true iff. s is declared as a type abbreviation

val resolve_alias : (string * TYPE) -> TERM;

Description  resolve_alias(s, \tau) returns a term of the form mk\_const(c, \tau) where c is the “best” resolution for the identifier s. This best resolution will be \( s \) if \( s \) has been introduced as a constant of type \( \tau' \) where \( \tau' \) is an instance of \( \tau \). If \( s \) is an alias then \( c \) is taken from the alias declaration for \( s \) in which the aliased constant has a type \( \tau' \) which can be instantiated to \( \tau \). If more than one such declaration is applicable the most recent one is used.

Errors
20304  The identifier \?

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
val restore_defaults : unit → unit;

**Description**  
`restore_defaults()` may be used to clear corrupted symbol table information in the current theory. It does this by restoring the theory to the state it would have if no terminator, fixity, alias, type abbreviations or language declarations had been performed. A warning message is issued (and the interactive user is prompted as to whether to continue) before the operation is performed.

**Errors**  
20703  This operation will delete all symbol table information from theory ?0

val set_current_language : string → unit;

**Description**  
`set_current_language s` sets the language indicator associated with the current theory to `s`.

val undeclare_alias : (string * TERM) → unit;

**Description**  
`undeclare_alias(s, c)` reverses the effect of a declaration of `s` as an alias for the constant `c` in the current context. This includes the possibility that `s` is the name of `c` itself.

The precise effect depends on the theory in which the alias was declared: if it was declared in the current theory, then the declaration is just removed (so that if `s` is declared as an alias for `c` in an ancestor theory, `s` will still act as an alias for `c` in the current theory); if in an ancestor theory then arrangements are made in the current theory to prevent `s` acting as an alias for `c`.

If `s` is the name of `c` itself, the type inferrer will no longer recognise `s` as a reference to `c`. In this case, `c` may be accessed either via an alias or via an ML quotation. This gives a work-around for the potential problem when a theory contains a constant whose name is needed as a variable name in some application using the theory.

**Errors**  
20301  The term ?0 is not a constant  
20303  The identifier ?0 is not declared as an alias for ?1

val undeclare_terminator : string → unit

**Description**  
`undeclare_terminator s` removes `s` from the list of identifiers which act as terminators for parsing purposes in the current context.

**Errors**  
20103  ?0 is not in the list of terminators in the current context

val undeclare_type_abbrev : string → unit;

**Description**  
`undeclare_type_abbrev(s,[α_1, ..., α_k],τ)` reverses the effect of a declaration of `s` as a type abbreviation.

The precise effect depends on the theory in which the type abbreviation was declared: if it was declared in the current theory, then the declaration is just removed (so that if `s` is declared as a type abbreviation in an ancestor theory, `s` will revert to whatever that declaration said); if in an ancestor theory then arrangements are made in the current theory to prevent `s` being treated as a type abbreviation.

**Errors**  
20403  The identifier ?0 is not declared as a type abbreviation
### 6.3 The Kernel Interface

#### SML

**signature KernelInterface =**

**Description**  This is the signature of the structure that gives the standard interface to the logical kernel. This interface adds a layer of additional services to the kernel functionality. E.g., it is used to notify the parser and type-inferrer so that they operate correctly when the current theory changes. The functions in the structure KernelInterface should always be used in preference to direct use of the functions in the structure pp'Kernel except in coding extensions to the system that need to bypass these services.

#### Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>6013</td>
<td>?0 is ill-formed in current theory: type name ?1 is not declared</td>
</tr>
<tr>
<td>6014</td>
<td>?0 is ill-formed in current theory: type name ?1 does not have arity used</td>
</tr>
<tr>
<td>6015</td>
<td>?0 is ill-formed in current theory: constant name ?1 not declared</td>
</tr>
<tr>
<td>6038</td>
<td>?0 is ill-formed in current theory: constant name ?1 cannot have type used</td>
</tr>
</tbody>
</table>

The above are error messages various kinds of well-formedness check failures. A well-formedness check occurs on any types, terms and theorems saved in a theory, and thus these errors may occur for any function in this signature which saves types, terms or theorems in a theory.

#### SML

**datatype KERNEL_INFERENCE =**

- KISubstRule of (THM * TERM) list * TERM * THM * THM
- KISimple\(\Rightarrow\)EqRule of TERM * THM * THM
- KIInst\(\Rightarrow\)TypeRule of (TYPE * TYPE) list * THM * THM
- KI\(\Rightarrow\)Intro of TERM * THM * THM
- KI\(\Rightarrow\)Elim of THM * THM * THM
- KIASmRule of TERM * THM
- KIREflConv of TERM * THM
- KISimple\(\Rightarrow\)βConv of TERM * THM
- KISucConv of TERM * THM
- KISimple\(\Rightarrow\)βConv of TERM * THM
- KIListSimple\(\forall\)Intro of TERM list * THM * THM
- KIEqTransRule of THM * THM * THM
- KIMkAppRule of THM * THM * THM
- KI\(\Rightarrow\)⇒\(\Rightarrow\)MPRule of THM * THM * THM
- KIListSimple\(\forall\)Elim of TERM list * THM * THM
- KIEqConv of THM * THM
- KIStringConv of TERM * THM
- KICCharConv of TERM * THM
- KIEqSymRule of THM * THM
- KIKListSimple\(\forall\)Intro of TERM * THM * THM
- KIPlusConv of TERM * THM;

**val on_kernel_inference : (KERNEL_INFERENCE --> unit) --> unit;**

**Description**  The call on_kernel_inference f registers the function f to be called whenever a kernel inference rule is called successfully. Several functions may be registered and they will be called in order of registration.

A value of type KERNEL_INFERENCE is passed to represent the instance of the rule that has been called. The tuple forming the argument to each constructor of the type gives the arguments and result of the corresponding rule.
Chapter 6. THE MANAGEMENT OF THEORIES AND THEOREMS

SML

(* compactification_mask : integer control: default: compiler-dependent *)
val get_compactification_cache : unit -> TYPE list;
val set_compactification_cache : TYPE list -> unit;
val clear_compactification_cache : unit -> unit;

Description These functions and associated control value support compactification of objects
stored in the theory database.

set_compactification_cache and get_compactification_cache may be used at the beginning and
end of a ProofPower session to preserve the contents of the cache of type information which is
used to implement compactification. Internally, the cache is held as a rather more complex, and
much larger, data structure than a simple list of types and so clear_compactification_cache is used
automatically to empty the cache at the end of a session, thereby avoiding saving the data structure
in the database file. Restoring the cache from the list returned by get_compactification_cache using
set_compactification_cache is time-consuming and is not done automatically; however, doing this
using, e.g., the following lines of ML, may improve the space-saving in applications which are
built up in several sessions:

SML Example - End of Every Session
val saved_compactification_cache = get_compactification_cache();

SML Example - Beginning of Second and Later Sessions
set_compactification_cache saved_compactification_cache;

ML functions which compute terms can often be coded so as to produce terms in which common
subterms are shared. The compactification algorithm may actually increase the space occupied
by such terms. Producers of such functions may therefore wish to suppress the compactification
when the computed terms are stored in the theory database.

compactification_mask is an integer control which is treated as a bit-mask and may used to
suppress selected aspects of the compactification algorithm. The default value of 0 should be
correct for most normal specification and proof work. The significance of the bits in the mask is
as follows:

<table>
<thead>
<tr>
<th></th>
<th>Suppress compactification in new_axiom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Suppress compactification in new_const</td>
</tr>
<tr>
<td>2</td>
<td>Suppress compactification in new_type_defn</td>
</tr>
<tr>
<td>4</td>
<td>Suppress compactification in new_spec</td>
</tr>
<tr>
<td>8</td>
<td>Suppress compactification in save_thm</td>
</tr>
<tr>
<td>16</td>
<td>Suppress compactification in simple_new_defn</td>
</tr>
</tbody>
</table>

So, for example, if the mask is set to 47 (= 1 + 2 + 4 + 8 + 32), then compactification will only
be performed when save_thm is called. The default value depends on the Standard ML compiler:
63 (i.e., no compactification) for Poly/ML and 0 (i.e., full compactification) for Standard ML of
New Jersey.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
### Description

This is an encoding of the arguments of the functions of signature `KernelInterface` which change the state of the theory database. When used to notify the system of a change that has been made certain additional information is also included. If used to notify the system before a change is made the slots will be given “null” default values (\(\text{``''}, [], \text{asm_rule mk_t}\)).

<table>
<thead>
<tr>
<th>Operation</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>open_theory</code></td>
<td>( (\text{thy}, (\text{inthys}, \text{outthys})) )</td>
<td><code>thy</code> names the theory which has been opened. <code>inthys</code> names the theories which have come into scope. <code>outthys</code> names the theories which have gone out of scope.</td>
</tr>
<tr>
<td><code>new_parent</code></td>
<td>( (\text{thy}, \text{inthys}) )</td>
<td><code>thy</code> names the new parent theory. <code>inthys</code> names the theories which have come into scope.</td>
</tr>
<tr>
<td><code>SimpleNewDefn</code></td>
<td>( (\text{arg}, \text{thm}) )</td>
<td><code>arg</code> gives the argument to the operation. <code>thm</code> is the new defining theorem.</td>
</tr>
<tr>
<td><code>NewTypeDefn</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>NewSpec</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>NewAxiom</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### SEE ALSO

`on_kernel_state_change`, `before_kernel_state_change`
val checkpoint : string -> CHECKPOINT;
val rollback : CHECKPOINT -> unit;

Description  This opaque type and its associated functions implement a system for checkpointing and restoring the state of the theory hierarchy. It is intended primarily for programmatic use in applications that may need to undo multiple extensions to the logical contents of the theory and changes to user data. The check-pointing scheme is unable to keep track of theories, theorems, definitions etc. that have been deleted. Applications that may delete such objects must make their own arrangements for restoring the deleted objects.

The parameter to checkpoint is a theory name. The checkpoint returned contains the information required by rollback to roll the indicated theory and all its descendants back to the state it had when the checkpoint was taken. The theory becomes the current theory after the rollback.

Rolling back is done using delete_constr etc. and so rolling back the state of definitions and axioms is restricted to changes made in theories which did not have children when the checkpoint was taken. For uniformity, rollback does not attempt to restore the state of the theorems and the user data in theories which had children when the checkpoint was taken. A theory that has been introduced and has become a parent of a theory that existed when the checkpoint was taken will not be deleted (otherwise the child theory would also have to be deleted).

Messages 12015 to 12017 are reported by rollback as comments. In general, rollback will just report on the problem and continue trying to restore other theories. For example, if rollback is unable to delete a theory, it continues to attempt to restore the state of the definitions, etc. in the theories that are to be retained. This is an unlikely situation, since rollback unlocks a theory if necessary before trying to delete it, so it will only happen if the application using rollback has created a new theory hierarchy and a theory to be deleted has obtained ancestor status. Message 12020 is reported by rollback as a failure.

Errors
12015 it was not possible to delete theory ?0
12016 the theory ?0 has been deleted since the checkpoint was taken; this change cannot be rolled back
12017 a failure was reported while trying to restore theory ?0 (?1)
12020 the theory ?0 has been deleted since this checkpoint was taken and a new theory of the same name has been created. Rolling back to this checkpoint is not possible.

val =|− : THM * THM -> bool;
val ~|=|− : THM * THM -> bool;
val =|# : SEQ * THM -> bool;
val ~|==# : SEQ * SEQ -> bool;

Description  =|− provides a strict equality test on the conclusion and assumptions of theorems, ~|=|− provides an equality test on the conclusion and assumptions of theorems up to $\alpha$-convertibility and order of assumptions. =|# provides a strict equality test on sequents, ~|==# provides an equality test on the sequents up to $\alpha$-convertibility and order of assumptions.

val asms : THM -> TERM list;

Description  This returns the assumptions(hypotheses) of a theorem.

See Also  dest_thm
6.3. The Kernel Interface

```sml
val before_kernel_state_change : (KERNEL_STATE_CHANGE -> unit) -> unit

Description before_kernel_state_change f nominates f to be called before the theory database
is to be modified by functions from the signature KernelInterface. The argument to f encodes the
operation which caused the modification together with its arguments and certain other additional
information (usually sets to null defaults for this function). A list of such functions is maintained,
and the new function is put at the end of the list, which means it may, if desired undo or overwrite
the effects of a function nominated by an earlier call of before_kernel_state_change.

Functions handled by before_kernel_state_change might be used to raise errors to prevent the
state change occurring. This will prevent further checks or actions being made. Thus a careful
choice between before_ or on_ is called for.

See Also KERNEL_STATE_CHANGE, on_kernel_state_change
```

```sml
val compact_type : TYPE -> TYPE;
val compact_term : TERM -> TERM;
val compact_thm : THM -> THM;

Description These functions compactify type, term and theorem values, currently by common-
ing up type information so that only one ML instance of any type is used in the compactified
value. Depending on the value of the integer control variable compactification_mask, q.v., these
interfaces are invoked automatically as values are stored in the theory database.

The compactify XXX interfaces act as identify functions: compactify XXX x returns a value
which is equal to x (in the sense of =:, = $ or = |− as appropriate), but which usually occupies
significantly less space than x.
```

```sml
val concl : THM -> TERM;

Description This returns the conclusion of a theorem.

See Also dest_thm
```
val delete_axiom : string -> unit

**Description**  
`delete_axiom key` deletes the axiom stored under key `key` and any other object which depends on it from the current theory. If any objects do depend on the axiom, the interactive user will be notified and asked whether to proceed with the deletion.

After the deletion any theorems which have been proven since the introduction of the axiom will no longer be usable for further proof.

Note that the deletion will attempt to delete all necessary theorems before deleting constants, types, and axioms in single steps, and thus may fail with a partially modified theory. This is because checks in the interface may not be as definitive as those of the kernel. The “on kernel state change” functions will be notified as if all necessary single step deletions, of theorems, constants, types and axioms to achieve the goal had been done, but after all the actual changes have been made. The “before kernel state change” functions will be notified of all the changes, as if single steps, before any are made.

Errors

6037 Theory ?0 is locked  
6071 Theory ?0 is a read-only ancestor  
6076 Theory ?0 has child theories  
12003 Theory ?0 does not contain an axiom under key ?1  
12012 Deletion of ?0 would require the deletion of ?1

val delete_const : TERM -> unit

**Description**  
`delete_const c` deletes the constant `c` (or the constant with the same type, up to renaming of type variables) and any other object which depends on `c` from the current theory. If `c` is the application of a constant to some arguments then that constant is the one deleted. If any saved objects other than `c` and its defining theorem do depend on `c`, the interactive user will be notified and asked whether to proceed with the deletion.

After the deletion any theorems which have been proven since the definition of `c` will no longer be usable for further proof.

Note that the deletion will attempt to delete all necessary theorems before deleting constants, types, and axioms in single steps, and thus may fail with a partially modified theory. This is because checks in the interface may not be as definitive as those of the kernel. The “on kernel state change” functions will be notified as if all necessary single step deletions, of theorems, constants, types and axioms to achieve the goal had been done, but after all the actual changes have been made. The “before kernel state change” functions will be notified of all the changes, as if single steps, before any are made.

Errors

6037 Theory ?0 is locked  
6071 Theory ?0 is a read-only ancestor  
6076 Theory ?0 has child theories  
12001 Theory ?0 does not contain the constant ?1 with the supplied type  
12012 Deletion of ?0 would require the deletion of ?1  
12014 ?0 is not a constant or a constant applied to some arguments
6.3. The Kernel Interface

SML

val delete_theory : string -> unit;

**Description**  `delete_theory thy` removes the theory `thy` from the theory database. This means, for instance, that all theorems that were proven with the deleted theory as the current theory, and all constants and types declared within the theory, will become out of scope.

**Errors**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>12035</td>
<td>Theory ?0 is not present in the current hierarchy</td>
</tr>
<tr>
<td>6037</td>
<td>Theory ?0 is locked</td>
</tr>
<tr>
<td>6069</td>
<td>Theory ?0 is in scope</td>
</tr>
<tr>
<td>6071</td>
<td>Theory ?0 is a read-only ancestor</td>
</tr>
<tr>
<td>6076</td>
<td>Theory ?0 has child theories</td>
</tr>
</tbody>
</table>

SML

val delete_thm : string -> THM;

**Description**  `delete_thm key` deletes the theorem stored under key `key` from the current theory. It returns the deleted theorem.

**Errors**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>6037</td>
<td>Theory ?0 is locked</td>
</tr>
<tr>
<td>6046</td>
<td>Key ?0 is not used for a theorem in theory ?1</td>
</tr>
<tr>
<td>6071</td>
<td>Theory ?0 is a read-only ancestor</td>
</tr>
</tbody>
</table>

SML

val delete_to_level :
  { do_warn : bool,
    caller : string,
    target : string,
    level : int } -> (string * int) list * (string * TYPE) list;

val thm_level : THM -> int;

**Description**  `delete_to_level` deletes constants, types and axioms (and any theorems that may depend on them) down to a specified level number. `do_warn` specifies whether or not the user should be warned before doing this. `caller` is the name of the calling function for use in error messages. `target` is the name of the target being deleted for use in the warning message. `level` is the level of the constant, type or axiom which is the target to be deleted. The returned value comprises the lists of types and constants that have been deleted (with their arities and types).

The level numbers for constants and types may be retrieved using the data structure returned by `get_theory`. `thm_level` returns the level number associated with a theorem or axiom.
**SML**

```sml
val delete_type : string -> unit
```

**Description**  
`delete_type` `t` deletes the type constructor `t` and any other object which depends on `t` from the current theory. If any objects other than `t` and its defining theorem do depend on `t`, the interactive user will be notified and asked whether to proceed with the deletion.

After the deletion ny theorems which have been proven since the definition of `ty` will no longer be usable for further proof.

Note that the deletion will attempt to delete all necessary theorems before deleting constants, types, and axioms in single steps, and thus may fail with a partially modified theory. This is because checks in the interface may not be as definitive as those of the kernel. The “on kernel state change” functions will be notified as if all necessary single step deletions, of theorems, constants, types and axioms to achieve the goal had been done, but after all the actual changes have been made. The “before kernel state change” functions will be notified of all the changes, as if single steps, before any are made.

**Errors**

- 6037 Theory ?0 is locked
- 6071 Theory ?0 is a read-only ancestor
- 6076 Theory ?0 has child theories
- 12002 Theory ?0 does not contain the type constructor ?1
- 12012 Deletion of ?0 would require the deletion of ?1

---

```sml
val dest_thm : THM -> SEQ;
```

**Description**  
This returns the representation of a theorem as a sequent, i.e. as a list of assumptions and a conclusion.

**See Also**  
asms, concl
val do_in_theory : string -> ('a -> 'b) -> 'a -> 'b;

Description: do_in_theory thy f a will change to the named theory thy, apply f to a, and return to the theory in which it was called. It will not notify the kernel state change functions (e.g. on_kernel_state_change) when it changes to the named theory, nor will it notify them on its return. Thus for instance the symbol table mechanism, and so term parsing, will behave as if no theory change had taken place before the application of f to a. This refusal to notify causes this function to be faster than the appropriate two uses of open_theory.

The function prevents the application of f from once more changing the current theory to another, or functions that may delete the original theory. The block will provoke error 12011. These functions are:

|open_theory new_theory delete_theory

It will also discard any changes made by before_kernel_state_change during the application of f at its end.

The function will intercept any exceptions (including keyboard interrupts), and will attempt to remove the block on changing the current theory, and then return to the original theory. However, in certain circumstances (such as multiple keyboard interrupts, or use of pp' functions) the exception handler itself may be interrupted or be otherwise unable to complete its work. In these cases open_theory must be used by hand to notify the proof system of the correct theory and its context. If this raises the error 12011 then repeat the use of open_theory, as each raising of the error involves the removal of one block put in place by do_in_theory before the message is generated.

Errors

12011 Blocked from changing the current theory.
    This particular block has now been removed.
    Exceptionally, further blocks, giving the same
    error message, may still be in place. These blocks
    should be cleared now by repeatedly trying open_theory
    until this error message is not provoked.

12013 An internal error has corrupted the current theory
    data. Immediately make a call of open_theory
    to clear this internal error.

12203 The kernel interface tables were in an inconsistent state.
    The tables are now being rebuilt.
val duplicate_theory : (string * string) -> unit;

Description duplicate_theory oldthy newthy creates a new theory, called newthy with the same contents and parents as oldthy, but without any children. The current theory remains unchanged.

Uses To allow the user to modify and experiment with a theory that has child theories that are not involved in the experiment, and would perhaps clash with the experiment.

Errors
6026 Theory ?0 may not be duplicated
(it must always be in the scope of any opened theory)
6042 Theory ?0 may not be duplicated (the duplicate would not be a descendant of ?1)
12035 Theory ?0 is not present in the current hierarchy
6040 Theory ?0 is already present in current theory hierarchy

To ensure that the duplicate theory can be opened by open_theory (q.v.) the system will prevent the duplication of theories which would give rise to error 6017 of open_theory if opened, and attempts to create such duplicates will give rise to error 6026 or 6042.

val get_ancestors : string -> string list;

Description This returns all the ancestors of the named theory, including the theory itself. The named theory is the last name in the list returned. The name of the parent first added to the named theory is next to last, preceded by its ancestors. All these are preceded by the second parent theory and its ancestors, apart from those already added. These are preceded by any unnoted ancestors of the third, fourth, etc parents of the named theory. The order in the list of the ancestors of the parent theories is determined recursively by this ordering.

Errors
12035 Theory ?0 is not present in the current hierarchy

val get_axioms : string -> (string list * THM) list;
val get_axiom_dict : string -> THM OE_DICT;

Description get_axioms returns all the axioms stored in the indicated theory together with the keys under which they are stored.

get_axiom_dict returns the mapping of keys to axioms represented as an order-preserving efficient dictionary.

Errors
12035 Theory ?0 is not present in the current hierarchy

val get_axiom : string -> string -> THM;

Description get_axiom theory key returns the axiom with key key, found in theory theory.

To improve performance, this function uses a cache containing the values of previous calls. This cache is rebuilt when open_theory is called, by removing entries that have gone out of scope. Opening a theory such as basic_hol that is low down in the theory hierarchy will reclaim the memory occupied by the cache.

Errors
12035 Theory ?0 is not present in the current hierarchy
12005 Theory ?0 does not have an axiom with key ?1
12010 Theory ?0 is not in scope

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
6.3. The Kernel Interface

SML

val `get children` : string -> string list;

Description  This returns the immediate children of the named theory, (not including the theory itself).

Errors

12035  Theory ?0 is not present in the current hierarchy

SML

val `get consts` : string -> TERM list;

Description  This returns (most general instances of) all the constants stored in a theory.

Errors

12035  Theory ?0 is not present in the current hierarchy

SML

val `get const keys` : string -> `E_KEY` list;

Description  This returns the efficient dictionary keys that represent the names of the constants stored in a theory.

Errors

12035  Theory ?0 is not present in the current hierarchy

SML

val `get const theory` : string -> string;

Description  `get const theory c` returns the name of the theory in which the constant `c` is defined.

Errors

12201  There is no constant called ?0 in the current context

SML

val `get const type` : string -> TYPE OPT;

Description  If a constant with the given name is in scope, then its type is returned, otherwise Nil.

Uses  This is likely to be often used just as a rapid test for a constant being in scope.

See Also  `get const info`

SML

val `get current theory name` : unit -> string;

Description  Returns the name of the current theory.

SML

val `get current theory status` : unit -> THEORY STATUS;

Description  This returns the current theory’s status.
SML

val get_defns : string —> (string list * THM) list;
val get_defn_dict : string —> THM OE_DICT;

Description  get_defns returns all the defining theorems stored in the indicated theory together with the keys under which they are stored.

get_defn_dict returns the mapping of keys to defining theorems represented as an order-preserving efficient dictionary.

Errors 12035 Theory ?0 is not present in the current hierarchy

SML

val get_defn : string —> string —> THM;

Description  get_defn theory key returns the definition with key key, found in theory theory.

To improve performance, this function uses a cache containing the values of previous calls. This cache is rebuilt when open_theory is called, by removing entries that have gone out of scope. Opening a theory such as basic_hol that is low down in the theory hierarchy will reclaim the memory occupied by the cache.

Errors 12035 Theory ?0 is not present in the current hierarchy
12004 Theory ?0 does not have a definition with key ?1
12010 Theory ?0 is not in scope

SML

val get_descendants : string —> string list;

Description  This returns all the descendants of the named theory, including itself.

Errors 12035 Theory ?0 is not present in the current hierarchy

SML

val get_parents : string —> string list;

Description  This returns the immediate parents of the named theory, (not including the theory itself).

Errors 12035 Theory ?0 is not present in the current hierarchy

SML

val get_theory_names : unit —> string list;
val theory_names : unit —> string list;

Description  These return the list of undeleted theories in the current hierarchy, whether in scope or not. theory_names is an alias for get_theory_names.

SML

val get_theory_status : string —> THEORY_STATUS;

Description  This returns the status of the indicated theory.

Errors 12035 Theory ?0 is not present in the current hierarchy
### 6.3. The Kernel Interface

SML

```sml
val get_theory : string -> THEORY;
val get_theory_info : string -> THEORY_INFO;
```

**Description**  These functions return the data structures associated with a theory in the logical kernel.

**Errors**  
12035 Theory ?0 is not present in the current hierarchy

SML

```sml
val get_thms : string -> (string list * THM) list;
val get_thm_dict : string -> THM OE_DICT;
```

**Description**  get_thms returns all the theorems stored in the indicated theory together with the keys under which they are stored.

get_thm_dict returns the mapping of keys to theorems represented as an order-preserving efficient dictionary.

**Errors**  
12035 Theory ?0 is not present in the current hierarchy

SML

```sml
val get_thm : string -> string -> THM;
```

**Description**  get_thm theory key returns the theorem with key key, found in theory theory.

To improve performance, this function uses a cache containing the values of previous calls. This cache is rebuilt when open_theory is called, by removing entries that have gone out of scope. Opening a theory such as basic_hol that is low down in the theory hierarchy will reclaim the memory occupied by the cache.

**Errors**  
12035 Theory ?0 is not present in the current hierarchy

12006 Theory ?0 does not have a theorem with key ?1

12010 Theory ?0 is not in scope

SML

```sml
val get_types : string -> TYPE list;
```

**Description**  This returns (canonical applications of) all the type constructors stored on a theory.

**Errors**  
12035 Theory ?0 is not present in the current hierarchy

SML

```sml
val get_type_arity : string -> int OPT;
```

**Description**  If a type with the given name is in scope, then its arity is returned, otherwise Nil.

**Uses**  This is likely to be often used just as a rapid test for a type being in scope.

**See Also**  get_type_info

SML

```sml
val get_type_keys : string -> E_KEY list;
```

**Description**  This returns the efficient dictionary keys that represent the names of the type constructors stored in a theory.

**Errors**  
12035 Theory ?0 is not present in the current hierarchy
142 Chapter 6. THE MANAGEMENT OF THEORIES AND THEOREMS

SML

val get_type_theory : string -> string;

Description get_type_theory ty returns the name of the theory in which the type constructor ty is defined.

Errors 12202 There is no type constructor called ?0 in the current context

SML

val get_user_datum : string -> string -> USER_DATUM;

Description get_user_datum thy key returns the value stored in the user data slot allocated to key in the theory thy, if any.

Errors 12035 Theory ?0 is not present in the current hierarchy 12009 No user data stored under key ?0 in theory ?1

SML

val is_theory_ancestor : string -> string -> bool;

Description is_theory_ancestor thy1 thy2 returns true if thy1 is an ancestor of thy2 within the current hierarchy.

Errors 12035 Theory ?0 is not present in the current hierarchy

This failure arises if either theory name is not present in the current hierarchy.

SML

val kernel_interface_diagnostics : bool -> {
  clean_flag : bool,
  const_thys : int list E_DICT list,
  type_thys: int list E_DICT list,
  int_thy_names : int E_DICT,
  in_scope : int list};

Description This function can be used to examine and optionally reset internal state used by the kernel interface module. It is intended for diagnostic purposes. If the argument is false, it just returns a representation of the state; if true, it also sets the internal state so that the next call on any operation such as get_const_theory will cause the state to be recalculated.

SML

val list_save_thm : (string list * THM) -> THM

Description list_save_thm(keys, thm) causes thm to be save under the keys keys in the current theory. The saved theorem is returned as the function’s result. If there is a conjecture stored under any of the keys in the current theory, the theorem must prove each such conjecture, i.e., its conclusion must be the same as the conjecture and it must have an empty assumption list.

See Also new_conjecture, is_proved_conjecture

Errors 6031 Key list may not be empty 6037 Theory ?0 is locked 6039 Key ?0 has already been used for a theorem in theory ?1 6071 Theory ?0 is a read-only ancestor 103101 This theorem does not prove the conjecture stored under key ?0
6.3. The Kernel Interface

SML
\begin{verbatim}
val lock_theory : string -> unit;
\end{verbatim}

**Description**  
`lock_theory thy` causes `thy` to be locked. The contents of a locked theory are protected from further changes. A locked theory may be unlocked using `unlock_theory(q.v.)`.

**Errors**
- 12035 Theory ?0 is not present in the current hierarchy
- 6037 Theory ?0 is locked
- 6071 Theory ?0 is a read-only ancestor

SML
\begin{verbatim}
val new_axiom : (string list * TERM) -> THM
\end{verbatim}

**Description**  
`new_axiom(keys, tm)` stores the boolean term `tm` an axiom in the current theory as an axiom under keys `keys`.

**Errors**
- 3031 ?0 is not of type `⌜BOOL⌝`
- 6031 Key list may not be empty
- 6037 Theory ?0 is locked
- 6047 Key ?0 has already been used for an axiom in theory ?1
- 6071 Theory ?0 is a read-only ancestor

SML
\begin{verbatim}
val new_const : (string * TYPE) -> TERM;
\end{verbatim}

**Description**  
`new_const(name, type)` introduces a new constant (with no defining theorem) called `name`, with most general type `type`, into the current theory.

**Errors**
- 6037 Theory ?0 is locked
- 6049 There is a constant called ?0 already in scope
- 6063 There is a constant called ?0 in the descendants of the current theory
- 6071 Theory ?0 is a read-only ancestor

SML
\begin{verbatim}
val new_parent : string -> unit;
\end{verbatim}

**Description**  
Adds the given parent theory to the list of parents of the current theory, considered as a set. It will fail if the parent theory does not exist; is already a parent of the current theory; or if making it a parent would cause a clash by bringing a new theory into scope (perhaps the new parent itself) that declares a new type or constant that is already in scope, or is declared in the descendants of the current theory.

**Errors**
- 12035 Theory ?0 is not present in the current hierarchy
- 6037 Theory ?0 is locked
- 6067 Making ?0 a parent would cause a clash
- 6071 Theory ?0 is a read-only ancestor
- 6082 Theory ?0 is already a parent
- 6084 Suggested parent ?0 is a child of the current theory
**val new_spec : (string list * int * THM) -> THM;**

**Description**  
`new_spec (keylist, ndef, ‘\(\exists x_1, \ldots, x_n \cdot p[x_1, \ldots, x_n]\)’)` will introduce `ndef` new constants named and typed from the `x_i`. It will also save a defining theorem under each of the keys in `keylist` in the current theory of the form ‘\(\vdash p[c_1, \ldots, c_n]\)’ where `c_i` is the constant with the name and type of `x_i`. If either the constant or theorem introduction fails then the function will not change the current theory.

**Errors**

6016  Existentially bound variable `0` is repeated in theorem `1`
6031  Key list may not be empty
6037  Theory `0` is locked
6044  Must define at least one constant
6049  There is a constant called `0` already in scope
6051  Key `0` has already been used for a definition in theory `1`
6053  `0` must not have assumptions
6056  `0` is a free variable in `1`
6062  `0` are free variables in `1`
6060  `0` is not of the form: ‘\(\vdash \exists x_1 \ldots x_n \cdot p[x_1, \ldots, x_n]\)’
where the ‘\(x_i\)’ are variables, and `n( = `1`) is the number of constants to be defined
6061  the body of `0` contains type variables not found in type of constants to be defined, the variables being: `1`
6063  There is a constant called `0` in the descendants of the current theory
6071  Theory `0` is a read-only ancestor
6081  Sets of type variables in `0` and `1` differ

**val new_theory : string -> unit;**

**Description**  
`new_theory thy` adds a new, empty, theory called `thy` to the theory database. The empty theory has no declarations within it, but does have the current theory as its sole parent. The new theory then becomes the current theory.

**Errors**

6040  Theory `0` is already present in current theory hierarchy
SML

\[
\text{val new_type_defn :}
\]
\[
\quad \text{(string list * string * string list * THM) -> THM;}
\]

**Description**  
`new_type_defn (keys, name, typars, defthm)` declares a new type with name `name`, and arity the length of `typars`. It creates a defining theorem for the type, saves it in the current theory under the keys `keys`. It returns the defining theorem. `defthm` must be a valid well-formed theorem of the form:

\[
\vdash \exists x : \text{type} \cdot p\ x
\]

with no assumptions. The defining theorem will then be of the form:

\[
\vdash \exists f : \text{typars} \cdot \text{name} \rightarrow \text{type} \cdot
\quad \text{TypeDefn (p : type \rightarrow BOOL) f}
\]

where `TypeDefn` asserts that its predicate argument `p` is non-empty, and its function argument `f` is a bijection between the new type and the subset of `type` delineated by `p`.

**Errors**

6031  Key list may not be empty
6034  There is a type called `?0` in the descendants of the current theory
6037  Theory `?0` is locked
6045  There is a type called `?0` already in scope
6052  Key `?0` has already been used for an type definition theorem in theory `?1`
6053  `?0` must not have assumptions
6054  `?0` is not of the form: `\vdash \exists x \cdot p\ x`
6055  `?0` is not of the form: `\vdash \exists x \cdot p\ y` where `\{x\}` is a variable
6056  `?0` is a free variable in `?1`
6062  `?0` are free variables in `?1`
6057  `?0` contains type variables not found in type variable parameter list,
\quad type variables being: `?1`
6071  Theory `?0` is a read-only ancestor
6079  `?0` repeated in type parameter list
6088  `?0` is not of the form: `\vdash \exists x \cdot p\ y` where `\{x\}` equals `\{y\}`

---

SML

\[
\text{val new_type : (string * int) -> TYPE;}
\]

**Description**  
`new_type (name, arity)` introduces a new type constructor (with no defining theorem) called `name` with arity `arity` into the current theory. The function returns the new type with sufficient arguments `'1`, `'2`, … to provide a well-formed type.

**Errors**

6034  There is a type called `?0` in the descendants of the current theory
6037  Theory `?0` is locked
6045  There is a type called `?0` already in scope
6071  Theory `?0` is a read-only ancestor
6088  The arity of a type must be \( \geq 0 \)
**on_kernel_state_change**

**Description** on_kernel_state_change f nominates f to be called whenever the theory database is modified by a function from the signature KernelInterface. The argument to f encodes the operation which caused the modification together with its arguments and certain other additional information. A list of such functions is maintained, and the new function is put at the end of the list, which means it may, if desired undo or overwrite the effects of a function nominated by an earlier call of on_kernel_state_change.

Functions handled by on_kernel_state_change should not be coded to raise errors that are not handled by themselves, as the handler will not catch such errors either. If the function is to prevent a change from happening before_kernel_state_change should be used instead.

**See Also** KERNEL_STATE_CHANGE, before_kernel_state_change

**open_theory**

**Description** All specification and proof work is carried out in the context of some theory, referred to as the current theory. open_theory thy makes an existing theory thy the current theory.

**Errors**

- 6017 Theory ?0 may not be opened (it is not a descendant of ?1 which must be in scope)
- 12035 Theory ?0 is not present in the current hierarchy

Certain theories created when the system is constructed may not be subsequently opened, and attempts to open them give rise to error 6017.

**pending_reset_kernel_interface**

**Description** This function, applied to () takes a “snapshot” of the current state of the kernel interface module (comprising the “On Kernel State Change”, “Before Kernel State Change” and “On Kernel Inference” functions). The resulting snapshot, when applied to () will restore these functions to their state at the time of making the snap shot.

**Uses** To assist in saving the overall system state.

**save_thm**

**Description** save_thm(key, thm) causes thm to be save under the key key in the current theory. The saved theorem is returned as the function’s result. If there is a conjecture stored under the same key in the current theory, the theorem must prove the conjecture, i.e., its conclusion must be the same as the conjecture and it must have an empty assumption list.

**See Also** new_conjecture, is_proved_conjecture

**Errors**

- 6037 Theory ?0 is locked
- 6039 Key ?0 has already been used for a theorem in theory ?1
- 6071 Theory ?0 is a read-only ancestor
- 103101 This theorem does not prove the conjecture stored under key ?0
**val** set_user_datum : (string * USER_DATUM) \(\rightarrow\) unit;

**Description**  
set_user_datum(key, ud) assigns the new value ud to the user data slot allocated to key in the current theory. If an old value was present it will be overwritten.

**Errors**
- 6037 Theory ?0 is locked
- 6071 Theory ?0 is a read-only ancestor

**val** simple_new_defn : (string list * string * TERM) \(\rightarrow\) THM;

**Description**  
simple_new_defn (keys, name, value) declares a new constant with name name, and with most general type being the type of value in the current theory. It creates an equational theorem (i.e. of the form ‘\(\vdash\) name = value’), and saves it as a definition under keys keys in the current theory, provided the theorem is well-formed. If either the constant or theorem introduction fails then the function does not change the current theory. The body of value may not contain type variables that are not in the type of value itself.

**Errors**
- 6031 Key list may not be empty
- 6037 Theory ?0 is locked
- 6049 There is a constant called ?0 already in scope
- 6051 Key ?0 has already been used for a definition in theory ?1
- 6058 the body of ?0 contains type variables not found in type of term itself,
  the variables being: ?1
- 6059 ?0 contains the following free variables: ?1
- 6063 There is a constant called ?0 in the descendants of the current theory
- 6071 Theory ?0 is a read-only ancestor

**val** string_of_thm : THM \(\rightarrow\) string;

**Description**  
This returns a display of a theorem in the form of a string, with no inserted new lines, suitable for use with diag_string and fail.

**See Also**  
format_thm, a formatted string display of a theorem.

**val** thm_fail : string \(\rightarrow\) int \(\rightarrow\) THM list \(\rightarrow\) 'a;

**Description**  
thm_fail area msg thml first creates a list of functions from unit to string, providing displays of the list of theorems. It then calls fail with the area, msg and this list of functions. This allows theorems to be presented in error messages.

**val** thm_theory : THM \(\rightarrow\) string;

**Description**  
thm_theory thm returns the name of the theory which was current when thm was proven. This will succeed even if the theory is out of scope, but not if the theory has been deleted.

**Errors**
- 12007 ?0 proven in theory with internal name ?1,  
  which is not present in current hierarchy
SML

val unlock_theory : string -> unit;

Description unlock_theorythy causes the locked theory thy to be unlocked, so that the contents of thy may be changed.

Errors

12035 Theory 0 is not present in the current hierarchy
6068 Theory 0 has not been locked

SML

val valid_thm : THM -> bool;

Description This function uses the check for the validity of theorems: returning true if valid and false otherwise: it cannot raise exceptions.

Uses To preempt errors caused by the primitive inference rules, which raise uncatchable errors when given invalid theorems, and so return more helpful error messages.
6.4 Conjectures Database

```sml
val is_proved_conjecture: string -> string -> bool;
val get_proved_conjectures: string -> string list;
val get_unproved_conjectures: string -> string list;
```

**Description**  
is_proved_conjecture thy key returns true if the conjecture with key key in theory thy has been proved (i.e., there is a theorem stored under the same key in the theory which has the conjecture as its conclusion and has no assumptions).

get_proved_conjectures thy (resp. get_unproved_conjectures thy) returns the list of conjectures in theory thy which have (resp. have not) been proved in the sense described above.

**See Also**  
save_thm, list_save_thm, new_conjecture

**Errors**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>20601</td>
<td>There is no theory called ?0</td>
</tr>
<tr>
<td>103101</td>
<td>This theorem does not prove the conjecture stored under key ?0</td>
</tr>
<tr>
<td>103102</td>
<td>The theorem with key ?0 does not prove this conjecture</td>
</tr>
<tr>
<td>103103</td>
<td>Theory ?0 is not in scope</td>
</tr>
<tr>
<td>103802</td>
<td>There is no conjecture called ?0 in theory ?1</td>
</tr>
<tr>
<td>103803</td>
<td>The conjectures database in theory ?0 is corrupt</td>
</tr>
</tbody>
</table>

(use delete_all_conjectures to clear).
```ml
val new_conjecture : (string list * TERM) -> unit;
val get_conjecture: string -> string -> TERM;
val get_conjectures: string -> (string list * (int * TERM)) list;
val delete_conjecture: string -> TERM;
val delete_all_conjectures: unit -> unit;
```

**Description**  
`new_conjecture(keys, tm)` stores the boolean term `tm` as a conjecture in the current theory under keys `keys`. If any of the keys is also the key of a theorem saved in the current theory, then each such theorem must prove the conjecture, i.e., its conclusion must be the same as `tm` and it must have an empty assumption list.

`delete_conjecture key` deletes the conjecture stored in the current theory under key `key`. It returns the deleted conjecture.

`delete_all_conjectures()` deletes all the conjectures stored in the current theory. This may be used if, for some reason, the data structure used to store the conjectures becomes corrupted.

Note, when a constants or a type is deleted from a theory, conjectures that contain the deleted constant or type are automatically deleted from the current theory. Message 103804 is used as a comment to inform the user when this happens.

**See Also**  
`save_thm`, `list_save_thm`, `is_proved_conjecture`

**Errors**
- `3031` "0 is not of type \(\text{⌜BOOL\⌝}\)
- `6031` Key list may not be empty
- `20601` There is no theory called `0`
- `103101` The theorem `0` does not prove the conjecture with key `1`
- `103801` Key `0` has already been used for a conjecture in the current theory
- `103802` There is no conjecture called `0` in theory `1`
- `103803` The conjectures database in theory `0` is corrupt
  (use `delete_all_conjectures` to clear).
- `103804` Deletion of `0` has caused deletion of conjecture `1`: `2`
### 6.5 Theorem Finder

**SML**

```sml
datatype 'a TEST =
    TFun of 'a -> bool
  | TAll of 'a TEST list
  | TAny of 'a TEST list
  | TNone of 'a TEST list;

type THM_INFO_TEST = THM_INFO TEST;
```

**Description** The type `THM_INFO_TEST` is used for the parameters of general theorem finder functions, `gen_find_thm` and `gen_find_thm_in_theories` that represent search criteria. The constructor `TFun` is used to represent a basic criterion. `TAll`, `TAny` and `TNone` construct new criteria from old by conjunction, disjunction and negated disjunction respectively.

**See Also** `any_substring_tt` etc. (for ways of constructing basic criteria).

**SML**

```sml
datatype THM_TYPE = TTAxiom | TTDefn | TTSaved;

type THM.INFO = {
    theory : string,
    names : string list,
    thm_type : THM_TYPE,
    thm : THM
};
```

**Description** The types `THM_TYPE` and `THM.INFO` are used by the theorem finder functions, `find_thm` etc., to represent information about a theorem stored in a theory. The representation gives: the name of the theory; the name or names under which the theorem is stored; an indicator of whether the theorem is an axiom, a definition or a theorem that has been proved and saved; and the actual theorem.

**SML**

```sml
val find_thm : TERM list -> THM.INFO list;
```

**Description** This is a simple interface for finding theorems. `find_thm pats` searches for any theorems in the current theory and its ancestors that contains subterms matching each of the pattern terms `tms`.

The return value is a list of records containing the conclusion of the theorem and other useful information, see the description of the type `THM.INFO` for more details.

For example, if the theory `R` of real numbers is in scope, the following will find all theorems containing both real number addition and real number multiplication.

```
find_thm [⌜ x + y : R\}, ⌜ x * y : R\}];
```

**See Also** `gen_find_thm`
SML

val gen_find_thm_in_theories : THM_INFO_TEST -> string list -> THM_INFO list;
val gen_find_thm : THM_INFO_TEST -> THM_INFO list;

val any_substring_tt : string list -> THM_INFO TEST;
val all_substring_tt : string list -> THM_INFO TEST;
val no_substring_tt : string list -> THM_INFO TEST;
val any_subterm_tt : TERM list -> THM_INFO TEST;
val all_subterm_tt : TERM list -> THM_INFO TEST;
val no_subterm_tt : TERM list -> THM_INFO TEST;
val any_submatch_tt : TERM list -> THM_INFO TEST;
val all_submatch_tt : TERM list -> THM_INFO TEST;
val no_submatch_tt : TERM list -> THM_INFO TEST;

Description  

*gen_find_thm_in_theories* is the general theorem finder function. Its first parameter specifies the search criteria and its second parameter specifies the names of the theories to be searched. It returns a list representing the theorems satisfying the criteria. See the definitions of the parameter and return data types for more details.

*gen_find_thm* calls *gen_find_thm_in_theories* with the specified search criteria and the list of all ancestors of the current theory as the list of theories to search (this include the current theory). Thus it finds all the theorems that are currently in scope that match the specified criteria.

The remaining functions give convenient ways of specifying typical search criteria. These functions support three kinds of basic criterion: substring search criteria test for a specified string appearing as a substring of the name of the theorem; subterm search criteria test for the presence (up to \(\alpha\)-equivalence) in the conclusion of the theorem of a specified subterm; submatch search criteria test for the presence in the conclusion of the theorem of a subterm that is an instance of a specified pattern term. Given a list of strings or terms giving basic criteria, the functions test for theorems satisfying all of the criteria (all...), at least one of the criteria (any...) or none of the criteria (no...). The constructors of the data type *THM_INFO_TEST*, q.v., allow more complex logical combinations of criteria to be built up from these.

For example, the following will find all theorems in scope that have names containing “plus” or “minus” as a substring and that does not contain any natural number additions.

```sml
|gen_find_thm(TAll[any_substring_tt["plus", "minus"], no_subterm_tt[⌜$+:\mathbb{N} \to \mathbb{N} \to \mathbb{N}\]]);
```

See Also  

*find_thm*
7.1 General Inference Rules

SML

signature DerivedRules1 = sig

Description This provides the derived rules of inference in Release 001 of ICL HOL. Though other rules of inference may be introduced, this document’s signature should provide a core set, at least covering the common rules of natural deduction. It subsumes the inference rules of the abstract data type THM.

SML

signature DerivedRules2 = sig

Description This provides the further derived rules of inference for ICL HOL. They are primarily concerned with handling paired abstractions.

SML

signature Rewriting = sig

Description This provides the derived rewriting rule, conversions and tactics for ICL HOL.

SML

(* "illformed_rewrite_warning" *)

Description This flag modifies the behaviour of REWRITE_MAP_C and ONCE_MAP_WARN_C. When false (its default) it will not warn of illformed rewriting in subterms, with message 26002, though if no other rewriting occurs then error message 26003 will still be used. If true, then the warning will be given if some rewriting is successful, but elsewhere it is illformed.

SML

type CANON = ( * = THM -> (THM list) );

Description This is the type abbreviation for a canonicalisation function; such functions are typically used to derive consequences of a theorem meeting some desired criteria. An example is the rewriting canonicalisations which are used to transform theorems into lists of equational theorems for use in the rewriting conversions, rules and tactics.

Combinators are available to assist in the construction of new canonicalisation functions from old.

See Also THEN_CAN, ORELSCE_CAN, REPEAT_CAN, FIRST_CAN, EVERY_CAN as combinators, fail.can and id.can as building blocks for the combinators.

SML

val ALL_SIMPLE_\forall_C : CONV -> CONV;

Description This conversional applies its conversion argument to the body of a repeated simple universal quantification.

Errors As the failure of the conversion argument.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
### all_simple_\texttt{\_\_elim}

**Description**  Specialises all the simple universally quantified variables in a theorem:

\[
\frac{\Gamma \vdash \forall x_1 \ldots x_n \cdot t[x_1, \ldots, x_n]}{\Gamma \vdash t[x_1', \ldots, x_n']}
\]

where \(x_1', \ldots, x_n'\) are renamed from \(x_1, \ldots, x_n\) as necessary to avoid clashes with free variables in the assumption list, or duplicated names in the list of specialisations.

### all_simple_\texttt{\_\_β\_conv}

**Description**  A conversion to eliminate all instances of simple \(\beta\) redexes in a term, regardless of nesting, or even that the \(\beta\) redex was created as the result of an earlier reduction in the conversion’s evaluation.

\[
\frac{t = t'}{\Gamma \vdash t'}
\]

\(t'\) is \(t\) with all simple \(\beta\) redexes reduced.

**Uses**  This uses an optimised term traversal algorithm, superior in speed to the general term traversal algorithms used with conversions, and should be used in preference to them and \texttt{\_\_β\_conv}.

**Errors**  \(7020\ ?0\ contains\ no\ β-redexes\)

### all_simple_\texttt{\_\_β\_rule}

**Description**  Eliminate all instances of simple \(\beta\) redexes in a theorem, regardless of nesting, or even that the \(\beta\) redex was created as the result of an earlier reduction in the rule’s evaluation.

\[
\frac{\Gamma \vdash t}{\Gamma \vdash t'}
\]

\(t'\) is \(t\) with all \(\beta\)-redexes reduced.

**Errors**  \(7020\ ?0\ contains\ no\ β-redexes\)
### 7.1. General Inference Rules

<table>
<thead>
<tr>
<th>SML</th>
<th>val ALL_∧_C : CONV \rightarrow CONV; val ALL_∨_C : CONV \rightarrow CONV;</th>
</tr>
</thead>
</table>

**Description** These respectively apply their conversion argument to:

- All the conjuncts of a structure of conjuncts (including a term that is not a conjunct at all) failing only if the conversion fails for all the conjuncts.

- All the disjuncts of a structure of disjuncts (including a term that is not a disjunct at all) failing only if the conversion fails for all the disjuncts.

The result is simplified at any conjunct or disjunct where at least one branch had a successful application of the conversion and matches the appropriate theorems of:

\[
\begin{align*}
\vdash \forall t \cdot (T \land t \iff t) \land (t \land T \iff t) \land \neg (F \land t) \land \neg (t \land F) \land (t \land t \iff t) \\
\vdash \forall t \cdot (T \lor t) \land (t \lor T) \land (F \lor t \iff t) \land (t \lor F \iff t) \land (t \lor t \iff t)
\end{align*}
\]

**Errors** As the failure of the conversion argument.

<table>
<thead>
<tr>
<th>SML</th>
<th>val all_⇒_intro : THM \rightarrow THM;</th>
</tr>
</thead>
</table>

**Description** Discharge all members of assumption list using ⇒\_intro.

**Rule**

\[
\begin{array}{c}
\{ t_1, \ldots, t_n \} \vdash t \\
\vdash t_1 \Rightarrow \ldots \Rightarrow t_n \Rightarrow t \\
\end{array}
\]

**See Also** all\_∀\_elim which is faster, though the results are slightly opaque. list\_∀\_elim.

<table>
<thead>
<tr>
<th>SML</th>
<th>val all_∀_arb_elim : THM \rightarrow THM;</th>
</tr>
</thead>
</table>

**Description** Specialise all the quantifiers of a possibly universally quantified theorem with a machine generated variables or variable structures.

**Rule**

\[
\begin{array}{c}
\Gamma \vdash \forall vs1[x_1, y_1, \ldots] \cdot vs2[x_2, y_2, \ldots] \cdot \bullet \\
\vdash p[x_1, y_1, \ldots, x_2, y_2, \ldots] \\
\end{array}
\]

\[
\begin{array}{c}
\vdash p[x'_1, y'_1, \ldots, x'_2, y'_2, \ldots] \\
\end{array}
\]

∀\_arb\_elim

where \(x'_1, y'_1, \ldots\) are renamed from \(x_1, y_1, \ldots\) as necessary to avoid name clashes with free variables in the assumption list.

<table>
<thead>
<tr>
<th>SML</th>
<th>val all_∀_elim : THM \rightarrow THM;</th>
</tr>
</thead>
</table>

**Description** Specialises all the outer universal quantifications in a theorem:

\[
\begin{array}{c}
\Gamma \vdash \forall x_1 \ldots x_n \cdot t[x_1, \ldots, x_n] \\
\vdash t[x'_1, \ldots, x'_n] \\
\end{array}
\]

**See Also** all\_∀\_arb\_elim which is faster, though the results are slightly opaque. list\_∀\_elim.
val all\_intro : THM → THM;

**Description**  Generalises all the free variables (other than those in the assumption list) in a theorem:

\[ \Gamma \vdash t \quad \Gamma \vdash \forall x_1 \ldots x_n \cdot t \]

where \( x_1, \ldots, x_n \) are all the free variables of \( t \). The function introduces variables in their order of occurrence, so:

**Example**

\[ \text{all\_intro} (\vdash a \vee b) = \vdash \forall a \ b \cdot a \vee b \]

val all\_uncurry\_conv : CONV;

**Description**  Apply \( \forall\_uncurry\_conv \) (q.v) to the outer universal quantifications of a term, flattening those binders.

\[ \text{all\_uncurry\_conv} \quad \forall \ vs1[x_1,y_1,...] \ vs2[x_2,y_2,...] \ldots \bullet \ f[x_1,y_1,...,x_2,y_2,...] \]

where the \( vs_i[x_{-i},y_{-i},...] \) are variable structures at least one of which must not be a simple variable, built from variables \( x_{-i}, y_{-i}, ... \).

**Errors**

\[ 27041 \ ?0 \ is \ not \ of \ the \ form: \ \forall \ \ldots (x,y) \ \ldots \cdot f \]^\n
val all\_\exists\_uncurry\_conv : CONV;

**Description**  Apply \( \exists\_uncurry\_conv \) (q.v) to the outer existential quantifications of a term, flattening those binders.

\[ \text{all\_\exists\_uncurry\_conv} \quad \exists \ vs1[x_1,y_1,...] \ vs2[x_2,y_2,...] \ldots \bullet \ f[x_1,y_1,...,x_2,y_2,...] \]

where the \( vs[x,y,...] \) are variable structures with variables \( x,y,... \), at least one of which must not be a simple variable.

**See Also**  all\_\forall\_uncurry\_conv

**Errors**

\[ 27048 \ ?0 \ is \ not \ of \ the \ form: \ \exists \ \ldots (x,y) \ \ldots \cdot f \]^\n
© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
7.1. General Inference Rules

SML
val all_\_\_conv : CONV;

**Description** A conversion to eliminate all instances of \( \beta \) redexes, including paired abstraction redexes, in a term, regardless of nesting, or even that the \( \beta \) redex was created as the result of an earlier reduction in the conversion’s evaluation.

\[
\begin{array}{c}
\vdash t = t' \\
\text{all_\_\_conv}
\end{array}
\]

\( t' \) is \( t \) with all \( \beta \) redexes reduced.

**Uses** This uses an optimised term traversal algorithm, superior in speed to the general term traversal algorithms used with conversions, and should be used in preference to them and \( \beta \_conv \).

**See Also** all_\_\_\_conv which only handles simple \( \beta \)-redexes, but does a faster traversal if that is all that is required. all_\_\_rule.

**Errors**
\[27049 \ ?0 \ contains \ no \ \beta\_\-redexes\]

SML
val all_\_\_rule : THM \rightarrow THM;

**Description** Eliminate all instances of \( \beta \) redexes, including paired abstraction redexes, in the conclusion of a theorem, regardless of nesting, or even that the \( \beta \) redex was created as the result of an earlier reduction in the rule’s evaluation.

\[
\begin{array}{c}
\Gamma \vdash t \\
\Gamma \vdash t' \\
\text{all_\_\_rule}
\end{array}
\]

\( t' \) is \( t \) with all \( \beta \)-redexes reduced.

**See Also** all_\_\_conv for the conversion. all_\_\_\_\_rule which only handles simple \( \beta \)-redexes, but does a faster traversal if that is all that is required.

**Errors**
\[27049 \ ?0 \ contains \ no \ \beta\_\-redexes\]

SML
val AND\_OR\_C : (CONV \ast CONV) \rightarrow CONV;

**Description** \( c1 \ AND\_OR\_C \ c2 \) will succeed if it can apply one or both of \( c1 \) or \( c2 \). If it cannot compose the results of applying both conversions successfully (indicating an ill-formed conversion result) it will return the result of the first conversion application.

**See Also** THEN\_TRY\_C, ORELSE\_C, THEN\_C

**Errors** As the failure message of the second conversion (implying that neither conversion was successfully applied).
Chapter 7. PROOF IN HOL

SML

\textbf{app\_arg\_rule} : \textit{TERM} $\rightarrow$ \textit{THM} $\rightarrow$ \textit{THM};

\textbf{Description} Apply both sides of an equational theorem to an argument.

\begin{align*}
\Gamma \vdash f = g & \quad \Rightarrow & \quad \text{app\_arg\_rule} \quad \Gamma \vdash f \ x = g \ x
\end{align*}

\textbf{Errors}

\begin{align*}
6020 & \quad ?0 \text{ is not of the form: } ' \Gamma \vdash t1 = t2' \\
7025 & \quad \text{Sides of equation may not be applied to term}
\end{align*}

SML

\textbf{APP\_C} : (\textit{CONV} * \textit{CONV}) $\rightarrow$ \textit{CONV};

\textbf{Description} Apply one conversion to the operator of a combination, and a second to the operand.

\begin{align*}
\vdash f \ a = f' a' & \quad \Rightarrow & \quad \text{APP\_C} \quad (c1 : \textit{CONV}, \\
& & c2 : \textit{CONV}) \quad \Gamma \vdash f \ a
\end{align*}

where $c1$ $f$ gives $\vdash f = f'$, and $c2$ $f$ gives $\vdash a = a'$.

\textbf{Errors}

\begin{align*}
3010 & \quad ?0 \text{ is not of form: } '\Gamma \vdash t1 \ t2' \\
7110 & \quad \text{Results of conversions, } ?0 \text{ and } ?1, \text{ ill-formed or cannot be combined}
\end{align*}

Also as the failure of the conversions.

SML

\textbf{app\_fun\_rule} : \textit{TERM} $\rightarrow$ \textit{THM} $\rightarrow$ \textit{THM} ;

\textbf{Description} Apply a function to both sides of an equational theorem.

\begin{align*}
\Gamma \vdash a = b & \quad \Rightarrow & \quad \text{app\_fun\_rule} \quad \Gamma \vdash f \ a = f \ b
\end{align*}

\textbf{Errors}

\begin{align*}
6020 & \quad ?0 \text{ is not of the form: } ' \Gamma \vdash t1 = t2' \\
7024 & \quad ?0 \text{ may not be applied to each side of equation}
\end{align*}

SML

\textbf{app\_if\_conv} : \textit{CONV};

\textbf{Description} Move a function application into a conditional.

\begin{align*}
\vdash f(\text{if } a \text{ then } b \text{ else } c) = & \quad \Rightarrow & \quad \text{app\_if\_conv} \quad f(\text{if } a \text{ then } b \text{ else } c)
\end{align*}

\textbf{Errors}

\begin{align*}
7098 & \quad ?0 \text{ is not of the form: } 'f(\text{if } a \text{ then } b \text{ else } c)'
\end{align*}
7.1. General Inference Rules

### asm_elim

<table>
<thead>
<tr>
<th>SML</th>
<th>val asm_elim : TERM (\rightarrow) THM (\rightarrow) THM (\rightarrow) THM;</th>
</tr>
</thead>
</table>

**Description** Eliminate an assumption with reference to contradictory assumption lists.

\[
\begin{array}{c}
\Gamma_1, a' \vdash t; \Gamma_2, \neg a'' \vdash t' \\
\Gamma_1 \cup \Gamma_2 \vdash t
\end{array}
\]

***asm_elim*** \(\vdash a' \rightarrow \neg a'' \rightarrow t' \rightarrow t\)

where \(a, a'\) and \(a''\) as well as \(t\) and \(t'\) are \(\alpha\)-convertible. Actually, the assumptions don’t have to be present for the function to succeed.

**Errors**

- 3031 \(?0\) is not of type \(\vdash:\text{BOOL}\)
- 7029 \(?0\) and \(?1\) are not of the form: \(\vdash \Gamma_1, aa \vdash t'\) and \(\vdash \Gamma_2, \neg aa \vdash ta\)

### asm_inst_term_rule

<table>
<thead>
<tr>
<th>SML</th>
<th>val asm_inst_term_rule : (TERM * TERM) list (\rightarrow) THM (\rightarrow) THM;</th>
</tr>
</thead>
</table>

**Description** Parallel instantiation of term variables within a theorem’s conclusion and assumptions to some other values.

\[
\begin{array}{c}
\Gamma \vdash t[x_1, ..., x_n] \\
\Gamma' \vdash t[t_1, ..., t_n]
\end{array}
\]

***asm_inst_term_rule*** \([\vdash t[x_1, ..., x_n] \rightarrow \vdash t[t_1, ..., t_n]]\)

**See Also** inst_term_rule

**Errors**

- 3007 \(?0\) is not a term variable
- 6027 Types of element \(?0, ?1\) in term association list differ

### asm_inst_type_rule

<table>
<thead>
<tr>
<th>SML</th>
<th>val asm_inst_type_rule : (TYPE * TYPE) list (\rightarrow) THM (\rightarrow) THM;</th>
</tr>
</thead>
</table>

**Description** Parallel instantiation of some of the type variables of both the conclusion and assumptions of a theorem.

\[
\begin{array}{c}
\Gamma \vdash t[v_{1}, ..., v_{m}] \\
\Gamma' \vdash t[\sigma_1, ..., \sigma_n]
\end{array}
\]

***asm_inst_type_rule*** \(\vdash t[v_{1}, ..., v_{m}] \rightarrow \vdash t[\sigma_1, ..., \sigma_n]\)

**See Also** inst_type_rule

**Errors**

- 3019 \(?0\) is not a type variable

### asm_intro

<table>
<thead>
<tr>
<th>SML</th>
<th>val asm_intro : TERM (\rightarrow) THM (\rightarrow) THM;</th>
</tr>
</thead>
</table>

**Description** Introduce a new assumption to an existing theorem.

\[
\begin{array}{c}
\Gamma \vdash t_2 \\
\Gamma \cup \{t_1\} \vdash t_2
\end{array}
\]

***asm_intro*** \(\vdash t_2 \rightarrow \vdash t_2\)

**Errors**

- 3031 \(?0\) is not of type \(\vdash:\text{BOOL}\)
**val** asm_rule : TERM \rightarrow THM;

**Description**  “A term is true on the assumption that it is true.”

\[
\begin{array}{c}
\text{Rule} \\
\hline
\vdash \ t \vdash t \\
\hline
\text{asm_rule}
\end{array}
\]

A primitive inference rule.

**Errors**

3031  ?0 is not of type \(\vdash \text{BOOL}\)

---

**val** BINDER_.C : CONV \rightarrow CONV;

**Description**  Apply a conversion to the body of a binder term:

\[
\begin{array}{c}
\text{Rule} \\
\hline
\vdash (B \ x\bullet p[x]) = (B \ x\bullet pa[x]) \\
\hline
\text{BINDER}.C \\
\hline
(c : \text{CONV}) \\
\hline
\vdash B \ x\bullet p
\end{array}
\]

where \(c\) \(p[x]\) gives \(\vdash p[x] = pa[x]\), and \(B\) is a binder.

**Errors**

27035  ?0 is not of the form: \(\vdash B \ x\bullet p[x]\) where \(\vdash B\) is a binder and \(\vdash x\) a varstruct

7104  Result of conversion, ?0, ill-formed

Also as the failure of the conversion.

---

**val** CHANGED_.C : CONV \rightarrow CONV;

**Description**  Applies a conversion, and fails if either the conversion fails, has ill-formed results in certain ways, or it causes no change. Even \(\alpha\)-convertible changes count as a change for this purpose.

**Errors**

7032  Conversion failed to cause a change

7104  Result of conversion, ?0, ill-formed

It may also fail with the error message of the conversion argument.

---

**val** char_conv : CONV;

**Description**  This function defines the character literal constants, by giving a relationship between character literal constants and their ASCII code (derived by the Standard ML function \(\text{ord}\)). A character literal is indicated by the constant’s name starting with single backquote (‘), being a single other character, as well as being of type \text{CHAR}.

\[
\begin{array}{c}
\text{Rule} \\
\hline
\vdash _{\text{ML}}(\text{mk_char("c")})\vdash = \\
\hline
\text{char_conv} \\
\hline
\text{AbsChar}_{\text{ML}}\text{ord }\text{"c"}\vdash
\end{array}
\]

A primitive inference rule(axiom schemata).

**See Also**  mk_char

**Errors**

3024  ?0 is not a character literal
7.1. General Inference Rules

SML

```
val COND_C : (TERM -> bool) -> CONV -> CONV -> CONV;
```

**Description**  
$COND_C\ pred\ cnv1\ cnv2\ tm$ will be, if the term predicate $pred$ applied to $tm$ is true, then $cnv1\ tm$ and otherwise the $cnv2\ tm$.

**Errors**  
As the failure of the predicate or either conversion.

---

SML

```
val cond_thm : THM;
```

**Description**  
A convenient variant of the definition of the conditional.

**Theorem**

\[
\vdash \forall\ a\ t_1\ t_2\bullet (a\ then\ t_1\ else\ t_2) = \epsilon\ x\bullet ((a\ Leftrightarrow T)\Rightarrow x = t_1)\wedge ((a\ Leftrightarrow F)\Rightarrow x = t_2)
\]

**Errors**

7001  ?0 is not of form: ‘$\Gamma\vdash F$’
7031  ?0 is not of type ‘:BOOL’

---

SML

```
val contr_rule : TERM -> THM -> THM;
```

**Description**  
Intuitionistic contradiction rule:

**Rule**

\[
\Gamma\vdash F \quad F\vdash t \quad \Gamma\vdash t\]

**Errors**

7104 Result of conversion, ?0, ill-formed

Also as the failure of the conversion upon the conclusion of the theorem.

---

SML

```
val conv_rule : CONV -> THM -> THM;
```

**Description**  
Apply a conversion to the conclusion of a theorem, and do $\iff$ modus ponens between the original theorem and the result of the conversion

**Rule**

\[
\Gamma_1\vdash t \quad \Gamma_1\cup\Gamma_2\vdash t'\quad (c:\ CONV)
\]

where $c\ t$ gives $\Gamma_2\vdash t\iff t'$.

**Errors**

7104 Result of conversion, ?0, ill-formed

---

SML

```
val ct hm_eqn_cxt : CANON -> THM -> EQN_CXT;
```

**Description**  
This function applies a canonicalisation (see CANON) to a theorem, and then attempts to convert each of the list of resulting theorems into an equational context entry using thm_eqn_cxt (q.v.). The results are composed into an equational context (which is only a Standard ML list of equational context entries). Canonicalised theorems that cannot be converted by thm_eqn_cxt will be discarded.
\begin{verbatim}
val c_contr_rule : TERM -> THM -> THM;

Description  Classical contradiction rule:

\[
\frac{\Gamma, \neg t' \vdash F}{\Gamma \vdash t}
\]

\(c_contr_rule\)

Note that the argument is the unnegated form of what must be present in the assumption list for success. Works up to \(\alpha\)-conversion.

Errors
\begin{itemize}
  \item 7001 \(?0\) is not of form: ‘\(\Gamma \vdash F’
  \item 3031 \(?0\) is not of type ‘\(\top\)
  \item 7003 Negation of \(?0\) is not in assumption list
\end{itemize}
\end{verbatim}

\begin{verbatim}
val disch_rule : TERM -> THM -> THM;

Description  Prove an implicative theorem, removing, if \(\alpha\)-convertibly present, the antecedent of the implication from the assumption list, and failing if it is not present.

\[
\frac{\Gamma, t_1' \vdash t_2}{\Gamma \vdash t_1 \Rightarrow t_2}
\]

\(disch_rule\)

\textbf{See Also}  \(\Rightarrow\_intro\) (which does not fail if term not in assumption list)

Errors
\begin{itemize}
  \item 7031 \(?0\) not \(\alpha\)-convertibly present in assumption list
\end{itemize}
\end{verbatim}

\begin{verbatim}
val eq_match_conv1 : THM -> CONV ;

Description  This matches the LHS of an universally quantified (simple or by varstruct) equational theorem to a term, instantiating the RHS accordingly. The conversion will only instantiate its universal quantifications, and type variables not found within the assumptions, not its free term variables.

\[
\frac{\Gamma \vdash t = v[t_1,\ldots,t_n]}{\Gamma \vdash t = v[t_1,\ldots,t_n]}
\]

\(eq_match_conv1\)

where \(\top\) is \(\alpha\)-convertible to \(\top\). If there are free variables on the RHS of the supplied equational theorem (when stripped of all universal quantification) they will be renamed as necessary to avoid identification with any variables in \(t\).

This conversion may be partially evaluated with only its theorem argument.

\textbf{Uses}  In producing a limited rewriting facility, that only instantiates explicitly identified variables.

Errors
\begin{itemize}
  \item 27003 \(?0\) is not of the form ‘\(\Gamma \vdash \forall x_1 \ldots x_n \bullet u = v’
  \item 7076 Could not match term \(?0\) to LHS of theorem \(?1\)
\end{itemize}
\end{verbatim}
7.1. General Inference Rules

SML

val eq_match_conv : THM -> CONV ;

**Description**  This matches the LHS of an equational theorem to a term, instantiating the RHS accordingly. The equational theorem may be partially or fully universally quantified (simple or by varstruct), without affecting the result of the conversion.

**Conversion**

\[
\frac{\Gamma \vdash t = v'}{(\Gamma \vdash \forall \ldots \cdot u = v)} \quad \text{eq_match_conv}
\]

where \( v' \) is the result of applying to \( v \) the instantiation rules required to match \( u \) to \( t \) (including both term and type instantiation). If there are free variables on the RHS of the supplied equational theorem (when stripped of all universal quantification) they will be renamed as necessary to avoid identification with any variables in \( t \).

This conversion may be partially evaluated with only its theorem argument.

**See Also**  eq_match_conv1

**Errors**

7044  Cannot match ?0 and ?1
val eq_rewrite_thm : THM
val ⇐_rewrite_thm : THM
val ¬_rewrite_thm : THM
val ∧_rewrite_thm : THM
val ∨_rewrite_thm : THM
val ⇒_rewrite_thm : THM
val if_rewrite_thm : THM
val ∀_rewrite_thm : THM
val ∃_rewrite_thm : THM
val β_rewrite_thm : THM

Description  These are some of the default list of theorems used by those rewriting rules, conversions and tactics whose names do not begin with ‘pure’:

\[
\begin{align*}
\text{eq_rewrite_thm} & \vdash \forall x (x = x) \iff T \\
\text{⇐_rewrite_thm} & \vdash \forall t ((T \iff t) = t) \land ((F \iff t) = (\neg t)) \land (t \iff F) = (\neg t) \\
\text{¬_rewrite_thm} & \vdash \forall t (\neg\neg t) = t \land ((\neg T) = F) \land (\neg F) = T \\
\text{∧_rewrite_thm} & \vdash \forall t ((T \land t) = t) \land ((t \land T) = t) \land ((\neg (F \land t)) \land (\neg (t \land F)) \land (t \land t) = t \\
\text{∨_rewrite_thm} & \vdash \forall t ((T \lor t) \land (t \lor T) \land ((F \lor t) = t) \land ((t \lor F) = t) \land (t \lor t) = t \\
\text{⇒_rewrite_thm} & \vdash \forall t ((T \Rightarrow t) = t) \land ((F \Rightarrow t) = (\neg t)) \land (t \Rightarrow F) = (\neg t) \\
\text{if_rewrite_thm} & \vdash \forall t1 t2 \forall a ((if T then t1 else t2) = t1) \land (if F then t1 else t2) = t2 \\
\text{∀_rewrite_thm} & \vdash \forall t (\forall x t) = t \\
\text{∃_rewrite_thm} & \vdash \forall t (\exists x t) = t \\
\text{β_rewrite_thm} & \vdash \forall t1 t2 \forall a; t2 \forall b ((\lambda x t1) t2) = t1 \\
\end{align*}
\]

The theorems are saved in the theory “misc”, and given their design in the design for that theory.

See Also  fst_rewrite_thm, snd_rewrite_thm, fst_snd_rewrite_thm.

val eq_sym_conv : CONV;

Description  Symmetry of equality:

\[
\begin{align*}
\text{eq_sym_conv} & \vdash (t1 = t2) \iff (t2 = t1) \\
\end{align*}
\]

See Also  eq_sym_rule

Errors

\[
\begin{align*}
3014 & \text{ ?0 is not of form: } \vdash t = u
\end{align*}
\]
7.1. General Inference Rules

val eq_sym_rule : THM -> THM;

Description Symmetry of equality:

\[
\begin{align*}
\Gamma \vdash t_1 = t_2 & \Rightarrow \Gamma \vdash t_2 = t_1 \\
\end{align*}
\]

eq_sym_rule

A built-in inference rule.

See Also eq_sym_conv

Errors 6020 ?0 is not of the form: ‘\(\Gamma \vdash t_1 = t_2\)’

val eq_trans_rule : THM -> THM -> THM;

Description Transitivity of equality:

\[
\begin{align*}
\Gamma_1 \vdash t_1 = t_2; \Gamma_2 \vdash t_2' = t_3 & \Rightarrow \Gamma_1 \cup \Gamma_2 \vdash t_1 = t_3 \\
\end{align*}
\]

eq_trans_rule

where \(t_2\) and \(t_2'\) are \(\alpha\) convertible. A built-in inference rule.

Errors 6020 ?0 is not of the form: ‘\(\Gamma \vdash t_1 = t_2\)’
6022 ?0 and ?1 are not of the form: ‘\(\Gamma_1 \vdash t_1 = t_2\)’ and ‘\(\Gamma_2 \vdash t_2' = t_3\)’

where ‘\(\Gamma_1\)’ and ‘\(\Gamma_2\)’ are \(\alpha\)-convertible

val EVERY_CAN : CANON list -> CANON

Description EVERY_CAN is a canonicalisation function combinator which combines the elements of its argument using THEN_CAN:

\[
[\text{EVERY}_\text{CAN} \ [\text{can1}, \text{can2}, \ldots] = \text{can1} \ \text{THEN}_\text{CAN} \ \text{can2} \ \text{THEN}_\text{CAN} \ldots]
\]

See Also CANON

val EVERY_C : CONV list -> CONV;

Description Apply each conversion in the list, in the sequence given.

See Also THEN_C(which this function iterates)

Errors 7103 List may not be empty

or as the failure of any constituent conversion, or as THEN_C.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
### ext_rule

**Description** Extensionality of functions in ICL HOL.

**Rule**

\[
\frac{\Gamma \vdash f = g}{\Gamma \vdash \forall x \cdot f \ x = g \ x}
\]

where \(x\) is a machine-generated variable of appropriate type, not found free in the equational theorem.

**Errors**

- 6020: ?0 is not of the form: \(\ddash \Gamma \vdash t_1 = t_2\)
- 7026: ?0 is not an equation of functions

### fail_canon

**Description** This is a canonicalisation function which always fails. It is the identity for ORELSE_CAN.

**See Also** CANON

**Errors**

- 26201: Failed as requested

### fail_conv

**Description** This conversion always fails.

**Errors**

- 7061: Failed as requested

### fail_with_canon

**Description** This is a canonicalisation function which always fails by passing its arguments to fail (q.v.).

**See Also** fail_can

### fail_with_conv

**Description** This conversion always fails, with the error message being its string argument.

**Errors**

- 7075: ?0

### FIRST_CAN

**Description** FIRST_CAN is a canonicalisation function combinator which combines the elements of its argument using ORELSE_CAN:

\[
\text{FIRST}_\text{CAN} \ [\text{can1}, \text{can2}, \ldots] = \text{can1} \ \text{ORELSE}_\text{CAN} \ \text{can2} \ \text{ORELSE}_\text{CAN} \ \ldots
\]

**See Also** CANON

**Errors**

- 26202: the list of canonicalisation functions is empty
7.1. General Inference Rules

\begin{verbatim}
val FIRST_C : CONV list -> CONV;

Description  Attempt to apply each conversion in the list, in the sequence given, until one succeeds, or all fail.

See Also  ORELSE_C (which this function iterates)

Errors  
\begin{enumerate}
\item \textit{List may not be empty}
\end{enumerate}

or as the failure of the last conversion.
\end{verbatim}

\begin{verbatim}
val FORWARD_CHAIN_CAN : CANON list -> CANON;
val FC_CAN : CANON list -> CANON;

Description  \textit{FORWARD\_CHAIN\_CAN}, which has the alias \textit{FC\_CAN}, is a parameterised variant of \textit{fc\_canon}. Given a list of canonicalisation functions \textit{cans}, \textit{FC\_CAN cans} behaves as \textit{fc\_canon} would do if the line
\[ \vdash A \rightarrow \text{FIRST\_CAN cans A} \]
were inserted at the beginning of the table of transformations given in the description of \textit{fc\_canon}.

For example, \textit{fc\_canon1}, q.v., is the same as:
\[ \text{FC\_CAN } \left( (\text{fn } (x, y) \Rightarrow [x, y]) \leftrightarrow \text{elim} \right) \]

Uses  In tactic programming, or, occasionally interactively, typically in circumstances where neither \textit{fc\_canon} nor \textit{fc\_canon1} is able to generate enough implications.
\end{verbatim}
The asymmetry in the rules is deliberate. E.g., they derive \( A \Rightarrow B \Rightarrow A \Rightarrow C \) is transformed into \( B \Rightarrow A \Rightarrow C \). The transformation for \( A \Rightarrow B \) is only applied if it changes the theorem, and the last of the transformations is only applied if \( A \) is neither an implication nor \( F \).

The asymmetry in the rules is deliberate. E.g., they derive \( A \Rightarrow B \Rightarrow C \) from \( A \land B \Rightarrow C \), but not \( B \Rightarrow A \Rightarrow C \). This is intended to give slightly finer control and to result in less duplication of results in the intended application in \textit{forward-chain-tac}(q.v.).

\textbf{See Also} \textit{forward-chain-rule}, \textit{forward-chain-tac}, \textit{FC\_CAN}
SML

val forward_chain_rule : THM list -> THM list -> THM list;
val fc_rule : THM list -> THM list -> THM list;

Description  This is a rule which uses a list of possibly universally quantified implications and a list of other theorems to infer new theorems, using the matching modus ponens rule from the proof context, if present, or \( \Rightarrow \) match mp rule2 if current_ad_mmp_rule() returns Nil. (fc_rule is an alias for forward_chain_rule.) fc_rule imps ants returns the list of all theorems which may be derived by applying the matching modus ponens rule to a theorem from imps and one from ants. As a special case, if any theorem to be returned is determined to have \( \{\lnot F\} \) as its conclusion, the first such found will be returned as a singleton list. In order to work well in conjunction with fc_canon and fc_tac the theorems returned by the matching modus ponens rule are transformed as follows:

1. Theorems of the form: \( \vdash \forall x_1 \ldots \bullet t_1 \Rightarrow t_2 \Rightarrow \ldots \Rightarrow \lnot t_k \Rightarrow F \) have their final implication changed to \( t_k \).
2. Theorems of the form: \( \vdash \forall x_1 \ldots \bullet t_1 \Rightarrow t_2 \Rightarrow \ldots \Rightarrow t_k \Rightarrow F \) have their final implication changed to \( \Rightarrow \lnot t_k \).
3. All theorems are universally quantified over all the variables which appear free in their conclusions but not in their assumptions (using all_\( \forall \) intro).

Note that when the matching modus ponens rule is either \( \Rightarrow \) match_mp_rule2 or \( \Rightarrow \) match_mp_rule1, there is some control over the number of results generated, since variables which appear free in imps are not considered as candidates for instantiation.

The rule does not check that the theorems in its first argument are (possible universally) quantified implications.

See Also  forward_chain_tac, forward_chain_canon.

SML

val FORWARD_CHAIN <=> CAN : CANON list -> CANON;
val FC <=> CAN : CANON list -> CANON;

Description  These are just like FORWARD_CHAIN_CAN, q.v., except that they do not break up bi-implications. Thus, given a list of canonicalisation functions cans, FC <=> CAN cans behaves as fc_canon would do if the line

\[ \vdash A \rightarrow FIRST_CAN cans A \]

were inserted at the beginning of the table of transformations given in the description of fc_canon and all transformations (including those coming from the proof context) that eliminate bi-implications were suppressed.

Uses  In tactic programming, or, occasionally interactively, typically in circumstances where fc <=> canon is not able to generate enough implications.
Description  *forward_chain_\Leftrightarrow_canon* is a canonicalisation function very similar to *forward_chain_canon*, q.v. The difference is that *forward_chain_\Leftrightarrow_canon* suppresses all transformations which break up bi-implications. It is intended for use in situations where a bi-implication is to be used as a conditional rewrite rule.

For example, the tactic *ALL_ASM_FC_T1 fc_\Leftrightarrow_canon rewrite_tac []* can instantiate an assumption of the form \( \forall x_1 x_2 \ldots A \Rightarrow B \Rightarrow (C \Leftrightarrow D) \) and use the result to rewrite instances of \( C \).

**See Also**  *FC_T1, ALL_FC_T1* etc.

---

**SML**

```sml
val f_thm : THM;
```

**Description**  “Not False” is true.

**Theorem**

\[ \vdash \neg F \]

\[ f_{\text{thm}} \]

---

**SML**

```sml
val id_canon : CANON
```

**Description**  This is the identity for the canonicalisation function combinator *THEN_CAN*:

\[ id_{\text{canon}} \text{ thm} = \text{[thm]} \]

**See Also**  *CANON*

---

**SML**

```sml
val id_conv : CONV;
```

**Description**  This is an alias for *refl_conv*, reflecting the fact that *refl_conv* is the identity for the conversional *THEN_C*.

**Errors**

- 7061  Failed as requested

---

**SML**

```sml
val if_app_conv : CONV;
```

**Description**  Move a function application out of a conditional.

**Conversion**

\[ \vdash (\text{if } a \text{ then } f \, b \text{ else } f' \, c) = (f (\text{if } a \text{ then } b \text{ else } c)) \]

\[ \text{if}_{\text{app_conv}} \]

\[ (\text{if } a \text{ then } f \, b \text{ else } f' \, c)^\neg \]

where \( f \) and \( f' \) are \( \alpha \)-convertible, and \( f \) is used on the RHS of the resulting equational theorem

**Errors**

- 7037  ?\( \theta \) is not of the form: \( \text{if } a \text{ then } (f \, b) \text{ else } (g \, c)^\neg \)
- 7038  ?\( \theta \) is not of the form: \( \text{if } a \text{ then } (f \, b) \text{ else } (fa \, c)^\neg \)

where \( f^\neg \) and \( fa^\neg \) are \( \alpha \)-convertible
7.1. General Inference Rules

\[ \text{SML} \]
\[ \text{val if\_else\_elim : THM } \rightarrow \text{ THM}; \]

**Description**  Give the dependence of the else branch of a conditional upon the condition.

\[ \text{Rule} \quad \Gamma \vdash \text{if } tc \text{ then } tt \text{ else } te \quad \text{if\_else\_elim} \]
\[ \Gamma \vdash \neg tc \Rightarrow te \]

**Errors**  
7012  ?0 is not of the form: ‘\( \Gamma \vdash \text{if } tc \text{ then } tt \text{ else } te \)’

\[ \text{SML} \]
\[ \text{val if\_intro : TERM } \rightarrow \text{ THM } \rightarrow \text{ THM } \rightarrow \text{ THM}; \]

**Description**  Introduce a conditional, based on the assumptions of two theorems.

\[ \text{Rule} \quad \Gamma_1, \ a \vdash tt \ ; \ \Gamma_2, \neg a' \vdash et \quad \text{if\_intro} \]
\[ \Gamma_1 \cup \Gamma_2 \vdash \text{if } a \text{ then } tt \text{ else } et \quad \text{if\_intro} \]

where \( a \) and \( a' \) are \( \alpha \)-convertible. Actually, the assumptions may be missing, and the rule still works.

**Example**  
\( \vdash x = tt \), \( \vdash x = te \) (* hypothesis *)
\( \vdash \text{if } a \text{ then } (x = tt) \text{ else } (x = te) \) (* if\_intro if\_intro *)
\( \vdash x = \text{if } a \text{ then } tt \text{ else } te \) (* if\_fun\_rule *)

**Errors**  
3031  ?0 is not of type \( \Gamma:\text{BOOL} \)

\[ \text{SML} \]
\[ \text{val if\_then\_elim : THM } \rightarrow \text{ THM}; \]

**Description**  Give the dependence of the then branch of a conditional upon the condition.

\[ \text{Rule} \quad \Gamma \vdash \text{if } tc \text{ then } tt \text{ else } te \quad \text{if\_then\_elim} \]
\[ \Gamma \vdash tc \Rightarrow tt \]

**Errors**  
7012  ?0 is not of the form: ‘\( \Gamma \vdash \text{if } tc \text{ then } tt \text{ else } te \)’
This is the initial rewrite canonicalisation function, defined as

\[
\begin{align*}
\text{val initial\_rw\_canon} & = \\
& \text{REWRITE\_CAN} \\
& (\text{REPEAT\_CAN}(\text{FIRST\_CAN} [ \\
& \text{simple\_}\forall\text{\_rewrite\_canon}, \\
& \land\text{\_rewrite\_canon}, \\
& \text{simple\_}\neg\text{\_rewrite\_canon}, \\
& f\text{\_rewrite\_canon}, \\
& \iff t\text{\_rewrite\_canon}])); \end{align*}
\]

This is the repeated application of the first applicable operation in the following list:

1. stripping universal quantifiers;
2. dividing conjunctive theorems into their conjuncts;
3. changing \( \vdash \neg(t1 \lor t2) \) to \( \neg t1 \land \neg t2 \);
4. changing \( \vdash \exists x \bullet t \) to \( \forall x \bullet \neg t \);
5. changing \( \vdash \neg \neg t \) to \( t \);
6. changing \( \vdash \neg t \) to \( t \iff F \);
7. changing \( \vdash F \) to \( \vdash \forall x \bullet x \);
8. if none of the above apply, changing \( \vdash t \) to \( \vdash t = T \).

Finally, after all this canonicalisation we then universally quantify the resulting theorems in all free variables other than those that were free in the original.

This built-in inference rule.

See Also

asm\_inst\_term\_rule

Error Codes

3007 "?0 is not a term variable"
6027 "Types of element (?0, ?1) in term association list differ"
6028 "Instantiation variable ?0 free in assumption list"
General Inference Rules

\[ \text{val inst\_type\_rule : (TYPE \times TYPE) list \rightarrow THM \rightarrow THM;} \]

**Description** Parallel instantiation of some of the type variables of the conclusion of a theorem.

\[ \frac{\Gamma \vdash t[tyv_1,\ldots,tyv_n]}{\Gamma \vdash t[\sigma_1,\ldots,\sigma_n]} \]

\[ \textit{inst\_type\_rule \ talist \ thm} \]

will instantiate each type variable in \( \text{talist} \) with its associated type. It will decorate free variables that would become identified with other variables (both in conclusion and assumptions) by their types becoming the same and the names originally being the same. To instantiate types in the assumption list, see \( \textit{asm\_inst\_type\_rule} \).

A primitive inference rule.

**See Also** \( \textit{asm\_inst\_type\_rule} \) for something that also works on type variables in the assumption list.

\[ \begin{align*}
3019 & \ ?0 \ is \ not \ a \ type \ variable \\
6006 & \ \text{Trying to instantiate type variable } \ ?0, \ which \ occurs \ in \ assumption \ list
\end{align*} \]

\[ \text{val LEFT\_C : CONV \rightarrow CONV;} \]

**Description** Apply a conversion to the first operand of a binary operator:

\[ \frac{\vdash f \ a \ b = f \ a' \ b}{\text{\( \textit{LEFT\_C} \) \ \( \vdash a = a' \) \ \( \vdash f \ a \ b \)}} \]

where \( \vdash a = a' \). \( f \) may itself be a function application.

\[ \begin{align*}
3013 & \ ?0 \ is \ not \ of \ form: f \ a \ b \\
7104 & \ \text{Result of conversion, } ?0, \ \text{ill-formed}
\end{align*} \]

Also as the failure of the conversion.

\[ \text{val let\_conv : CONV;} \]

**Description** Eliminate an outermost \( \text{let...and...in...} \) construct.

\[ \frac{\vdash (\text{let } vs1[x_1,y_1,\ldots] = t_1 \\
\text{and } \ldots \text{ and } vsn[x_n,y_n,\ldots] = t_n \\
in t[x_1,\ldots,x_n,\ldots]} \}
\]

\[ \vdash t[t_1x_1,\ldots,t_1y_1,\ldots,t_nx_n,\ldots] \]

Where the \( t_\_ix \) is the component of \( t_\_i \) matching \( x_\_i \) when \( t_\_i \) matches \( vs_\_i[x_\_i,y_\_i,\ldots] \).

\[ 4009 \ \ ?0 \ is \ not \ of \ form: \text{\( let \ ... \ in \ ... \)} \]
val list_simple_\forall_elim : TERM list -> THM -> THM;

Description Generalised $\forall$ elimination.

\[
\begin{array}{c}
\Gamma \vdash \forall x_1 \ldots \ x_n \bullet t[x_1, \ldots, x_n] \\
\hline
\Gamma \vdash t[x_1, \ldots, t_n]
\end{array}
\]
\[
\text{list_simple}_\forall\text{elim}
\]
\[
\Gamma \vdash t^\prime_1, \ldots, t^\prime_{n^\prime}
\]

A built-in inference rule. The instantiation is done simultaneously, rather than by iteration of a single instantiation, which may affect renaming.

See Also $\forall$ elim

Errors
3012 ?0 and ?1 do not have the same types
6018 ?0 is not of the form $\Gamma \vdash \forall \ldots x_i \ldots \bullet t$ where the $\langle x_i \rangle$ are ?1 variables

val list_simple_\forall_intro : TERM list -> THM -> THM;

Description Generalised simple $\forall$ introduction.

\[
\begin{array}{c}
\Gamma \vdash t[x_1, \ldots, x_n] \\
\hline
\Gamma \vdash \forall x_1 \ldots x_n \bullet t[x_1, \ldots, x_n]
\end{array}
\]
\[
\text{list_simple}_\forall\text{intro}
\]
\[
\Gamma \vdash t^\prime_1, \ldots, t^\prime_{n^\prime}
\]

See Also $\forall$ intro

Errors Same messages as simple$\forall$ intro.

val list_simple_\exists_intro : TERM list -> TERM -> THM -> THM ;

Description Introduce an iterated existential quantifier by providing a list of witnesses and a theorem asserting that the desired property holds of these witnesses.

\[
\begin{array}{c}
\Gamma \vdash t[ t_1, t_2, \ldots] \\
\hline
\Gamma \vdash \exists x_1 x_2 \ldots \bullet t[x_1, x_2, \ldots]
\end{array}
\]
\[
\text{list_simple}_\exists\text{intro}
\]
\[
\Gamma \vdash t^\prime_1, t^\prime_2, \ldots
\]
\[
\Gamma \vdash \exists x_1 x_2 \ldots \bullet t[x_1, x_2, \ldots]
\]

Errors
7047 ?0 cannot be matched to conclusion of theorem ?1

val list_\&_intro : THM list -> THM;

Description Conjoin a list of theorems.

\[
\begin{array}{c}
\Gamma_1 \vdash t_1, \ldots, \Gamma_n \vdash t_n \\
\hline
\Gamma_1 \lor \ldots \Gamma_n \vdash t_1 \lor \ldots t_n
\end{array}
\]
\[
\text{list}_\&\text{intro}
\]

Errors
7107 List may not be empty
### 7.1. General Inference Rules

#### SML `list_\forall_elim`

**Description**  Generalised `\forall` elimination. Specialise a universally quantified theorem with given values, instantiating the types of the theorem as necessary.

**Rule**

\[
\frac{\Gamma \vdash \forall x_1 \ldots x_n \cdot t[x_1, \ldots, x_n]}{\Gamma \vdash \forall x_1 \ldots x_n \cdot t'[I'_1, \ldots, I'_n]} \quad \text{list}_\forall_{\text{elim}}
\]

where `t'` is renamed from `t` to prevent bound variable capture and type instantiated as necessary, the `x_i` are varstructs, instantiable to the structures of `t_i`. The values will be expanded using `Fst` and `Snd` as necessary to match the structure of `⌜x⌝`.

Note that due to the type instantiation this function is somewhat more that a `fold` of `\forall_{\text{elim}}`.

**See Also**  `\forall_{\text{elim}}, all_\forall_{\text{elim}}`.

**Errors**

- `27014`  
  `?0` is not of the form: `\Gamma \vdash \forall vs_1 \ldots vs_i \cdot t'` where `i \geq ?1`
- `27015`  
  `?0` is not of the form: `\Gamma \vdash \forall vs_1 \ldots vs_i \cdot t'` where the types of the `vs_i` are instantiable to the types of `?1`
- `27016`  
  `?0` is not of the form: `\Gamma \vdash \forall vs_1 \ldots vs_i \cdot t'` where the types of the `vs_i` are instantiable to the types of `?1` without instantiating type variables in the assumptions

#### SML `list_\forall_intro`

**Description**  Generalised `\forall` introduction.

**Rule**

\[
\frac{\Gamma \vdash t[x_1, \ldots, x_n]}{\Gamma \vdash \forall x_1 \ldots x_n \cdot t[x_1, \ldots, x_n]} \quad \text{list}_\forall_{\text{intro}}
\]

**See Also**  `\forall_{\text{intro}}, all_\forall_{\text{intro}}`.

**Errors**  Same messages as `\forall_{\text{intro}}`.

#### SML `MAP_C`

**Description**  This traverses a term from its leaves to its root node. It will repeat the application of its conversion argument, until failure, on each subterm encountered en route. At each node the conversion is applied to the sub-term that results from the application of the preceding traversal, not the original. It traverses from left to right, though this should only matter for conversions that work by side-effect. It fails if the conversion applies nowhere within the tree.

**Errors**

- `7005` Conversion fails on term and all its subterms
val mk_app_rule : THM -> THM -> THM;

Description  Given two equational theorems, one being between two functions, apply the two functions to the LHS and RHS of the other equation.

\[
\Gamma_1 \vdash u_1 = u_2; \Gamma_2 \vdash v_1 = v_2 \\
\Gamma_1 \cup \Gamma_2 \vdash u_1 v_1 = u_2 v_2
\]

mk_app_rule

The second input theorem or the result may be expressed using \(\Leftrightarrow\).

A built-in inference rule.

Errors

6020  ?0 is not of the form: 'Γ \vdash t_1 \Rightarrow t_2'
6023  ?0 and ?1 are not of the form: 'Γ_1 \vdash u_1 = u_2' and 'Γ_2 \vdash v_1 = v_2'

where 'u_1' can be functionally applied to 'v_1'

val modus_tollens_rule : THM -> THM -> THM;

Description  If the consequent of an implicative theorem is false, then so must be the antecedent (modus tollens).

\[
\Gamma_1 \vdash t_1 \Rightarrow t_2; \Gamma_2 \vdash \neg t_2' \\
\Gamma_1 \cup \Gamma_2 \vdash \neg t_1
\]

modus_tollens_rule

where \(t_2\) and \(t_2'\) are \(\alpha\)-convertible.

Errors

7040  ?0 is not of the form: 'Γ \vdash t_1 \Rightarrow t_2'
7051  ?0 and ?1 are not of the form: 'Γ_1 \vdash t_1 \Rightarrow t_2' and 'Γ_2 \vdash \neg t_2a'

where 't_2' and 't_2a' are \(\alpha\)-convertible

val ONCE_MAP_C : CONV -> CONV;

Description  This traverses a term from the root node to its leaves, attempting to apply its conversion argument. If it successfully applies the conversion to any subterm then it will not further traverse that subterm, but will still continue on other branches. If it fails to apply its conversion to a leaf, its functionality is equivalent to then applying refl_conv. It traverses from left to right, though this should only matter for conversions that work by side-effect. It will fail if the conversion succeeds nowhere in the tree, or if the results of certain conversion applications are ill-formed.

Errors

7005  Conversion fails on term and all its subterms
7.1. General Inference Rules

SML

```sml
val ONCE_MAP_WARN_C : string -> CONV -> CONV;
```

**Description** This is an equivalent to `ONCE_MAP_C` (q.v.) except that it warns the user if it failed to recompose the theorems from the term it just traversed.

This traverses a term from the root node to its leaves, attempting to apply its conversion argument. If it successfully applies the conversion to any subterm then it will not further traverse that subterm, but will still continue on other branches. If it fails to apply its conversion to a leaf, its functionality is equivalent to then applying `refl_conv`. It traverses from left to right, though this should only matter for conversions that work by side-effect. It will fail if the conversion succeeds nowhere in the tree, or if the results of certain conversion applications are ill-formed.

**Errors**

- 26001 no rewriting occurred
- 26003 no successful rewriting occurred, rewriting gave ill-formed results on some subterms

It issues the following warning message if at any point it fails to recompose the theorems from the subterm it just traversed, some successful rewriting occurs, and the flag “illformed_rewrite_warning” is true.

**Errors**

- 26002 rewriting gave ill-formed results on some subterms

Errors and warnings are from the area indicated by the string argument.

SML

```sml
val ORELSE_CAN : (CANON * CANON) -> CANON;
```

**Description** `ORELSE_CAN` is a canonicalisation function combinator written as an infix operator. `(can1 ORELSE_CAN can2)thm` is the same `can1 thm` unless evaluation of `can1 thm` fails in which case it is the same as `can2 thm`.

**See Also** `CANON`

SML

```sml
val ORELSE_C : (CONV * CONV) -> CONV;
```

**Description** Attempt to apply one conversion, and if that fails, try the second one.

**Rule**

\[ \Gamma \vdash t = t' \]

\[ (c1 : CONV) \text{ ORELSE } C (c2 : CONV) \]

\[ \Gamma \vdash t \]

where `c1 t` returns `Γ ⊢ t = t'`, or `c1` fails, and `c2 t` returns `Γ ⊢ t = t'`.

**See Also** `FIRST_C` (the iterated version of this function), `THEN_C`, `AND_OR_C`, and `THEN_TRY_C`

**Errors** As the failure of second conversion, should both conversions fail.
val plus_conv : CONV;

**Description** Provides the value of the addition of two numeric literals.

\[ \vdash \text{ML} \text{mk_N} m + \text{ML} \text{mk_N} n = \text{ML} \text{mk_N} (m + n) \]

**Uses** For doing fast arithmetic proofs.

**Errors** 6085 ?0 is not of the form: \[ \text{ML} \text{mk_N} m + \text{ML} \text{mk_N} n \]
7.1. General Inference Rules

SML

```ml
val prim_rewrite_conv : CONV NET \rightarrow CANON \rightarrow (THM \rightarrow TERM \times CONV) OPT \rightarrow (CONV \rightarrow CONV) \rightarrow EQN_CXT \rightarrow THM list \rightarrow CONV;
```

**Description** The primitive rewrite conversion.

```plaintext
\[
\Gamma \vdash t = t'
\]
```

where \(\Gamma t = t'\) is \(t\) rewritten according to the parameters of the conversion, and \(\Gamma\) are the assumptions required to allow the rewriting. The failure of the conversion constructed by `prim_rewrite_conv` will not be caught by `prim_rewrite_conv`.

The arguments have the following effects:

- **initial_net** This is a pre-calculated conversion net, that will serve as the initial rewriting that may be done.

- **canon** This canonicalisation function will be applied to all of the `with_thms` theorems, to produce a list of theorems to be rewritten with from these inputs. This will generally involve producing canonical or simplified forms of the original theorems.

  The resulting theorems are intended to be simply universally quantified equations, and theorems which are not of this form are discarded. Rewriting attempts to instantiate some or all of the universally quantified variables, or any type variables (which do not appear in the assumptions), so as to to match the left-hand side of an equation to the term being rewritten. N.b. free variables are not instantiated. An equation whose left-hand side matches the term being rewritten in such a way that rewriting would not change the term is treated as if it did not match the term.

- **eqm_rule** This equation matcher is mapped over the theorems resulting from the canonicalisation to convert them into an equation context. `thm_eqn_cxt` is used if `Nil` is supplied.

- **traverse** This is a conversional, which defines the traversal of term \(t\) by the rewriting conversion derived from `prim_rewrite_conv`'s other arguments.

- **with_eqn_cxt** This is additional equational context to be added directly into the rewriting conversion net.

- **with_thms** This is an additional set of theorems to be processed by `canon` and the results used in added directly into the rewriting conversion net.

**Uses** This is the basis of the primary rewriting tools, by varying the first four parameters.

`prim_rewrite_conv` preprocesses its arguments in various ways. The preprocessing for an argument takes place as soon as that argument is supplied, so, for example, the overhead of preprocessing `with_eqn_cxt` need not be incurred in calls with the same `with_eqn_cxt` but different `with_thms`. 
val prim_rewrite_rule : CONV NET -> CANON -> (THM -> TERM * CONV) OPT ->
(CONV -> CONV) -> EQN_CXT -> THM list -> THM -> THM;

Description  This is the inference rule based on prim_rewrite_conv (q.v.), with the same parameters as that function, except for the last argument:

\[ \Gamma \vdash t \]
\[ \Gamma \cup \Gamma' \vdash t' \]

where \( t' \) is the result of rewriting \( t \) in the manner prescribed by the arguments, and \( \Gamma' \) are the assumptions required to allow this rewriting.

val prim_suc_conv : CONV;

Description  This conversion gives the definition schema for all natural number literals.

\[ \vdash \text{mk}_{\mathbb{N}}(m+1) = \text{Suc}_{\mathbb{N}} \text{mk}_{\mathbb{N}} m \]

\[ \vdash \text{mk}_{\mathbb{N}} 0 = \text{Zero} \]

Errors

3026  ?0 is not a numeric literal

See Also  mk_{\mathbb{N}}, suc_conv

val prove_asm_rule : THM -> THM -> THM;

Description  Eliminate an assumption with reference to a the assumption being a conclusion of a theorem.

\[ \Gamma 1 \vdash t 1; \Gamma 2, t 1 \vdash t 2 \]
\[ \Gamma 1 \cup \Gamma 2 \vdash t 2 \]

This will in fact work even if the assumption is not present.
7.1. General Inference Rules

**SML**

```sml
val RANDS_C : CONV -> CONV;
```

**Description**  Apply a conversion to each of the arguments of a function

```
\vdash \Gamma \vdash f \ a \ldots \ z = f \ a' \ldots z' \\
RANDS_C \ (c : \text{CONV}) \ \Gamma f \ a \ldots z'\Gamma
```

where \( c \ a \) gives \( \vdash a = a' \), etc. The function \( f \) may have no arguments in which case \( \text{refl\_conv} \ f \) is returned.

**Errors**

```
7104 Result of conversion, \texttt{?0, ill-formed}
```

Also as the failure of the conversion.

**SML**

```sml
val RAND_C : CONV -> CONV;
```

**Description**  Apply a conversion to the operand of a combination:

```
\vdash \Gamma \vdash f \ a = f \ a' \\
RAND_C \ (c : \text{CONV}) \ \Gamma f \ a'\Gamma
```

where \( c \ a \) gives \( \vdash a = a' \).

**Errors**

```
3010 \ ?0 is not of form: \texttt{\Gamma t1 t2}\Gamma
7104 Result of conversion, \texttt{?0, ill-formed}
```

Also as the failure of the conversion.

**SML**

```sml
val RATOR_C : CONV -> CONV;
```

**Description**  Apply a conversion to the operator of a combination:

```
\vdash \Gamma \vdash f \ a = f' \ a \\
RATOR_C \ (c : \text{CONV}) \ \Gamma f' \ a'\Gamma
```

where \( c \ f \) gives \( \vdash f = f' \).

**Errors**

```
3010 \ ?0 is not of form: \texttt{\Gamma t1 t2}\Gamma
7104 Result of conversion, \texttt{?0, ill-formed}
```

Also as the failure of the conversion.

**SML**

```sml
val refl\_conv : CONV;
```

**Description**  The reflexivity of equality implemented as a conversion.

```
\vdash t = t \\
refl\_conv \ \Gamma t\Gamma
```

A primitive inference rule.
Chapter 7. PROOF IN HOL

### REPEAT

**Description** Repeatedly apply a conversion to a term, failing if not successfully applied at least once. To be more precise, the functionality is equivalent that of the following definition:

\[
\text{fun REPEAT\_C1 (c:CONV) = (c \text{ THEN\_TRY\_C REPEAT\_C1 c)}
\]

**Errors** As the error of the conversion if it cannot be applied at least once.

### REPEAT\_CAN

**Description** `REPEAT\_CAN` is a canonicalisation function combinator which repeatedly applies its argument until it fails:

\[
\text{fun REPEAT\_CAN can thm} = ((\text{can THEN\_CAN REPEAT\_CAN can}) \text{ ORELSE\_CAN id\_can}) \text{ thm}
\]

### REPEAT\_C

**Description** Repeatedly apply a conversion to a term. To be more precise, the functionality is equivalent that of the following definition:

\[
\text{fun REPEAT\_C (c:CONV) =}
\]

\[
\text{c \text{ THEN\_C (REPEAT\_C c)) ORELSE\_C refl\_conv}
\]

### REPEAT\_MAP\_C

**Description** This traverses a term from its leaves to its root node. It will attempt the application of its conversion argument on each subterm encountered en route. If the conversion is successfully applied to a given sub-term, then the resulting sub-term from the conversion is re-traversed by the function. It traverses from left to right, though this should only matter for conversions that work by side-effect. It fails if the conversion is not applicable anywhere within the term, or if certain applications of the conversion have ill-formed results.

**Errors**

7005 Conversion fails on term and all its subterms.

### REWRITE\_CAN

**Description** For rewriting, after all other canonicalisation we will usually wish to then universally quantify the resulting theorems in all free variables that are only in in the conclusion, other than those that were free anywhere in the original theorem, before any canonicalisation. A canonicalisation is transformed to work this way by `REWRITE\_CAN`.

When evaluating proof contexts (see, e.g., `commit\_pc`) the list of rewrite canonicalisations in the argument (see `get\_rw\_canons`), `arg`, will be converted to a single canonicalisation in the result by:

\[
\text{REWRITE\_CAN}
\]

\[
(\text{REPEAT\_CAN( FIRST\_CAN (arg @ [\\_t\_rewrite\_canon])})));
\]
7.1. General Inference Rules

SML

```sml
val rewrite_conv : THM list -> CONV;
val pure_rewrite_conv : THM list -> CONV;
val once_rewrite_conv : THM list -> CONV;
val pure_once_rewrite_conv : THM list -> CONV;
```

**Description** These are the standard rewriting conversions. They use the canonicalisation rule held by the proof context (see, e.g., `push_pc`) preprocess the theorem list. The context is accessed at the point when the rules are given a list of theorems.

If a conversion is “pure” then there is no default rewriting, otherwise the default rewriting conversion net held by the proof context will be used in addition to user supplied material.

If a conversion is “once” then rewriting will proceed from the root of the of the conclusion of the theorem to be rewritten, towards the leaves, and will not descend through any rewritten subterm, using `ONCE_MAP_WARN_C`. If not, rewriting will continue, moving from the root to the leaves, repeating if any rewriting is successful, until there is no rewriting redex anywhere within the rewritten conclusion, using `REWRITE_MAP_C`. This may cause non-terminating looping.

**Errors**

| 26001 | no rewriting occurred |

Also as error 26003 and warning 26002 of `REWRITE_MAP_C` (q.v.).

SML

```sml
val REWRITE_MAP_C : string -> CONV -> CONV;
```

**Description** This conversional is an equivalent to `TOP_MAP_C` (q.v.) except that it warns the user if it failed to recompose the theorems from the term it just traversed.

`REWRITE_MAP_C conv tm` traverses `tm` from its root node to its leaves. It will repeat the application of `conv` until failure, on each subterm encountered en route. It then descends through the sub-term that results from the repeated application of `conv`. If the descent causes any change, on “coming back out” to the sub-term the conversional will attempt to reapply `conv`, and if successful will then (recursively) reapply `REWRITE_MAP_C conv` once more. If `conv` cannot be reapplied then the conversional continues to ascend back to the root.

It traverses from left to right, though this should only matter for conversions that work by side-effect. It fails if the conversion is applied nowhere within the term.

**Errors**

| 26001 | no rewriting occurred |
| 26003 | no successful rewriting occurred, rewriting gave ill-formed results on some subterms |

It issues the following warning message if at any point it fails to recompose the theorems from the subterm it just traversed, some successful rewriting occurs, and the flag “illformed_rewrite_warning” is true.

**Errors**

| 26002 | rewriting gave ill-formed results on some subterms |

Errors and warnings are from the area indicated by the string argument.
Description  These are the standard rewriting rules. They use the canonicalisation rule held
by the proof context (see, e.g. push_pc) to preprocess the theorem list. The context is accessed
at the point when the rules are given a list of theorems.

If a rule is “pure” then there is no default rewriting, otherwise the default rewriting conversion
net held by the proof context will be used in addition to user supplied material.

If a rule is “once” then rewriting will proceed from the root of the of the conclusion of the theorem
to be rewritten, towards the leaves, and will not descend through any rewritten subterm, using
ONCE_MAP_WARN. If not, rewriting will continue, moving from the root to the leaves,
repeating if any rewriting is successful, until there is no rewriting redex anywhere within the
rewritten conclusion, using REWRITE_MAP_C. This may cause non-terminating looping.

If a rule is “asm” then the theorems rewritten with will include the canonicalised asm_ruled
assumptions of the theorem being rewritten.

See Also  prim_rewrite_rule

Errors

26001 no rewriting occurred
Also as error 26003 and warning 26002 of REWRITE_MAP_C (q.v.).
7.1. General Inference Rules

SML
\[
\text{val SIMPLE\_BINDER\_C : CONV \rightarrow CONV;}
\]

**Description** Apply a conversion to the body of a simple binder term:

\[
\begin{align*}
\Gamma \vdash (B \ x \bullet \ p[x]) &= (B \ x \bullet \ p'[x]) \\
(\forall B \ x \bullet \ p) &= (\forall B \ x \bullet \ p')
\end{align*}
\]
where \(c \ p[x]\) gives \(\vdash p[x] = p'[x]\), and \(B\) is a binder.

**Errors**
- 7059: ?0 is not of the form: \(\Gamma \ B \ x \bullet \ p[x]\) where \(\Gamma \ B\) is a binder and \(\Gamma \ x\) a variable
- 7104: Result of conversion, ?0, ill-formed

Also as the failure of the conversion.

SML
\[
\text{val simple\_eq\_match\_conv : THM \rightarrow CONV ;}
\]

**Description** This matches the LHS of an equational theorem to a term, instantiating the RHS accordingly. In fact the equation may be partially or fully universally quantified (simple quantification only), without affecting the result of the conversion.

\[
\begin{align*}
\Gamma' \vdash t &= v' \\
(\forall \ldots \bullet \ u = v)
\end{align*}
\]
where \(v'\) is the result of applying to \(v\) the instantiation rules required to match \(u\) to \(t\) (including both term and type instantiation). If there are free variables on the RHS of the supplied equational theorem (when stripped of all universal quantification) they will be renamed as necessary to avoid identification with any variables in \(t\).

**Errors**
- 7044: Cannot match ?0 and ?1
**VAL** simple_eq_match_conv1 : THM -> CONV ;

**Description**  This matches the LHS of an universally quantified (simple quantifiers only) equational theorem to a term, instantiating the RHS accordingly. The conversion will only instantiate its universal quantifications, and type variables not present in the assumptions, and not its free term variables.

**Conversion**

\[ \Gamma \vdash t = v[t_1, ..., t_n] \]

where \((u[t_1, ..., t_n])\) is \(\alpha\)-convertible to \(\tau\). If there are free variables on the RHS of the supplied equational theorem (when stripped of all universal quantification) they will be renamed as necessary to avoid identification with any variables in \(t\).

This conversion may be partially evaluated with only its theorem argument.

**Uses** In producing a limited rewriting facility, that only instantiates explicitly identified variables.

**Errors**

7095  ?0 is not of the form ‘\(\Gamma \vdash \forall x_1 \ldots x_n \bullet u = v\)’ where \(\tau_x\) are variables

7076  Could not match term ?0 to LHS of theorem ?1

**VAL** simple_ho_eq_match_conv : THM -> CONV

**Description**  This conversion is like simple_eq_match_conv but uses higher-order matching. It uses ho_match (q.v.) to match the LHS of an equational theorem to a term \(t\). It then instantiates the theorem (including both term and type instantiation) and carries out any \(\beta\eta\)-reductions required to give a theorem of the form \(t = v'\). The equation may be partially or fully universally quantified (simple quantification only, not quantification over pairs).

**Conversion**

\[ \Gamma' \vdash t = v' \]

where \(v'\) is the result of applying to \(v\) the instantiations required to match \(u\) to \(t\) (including term and type instantiation). If there are free variables on the RHS of the supplied equational theorem (when stripped of all universal quantification) they will be renamed as necessary to avoid identification with any variables in \(t\).

**Errors**

7095  ?0 is not of the form ‘\(\Gamma \vdash \forall x_1 \ldots x_n \bullet u = v\)’ where \(\tau_x\) are variables

7076  Could not match term ?0 to LHS of theorem ?1
### 7.1. General Inference Rules

**SML**

| val simple\_ho\_eq\_match\_conv1 : THM → CONV |

**Description**  This conversion is like `simple\_eq\_match\_conv1` but uses higher-order matching. It uses `ho\_match` (q.v.) to match the LHS of an equational theorem to a term \( t \). The equation may be partially or fully universally quantified (simple quantification only, not quantification over pairs). It instantiates the theorem (including both term and type instantiation) and carries out any \( \beta\eta \)-reductions required to give a theorem of the form \( t = v' \). Only type variables that do not appear in the assumptions of the theorem and universally quantified term variables will be instantiated.

**Conversion**

\[
\Gamma \vdash t = v' \\
\Gamma \vdash_1 \bigl( (\forall \ldots \bullet u = v) \bigl)
\]

where \( v' \) is the result of applying to \( v \) the instantiation rules required to match \( u \) to \( t \) (including term and type instantiation). If there are free variables on the RHS of the supplied equational theorem (when stripped of all universal quantification) they will be renamed as necessary to avoid identification with any variables in \( t \).

### Errors

- 7095   \(?0\) is not of the form \( \forall x_1 \ldots x_n \bullet u = v^t \) where \( \Gamma x_i \) are variables
- 7076   Could not match term \(?0\) to LHS of theorem \(?1\)

**SML**

| val simple\_\leftrightarrow\_match\_mp\_rule : THM → THM → THM; |

**Description**  A matching Modus Ponens for \( \leftrightarrow \).

**Rule**

\[
\Gamma_1 \vdash_1 \forall x_1 \ldots \bullet t_1 \leftrightarrow t_2; \quad \Gamma_2 \vdash t_1^t \quad \Gamma_1' \cup \Gamma_2 \vdash t_2' \\
\]  \[\text{simple\_\leftrightarrow\_match\_mp\_rule}\]

where \( t_1' \) is an instance of \( t_1 \) under type instantiation and substitution for the \( x_i \) and the free variables of the first theorem, and where \( t_2' \) is the corresponding instance of \( t_2 \). No type instantiation or substitution will occur in the assumptions of either theorem.

**See Also**  \( \Rightarrow_{\text{elim}} \) (Modus Ponens on \( \Rightarrow \)), `simple\_\leftrightarrow\_match\_mp\_rule`

### Errors

- 7044   Cannot match \(?0\) and \(?1\)
- 7046   \(?0\) is not of the form \( \forall x_1 \ldots x_n \bullet u \leftrightarrow v^t \)

**SML**

| val simple\_\leftrightarrow\_match\_mp\_rule1 : THM → THM → THM; |

**Description**  A matching Modus Ponens for \( \leftrightarrow \) that doesn’t affect assumption lists.

**Rule**

\[
\Gamma_1 \vdash_1 \forall x_1 \ldots \bullet t_1 \leftrightarrow t_2; \quad \Gamma_2 \vdash t_1^t \quad \Gamma_1' \cup \Gamma_2 \vdash t_2' \\
\]  \[\text{simple\_\leftrightarrow\_match\_mp\_rule1}\]

where \( t_1' \) is an instance of \( t_1 \) under type instantiation and substitution for the \( x_i \) (but not free variables), and where \( t_2' \) is the corresponding instance of \( t_2 \). Types in the assumptions of the theorems will not be instantiated.

**See Also**  \( \Rightarrow_{\text{elim}} \) (Modus Ponens on \( \Rightarrow \)), `simple\_\leftrightarrow\_match\_mp\_rule1`

### Errors

- 7044   Cannot match \(?0\) and \(?1\)
- 7046   \(?0\) is not of the form \( \forall x_1 \ldots x_n \bullet u \leftrightarrow v^t \)

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
\[
\text{\texttt{val simple.$\Rightarrow$\_match\_mp\_rule}} : \text{THM} \rightarrow \text{THM} \rightarrow \text{THM} ;
\]

\textbf{Description}  A matching Modus Ponens rule for an implicative theorem.

\begin{align*}
\Gamma_1 \vdash \forall x_1 \ldots \bullet t_1 \Rightarrow t_2; \quad \Gamma_2 \vdash t_1' & \quad \text{simple.$\Rightarrow$\_match\_mp\_rule} \\
\Gamma_1 \cup \Gamma_2 \vdash t_2' & \quad \text{simple.$\Rightarrow$\_match\_mp\_rule}
\end{align*}

where \(t_1'\) is an instance of \(t_1\) under type instantiation and substitution for the \(x_i\) and the free variables of the first theorem, and where \(t_2'\) is the corresponding instance of \(t_2\). No type instantiation or substitution will occur in the assumptions of either theorem.

\textbf{See Also}  \texttt{simple.$\Rightarrow$\_match\_mp\_rule1}, \texttt{simple.$\Rightarrow$\_match\_mp\_rule2}

\textbf{Errors}  7044 Cannot match ?0 and ?1 \\
7045 ?0 is not of the form \(\forall x_1 \ldots x_n \bullet u \Rightarrow \nu\)

\[\text{\texttt{val simple.$\Rightarrow$\_match\_mp\_rule1}} : \text{THM} \rightarrow \text{THM} \rightarrow \text{THM} ;\]
\[\text{\texttt{val simple.$\Rightarrow$\_match\_mp\_rule2}} : \text{THM} \rightarrow \text{THM} \rightarrow \text{THM} ;\]

\textbf{Description}  Two variants on a matching Modus Ponens rule for an implicative theorem.

\begin{align*}
\Gamma_1 \vdash \forall x_1 \ldots \bullet t_1 \Rightarrow t_2; \quad \Gamma_2 \vdash t_1' & \quad \text{simple.$\Rightarrow$\_match\_mp\_rule1} \\
\Gamma_1 \cup \Gamma_2 \vdash t_2' & \quad \text{simple.$\Rightarrow$\_match\_mp\_rule2}
\end{align*}

where \(t_1'\) is an instance of \(t_1\) under type instantiation and substitution for the \(x_i\) (but not free variables), and where \(t_2'\) is the corresponding instance of \(t_2\).

\texttt{simple.$\Rightarrow$\_match\_mp\_rule2} is just like \texttt{simple.$\Rightarrow$\_match\_mp\_rule1} except that the instantiations and substitutions returned by \texttt{term\_match} are extended to replace type variables that do not occur in \(t_1\) or in \(\Gamma_1\) and \(x_i\) that do not occur free in \(t_1\) by fresh variables to avoid clashes with each other and with the type variables and free variables of \(\Gamma_1\) and \(\Gamma_2\).

Types in the assumptions of the theorems will not be instantiated.

\textbf{See Also}  \texttt{simple.$\Rightarrow$\_match\_mp\_rule}

\textbf{Errors}  7044 Cannot match ?0 and ?1 \\
7045 ?0 is not of the form \(\forall x_1 \ldots x_n \bullet u \Rightarrow \nu\)

\[\text{\texttt{val simple.$\forall$\_elim}} : \text{TERM} \rightarrow \text{THM} \rightarrow \text{THM} ;\]

\textbf{Description}  Instantiate a universally quantified variable to a given value.

\begin{align*}
\Gamma \vdash \forall x \bullet t_2[x] & \quad \text{simple.$\forall$\_elim} \\
\Gamma \vdash t_2'[t_1] & \quad \text{simple.$\forall$\_elim}
\end{align*}

where t_2' is renamed from t_2 to prevent bound variable capture, and x is a variable.

\textbf{Errors}  3012 ?0 and ?1 do not have the same types \\
7039 ?0 is not of the form \(\forall x \bullet \nu\) where \(\nu\) is a variable
7.1. General Inference Rules

SML

val simple ∀ intro : TERM \(\rightarrow\) THM \(\rightarrow\) THM;

**Description**  Introduce a simple universally quantified theorem.

### Rule

\[
\begin{array}{c}
\Gamma \vdash t \\
\hline
\Gamma \vdash \forall x \cdot t
\end{array}
\]

simple ∀ intro \(\forall x\)^\(\exists\)

A built-in inference rule.

**See Also**  ∀ intro

**Errors**  
- 3007  \(?0\) is not a term variable
- 6005  \(?0\) occurs free in assumption list

SML

val simple ∀ ∃ conv : CONV;

**Description**  Swap the order of a simple ∀ and ∃:

### Conversion

\[
\vdash (\forall x \cdot \exists y \cdot P[x, y]) \Leftrightarrow (\exists y' \cdot \forall x \cdot P[x, y'])
\]

simple ∀ ∃ conv \(\forall x\)^\(\exists\) \(\exists\)

where \(y'\) is renamed to distinguish it from \(y\) (for the types differ) and every other term variable in the argument.

**Errors**  
- 27031  \(?0\) is not of the form: \(\forall x \cdot \exists y \cdot P[x, y]\)

SML

val simple ∃ elim : TERM \(\rightarrow\) THM \(\rightarrow\) THM \(\rightarrow\) THM;

**Description**  Eliminate an existential quantifier.

### Rule

\[
\begin{array}{c}
\Gamma 1 \vdash \exists x \cdot t1[x]; \Gamma 2, t1[y] \vdash t2 \\
\hline
\Gamma 1 \cup \Gamma 2 \vdash t2
\end{array}
\]

simple ∃ elim \(\exists\)^\(\forall\)

where \(y\) must be variable which is not present elsewhere in the second theorem, nor in the conclusion of the first. \(t1[y]\) need not actually be present in the assumptions of the second theorem.

**Errors**  
- 3007  \(?0\) is not a term variable
- 7014  \(?0\) has the wrong type
- 7109  \(?0\) is not of the form \(\forall x \cdot t[x]\)
- 7120  \(?0\) occurs free in conclusion of \(?1\)
- 7121  \(?0\) occurs free in hypotheses of \(?1\) other than \(?2\)
### Simple Existential Introduction (simple\_∃\_intro)

**Description**
Introduce an existential quantifier by reference to a witness.

**Rule**
\[
\begin{align*}
\Gamma & \vdash t_1[t_2] \\
\Gamma & \vdash \exists \ x \bullet t_1[x] \\
\end{align*}
\]

where \( \exists x \) is a variable.

**Errors**
- 3034: ?0 is not of form: \( \exists \ \text{var} \bullet \text{body} \)
- 7047: ?0 cannot be matched to conclusion of theorem ?1

### Simple Existential-All Conversion (simple\_∃\_∀\_conv)

**Description**
Swap the order of a simple \( \exists \) and \( ∀ \):

**Conversion**
\[
\begin{align*}
\vdash (\exists \ x \bullet \forall \ y \bullet \ P[x,y]) & \iff \\
(\forall \ y' \bullet \exists \ x \bullet \ P[x, y'])
\end{align*}
\]

where \( y' \) is renamed to distinguish it from \( y \) (for the types differ) and every other term variable in the argument.

**Errors**
- 27032: ?0 is not of the form: \( \exists \ \text{f} \bullet \forall \ 

### Simple Existential-All Conversion 1 (simple\_∃\_∀\_conv1)

**Description**
Swap the order of a simple \( \exists \) and \( ∀ \), where the first variable is always applied to the second:

**Conversion**
\[
\begin{align*}
(\exists f' \bullet \forall \ x \bullet \ P[f', x]) & \iff \\
(\forall \ x \bullet \exists f \bullet \ P[f, x])
\end{align*}
\]

where \( f' \) is renamed to distinguish it from \( f \) (for the types differ) and every other term variable in the argument.

**Errors**
- 27033: ?0 is not of the form: \( \exists \ f \bullet \forall \ x \bullet \ P[f, x] \)

### Simple Existential-Epsilon Conversion (simple\_∃\_ε\_conv)

**Description**
Give that \( \epsilon \) of a predicate satisfies the predicate by reference to an \( \exists \) construct.

**Rule**
\[
\begin{align*}
\Gamma & \vdash (\exists x \bullet p[x]) \iff p[\epsilon \ x \bullet p \ x] \\
\end{align*}
\]

**See Also** \( \exists \_\epsilon \_\text{rule} \)

**Errors**
- 3034: ?0 is not of form: \( \exists \ \text{var} \bullet \text{body} \)
### 7.1. General Inference Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \frac{\Gamma \vdash \exists x \cdot p[x]}{\Gamma \vdash p[\epsilon \cdot x \cdot p[x]]} ]</td>
<td>Give that ( \epsilon ) of a predicate satisfies the predicate by reference to an ( \exists ) construct. It can properly handle paired existence.</td>
</tr>
<tr>
<td>[ \frac{\Gamma \vdash \exists_1 x \cdot P[x]}{\Gamma \vdash \exists x \cdot P[x] \land \forall y \cdot P[y] \Rightarrow y = x} ]</td>
<td>Express a ( \exists_1 ) in terms of ( \exists ) and a uniqueness property.</td>
</tr>
<tr>
<td>[ \frac{\Gamma_1 \vdash P'[t'] \quad \Gamma_2 \vdash \forall x \cdot P[x] \Rightarrow x = t}{\Gamma_1 \cup \Gamma_2 \vdash \exists_1 x \cdot P[x]} ]</td>
<td>Introduce ( \exists_1 ) by reference to a witness, and a uniqueness theorem.</td>
</tr>
<tr>
<td>[ \frac{\neg \exists \epsilon \cdot \text{pred}}{\text{pred}} ]</td>
<td>Simple ( \exists \epsilon ) rule</td>
</tr>
</tbody>
</table>

**See Also** \( \exists \epsilon \) conv

**Errors**
- 7092 \( \exists \) is not of the form: '\( \Gamma \vdash \exists x \cdot p[x] \)
- 7015 \( \exists \) is not of the form: '\( \Gamma \vdash \exists_1 x \cdot P[x] \)
- 7066 \( \exists \) is not of the form: '\( \Gamma \vdash \forall x \cdot P[x] \Rightarrow x = t \)
- 7067 \( \exists \) and ?1 are not of the form: '\( \Gamma \vdash P[a][t] \) and '\( \Gamma \vdash \forall x \cdot P[x] \Rightarrow x = t \) where \( \Gamma \), \( \forall \), \( \exists \), \( a \) and \( t \) are \( \alpha \)-convertible

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \frac{\Gamma \vdash \exists_1 x \cdot P[x]}{\Gamma \vdash \exists x \cdot P[x] \land \forall y \cdot P[y] \Rightarrow y = x} ]</td>
<td>Rename a bound variable name, as a conversion. This only works with simple abstractions.</td>
</tr>
<tr>
<td>[ \frac{\neg \lambda \text{var} \cdot t[\text{var}]}{\lambda \text{var} \cdot t[\text{var}]} ]</td>
<td>Simple ( \alpha ) conv</td>
</tr>
</tbody>
</table>

**Errors**
- 3011 \( \alpha \) is not of form: '\( \lambda \text{var} \cdot t \)
- 7035 Cannot rename bound variable ?0 to ?1 as this would cause variable capture

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
val simple_\(\beta\)_conv : CONV;

**Description**  Apply a \(\beta\)-reduction to a simple abstraction.

\[
\vdash (\lambda x \cdot t_1[x]) t_2 = t_1[t_2]\]

\(\text{simple}_\(\beta\)\_conv\)

\[
\Gamma (\lambda x \cdot t_1[x]) t_2 \\gamma
\]

A primitive inference rule.

**See Also**  \(\beta\)_conv

**Errors**

6012 ?0 is not of the form: \(\Gamma (\lambda x \cdot t_1[x]) t_2 \\gamma\) where \(\gamma\) is a variable

---

val simple_\(\beta\)\_\(\eta\)\_conv : TERM \(\rightarrow\) CONV;

**Description**  If \(t\) is any term, \(\text{simple}_\(\beta\)\_\(\eta\)\_conv\) \(t\) is a conversion which will prove all theorems of the form \(\vdash t = s\) where \(t\) and \(s\) are simply \(\alpha\beta\eta\)-equivalent, i.e., can be reduced to \(\alpha\)-equivalent normal forms by \(\beta\)- and \(\eta\)-reduction involving only simple (rather than paired) \(\lambda\)-abstractions.

**Errors**

7131 ?0 and ?1 are not simply \(\alpha\beta\eta\)-equivalent

---

val simple_\(\beta\)\_\(\eta\)\_norm_conv : CONV;

**Description**  This conversion eliminates all simple \(\beta\)- and \(\eta\)-redexes from a term giving the \(\beta\eta\) normal form. It does not eliminate \(\beta\)- and \(\eta\)-redexes involving abstraction over pairs. It fails if the term is already in normal form.

**Errors**

7130 ?0 contains no simple \(\beta\)- or \(\eta\)-redexes

---

val simple_\(\epsilon\)\_elim_rule : TERM \(\rightarrow\) THM \(\rightarrow\) THM \(\rightarrow\) THM;

**Description**  Given that \(\epsilon\) of a predicate satisfies that predicate, then in a different theorem we may eliminate an assumption that claims an otherwise unused variable satisfies the predicate.

\[
\Gamma_1 \vdash t' (\$\epsilon t''); \quad \Gamma_2, t x \vdash s
\]

\[
\Gamma_1 \cup \Gamma_2 \vdash s \quad \text{simple}_\(\epsilon\)\_elim\_rule
\]

where \(t\), \(t'\) and \(t''\) are \(\alpha\)-convertible, and \(x\) is a free variable whose only free occurrence in the second theorem is the one shown and which does not appear free in the conclusion of the first theorem. In fact, \((\$\epsilon t'')\) here can be any term, it is not constrained to be an application of the choice function.

**Errors**

3007 ?0 is not a term variable
7019 ?0 is not of the form: \(\Gamma \vdash t_1(\$\epsilon t)\)
7054 ?0 is not of same type as choice sub-term of first theorem
7108 Arguments not of the form \(\Gamma \vdash t (\$\epsilon t)\) and \(\Gamma 2, (t ?0) \vdash s\)
7120 ?0 occurs free in conclusion of ?1
7121 ?0 occurs free in hypotheses of ?1 other than ?2
7122 ?0 occurs free in operator of the conclusion of ?1
7.1. General Inference Rules

val SIMPLE_\lambda_C : CONV -> CONV;

**Description**  Apply a conversion to the body of a simple abstraction:

\[ \Gamma \vdash (\lambda x \bullet p[x]) = (\lambda x \bullet p'[x]) \]

\( SIMPLE_\lambda_C \)
\( (c : CONV) \)
\( \Gamma \lambda x \bullet p \)

where \( c \ p[x] \) gives \( \vdash p[x] = p'[x] \).

**See Also**  SIMPLE_BINDER_C

**Errors**

3011  ?0 is not of form: \( \Gamma \ \lambda \ \text{var} \bullet t \)

7104  Result of conversion, ?0, ill-formed

Also as the failure of the conversion.

val simple_\lambda_eq_rule : TERM -> THM -> THM;

**Description**  Given an equational theorem, return the equation formed by abstracting the term argument (which must be a variable) from both sides.

\[ \Gamma \vdash t1[x] = t2[x] \]

\[ \Gamma \vdash (\lambda x \bullet t1[x]) = (\lambda x \bullet t2[x]) \]

\( simple_\lambda_eq_rule \)
\( \Gamma \)
\( x \)

A primitive inference rule.

**See Also**  \_eq_rule

**Errors**

3007  ?0 is not a term variable

6005  ?0 occurs free in assumption list

6020  ?0 is not of the form: \( \Gamma \vdash t1 = t2 \)
\textbf{Description} This function defines the constants with names starting with ", and type \textit{CHAR LIST} (an abbreviation of \textit{CHAR LIST}). A string literal constant is indicated by the constant name starting with a double quote("), as well as being of type \textit{CHAR LIST}. This is equivalent to a list of character literal constants, one for each but the first (") character of the string constant’s name. This conversion defines this relationship, by returning the head and un-exploded tail of the list of characters. A character literal is indicated by the constant’s name starting with single backquote (‘), as well as being of type \textit{CHAR}.

\begin{align*}
\vdash \text{mk_string("c...")}\triangledown &= \text{Cons}_{\text{ML}}(\text{mk_char("c")}\triangledown) \\
&\quad \text{mk_string("...")}\triangledown
\end{align*}

Or:

\begin{align*}
\vdash \text{mk_string("")}\triangledown &= \text{Nil}
\end{align*}

A primitive inference rule(axiom schemata).

\textbf{See Also} \textit{mk_string}

\textbf{Errors} 3025 0 is not a string literal

\textbf{Errors} 3025 0 is not a string literal

\textbf{Description} Break a theorem into conjuncts as far as possible.

\begin{align*}
\Gamma \vdash t \\
\Gamma \vdash t_1, \ldots, \Gamma \vdash t_n
\end{align*}

\begin{align*}
\vdash \text{strip_\&_rule}(a \land b) \land (a \land c \land d) \\
\vdash a', \vdash b', \vdash a', \vdash c', \vdash d'
\end{align*}

\textbf{Description} Repeatedly apply undisch\_rule:

\begin{align*}
\Gamma \vdash t_1 \Rightarrow \ldots \Rightarrow t_n \Rightarrow t \\
\Gamma \cup \{t_1, \ldots, t_n\} \vdash t
\end{align*}
7.1. General Inference Rules

\[ \text{val subst_conv : } (\text{THM } \ast \text{ TERM}) \text{ list } \rightarrow \text{ TERM } \rightarrow \text{ CONV}; \]

**Description**  
Substitution of equational theorems according to a template.

\[
\Gamma_1 \cup ... \Gamma_n \vdash t[\ldots,t_i\ldots] = t'[\ldots,t'_i\ldots]
\]

\[
\text{subst_conv }\]
\[
[[\ldots(\Gamma_i \vdash t_i = t_i', \Gamma_i), \ldots]]
\]
\[
\Gamma_i \vdash t'[\ldots,x_i\ldots]\]
\[
\Gamma_i \vdash t[\ldots,t_i\ldots]
\]

\text{subst_conv } \[(\text{thm}_1, x_1), \ldots, (\text{thm}_n, x_n)]\] template term returns a theorem in which template determines where in term the \text{thm}_i are substituted, when forming the RHS of the equation. The \(x_i\) must be variables. The template is of the form \(t[x_1, \ldots, x_n]\), and wherever the \(x_i\) are free in template their associated equational theorem, \text{thm}_i, is substituted into \text{thm}. The rule will rename as necessary to avoid bound variable capture. The assumption list of the resulting theorem will be the union of all substitution theorems, regardless of use.

The RHS of the resulting theorem will take its bound variable names from template, not term, as shown in the following example. This provides an \(\alpha\)-conversion facility.

This function may be partially evaluated with only one argument.

**Example**

\[
\text{subst_conv } \[(\vdash p = q', \Gamma x1\gamma), (\vdash r = s', \Gamma x2\gamma)]
\]
\[
(\forall y \bullet f x1 r y + g x2 p = h y)\]
\[
(\forall x \bullet f p r x + g r p = h x)\]
\[
=\]
\[
(\forall x \bullet f r x + g r p = h x) \Leftrightarrow
\]
\[
\forall y \bullet f q r y + g s p = h y'
\]

**See Also**  
\text{subst_rule}

**Errors**

3007 ?0 is not a term variable
3012 ?0 and ?1 do not have the same types
6001 ?0 does not substitute to conclusion of theorem ?1
6002 Substitution theorem ?0 is not of the form: \(\Gamma \vdash t_1 = t_2\)
6029 Substitution list contains entry (?0,?1) where the type of the variable differs from the type of the LHS of the theorem
val subst_rule : (THM * TERM) list -> TERM -> THM -> THM;

**Description**  Substitution of equational theorems according to a template.

\[
\frac{\Gamma I \vdash t_1 = t_{1}', \ldots , \Gamma n \vdash t_n = t_{n}'}{\Gamma I \cup \ldots \Gamma n \cup \Gamma \vdash t[t_{1}', \ldots t_{n}'] \quad \text{subst_rule}}
\]

\( \text{subst_rule } [(\text{thm}_1, x_1), \ldots , (\text{thm}_n, x_n)] \) template \( \text{thm} \) returns a theorem in which \( \text{template} \) determines where in \( \text{thm} \) the \( \text{thm}_i \) are substituted. The \( x_i \) must be variables. The template is of the form \( t[x_1, \ldots , x_n] \), and wherever the \( x_i \) are free in \( \text{template} \) their associated equational theorem, \( \text{thm}_i \), is substituted into \( \text{thm} \). The rule will rename as necessary to avoid bound variable capture. The assumption list of the resulting theorem will be the union of all substitution theorems, regardless of use.

The conclusion of the resulting theorem will take its bound variable names from \( \text{template} \), not \( \text{thm} \), as shown in the following example. This provides an \( \alpha \)-conversion facility.

The function may be usefully partially evaluated with one or two arguments.

A primitive inference rule.

**Example**

\[
(\forall y \cdot f \ x_1 \ r \ y + g \ x_2 \ p = h \ y') \quad = \quad \forall x \cdot f \ p \ r \ x + g \ r \ p = h \ x'
\]

**See Also**  subst_conv

**Errors**

- 3007  ?0 is not a term variable
- 6001  ?0 does not substitute to conclusion of theorem ?1
- 6002  Substitution theorem ?0 is not of the form: ‘{\Gamma' \vdash t_1 = t_2'}
- 6029  Substitution list contains entry (?0,?1) where the type of the variable differs from the type of the LHS of the theorem
SML
| val SUB_C1 : CONV -> CONV;

**Description**  Apply a conversion to each of the constituents of a term, failing if the term cannot be broken up, or the conversion fails on all constituents (if only one of the two constituents of a `mk_app` have failures, then the offending term will be `refl_conved` instead). Thus:

```
| SUB_C1 cnv var = fail_cnv var
| SUB_C1 cnv const = fail_cnv const
| SUB_C1 cnv (f x) = Γ ⊢ f x = f' x'
  where cnv f = Γ1 ⊢ f = f'
  and cnv x = Γ2 ⊢ x = x'
  and Γ = Γ1 ∪ Γ2
| SUB_C1 cnv (λ x • t) = Γ ⊢ (λ x • t) = (λ x • t')
  where cnv t = Γ ⊢ t = t'
```

**Errors**
- **7104** Result of conversion, ?0, ill-formed
- **7105** ?0 has no constituents

There may be failure messages from the conversions.

---

SML
| val SUB_C : CONV -> CONV;

**Description**  Apply a conversion to each of the constituents of a term, however that term might be constructed, and recombine the results. Thus:

```
| SUB_C cnv var = refl_cnv var
| SUB_C cnv const = refl_cnv const
| SUB_C cnv (f x) = Γ ⊢ f x = f' x'
  where cnv f = Γ1 ⊢ f = f'
  and cnv x = Γ2 ⊢ x = x'
  and Γ = Γ1 ∪ Γ2
| SUB_C cnv (λ x • t) = Γ ⊢ (λ x • t) = (λ x • t')
  where cnv t = Γ ⊢ t = t'
```

**See Also**  SUB_C1
\[
\begin{align*}
\text{val} & \quad \text{suc\_conv} : \text{CONV}; \\
\text{Description} & \quad \text{This conversion gives the definition schema for non-zero natural number literals.}
\end{align*}
\]

\[
\begin{align*}
\Gamma & \vdash mk_\mathbb{N}(m+1) = Suc mk_\mathbb{N} m
\end{align*}
\]

The conversion fails if given 0.

\[
\begin{align*}
3026 & \quad ?0 \text{ is not a numeric literal} \\
7100 & \quad ?0 \text{ must be numeric literal} > 0
\end{align*}
\]

See Also \text{mk\_N}, \text{prim\_suc\_conv}

---

\[
\begin{align*}
\text{val} & \quad \text{THEN\_CAN} : (\text{CANON} * \text{CANON}) \rightarrow \text{CANON}
\end{align*}
\]

\[
\begin{align*}
\text{Description} & \quad \text{THEN\_CAN is a canonicalisation function combinator written as an infix operator. (can1 THEN\_CAN can2)thm is the result of applying can2 to each of the theorems in the list can1 thm and then flattening the resulting list of lists.}
\end{align*}
\]

See Also \text{CANON}

---

\[
\begin{align*}
\text{val} & \quad \text{THEN\_C} : (\text{CONV} * \text{CONV}) \rightarrow \text{CONV};
\end{align*}
\]

\[
\begin{align*}
\text{Description} & \quad \text{Combine the effect of two successful conversions.}
\end{align*}
\]

\[
\begin{align*}
\Gamma & \vdash t = t''
\end{align*}
\]

\[\Gamma \vdash t = t', \quad c2 t' \text{ returns } \Gamma 2 \vdash t'' = t''', \quad t' \text{ and } t'' \text{ are } \alpha\text{-convertible and } \Gamma \text{ equals } \Gamma 1 \cup \Gamma 2.\]

See Also \text{EVERY\_C (the iterated version of this function), as well as THEN\_TRY\_C, AND\_OR\_C, and ORELSE\_C}

\[
\begin{align*}
7101 & \quad \text{Result of first conversion, } ?0 , \text{ not an equational theorem} \\
7102 & \quad \text{LHS (if any) of result of second conversion, } ?0 , \text{ not } \alpha\text{-convertible to RHS of first, } ?1
\end{align*}
\]

Errors If any, as the failures of \text{c1} and \text{c2} applied to \text{t} and \text{t'} respectively.

---

\[
\begin{align*}
\text{val} & \quad \text{THEN\_LIST\_CAN} : (\text{CANON} * \text{CANON list}) \rightarrow \text{CANON}
\end{align*}
\]

\[
\begin{align*}
\text{Description} & \quad \text{THEN\_LIST\_CAN is a canonicalisation function combinator written as an infix operator. (can1 THEN\_LIST\_CAN cans)thm is the result of applying each element of the list cans to the corresponding element of the list can1 thm and then flattening the resulting list of lists.}
\end{align*}
\]

See Also \text{CANON}

\[
\begin{align*}
26204 & \quad \text{wrong number of canonicalisation functions in the list}
\end{align*}
\]
7.1. General Inference Rules

\begin{verbatim}
SML
| val THENTRY_C : (CONV * CONV) --> CONV;

Description  Combine the effect of two conversions, ignoring the failure of the second if necessary. That is, if the first conversion results in an equational theorem whose RHS can have the second conversion applied, and the two resulting theorems composed, then that composition; otherwise the result of the first conversion alone is returned.

See Also  THEN_C, AND_OR_C, ORElse_C

Errors  As the failure of \textit{c1}.
\end{verbatim}

\begin{verbatim}
SML
| val TOPMAP_C : CONV --> CONV;

Description  TOPMAP_C \textit{conv tm} traverses \textit{tm} from its root node to its leaves. It will repeat the application of \textit{conv}, until failure, on each subterm encountered en route. It then descends through the sub-term that results from the repeated application of the conversion. If the descent causes any change, on “coming back out” to the sub-term the conversional will attempt to reapply \textit{conv}, and if successful will then (recursively) reapply \textit{TOPMAP_C conv} once more. If \textit{conv} cannot be reapplied then the conversional continues to ascend back to the root.

It traverses from left to right, though this should only matter for conversions that work by side-effect. It fails if the conversion is applied nowhere within the term.

Errors  \textbf{7005}  Conversion fails on term and all its subterms
\end{verbatim}

\begin{verbatim}
SML
| val TRY_C : CONV --> CONV;

Description  Attempt to apply a conversion, and if it fails, apply refl_conv.
\end{verbatim}

\begin{verbatim}
SML
| val t_thm : THM;

Description  “True” is true.

| \[ \vdash T \]

| t_thm
\end{verbatim}

\begin{verbatim}
SML
| val undisch_rule : THM --> THM ;

Description  UndischARGE the antecedent of an implicative theorem into the assumption list.

| \[ \Gamma \vdash a \Rightarrow b \]

| \[ \Gamma \cup \{a\} \vdash b \]

| undisch_rule

Errors  \textbf{7011}  ?0 is not of the form: ‘\( \Gamma \vdash a \Rightarrow b \)’
\end{verbatim}
Chapter 7. PROOF IN HOL

val varstruct_variant : TERM list -> TERM -> TERM;

Description  varstruct_variant avoid vs will recreate the variable structure vs using only names that are not found in the avoid list of variables, and also renaming to avoid duplicate variable names in the structure. Variant names are found using string_variant (q.v.). If there are duplicates to be renamed, then the original name will be the rightmost in the variable structure.

Errors

3007  ?0 is not a term variable
4016  ?0 is not an allowed variable structure

Message 3007 applies to the avoid list, 27060 to the variable structure.

val v_∃_intro : TERM -> THM -> THM;

Description  Introduce an existential quantified variable structure into a theorem.

Rule

\[ \Gamma \vdash t[x,y,\ldots] \]
\[ \Gamma \vdash \exists \ vs[x,y,\ldots] \bullet t[x,y,\ldots] \]
\[ \Gamma \vdash t[x,y,\ldots] \]

where \( \Gamma \vdash \exists \ vs[x,y,\ldots] \) is a varstruct built from variables \( \Gamma \vdash x \), \( \Gamma \vdash y \), etc, which may contain duplicates.

Uses  If the functionality is sufficient, this is superior in efficiency to both \( \exists \_intro \) and simple \( \exists \_intro \) (q.v.).

Errors

4016  ?0 is not an allowed variable structure

val ⇔_elim : THM -> (THM * THM);

Description  Split a bi-implicative theorem into two implicative theorems.

Rule

\[ \Gamma \vdash t1 \leftrightarrow t2 \]
\[ \Gamma \vdash t1 \Rightarrow t2; \Gamma \vdash t2 \Rightarrow t1 \]
\[ \Gamma \vdash t1 \leftrightarrow t2 \]

Errors

7062  ?0 is not of the form: \' \Gamma \vdash t1 \leftrightarrow t2 \'

val ⇔_intro : THM -> THM -> THM;

Description  Join two implicative theorems into an bi-implicative theorem.

Rule

\[ \Gamma \vdash t1 \Rightarrow t2; \Gamma \vdash t1' \Rightarrow t2' \]
\[ \Gamma \vdash t1 \leftrightarrow t2 \]
\[ \Gamma \vdash t1' \leftrightarrow t2' \]

where \( t1 \) and \( t1' \) are \( \alpha \)-convertible, as are \( t2 \) and \( t2' \).

Errors

7040  ?0 is not of the form: \' \Gamma \vdash t1 \Rightarrow t2 \'
7064  ?0 and ?1 are not of the form: \' \Gamma \vdash t1 \Rightarrow t2; \Gamma \vdash t2a \Rightarrow t1a \'

where \( \Gamma \vdash t1 \), \( \Gamma \vdash t2 \), \( \Gamma \vdash t1a \), and \( \Gamma \vdash t2a \), are \( \alpha \)-convertible
### 7.1. General Inference Rules

<table>
<thead>
<tr>
<th>SML</th>
<th>(\text{val} \iff _\text{match_mp_rule} : \text{THM} \rightarrow \text{THM} \rightarrow \text{THM};)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>A matching Modus Ponens for (\iff).</td>
</tr>
</tbody>
</table>
| **Rule** | \[
\frac{\Gamma \vdash \forall x_1 \ldots \bullet t_1 \iff t_2; \; \Gamma \vdash t_1'}{\Gamma' \cup \Gamma' \vdash t_2'} \iff \_\text{match\_mp\_rule}
\] |
| where we type instantiate, generalise and specialise both conclusion and assumptions to get the first theorem’s LHS to match the conclusion of the second theorem. Universal quantification, or the lack of it, in the first theorem makes no difference to the matching. |
| This may be partially evaluated with only first argument. |
| **See Also** | \(\Rightarrow_{\_\text{elim}}\) (Modus Ponens on \(\Rightarrow\)), \(\text{simple}_{\_\text{iff\_match\_mp\_rule}} \iff_{\_\text{mp\_rule}} \iff_{\_\text{match\_mp\_rule1}}\) |
| **Errors** | 7044 Cannot match ?0 and ?1 |

<table>
<thead>
<tr>
<th>SML</th>
<th>(\text{val} \iff _\text{match_mp_rule1} : \text{THM} \rightarrow \text{THM} \rightarrow \text{THM};)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>A matching Modus Ponens for (\iff) that doesn’t affect assumption lists.</td>
</tr>
</tbody>
</table>
| **Rule** | \[
\frac{\Gamma \vdash \forall x_1 \ldots \bullet t_1 \iff t_2; \; \Gamma \vdash t_1'}{\Gamma' \cup \Gamma' \vdash t_2'} \iff \_\text{match\_mp\_rule1}
\] |
| where \(t_1'\) is an instance of \(t_1\) under type instantiation and substitution for the \(x_i\) and the free variables of the first theorem, and where \(t_2'\) is the corresponding instance of \(t_2\). No type instantiation or substitution will occur in the assumptions of either theorem. |
| This may be partially evaluated with only first argument. |
| **See Also** | \(\Rightarrow_{\_\text{elim}}\) (Modus Ponens on \(\Rightarrow\)), \(\text{simple}_{\_\text{iff\_match\_mp\_rule1}}\) |
| **Errors** | 7044 Cannot match ?0 and ?1 \[\text{Errors}]

<table>
<thead>
<tr>
<th>SML</th>
<th>(\text{val} \iff _\text{mp_rule} : \text{THM} \rightarrow \text{THM} \rightarrow \text{THM};)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This is reminiscent of Modus Ponens, but upon bi-implicative theorems.</td>
</tr>
</tbody>
</table>
| **Rule** | \[
\frac{\Gamma \vdash t_1 \iff t_2; \; \Gamma \vdash t_1'}{\Gamma' \cup \Gamma' \vdash t_2} \iff \_\text{mp\_rule}
\] |
| where \(t_1\) and \(t_1'\) must be \(\alpha\)-convertible. |
| A built-in inference rule. |
| **See Also** | \(\Rightarrow_{\_\text{elim}}\) (true Modus Ponens, on \(\Rightarrow\)), \(\_\text{match\_mp\_rule}\) (a “matching” version of \(\iff_{\_\text{mp\_rule}}\)) |
| **Errors** | 6024 ?0 and ?1 are not of the form: \(t_1' \iff t_2\) and \(t_1' \iff t_2\) \[\text{where} t_1' \iff t_1'\] where \(\_\text{mp\_rule}\) are \(\alpha\)-convertible \[\text{Errors}]

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
<table>
<thead>
<tr>
<th>val _t_elim : THM \rightarrow THM;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong> We can always eliminate ( \leftrightarrow T ).</td>
</tr>
<tr>
<td><strong>Rule</strong></td>
</tr>
<tr>
<td>( \Gamma \vdash t \leftrightarrow T \rightarrow \Gamma \vdash t \leftrightarrow T \leftrightarrow t _elim )</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
</tr>
<tr>
<td>7106 ?0 not of the form (' \Gamma \vdash t \leftrightarrow T')</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>val _t_intro : THM \rightarrow THM;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong> The conclusion of a theorem is equal to ( T ).</td>
</tr>
<tr>
<td><strong>Rule</strong></td>
</tr>
<tr>
<td>( \Gamma \vdash t \rightarrow \Gamma \vdash t \leftrightarrow T \leftrightarrow t _intro )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>val _&amp;_intro : THM \rightarrow THM \rightarrow THM;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong> Conjoin two theorems.</td>
</tr>
<tr>
<td><strong>Rule</strong></td>
</tr>
<tr>
<td>( \Gamma \vdash t1 &amp; t2 \rightarrow \Gamma1 \cup \Gamma2 \vdash t1 &amp; t2 \leftrightarrow &amp; _intro )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>val _&amp;_left_elim : THM \rightarrow THM;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong> Give the left conjunct of a conjunction.</td>
</tr>
<tr>
<td><strong>Rule</strong></td>
</tr>
<tr>
<td>( \Gamma \vdash t1 &amp; t2 \rightarrow \Gamma \vdash t1 \leftrightarrow &amp; _left_elim )</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
</tr>
<tr>
<td>7007 ?0 is not of the form (' \Gamma \vdash t1 &amp; t2')</td>
</tr>
</tbody>
</table>
7.1. General Inference Rules

SML

<table>
<thead>
<tr>
<th>val _rewr...</th>
<th>( THM \rightarrow THM )</th>
</tr>
</thead>
<tbody>
<tr>
<td>val _rewr...</td>
<td>( THM \rightarrow THM )</td>
</tr>
<tr>
<td>val _rewr...</td>
<td>( THM \rightarrow THM )</td>
</tr>
<tr>
<td>val _rewr...</td>
<td>( THM \rightarrow THM )</td>
</tr>
<tr>
<td>val _rewr...</td>
<td>( THM \rightarrow THM )</td>
</tr>
</tbody>
</table>

**Description**  These are some of the standard canonicalisation functions used for breaking theorems up into lists of equations for use in rewriting. They perform the following transformations:

<table>
<thead>
<tr>
<th>( \land_{\text{rewrite_canon}} )</th>
<th>( \Gamma \vdash t1 \land t2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \land_{\text{rewrite_canon}} )</td>
<td>( \Gamma \vdash t1 ; \Gamma \vdash t2 )</td>
</tr>
<tr>
<td>( \land_{\text{rewrite_canon}} )</td>
<td>( \Gamma \vdash \neg (t1 \lor t2) )</td>
</tr>
<tr>
<td>( \land_{\text{rewrite_canon}} )</td>
<td>( \Gamma \vdash \forall x \bullet (x \bullet t) )</td>
</tr>
<tr>
<td>( \land_{\text{rewrite_canon}} )</td>
<td>( \Gamma \vdash \neg t )</td>
</tr>
<tr>
<td>( \iff_{t_{\text{rewrite_canon}}} )</td>
<td>( \Gamma \vdash t1 = t2 )</td>
</tr>
<tr>
<td>( \iff_{t_{\text{rewrite_canon}}} )</td>
<td>( \Gamma \vdash t )</td>
</tr>
<tr>
<td>( \iff_{t_{\text{rewrite_canon}}} )</td>
<td>( \Gamma \vdash F )</td>
</tr>
<tr>
<td>( \iff_{t_{\text{rewrite_canon}}} )</td>
<td>( \Gamma \vdash \forall x \bullet x )</td>
</tr>
<tr>
<td>( \iff_{t_{\text{rewrite_canon}}} )</td>
<td>( \Gamma \vdash t )</td>
</tr>
</tbody>
</table>

Note that the functions whose names begin with `simple` do not handle paired quantifiers. Versions which do handle these quantifiers are also available.

**See Also**  \( \neg_{\text{rewrite\_canon}}, \forall_{\text{rewrite\_canon}} \).

**Errors**  26203 the conclusion of the theorem is already an equation

---

SML

<table>
<thead>
<tr>
<th>val _rewr...</th>
<th>( THM \rightarrow THM )</th>
</tr>
</thead>
<tbody>
<tr>
<td>val _rewr...</td>
<td>( THM \rightarrow THM )</td>
</tr>
<tr>
<td>val _rewr...</td>
<td>( THM \rightarrow THM )</td>
</tr>
<tr>
<td>val _rewr...</td>
<td>( THM \rightarrow THM )</td>
</tr>
<tr>
<td>val _rewr...</td>
<td>( THM \rightarrow THM )</td>
</tr>
</tbody>
</table>

**Description**  Give the right conjunct of a conjunction.

**Rule**

\[
\frac{\Gamma \vdash t1 \land t2}{\Gamma \vdash t2} \quad \land_{\text{right\_elim}}
\]

**Errors**  7007 ?0 is not of the form: \( \Gamma \vdash t1 \land t2 \)

---

SML

<table>
<thead>
<tr>
<th>val _rewr...</th>
<th>( THM \rightarrow THM )</th>
</tr>
</thead>
<tbody>
<tr>
<td>val _rewr...</td>
<td>( THM \rightarrow THM )</td>
</tr>
<tr>
<td>val _rewr...</td>
<td>( THM \rightarrow THM )</td>
</tr>
<tr>
<td>val _rewr...</td>
<td>( THM \rightarrow THM )</td>
</tr>
<tr>
<td>val _rewr...</td>
<td>( THM \rightarrow THM )</td>
</tr>
</tbody>
</table>

**Description**  Expanded form of definition of \( \land \)

**Theorem**

\[
\forall t1 t2 \bullet ((t1 \land t2) \iff (\forall b \bullet (t1 \Rightarrow t2 \Rightarrow b) \Rightarrow b))
\]

---

SML

<table>
<thead>
<tr>
<th>val _rewr...</th>
<th>( THM \rightarrow THM )</th>
</tr>
</thead>
<tbody>
<tr>
<td>val _rewr...</td>
<td>( THM \rightarrow THM )</td>
</tr>
<tr>
<td>val _rewr...</td>
<td>( THM \rightarrow THM )</td>
</tr>
<tr>
<td>val _rewr...</td>
<td>( THM \rightarrow THM )</td>
</tr>
<tr>
<td>val _rewr...</td>
<td>( THM \rightarrow THM )</td>
</tr>
</tbody>
</table>

**Description**  A theorem whose conclusion is an implication from a conjunction is an equivalent to one whose conclusion is an implication of an implication.

**Rule**

\[
\frac{\Gamma \vdash (a \land b) \Rightarrow c}{\Gamma \vdash a \Rightarrow b \Rightarrow c} \quad \land_{\Rightarrow_{\text{rule}}}
\]

**Errors**  7009 ?0 is not of the form: \( \Gamma \vdash (a \land b) \Rightarrow c \)
Description If we know a disjunction is true, and one of its disjuncts is false, then the other must be true. If the second theorem is the negation of both disjuncts, then the second disjunct will be eliminated. (modus tollendo ponens)

\[
\begin{array}{c}
\Gamma_1 \vdash t_1 \lor t_2; \Gamma_2 \vdash \lnot t_1' \\
\hline
\Gamma_1 \cup \Gamma_2 \vdash t_2 \\
\end{array}
\]

\texttt{\_\_\_cancel\_rule}

And:

\[
\begin{array}{c}
\Gamma_1 \vdash t_1 \lor t_2; \Gamma_2 \vdash \lnot t_2' \\
\hline
\Gamma_1 \cup \Gamma_2 \vdash t_1 \\
\end{array}
\]

\texttt{\_\_\_cancel\_rule}

where \(t_1'\) and \(t_1\) are \(\alpha\)-convertible, as are \(t_2\) and \(t_2'\).

Errors

| 7010 | ?0 is not of the form: \(\Gamma \vdash t_1 \lor t_2'\) |
| 7050 | ?0 and ?1 are not of the form: \(\Gamma_1 \vdash t_1 \lor t_2'\) and \(\Gamma_2 \vdash \lnot t_3'\) where \(\lnot t_3'\) is \(\alpha\)-convertible to \(\lnot t_1\) or \(\lnot t_2\) |

Description Given a disjunctive theorem, and two further theorems, each containing one of the disjuncts in their assumptions, but with the same conclusion, we may eliminate the disjunct assumption from the second of the theorems.

\[
\begin{array}{c}
\Gamma_1 \vdash t_1 \lor t_2 \\
\Gamma_2, t_1' \vdash t \\
\Gamma_3, t_2' \vdash t' \\
\hline
\Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \vdash t \\
\end{array}
\]

\texttt{\_\_\_elim}

where \(t_1\) and \(t_1'\) are \(\alpha\)-convertible, as are \(t_2\) and \(t_2'\), and \(t\) and \(t'\). Actually, \(t_1'\) and \(t_2'\) do not have to be present in the assumption lists for this function to work.

Errors

| 7010 | ?0 is not of the form: \(\Gamma \vdash t_1 \lor t_2'\) |
| 7083 | ?0, ?1 and ?2 are not of the form: \(\Gamma_1 \vdash t_1 \lor t_2'\), \(\Gamma_2, t_1a \vdash t_3'\) and \(\Gamma_3, t_2a \vdash t_3a'\), where \(\lnot t_1\) and \(\lnot t_1a\), \(\lnot t_2\) and \(\lnot t_2a\), \(\lnot t_3\) and \(\lnot t_3a\) are each \(\alpha\)-convertible |

Description Introduce a disjunct to the left of a theorem’s conclusion.

\[
\begin{array}{c}
\Gamma \vdash b \\
\hline
\Gamma \vdash a \lor b \\
\end{array}
\]

\texttt{\_\_\_left\_intro}

Errors

| 3031 | ?0 is not of type \(\lnot :BOOL\) |
7.1. General Inference Rules

**val \_right_intro : TERM \rightarrow THM \rightarrow THM;**

**Description** Introduce a disjunct to the right of a theorem’s conclusion.

<table>
<thead>
<tr>
<th>Rule</th>
<th>( \Gamma \vdash b )</th>
<th>( \Gamma \vdash b \lor a )</th>
</tr>
</thead>
</table>

**Errors**

\[3031\] ?0 is not of type \｢BOOL｣

---

**val \_thm : THM;**

**Description** Expanded form of definition of \( \lor \)

<table>
<thead>
<tr>
<th>Theorem</th>
<th>( \forall t1 \ t2 \bullet (t1 \lor t2) \Leftrightarrow (\forall b \bullet (t1 \Rightarrow b) \Rightarrow (t2 \Rightarrow b) \Rightarrow b) )</th>
</tr>
</thead>
</table>

**Errors**

\[3031\] ?0 is not of type \｢BOOL｣

---

**val \_elim : TERM \rightarrow THM \rightarrow THM \rightarrow THM;**

**Description** Given two contradictory theorems with the same assumptions, conclude any other fact from the assumptions: input theorems may be in either order.

<table>
<thead>
<tr>
<th>Rule</th>
<th>( \Gamma_1 \vdash a ); ( \Gamma_2 \vdash \neg a )</th>
<th>( \Gamma_1 \cup \Gamma_2 \vdash b )</th>
</tr>
</thead>
</table>

**Errors**

\[3031\] ?0 is not of type \｢BOOL｣
\[7004\] ?0 and ?1 are not of the form: ‘\( \Gamma_1 \vdash a \)’ and ‘\( \Gamma_2 \vdash \neg a \)’

---

**val \_eq_sym_rule : THM \rightarrow THM ;**

**Description** If \( a \) is not equal to \( b \) then \( b \) is not equal to \( a \).

<table>
<thead>
<tr>
<th>Rule</th>
<th>( \Gamma \vdash \neg(a = b) )</th>
<th>( \Gamma \vdash \neg(b = a) )</th>
</tr>
</thead>
</table>

**Errors**

\[7091\] ?0 is not of form: ‘\( \Gamma \vdash \neg(a = b) \)’

---

**val \_intro : TERM \rightarrow THM \rightarrow THM \rightarrow THM;**

**Description** Given two theorems with contradictory conclusions (up to \( \alpha \)-convertibility), their assumptions must be inconsistent, and thus any member of the lists (or indeed, anything else) may be proven false on the assumption of the remainder (reductio ad absurdum).

<table>
<thead>
<tr>
<th>Rule</th>
<th>( \Gamma_1 \vdash b ); ( \Gamma_2 \vdash \neg b )</th>
<th>( (\Gamma_1 \cup \Gamma_2) \setminus {a} \vdash \neg a )</th>
</tr>
</thead>
</table>

Works up to \( \alpha \)-conversion, and input theorems may be in either order.

**Errors**

\[3031\] ?0 is not of type \｢BOOL｣
\[7004\] ?0 and ?1 are not of the form: ‘\( \Gamma_1 \vdash a \)’ and ‘\( \Gamma_2 \vdash \neg a \)’

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
val ¬_simple_∀_conv : CONV;

Description  Move ¬ into a ∀ construct.

\[ \vdash (\neg (\forall x \cdot t[x])) \iff \exists x \cdot \neg t[x] \]

This will work with any simple universal quantifier.

Errors 7036  ?0 not of the form: \( \neg (\forall x \cdot t[x]) \)

---

val ¬_simple_∃_conv : CONV;

Description  Move ¬ into an ∃ construct.

\[ \vdash (\neg (\exists x \cdot t[x])) \iff \forall x \cdot \neg t[x] \]

This will work with any simple existential quantifier.

Errors 7058  ?0 is not of the form: \( \neg (\exists x \cdot t[x]) \)

where \( x \) is a variable

---

val ¬_thm1 : THM;

Description  “Not t if and only if t is false.”

\[ \vdash \forall t \cdot (\neg t) \iff (t \iff F) \]

¬_thm1

---

val ¬_thm : THM;

Description  Expanded form of definition of ¬:

\[ \forall t \cdot (\neg t) \iff (t \Rightarrow F) \]

¬_thm

---

val ¬_t_thm : THM;

Description  “Not true is false”.

\[ \neg T \iff F \]

¬_t_thm

---

val ¬_¬_conv : CONV;

Description  A double negation is redundant.

\[ \Gamma \vdash \neg (\neg t) \iff t \]

¬_¬_conv

Errors 7022  ?0 not of the form: \( \neg (\neg t) \)
7.1. General Inference Rules

SML
val \neg\neg\text{elim} : \text{THM} \rightarrow \text{THM};

Description  A double negation is redundant.

Rule
\begin{align*}
\Gamma & \vdash \neg (\neg t) \\
\Gamma & \vdash t
\end{align*}
\neg\neg\text{elim}

Errors  7006 ?0 is not of the form: ‘\(\Gamma \vdash \neg (\neg t)\)’

SML
val \neg\neg\text{intro} : \text{THM} \rightarrow \text{THM};

Description  We may always introduce a double negation.

Rule
\begin{align*}
\Gamma & \vdash t \\
\Gamma & \vdash \neg (\neg t)
\end{align*}
\neg\neg\text{intro}

SML
val \neg\forall\text{conv} : \text{CONV};

Description  Move \(\neg\) into a \(\forall\) construct.

Rule
\begin{align*}
\vdash (\neg (\forall x \bullet t[x])) & \leftrightarrow \exists x \bullet \neg t[x] \\
\neg\forall\text{conv} & \neg (\forall x \bullet t[x]) \neg
\end{align*}

See Also  \neg\text{simple.\forall.conv} which only works with simple \(\forall\)-abstractions, \neg\exists\text{conv}

Errors  27019 ?0 not of the form: \(\neg (\forall x \bullet t[x])\neg\)

\text{where } {\neg x} \neg \text{ is a varstruct}

SML
val \neg\forall\text{thm} : \text{THM};

Description  Used in pushing negations through simple universal quantifications.

Theorem
\begin{align*}
\vdash \forall p \bullet \neg \forall p & \leftrightarrow (\exists x \bullet \neg p x) \\
\neg\forall\text{thm}
\end{align*}

SML
val \neg\exists\text{conv} : \text{CONV};

Description  Move \(\neg\) into an \(\exists\) construct.

Rule
\begin{align*}
\vdash (\neg (\exists x \bullet t[x])) & \leftrightarrow \forall x \bullet \neg t[x] \\
\neg\exists\text{conv} & \neg (\exists x \bullet t[x]) \neg
\end{align*}

See Also  \neg\text{simple.\exists.conv} which only works with simple \(\exists\)-abstractions, \neg\forall\text{conv}

Errors  27020 ?0 is not of the form: \(\neg (\exists x \bullet t[x])\neg\)

\text{where } {\neg x} \neg \text{ is a varstruct}
\[
\text{val } \neg \exists \text{thm} : \text{THM};
\]

**Description**  Used in pushing negations through simple existential quantifications.

\[
\begin{array}{cc}
\vdash \forall x \bullet \neg \exists x \iff (\forall x \bullet p x) & \neg \exists \text{thm}
\end{array}
\]

\[
\text{SML}
\begin{align*}
\text{val } \Rightarrow \text{elim} & : \text{THM } \rightarrow \text{THM } \rightarrow \text{THM}; \\
\text{val } \Rightarrow \text{mp_rule} & : \text{THM } \rightarrow \text{THM } \rightarrow \text{THM};
\end{align*}
\]

**Description**  Modus Ponens (which is why we introduce the alias \(\Rightarrow \text{mp_rule}\), though \(\Rightarrow \text{elim}\) is shorter, conventional, and the preferred name).

\[
\frac{\Gamma \vdash t_1 \Rightarrow t_2; \Gamma_2 \vdash t_1'}{\Gamma_1 \cup \Gamma_2 \vdash t_2} \Rightarrow \text{elim}
\]

where \(t_1\) and \(t_1'\) must be \(\alpha\)-convertible. A primitive inference rule.

**See Also**  \(\Leftarrow \text{mp_rule}\) (Modus Ponens on \(\Leftarrow\)), \(\Leftarrow \text{match_mp_rule}\) (a “matching” version of this function).

\[
\begin{align*}
6010 & \text{?0 is not of the form: } \Gamma \vdash t_1 \Rightarrow t_2 \text{'} \\
6011 & \text{?0 and ?1 are not of the forms: } \Gamma \vdash t_1 \Rightarrow t_2 \text{'} \text{ and } \Gamma_2 \vdash t_1' \text{'} \text{ where } \Gamma t_1 \text{'}\text{ and } \Gamma t_1' \text{ are } \alpha-\text{convertible}
\end{align*}
\]

\[
\text{SML}
\begin{align*}
\text{val } \Rightarrow \text{intro} & : \text{TERM } \rightarrow \text{THM } \rightarrow \text{THM};
\end{align*}
\]

**Description**  Prove an implicative theorem, removing, if \(\alpha\)-convertibly present, the antecedent of the implication from the assumption list.

\[
\frac{\Gamma \vdash t_2}{\Gamma - \{t_1\} \vdash t_1 \Rightarrow t_2} \Rightarrow \text{intro}
\]

A primitive inference rule.

**See Also**  \(\text{disch_rule}\) (which fails if term not in assumption list)

\[
\begin{align*}
3031 & \text{?0 is not of type } \Gamma : \text{BOOL}\text{'}
\end{align*}
\]
7.1. General Inference Rules

**Rule**
\[
\frac{\Gamma \vdash t_1 \Rightarrow t_2; \Gamma \vdash t_1'}{\Gamma \cup \Gamma \vdash t_2'} \Rightarrow \text{match\_mp\_rule}
\]

where we type instantiate, generalise and specialise to get the first theorem’s antecedent to match the conclusion of the second theorem. Universal quantification, or the lack of it, in the first theorem makes no difference to the matching.

This may be partially evaluated with only the first argument.

**See Also**  \( \Rightarrow \text{match\_mp\_rule1}, \Rightarrow \text{elim} \)

**Errors**
7044 Cannot match ?0 and ?1

**Description**  A matching Modus Ponens rule for an implicative theorem.

**Rule**
\[
\frac{\Gamma \vdash \forall x_1 \ldots \bullet t_1 \Rightarrow t_2; \Gamma \vdash t_1'}{\Gamma \cup \Gamma \vdash t_2'} \Rightarrow \text{match\_mp\_rule}
\]

where \( t_1' \) is an instance of \( t_1 \) under type instantiation and substitution for the \( x_i \) and the free variables of the first theorem, and where \( t_2' \) is the corresponding instance of \( t_2 \). The type instantiations and substitutions are obtained by matching \( t_1 \) and \( t_1' \) using \text{term\_match}.

\( \Rightarrow \text{match\_mp\_rule2} \) is just like \( \Rightarrow \text{match\_mp\_rule1} \) except that the instantiations and substitutions returned by \text{term\_match} are extended to replace type variables that do not occur in \( t_1 \) or in \( \Gamma_1 \) and \( x_{-i} \) that do not occur free in \( t_1 \) by fresh variables to avoid clashes with each other and with the type variables and free variables of \( \Gamma_1 \) and \( \Gamma_2 \).

Types in the assumptions of the theorems will not be instantiated.

Both rules may be partially evaluated with only the first argument.

**Errors**
7044 Cannot match ?0 and ?1
7045 ?0 is not of the form \( '\Gamma \vdash \forall x_1 \ldots x_n \bullet u \Rightarrow v' \)

**Rule**
\[
\frac{\Gamma \vdash t_1 \Rightarrow t_2; \Gamma \vdash t_2'}{\Gamma \cup \Gamma \vdash t_3} \Rightarrow \text{trans\_rule}
\]

where \( t_2 \) and \( t_2' \) are \( \alpha \)-convertible.

**Errors**
7040 ?0 is not of the form: \( '\Gamma \vdash t_1 \Rightarrow t_2' \)
7042 ?0 and ?1 are not of the form: \( '\Gamma_1 \vdash t_1 \Rightarrow t_2', '\Gamma_2 \vdash t_2a \Rightarrow t_3' \)

where \( 't_2' \) and \( 't_2a' \) are \( \alpha \)-convertible.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
\[ val \Rightarrow \land \text{rule} : \text{THM} \to \text{THM}; \]

**Description** A theorem whose conclusion is an implication of an implication is equivalent to one whose conclusion is a conjunction and an implication.

\[
\begin{align*}
\Gamma & \vdash a \Rightarrow b \Rightarrow c \\
\Gamma & \vdash (a \land b) \Rightarrow c
\end{align*}
\]

\[ \Rightarrow \land \text{rule} \]

**Errors** 7008 \( \text{?0 is not of the form: '}' \Gamma \vdash a \Rightarrow b \Rightarrow c' \)

\[ val \ \forall \text{arb\_elim} : \text{THM} \to \text{THM}; \]

**Description** Specialise a universally quantified theorem with a machine generated variable or variable structure.

\[
\begin{align*}
\Gamma & \vdash \forall \text{vs}[x,y,...]\bullet p[x,y,...] \\
\Gamma & \vdash p[x',y',...]
\end{align*}
\]

\[ \forall \text{arb\_elim} \]

where \( x', y', \text{etc}, \text{are not variables (free or bound) in } p \text{ or } \Gamma, \text{created by } \text{gen\_vars}(q.v). \)

**See Also** \( \forall \text{\_elim} \)

**Errors** 27011 \( \text{?0 is not of the form: '}' \Gamma \vdash \forall x \bullet t' \text{where } \uparrow x \downarrow \text{is a varstruct} \)

\[ val \ \forall \text{asm\_rule} : \text{TERM} \to \text{TERM} \to \text{THM} \to \text{THM}; \]

**Description** Generalise an assumption (Left \( \forall \) introduction).

\[
\begin{align*}
\Gamma, p'[x] & \vdash q[x] \\
\Gamma & \vdash x \bullet p'[x] \vdash q[x]
\end{align*}
\]

\[ \forall \text{asm\_rule} \]

\[ \forall p[x] \downarrow \]

where \( p \text{ and } p' \) are \( \alpha \)-convertible. \( x \text{ may be free in } \Gamma \). The function will work even if \( p'[x] \) is not present in the assumption list.

**Errors** 4016 \( \text{?0 is not an allowed variable structure} \)
7.1. General Inference Rules

SML

\text{val } \forall \text{ elim } : \text{TERM } \rightarrow \text{THM } \rightarrow \text{THM};

**Description**  Specialise a universally quantified theorem with a given value, instantiating the type of the theorem as necessary.

\[
\begin{array}{c}
\Gamma \vdash \forall \ x \bullet t2[x] \\
\Gamma \vdash t2'[t1]
\end{array}
\quad \forall \text{ elim}
\quad \Gamma \vdash t1^\gamma
\]

where \(t2'\) is renamed from \(t2\) to prevent bound variable capture and possibly type instantiated, and \(x\) is a varstruct, instantiable to the structure of \(t1\). The value \(t1\) will be expanded using \(\text{Fst}\) and \(\text{Snd}\) as necessary to match the structure of \(\Gamma x\).

**See Also**  list.\(\forall\) elim, all.\(\forall\) elim.

**Errors**

27011  ?0 is not of the form: \(\Gamma \vdash \forall \ x \bullet t\) where \(\Gamma x\) is a varstruct

27012  ?0 is not of the form: \(\Gamma \vdash \forall \ x \bullet t\) where the type of \(\Gamma x\) is instantiable to the type of \(?1\)

27013  ?0 is not of the form: \(\Gamma \vdash \forall \ x \bullet t\) where the type of \(\Gamma x\) is instantiable to the type of \(?1\) without instantiating type variables in the assumptions

SML

\text{val } \forall \text{ intro } : \text{TERM } \rightarrow \text{THM } \rightarrow \text{THM};

**Description**  Introduce a universally quantified theorem.

\[
\begin{array}{c}
\Gamma \vdash t \\
\Gamma \vdash \forall \ x' \bullet t
\end{array}
\quad \forall \text{ intro}
\quad \Gamma \vdash \Gamma x^\gamma
\]

Where \(\Gamma x'^\gamma\) is an allowed variable structure based on \(\Gamma x\), but with duplicate variables renamed, the original name being rightmost in the resulting variable structure.

**See Also**  list.\(\forall\) intro, all.\(\forall\) intro.

**Errors**

4016  ?0 is not an allowed variable structure

6005  ?0 occurs free in assumption list
val ∀ reorder conv : TERM → CONV;

Description  Reorder universal quantifications.

\[ ∀ \, \text{reorder conv} \quad ∀ \mathbin{\Rightarrow} \quad \forall x_1 \ldots \forall x_n \mathbin{\Rightarrow} \forall y_1 \ldots \forall y_m \mathbin{\Rightarrow} \]

where the \( x_i \) and \( y_i \) are varstructs, and the reordering, restructuring (by pairing) and renaming requested is provable by this function. The presence of redundant quantifiers, including duplicates, is also handled.

Example:
\[ ∀ \, \text{reorder conv} \quad ∀ (x, q) \, z \, x \land z \mathbin{\Rightarrow} ∀ (z, y) \, x \land x \quad : \text{THM} \]

Note that before more sophisticated attempts, the conversion will try $\alpha$-conversion on the two term arguments.

See Also  $∃ \, \text{reorder conv}$

Errors
\[ 27050 \quad \text{Cannot prove equality of } ?0 \text{ and } ?1 \]

val ∀ uncurry conv : CONV;

Description  Convert a paired universally quantified term into simple universal quantifications of the same term.

\[ ∀ \, \text{uncurry conv} \quad ∀ \mathbin{\Rightarrow} \quad ∀ vs \mathbin{\Rightarrow} \forall x \, y \ldots \mathbin{\Rightarrow} ∀ [x, y, \ldots] \mathbin{\Rightarrow} f [x, y, \ldots] \]

where \( vs \) is an allowed variable structure with variables \( x, y, \ldots \). It may not be a simple variable.

See Also  $λ \, \text{varstruct conv}, \, \text{all}_\forall \, \text{uncurry conv}.$

Errors
\[ 27038 \quad ?0 \text{ is not of the form: } ∀ (x, y) \mathbin{\Rightarrow} f \]

val ∀ _⇔_rule : TERM → THM → THM;

Description  Universally quantify a variable on both sides of an equivalence.

\[ ∀ \, _⇔_rule \quad ∀ \mathbin{\Rightarrow} \quad ∀ p \mathbin{\Rightarrow} q \quad ∀ (\forall x \mathbin{\Rightarrow} p \mathbin{\Rightarrow} q) \mathbin{\Rightarrow} \quad ∀ \mathbin{\Rightarrow} \]

where \( x \) is a varstruct.

Errors
\[ 6005 \quad ?0 \text{ occurs free in assumption list} \]
\[ 6020 \quad ?0 \text{ is not of the form: } 'Γ ⊢ t1 = t2' \]
\[ 7062 \quad ?0 \text{ is not of the form: } 'Γ ⊢ t1 \iff t2' \]
\[ 4016 \quad ?0 \text{ is not an allowed variable structure} \]
7.1. General Inference Rules

**val **\( \exists \text{asm\_rule} : \text{TERM} \rightarrow \text{TERM} \rightarrow \text{THM} \rightarrow \text{THM} \);**

**Description **Existentially quantify an assumption (Left \( \exists \) introduction).

\[
\frac{\Gamma, p'[x] \vdash q \quad \exists_{\text{asm\_rule}}}{\Gamma, \exists x \bullet p'[x] \vdash q}
\]

where \( p \) and \( p' \) are \( \alpha \)-convertible, where the variables of the varstruct \( x \) are not free in \( \Gamma \) or \( q \). The assumption need not be present for the rule to apply.

**Errors**

- 3015 ?1 is not of type \( \vdash \text{BOOL} \)
- 4016 ?0 is not an allowed variable structure
- 6005 ?0 occurs free in assumption list
- 27052 ?0 has members appearing free in ?1 other than in assumption ?2

**Message** 3015 is just passed on from low level functions, which is why it has ”?1” not ”?0”.

**val **\( \exists \text{elim} : \text{TERM} \rightarrow \text{THM} \rightarrow \text{THM} \rightarrow \text{THM} \);**

**Description **Eliminate an existential quantifier by reference to an arbitrary varstruct satisfying the predicate.

\[
\frac{\Gamma_1 \vdash \exists \text{vs}[x_1,x_2,...]\bullet t_1[x_1,x_2,...]; \quad \exists_{\text{elim}} \quad \Gamma_2, t_1[y_1,y_2,...] \vdash t_2}{\Gamma_1 \cup \Gamma_2 \vdash t_2}
\]

where \( \Gamma_1 \) need not actually be present in the assumptions of the second theorem. The \( y_i \) must be free variables, none of whom are present elsewhere in the second theorem, or in the conclusion of the first. The \( y_i \) may contain duplicates as long as the end pattern matches the \( x_i \) in required duplicates. The term argument may be a less complex variable structure than the bound variable structure of the theorem, as \( \text{Fst} \) and \( \text{Snd} \) are used to make them match. For example, the following rule holds true:

\[
\frac{\Gamma_1 \vdash \exists (p,q)\bullet t_1[p,q]; \quad \exists_{\text{elim}} \quad \Gamma_2, t_1[\text{Fst } x, \text{Snd } x] \vdash t_2}{\Gamma_1 \cup \Gamma_2 \vdash t_2}
\]

**Errors**

- 27042 ?0 does not match the bound varstruct of ?1
- 27046 ?0 is not of the form ‘\( \Gamma \vdash \exists \text{vs}\bullet t'\)
- 27051 ?0 has members appearing free in conclusion of ?1
- 27052 ?0 has members appearing free in ?1 other than in assumption ?2

**val **\( \exists \text{intro\_thm} : \text{THM} \);**

**Description **Introduction of existential quantification.

\[
\frac{\vdash \forall P \ x \bullet P \ x \Rightarrow \exists P}{\exists_{\text{intro\_thm}}}
\]
\begin{center}
\begin{verbatim}
SML
val \exists_intro : TERM -> THM -> THM ;

Description Introduce an existential quantifier by reference to a witness.

Rule
\[
\Gamma \vdash t \left[ t_1, t_2, \ldots \right] \quad \exists_intro \quad \Gamma \vdash \exists vs[x', y', \ldots ] \bullet t[x, y, \ldots ]
\]
where \(\Gamma \vdash \exists vs[x, y, \ldots ]\) is varstruct built from variables \(\Gamma x, \Gamma y, \ldots\), and the \(\Gamma x'\) are renamed if duplicated inside the varstruct, all but the rightmost being so renamed.

Errors
4020 \(\exists\) is not of form: \(\exists vs \bullet t\)
7047 \(\exists\) cannot be matched to conclusion of theorem $\exists$

\end{verbatim}
\end{center}

\begin{center}
\begin{verbatim}
SML
val \exists_reorder_conv : TERM -> CONV ;

Description Reorder existential quantifications.

Rule
\[
(\exists y_1 \ldots y_m \bullet t_2) \Leftrightarrow (\exists x_1 \ldots x_n \bullet t_1) \quad \exists_reorder_conv \quad (\exists x_1 \ldots x_n \bullet t_1) \Leftrightarrow (\exists y_1 \ldots y_m \bullet t_2)
\]
where the \(x_i\) and \(y_i\) are varstructs, and the reordering, restructuring (by pairing) and renaming requested is provable by this function. The presence of redundant quantifiers, including duplicates, is also handled.

Example
\begin{verbatim}
: \exists_reorder_conv \(\exists (x, q) \bullet x \land z \land \exists (z, z) \bullet x \land z^\prime\)
val it = \(\exists (x, q) \bullet x \land z \land \exists (z, z) \bullet x \land z) : THM
\end{verbatim}
\end{verbatim}
\end{center}

\begin{center}
\begin{verbatim}
SML
val \exists_uncurry_conv : CONV ;

Description Convert a paired existentially quantified term into simple universal quantifications of the same term.

Conversion
\[
\Gamma \vdash \exists vs[x, y, \ldots ] \bullet f[x, y, \ldots ] \quad \exists_uncurry_conv \quad \exists vs[x, y, \ldots ] \bullet f[x, y, \ldots ]
\]
where \(\exists vs[x, y, \ldots ]\) is an allowed variable structure with variables \(x, y, \ldots\). It may not be a simple variable.

See Also \(\lambda_varstruct_conv, all_\exists_uncurry_conv, \forall_uncurry_conv\)

Errors
27047 \(\exists\) is not of the form: \(\exists (x, y) \bullet f\)
\end{verbatim}
\end{center}
7.1. General Inference Rules

SML
\begin{verbatim}
val \exists_{-\epsilon}.conv : CONV;
\end{verbatim}

**Description**
Give that \( \epsilon \) of a predicate satisfies the predicate by reference to an \( \exists \) construct. It can properly handle paired existence.

\begin{align*}
\Gamma \vdash (\exists x \cdot p(x)) &= p(\epsilon x \cdot p x) \\
\implies \exists_{-\epsilon}.conv \quad \exists x \cdot p[x]^\top
\end{align*}

If \( x \) is formed by paired then the \( Fst \) and \( Snd \) are used to extract the appropriate bits of the \( \epsilon \)-term for distribution in \( p[\epsilon x \cdot p x] \).

**See Also** \( \exists_{-\epsilon}.rule \)

**Errors**
\texttt{27024} ?0 is not of the form: \texttt{‘\( \Gamma \vdash \exists x \cdot p[x] \)’}
where \texttt{‘\( x \)’} is a varstruct

SML
\begin{verbatim}
val \exists_{-\epsilon}.rule : THM \rightarrow THM;
\end{verbatim}

**Description**
Give that \( \epsilon \) of a predicate satisfies the predicate by reference to an \( \exists \) construct. It can properly handle paired existence.

\begin{align*}
\Gamma \vdash \exists x \cdot p[x] \\
\implies \exists_{-\epsilon}.rule \quad \exists_{-\epsilon}.rule
\end{align*}

If \( x \) is formed by paired then the \( Fst \) and \( Snd \) are used to extract the appropriate bits of the \( \epsilon \)-term for distribution in \( p[\epsilon x \cdot p x] \).

**See Also** \( \exists_{-\epsilon}.conv \)

**Errors**
\texttt{27024} ?0 is not of the form: \texttt{‘\( \Gamma \vdash \exists x \cdot p[x] \)’}
where \texttt{‘\( x \)’} is a varstruct

SML
\begin{verbatim}
val \exists_{1}.conv : CONV;
\end{verbatim}

**Description**
This is a conversion which turns a unique existential quantifier into an equivalent existential quantifier

\begin{align*}
\vdash (\exists_{1} vs[x_{1},...]\cdot t[x_{1},...]) &\iff \\
(\exists vs[x_{1},...]\cdot t[x_{1},...] \land \\
\forall vs[x_{1}',...,t[x_{1}',...] \Rightarrow \\
vs[x_{1}',...] = vs[x_{1},...]) &\implies \exists_{1}.conv \quad \exists_{1} vs[x_{1},...]\cdot t[x_{1},...]^\top
\end{align*}

**Uses** Tactic and conversion programming.

**See Also** \texttt{strip_tac}, \texttt{simple \_1.conv}

**Errors**
\texttt{27053} ?0 is not of the form: \texttt{‘\( \exists_{1} vs\cdot t \)’}
\textbf{SML} \texttt{val \exists_1\textunderscore elim : THM \rightarrow THM;}

\textbf{Description} Express a \(\exists_1\) in terms of \(\exists\) and a uniqueness property.

\begin{align*}
\text{Rule} & \quad \Gamma \vdash \exists \text{ vs } [a,b,...] \bullet P[a,b,...] \\
& \quad \Gamma \vdash \exists \text{ vs } [a,b,...] \bullet P[a,b,...] \land \\
& \quad \quad \forall \text{ vs } [a',b',...] \bullet P[x1,x2,...] \\
& \quad \Rightarrow \\
& \quad \quad \text{vs}[a',b',...] = \text{vs}[a,b,...]
\end{align*}

where the \(a',\) etc, are variants of the \(a\).

\textbf{Errors} \\
\(27022\) ?0 is not of the form: \(\Gamma \vdash \exists_1 x \bullet P[x]\) where \(\Gamma\) is a varstruct

\textbf{SML} \texttt{val \exists_1\textunderscore intro : THM \rightarrow THM \rightarrow THM;}

\textbf{Description} Introduce \(\exists_1\) by reference to a witness, and a uniqueness theorem.

\begin{align*}
\text{Rule} & \quad \Gamma 1 \vdash P'[t'] \\
& \quad \Gamma 2 \vdash \forall x \bullet P[x] \Rightarrow x = t \\
& \quad \Gamma 1 \cup \Gamma 2 \vdash \exists_1 x \bullet P[x]
\end{align*}

Where \(P'\) is \(\alpha\)-convertible to \(P\), and \(t'\) is \(\alpha\)-convertible to \(t\). Notice that for the resulting theorem we take the varstruct, \(x\), and the form of the predicate, \(P\), from the second theorem.

\textbf{Errors} \\
\(27021\) ?0 and ?1 are not of the form: \(\Gamma 1 \vdash Pa[ta]\) and \(\Gamma 2 \vdash \forall vs[x,y,...] \bullet P[x,y,...] \Rightarrow vs[x,y,...] = t'\) where \(\Gamma Pa\) and \(\Gamma P\), \(\Gamma ta\) and \(\Gamma t\) are \(\alpha\)-convertible and \(\Gamma x\) is a varstruct

\(27054\) ?0 not of the form: \(\Gamma \vdash \forall vs[x,y,...] \bullet P[x,y,...] \Rightarrow vs[x,y,...] = t'\)

\textbf{SML} \texttt{val \exists_1\textunderscore thm : THM;}

\textbf{Description} Expanded form of definition of \(\exists_1\)

\begin{align*}
\text{Theorem} & \quad \vdash \forall P \bullet (\exists_1 P) \iff \\
& \quad \quad (\exists t \bullet (P \ t) \land \\
& \quad \quad \quad (\forall x \bullet (P \ x) \Rightarrow x = t))
\end{align*}

\textbf{SML} \texttt{val \alpha\textunderscore conv : TERM \rightarrow CONV;}

\textbf{Description} Returns a theorem that two terms are equal, should they be \(\alpha\)-convertible.

\begin{align*}
\text{Rule} & \quad \vdash t1 = t2
\end{align*}

\textbf{Errors} \\
\(3012\) ?0 and ?1 do not have the same types

\(7034\) ?0 and ?1 are not \(\alpha\)-convertible
7.1. General Inference Rules

SML

\texttt{val } \beta\texttt{-conv } : \texttt{CONV};

\textbf{Description} Apply a \( \beta \)-reduction to an abstraction.

\textbf{Rule}

\[ \Gamma \vdash (\lambda x \bullet t[x])y = t'[y] \]

\( \beta\texttt{-conv} \)

where \( x \) may be any \texttt{varstruct} allowed by the ICL HOL syntax, \( y \) is an instance of this structure, and \( t' \) is \( \alpha \)-convertible to \( t \), changed to avoid variable capture.

When the bound variable structure has a pair, where the value applied to does not, then \texttt{Fst} and \texttt{Snd} are introduced as necessary, e.g.:

\textbf{Example}

\[ \beta\texttt{-conv} \Gamma \vdash (\lambda (x,y) \bullet f x y) p \]

\[ = \Gamma \vdash (\lambda (x,y) \bullet f x y) p = f \ (\texttt{Fst} p) \ (\texttt{Snd} p) \]

\textbf{See Also} \texttt{simple-\beta\texttt{-conv}}, \texttt{\beta\texttt{-rule}}

\textbf{Errors}

\texttt{27008} \ ?0 is not of the form: \( \Gamma \vdash (\lambda x \bullet t1[x])t2 \)

where \( \Gamma \vdash x \) is a \texttt{varstruct}

\begin{verbatim}
SML
\texttt{val } \beta\texttt{-rule } : \texttt{THM } \rightarrow \texttt{THM};

\textbf{Description} An elimination rule for \( \lambda \), which can handle paired abstractions.

\textbf{Rule}

\[ \Gamma \vdash (\lambda x \bullet t[x]) y \]

\[ \Gamma \vdash t[y] \]

\( \beta\texttt{-rule} \)

\textbf{See Also} \texttt{\beta\texttt{-conv}}

\textbf{Errors}

\texttt{27007} \ ?0 is not of the form: \( \large\Gamma \vdash (\lambda x \bullet t[x]) \)

\( y \)

where \( \Gamma \vdash x \) is a \texttt{varstruct}
\end{verbatim}
val ϵ_elim_rule : TERM -> THM -> THM -> THM;

Description  Given that ϵ of a predicate satisfies that predicate, then in a different theorem we may eliminate an assumption that claims an otherwise unused variable structure satisfies the predicate.

\[
\begin{array}{c}
\Gamma_1 \vdash t' (\$\epsilon\ t''); \\
\Gamma_2, t vs \vdash s \\
\Gamma_1 \cup \Gamma_2 \vdash s
\end{array}
\]  

\(\epsilon\_elim\_rule\)

where \(t, t'\) and \(t''\) are α-convertible, and \(vs\) is a varstruct, with no duplicates, and with its free variables occurring nowhere else in the second theorem, or in the conclusion of the first. In fact, \((\$\epsilon\ t'')\) here can be any term, it is not constrained to be an application of the choice function.

Errors

4016  ?0 is not an allowed variable structure
7019  ?0 is not of the form: ‘\(\Gamma \vdash t_1 (\epsilon t_1)\)’
7054  ?0 is not of same type as choice sub-term of first theorem
27043  ?0 is repeated in the varstruct ?1
27045  Arguments ?0; ?1 and ?2 not of the form \(\Gamma vs\); ‘\(\Gamma_1 \vdash t (\epsilon t)\)’ and ‘\(\Gamma_2, (t vs) \vdash s\)’
27051  ?0 has members appearing free in conclusion of ?1
27052  ?0 has members appearing free in ?1 other than in assumption ?2

val ϵ_intro_rule : THM -> THM;

Description  Given a theorem whose conclusion is a function application, we know that the “function” is a predicate, and the rule states that ϵ of this predicate will satisfy the predicate.

\[
\begin{array}{c}
\Gamma \vdash t_1 t_2 \\
\Gamma \vdash t_1 (\$\epsilon t_1)
\end{array}
\]  

\(\epsilon\_intro\_rule\)

Errors

7016  ?0 is not of the form: ‘\(\Gamma \vdash t_1 t_2\)’

val η_conv : CONV;

Description  The rule for η conversion.

\[
\begin{array}{c}
\vdash (\lambda vs\bullet t vs) = t
\end{array}
\]  

\(\eta\_conv\)

where \(t\) contains no free instances of the variables of varstruct \(vs\).

Errors

27018  ?0 is not of the form: ‘\(\lambda vs\bullet t vs\)’
where ‘\(vs\)’ is a varstruct
27023  ?0 is not of the form: ‘\(\lambda vs\bullet t vs\)’ where ‘\(t\)’ should not contain ‘\(vs\)’
7.1. General Inference Rules

**SML**

\[
\text{val } \lambda C : \text{CONV } \rightarrow \text{CONV};
\]

**Description**  Apply a conversion to the body of an abstraction:

\[
\Gamma \vdash (\lambda x \bullet p[x]) = (\lambda x \bullet pa[x])
\]

where \( c \ p[x] \) gives \( \Gamma \vdash \Gamma \).

**Errors**

| 4002 | ?0 is not of form: \( \Gamma \ vs \bullet t \) |
| 7104 | Result of conversion, ?0, ill-formed |

Also as the failure of the conversion.

**SML**

\[
\text{val } \lambda \text{_eq\_rule } : \text{TERM } \rightarrow \text{THM } \rightarrow \text{THM};
\]

**Description**  Given an equational theorem, return the equation formed by abstracting the term argument (which must be an allowed variable structure) from both sides.

\[
\Gamma \vdash t1[x] = t2[x] \\
\Gamma \vdash (\lambda x \bullet t1[x]) = (\lambda x \bullet t2[x])
\]

**Errors**

| 4016 | ?0 is not an allowed variable structure |
| 6005 | ?0 occurs free in assumption list |
| 6020 | ?0 is not of the form: \( \Gamma \vdash t1 = t2 \) |

**SML**

\[
\text{val } \lambda \text{_pair\_conv } : \text{CONV};
\]

**Description**  This conversion eliminates abstraction over pairs in favour of abstraction over elements of pairs. The bound variables of the resulting \( \lambda \)-abstraction do not have pair types.

\[
\Gamma \vdash (\lambda v \bullet t) = (\lambda (v1, v2) \bullet t'[\{v1, v2}/v])
\]

\[
\Gamma \vdash (\lambda (v, w) \bullet t) = (\lambda ((v1, v2), (w1, w2)) \bullet t'[\{v1, v2}/v, (w1, w2)/v])
\]

and so on.

**Errors**

| 27055 | The type of ?0 is not of the form \( \sigma \times \tau \) |
**SML**

```sml
val λ_rule : TERM -> THM -> THM;
```

**Description**  An introduction rule for λ:

\[
\Gamma \vdash s[t] \\
\Gamma \vdash (\lambda x \cdot s[x]) t
\]

where \( x \) is a machine generated variable.

```
val λ_varstruct_conv : TERM -> CONV;
```

**Description**  This conversion is a generalisation of \( \alpha_{conv} \) allowing one to convert a \( \lambda \)-abstraction into an equivalent \( \lambda \)-abstraction that differs only in the form of the varstruct and the corresponding use of \( \text{Fst} \) in the \( \text{Snd} \) in the body of the abstraction.

\[
\Gamma \vdash (\lambda vs2[x2,y2,...]\bullet t[x2,y2,...]) =
(\lambda vs1[x1,y1,...]\bullet t'[x1,y1,...])
\]

Where the types of \( vs1[x1,y1,...] \) and \( vs2[x2,y2,...] \) are the same, and \( t' \) and \( t \) differ only in applications of \( \text{Fst} \) and \( \text{Snd} \) to the bound variables.

For example,

\[
\Gamma \vdash ((\lambda x \bullet \text{Fst } x + \text{Snd } x = 1) =
(\lambda (a, b) \bullet a + b = 1)
\]

**See Also**  \( \alpha_{conv} \) for a more limited form of renaming.

**Errors**

`27050 Cannot prove equality of ?0 and ?1`
7.2 Subgoal Package

**SML**

```sml
signature SubgoalPackage = sig
```

**Description** This provides the subgoal package, which provides an interactive backward proof mechanism, based on the application of tactics.

**Errors**

- **30009** There are no goals to prove
- **30017** Label ?0 has no corresponding goal
- **30023** ?0 cannot be interpreted as a goal
- **30028** Label may not contain ?0, as less than 1
- **30041** Label ?0 has been superseded
- **30042** Label may not contain 0
- **30043** Label ?0 has been achieved
- **30045** Label cannot be empty
- **30055** The last change to the subgoal package state was made in a context which is no longer valid
- **30056** The current goal contains distinct free variables with the same names but different types, the names being ?0, and a typing context is being maintained.
- These free variables have not been put in the typing context
- **30059** The current goal contains two or more distinct free variables with the same name but different types, the name being ?0, and a typing context is being maintained.
- These free variables have not been put in the typing context
- **30061** The tactic generated an invalid proof (?0). The goal state has not been changed

These messages are common to various functions in this document. Message 30055 indicates that the goal state theorem failed the `valid_thm` test: this could be a theory out of scope, a deletion of a definition, etc. Messages 30056 and 30057 are just for the user’s information, though they should give cause to worry.

**SML**

```sml
(* pp'TS *)
```

**Description** The theory will contain a constant named `pp'TS`, defined by a definition with key “pp'TS”.

**Definition**

\[ \forall x \ (pp'TS \ x) \Leftrightarrow x \]

This is used in creating a term form goal. Using this constant explicitly within the subgoal package may cause unexpected behaviour.

**Uses** The definition may be used when analysing goal state theorems, or using `modify_goal_state_thm` (q.v.) - both operations are only for the advanced user or extender of the system.

**SML**

```sml
(* subgoal_package_quiet : bool *)
```

**Description** This is a system control, handled by `set_flag`. If set to false (the default) then the package narrates its progress as described in the design of its components. If set to true then the package will cause no output other than the actual results of functions. This includes, e.g., `print_goal` and `apply_tactic`.

**Uses** For running the package offline.
Chapter 7. PROOF IN HOL

(* subgoal_package_ti_context : bool *)

**Description**  
subgoal_package_ti_context is a system control flag, as handled by set_flag, etc. If set to true (the default) then the type context will be set and maintained, via set_ti_context(q.v.), to be just the free variables of the current goal, each time the current goal changes. If false, then the type context will be cleared and left unchanged by goal state changes. If the current goal has free variables with the same name and differing types this will cause set_ti_context to ignore those variables, raising the comment message 30056.

(* tactic_subgoal_warning : integer control *)

**Description**  
Warning 30018 will be issued by apply_tactic (and a) if the tactic requests more subgoals than the number set by this control. This allows the user to avoid processing and printing large numbers of subgoals when these are probably unwanted. The default value is 20. If the value is less than zero then the warning will never be issued.

(* undo_buffer_length : int *)

**Description**  
This is a system control, handled by set_int_control, etc, which sets the maximum number of entries that can be held on the undo buffer for each main goal: i.e. how many tactic applications, etc, may be undone. It is initially set to 12, and cannot be made negative. Any changes to this parameter will take immediate effect upon the undo buffers stored for all the main goals, i.e. if necessary they will be shortened at the point of changing the value, rather than at the point of, e.g., applying a new tactic.

**GOAL_STATE;**

**Description**  
This is an abstract data type that embodies a goal state, in particular it contains which goals are yet to be achieved and a theorem embedding the inference work so far. The subgoal package has a current goal state, a stack of goal states for different main goals, and a buffer of goal states to allow some operations to be undone.

**See Also**  
print_goal_state
7.2. Subgoal Package

```sml
val apply_tactic : TACTIC -> unit;
val a : TACTIC -> unit;
```

**Description**  
`apply_tactic` applies a tactic to the current goal, and `a` is an alias for it. If successful, the previous goal state will be put in the undo buffer, and the new goal state, current goal, etc, will be based on the tactic’s application. If the tactic returns some subgoals then the “first” of these will become the new current goal. If there is only one subgoal it will inherit the label of the previous current goal, otherwise if the old label was “label” then it will be considered in the goal state as superseded, and the new subgoals will be labeled “label.1”, “label.2”, etc. If it produces a theorem that achieves the current goal (i.e. the list of subgoals is empty), then the “next” goal will become the current one, and the previous goal’s label will be noted as achieved.

The subgoals created, or if none, the “next” goal, will be displayed, using the format of `print_goal(q.v)`, but with goal labels also given. Following the display of the new goals the subgoal package will issue warning messages about these goals if they are somehow “suspicious”: for example it will warn if the goal state is not changed by applying the tactic.

Warning 30018 will be issued if the tactic requests more subgoals than the number set by control `tactic_subgoal_warning`. This allows the user to avoid processing and printing large numbers of subgoals when these are probably unwanted.

**See Also**  
`print_goal` for the display format of the goals.

**Errors**

```text
30007  There is no current goal
30008  Result of tactic, ?0, did not match the current goal
30018  Tactic has requested ?0 subgoals, which exceeds the threshold
       set by tactic_subgoal_warning
```

```sml
val drop_main_goal : unit -> GOAL;
```

**Description**  
Pop the current goal state from the main goal stack throwing away it and any work upon it, and making the previous entry on the stack the new current goal state, displaying the current top goal, if appropriate. The function returns the main goal dropped.

**Errors**

```text
30010  The subgoal package is not in use
```

```sml
val get_asm : int -> TERM;
```

**Description**  
`get_asm n` returns the nth assumption of the current goal.

**Errors**

```text
30026  There is no current goal
30027  There is no assumption ?0 in the current goal
```
val modify_goal_state_thm : (THM -> THM) -> ((string list * GOAL) list) -> unit;

Description  modify_goal_state_thm rule label is a powerful hook into the subgoal package that works as follows:

1. Extract the goal state theorem
2. Apply a user-supplied inference rule rule to the theorem.
3. Make a new goal state, in which the goal state theorem is this new theorem.
4. In the new goal state any goals found (up to α-conversion) in the association list label will be labelled with their corresponding labels in the association list. Multiple entries for the same goal in the list will cause the labels to be accumulated, resulting in duplicated goals in the new goal state. If top_goals() (q.v.) is used for this association list then all unchanged goals will gain their original labels.
5. Label otherwise unlabelled goals with unused single natural number labels (the first available ones from the list “1”, “2”,...)
6. Treat this new goal state as if it had been created by a tactics application, e.g. it becomes the current goal state, the previous goal state is put on the undo list, the user is told the next goal to prove, etc.

This will issue a warning on its use should the main goal have changed, and on attempting to extract an achieved, or goal state, theorem from a goal state that is derived from the modified one. This is so that the user is warned that the result of an apparently successful pop_thm is not an achievement of the initially set main goal.

Uses  This function is intended for system builders wishing to write extensions to the package that change the overall proof tree, not an individual goal.

Errors
30039  Two labels clash: ?0 and ?1
30040  Duplicate labels ?0 given for different terms
30051  Inference rule returned '?[0]' which is not a goal state theorem

val pending_reset_subgoal_package : unit -> unit -> unit;

Description  This function, applied to () takes a snapshot of the current subgoal package state - its stack of goal states, undo and redo buffers, and implicitly the current goal label, etc. This snapshot, if then applied to () will overwrite the then current subgoal package state with the snapshot. This does not reset, e.g., the current theory to the one at the time of taking the snapshot, so care must be taken in using this function.

Uses  Primarily in saving the subgoal package state between sessions of ProofPower, via save_and_quit.
7.2. Subgoal Package

SML

val pop_thm : unit -> THM;

**Description** If the top achieved theorem is available (i.e. the theorem whose sequent is the main goal has been achieved), this function returns it, and then pops the previous goal state (if any) off the main goal stack, restoring its current goal and labelling. If present, the new current top goal will be displayed in the format used by print_goal. If the current proof is incomplete the function fails, having no effect.

If the user wishes to examine the top achieved theorem without popping the main goal stack, then they should use top_thm (q.v.).

The user will be informed if main goal has changed from the initially set main goal, by using modify_goal_state_thm (q.v).

**See Also** save_pop_thm, top_thm

**Errors**

| 30010 | The subgoal package is not in use |
| 30011 | The current proof is incomplete |

SML

val print_current_goal : unit -> unit;

**Description** Displays, with its label, the current goal of the current goal state: the goal to which a tactic will be applied.

**Errors**

| 30026 | There is no current goal |

SML

val print_goal_state : GOAL_STATE -> unit;

**Description** Display the given goal state. This displays the main goal, the goals yet to be proven, and the current goal.

SML

val print_goal : GOAL -> unit;

**Description** Display a goal (i.e. a conclusion and a list of assumptions) in the manner of the other subgoal package functions. This presents the list of assumptions in the goal first, numbered by their position, and in reverse order, and then the conclusion, distinguished from the assumptions by a turnstile.

**Example**

```
(* 3 *)  ρa ⇒ ¬ b
(* 2 *)  ρa ⇒
         a ⇒
         a ⇒ b
(* 1 *)  ρ¬ b ⇒ a
(* ?¬ *)  ρa ∨ b
```

where ρ¬ b ⇒ a is the first assumption, and the second assumption is too long to fit on one line. Then with no assumptions:

**Example**

```
(* ?¬ *)  ρa ∨ b
```
val push_goal_state_thm : THM -> unit;

Description  Given a theorem that is of the form of a goal state theorem (e.g. gained by top_goal_state_thm, q.v.), set a new current main goal to be the conclusion of the input theorem (viewed as a term form goal). The current goal in the new goal state will be the first assumption of the input theorem, viewed as a term form goal. If it is the only assumption of the theorem argument then the corresponding goal will have label "1"; otherwise label "1", and the other assumptions of the theorem will become subsequent goals with labels "2", "3", .... This new goal state is pushed onto the main goal stack. The current undo buffer will also be stacked, and a new empty one made current.

Uses  For the advanced user, interested in partial proof.

Errors  30005  ?0 cannot be viewed as a goal state theorem
        30058  Two distinct variables with name ?0 occur free in the goal

val push_goal_state : GOAL_STATE -> unit;

Description  If the value given is “well-formed”, then this function pushes the current goal state onto the main goal stack, and sets the given value as the current goal state. The most likely reason that a goal state value is ill-formed is that it is not being pushed in the same context as it was formed, e.g. it was formed in a theory that is now out of scope, e.g. because the user has changed theory since the states creation. The current undo buffer will also be stacked, and a new empty one made current.

See Also  top_goal_state

val push_goal : GOAL -> unit;

Description  Sets a new current main goal, creating an appropriate goal state and pushing it onto the main goal stack. The current (and only) goal in the new goal state will be the main goal, with label "1". The current goal will be displayed. The current undo buffer will also be stacked, and a new empty one made current.

See Also  set_goal

Errors  30002  The conclusion of the goal, ?0, is not of type BOOL
        30003  An assumption of the goal, ?0, is not of type BOOL
        30004  Two assumption of the goal (?0 and ?1) are α—convertible
        30058  Two distinct variables with name ?0 occur free in the goal

val redo : unit -> unit;

Description  If the last command to affect the goal state was an undo(q.v) then this command will undo its effect (including leaving the undo buffer in its previous form, without mention of the undo or redo).

Errors  30014  The last command to affect the goal state was not an undo
7.2. Subgoal Package

SML

val save_pop_thm : string -> THM;

Description If the top achieved theorem is available (i.e. the theorem whose sequent is the main goal has been achieved), this function returns it, as well as saving it under the given string key on the current theory, and then pops the previous goal state (if any) of the main goal stack, restoring its current goal and labelling. If present, the new current top goal will be displayed in the format used by print_goal. If the current proof is incomplete, or the key is already used in the current theory, the function fails, having no effect.

The user will be informed if main goal has changed from the initially set main goal, by using modify_goal_state_thm(q.v).

The user will be informed if main goal has changed from the initially set main goal, by using modify_goal_state_thm(q.v).

See Also pop_thm, top_thm

Errors
30010 The subgoal package is not in use
30011 The current proof is incomplete

Failures also as save_thm, but given as originating from this function.

SML

val set_goal : GOAL -> unit;

Description This first discards, if it exists, the current main goal (but not any previously pushed main goals). It then sets a new current main goal, creating an appropriate goal state and pushing it onto the main goal stack. The current (and only) goal in the new goal state will be the main goal, with label "". The current goal will be displayed. The current undo buffer will also be stacked, and a new empty one made current.

Defn
set_goal gl = (drop_main_goal() handle (Fail _) => () ;
push_goal gl);

Uses In restarting a proof that has “gone wrong”, perhaps by

|set_goal(top_main_goal());

See Also push_goal

Errors
30002 The conclusion of the goal, ?0, is not of type BOOL
30003 An assumption of the goal, ?0, is not of type BOOL
30004 Two assumption of the goal (?0 and ?1) are α-convertible
30058 Two distinct variables with name ?0 occur free in the goal

SML

val set_labelled_goal : string -> unit;

Description If the string is a valid label in the current goal state, then set the corresponding goal as the current goal, and then display it.

Errors
30010 The subgoal package is not in use
30016 ?0 is not of the form "n1.n2....nm"
### simplify_goal_state_thm

**Type**: `unit -> THM;

**Description**: This will simplify a goal state theorem (e.g. from `top_goal_state_thm`, q.v.), stripping off assumptions from the conclusion of the theorem up to the turnstile place marker, then removing the place marker itself in both conclusion and assumptions.

**Uses**: For the advanced user, interested in partial proofs.

**Errors**: 30005 ?0 cannot be viewed as a goal state theorem

### subgoal_package_size

**Type**: `unit -> int;

**Description**: This returns the size of the subgoal package’s storage, in words - where one word is four bytes.

This facility is not available in all versions of ProofPower. The function will produce the following warning message and return −1 in this case

**Errors**: 30060 This function is not supported in this version of ProofPower

### top_current_label

**Type**: `unit -> string;

**Description**: Returns the label of the current goal: the goal to which a tactic will be applied.

**Errors**: 30026 There is no current goal

### top_goals

**Type**: `unit -> (string list * GOAL)list;

**Description**: Returns all the goals yet to be achieved, and their associated labels (they may have more than one), in the current goal state.

**Uses**: To determine what goals are left to achieve.

**Errors**: 30010 The subgoal package is not in use

### top_goal_state_thm

**Type**: `unit -> THM;

**Description**: This returns the goal state theorem of the current goal state. It is a partial proof of the main goal, though in a somewhat unwieldy form, as it encodes the main goal, and its other goals in a term form. It may be simplified by using `simplify_goal_state_thm`(q.v). The theorem is suitable for setting a new main goal, by using `push_goal_state_thm`(q.v). The user is informed if the goal state has achieved its theorem The user will also be informed if main goal has changed from the initially set main goal, by using `modify_goal_state_thm`(q.v).

**Uses**: For the advanced user, interested in partial proofs.

**Errors**: 30010 The subgoal package is not in use

---

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
7.2. Subgoal Package

```
SML
|val top_goal_state : unit -> GOAL_STATE;

Description  This provides the current goal state as a value: note that a goal state does not
contain an undo buffer, and thus function does not return the current undo buffer.

See Also  push_goal_state

Errors
|30010 The subgoal package is not in use
```

```
SML
|val top_goal : unit -> GOAL;

Description  Returns the current goal of the current goal state: the goal to which a tactic will
be applied.

Errors
|30026 There is no current goal
```

```
SML
|val top_labelled_goal : string -> GOAL;

Description  Returns the goal with the given label, should it exist in the current goal state.
Note that superseded and achieved goals are not available from the goal state.

Errors
|30016 ?0 is not of the form "n1.n2....nm"
```

```
SML
|val top_main_goal : unit -> GOAL;

Description  Return the current main goal: the objective of the current proof attempt.

Errors
|30025 There is no current main goal
```

```
SML
|val top_thm : unit -> THM;

Description  If the top achieved theorem (i.e. the theorem whose sequent is the main goal has
been achieved) is available, this function returns it, without affecting the current goal state. If
the current proof is incomplete the function fails.

The user will be informed if main goal has changed from the initially set main goal, by using
modify_goal_state_thm(q.v).

See Also  pop_thm, save_pop_thm

Errors
|30010 The subgoal package is not in use
|30011 The current proof is incomplete
```
val undo : int -> unit;

**Description**  
undo n will take the n-th entry from the undo buffer, if there are sufficient, as the current goal state. Attempting to go past the end of the buffer will cause a failure, rather than a partial undoing. A single undo command can itself be undone by redo(q.v), but otherwise entries on the undo buffer between its start and the n-th entry will be discarded.

Note that the undo buffer is stacked on starting a new main goal (e.g. with push_goal), and unstacked on popping the current main goal (e.g. with pop_thm or drop_main_goal).

**Errors**

30010  *The subgoal package is not in use*
30012  *Attempted to undo 0 time?1 with only ?2 entr?3 in the undo buffer*
30013  *Must undo a positive number of times*
7.3 General Tactics and Tacticals

**SML**

\[ \text{signature Tactics1} = \text{sig} \]

**Description** This provides the first group of tactics and tacticals in ICL HOL.

**SML**

\[ \text{signature Tactics2} = \text{sig} \]

**Description** This provides the second group of tactics and tacticals in ICL HOL. These are mainly concerned with the predicate calculus.

**SML**

\[ \text{signature Tactics3} = \text{sig} \]

**Description** This provides a third group of tactics. They are primarily concerned with adding handling for paired abstractions.

**SML**

\[ \text{type GOAL} \quad (*) = \text{SEQ} \quad *; \]
\[ \text{type PROOF} \quad (*) = \text{THM list} \rightarrow \text{THM} \quad *; \]
\[ \text{type TACTIC} \quad (*) = \text{GOAL} \rightarrow (\text{GOAL list} \times \text{PROOF} ^ \ast); \]

**Description** TACTIC is the type of tactics. The types GOAL and PROOF help to abbreviate its definition.

**SML**

\[ \text{val accept_tac : THM} \rightarrow \text{TACTIC}; \]

**Description** Prove a goal by a theorem which is \( \alpha \)-convertible to it.

\[
\{ \Gamma 2 \} \quad t2 \quad \text{accept_tac} \quad \Gamma 1 \vdash t1
\]

where \( t1 \) and \( t2 \) are \( \alpha \)-convertible.

**Errors**

\[ 9102 \quad ?0 \text{ is not } \alpha \text{–convertible to the goals conclusion } ?1 \]

**SML**

\[ \text{val all_asm_ante_tac : TACTIC}; \]

**Description** Apply \( \text{asm_ante_tac} \) to every assumption in turn:

\[
\{ \text{t1, ... , t}_n \} \quad t \quad \text{all_asm_ante_tac} \quad \{ \text{t}_n \Rightarrow \ldots \Rightarrow \text{t1} \Rightarrow t \}
\]

\( \alpha \)-equivalent assumptions will only appear once in the resulting goal. Notice that the first assumption becomes the rightmost antecedent.

**See Also** \( \text{asm_ante_tac, list_asm_ante_tac} \)

**Errors**

\[ 28055 \quad \text{The conclusion or an assumption of goal does not have type } "\text{BOOL}" \]
val all_var_elim_asm_tac : TACTIC;
val all_var_elim_asm_tac1 : TACTIC;
val ALL_VAR_ELIM_ASM_T : (THM -> TACTIC) -> TACTIC;
val ALL_VAR_ELIM_ASM_T1 : (THM -> TACTIC) -> TACTIC;

Description These tactics and tacticals do variable elimination with all the appropriate assumptions of the goal. They process one or more assumptions of the form: \( \text{"var = value"} \) or \( \text{"value = var"} \), where \( \text{var} \) is a variable and the subterm \( \text{value} \) satisfies a tactic-specific requirement, eliminating the variable \( \text{var} \) in favour of the \( \text{value} \).

If an assumption is an equation of variables, which all of the listed tactics accept, then the tactic will strip digits and the current variant suffix from the right of the two variable names, and will choose to eliminate the variable with the shortest remaining name string, taking eliminating the left hand side variable if the strings are of equal length (this is a heuristic). If the variables are the same then the assumption is just discarded with no further effect.

\textit{all_var_elim_asm_tac} will first extract all the goal’s assumptions, holding them in a “pool”. It will examine each assumption of the required form in turn, starting at the assumptions from the head of the assumption list. To eliminate a variable \( \text{var} \) using an assumption it requires that the \( \text{value} \) to which it is equated is also a variable, or an isolated constant (this is more restrictive than \textit{var_elim_asm_tac}). All the occurrences of the variable will be eliminated from the rest of the assumptions in the pool, and from the conclusion of the goal, and the assumption discarded from the pool. Each of the assumptions in the pool will be examined once, as the process described so far will only exceptionally introduce new equations that can be used for variable elimination.

Finally, the remaining assumptions in the pool will be returned to the goal’s assumption list - if an individual assumption is unchanged then it will be returned by \textit{check_asm_tac}, otherwise it will be stripped back into the assumption list by \textit{strip_asm_tac}. This stripping may result in further possible variable eliminations being enabled, and indeed certain fairly unlikely combinations of assumptions and proof contexts may result in \textit{REPEAT all_var_elim_asm_tac} not halting. \textit{ALL_VAR_ELIM_ASM_T} allows the users choice of function to be applied to the modified assumptions, rather than \textit{strip_asm_tac}.

\textit{all_var_elim_asm_tac1} works as \textit{all_var_elim_asm_tac}, except that an assumption will be used to eliminate a variable \( \text{var} \) if the \( \text{value} \) to which it is equated does not contain \( \text{var} \) free (i.e. its requirement is as \textit{var_elim_asm_tac}). \textit{ALL_VAR_ELIM_ASM_T1} allows the users choice of function to be applied to the modified assumptions.

All the functions fail if they find no assumptions that can be used to eliminate variables.

Uses General purpose, and in \textit{basic_prove_tac}.

See Also \textit{prop_eq_prove_tac} for more sophisticated approach to these kinds of problems.

Errors 29028 This tactic is unable to eliminate any variable
7.3. General Tactics and Tacticals

SML
\texttt{val all\_\beta\_tac : TACTIC;}

\textbf{Description}  This tactic will \(\beta\)-reduce all \(\beta\)-redexes in the goal’s conclusion, including those redexes introduced by preceding \(\beta\)-reductions in the same tactic application.

\textbf{Uses}  In most proof contexts \(\beta\)-reduction will be a side effect of rewriting: this tactic is intended for cases where rewriting would do “too much”.

\textbf{See Also}  all\_\beta\_rule, all\_\beta\_conv

Errors
\texttt{[27049} ?0 contains no \(\beta\)-redexes

\texttt{val all\_\epsilon\_tac : TACTIC;}
\texttt{val ALL\_\epsilon\_T : (THM \rightarrow TACTIC) \rightarrow TACTIC;}

\textbf{Description}  \texttt{all\_\epsilon\_tac} applies \(\epsilon\_tac\) to all subterms of the conclusion of the goal of the form \(\epsilon x \bullet t\). \texttt{ALL\_\epsilon\_T} is similar but uses \(\epsilon T\) rather than \(\epsilon\_tac\). The effect is to set the corresponding terms of the form \(\exists x \bullet t\) as lemmas, and to derive new assumptions of the form \(t[\epsilon x \bullet t/x]\).

\begin{align*}
\text{Tactic} & \frac{\{ \Gamma \} \ t[\epsilon x_1 \bullet t_1/y_1, \ldots, \epsilon x_k \bullet t_k/y_k]}{\{ \Gamma \} \ \exists x_1 \bullet t_1; \ldots; \{ \Gamma \} \ \exists x_k \bullet t_k; \ \{ \text{strip } t_1[\epsilon x_1 \bullet t_1/x_1], \ldots, \ \text{strip } t_k[\epsilon x_k \bullet t_k/x_k], \Gamma \} \ t} \ \epsilon\_tac
\end{align*}

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable. This occurs when the use of the choice function is in some sense irrelevant to the truth of the goal, e.g., \((\epsilon x \bullet T) = (\epsilon x \bullet T)\).

\texttt{val ante\_tac : THM \rightarrow TACTIC;}

\textbf{Description}  Replace a goal with conclusion \(t_2\) by \(t_1 \Rightarrow t_2\), where the antecedent, \(t_1\), of the implication is the conclusion of a theorem:

\begin{align*}
\text{Tactic} & \frac{\{ \Gamma \} \ t_2}{\{ \Gamma \} \ t_1 \Rightarrow t_2} \ \ante\_tac \ (\Gamma \vdash t_1)
\end{align*}

where the assumptions, \(\Gamma\), of the theorem are contained in the assumptions, \(\Gamma\) of the goal.

\textbf{Uses}  This is often useful if one needs to transform the conclusion of theorem e.g. by rewriting with the assumptions.

\textbf{See Also}  asm\_tac, strip\_asm\_tac

Errors
\texttt{[28027} Conclusion of goal does not have type \(\vdash \text{BOOL}\)

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
Description
Bring a term out of the assumption list into the goal as the antecedent of an implication.

Rule
\[
\begin{align*}
\{ \Gamma', t_1' \} & \vdash t_2 \\
\{ \Gamma \} & \vdash t_1 \Rightarrow t_2
\end{align*}
\]

where \( t_1 \) and \( t_1' \) are \( \alpha \)-convertible. Note that all assumptions \( \alpha \)-convertible with \( t_1 \) are removed.

Uses
Typically to make the assumption amenable to manipulation, e.g. by a rewriting tactic.

See Also
list_asm_ante_tac, all_asm_ante_tac, swap_asm_concl_tac, DROP_ASM_T.

Errors
28052 Term ?0 is not in the assumptions
28055 The conclusion or an assumption of goal does not have type "BOOL"
7.3. General Tactics and Tacticals

\begin{verbatim}
val back_chain_tac : THM list -> TACTIC;
val bc_tac : THM list -> TACTIC;
\end{verbatim}

**Description**  
`back_chain_tac` is a tactic which uses theorems whose conclusions are possibly universally quantified implications or bi-implications, to reason backwards from the conclusion of a goal.  (`bc_tac` is an alias for `back_chain_tac`. ) The tactic repeatedly performs the following steps:

1. Scan the list of theorems looking for an implication, \( t_1 \Rightarrow t_2 \), or a bi-implication \( t_1 \Leftrightarrow t_2 \) for which the conclusion of the goal is a substitution instance, \( t_2' \) say, of \( t_2 \). If no such theorem is found then stop.

2. If in step 1, an applicable theorem, say \( \text{thm} \), has been found reduce the goal to the corresponding instance of \( t_1 \) (or an existentially quantified version thereof) using \( bc_\text{thm}_\text{tac} \), q.v.

3. Repeatedly apply \( \forall \text{tac} \) or \( \land \text{tac} \) until neither of these is applicable.

4. Delete \( \text{thm} \) from the list of theorems and return to step 1.

In step 4, only the first appearance of \( \text{thm} \) is removed from the list, so that one can arrange for a theorem to be used more than once by the tactic by putting several copies of it in the list.

For example:

\[ \begin{align*}
\{ \Gamma \} & \quad t_3' \\
\{ \Gamma \} & \quad \Rightarrow \text{tac} \\
\{ \Gamma \} & \quad t_4'; \quad \{ \Gamma \} \quad t_5'
\end{align*} \]

\[ \Rightarrow \text{tac} \\
\Gamma_1 \vdash t_1 \land (\forall x \bullet t_2) \Rightarrow t_3, \\
\Gamma_2 \vdash t_4 \Leftrightarrow t_1, \\
\Gamma_3 \vdash t_5 \Rightarrow t_2, \\
\]

(Here \( t_3' \) is some substitution instance of \( t_3 \) and \( t_4' \) and \( t_5' \) are the corresponding instances of \( t_4 \) and \( t_5 \).)

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

**See Also**  
`bc_\text{thm}_\text{tac}` (which is used to perform step 2).

\begin{verbatim}
29012 Theorem ?0 is not of the form \( ' \Gamma \vdash \forall x_1 \ldots x_n \bullet u \leftrightarrow v \) \n\text{or} \quad ' \Gamma \vdash \forall x_1 \ldots x_n \bullet u \Rightarrow v
\end{verbatim}
val back_chain_thm_tac : THM -> TACTIC;
val bc_thm_tac : THM -> TACTIC;

Description  back_chain_thm_tac is a tactic which uses a theorem whose conclusion is a possibly universally quantified implication or bi-implication to chain backwards one step from the conclusion of a goal. (bc_thm_tac is an alias for back_chain_thm_tac.) The effect is as follows:

\[
\begin{align*}
& \{ \Gamma \} \ t2' & & \text{bc_thm_tac} & & \Gamma I \vdash t1 \Rightarrow t2 \\
& \{ \Gamma \} \ t1' & & \text{bc_thm_tac} & & \Gamma I \vdash t1 \iff t2
\end{align*}
\]

where \( t2' \) is an instance (under type instantiation and substitution) of \( t2 \) and \( t1' \) is the corresponding instance of \( t1 \). If \( t1' \) contains free variables which do not appear in the assumptions of the instantiated theorem or in \( t2' \), then the new subgoal \( t1' \) will be existentially quantified over these variables. For example,

\[
\begin{align*}
& \{ \Gamma \} \ a < b & & \text{bc_thm_tac} & & \vdash \forall i \cdot a < i \land i < b \\
& \{ \Gamma \} \exists i \cdot a < i & & \text{bc_thm_tac} & & \vdash \forall m \ i \ n \cdot m < i \land i < n \Rightarrow m < n
\end{align*}
\]

Note that, bi-implications are in effect treated as right-to-left rewrite rules at the top level by this tactic. The standard rewriting mechanisms may be used for left-to-right rewriting.

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

See Also  back_chain_tac (which supplies a more general facility).

Errors

29011  Conclusion of the goal is not an instance of: ?0
29012  Theorem ?0 is not of the form '\( \Gamma \vdash \forall x1 \ldots xn \cdot u \Leftrightarrow v' \) or '\( \Gamma \vdash \forall x1 \ldots xn \cdot u \Rightarrow v' \)

Uses  Specialised low-level tactic programming.

val bad_proof : string -> 'a

Description  bad_proof name is equivalent to error name 9001 []

Errors  9001  the proof of the subgoals has produced the wrong number of theorems

Uses  Specialised low-level tactic programming.
7.3. General Tactics and Tacticals

SML

\[
\begin{aligned}
\text{val } \text{CASES\_T2} & : \text{TERM }\rightarrow (\text{THM }\rightarrow \text{TACTIC}) \rightarrow \\
& \quad (\text{THM }\rightarrow \text{TACTIC}) \rightarrow \text{TACTIC}; \\
\text{Description} & \quad \text{Do a case split on a given boolean term using two tactic generating functions:}
\end{aligned}
\]

\[
\text{CASES\_T2 } t1 \ t tac1 \ t tac2 \ ((\Gamma) \ t2) = \ t tac1(t1 \vdash t1)(\{\Gamma\} \ t2) ; \ t tac2(\neg t1 \vdash \neg t1)(\{\Gamma\} \ t2)
\]

\text{See Also} \ cases\_tac, \lor\ _THEN, \ CASES\_T

\text{Errors} \ [28022 ] \ ?0 \ is \ not \ boolean

SML

\[
\begin{aligned}
\text{val } \text{cases\_tac} & : \text{TERM }\rightarrow \text{TACTIC}; \\
\text{Description} & \quad \text{Do a case split on a given boolean term.}
\end{aligned}
\]

\[
\begin{aligned}
\text{Tactic} & \quad \{ \Gamma \} \ t2 \\
\text{cases\_tac} & \quad \text{\texttt{\{}strip } t1, \Gamma \text{\texttt{\}}} \ t2; \ {\texttt{\{}strip } \neg t1, \Gamma \text{\texttt{\}}} \ t2; \\
\text{\texttt{\"\{}tt1\texttt{\}}} \\
\text{See Also} & \quad \text{CASES\_T, \lor\ _THEN}
\end{aligned}
\]

\text{Errors} \ [28022 ] \ ?0 \ is \ not \ boolean

SML

\[
\begin{aligned}
\text{val } \text{CASES\_T} & : \text{TERM }\rightarrow (\text{THM }\rightarrow \text{TACTIC}) \rightarrow \text{TACTIC}; \\
\text{Description} & \quad \text{Do a case split on a given boolean term using a tactic generating function:}
\end{aligned}
\]

\[
\text{CASES\_T } t1 \ t tac \ ((\Gamma) \ t2) = \ t tac(t1 \vdash t1)(\{\Gamma\} \ t2) ; \ t tac(\neg t1 \vdash \neg t1)(\{\Gamma\} \ t2)
\]

\text{See Also} \ cases\_tac, \lor\ _THEN, \ CASES\_T2

\text{Errors} \ [28022 ] \ ?0 \ is \ not \ boolean

SML

\[
\begin{aligned}
\text{val } \text{CHANGED\_T} & : \text{TACTIC }\rightarrow \text{TACTIC}; \\
\text{Description} & \quad \text{CHANGED\_T } t ac \text{ is a tactic which applies } t ac \text{ to the goal and fails if this results in a single subgoal which is } \alpha \text{–convertible to the original goal.}
\end{aligned}
\]

\text{Uses} \ CHANGED\_T \text{ can be a useful way of ensuring termination of, e.g., rewriting tactics.}

\text{Errors} \ [9601 ] \ \text{the tactic did not change the goal
**SML**

```sml
val check_asm_tac : THM -> TACTIC;
```

**Description**  
`check_asm_tac thm` is a tactic which checks the form of the theorem, `thm`, and then takes the first applicable action from the following table:

<table>
<thead>
<tr>
<th><code>thm</code></th>
<th><code>action</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Γ ⊢ t</code></td>
<td>proves goal if its conclusion is <code>t</code></td>
</tr>
<tr>
<td><code>Γ ⊢ T</code></td>
<td><code>as id_tac</code> (i.e. the theorem is discarded)</td>
</tr>
<tr>
<td><code>Γ ⊢ F</code></td>
<td>proves goal</td>
</tr>
<tr>
<td><code>Γ ⊢ ¬t</code></td>
<td>proves goal if <code>t</code> in assumptions, else <code>asm_tac</code></td>
</tr>
<tr>
<td><code>Γ ⊢ t</code></td>
<td>proves goal if <code>¬t</code> in assumptions, else <code>asm_tac</code></td>
</tr>
</tbody>
</table>

During the search through the assumptions in the last two cases, `check_asm_tac` also checks to see whether any of the assumptions is equal to the conclusion of the goal, and if so proves the goal. It also checks to see if the conclusion of the theorem is already an assumption, in which case the tactic has no effect. When all the assumptions have been examined, if none of the above actions is applicable, the conclusion of the theorem is added to the assumption list.

**Uses**  
Tactic programming.

**See Also**  
`strip_asm_tac`, `strip_tac`.

**SML**

```sml
val concl_in_asm_tac : TACTIC;
```

**Description**  
`concl_in_asm_tac` is a tactic which checks whether the conclusion of the goal is also in the assumptions, and if so proves the goal.

```sml
Tactic { Γ, t } t' concl_in_asm_tac
```

where `t` and `t'` are α-convertible.

**Uses**  
Tactic programming.

**See Also**  
`strip_tac`.

**Errors**  
28002  
`Goal does not appear in the assumptions`

**SML**

```sml
val COND_T : (GOAL -> bool) -> TACTIC -> TACTIC -> TACTIC;
```

**Description**  
`COND_T p tac1 tac2` is a tactic which acts as `tac1` if the predicate `p` holds for the goal, otherwise it acts as `tac2`.

**Example**  

```
COND_T (is_¬ o snd) (cases_tac (⌜X:BOOL⌝)) strip_tac
```

is a tactic which does a case split on `⌜X⌝` if the goal is a negation and behaves as `strip_tac` otherwise.

**Uses**  
For constructing larger tactics, in cases where the more common idiom using `ORELSE` would not have the desired effect.

**See Also**  
`ORELSE`

**Errors**  
As determined by the arguments.
7.3. General Tactics and Tacticals

SML
|val contr_tac : TACTIC;

**Description**  A form of proof by contradiction: \( t \) holds if \( \neg t \vdash F \).

(The name stands for classical contradiction, as opposed to the intuitionistic contradiction proof of \( i\text{\_contr\_tac} \).

\[
\begin{array}{c}
\text{Tactic} \\
\{ \Gamma \} t \\
\{ \text{strip } \neg t, \Gamma \} F
\end{array}
\]

**Uses**  Proof by contradiction.

**See Also**  strip_tac, \( \neg \_tac \).

**Errors**  
28027 Conclusion of goal does not have type \( \vdash BOOL \)

---

SML
|val CONTR_T : (THM -> TACTIC) -> TACTIC;

**Description**  A form of proof by contradiction as a tactical. \( CONTR\_T \_thmtac \) is a tactic which attempts to solve a goal \((\Gamma, t)\), by applying \( thmtac(\neg t \vdash \neg t) \) to the goal \((\Gamma, F)\).

\[
\begin{array}{c}
\text{Tactic} \\
\{ \Gamma \} t \\
\text{thmtac } (\neg t \vdash \neg t) (\{ \Gamma \} F)
\end{array}
\]

**Uses**  Proof by contradiction in combination with a theorem tactic.

**See Also**  contr_tac, \( \neg \_T \).

**Errors**  
28027 Conclusion of goal does not have type \( \vdash BOOL \)

---

SML
|val conv_tac : CONV -> TACTIC;

**Description**  \( conv\_tac conv \) is a tactic which applies the conversion \( conv \) to the conclusion of a goal, and replaces the conclusion of the goal with the right-hand side of the resulting equational theorem if this is successful:

\[
\begin{array}{c}
\text{Tactic} \\
\{ \Gamma 2 \} t2 \\
\{ \Gamma 2 \} t1
\end{array}
\]

where \( conv t2 = (\Gamma 1 \vdash t2 = t1) \).

**Errors**  
9400 the conversion returned ‘?0’ which is not of the form:
‘... \vdash ?1 \iff ...’
CONV THEN : CONV -> THM _ TACTICAL;

Description  CONV THEN conv thmtac is a theorem tactic which first uses conv to transform the conclusion of a theorem and then acts as thmtac.

Uses  For use in programming theorem tacticals. The function may be partially evaluated with only its conversion, theorem tactic and theorem arguments.

Errors  
9400 the conversion returned ?0 which is not of the form: 
'... |- ?1 <-> ...' 

val discard_tac : 'a -> TACTIC;
val k_id_tac : 'a -> TACTIC;

Description  A tactic that discards its argument, but otherwise has no effect. k_id_tac is an alias for discard_tac.

Uses  Can be used to remove unwanted assumptions: a (POP_ASM_T discard_tac) discards the top-most assumption. This usage of discard_tac may strengthen the goal. ie it may result in unprovable subgoals even when the original goal was provable.

DROP ASMS_T : (THM list -> TACTIC) -> TACTIC;

Description  DROP ASMS_T thmstac is a tactic which applies asm_rule to each assumption of the subgoal, giving a list of theorems, thms say, then removes all the assumptions of the goal and then acts as thmstac thms.

Tactic  
\[ \begin{array}{c}
\{ \Gamma \} t \\
\hline
\text{thmstac (map asm_rule } \Gamma \text{)} (\{\} t)
\end{array} \]

DROP ASMS_T

thmstac

Uses  To use all the assumptions as theorems.

Errors  As for thmstac.

DROP ASM_T : TERM -> (THM -> TACTIC) -> TACTIC;

Description  DROP ASM_T asm thmtac is a tactic which removes asm from the assumption list and then acts as thmtac(asm'=asm).

Tactic  
\[ \begin{array}{c}
\{ \Gamma, asm' \} t \\
\hline
\text{thmstac (asm' \vdash asm)} (\{ \Gamma \} t)
\end{array} \]

DROP ASM_T

\[ ^{r_{\text{asm'}}} \]

thmstac

where asm and asm' are α-convertible.

Uses  To use an assumption as a theorem.

Errors  
9301 the term ?0 is not in the assumption list
## 7.3. General Tactics and Tacticals

### DROP\(_{\text{FILTER-ASMS}}\)\(_T\)

**SML**

\[
\text{val DROP\(_{\text{FILTER-ASMS}}\)\(_T\) : (TERM \rightarrow \text{bool}) \rightarrow (\text{THM list} \rightarrow \text{TACTIC}) \rightarrow \text{TACTIC};}
\]

**Description**

\(\text{DROP\(_{\text{FILTER-ASMS}}\)\(_T\) pred thmstac}\) is a tactic which applies \text{asm\(_\text{rule}\)} to each assumption of the subgoal that satisfies \text{pred}, giving a list of theorems, \text{thms} say, then removes all the selected assumptions of the goal and then acts as \text{thmstac thms}.

**Tactic**

\[
\begin{align*}
\{ \Gamma \} \ t & \\
\text{thmstac (map asm\(_\text{rule}\) (\(\Gamma \cap \text{pred}\)))} & \\
(\{ \Gamma \setminus \text{pred} \} \ t)
\end{align*}
\]

**Uses**

To use all the selected assumptions as theorems.

**Errors**

As for \text{thmstac}.

### DROP\(_{\text{NTH-ASM}}\)\(_T\)

**SML**

\[
\text{val DROP\(_{\text{NTH-ASM}}\)\(_T\) \ : \ int \rightarrow (\text{THM} \rightarrow \text{TACTIC}) \rightarrow \text{TACTIC};}
\]

**Description**

\(\text{DROP\(_{\text{NTH-ASM}}\)\(_T\) i thmstac}\) is a tactic which applies \text{asm\(_\text{rule}\)} to the \(i\)-th assumption of the goal, giving a theorem, \text{thm} say, and then removes \text{asm} from the assumptions and acts as \text{thmstac thm}.

Assumptions are numbered 1, 2..., so that, e.g., \(\text{DROP\(_{\text{NTH-ASM}}\)\(_T\) 1}\) is the same as \(\text{POP\(_{\text{ASM}}\)\(_T\)}\)

**Tactic**

\[
\begin{align*}
\{ a1, \ldots, an \} \ t & \\
\text{thmstac (asm\(_\text{rule}\) [ai])} & \\
(\{ \Gamma \setminus ai \} \ t)
\end{align*}
\]

**Uses**

To use an assumption as a theorem, treating the assumption list as an array.

**Errors**

\(9303\) the index ?0 is out of range
**Description** These two tactics identify an assumption (either by being equal to the term argument, or by index number). They take it from the assumption list, use symmetry upon it to reverse any equations (or bi-implications) (though equations embedded within other equations will not be reversed), and then strip the result into the assumption list. The tactics fail if there are no equations to reverse.

Tactic

\[ \{ \Gamma_1, t[x = y, p = q, ...], \Gamma_2 \} \]

\[ \{ \text{strip } t[y = x, q = p, ...], \Gamma_1, \Gamma_2 \} \]

\[ \text{eq\_sym\_asm\_tac} \]

\[ t[x = y, p = q, ...] \]

Tactic

\[ \{ t_1, ..., t_{n-1}, t[n = x, p = q, ...], t_{n+1}, ..., \} \]

\[ \{ \text{strip } t[n = y, q = p, ...], t_1, ..., t_{n-1}, t_{n+1}, ..., \} \]

\[ \text{eq\_sym\_nth\_asm\_tac} \]

\[ n \]

**Definition**

\[ \text{fun eq\_sym\_asm\_tac asm} = \text{DROP\_ASM\_T asm} \]

\[ (\text{strip\_asm\_tac o conv\_rule(ONCE\_MAP\_C eq\_sym\_conv))} \]

\[ \text{fun eq\_sym\_nth\_asm\_tac n} = \text{DROP\_NTH\_ASM\_T n} \]

\[ (\text{strip\_asm\_tac o conv\_rule(ONCE\_MAP\_C eq\_sym\_conv))} \]

**Example**

<table>
<thead>
<tr>
<th>Assumption</th>
<th>Becomes</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ x = y ]</td>
<td>[ y = x ]</td>
</tr>
<tr>
<td>[ \forall x y . x \iff y ]</td>
<td>[ \forall x y . y \iff x ]</td>
</tr>
<tr>
<td>[ f(x = (p = q)) ]</td>
<td>[ f((p = q) = x) ]</td>
</tr>
<tr>
<td>[ x = y \land p = q ]</td>
<td>[ x = y \iff p = q ]</td>
</tr>
</tbody>
</table>

**Errors**

- 9301 the term ?0 is not in the assumption list
- 9303 the index ?0 is out of range
- 28053 ?0 contains no equations

---

**SML**

\[ \text{val EVERY\_TTCL : THM\_TACTICAL list -> THM\_TACTICAL;} \]

**Description** EVERY\_TTCL is a theorem tactical combinator.

\[ \text{EVERY\_TTCL [ttc1, ttc2, ...] = ttc1 THEN\_TTCL ttc2 THEN\_TTCL ...} \]

\[ \text{EVERY\_TTCL [ ] acts as ID\_THEN.} \]

**Uses** For use in programming theorem tactics.
7.3. General Tactics and Tacticals

### EVERY

```sml
val EVERY : TACTIC list -> TACTIC;
val EVERY_T : TACTIC list -> TACTIC;
```

**Description**
EVERY applies the head of `tlist` to its subgoal, and recursively applies the tail of `tlist` to each resulting subgoal. EVERY_T is an alias for EVERY.

**Example**
```
EVERY [\_tac, \_tac, \_tac]
  is equivalent to
\_tac THEN \_tac THEN \_tac
```

**Errors**
As for the tactics in the list.

### fail_tac

```sml
val fail_tac : TACTIC;
```

**Description**
A tactic that always fails. This is the identity for the tactical ORELSE_T.

**Uses**
For constructing larger tactics.

**Errors**
9201 failed as requested

### FAIL_THEN

```sml
val FAIL_THEN : THM_TACTICAL;
```

**Description**
This is a theorem tactical which always fails at the point it receives its theorem (having already been given a theorem tactic). It acts as the identity for the theorem tactical ORELSE_TTCL.

**Uses**
For use in programming theorem tacticals.

**Errors**
9401 failed as requested

### fail_with_tac

```sml
val fail_with_tac : string -> int -> (unit -> string) list -> TACTIC;
```

**Description**
fail_with_tac area msg inserts is a tactic that always fails, reporting an error message via the call fail area msg inserts.

**Uses**
For constructing larger tactics.

**See Also**
fail

**Errors**
As determined by the arguments.

### FAIL_WITH_THEN

```sml
val FAIL_WITH_THEN : string -> int -> (unit -> string) list -> THM_TACTICAL;
```

**Description**
FAIL_WITH_THEN area msg inserts is a theorem tactical that always fails when given its theorem (having already been given a theorem tactic), reporting an error message via the call fail area msg inserts.

**Uses**
For constructing larger theorem tacticals.

**See Also**
fail

**Errors**
As determined by the arguments.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
val FIRST_TTCL : THM_TACTICAL list -> THM_TACTICAL;

Description  FIRST_TTCL is a theorem tactical combinator.  FIRST_TTCL [] fails on being applied to its theorem tactic and then theorem.

FIRST_TTCL [ttcl1, ttcl2, ...] =
  ttcl1 ORELSE_TTCL ttcl2 ORELSE_TTCL ...

Uses  For use in programming theorem tacticals.

Errors  [9402  the list of theorem tactics is empty]

val FIRST_T : TACTIC list -> TACTIC;
val FIRST : TACTIC list -> TACTIC;

Description  FIRST_T tlist is a tactic that attempts to apply each tactics in tlist until one succeeds, or all fail.  The first successful application will be the result of the tactic, and it fails if all the attempts fail.  FIRST is an alias for FIRST_T.  FIRST [] fails on being applied to any goal.

Errors  [9105  the list of tactics is empty]

Also as the failure of last member of a non-empty list.
These are tactics which use theorems whose conclusions are implications, or from which implications can be derived using the canonicalisation function `fc_canon`, q.v., to reason forwards from the assumptions of a goal. (The names with `fc` are aliases for the corresponding ones with `forward_chain`.)

The basic step is to take a theorem of the form $\Gamma \vdash t_1 \Rightarrow t_2$ and an assumption of the form $t_1'$ where $t_1'$ is a substitution instance of $t_1$ and to deduce the corresponding instance of $t_2'$. The new theorem, $\Delta \vdash t_2'$ say, may then be stripped into the assumptions.

In the case of `fc_tac` the implicative theorem is always derived from the list of theorems given as an argument. In the case of `asm_fc_tac` the assumptions are also used. In all of the tactics the rule `fc_canon` is used to derive an implicative canonical form from the candidate implicative theorems. Normally combination of an implicative theorem and an assumption is then tried in turn and all resulting theorems are stripped into the assumptions of the goal. However, if the chaining results contain a theorem whose conclusion is $\{F\}$ then the first such found will be stripped into the assumptions, and all other theorems discarded.

If one of the implications has the form $t_1 \Rightarrow t_2 \Rightarrow t_3$ or $t_1 \land t_2 \Rightarrow t_3$ and if assumptions matching $t_1$ and $t_2$ are available, `fc_tac` or `asm_fc_tac` will derive an intermediate implication $t_2 \Rightarrow t_3$ and `asm_fc_tac` could then be used to derive $t_3$. The variants with `all_` may be used to derive $t_3$ directly without generating any intermediate implications in the assumptions. They work like the corresponding tactic without `all_` but any theorems which are derived which are themselves implications are not stripped into the assumptions but instead are used recursively to derive further theorems. When no new implications are derivable all of the non-implicative theorems derived during the process are stripped into the assumptions.

Note that the use of `fc_canon` implies that conversions from the proof context are applied to generate implications. E.g., in an appropriate proof-context covering set theory, $a \subseteq b$ might be treated as the implication $\forall x \bullet x \in a \Rightarrow x \in b$. Also variables which appear free in a theorem are not considered as candidates for instantiation (in order to give some control over the number of results generated). The tacticals, `FC_T1` and `ASM_FC_T1` may be used to avoid the use of `fc_canon`.

For example, the tactic:

```
[asm_fc_tac[] THEN asm_fc_tac[]]
```

will prove the goal:

\[ \{p \cdot x, \forall x \bullet p \cdot x \Rightarrow q \cdot x, \forall x \bullet q \cdot x \Rightarrow r \cdot x\} r \cdot x. \]

See Also `bc_tac`, `FC_T`, `ASM_FC_T`, `FC_T1`, `ASM_FC_T1`. 

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
<table>
<thead>
<tr>
<th>val FORWARD_CHAIN_T :</th>
</tr>
</thead>
<tbody>
<tr>
<td>(THM list -&gt; TACTIC) -- THM list -&gt; TACTIC;</td>
</tr>
<tr>
<td>val FC_T :</td>
</tr>
<tr>
<td>(THM list -&gt; TACTIC) -- THM list -&gt; TACTIC;</td>
</tr>
<tr>
<td>val ALL_FORWARD_CHAIN_T :</td>
</tr>
<tr>
<td>(THM list -&gt; TACTIC) -- THM list -&gt; TACTIC;</td>
</tr>
<tr>
<td>val ALL_FC_T :</td>
</tr>
<tr>
<td>(THM list -&gt; TACTIC) -- THM list -&gt; TACTIC;</td>
</tr>
<tr>
<td>val ASM_FORWARD_CHAIN_T :</td>
</tr>
<tr>
<td>(THM list -&gt; TACTIC) -- THM list -&gt; TACTIC;</td>
</tr>
<tr>
<td>val ASM_FC_T :</td>
</tr>
<tr>
<td>(THM list -&gt; TACTIC) -- THM list -&gt; TACTIC;</td>
</tr>
<tr>
<td>val ALL_ASM_FORWARD_CHAIN_T :</td>
</tr>
<tr>
<td>(THM list -&gt; TACTIC) -- THM list -&gt; TACTIC;</td>
</tr>
<tr>
<td>val ALL_ASM_FC_T :</td>
</tr>
<tr>
<td>(THM list -&gt; TACTIC) -- THM list -&gt; TACTIC;</td>
</tr>
</tbody>
</table>

**Description**
These are tacticals which use theorems whose conclusions are implications, or from which implications can be derived, to reason forwards from the assumptions of a goal. (The tacticals with \texttt{FC} are aliases for the corresponding ones with \texttt{FORWARD_CHAIN}.)

The description of \texttt{fc_tac} should be consulted for the basic forward chaining algorithms used. The significance of the final argument and of the presence or absence of \texttt{ASM} and \texttt{ALL} in the name is exactly as for \texttt{fc_tac} and its relatives.

The tacticals allow variation of the tactic generating function used to process the theorems derived by the forward inference. The tactic generating function to be used is given as the first argument.

**Examples**
\texttt{fc_tac} is the same as: \texttt{FC_T (MAP_EVERY strip_asm_tac)}.

To rewrite the goal with the results of the forward inference one could use \texttt{FC_T rewrite_tac}.

**See Also** \texttt{fc_tac, asm_fc_tac, be_tac, FC_T1}. 
7.3. General Tactics and Tacticals

These are tacticals which use theorems whose conclusions are implications, or from which implications can be derived, to reason forwards from the assumptions of a goal. (The tacticals with FC are aliases for the corresponding ones with FORWARD_CHAIN.)

The description of fc_tac should be consulted for the basic forward chaining algorithms used. The significance of the final argument and of the presence or absence of ASM and ALL in the name is exactly as for fc_tac and its relatives.

The tacticals allow variation of the canonicalisation function used to obtain implications from the argument theorems and of the tactic generating function used to process the theorems derived by the forward inference. The canonicalisation function to use is the first argument and the tactic generating function is the second. (Related tacticals with names ending in T rather than T1 are also available for the simpler case when wants to use the same canonicalisation function as fc_tac and just to vary the tactic generating function.)

Examples If the theorem argument comprises only implications which are to be used without canonicalisation, one might use: FC_T1 id_canon (MAP_EVERY strip_asm_tac).

If one has an instance of t1 as an assumption and one wishes to use the bi-implication in a theorem of the form ⊢ t1 ⇒ (t2 ⇔ t3) for rewriting, one might use FC_T1 id_canon rewrite_tac.

See Also fc_tac, asm_fc_tac, be_tac, FC_T.

SML

```snippet
val FORWARD_CHAIN_T1 : (THM --> THM list) --> (THM list --> TACTIC) --> THM list --> TACTIC;
val FC_T1 : (THM --> THM list) --> (THM list --> TACTIC) --> THM list --> TACTIC;
val ALL_FORWARD_CHAIN_T1 : (THM --> THM list) --> (THM list --> TACTIC) --> THM list --> TACTIC;
val ALL_FC_T1 : (THM --> THM list) --> (THM list --> TACTIC) --> THM list --> TACTIC;
val ASM_FORWARD_CHAIN_T1 : (THM --> THM list) --> (THM list --> TACTIC) --> THM list --> TACTIC;
val ASM_FC_T1 : (THM --> THM list) --> (THM list --> TACTIC) --> THM list --> TACTIC;
val ALL_ASM_FORWARD_CHAIN_T1 : (THM --> THM list) --> (THM list --> TACTIC) --> THM list --> TACTIC;
val ALL_ASM_FC_T1 : (THM --> THM list) --> (THM list --> TACTIC) --> THM list --> TACTIC;
```

Description These are tacticals which use theorems whose conclusions are implications, or from which implications can be derived, to reason forwards from the assumptions of a goal. (The tacticals with FC are aliases for the corresponding ones with FORWARD_CHAIN.)

The description of fc_tac should be consulted for the basic forward chaining algorithms used. The significance of the final argument and of the presence or absence of ASM and ALL in the name is exactly as for fc_tac and its relatives.

The tacticals allow variation of the canonicalisation function used to obtain implications from the argument theorems and of the tactic generating function used to process the theorems derived by the forward inference. The canonicalisation function to use is the first argument and the tactic generating function is the second. (Related tacticals with names ending in T rather than T1 are also available for the simpler case when wants to use the same canonicalisation function as fc_tac and just to vary the tactic generating function.)

Examples If the theorem argument comprises only implications which are to be used without canonicalisation, one might use: FC_T1 id_canon (MAP_EVERY strip_asm_tac).

If one has an instance of t1 as an assumption and one wishes to use the bi-implication in a theorem of the form ⊢ t1 ⇒ (t2 ⇔ t3) for rewriting, one might use FC_T1 id_canon rewrite_tac.

See Also fc_tac, asm_fc_tac, be_tac, FC_T.
val GEN_INDUCTION_T : THM -> (THM -> TACTIC) -> TERM -> TACTIC;
val gen_induction_tac : THM -> TERM -> TACTIC;

Description These give general means for constructing an induction tactic from an induction
principle formulated as a theorem. The term argument is the induction variable, which must be
free in the conclusion of the goal to which the tactic is applied but not in the assumptions.

GEN_INDUCTION_T causes any inductive hypotheses (see below) to be passed to a tactic
generating function.

gen_induction_tac thm is the same as GEN_INDUCTION_T thm strip_asm_tac.

The discussion below is for the tactic computed by the call GEN_INDUCTION_T thm ttac y
applied to a goal with conclusion t.

The induction principle, thm has the form:

\[ \forall p \cdot a \Rightarrow \forall x \cdot p x \]

E.g. the usual principle of induction for the natural numbers:

\[ \forall p \cdot p 0 \land (\forall n \cdot p n \Rightarrow p (n + 1)) \Rightarrow (\forall n \cdot p n) \]

The induction tactic takes the following steps:

1. Use \( \forall \)-elimination on thm, (with the term \( \lambda y \cdot t \)) and \( \beta \)-reduction to give an implicative
   theorem, \( \vdash a' \Rightarrow t \) and use it to reduce the goal to a subgoal with conclusion \( a' \).

2. Repeatedly apply \( \land \_ \_ \_ \_ tac \) and then repeatedly apply \( \forall \_ \_ \_ \_ tac \).

3. To any of the resulting subgoals whose principal connective corresponds to an an implication
   in thm apply \( \Rightarrow \_ T ttac \). E.g., with the usual principle of induction for the natural numbers
   as formulated above \( \Rightarrow \_ T ttac is applied in the inductive step but not in the base case, even
   if the conclusion of the goal is an implication.

The tactic also renames bound variables so that names which begin with the name of the variable
in the theorem now begin with the name of the induction variable passed to the tactic.

Errors

29021 '0 does not have the form '‘\鞭 a \cdot p a \Rightarrow \forall x \cdot p x''
29023 '0 does not have the form '‘\vdash a \Rightarrow t''
29024 '0 is not an instance of '‘?1''
29025 '0 is not a variable
29026 '0 appears free in the assumptions of the goal
29026 '0 does not appear free in the conclusion of the goal
7.3. General Tactics and Tacticals

SML

val GEN_INDUCTION_T1 : THM -> (THM -> TACTIC) -> TACTIC;
val gen_induction_tac1 : THM -> TACTIC;

**Description**  These give a means for constructing an induction tactic from an induction principle formulated as a theorem, in cases where the induction variable can be inferred from the form of the theorem and the goal. They are in other respects very like `GEN_INDUCTION_T` and `gen_induction_tac thm`, q.v.

The induction theorem must be a theorem of the form:

\[ \forall p \cdot a \Rightarrow \forall x \cdot t[p/x/b] \]

Where \( t \) contains at least one occurrence of \( x \). For example,

\[ \forall p \cdot \{ \forall a \cdot a \in \text{Finite} \land p a \land \neg x \in a \Rightarrow p (\{x\} \cup a) \} \Rightarrow (\forall a \cdot a \in \text{Finite} \Rightarrow p a) \]

(for which \( t \) is \( a \in \text{Finite} \Rightarrow b \)).

The induction tactic matches the conclusion, \( c \), of the goal with \( t \), uses the result to derive a theorem of the form \( a' \Rightarrow c \) and then proceeds exactly like the corresponding induction tactic produced by `GEN_INDUCTION_T` and `gen_induction_tac thm` q.v.

**Errors**

- 29007 ?0 does not have the form `\forall p : \tau \rightarrow BOOL \cdot a \Rightarrow \forall x \cdot t[p/x/b]`
  (where \( \lceil x \rceil \) must also appear in \( \lceil t \rceil \) other than as an argument of \( \lceil p \rceil \))
- 29009 The conclusion of the goal cannot be rewritten in the form \( ?0 \)
- 29014 The term ?0 which matches the induction variable is not a variable

SML

val GET_ASMS_T : (THM list -> TACTIC) -> TACTIC;

**Description**  GET_ASMS_T thmstac is a tactic which applies `asm_rule` to each assumption of the goal, giving a list of theorems, `thms` say, and then acts as `thmstac thms`.

\[
\begin{array}{c}
\{ \Gamma \} \ t \\
\hline
\text{thmstac (map asm_rule } \Gamma) \\
\{ a_1, \ldots, a_n \} \ t
\end{array}
\]

\[ \text{GET ASMS T} \text{thmstac} \]

**Uses**  To use all the assumptions as theorems.

**Errors**  As for `thmstac`.

SML

val GET_ASM_T : TERM -> (THM -> TACTIC) -> TACTIC;

**Description**  GET_ASM_T asm thmtac is a tactic which checks that `asm` is in the assumption list and then acts as `thmtac(asm\dash asm)`.

\[
\begin{array}{c}
\{ \Gamma, asm' \} \ t \\
\hline
\text{thmtac (asm \vdash asm) (\{ } \Gamma, asm' \} \ t)
\end{array}
\]

\[ \text{GET ASM T} \ \Gamma \text{asm} \text{thmtac} \]

where `asm` and `asm'` are \( \alpha \)-convertible.

**Uses**  To use an assumption as a theorem

**Errors**  9301 the term \( ?0 \) is not in the assumption list

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
**SML**

```sml
val GET_FILTER_ASMS_T : (TERM -> bool) ->
  (THM list -> TACTIC) -> TACTIC;
```

**Description**  
`GET_FILTER_ASMS_T pred thmstac` is a tactic which applies `asm_rule` to each assumption of the subgoal that satisfies `pred`, giving a list of theorems, `thms` say and then acts as `thmstac thms`.

**Tactic**

\[
\begin{align*}
\{ \Gamma \} \ t \\
\text{thmstac (map asm_rule (} \{ \Gamma \} \cap \text{ pred)}\}
\end{align*}
\]

\[
\text{GET_FILTER_ASMS_T pred thmstac}
\]

**Uses**  
To use all the selected assumptions as theorems.

**Errors**  
As for `thmstac`.

---

**SML**

```sml
val GET_NTH_ASM_T : int -> (THM -> TACTIC) -> TACTIC;
```

**Description**  
`GET_NTH_ASM_T i thmtac` is a tactic which applies `asm_rule` to the `i`-th assumption of the goal, giving a theorem, `thm` say, and then acts as `thmtac thm`.

Assumptions are numbered `1, 2, ...`, so that, e.g., `GET_NTH_ASM_T 1` is the same as `TOP_ASM_T`

**Tactic**

\[
\begin{align*}
\{ \ a1, ..., an \} \ t \\
\text{thmtac (asm_rule [ai]} \}) \ (\{ \Gamma \} \ t)
\end{align*}
\]

\[
\text{GET_NTH_ASM_T i thmtac}
\]

**Uses**  
To use an assumption as a theorem, treating the assumption list as an array.

**Errors**  
`9303` the index `?0` is out of range

---

**SML**

```sml
val id_tac : TACTIC;
```

**Description**  
A tactic that always succeeds, having no effect. This is the identity for the tactical `THEN_T T`.

**Tactic**

\[
\begin{align*}
\{ \Gamma \} \ t \\
\{ \Gamma \} \ t
\end{align*}
\]

\[
\text{id_tac}
\]

**Uses**  
For constructing larger tactics.

---

**SML**

```sml
val ID_THEN : THM_TACTICAL;
```

**Description**  
This is the identity for the theorem tactical combinator `THEN_TTCL`.

\[
(ID_THEN) \ thmtac = \ thmtac
\]

**Uses**  
For use in programming theorem tacticals.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
7.3. General Tactics and Tacticals

SML

val IF_T2 : (THM -> TACTIC) -> (THM -> TACTIC) -> TACTIC;

Description Reduce a conditional by applying tactic generating functions to the two cases for the selector.

\[
\begin{array}{c}
\mathrm{Tactic} \\
\{ \Gamma \} \quad \text{if} \quad a \quad \text{then} \quad tt \quad \text{else} \quad et \\
\hline
\begin{array}{c}
\text{ttac1} \{ a, \Gamma \} \vdash tt \\
\text{ttac2} \{ \neg a, \Gamma \} \vdash et
\end{array}
\end{array}
\]

IF_T2 

ttac1 ttac2

See Also ⇔_T, STRIP_CONCL_T

Errors

28071 Goal is not of the form: \{ \Gamma \} \quad \text{if} \quad a \quad \text{then} \quad tt \quad \text{else} \quad et

SML

val if_tac : TACTIC;

Description Reduce a conditional subgoal by performing a case split on the selector.

\[
\begin{array}{c}
\mathrm{Tactic} \\
\{ \Gamma \} \quad \text{if} \quad a \quad \text{then} \quad tt \quad \text{else} \quad et \\
\hline
\{ \text{strip} \ a, \Gamma \} \quad \text{tt} \\
\{ \text{strip} \ \neg a, \Gamma \} \quad \text{et}
\end{array}
\]

if_tac

See Also strip_tac

Errors

28071 Goal is not of the form: \{ \Gamma \} \quad \text{if} \quad a \quad \text{then} \quad tt \quad \text{else} \quad et

SML

val IF_THEN2 : (THM -> TACTIC) -> (THM -> TACTIC) -> (THM -> TACTIC);

Description A theorem tactical to apply given theorem tactics to the result of eliminating the conditional from a theorem with a conditional as its conclusion.

\[\text{IF_THEN} \quad \text{ttac} (\Gamma \vdash \text{if} \ a \ \text{then} \ tt \ \text{else} \ et) = \text{ttac1} (a, \Gamma \vdash tt) \ \text{THEN} \ \text{ttac2} (\neg a, \Gamma \vdash et)\]

The function is partially evaluated with only the theorem tactic and theorem arguments.

See Also IF_THEN, STRIP_THM_THEN

Errors

7012 ?0 is not of the form: \(\Gamma \vdash \text{if} \ tc \ \text{then} \ tt \ \text{else} \ te\)

SML

val IF_THEN : (THM -> TACTIC) -> (THM -> TACTIC);

Description A theorem tactical to apply a given theorem tactic to the result of eliminating the conditional from a theorem with a conditional as its conclusion.

\[\text{IF_THEN} \quad \text{ttac} (\Gamma \vdash \text{if} \ a \ \text{then} \ tt \ \text{else} \ et) = \text{ttac} (\Gamma \vdash a \Rightarrow tt) \ \text{THEN} \ \text{ttac} (\Gamma \vdash \neg a \Rightarrow et)\]

The function is partially evaluated with only the theorem tactic and theorem arguments.

See Also IF_THEN2, STRIP_THM_THEN

Errors

7012 ?0 is not of the form: \(\Gamma \vdash \text{if} \ tc \ \text{then} \ tt \ \text{else} \ te\)
val IF_T : (THM -> TACTIC) -> TACTIC;

Description  Reduce a conditional by applying a tactic generating function to the two cases for the selector.

\[
\begin{align*}
&\text{Tactic} \quad \{ \Gamma \} \text{ if } a \text{ then } tt \text{ else } et \quad \text{IF}_T \\
&ttac\{ a, \Gamma \} \vdash \text{tt} ; ttac\{ \neg a, \Gamma \} \vdash \text{et} \\
&\text{ttac}
\end{align*}
\]

See Also  IF_T2, STRIP_CONCL_T

Errors  
28071  Goal is not of the form: \{ \Gamma \} if \ a \ then \ tt \ else \ et

val intro∀_tac : (TERM * TERM) -> TACTIC;
val intro∀_tac1 : TERM -> TACTIC;

Description  Sometimes it is helpful to prove a goal by proving a more general conjecture has the goal as a special case. \text{intro}_∀_tac \ introduces a universal quantifier into the conclusion of a goal in order to do this.

\[
\begin{align*}
&\text{Tactic} \quad \{ \Gamma \} \ t[t1/x] \\
&\{ \Gamma \} \forall x \bullet t
\end{align*}
\]

\text{intro}_∀_tac \ (t1, x)

where \( t \) is a term in which \( x \) appears free and where either \( t1 \) the same as \( x \) or \( x \) does not appear free in the conclusion, \( t[t1/x] \), of the original goal.

Note that \( t1 \) need not be a variable, e.g.,

\[
\begin{align*}
&\text{Example} \quad \{ \Gamma \} \ 1 + 2 > 0 \Rightarrow \neg 1 + 2 = 0 \\
&\{ \Gamma \} \forall i \bullet i > 0 \Rightarrow \neg i = 0 \quad \text{intro}_∀_tac \ (\gamma 1 + 2 \gamma, \gamma i; \mathbb{N} \gamma)
\end{align*}
\]

\text{intro}_∀_tac1 is for use in the common case where one simply wants to take replace the goal by its universal closure over some variable. \text{intro}_∀_tac1 \ \gamma x \gamma is equivalent to \text{intro}_∀_tac \ (\gamma x \gamma, \gamma x \gamma).

N.B. these tactics may strengthen the goal, i.e. they may result in unprovable subgoals even when the original goal was provable.

Uses  The most common use is in preparation for an inductive proof when it is necessary to generalise the conclusion in order to give stronger assumptions in the inductive step or steps.

See Also  \_reorder\_conv

Errors  
28082  \?0 does not appear free in the goal
28083  \?0 appears free in the goal and is not the same as \?1
7.3. General Tactics and Tacticals

SML
defval i_contrad : TACTIC;

Description Prove a goal by showing that the assumptions are contradictory, in an intuitionistic manner.

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

Tactic  
\[
\begin{array}{c}
\{ \Gamma \} t \\
\{ \Gamma \} F
\end{array}
\]  
i_contrad

Uses If a proof is to be carried out by showing the assumptions inconsistent, then the conclusion of the subgoal is irrelevant and may be removed.

SML
defval lemma_tac : TERM -> TACTIC;

Description Introduce a lemma (the term argument) to be proved, and then added as an assumption.

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

Tactic  
\[
\begin{array}{c}
\{ \Gamma \} t2 \\
\{ \Gamma \} t1; \\
\{ \text{strip } t1, \Gamma \} t2
\end{array}
\]  
lemma_tac

See Also LEMMA_T

Errors
9603 the term ?0 is not boolean

SML
defval LEMMA_T : TERM -> (THM -> TACTIC) -> TACTIC;

Description LEMMA_T newsg thmtac is a tactic which sets newsg as a new subgoal and applies thmtac(newsg |= newsg) to the original goal.

Tactic  
\[
\begin{array}{c}
\{ \Gamma \} t \\
\{ \Gamma \} \text{newsg;} \\
\text{thmtac}(\text{newsg} \vdash \text{newsg}) (\{ \Gamma \} t)
\end{array}
\]  
LEMA_T

newsg thmtac

Uses For use in tactic programming and in interactive use where lemma_tac is not appropriate.

Errors
9603 the term ?0 is not boolean

See Also lemma_tac.
**SML**

```
val list_asm_ante_tac : TERM list -> TACTIC;
```

**Description**  Repeatedly apply `asm_ante_tac`.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ Γ, t₁, ..., tₙ } t</code></td>
<td><code>list_asm_ante_tac { Γ } t₁ ⇒ ... ⇒ tₙ ⇒ t</code></td>
</tr>
</tbody>
</table>

α-equivalent assumptions will only appear once in the resulting goal, in their rightmost position, (which also means that duplicates in the list are ignored).

**See Also**  `asm_ante_tac`, all `asm_ante_tac`

**Errors**

- 28052  Term ?0 is not in the assumptions
- 28055  The conclusion or an assumption of goal does not have type `⌜:BOOL⌝`

---

**SML**

```
val LIST_DROP_ASM_T : TERM list -> (THM list -> TACTIC) -> TACTIC;
```

**Description**  `LIST_DROP_ASM_T [asm_1, asm_2, ...] thmtac` is a tactic which removes the `asm_1, asm_2, ...` from the assumption list and then acts as

```
thmtac[(asm_1 ⊢ asm_1), (asm_2 ⊢ asm_2), ...]
```

<table>
<thead>
<tr>
<th>Tactic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ Γ, asm_i', ..., } t</code></td>
<td><code>LIST_DROP_ASM_T (asm_i ⊢ asm_i') thmtac</code></td>
</tr>
</tbody>
</table>

where `asm_i` and `asm_i'` are α-converible.

**Uses**  To use assumptions as theorems

**Errors**

- 9301  the term ?0 is not in the assumption list

---

**SML**

```
val LIST_DROP_NTH_ASM_T : int list ->
(THM list -> TACTIC) -> TACTIC;
```

**Description**  `LIST_DROP_NTH_ASM_T [i, j, ...] thmtac` is a tactic which applies `asm_rule` to the `i-th, j-th, etc` assumptions of the goal, giving theorems, `thm_i, thm_j, etc`, say, and then removes the `asm_i, asm_j` from the assumptions and acts as `thmtac [thm_i, thm_j, ...]`.

<table>
<thead>
<tr>
<th>Tactic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ a₁, ..., an } t</code></td>
<td><code>DROP_NTH_ASM_T[i,j,...] thmtac</code></td>
</tr>
</tbody>
</table>

| (asm_rule [ai]), (asm_rule [aj]),... | ( { Γ \ { ai, aj, ...} } t ) |

**Uses**  To use assumptions as theorems, treating the assumption list as an array.

**Errors**

- 9303  the index ?0 is out of range
7.3. General Tactics and Tacticals

SML

```sml
val LIST_GET_ASM_T : TERM list -> (THM list -> TACTIC) -> TACTIC;
```

**Description**  
`LIST_GET_ASM_T [asm_1, asm_2, ...] thmtac` is a tactic which checks that all the `asm_1, asm_2, ...` are in the assumption list and then acts as

\[
\text{thmtac}[(\text{asm}_1 \vdash \text{asm}_1), (\text{asm}_2 \vdash \text{asm}_2), ...]
\]

\[
\text{Tactic: }
\begin{array}{c}
\{ \Gamma, \text{asm}_1', ..., \} \ t
\rightarrow \\
\text{thmtac} [(\text{asm}_1 \vdash \text{asm}_1), ...] \\
\{ \{ \Gamma, \text{asm}_1', ..., \} \ t
\end{array}
\]

LIST_GET_ASM_T

\[
\text{thmtac}
\]

where `asm_i` and `asm_i'` are α-convertible.

**Uses**  
To use a list of assumptions as theorems

**Errors**  
9301 the term ?0 is not in the assumption list

---

SML

```sml
val LIST_GET_NTH_ASM_T : int list -> (THM list -> TACTIC) -> TACTIC;
```

**Description**  
`LIST_GET_NTH_ASM_T [i, j, ...] thmtac` is a tactic which applies `asm_rule` to the `i`-th, `j`-th, etc., assumption of the goal, giving theorems, `thm_i, thm_j`, etc, say, and then acts as `thmtac [thm_i, thm_j, ...]`.

\[
\text{Tactic: }
\begin{array}{c}
\{ a1, ..., an \} \ t
\rightarrow \\
\text{thmtac} [(\text{asm_rule } [a1],...], \{ \{ \Gamma \} \ t
\end{array}
\]

LIST_GET_NTH_ASM_T

\[
\text{thmtac}
\]

**Uses**  
To use assumptions as theorems, treating the assumption list as an array.

**Errors**  
9303 the index ?0 is out of range

---

SML

```sml
val list_simple_∃_tac : TERM list -> TACTIC ;
```

**Description**  
Provide a list of witnesses for an iterated existential subgoal.

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

\[
\text{Tactic: }
\begin{array}{c}
\{ \Gamma \} \ \exists x_1,x_2,... \bullet t2[x_1,x_2,...] \\
\rightarrow \\
\{ \Gamma \} \ t2[t1',t2',...]
\end{array}
\]

list_simple_∃_tac

\[
\text{[t1',t2',...]
\]

where `t1', t2', ...` are `t1, t2, ...`, type instantiated to have the same type as `x_1, x_2, ...`

**See Also**  
`simple_∃_tac`

**Errors**  
29008 Cannot match witness ?0 to variable ?1
29015 The list of witnesses is longer than the list of existentially quantified variables in ?0
29016 The list of witnesses is empty
29017 Goal is not of the form: \{ \Gamma \} \ \exists x \bullet t2[x]
val list_swap_asm_concl_tac : TERM list -> TACTIC;
val list_swap_nth_asm_concl_tac : int list -> TACTIC;

Description  Strip the negation of current goal into the assumption list and make some assumptions, suitably negated, into a disjunction forming the current goal. If the list is empty then the conclusion will become '⌜F⌝'.

Tactic  
\[
\frac{\{ \Gamma \} \ t2}{\Gamma \vdash \neg t2, \Gamma - \{ \neg t1, \ldots, \neg tn \}} \quad \text{list_swap_asm_concl_tac}
\]
\[
\frac{\{ \Gamma \} \ t}{\Gamma \vdash \neg t, \Gamma - \{ \neg tp, \ldots, \neg tq \}} \quad \text{list_swap_nth_asm_concl_tac}
\]

If any assumption is a negated term then the double negation will be eliminated.

See Also  Other swap and SWAP functions.

Errors  
9303  the index \(0\) is out of range
28052  Term \(0\) is not in the assumptions

val LIST_SWAP_ASM_CONCL_T : TERM list -> (THM -> TACTIC) -> TACTIC;
val LIST_SWAP_NTH_ASM_CONCL_T : int list -> (THM -> TACTIC) -> TACTIC;

Description  Process the negation of current goal with the supplied theorem tactic and make some assumptions, suitably negated, into a disjunction forming the current goal.

Tactic  
\[
\frac{\{ \Gamma \} \ t}{\text{ttac(asm_rule } \neg t^\gamma \text{) \quad LIST_SWAP_ASM_CONCL_T \quad \{ \neg tp^\gamma, \ldots, \neg tq^\gamma \} \quad \text{ttac}}
\]
\[
\frac{\{ \Gamma \} \ t}{\text{ttac(asm_rule } \neg t^\gamma \text) \quad \{ \neg tp^\gamma, \ldots, \neg tq^\gamma \} \quad \text{ttac} \quad \text{LIST_SWAP_NTH_ASM_CONCL_T}}
\]

If an assumption is a negated term then the double negation will be eliminated. If the list is empty then the conclusion (before applying the tactic argument) will become '⌜F⌝'.

See Also  Other swap and SWAP functions.

Errors  
9303  the index \(0\) is out of range
28052  Term \(0\) is not in the assumptions
28027  Conclusion of goal does not have type '⌜BOOL⌝'
7.3. General Tactics and Tacticals

**SML**

```sml
val MAP_EVERY_T = (\a -> TACTIC) -> \a list -> TACTIC;
val MAP_EVERY = (\a -> TACTIC) -> \a list -> TACTIC;
```

**Description**  
*MAP_EVERY_T* mapf alist maps *mapf* over alist, and then applies the resulting list of tactics to the goal in sequence (in the same manner as *EVERY*, q.v.). *MAP_EVERY* is an alias for *MAP_EVERY_T*.

**Errors**  
As the individual items generated by mapping the tactic over the list.

**SML**

```sml
val MAP_FIRST_T = (\a -> TACTIC) -> \a list -> TACTIC;
val MAP_FIRST = (\a -> TACTIC) -> \a list -> TACTIC;
```

**Description**  
*MAP_FIRST_T* mapf alist maps *mapf* over alist, and then attempts to apply each resulting tactic in order, until one succeeds or all fail (in the same manner as *FIRST*, q.v.). *MAP_FIRST* is an alias for *MAP_FIRST_T*.

**Errors**  
As the last tactic.

**SML**

```sml
val map_shape = (\'(\a list -> \b) * int\) list -> \'a list -> \'b list
```

**Description**  
*map_shape* is a means of composing functions on lists. It is intended for composing the proofs produced by tactics in tacticals such as *THEN*. Its effect is as follows:

\[
\text{map_shape } [(f1, n1), (f2, n2), ...] \ [a1, a2, ...] \\
= \ [f1[a1, ..., a(n1)], f2[a(n1+1), ..., a(n1+n2)], ...]
\]

where, if there are not enough \(a_i\), then unused \(f_j\) are ignored and the last \(f_j\) to be used may receive less than \(n_j\) elements in its argument. (This case is not expected to occur in the application of *map_shape* in tactic programming.)

**Uses**  
Specialised low-level tactic programming.

**SML**

```sml
val ORELSE_TTCL = (THM_TACTICAL * THM_TACTICAL) -> THM_TACTICAL;
```

**Description**  
*ORELSE_TTCL* is a theorem tactical combinator. It is an infix operator.  
\((tcl1 ORELSE_TTCL tcl2)\)th is *tcl1* th unless evaluation of *tcl1* th fails, in which case it is *tcl2* th.

**Uses**  
For use in programming theorem tacticals.

**SML**

```sml
val ORELSE_T = (TACTIC * TACTIC) -> TACTIC;
val ORELSE = (TACTIC * TACTIC) -> TACTIC;
```

**Description**  
*ORELSE_T* is a tactical used as an infix operator. *tac1* ORELSE_T *tac2* is a tactic which behaves as *tac1* unless application of *tac1* fails, in which case it behaves as *tac2*. *ORELSE* is an alias for *ORELSE_T*.

**See Also**  
*LIST_ORELSE_T*

**Errors**  
As the failure of *tac2*.
|val pair_rw_canon : CANON;

**Description**  
This is the rewrite canonicalisation function for the theory of pairs, defined as

```sml
val pair_rw_canon =  
  REWRITE_CAN  
  (REPEAT_CAN(FIRST_CAN [  
    ∀_rewrite_canon,  
    ∧_rewrite_canon,  
    ¬_rewrite_canon,  
    ∃_rewrite_canon,  
    f_rewrite_canon,  
    ⟷_t_rewrite_canon]));
```

This is the repeated application of the first applicable operation in the following list:

1. stripping universal quantifiers (paired or simple);
2. dividing conjunctive theorems into their conjuncts;
3. changing \( \vdash \neg(t_1 \lor t_2) \) to \( \neg t_1 \land \neg t_2 \);
4. changing \( \vdash \exists vs \bullet t \) to \( \forall vs \bullet \neg t \);
5. changing \( \vdash \neg t \) to \( t \equiv F \);
6. changing \( \vdash \neg \neg t \) to \( t \equiv F \);
7. if none of the above apply, changing \( \vdash t \) to \( \vdash t \equiv T \).

Finally, after all this canonicalisation we then universally quantify the resulting theorems in all free variables other than those that were free in the original.

|val POP_ASM.T : (THM -> TACTIC) -> TACTIC;

**Description**  
POP_ASM.T thmtac is a tactic which removes the top entry, asm say, from the assumption list and then acts as thmtac(asn\(asm\))...

<table>
<thead>
<tr>
<th>Tactic</th>
</tr>
</thead>
</table>
|\[ \{ asm, \Gamma \} t \]
|thmtac (asm \(\vdash\) asm) (\(\{ \Gamma \} \) t) |

**Uses**  
To use an assumption as a theorem

**Errors**  
9302 the assumption list is empty
7.3. General Tactics and Tacticals

SML

\[
\text{val } \text{prim_rewrite_tac} : \text{CONV NET} \rightarrow \text{CANON} \rightarrow (\text{THM} \rightarrow \text{TERM} \ast \text{CONV}) \text{OPT} \rightarrow (\text{CONV} \rightarrow \text{CONV}) \rightarrow \text{EQN\_CXT} \rightarrow \text{THM list} \rightarrow \text{TACTIC};
\]

**Description** This is the tactic based on `prim_rewrite_conv` (q.v.), with the same parameters as that function, except for the last argument:

\[
\text{Tactic}
\]

\[
\{ \Gamma \} \vdash t \\
\{ \Gamma \} \vdash t'
\]

where \(t'\) is the result of rewriting \(t\) in the manner prescribed by the arguments.

\[
\text{val } \text{prove_tac} : \text{THM list} \rightarrow \text{TACTIC};
\]

**Description** This tactic is an automatic proof procedure appropriate to the current proof context.

At the point of applying this tactic to its theorems it will access the current setting of proof context field `pr_tac`, apply it to the theorem list immediately, and then to the goal, with its assumptions temporarily removed when available (i.e. the result is partially evaluated with only the list of theorems). The original assumptions will be returned to the resulting subgoals using `check_asm_tac`.

\[
\text{Tactic}
\]

\[
\{ \Gamma \} \vdash t \\
\text{current_ad_pr_tac(thms({}, t)) THEN MAP\_EVERY check_asm_tac } \Gamma
\]

**See Also** `PC_T1` to defer accessing the proof context until application to the goal; and, `asm_prove_tac` for the form that does react to the presence of assumptions.

**Errors**

51021 The current proof context was created in theory ?0 at a point now either not in scope, deleted or modified

and as the proof context setting.

It is possible that if `prove_tac` does not prove all its subgoals, then there may be an identification of newly introduced variables with free variables in the assumptions that were temporarily put to one side. This will lead to failures in the execution of the proof parts of the tactics that constitute the current proof context’s automatic prover. Such a failure may not give particularly helpful messages concerning the cause of the failure. The problem is avoided by using `asm_prove_tac`, or by a call to `rename_tac` to change the offending variable names.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
Chapter 7. PROOF IN HOL

SML

val prove_thm : (string * TERM * TACTIC) \rightarrow THM;

Description  prove_thm (key, gl, tac) applies the tactic tac to the goal ([], tm), and, if the tactic succeeds in proving the goal saves the theorem under the key given, and returns the resulting theorem.

prove_thm performs \( \alpha \)-conversion as necessary to ensure that the theorem returned has the same form as the specified goal. In circumstances where these adjustments are known not to be necessary, simple_tac_proof may be used to avoid the overhead.

Defn  prove_thm (key, tm, tac) = save_thm(key, tac_proof(([[],tm]).tac));

Uses  The subgoal package is the normal interactive mechanism for developing proofs using tactics. prove_thm is typically used in tactic programming and other proof procedures, in cases where it is necessary to ensure that the correct goal is proved and saved.

Errors
9501 the tactic returned unsolved subgoals: ?0
9502 evaluation of the tactic failed: ?0
9503 the proof returned by the tactic failed: ?0
9504 the proof returned by the tactic proved ?0 which could not be converted into the desired goal.
9507 the conclusion ?0 is not of type \( \vdash BOOL \)

See Also  simple_tac_proof, prove_thm.

SML

val prove_∃_tac : TACTIC;

Description  This tactic is an automatic proof procedure for existential proofs, appropriate to the current proof context.

At the point of applying this tactic to a goal it will access the current setting of proof context field prove_∃, apply it to the goal, with its assumptions temporarily removed, using conv_tac. The original assumptions will be returned to the resulting subgoals using check_asm_tac.

Tactic
\[
\begin{align*}
\{ \Gamma \} & \quad t \\
conv_tac(current_ad_cs_∃_conv())([]), t & \quad \text{prove}_∃_tac \text{ thms} \\
\text{THEN MAP_EVERY check_asm_tac } & \Gamma
\end{align*}
\]

See Also  asm_prove_∃_tac that does react to any assumptions that are present.

Errors
51021 The current proof context was created in theory ?0 at a point now either not in scope, deleted or modified

and as the proof context setting.
7.3. General Tactics and Tacticals

SML

```sml
val rename_tac : (TERM * string)list -> TACTIC;
```

Description  `rename_tac` renames variables (bound or free) in a goal. It is typically used when a goal contains several variables with the same name or to introduce names which are better mnemonics. For the latter purpose, the argument controls the algorithm used to make variants of the names.

The renaming affects both the conclusions and the assumptions of the goal. Variables are renamed to ensure that the new goal satisfies the following conditions:

- No two free variables with different types have the same name.
- No bound variable has the same name as a free variable or a variable which is bound in an outer scope.
- No variable shall have the same name as any constant in scope.

Before a variable is checked, it is looked up in the `renaming` association list, and if present it is treated as if the name were the corresponding string. The function `variant`, q.v., is used to rename variables.

The function may be partially evaluated with only the renamings argument.

Note that applying the tactic in the subgoal package will give rise to the message “The subgoal <label> is α-convertible to its goal”.

For example,

```
Tactic
{ k = 1 }
(∀ i:N×N• ∃i:N• i = 0)
∧ (∀ j:N×N
   • (∃k:N• k = Fst j)
     ∧ ∀j:N• j = k)
```

```
rename_tac
[⌜j:N×N⌝, "apple"],
[⌜j:N⌝, "banana"],
[⌜k:N⌝, "carrot"],
```

```
{ apple = 1 }
(∀ i• ∃ i'• i' = 0)
∧ (∀ apple
   • (∃ carrot'• carrot' = Fst apple)
     ∧ ∀ banana• banana = carrot)
```

Uses  In clarifying goals where the variable names clash or are unparseable or are inconvenient.

Errors  `[3007] 0 is not a term variable`

SML

```sml
val REPEAT_N_T : int -> TACTIC -> TACTIC;
val REPEAT_N : int -> TACTIC -> TACTIC;
```

Description  `REPEAT_N_T n` is a tactical which repeatedly applies its tactic argument `n` times. Unlike `REPEAT` it fails if the tactic fails. If `n` is not greater than 0 then `REPEAT_N_T n tac` is a tactic which has no effect.

`REPEAT_N` is an alias for `REPEAT_N_T`.

Errors  As for the tactic being repeated.
val REPEAT_TTCL : THM_TACTICAL -> THM_TACTICAL;

Description  REPEAT_TTCL ttcl is a theorem tactical which applies ttcl repeatedly until it fails.

Uses  For use in programming theorem tacticals. As for the argument theorem tactic.

val REPEAT_T : TACTIC -> TACTIC;
val REPEAT : TACTIC -> TACTIC;

Description  REPEAT_T is a tactical which repeatedly applies its tactic argument until it fails. This may cause an infinite loop of evaluation, or even no change, if the tactic fails on the first application. REPEAT is an alias for REPEAT_T.

val REPEAT_UNTIL_T1 : (GOAL -> bool) -> TACTIC -> TACTIC;
val REPEAT_UNTIL1 : (GOAL -> bool) -> TACTIC -> TACTIC;

Description  REPEAT_UNTIL1_T1 p tac is a tactical which repeatedly applies its tac until all outstanding subgoals either satisfy the predicate p or cause tac to fail.

If the goal already satisfies p, then REPEAT_UNTIL1_T1 p tac is a tactic which has no effect.

REPEAT_UNTIL1 is an alias for REPEAT_UNTIL1_T1.

Example
REPEAT_UNTIL1 (is or o snd) strip_tac
will repeatedly apply strip_tac until all outstanding subgoals have disjunctive conclusions or cause strip_tac to fail.

val REPEAT_UNTIL_T : (TERM -> bool) -> TACTIC -> TACTIC;
val REPEAT_UNTIL : (TERM -> bool) -> TACTIC -> TACTIC;

Description  REPEAT_UNTIL1_T p tac is a tactical which repeatedly applies its tac until all outstanding subgoals either have conclusions which satisfy the predicate p or cause tac to fail.

If the conclusion of the goal already satisfies p, then REPEAT_UNTIL1_T p tac is a tactic which has no effect.

REPEAT_UNTIL is an alias for REPEAT_UNTIL_T.

Example
REPEAT_UNTIL is or strip_tac
will repeatedly apply strip_tac until all outstanding subgoals have disjunctive conclusions or cause strip_tac to fail.
7.3. General Tactics and Tacticals

SML

val rewrite_tac : THM list -> TACTIC;
val pure_rewrite_tac : THM list -> TACTIC;
val once_rewrite_tac : THM list -> TACTIC;
val asm_once_rewrite_tac : THM list -> TACTIC;
val pure_asm_rewrite_tac : THM list -> TACTIC;
val once_asm_rewrite_tac : THM list -> TACTIC;
val pure_once_asm_rewrite_tac : THM list -> TACTIC;

**Description**  These are the rewriting tactics. They use the canonicalisation rule held by the current proof context (see, e.g., push_pc) to preprocess the theorem list. The context is accessed at the point when the rules are given a list of theorems.

If a tactic is “pure” then there is no default rewriting, otherwise the default rewriting conversion net held by the current proof context will be used in addition to user supplied material.

If a tactic is “once” then rewriting will proceed from the root of the of the conclusion of the theorem to be rewritten, towards the leaves, and will not descend through any rewritten subterm, using ONCE_MAP_WARN_C. If not, rewriting will continue, moving from the root to the leaves, repeating if any rewriting is successful, until there is no rewriting redex anywhere within the rewritten conclusion, using REWRITE_MAP_C. This may cause non-terminating looping.

If a tactic is “asm” then the theorems rewritten with will include the canonicalised asm_ruled assumptions of the goal.

**Errors**

26001 no rewriting occurred

Also as error 26003 and warning 26002 of REWRITE_MAP_C (q.v.).

SML

val rewrite_thm_tac : THM -> TACTIC;
val pure_rewrite_thm_tac : THM -> TACTIC;
val once_rewrite_thm_tac : THM -> TACTIC;
val asm_once_rewrite_thm_tac : THM -> TACTIC;
val pure_asm_rewrite_thm_tac : THM -> TACTIC;
val once_asm_rewrite_thm_tac : THM -> TACTIC;
val pure_once_asm_rewrite_thm_tac : THM -> TACTIC;

**Description**  These are rewriting tactics parameterised to take only one theorem. This parameterisation is convenient to use with the many tactic generating functions, such as LEMMA_T, which take a theorem tactic as an argument.

See, e.g. rewrite_tac for the details of the differences between these tactics.

**Errors**

26001 no rewriting occurred

Errors will be reported as if they are from the corresponding _tac: e.g. from rewrite_tac rather than rewrite_thm_tac. This allows a simple implementation, and for there to be no functionality change even in errors between using singleton lists with the originals, and these functions. The following warning indicates the result of, perhaps only some, of the rewriting was discarded.

26002 rewriting gave ill-formed results on some subterms
\begin{verbatim}
SML
val ROTATE_T : int -> TACTIC -> TACTIC;

Description ROTATE_T i tac is a tactic which first applies tac and, if this does not achieve the goal, rotates the resulting subgoals by i places. i is taken modulo the number of subgoals produced by tac.

Thus if the result of tac is:

\[
\begin{array}{c}
\{ \Gamma \} \ t \\
{ (\Gamma \backslash i \, 1) \, t_1; \ldots; \, \{ \Gamma_k \} \, t_k}
\end{array}
\]

then the result of ROTATE_T i t will be:

\[
\begin{array}{c}
\{ \Gamma \} \ t \\
{ (\Gamma \backslash (i+1) \, i+1) \, t_{i+1}; \ldots; \, \{ \Gamma_k \} \, t_k; \\
{ \{ \Gamma_1 \} \, t_1; \ldots; \, \{ \Gamma_i \} \, t_i}
\end{array}
\]

Uses For use in tactic programming to handle tactics which return their subgoals in an inconvenient order for the task at hand.

Errors As for tac.
\end{verbatim}
7.3. General Tactics and Tacticals

SML

val simple_taut_tac : TACTIC;

**Description**  A tautology prover. If the conclusion of the goal is a tautology then `taut_tac` will prove the goal. A tautology is taken to be any substitution instance of a term which is formed from boolean variables, the constants `T` and `F` and the following connectives:

\[ \land, \lor, \rightarrow, \leftrightarrow, \neg, \text{if } ... \text{ then } ... \text{ else} \]

and which is true for any assignment of truth values to the variables.

**Tactic**

\[
\begin{array}{c}
\{ \Gamma \} t \\
\text{simple_taut_tac}
\end{array}
\]

**See Also** `strip_tac`

**Errors**

[28121 Conclusion of the goal is not a tautology]

---

SML

val simple_\neg\neg_in_conv : CONV;

**Description**  This is a conversion which moves negations inside other predicate calculus connectives using whichever of the following rules applies:

\[
\begin{array}{c}
\neg t \\
\neg(t1 \land t2) \\
\neg(t1 \lor t2) \\
\neg(t1 \rightarrow t2) \\
\neg(t1 \leftrightarrow t2) \\
\neg(\text{if } a \text{ then } t1 \text{ else } t2) \\
\neg\forall x . t \\
\neg\exists x . t \\
\neg\exists_1 x . t = \forall x . \neg(t \land \forall x'. t[x'] \Rightarrow x' = x) \\
\neg T \\
\neg F
\end{array}
\]

\[
\begin{array}{c}
t \\
\neg t1 \lor \neg t2 \\
\neg t1 \land \neg t2 \\
t1 \land \neg t2 \\
(t1 \land \neg t2) \lor (t2 \land \neg t1) \\
(\text{if } a \text{ then } \neg t1 \text{ else } \neg t2) \\
\exists x . \neg t \\
\forall x . \neg t \\
\forall x . \neg(t \land \forall x'. t[x'] \Rightarrow x' = x) \\
F \\
T
\end{array}
\]

It does not handle paired quantifiers.

**Uses**  Tactic and conversion programming. The more general `\neg\neg_in_conv` is just as efficient as `simple_\neg\neg_in_conv` in cases where both succeed.

**See Also** `strip_tac`

**Errors**

[28131 No applicable rules for the term ?0]
\begin{verbatim}
\val simple_{\neg in tac} : TACTIC;
\end{verbatim}

**Description**  This is a tactic which moves negations inside other predicate calculus connectives using the following rules:

\[
\begin{align*}
\neg \neg t & \rightarrow t \\
\neg(t_1 \land t_2) & \rightarrow \neg t_1 \lor \neg t_2 \\
\neg(t_1 \lor t_2) & \rightarrow \neg t_1 \land \neg t_2 \\
\neg(t_1 \Rightarrow t_2) & \rightarrow t_1 \land \neg t_2 \\
\neg(t_1 \Leftrightarrow t_2) & \rightarrow (t_1 \land \neg t_2) \lor (t_2 \land \neg t_1) \\
\neg \forall x \bullet t & \rightarrow \exists x \bullet \neg t \\
\neg \exists x \bullet t & \rightarrow \forall x \bullet \neg t \\
\neg \exists_1 x \bullet t & \rightarrow \forall x \bullet (t \land \forall x' \bullet t[x'] \Rightarrow x' = x) \\
\neg T & \rightarrow F \\
\neg F & \rightarrow \text{goal solved}
\end{align*}
\]

It fails with paired quantifiers.

**Uses**  The more general \(_\neg in tac\) is just as efficient as \(simple_{\neg in tac}\) in cases where both succeed.

**See Also**  \(strip_tac\), \(contr_tac\), \(\neg T\), \(\neg in tac\)

**Errors**  
28025  No applicable rule for this goal

\begin{verbatim}
\val SIMPLE_{\neg IN THEN} : THM_TACTICAL;
\end{verbatim}

**Description**  This is a theorem tactical which applies a given theorem tactic to the result of transforming a theorem by moving a top level negation inside other predicate calculus connectives using the following rules:

\[
\begin{align*}
\neg \neg t & \rightarrow t \\
\neg(t_1 \land t_2) & \rightarrow \neg t_1 \lor \neg t_2 \\
\neg(t_1 \lor t_2) & \rightarrow \neg t_1 \land \neg t_2 \\
\neg(t_1 \Rightarrow t_2) & \rightarrow t_1 \land \neg t_2 \\
\neg(t_1 \Leftrightarrow t_2) & \rightarrow (t_1 \land \neg t_2) \lor (t_2 \land \neg t_1) \\
\neg \forall x \bullet t & \rightarrow \exists x \bullet \neg t \\
\neg \exists x \bullet t & \rightarrow \forall x \bullet \neg t \\
\neg \exists_1 x \bullet t & \rightarrow \forall x \bullet (t \land \forall x' \bullet t[x'] \Rightarrow x' = x) \\
\neg T & \rightarrow F \\
\neg F & \rightarrow T
\end{align*}
\]

The function may be partially evaluated with only its theorem tactic and theorem arguments. It fails with paired quantifiers.

**Uses**  The more general \(_\neg IN THEN\) is just as efficient as \(SIMPLE_{\neg IN THEN}\) in cases where both succeed.

**See Also**  \(strip_tac\), \(STRIP_THM_THEN\)

**Errors**  
28026  No applicable rule for this theorem
7.3. General Tactics and Tacticals

\textbf{SML}

\texttt{val simple_\forall\_tac : TACTIC;}

\textbf{Description} Reduce a universally quantified goal. It fails with paired quantifiers.

\begin{align*}
\text{Tactic} & \quad \{ \Gamma \} \forall x \bullet t[x] \\
& \quad \{ \Gamma' \} t[x'] \\
\end{align*}

\textit{simple_\forall\_tac}

where \(x'\) is a variant name of \(x\), different from any variable in \(\Gamma\) or \(t\).

\textbf{Uses} Tactic programming. The more general \(\forall\_tac\) is just as efficient as \textit{simple_\forall\_tac} in cases where both succeed.

\textbf{See Also} \(\forall\_tac\)

\textbf{Errors}

\texttt{28081 Goal is not of the form: \{ \Gamma \} \forall x \bullet t[x]}

\texttt{28091 Goal is not of the form: \{ \Gamma \} \exists x \bullet t2[x]}

\texttt{28092 Term ?0 has the wrong type}

\texttt{28091 Goal is not of the form: \{ \Gamma \} \exists x \bullet t2[x]}

\texttt{28092 Term ?0 has the wrong type}
**SML**
\[
\text{val SIMPLE.}\exists\_\text{THEN} : (\text{THM} \to \text{TACTIC}) \to (\text{THM} \to \text{TACTIC});
\]

**Description** A theorem tactical which applies a given theorem tactic to the result of eliminating the outermost quantifier from a theorem of the form \( \Gamma \vdash \exists x \cdot t \). It fails with paired quantifiers.

\[\text{SIMPLE.}\exists\_\text{THEN} \text{thmtac} (\Gamma \vdash \exists x \cdot t) = \text{thmtac} (\Gamma \vdash t [x'/x])\]

where \( x' \) is a variant of \( x \) which does not appear in \( \Gamma \) or in the assumption or conclusion of the goal. The function is partially evaluated with only the theorem tactic and theorem arguments.

**Uses** Tactic programming. Note that the more general \( \exists\_\text{THEN} \) is just as efficient as \( \text{SIMPLE.}\exists\_\text{THEN} \) in cases where both succeed.

Error 28094 normally arises when \( x' \) is also introduced by the proof of \( \text{ttac} \), and occurs during the application of the proof of \( \text{SIMPLE.}\exists\_\text{THEN} \). The bound variable \( x' \) should be renamed to something that doesn’t cause this identification of distinct variables, by using \( \text{rename_tac} \) (q.v.).

**See Also** \( \exists\_\text{THEN} \)

**Errors**
- 28093 \( ?0 \) is not of the form: \( \Gamma \vdash \exists x \cdot t' \)
- 28094 Error in proof of \( \text{SIMPLE.}\exists\_\text{THEN} \).
  - Usually indicates chosen skolem variable \( ?0 \) also introduced by proof of supplied theorem tactic,
  - which gave \( ?1' \), and the two became identified: use \( \text{rename_tac} \) to rename original bound variable \( ?2 \)

**SML**
\[
\text{val simple.}\exists_1\_\text{conv} : \text{CONV};
\]

**Description** This is a conversion which turns a unique existential quantifier into an equivalent existential quantifier

\[
\begin{align*}
\Gamma \vdash (\exists_1 x \cdot t[x]) & \iff \\text{simple.}\exists_1\_\text{conv} \\
(\exists x \cdot t[x]) & \iff (\exists x \cdot t) \land \forall x' \cdot t[x] \Rightarrow x' = x
\end{align*}
\]

**Uses** Tactic and conversion programming. The more general \( \exists_1\_\text{conv} \) is just as efficient as \( \text{simple.}\exists_1\_\text{conv} \) in cases where both succeed.

**See Also** \( \text{strip_tac} \)

**Errors**
- 4019 \( ?0 \) is not of form: \( \exists_1 v \cdot t \)
7.3. General Tactics and Tacticals

**SML**
```sml
val simple_∃₁_tac : TERM -> TACTIC;
```

**Description** Simplify a unique existentially quantified goal with a particular witness. It fails with paired quantifiers.

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

**Tactic**

\[
\begin{array}{c}
\{ \Gamma \} \quad \text{simple}_\exists \exists_1 x \cdot P[x] \\
\{ \Gamma \} \quad P[t]; \\
\{ \Gamma \} \quad \forall x' \cdot P[x'] \Rightarrow x' = t \\
\end{array}
\]

where \(x'\) is a variant of \(x\) which does not occur free in \(t\).

**Uses** Tactic programming. The more general \(\exists_1 \text{tac}\) is just as efficient as \(\text{simple}_\exists\exists_1 \text{tac}\) in cases where both succeed.

**Errors**
- 28101 Goal is not of the form: \(\{ \Gamma \} \exists_1 x \cdot P[x]\)
- 28092 Term \(?0\) has the wrong type

**SML**
```sml
val SIMPLE_∃₁_THEN : (THM -> TACTIC) -> (THM -> TACTIC);
```

**Description** A theorem tactical which applies a given theorem tactic to the result of eliminating the outermost quantifier from a theorem of the form \(\Gamma \vdash \exists_1 x \cdot t\). It fails with paired quantifiers.

\[
\begin{array}{l}
\text{SIMPLE}_\exists_1 \text{THEN } \text{thmtac} (\Gamma \vdash \exists_1 x \cdot t) = \\
\text{thmtac} (\Gamma \vdash t[x'/x] \land \forall x'' \cdot P[x''] \Rightarrow x'' = x)
\end{array}
\]

where \(\Gamma x'\) and \(\Gamma x''\) are distinct variants of \(\Gamma x\) which do not appear free in \(\Gamma\) or in the assumptions or conclusion of the goal.

**Uses** Tactic programming. The more general \(\exists_1 \text{THEN}\) is just as efficient as \(\text{SIMPLE}_\exists_1 \text{THEN}\) in cases where both succeed.

**Errors**
- 28102 \(?0\) is not of the form: \('\Gamma \vdash \exists_1 x \cdot t'\)

**SML**
```sml
val SOLVED_T : TACTIC -> TACTIC;
```

**Description** \(\text{SOLVED}_T \text{tac}\) is a tactic which applies \(\text{tac}\) to the goal and fails if it does not solve the goal. I.e. it fails unless the tactic returns an empty list of subgoals.

\(\text{SOLVED}_T\) does not check that the proof delivered by the tactic is valid. \(\text{tac}\_\text{proof}\) may be used to achieve this type of effect.

**Uses** Tactic programming, for when a tactic that fails to prove a goal is likely to leave an untidy goal state.

**See Also** \(\text{tac}\_\text{proof}\)

**Errors**
- 9602 the tactic did not solve the goal

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
**Chapter 7. PROOF IN HOL**

**SML**

```sml
val spec_asm_tac : TERM -> TERM -> TACTIC;
val list_spec_asm_tac : TERM -> TERM list -> TACTIC;
val spec_nth_asm_tac : int -> TERM -> TACTIC;
val list_spec_nth_asm_tac : int -> TERM list -> TACTIC;
```

**Description**  These are four methods of specialising assumptions, differing by single or lists of values to specialise to, and in the method of selection of the assumption. All of them leave the old assumption in place, and place the instantiated assumption onto the assumption list using `strip_asm_tac`. If the desired behaviour differs from any of those supplied then use `GET_ASM_T` and its cousins to create the desired functionality.

```
Tactic
\{ \Gamma, \forall vs[x_1,...] \cdot f [x_1,...]^\| \} \ t1
```

The following all handle paired abstractions in a similar manner.

```
Tactic
\{ \Gamma, \forall x_1 ... \cdot f [x_1,...]^\| \} \ t1
```

```
Tactic
\{ \Gamma_1...n-1, \forall x' \cdot f [x']^\|, \Gamma \} \ t1
```

```
Tactic
\{ \Gamma_1...n-1, \forall x_1 ... \cdot f [x_1,...]^\|, \Gamma \} \ t1
```

**Definitions**

```sml
fun spec_asm_tac asm instance =
    GET_ASM_T asm (strip_asm_tac o \forall\ elim instance);
fun list_spec_asm_tac asm instances =
    GET_ASM_T asm (strip_asm_tac o list\forall\ elim instances);
fun spec_nth_asm_tac n instance =
    GET_NTH_ASM_T n (strip_asm_tac o \forall\ elim instance);
fun list_spec_nth_asm_tac n instances =
    GET_NTH_ASM_T n (strip_asm_tac o list\forall\ elim instances);
```

**Errors**  As the constituents of the implementing functions.
7.3. General Tactics and Tacticals  271

```sml
val SPEC_ASM_T : TERM -> TERM -> (THM -> TACTIC) -> TACTIC;
val LIST_SPEC_ASM_T : TERM -> TERM list -> (THM -> TACTIC)
  -> TACTIC;
val SPEC_NTH_ASM_T : int -> TERM -> (THM -> TACTIC) -> TACTIC;
val LIST_SPEC_NTH_ASM_T : int -> TERM list -> (THM -> TACTIC)
  -> TACTIC;
```

**Description**  These are four methods of specialising assumptions, differing by single or lists of values to specialise to, and in the method of selection of the assumption. All of them leave the old assumption in place, and place the instantiated assumption onto the assumption list using their theorem tactic. If the desired behaviour differs from any of those supplied then use `GET_ASM_T` and its cousins to create the desired functionality.

```
val asm instance thmtac = SPEC_ASM_T asm (thmtac o \_.elim instance);
val asm instances thmtac = LIST_SPEC_ASM_T asm (thmtac o list\_.elim instances);
val n instance thmtac = SPEC_NTH_ASM_T n (thmtac o \_.elim instance);
val n instances thmtac = LIST_SPEC_NTH_ASM_T n (thmtac o list\_.elim instances);
```

**Errors**  As the constituents of the implementing functions.
val step_strip_tac : TACTIC;
val step_strip_asm_tac : THM -> TACTIC;

Description  These functions provide methods of single-stepping through the application of strip_tac and strip_asm_tac (q.v.).

When stripping the antecedent of an implication, or a theorem, into the assumption list strip_tac and strip_asm_tac respectively do all their stripping in one application of the tactic. This is not appropriate behaviour when:

1. Explaining the detailed behaviour of these functions by example applications.
2. Attempting to “debug” a failed or inappropriate stripping.
3. When a partial strip into the assumption list is desired.

The two functions provided give a single-step stripping of antecedents and theorems. They represent sets of objects that are partially stripped into the assumption list by making the conclusion of the resulting goal an implication with the antecedent being the conjunction of the partially stripped objects and the consequent being the unstripped part of the goal. Repeated use of the provided functions closely corresponds to the processing order and effect of strip_tac and strip_asm_tac. Under certain unusual circumstances the match may not be exact.

Example

\[ \vdash (a \lor b) \land c \Rightarrow ((a \land c) \lor (b \land c)) \]

Single steps to:

\[ \vdash (a \land c) \Rightarrow ((a \land c) \lor (b \land c)) \]
and \[ \vdash (b \land c) \Rightarrow ((a \land c) \lor (b \land c)) \]

Each single step to:

\[ a \vdash c \Rightarrow ((a \land c) \lor (b \land c)) \]
and \[ b \vdash c \Rightarrow ((a \land c) \lor (b \land c)) \]

Each single step to:

\[ a, c \vdash \Rightarrow ((a \land c) \lor (b \land c)) \]
and \[ b, c \vdash \Rightarrow ((a \land c) \lor (b \land c)) \]

These five steps (two on each branch) map onto one call of strip_tac.

Errors

28003  There is no stripping technique for ?0 in the current proof context
7.3. General Tactics and Tacticals

SML

\[
\text{val strip_asm_tac : THM -> TACTIC;}
\]

**Description**  
`strip_asm_tac` is a general purpose tactic for splitting a theorem up into useful pieces using a range of simplification techniques, including a parameterised part, before using it to increase the stock of assumptions.

First, before attempting to use the transformations below, `strip_asm_tac` uses the current proof context’s theorem stripping conversion to attempt to rewrite the outermost connective in the theorem.

Then the following simplification techniques will be tried. Using `sat` as an abbreviation for `strip_asm_tac`:

\[
\begin{align*}
\text{sat} \ (\vdash a \land b) & \quad \rightarrow \quad \text{sat} \ (\vdash a) \ \text{THEN} \ \text{sat} \ (\vdash b) \\
\text{sat} \ (\exists x \cdot a) & \quad \rightarrow \quad \text{sat} \ (a[x'/x] \vdash a[x'/x]) \\
\text{sat} \ (\vdash a \lor b)(\{\Gamma\} \ t) & \quad \rightarrow \quad \text{sat} \ (a \vdash a) \ (\{\Gamma\} \ t) ; \text{sat} \ (b \vdash b) \ (\{\Gamma\} \ t)
\end{align*}
\]

I.e. `strip_asm_tac` does a case split resulting in two subgoals when it processes a disjunction.

After all of the available simplification techniques have been attempted `strip_asm_tac` then proceeds as `check_asm_tac` (q.v.) to use the simplified theorem either to prove the goal or to generate additional assumptions.

**See Also**  
`STRIP_THM_THEN`, used to implement this function. `check_asm_tac`, `strip_tac`, `strip_asm_conv`.

SML

\[
\text{val strip_concl_conv : CONV;}
\]

\[
\text{val strip_asm_conv : CONV;}
\]

**Description**  
`strip_concl_conv tm`; applies the conclusion stripping conversion from the current proof context, to rewrite the outermost connective in the term `tm`.

`strip_asm_conv tm`; applies the assumption stripping conversion from the current proof context, to rewrite the outermost connective in the term `tm`.

**Errors**  
28003 There is no stripping technique for ?0 in the current proof context
val strip_concl_tac : TACTIC;
val strip_tac : TACTIC;

Description strip_concl_tac, more usually known by its alias, strip_tac, is a general purpose tactic for simplifying away the outermost connective of a goal. It first tries to apply the conclusion stripping conversion from the current proof context, to rewrite the outermost connective in the goal. If that conversion fails, it tries to simplify the goal by applying an applicable member of the following collection of tactics (only one could possibly apply):

\[
\text{simple}_\forall\text{tac}, \quad \land\text{tac}, \quad \Rightarrow T \text{ strip_asm_tac, } t\text{tac}
\]

Failing either being successful, it tries concl_in_asm_tac to prove the goal, and failing that, returns the error message below.

Note how new assumptions generated by the tactic are processed using strip_asm_tac, which uses the current proof context’s theorem stripping conversion. strip_tac may produce several new subgoals, or may prove the goal.

REPEAT strip_tac in the proof context “basic_hol” (amongst others) will prove all tautologies automatically. It will, however, not succeed in proving some substitution instances of tautologies involving positive and negative instances of a quantified subterm.

Uses This is the usual way of simplifying a goal involving predicate calculus connectives, and other functions “understood” by the current prof context.

See Also STRIP T and STRIP_THM_THEN which are used to implement this function. taut_tac for an alternative simplifier. swap_\lor_tac to rearrange the conclusion for tailored stripping. Also strip_concl_conv, strip_asm_conv.

Errors

28003 There is no stripping technique for ?0 in the current proof context

val STRIP_CONCL_T : (THM \rightarrow TACTIC) \rightarrow TACTIC;
val STRIP_T : (THM \rightarrow TACTIC) \rightarrow TACTIC;

Description STRIP_CONCL_T ttac is a general purpose way of stripping goals and passing any new assumptions generated by the stripping to a tactic generating function, ttac. STRIP_CONCL_T attempts to apply the conversion held for it in the current proof context to rewrite the goal. The conversion is extracted from the current proof context by current_ad_sc_conv. If that fails it attempts to apply one of the following list of tactics (in order):

\[
\text{simple}_\forall\text{tac}, \quad \land\text{tac}, \quad \Rightarrow T \text{ ttac, } t\text{tac}
\]

If none of the above apply it tries concl_in_asm_tac, and failing that, return the error message below.

The conversion in the current proof context held by current_ad_sc_conv (q.v.) is derived by applying eqn_cxt_conv to an equational context in the proof context, extracted by get_sc_eqn_cxt.

STRIP_T is an alias for STRIP_CONCL_T.

Uses Tactic programming.

See Also strip_asm_tac, strip_tac, strip_concl_conv.

Errors

28003 There is no stripping technique for ?0 in the current proof context
7.3. General Tactics and Tacticals

**Description**

*STRIP_THM_THEN* provides a general purpose way of stripping theorems into primitive constituents before using them in a tactic proof. *STRIP_THM_THEN* attempts to apply the conversion held for the function in the current proof context, which is extracted by *current_ad_st_conv*. to rewrite the theorem. If that fails it attempts to apply a theorem tactical from the following list (in order):

\[ ∧ \_ THEN, \quad ∨ \_ THEN, \quad SIMPLE ∃ \_ THEN \]

The conversion in the current proof context got by *current_ad_st_conv* (q.v.) is derived by applying *eqn_ext_conv* to an equational context in the proof context extracted by *get_st_eqn_ext*.

The function is partially evaluated with only the theorem tactic and theorem arguments.

**Uses**

Tactic programming.

**See Also**

*strip_asm_tac*, *strip_tac*.

**Errors**

28003 There is no stripping technique for ?0 in the current proof context

---

**Description**

Strip the negation of current goal into the assumption list and make an assumption, suitably negated, into the current goal. If the simplifications it does are ignored, *swap_asm_concl_tac* asmis equivalent to

Example

\[ \text{contr_tac THEN asm_ante_tac asm} \]

and *swap_nth_asm_concl_tac* nis equivalent to

Example

\[ \text{contr_tac THEN DROP_NTH_ASM_T n ante_tac} \]

Tactic

\[
\frac{\{ Γ, \neg t1 \}}{\{ Γ \neg \neg t1, Γ \} \neg t1} \quad \text{swap_asm_concl_tac}
\]

\[
\frac{\{ Γ, \neg t1, \neg tm, \ldots, \neg tn \}}{\{ Γ \neg \neg t1, Γ \neg \neg tm, \ldots, Γ \neg \neg tn \} \neg tm} \quad \text{swap_nth_asm_concl_tac}
\]

If the assumption is a negated term then the double negation will be eliminated.

**See Also**

Other *swap* and *SWAP* functions.

**Errors**

9303 the index ?0 is out of range

28052 Term ?0 is not in the assumptions
\textbf{Description}  \hspace{1em} Process the negation of current goal with the supplied theorem tactic and make an assumption, suitably negated, into the current goal. If the simplifications it does are ignored, \texttt{SWAP\_ASM\_CONCL\_T asm ttac} is equivalent to

\begin{verbatim}
Example
\end{verbatim}

\begin{verbatim}
Example
\end{verbatim}

\begin{verbatim}
Tactic
\end{verbatim}

\begin{verbatim}
Tactic
\end{verbatim}

If the assumption is a negated term then the double negation will be eliminated.

\textbf{See Also}  \hspace{1em} Other \texttt{swap} and \texttt{SWAP} functions.

\textbf{Errors}  
9303  the index \texttt{?0} is out of range  
28027  Conclusion of goal does not have type \texttt{\lbrack;}\texttt{BOOL}\lbrack;  
28052  Term \texttt{?0} is not in the assumptions

\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}

\textbf{Description}  \hspace{1em} Interchange the disjuncts of a disjunctive goal.

\begin{verbatim}
Tactic
\end{verbatim}

\textbf{Uses}  \hspace{1em} For use in conjunction with \texttt{strip\_tac} (q.v.) when the reduction of \texttt{\{\Gamma\} a \lor b} to \texttt{\{\neg a, \Gamma\} b} is inappropriate.

\textbf{See Also}  \hspace{1em} \texttt{\lor\_left\_tac}, \texttt{\lor\_right\_tac}, \texttt{swap\_\lor\_tac}, \texttt{strip\_tac}

\textbf{Errors}  
28041  Goal is not of the form: \texttt{\{\Gamma\} a \lor b}
7.3. General Tactics and Tacticals

SML

val tac_proof : (GOAL * TACTIC) -> THM;

**Description**  
`tac_proof(gl, tac)` applies the tactic `tac` to the goal `gl`, and, if the tactic succeeds in proving the goal returns the resulting theorem.

`tac_proof` performs α-conversion, introduces additional assumptions, and reorders assumptions as necessary to ensure that the theorem returned has the same form as the specified goal (note that this is not possible if the goal has α-equivalent assumptions). In circumstances where these adjustments are known not to be necessary, `simple_tac_proof` may be used to avoid the overhead.

**Uses**  
The subgoal package is the normal interactive mechanism for developing proofs using tactics. `tac_proof` is typically used in tactic programming and other proof procedures, in cases where it is necessary to ensure that the correct goal is proved.

**Errors**

9501 the tactic returned unsolved subgoals: ?0
9502 evaluation of the tactic failed: ?0
9503 the proof returned by the tactic failed: ?0 which could not be converted into the desired goal.
9504 the goal contains alpha-equivalent assumptions (?0 and ?1)
9505 the assumption ?0 is not of type ¦:BOOL
9506 the conclusion ?0 is not of type ¦:BOOL
9507 the goal contains alpha-equivalent assumptions (?0 and ?1)

See Also  
`simple_tac_proof`, `prove_thm`.

SML

val taut_conv : CONV;

**Description**  
A tautology prover. A tautology is taken to be any universally quantified substitution instance of a term which is formed from boolean variables, the constants `T` and `F` and the following connectives:

\[ \land, \lor, \Rightarrow, \Leftrightarrow, \neg, \text{if} \ldots \text{then} \ldots \text{else} \]

and which is true for any assignment of truth values to the variables. If its argument is a tautologically true term, then the function will return a theorem that the term is equivalent to `T`.

**Conversion**

| \( \vdash t \Leftrightarrow T \) | \[ t \] |

\[ \text{taut_conv} \]

See Also  
`taut_tac`, `taut_rule`, `simple_taut_tac`.

**Errors**

27037 ?0 is not tautologically true
val taut_rule : TERM -> THM;

**Description** A tautology prover. A tautology is taken to be any universally quantified substitution instance of a term which is formed from boolean variables, the constants $T$ and $F$ and the following connectives:

$\land, \lor, \Rightarrow, \Leftrightarrow, \neg, \text{ if ... then ... else}$

and which is true for any assignment of truth values to the variables. If its argument is such a tautology then the function will return that term as a theorem.

\[ \vdash t \quad \text{taut_rule} \]

\[ t \\wedge \]

**See Also** taut_tac, taut_conv, simple_taut_tac.

**Errors**

27037 ?0 is not tautologically true

---

val taut_tac : TACTIC;

**Description** A tautology prover. If the conclusion of the goal is a tautology then taut_tac will prove the goal. A tautology is taken to be any (perhaps universally quantified) substitution instance of a term which is formed from boolean variables, the constants $T$ and $F$ and the following connectives:

$\land, \lor, \Rightarrow, \Leftrightarrow, \neg, \text{ if ... then ... else}$

and which is true for any assignment of truth values to the variables.

\[ \{ \Gamma \} t \quad \text{taut_tac} \]

**See Also** strip_tac, taut_rule, taut_conv, simple_taut_tac.

**Errors**

29020 Conclusion of the goal is not a universally quantified tautology

---

val THEN_LIST_T : (TACTIC * TACTIC list) -> TACTIC;

val THEN_LIST : (TACTIC * TACTIC list) -> TACTIC;

**Description** THEN_LIST_T is a tactical used as an infix operator. tac THEN_LIST_T tlist is a tactic that applies tac, and then applies the first member of tlist to the first resulting subgoal, the second to the second, etc. If there are not the correct number of tactics in the list then an error will be raised. THEN_LIST is an alias for THEN_LIST_T.

**Errors**

9101 number of tactics must equal the number of subgoals

As failures of the initial tactic or the tactics in the list.
7.3. General Tactics and Tacticals

SML

val THEN.T1 : (TACTIC * TACTIC) -> TACTIC;
val THEN1 : (TACTIC * TACTIC) -> TACTIC;

Description

THEN.T1 is a tactical used as an infix operator. tac1 THEN.T1 tac2 is the tactic that applies tac1 and then applies tac2 to the first of the resulting subgoals and id_tac to any other subgoals. If tac1 returns no subgoals, then nor will tac1 THEN.T1 tac2. THEN1 is an alias for THEN.T1.

It is intended for use in conjunction with induction tactics or tactics like lemma_tac for which the first subgoal (i.e., the base case of the induction or the lemma) often has a simple proof.

See Also

THEN

Errors

As the errors of tac1 and tac2.

SML

val THEN_TRY_LIST_T : (TACTIC * TACTIC list) -> TACTIC;
val THEN_TRY_LIST : (TACTIC * TACTIC list) -> TACTIC;

Description

THEN.Try_LIST_T is a tactical used as an infix operator. tac THEN.Try_LIST_LIST_T list is a tactic that applies tac, and then attempts to apply the first member of list to the first resulting subgoal, the second to the second, etc. If there are not the correct number of tactics in the list then an error will be raised. If any member of list fails on a particular subgoal, then that subgoal is returned unchanged. THEN_LIST is an alias for THEN_LIST_T.

Errors

| 9101 number of tactics must equal the number of subgoals |

As failures of the initial tactic.

SML

val THEN.Try_TTCL : (THM_TACTICAL * THM_TACTICAL) -> THM_TACTICAL;

Description

THEN.Try_TTCL is a theorem tactical combinator. It is an infix operator which applies the first theorem tactical, and then, if it succeeds, the second theorem tactical, using only the first result if the second fails.

Uses

For use in programming theorem tacticals.

SML

val THEN.Try_T : (TACTIC * TACTIC) -> TACTIC;
val THEN.Try : (TACTIC * TACTIC) -> TACTIC;

Description

THEN.Try_T is a tactical used as an infix operator. tac1 THEN.Try_T tac2 is the tactic that applies tac1 and then attempts to apply tac2 to each resulting subgoal (perhaps none). If tac2 fails on any particular subgoal then that subgoal will be unchanged from the result of tac1. If tac1 fails then the overall tactic fails. THEN.Try is an alias for THEN.Try_T.

Errors

As the errors of tac1.

SML

val THEN.TTCL : (THM_TACTICAL * THM_TACTICAL) -> THM_TACTICAL;

Description

THEN.TTCL is a theorem tactical combinator. It is an infix operator which composes two theorem tacticals using ordinary function composition:

\[(tcl1 \text{ THEN.TTCL tcl2}) \text{ thmtac thm} = (tcl1 o tcl2) \text{ thmtac thm}\]

Uses

For use in programming theorem tacticals.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
**THEN.T**

\[ \text{THEN} : (\text{TACTIC} \times \text{TACTIC}) \rightarrow \text{TACTIC}; \]

**Description**  
THEN.T is a tactical used as an infix operator. tac1 THEN.T tac2 is the tactic that applies tac1 and then applies tac2 to each resulting subgoal (perhaps none). \text{THEN} is an alias for \text{THEN.T}.

**Errors**  
As the errors of tac1 and tac2.

**TOP.ASM.T**

\[ \text{TOP.ASM} : (\text{THM} \rightarrow \text{TACTIC}) \rightarrow \text{TACTIC}; \]

**Description**  
If the top entry in the assumption list is asm say, \text{TOP.ASM.T} thmtac acts as thmtac(asm ⊢ asm).

\[ \text{Tactic} \quad \{ \text{asm}, \Gamma \} \Rightarrow \text{thmtac (asm \vdash \text{asm}) ( \{ \text{asm}, \Gamma \} \Rightarrow \text{thmtac})} \]

**Uses**  
To use an assumption as a theorem

**Errors**  
9302 the assumption list is empty

**TRY.TTCL**

\[ \text{TRY.TTCL} : \text{THM.TACTICAL} \rightarrow \text{THM.TACTICAL}; \]

**Description**  
\text{TRY.TTCL} ttcl is a theorem tactical which applies ttcl if it can, and otherwise acts as \text{ID}. \text{THEN}.

**Uses**  
For use in programming theorem tacticals.

**TRY.T**

\[ \text{TRY.T} : \text{TACTIC} \rightarrow \text{TACTIC}; \]

\[ \text{TRY} : \text{TACTIC} \rightarrow \text{TACTIC}; \]

**Description**  
\text{TRY.T} tac is a tactic which applies tac to the goal and if that fails leaves the goal unchanged. It is the same as \text{tac ORELSE id_tac}. \text{TRY} is an alias for \text{TRY.T}.

**t_tac**

\[ \text{t_tac} : \text{TACTIC}; \]

**Description**  
Prove a goal with conclusion ‘T’.

\[ \text{Tactic} \quad \{ \Gamma \} \Rightarrow \text{t_tac} \]

**See Also**  
\text{strip_tac}, \text{taut_tac}.

**Uses**  
Tactic programming.

**Errors**  
28011 Goal does not have the form \{\Gamma\} T
7.3. General Tactics and Tacticals

SML

val var_elim_asm_tac : TERM -> TACTIC;
val var_elim_nth_asm_tac : int -> TACTIC;
val VAR_ELIM_ASM_T : TERM -> (THM -> TACTIC) -> TACTIC;
val VAR_ELIM_NTH_ASM_T : int -> (THM -> TACTIC) -> TACTIC;

**Description** These tactics and tacticals do variable elimination with a chosen assumption of the goal. They take an assumption of the form: \(\text{⌜} \text{var} = \text{value} \text{⌟} \) or \(\text{⌜} \text{value} = \text{var} \text{⌟} \), where \(\text{var}\) is a variable and, if the subterm \(\text{value}\) does not contain \(\text{var}\) free, they substitute \(\text{value}\) for the free variable \(\text{var}\) throughout the goal (discarding the original assumption).

If an assumption is an equation of variables, then the tactic will strip digits and the current variant suffix from the right of the two variable names, and will choose to eliminate the variable with the shortest remaining name string, taking eliminating the left hand side variable if the strings are of equal length (this is a heuristic). If the variables are the same then the assumption is just discarded with no further effect.

\(\text{var\_elim\_asm\_tac}\) will determine whether its term argument is an assumption of the above form. If so, it will substitute for the free variable \(\text{var}\) with \(\text{value}\) throughout the goal, stripping any changed assumptions back into the goal (returning the rest by \(\text{check\_asm\_tac}\)), and then discard the original assumption. \(\text{VAR\_ELIM\_ASM\_T}\) allows the users choice of function to be applied to the modified assumptions.

\(\text{var\_elim\_nth\_asm\_tac}\) works as \(\text{var\_elim\_asm\_tac}\), except it takes an integer indicating the “nth” assumption is to be used. \(\text{VAR\_ELIM\_NTH\_ASM\_T}\) allows the users choice of function to be applied to the modified assumptions.

**See Also** \(\text{all\_var\_elim\_asm\_tac}\) and its kin to apply this sort of functionality to all the assumptions simultaneously. \(\text{prop\_eq\_prove\_tac}\) for more sophisticated approach to these kinds of problems.

**Errors**

9301 the term \?0 is not in the assumption list
9303 the index \?0 is out of range
29027 \?0 is not of the form \(\text{⌜} \text{var} = ... \text{⌟} \) or \(\text{⌜} ... = \text{var} \text{⌟} \) where the variable \(\text{⌜} \text{var} \text{⌟} \) is not free in \(\text{⌜} ... \text{⌟} \)

SML

val \(\Leftrightarrow\_T2\) : (THM -> TACTIC) -> (THM -> TACTIC) -> TACTIC;

**Description** Reduce a bi-implication by passing the operands to tactic generating functions.

\[
\begin{align*}
\{ \Gamma \} & \ t1 \ \Leftrightarrow \ t2 \\
\text{ttac1}\{ \ t1, \ \Gamma \ \} & \vdash \ t2; \ \text{ttac2}\{ \ t2, \ \Gamma \ \} & \vdash \ t1 \\
\Leftrightarrow\_T2 & \ \text{ttac1 ttac2}
\end{align*}
\]

**See Also** \(\Leftrightarrow\_T, \text{STRIP\_CONCL\_T}\)

**Errors**

28061 Goal is not of the form: \(\{ \ \Gamma \ \} \ t1 \ \Leftrightarrow \ t2\)
\begin{verbatim}
SML
| val \_\_tac : TACTIC;

**Description** Reduce a bi-implication to two subgoals.

\[ \begin{array}{c}
\{ \Gamma \} \ t \ \leftrightarrow \ T \\
\{ \Gamma' \} \ t
\end{array} \Rightarrow \_\_tac \]

**See Also** \_\_T

**Errors** 
28061 Goal is not of the form: \{ \Gamma \} t \ \leftrightarrow \ t

\end{verbatim}

\begin{verbatim}
SML
| val \_\_THEN2 : (THM -> TACTIC) -> (THM -> TACTIC) -> (THM -> TACTIC);

**Description** A theorem tactical to apply given theorem tactics to the the result of eliminating \( \leftrightarrow \) from a theorem of the form \( \Gamma \vdash t1 \leftrightarrow t2 \).

\[ \_\_THEN2 \ ttac1 \ ttac2(\Gamma \vdash t1 \leftrightarrow t2) = ttac1(\Gamma \vdash t1 \Rightarrow t2) \ \text{THEN} \ ttac2(\Gamma \vdash t2 \Rightarrow t1) \]

The function is partially evaluated with only the theorem tactic and theorem arguments.

**See Also** \_\_THEN, STRIP_THM_THEN

**Errors** 
28062 ?0 is not of the form: ‘\( \Gamma \vdash t1 \leftrightarrow t2 \)’

\end{verbatim}

\begin{verbatim}
SML
| val \_\_THEN : (THM -> TACTIC) -> (THM -> TACTIC);

**Description** A theorem tactical to apply a given theorem tactic to the result of eliminating \( \leftrightarrow \) from a theorem of the form \( \Gamma \vdash t1 \leftrightarrow t2 \).

\[ \_\_THEN \ thmtac (\Gamma \vdash t1 \leftrightarrow t2) = thmtac (\Gamma \vdash t1 \Rightarrow t2) \ \text{THEN} \ thmtac (\Gamma \vdash t2 \Rightarrow t1) \]

The function is partially evaluated with only the theorem tactic and theorem arguments.

**See Also** \_\_THEN2, STRIP_THM_THEN

**Errors** 
28062 ?0 is not of the form: ‘\( \Gamma \vdash t1 \leftrightarrow t2 \)’

\end{verbatim}

\begin{verbatim}
SML
| val \_\_t_tac : TACTIC;

**Description** Simplifies a goal of the form: ...\( \leftrightarrow \)T or T\( \leftrightarrow \)....

\[ \begin{array}{c}
\{ \Gamma \} \ t \ \leftrightarrow \ T \\
\{ \Gamma' \} \ t
\end{array} \Rightarrow \_\_t\_tac \]

\[ \begin{array}{c}
\{ \Gamma \} \ T \ \leftrightarrow \ t \\
\{ \Gamma' \} \ t
\end{array} \Rightarrow \_\_t\_tac \]

**Errors** 
28012 Goal not of form: \{ \Gamma \} t \ \leftrightarrow \ T or \{ \Gamma \} T \ \leftrightarrow \ t

**See Also** strip_tac

**Uses** Tactic programming.

\end{verbatim}
7.3. General Tactics and Tacticals

\[
\text{val } \iff \_T : (\text{THM } \rightarrow \text{TACTIC}) \rightarrow \text{TACTIC};
\]

**Description**  
Reduce a bi-implication by passing each operand to a tactic generating function.

\[
\begin{align*}
\text{Tactic} & \quad \{ \Gamma \} \ t1 \iff t2 \quad \iff \_T \\
& \quad \text{ttac} \{ t1, \Gamma \} \vdash \text{ttac} \{ t2, \Gamma \} \vdash t1
\end{align*}
\]

**See Also**  
\_T2, STRIP\_CONCL\_T

**Errors**  
28061  Goal is not of the form: \{ \Gamma \} \ t1 \iff t2

\[
\text{val } \_\_\text{tac} : \text{TACTIC};
\]

**Description**  
Reduce the proof of a conjunction to the proof of its conjuncts.

\[
\begin{align*}
\text{Tactic} & \quad \{ \Gamma \} \ t1 \land t2 \\
& \quad \text{\_tac} \\
& \quad \; \{ \Gamma \} \ t1; \; \{ \Gamma \} \ t2
\end{align*}
\]

**See Also**  
strip\_tac

**Errors**  
28031  Goal is not of the form: \{ \Gamma \} \ t1 \land t2

\[
\text{val } \_\_\text{THEN2} : (\text{THM } \rightarrow \text{TACTIC}) \rightarrow (\text{THM } \rightarrow \text{TACTIC}) \rightarrow (\text{THM } \rightarrow \text{TACTIC});
\]

**Description**  
A theorem tactical to apply given theorem tactics to the conjuncts of a theorem of the form \( \Gamma \vdash t1 \land t2 \).

\[
\begin{align*}
\_\_\text{THEN2} \ thmtac1 \ thmtac2 \ (\Gamma \vdash t1 \land t2) &= thmtac1 \ (\Gamma \vdash t1) \text{ THEN thmtac2 } (\Gamma \vdash t2)
\end{align*}
\]

**See Also**  
\_\_\text{THEN}, STRIP\_\_THM\_\_THEN

**Errors**  
28032  \?0 is not of the form: \( \Gamma \vdash t1 \land t2 \)

\[
\text{val } \_\_\text{THEN} : (\text{THM } \rightarrow \text{TACTIC}) \rightarrow (\text{THM } \rightarrow \text{TACTIC});
\]

**Description**  
A theorem tactical to apply a given theorem tactic to the conjuncts of a theorem of the form \( \Gamma \vdash t1 \land t2 \).

\[
\begin{align*}
\_\_\text{THEN} \ thmtac \ (\Gamma \vdash t1 \land t2) &= thmtac \ (\Gamma \vdash t1) \text{ THEN thmtac } (\Gamma \vdash t2)
\end{align*}
\]

The function may be partially evaluated with only its theorem tactic and theorem arguments.

**See Also**  
\_\_\text{THEN2}, STRIP\_\_THM\_\_THEN

**Errors**  
28032  \?0 is not of the form: \( \Gamma \vdash t1 \land t2 \)
<table>
<thead>
<tr>
<th>val ∨_left_tac : TACTIC;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong>  Take the left disjunct of the current goal as the subgoal.</td>
</tr>
<tr>
<td>N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.</td>
</tr>
<tr>
<td>Tactic</td>
</tr>
</tbody>
</table>
| \[
\frac{\{ \Gamma \} \ a \lor \ b}{\{ \Gamma \} \ a} \quad \lor\_left\_tac
\] |
| **See Also**  ∨_left_tac, swap_∨_tac, strip_tac |
| **Errors**  |
| 28041  Goal is not of the form: \{ \Gamma \} \ a \lor \ b |

<table>
<thead>
<tr>
<th>val ∨_right_tac : TACTIC;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong>  Take the right disjunct of the current subgoal as the new subgoal.</td>
</tr>
<tr>
<td>N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.</td>
</tr>
<tr>
<td>Tactic</td>
</tr>
</tbody>
</table>
| \[
\frac{\{ \Gamma \} \ a \lor \ b}{\{ \Gamma \} \ b} \quad \lor\_right\_tac
\] |
| **See Also**  ∨_right_tac, swap_∨_tac, strip_tac |
| **Errors**  |
| 28041  Goal is not of the form: \{ \Gamma \} \ a \lor \ b |

<table>
<thead>
<tr>
<th>val ∨_THEN2 : (THM −&gt; TACTIC) −&gt; (THM −&gt; TACTIC) −&gt; (THM −&gt; TACTIC);</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong>  A theorem tactical to perform a case split on a given disjunctive theorem applying tactic generating functions to the extra assumption in each branch.</td>
</tr>
<tr>
<td>∨_THEN2 ttac1 ttac2 (Δ ⊢ t1 \lor t2) ({\Gamma} t) = ttac1 (t1 ⊢ t1) ({\Gamma} t); ttac2 (t2 ⊢ t2)({\Gamma} t)</td>
</tr>
<tr>
<td>The function may be partially evaluated with only its theorem tactic and theorem arguments.</td>
</tr>
<tr>
<td><strong>See Also</strong>  STRIP_THM_THEN, ∨_THEN</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
</tr>
<tr>
<td>28042  ?0 is not of the form: ‘(\Gamma \vdash t1 \lor t2)’</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>val ∨_THEN : (THM −&gt; TACTIC) −&gt; (THM −&gt; TACTIC);</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong>  A theorem tactical to perform a case split on a given disjunctive theorem applying a tactic generating function to the extra assumption in each branch.</td>
</tr>
<tr>
<td>∨_THEN ttac (Δ ⊢ t1 \lor t2) ({\Gamma} t) = ttac (t1 ⊢ t1) ({\Gamma} t); ttac (t2 ⊢ t2)({\Gamma} t)</td>
</tr>
<tr>
<td>The function may be partially evaluated with only its theorem tactic and theorem arguments.</td>
</tr>
<tr>
<td><strong>See Also</strong>  STRIP_THM_THEN, ∨_THEN2</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
</tr>
<tr>
<td>28042  ?0 is not of the form: ‘(\Gamma \vdash t1 \lor t2)’</td>
</tr>
</tbody>
</table>
### SML

```sml
val \neg\_elim\_tac : TERM \to TACTIC;
```

**Description**  
Proof by showing assumptions give rise to two contradictory subgoals.  

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

The function may be partially evaluated with only its term argument.

**Uses**  
In tactic programming. If an assumption has its negation also in the assumption list this will make for a rapid proof. `asm\_ante\_tac t1 THEN strip\_tac` is a more memorable idiom for handling such a case in interactive use but is a little slower.

**See Also**  
`strip\_tac`

**Errors**  
28022 \(?0 is not boolean\)

---

### SML

```sml
val \neg\_in\_conv : CONV;
```

**Description**  
This is a conversion which moves a top level negation inside other predicate calculus connectives using whichever one of the following rules applies:

\[
\begin{align*}
\neg t &= t \\
\neg (t_1 \land t_2) &= \neg t_1 \lor \neg t_2 \\
\neg (t_1 \lor t_2) &= \neg t_1 \land \neg t_2 \\
\neg (t_1 \rightarrow t_2) &= t_1 \land \neg t_2 \\
\neg (t_1 \leftrightarrow t_2) &= (t_1 \land \neg t_2) \lor (t_2 \land \neg t_1) \\
\neg (if\ a\ then\ t_1\ else\ t_2) &= (if\ a\ then\ \neg t_1\ else\ \neg t_2) \\
\neg \forall v\_s t &= \exists v\_s \neg t \\
\neg \exists v\_s t &= \forall v\_s \neg t \\
\neg \exists_1 v\_s t &= \forall v\_s \neg (t \land \forall v\_s' t[v\_s' = v\_s]) \\
\neg T &= F \\
\neg F &= T
\end{align*}
\]

**Uses**  
Tactic and conversion programming.

**See Also**  
`simple\_\neg\_in\_conv`, `\neg\_in\_tac`

**Errors**  
28131 \(No\ applicable\ rules\ for\ the\ term\ ?0\)
val \neg\ _\text{in\_tac} : TACTIC;

\textbf{Description}  \ This is a tactic which moves a top level negation in the conclusion of the goal inside other predicate calculus connectives using the following rules:

\[
\begin{align*}
\neg t & \rightarrow t \\
\neg (t1 \land t2) & \rightarrow \neg t1 \lor \neg t2 \\
\neg (t1 \lor t2) & \rightarrow \neg t1 \land \neg t2 \\
\neg (t1 \Rightarrow t2) & \rightarrow t1 \land \neg t2 \\
\neg (t1 \Leftrightarrow t2) & \rightarrow (t1 \land \neg t2) \lor (t2 \land \neg t1) \\
\neg \forall vs\bullet t & \rightarrow \exists vs\bullet \neg t \\
\neg \exists vs\bullet t & \rightarrow \forall vs\bullet \neg t \\
\neg \exists_1 vs\bullet t & \rightarrow \forall vs\bullet (t \land \forall vs'\bullet t[vs'] \Rightarrow vs' = vs) \\
\neg T & \rightarrow F \\
\neg F & \rightarrow T
\end{align*}
\]

\textbf{Uses}

See Also  simple\_\neg\ _\text{in\_tac}, \neg\ _\text{in\_conv}

Errors  28025 No applicable rule for this goal

val \neg\ _\text{IN\_THEN} : THM\_TACTICAL;

\textbf{Description}  \ This is a theorem tactical which applies a given theorem tactic to the result of transforming a theorem by moving a top level negation inside other predicate calculus connectives using the following rules:

\[
\begin{align*}
\neg t & \rightarrow t \\
\neg (t1 \land t2) & \rightarrow \neg t1 \lor \neg t2 \\
\neg (t1 \lor t2) & \rightarrow \neg t1 \land \neg t2 \\
\neg (t1 \Rightarrow t2) & \rightarrow t1 \land \neg t2 \\
\neg (t1 \Leftrightarrow t2) & \rightarrow (t1 \land \neg t2) \lor (t2 \land \neg t1) \\
\neg \forall vs\bullet t & \rightarrow \exists vs\bullet \neg t \\
\neg \exists vs\bullet t & \rightarrow \forall vs\bullet \neg t \\
\neg \exists_1 vs\bullet t & \rightarrow \forall vs\bullet (t \land \forall vs'\bullet t[vs'] \Rightarrow vs' = vs) \\
\neg T & \rightarrow F \\
\neg F & \rightarrow T
\end{align*}
\]

This function partially evaluates given only the theorem and theorem-tactical.

See Also  SIMPLE\_\neg\ _\text{IN\_THEN}

Errors  29010 No applicable rule for ?0
### 7.3. General Tactics and Tacticals

**SML**

```sml
val _rewrite_canon : THM -> THM list
val _rewrite_canon : THM -> THM list
```

**Description**  These are some of the standard canonicalisation functions used for breaking theorems up into lists of equations for use in rewriting. They four perform the following transformations:

- \(\neg\) \_rewrite\_canon  \(\Gamma \vdash \neg(t_1 \lor t_2)\) =  \(\Gamma \vdash t_1 \land \neg t_2\)
- \(\neg\) \_rewrite\_canon  \(\Gamma \vdash \neg\exists_vs\cdot t\) =  \(\Gamma \vdash \forall_vs\cdot\neg t\)
- \(\neg\) \_rewrite\_canon  \(\Gamma \vdash \neg t\) =  \(\Gamma \vdash t\)
- \(\forall\) \_rewrite\_canon  \(\Gamma \vdash \forall_vs\cdot t\) =  \(\Gamma \vdash t\)

**See Also**  simple\_\neg\_rewrite\_canon, simple\_\forall\_rewrite\_canon.

**Errors**

| 26201 | Failed as requested |

The area given by the failure will be fail\_canon.

### SML

```sml
val _T2 : TERM -> (THM -> TACTIC) -> (THM -> TACTIC) -> TACTIC;
```

**Description**  A form of proof by contradiction using two theorem tactics to simplify the subgoals.

Note that strip\_tac may be used to push a negation inside other logical connectives, which is often the best way of handling a negated goal.

**Tactic**

\[
\begin{array}{c}
\{ \Gamma \} \neg t_2 \\
\text{ttac1} (t_2 \vdash t_2) \{ \Gamma \} t_1; \\
\text{ttac2} (t_2 \vdash t_2) \{ \Gamma \} \neg t_1
\end{array}
\]

- \(\neg\) _T2
- \text{ttac1} \text{ttac2}
- \(\Gamma t_1\neg\)

**Uses**  To prove a negated term by showing that assuming the term gives rise to a contradiction.

**See Also**  strip\_tac, contr\_tac, \(\neg\)\_tac, STRIP\_CONCL\_T, \(\neg\)\_in\_conv

**Errors**

| 28022 | ?0 is not boolean |
| 28023 | Goal is not of the form \(\Gamma \neg t\neg\) |
SML

```sml
val _-tac : TERM -> TACTIC;
```

**Description** A form of proof by contradiction as a tactic: \( \neg t2 \) holds if \( t2 \vdash t1 \) and \( t2 \vdash \neg t1 \) for some term \( t1 \).

Note that `strip_tac` may be used to push a negation inside other logical connectives, which is often the best way of handling a negated goal.

<table>
<thead>
<tr>
<th>Tactic</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ { \Gamma } \vdash \neg t2 ]</td>
</tr>
<tr>
<td>[ { \Gamma } \vdash t1 ]</td>
</tr>
<tr>
<td>[ { strip t2, \Gamma } \vdash \neg t1 ]</td>
</tr>
<tr>
<td>[ \vdash \neg \text{tac} ]</td>
</tr>
<tr>
<td>[ \Gamma \vdash t1 ]</td>
</tr>
</tbody>
</table>

**Uses** To prove a negated term by showing that assuming the term gives rise to a contradiction.

**See Also** `strip_tac`, `contr_tac`, `\neg T`

**Errors**

28022 ?0 is not boolean
28023 Goal is not of the form \( \vdash \neg t \)

---

SML

```sml
val _-T : TERM -> (THM -> TACTIC) -> TACTIC;
```

**Description** A form of proof by contradiction using a theorem tactic to simplify the subgoals.

Note that `strip_tac` may be used to push a negation inside other logical connectives, which is often the best way of handling a negated goal.

<table>
<thead>
<tr>
<th>Tactic</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ { \Gamma } \vdash \neg t2 ]</td>
</tr>
<tr>
<td>[ { \Gamma } \vdash t1 ]</td>
</tr>
<tr>
<td>[ { ttac t2, \Gamma } \vdash \neg t1 ]</td>
</tr>
<tr>
<td>[ \vdash \neg \text{Tac} ]</td>
</tr>
<tr>
<td>[ \Gamma \vdash t1 ]</td>
</tr>
</tbody>
</table>

**Uses** To prove a negated term by showing that assuming the term gives rise to a contradiction.

**See Also** `strip_tac`, `contr_tac`, `\neg_T`, `\neg in conv`

**Errors**

28022 ?0 is not boolean
28023 Goal is not of the form \( \vdash \neg t \)
7.3. General Tactics and Tacticals

Description These theorems are tautologies saved in the theory “misc” because they are frequently used in tactic and conversion programming.

The first seven theorems are De Morgan’s laws for the various propositional connectives formulated so that they can be used to normalise a propositional term by moving all negations inside other connectives. ¬_t_thm is also provided but is documented elsewhere.

The last three theorems give definitions for implication, bi-implication and conditional in terms of disjunction, conjunction and negation.

See Also ¬_t_thm.

| SML | val ¬_¬_thm : THM |
| val ¬_∨_thm : THM |
| val ¬_∧_thm : THM |
| val ¬_⇒_thm : THM |
| val ¬_if_ thm : THM |
| val ¬_f_ thm : THM |
| val ⇒_thm : THM |
| val ⇔_thm : THM |
| val if_thm : THM |

| val ⇒_tac : TACTIC; |

Description Strip the antecedent of an implicative goal into the assumption list.

Tactic

$$\{ \Gamma \} \ t1 \Rightarrow \ t2$$

strip t1, Γ$$ \\Rightarrow \_tac$$

Errors

28051 Goal is not of form: \{ \Gamma \} \ t1 \Rightarrow \ t2
val ⇒ _THEN : (THM → TACTIC) → (THM → TACTIC);

Description A theorem tactical to apply a given theorem tactic to the result of eliminating ⇒ from a theorem of the form Γ ⊢ t1 ⇒ t2.

⇒ _THEN thmtac (Γ ⊢ t1 ⇒ t2) = thmtac (Γ ⊢ ¬ t1 ∨ t2)

The function is partially evaluated with only the theorem tactic and theorem arguments.

Errors 28054 ?0 is not of the form: ‘Γ ⊢ t1 ⇒ t2’

val ⇒ _thm_tac : THM → TACTIC;

Description A tactic which uses a theorem whose conclusion is an implication, t1 ⇒ t2, to reduce a goal with conclusion t2 to t1.

\[
\begin{array}{c}
\{ \Gamma \} \quad t2 \\
\{ \Gamma \} \quad t1 \\
\end{array}
\]
⇒ _thm_tac

Γ1 ⊢ t1 ⇒ t2

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

Uses Mainly for use in tactic programming where the extra generality of bc _thm_tac and bc _tac is not required.

See Also bc _thm_tac, bc _tac.

Errors 29013 Conclusion of the goal is not ?0

val ⇒ _T : (THM → TACTIC) → TACTIC;

Description Reduce an implicative goal by passing the antecedent to a tactic generating function.

\[
\begin{array}{c}
\{ \Gamma \} \quad t1 ⇒ t2 \\
ttac \{ t1, \Gamma \} \quad t2 \\
\end{array}
\]
⇒ _T

ttac

Errors 28051 Goal is not of form: \{ \Gamma \} \quad t1 ⇒ t2

val ∀ _tac : TACTIC;

Description Reduce a universally quantified goal.

\[
\begin{array}{c}
\{ \Gamma \} \quad ∀ \ vs[\ x1 ,... \ ] \bullet \ t[\ x1 ,... \ ] \\
\{ \Gamma \} \quad t[\ x1 ,... \ ] \\
\end{array}
\]
∀ _tac

where x1 ‘ is a variant name of x1, etc, different from any variable in \( \Gamma \) or t.

See Also simple ∀ _tac

Errors 29001 Goal is not of the form: \{ \Gamma \} \quad ∀ \ vs \bullet t[\ vs]
7.3. General Tactics and Tacticals

| val | ∃_tac : TERM -> TACTIC ; |
| Description | Provide a witness for an existential subgoal. |
N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

\[
\frac{\{ \Gamma \} \exists vs[x_1,...] \cdot t_2[x_1,...]}{\{ \Gamma \} t_2[t'_1,...]} \quad \exists \text{tac} \quad t' \gamma
\]
where \(vs[t'_1,...] is t, type instantiated to have the same type as \(vs[x_1,...], and broken up using Fst and Snd as necessary.

See Also | simple_∃_tac |

Errors

| 29002 | Goal is not of the form: \{ Γ \} \exists vs \cdot t_2[vs] |
| 29008 | Cannot match witness ?0 to varstruct ?1 |

| val | ∃_THEN : (THM -> TACTIC) -> (THM -> TACTIC); |
| Description | A theorem tactical which applies a given theorem tactic to the result of eliminating the outermost quantifier from a theorem of the form \(Γ ⊢ ∃vs\bullet t\).

\[∃\_THEN \ thmtac (Γ ⊢ ∃vs[x_1,...]\bullet t) = thmtac (Γ' ⊢ t[x_1'/x_1,...])\]
where \(x_1'\gamma is a variant of \(x_1\gamma , etc, which does not appear in \(Γ or in the assumption or conclusion of the goal.

See Also | SIMPLE_∃_THEN |

Errors

| 29003 | ?0 is not of the form: 'Γ ⊢ ∃ vs \bullet t' |

| val | ∃₁_tac : TERM -> TACTIC ; |
| Description | Provide a witness for a goal with conclusion of the form \(∃_1x\bullet t\).
N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

\[
\frac{\{ \Gamma \} \exists_1 vs[x_1,...] \cdot P[x_1,...]}{\{ \Gamma \} P[t'_1,...] ; \{ \Gamma \} \forall vs[x_1',...]\bullet P[x_1',...] ⇒ vs[x_1',...] = t'} \quad \exists_1\text{tac}1 \quad t' \gamma
\]
where \(x_1'\) is a variant of \(x_1\) which does not occur free in \(t, t' is equal to \(t type instantiated to the type of \(vs[x_1,...], and \(vs[t'_1,...] equals \(t' (perhaps using Fst and Snd).

Errors

| 29004 | Goal is not of the form: \{ Γ \} ∃_1 vs \bullet t |
| 29008 | Cannot match witness ?0 to varstruct ?1 |
val \exists_1\mbox{THEN} : (THM \to TACTIC) \to (THM \to TACTIC);

Description  A theorem tactical which applies a given theorem tactic to the result of eliminating the outermost quantifier from a theorem of the form \( \Gamma \vdash \exists_1 vs \cdot t \).

\[ \exists_1\mbox{THEN} \ \mbox{thmtac} (\Gamma \vdash \exists_1 vs \cdot t) = \]
\[ \mbox{thmtac} (\Gamma \vdash t[x1'/x1,...] \land \forall vs[x1'",...] \ characterized\ P[x1'",...] \Rightarrow vs[x1'",...] = vs[x1',...]) \]

where \( \forall x1\) and \( \forall x1" \) are distinct variants of \( \forall x1 \), etc, which do not appear free in \( \Gamma \) or in the assumptions or conclusion of the goal.

Errors 29005 ?0 is not of the form: ‘\( \Gamma \vdash \exists_1 vs \cdot t \)’

val \epsilon\_tac : TERM \to TACTIC;
val \epsilon\_T : TERM \to (THM \to TACTIC) \to TACTIC;

Description  Given a choice term, \( \epsilon x \cdot t \) say, \( \epsilon\_tac \) sets \( \exists x \cdot t \) as a lemma, and derives the new assumption \( t[\epsilon x \cdot t/x] \) from it.

\( \epsilon\_T \) is the same as \( \epsilon\_tac \) except that it passes the new assumption to a tactic generating function.

Tactic
\[ \begin{array}{ccc}
\{ \Gamma \} & t1 \\
\{ \Gamma \} & \exists x : \{ \mbox{strip } t[\epsilon x \cdot t/x], \Gamma \} & t1 \\
\end{array} \]
\[ \epsilon\_tac \]
\[ \epsilon x \cdot t \]

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable. This occurs when the use of the choice function is in some sense irrelevant to the truth of the goal, e.g., \( (\epsilon x \cdot T) = (\epsilon x \cdot T) \).

See Also  all\_\epsilon\_tac, all\_\epsilon\_T (which are easier to use in most cases).

Errors 29050 ?0 is not of the form ‘\( \epsilon x \cdot p \)’
7.4 Propositional Equational Reasoning

SML

signature PropositionalEquality = sig

Description This is the signature of a structure containing proof procedures for propositional calculus with equality.

SML

(* Proof Context: prop_eq *)

Description This is a complete proof context whose purpose is to supply a decision procedure for problems involving sets of equalities and the propositional calculus.

Contents The rewriting, theorem stripping and conclusion stripping components are as for the proof context predicates (q.v.). The automatic proof tactic is prop_eq.prove_tac (q.v.) The automatic proof conversion just tries to prove its argument, t say, using the automatic proof tactic and returns \( t \equiv T \) if it succeeds.

SML

(* Proof Context: 'prop_eq *)

Description This is a component proof context whose purpose is to supply a decision procedure for problems involving sets of equalities and the propositional calculus.

Contents The automatic proof components are as for proof context prop_eq. Other components are blank.

SML

(* Proof Context: prop_eq_pair *)

Description This is a complete proof context whose main purpose is to supply a decision procedure for problems involving sets of equalities, the propositional calculus and pairing.

Contents The rewriting, theorem stripping and conclusion stripping components are as for the proof context predicates (q.v.) each augmented with conversion pair_eq.conv (q.v.) which effect the following transformations:

\[
\begin{align*}
Fst(a,b) = x & \rightarrow a = x \\
Snd(a,b) = y & \rightarrow b = y \\
x = Fst(a,b) & \rightarrow x = a \\
y = Snd(a,b) & \rightarrow y = b \\
(a,b) = (c,d) & \rightarrow a = c \land b = d \\
(a,b) = z & \rightarrow a = Fst z \land b = Snd z \\
z = (a,b) & \rightarrow Fst z = a \land Snd z = b \\
z = w & \rightarrow Fst z = Fst w \land Snd z = Snd w
\end{align*}
\]

The automatic proof tactic is prop_eq.prove_tac (q.v.). The automatic proof conversion just tries to prove its argument, t say, using the automatic proof tactic and returns \( t \equiv T \) if it succeeds.

SML

(* Proof Context: 'prop_eq_pair *)

Description This is a component proof context whose purpose is to supply a decision procedure for problems involving sets of equalities, the propositional calculus and pairing.

Contents The rewriting, theorem stripping and conclusion stripping components contain only the pair_eq.conv conversion. The automatic proof components are as for prop_eq.pair. Other components are blank.
val ASM_PROP_EQ_T : (THM list -> TACTIC) -> THM list -> TACTIC
val PROP_EQ_T : (THM list -> TACTIC) -> THM list -> TACTIC

Description These are theorem tacticals which process the argument theorems and (for ASM_PROP_EQ_T) the assumptions before calling the argument theorem tactic. A call of “ASM_PROP_EQ_T thm_tac thms” takes thms plus theorems representing any equations from the assumptions, these are canonicalised by the rewriting canon of the current proof context, then processed by prop_eq_rule (q.v.) to form the arguments passed to function thm_tac. The order of the assumptions may be changed. Tactical PROP_EQ_T does not use the assumptions.

Uses With the rewriting tactics.

val pair_eq_conv : CONV

Description This conversion transforms equations involving pairs and the constants Fst and Snd into new equations whose comparands have simpler types by using the first match found in the following rules:

\[
\begin{align*}
Fst(a,b) = x & \rightarrow a = x \\
Snd(a,b) = y & \rightarrow b = y \\
x = Fst(a,b) & \rightarrow x = a \\
y = Snd(a,b) & \rightarrow y = b \\
(a,b) = (c,d) & \rightarrow a = c \land b = d \\
(a,b) = z & \rightarrow a = Fst z \land b = Snd z \\
z = (a,b) & \rightarrow Fst z = a \land Snd z = b \\
z = w & \rightarrow Fst z = Fst w \land Snd z = Snd w
\end{align*}
\]

Uses The conversion is intended for use in tactic and conversion programming. It is usefully applied before using prop_eq_prove_tac or ASM_PROP_EQ_T (q.v.). The normal interactive interface is via rewriting or stripping in the proof context prop_eq_pair (q.v.).

Errors 84001 ?0 is not an equation involving pairs
val prop_eq_prove_tac : THM list -> TACTIC;

Description  This tactic is suitable to be used as an automatic proof procedure in a proof context, it aims to solve problems which may be solved by reasoning in the propositional calculus with equality.

The tactic has the following steps:

1. It strips all of the assumptions, using the stripping functions of the current proof context, back into the assumptions. More precisely, ‘DROP ASMS T (MAP EVERY strip_asm_tac)’ is used.

2. It applies contr_tac to increase the stock of assumptions.

3. It splits all of the assumptions into two groups, those which are equations and those which are not.

4. Using the equation assumptions and the given theorems, a new set of theorems is produced using prop_eq_rule (q.v.) which equate all members of an equivalence classes to a common member of the class.

5. It rewrites all of the other assumptions with these new theorems and with the rewriting theorems of the current proof context.

6. It strips the rewritten assumptions and the equational assumptions from step 3 back into the goal.
val prop_eq_rule : THM list -> THM list * THM list;

**Description**  Given a list of theorems with conclusions of the form $a_i = b_i$ for various $a_i$ and $b_i$ this function produces a set of theorems that equate all members of each equivalence class determined by the equations to a common value. The equivalence classes are the sets of all $a_i$ and $b_i$ that are equated either directly or transitively, they comprise terms that are $\alpha$-convertible rather than requiring strict equality. For each of the equivalence classes a set of theorems equating each term in the class to the “simplest” (see below) term in the class is generated. These new theorems have the simplest term as their right hand comparand, duplicated theorems and identity theorems are excluded. The first list in the result tuple contains the new theorems from all of the equivalence classes. The second list in the result tuple comprises all the argument theorems which were not equasions. The new theorems are intended to be used as arguments for a rewriting operation.

The choice of the “simplest” term is intended to give the most useful rewriting theorems and those which are least likely to loop. HOL constants are considered the most simple, variables next, then functional applications, with lambda abstractions considered the most complex. A simple recursive counting function is used to traverse each term to evaluate its complexity. Function *term_order* (q.v.) is used when the counting function cannot decide.

**Example**

Applying this rule to a list of theorems with the following conclusions:

\[
\begin{align*}
\vdash a1=b1 & \quad \vdash a1=c1 & \quad \vdash d1=c1 & \quad \vdash z1=x1 \\
\vdash b1=y1 & \quad \vdash z1=w1 & \quad \vdash w1=y1 & \quad \vdash c1=y1 \\
\vdash a2=b2 & \quad \vdash a2=c2 & \quad \vdash d2=c2 & \quad \vdash z2=x2 \\
\vdash b2=y2 & \quad \vdash z2=w2 & \quad \vdash w2=y2 & \quad \vdash c2=y2 \\
\vdash x \land y & 
\end{align*}
\]

will produce a list of theorems with the following conclusions as the first element of the result tuple:

\[
\begin{align*}
\vdash x1=a1 & \quad \vdash z1=a1 & \quad \vdash w1=a1 & \quad \vdash d1=a1 \\
\vdash y1=a1 & \quad \vdash b1=a1 & \quad \vdash c1=a1 \\
\vdash x2=a2 & \quad \vdash z2=a2 & \quad \vdash w2=a2 & \quad \vdash d2=a2 \\
\vdash y2=a2 & \quad \vdash b2=a2 & \quad \vdash c2=a2 
\end{align*}
\]

plus the non equational theorems the second element of the result tuple.
7.5 Algebraic Normalisation

SML
signature Normalisation = sig

Description  This is the signature of a structure containing conversions for monomial and polynomial term normalisation and related metalanguage functions.

SML
val anf_conv : CONV;
val ANF.C : CONV -> CONV;

Description  anf_conv is a conversion which proves theorems of the form ⊢ t1 = t2 where t1 is a term formed from atoms of type N and t2 is in what we may call additive normal form, i.e. it has the form: t1 + t2 + ..., where the ti have the form s1 * s2 * ... where the si are atoms. Here, by atom we mean a term which is not of the form t1 + t2 + ... or s1 * s2 * ....

The summands t1 and, within them, the factors s_j are given in increasing order with respect to the ordering on terms given by the function term_order, q.v. Arithmetic computation is carried out on atoms to ensure that at most one of the summands is a numeric literal and that, within each summand, at most one factor is a numeric literal. Any literal appears at the beginning of its factor or summand and addition of 0 or multiplication by 1 is simplified out.

ANF.C conv is a conversion which acts like anf_conv but which applies conv to each atom as it is encountered (and normalises the result recursively). The argument conversion may signal that it does not wish to change a subterm, t say, either by failing or by returning t = t, the former approach is more efficient.

The conversions fail with error number 81032 if there are no changes to be made to the term.

Errors 81032 ?0 is not of type "N" or is already in additive normal form

SML
val ASYM.C : CONV -> CONV
val GEN ASYM.C : TERM ORDER -> CONV -> CONV

Description  These conversions allow one to control the behaviour of a conversion by making it asymmetric with respect to an ordering relation on terms (in the sense that the resulting conversion will only prove theorems of the form t1 = t2 in which t2 strictly precedes t1 in the ordering.

ASYM.C c is a conversion which behaves like c on terms t1 for which c t1 is a theorem with conclusion t1 = t2 where t2 (strictly) precedes t1 in the standard ordering on terms given by term_order q.v. and fails on other terms.

GEN ASYM.C is like ASYM.C but allows the ordering function used to be supplied as a parameter. The parameter is interpreted as an ordering relation on terms in the same sense as the ordering relations used by sort, q.v.

Errors 81010 The conversion did not decrease the order of the term 81011 On argument ?0 the conversion returned ?1 which is not an equation
val cnf_conv : CONV;

**Description**  This is a conversion which proves theorems of the form \( t_1 \iff t_2 \) where \( t_2 \) is in conjunctive normal form, i.e. either \( T \) or \( F \) or a conjunction of one or more disjunctions in which each disjunct is a propositional atom. Here, by atom we mean either a term whose principal connective is not a propositional calculus connective or the negation of such a term.

The conversion simplifies disjunctions and conjunctions as they are generated according to the following schema.

\[
\begin{align*}
    a \land T & \rightarrow a \\
    T \land a & \rightarrow a \\
    F \land a & \rightarrow F \\
    a \land F & \rightarrow F \\
    a \land a & \rightarrow a \\
    a \land \neg a & \rightarrow F \\
    a \lor T & \rightarrow T \\
    T \lor a & \rightarrow T \\
    F \lor a & \rightarrow a \\
    a \lor F & \rightarrow a \\
    a \lor a & \rightarrow a \\
    a \lor \neg a & \rightarrow T \\
    \neg T & \rightarrow F \\
    \neg F & \rightarrow T
\end{align*}
\]

Note, however, that more global simplifications are not done, e.g. there is no attempt to eliminate a conjunct all of whose constituent atoms are contained in another conjunct. Thus, the conversion will not automatically prove tautologies.

The conversion fails with error number 81030 if there are no changes to be made to the term.

**See Also**  **strip_tac** and **taut_rule** which supply a more useful and efficient means for working with the propositional calculus in most cases.

**Errors**

- **81030**  : ?0 is not of type \( \langle \text{BOOL} \rangle \) or is already in conjunctive normal form
val dnf_conv : CONV;

**Description** This is a conversion which proves theorems of the form $\vdash t_1 \leftrightarrow t_2$ where $t_2$ is in disjunctive normal form, i.e. either $T$ or $F$ or a disjunction of one or more conjunctions in which each conjunct is a propositional atom. Here, by atom we mean either a term whose principal connective is not a propositional calculus connective or the negation of such a term.

The conversion simplifies disjunctions and conjunctions as they are generated according to the following schema.

| $a \land T$ | $a$ |
| $T \land a$ | $a$ |
| $F \land a$ | $F$ |
| $a \land F$ | $F$ |
| $a \land a$ | $a$ |
| $a \land \neg a$ | $F$ |
| $a \lor T$ | $T$ |
| $T \lor a$ | $T$ |
| $F \lor a$ | $a$ |
| $a \lor F$ | $a$ |
| $a \lor a$ | $a$ |
| $a \lor \neg a$ | $T$ |
| $\neg T$ | $F$ |
| $\neg F$ | $T$ |

Note, however, that more global simplifications are not done, e.g. there is no attempt to eliminate a disjunct all of whose constituent atoms are contained in another disjunct. Thus, the conversion will not automatically prove tautologies.

The conversion fails with error number 81031 if there are no changes to be made to the term.

**See Also** strip_tac and taut_rule which supply a more useful and efficient means for working with the propositional calculus in most cases.

**Errors**

| 81031 | "0 is not of type $\vdash \text{BOOL}$ or is already in disjunctive normal form" |
val gen_term_order : (TERM -> (TERM * INTEGER)) -> TERM -> TERM -> int;

Description  gen_term_order gives a means of creating orderings on terms. It is retained for backwards compatibility, make_term_order now being the recommended way of constructing term orderings.

In the call gen_term_order special, the idea is that whenever two terms, tm1 and tm2 say, are compared, special is applied to them to produce two pairs, (tm1', k1) and (tm2', k2) say. These pairs are then compared lexicographically (using the ordering recursively for the first components, in a similar way to term_order, q.v.). It is the caller’s responsibility to provide an argument special which will ensure that this procedure terminates. A sufficient condition is only to use functions special with the property that for some disjoint sets of terms X_1, X_2, ..., we have that special tm = (tm, 0) if tm /∈ X_i for any i and that special tm = (x_i, f_i(tm)) if tm ∈ X_i, where x_i is a fixed element of X_i and f_i is a fixed injection of X_i into the natural numbers.

See Also  make_term_order1 which is now the recommended way of constructing new term orderings.

val make_term_order : (TERM ORDER -> TERM ORDER) list -> TERM ORDER;

Description  make_term_order provides a systematic method for constructing term orderings. Its argument is a list of term order combinators: i.e., endofunctions on the type of term orderings.

The orderings make_term_order returns are derived from a base ordering on terms which works as follows:

1. Constants are ordered lexicographically by name (using ascii_order), then type (using type_order).
2. Variables are ordered lexicographically by name (using ascii_order), then type (using type_order).
3. Simple λ-abstractions are ordered lexicographically by recursion, bound variable first, then matrix.
4. Applications ordered lexicographically by recursion, function first, then operand.

If the above function ewre called base, then the ordering make_term_order [f, g, ..., h] acts as: f(g(...(h(base))...)) where each recursive call in base is a call on make_term_order [f, g, ..., h].

For example, the following defines an ordering on terms whcih makes the immediate successor of any term of type BOOL its immediate successor:

fun f t = (dest¬ t, 1) handle Fail => (t, 0);
val ¬_order = make_term_order [fn r => induced_order(f, pair_order r int_order)];
7.5. Algebraic Normalisation

val poly_conv : TERM ORDER ->
  THM -> THM -> THM -> THM -> THM ->
  CONV -> CONV -> CONV -> CONV;

Description This conversion normalises terms constructed from atoms using two operators,
both associative and commutative, the second of which, say \( \mathit{op}^\ast \) distributes over the other, say \( \mathit{op}^+ \). For clarity, we write the two operators with infix syntax although they need not actually
be infix constants. Here, by “atom” we mean any term which is not of the form \( t_1 \mathit{op}^+ t_2 \) or \( t_1 \mathit{op}^\ast t_2 \). The theorems computed by the conversion have the form \( t = t_1 \mathit{op}^+ t_2 \mathit{op}^+ \ldots \),
where the \( t_i \) are in non-decreasing order with respect to the ordering on terms given by the first
parameter and have the form \( s_1 \mathit{op}^\ast s_2 \mathit{op}^\ast \ldots \), where the \( s_i \) are atoms and are in non-decreasing
order.

The associativity and commutativity of the operators and the distributivity are given as the five
theorem parameters (which are also used to infer what the two operators are; n.b. the operators
can be arbitrary terms, they need not be constants). The remaining parameters are conversions
which are applied to each atom as it is encountered and to each subterm of the form \( t_i \mathit{op}^+ \ldots \) or
\( t_i \mathit{op}^\ast \ldots \) as it id created. In more detail the parameters are, in order, as follows:

1. A term ordering, such as \( \mathit{term\_order} \), q.v.
2. A theorem of the form \( \vdash \forall x \ y \ x \mathit{op}^+ y = y \mathit{op}^+ x \).
3. A theorem of the form \( \vdash \forall x \ y \ z \mathit{(x \mathit{op}^+ y)} \mathit{op}^+ z = x \mathit{op}^+ y \mathit{op}^+ z \).
4. A theorem of the form \( \vdash \forall x \ y \mathit{op}^\ast y = y \mathit{op}^\ast x \).
5. A theorem of the form \( \vdash \forall x \ y \ z \mathit{(x \mathit{op}^\ast y)} \mathit{op}^\ast z = x \mathit{op}^\ast y \mathit{op}^\ast z \).
6. A theorem of the form \( \vdash \forall x \ y \mathit{z \mathit{x \mathit{op}}} (y \mathit{op}^\ast z) = (x \mathit{op}^\ast y) \mathit{op}^\ast (x \mathit{op}^\ast z) \).
7. A conversion to be applied to any subterm of the form \( t_i \mathit{op}^+ \ldots \) whenever such a subterm
is created. The result of the conversion will not be further normalised.
8. A conversion to be applied to any subterm of the form \( t_i \mathit{op}^\ast \ldots \) whenever such a subterm
is created. The result of the conversion will not be further normalised.
9. A conversion to be applied to any atom as it is encountered. If the conversion produces a
non-atomic term, this is normalised recursively as it is produced.

The conversions supplied as parameters may signal that they do not wish to change a subterm,
\( t \) say, either by failing or by returning \( t = t \), the former approach is more efficient. The whole
conversion fails with error number 81025 if there are no changes to be made to the term.

Errors

81023 \( ?0 \) does not have the form \( \vdash t_1 \mathit{op}^1 (t_2 \mathit{op}^2 t_3) = (t_1 \mathit{op}^1 t_2) \mathit{op}^2 (t_1 \mathit{op}^1 t_3) \)
81024 \( ?0 \) and \( ?1 \) do not have the forms \( \vdash t_1 \mathit{op}^1 t_2 = t_2 \mathit{op}^1 t_1 \)
and \( \vdash t_1 \mathit{op}^1 (t_2 \mathit{op}^2 t_3) = (t_1 \mathit{op}^1 t_2) \mathit{op}^2 (t_1 \mathit{op}^1 t_3) \)
81025 \( ?0 \) is already sorted
**val** sort_conv : TERM ORDER $\rightarrow$ THM $\rightarrow$ THM $\rightarrow$ CONV $\rightarrow$ CONV $\rightarrow$ CONV;

**Description**  This conversion normalises a term constructed from atoms using an associative and commutative binary operator, $op$ say. For clarity, we write two operator with infix syntax although it need not actually be an infix constant. Here, by “atom” we mean any term which is not of the form $t_1 \ op \ t_2$. The theorems computed by the conversion have the form $t = t_1 \ op \ t_2 \ op \ ...$, where the $t_i$ are in non-decreasing order with respect to the ordering on terms given by the first parameter.

The associativity and commutativity of the operator are given as the two theorem parameters (which are also used to infer what $op$ is; n.b. $op$ can be an arbitrary term, it need not be a constant). The remaining parameters are conversions which are applied to each atom as it is encountered and to each subterm of the form $t = t_i \ op \ ...$ as it is created. In more detail the parameters are, in order, as follows:

1. A term ordering, such as term_order, q.v.
2. A theorem of the form $\vdash \forall x \ y \cdot t \ x \ y = t \ y \ x$.
3. A theorem of the form $\vdash \forall x \ y \ z \cdot (x \ op \ y) \ op \ z = x \ op \ y \ op \ z$.
4. A conversion to be applied to each subterm of the form: $t_i \ op \ ...$ whenever such a subterm is created. The result of the conversion will not be further normalised.
5. A conversion to be applied to each atom as it is encountered. If the conversion produces a non-atomic term, this is normalised recursively.

The conversions supplied as parameters may signal that they do not wish to change a subterm, $t$ say, either by failing or by returning $t = t$, the former approach is more efficient. The whole conversion fails with error number 81025 if there are no changes to be made to the term.

**Errors**

81021 ?0 does not have the form $\vdash t_1 \ op \ t_2 = t_2 \ op \ t_1$
81022 ?0 does not have the form $\vdash (t_1 \ op \ t_2) \ op \ t_3 = t_1 \ op (t_2 \ op \ t_3)$
81025 ?0 is already sorted
81029 Internal error: unexpected error in term normalisation package

**val** term_order : TERM $\rightarrow$ TERM $\rightarrow$ int;

**Description**  term_order gives an ordering relation on HOL terms. The ordering relation follows the same conventions as those used by the sorting function sort, namely, term_order $t_1 \ t_2$ is negative if $t_1$ precedes $t_2$, 0 if $t_1$ and $t_2$ are equivalent and positive if $t_2$ precedes $t_1$. The ordering used is, with some exceptions, that all constants precede all variables which precede all abstractions which precede all applications. Lexicographic ordering on the immediate constituents gives the ordering within each of these four classes (using alphabetic ordering of strings, type_order or term_order recursively to order the constituents as appropriate). The exceptions are (i) that any term of the form $\neg t$ comes immediately after $t$, (ii) that the numeric literals 0, 1, ... are taken in numeric rather than alphabetic order and come before all other terms, and (iii) that terms of the form $i \ * \ x$ where $i$ is a numeric literal are ordered so that the terms $x, 0 \ * \ x, 1 \ * \ x, 2 \ * \ x, ...$ are consecutive.

**See Also**  gen_term_order1 which is the recommended way of constructing new term orderings.
val type_order : TYPE -> TYPE -> int;

**Description**  *type_order* gives a useful ordering relation HOL types. The ordering relation follows the same conventions as those used by the sorting function *sort*, namely, *type_order t1 t2* is negative if *t1* precedes *t2*, 0 if *t1* and *t2* are equivalent and positive if *t2* precedes *t1*. The ordering used is essentially that type variables are ordered by the alphabetic ordering of their names and precede all compound types which are ordered by the lexicographic ordering on their immediate constituents (using the alphabetic ordering for the type constructor names and the type ordering recursively for its operands).
7.6 First Order Resolution

\begin{verbatim}
SML
|signature Resolution = sig
Description  This is the signature of a structure providing Resolution facilities to ICL HOL.

|(* resolution_diagnostics – boolean flag declared by new_flag *)
Description  This is by default false, but if set true then the resolution mechanism will report
the generation of new, unsubsumed theorems, and whether these subsume pre-existing theorems.
Uses  Provide the designer of the resolution functions access to detailed diagnostics. Not int-
tended for use by others. May be withdrawn.
\end{verbatim}
**SML**

```sml
type BASIC_RES_TYPE
  (* TERM * bool * (TERM * (TERM -> THM -> THM)) list
   * TYPE list * THM * TERM list * TYPE list * int
   * FRAG_PRIORITY *)
);

type RES_DB_TYPE (* = BASIC_RES_TYPE list * BASIC_RES_TYPE list *
                   BASIC_RES_TYPE list * THM list *);
```

**Description** These are type abbreviation for the basic resolution tool based on `prim_res_rule`. The arguments to `BASIC_RES_TYPE` are:

1. The term is a subterm of the theorem argument(5), reached through outer universal quantifications and all propositional connectives.
2. The bool is false if and only if the subterm occurs “negatively” in the conclusion of the theorem.
3. This list states how to specialise the given term to some other value in a theorem already specialised by the preceding entries in the list, and appropriately type instantiated.
4. The type list is the instantiable type variables of the subterm.
5. The theorem is the source of the fragment.
6. The term list is the term variables that may not be used in unifying the fragment.
7. The next type list is the type variables that may not be used in unifying the fragment.
8. The integer indicates the “generation”, i.e. the number of resolutions involved in creating the fragment (initial theorems are at 0).
9. This argument indicates the priority given to taking this fragment from the `toprocess` list to use next.

The arguments to `RES_DB_TYPE`:

1. Items yet to be checked against (`against`).
2. Items checked against, but to be rechecked against new items to check with (`done`).
3. Items to check with (`toprocess`).
4. Theorems used to derive current items (`dbdata`).
val BASIC_RESOLUTION_T : int -> THM list -> (THM -> TACTIC) -> (THM -> TACTIC) -> TACTIC;

Description  BASIC_RESOLUTION_T limit thms thmtac1 thmtac2 (seqasms, conc) will first apply thmtac1 to the negated goal, probably adding it into the assumption list in some manner. The assumptions derived from this will become the set of support, the pre-existing assumptions and the input thms will be the rest of the theorems. These theorems will be resolved against each other until only theorems with default generation past limit can be derived, or until \( \vdash F \) is derived, or until no further resolution can be done. An assumption’s or input theorem’s generation is 0, and a theorem that is the result of resolution has a default generation of 1 plus the sum of the generations of the resolved theorems. Its actual generation will be the minimum of its default generation, and the generations of any previous theorems it subsumes. In any resolution attempt at least one of the two theorems will be assumed fragments from the stripped goal, or be derived from an earlier resolution in the evaluation. Duplicates and pure specialisations will be discarded.

Resolution will be attempted on subterms reached through outer universal quantification, and propositional connectives, by specialising the outer quantifications, and by type instantiation on the input theorems thms where necessary and possible.

The resulting list of theorems will have all the thms removed, all the theorems derived from stripping and negating the goal, and all the old assumptions removed. MAP_EVERY thmtac is then applied to the new theorems, and then to the goal. As a special case, \( \vdash F \) is checked for, before any further processing. If present it will be used to prove the goal.

Uses  On its own, or in combination with some canonicalisation of the input theorems.

Errors

| 67003  | The limit, 0, must be a positive integer |
| 67004  | No resolution occurred |

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
7.6. First Order Resolution

SML

val BASIC_RESOLUTION_T1 : int → THM list → (THM → TACTIC) → TACTIC;

Description  BASIC_RESOLUTION_T1 limit thms thmtac (seqasms, conc) will take the theorems gained by asm_rule'ing the assumptions and thms as inputs. These theorems will be resolved against each other until only theorems with default generation past limit can be derived, or until \( \vdash F \) is derived, or until no further resolution can be done. An assumption’s or input theorem’s generation is 0, and a theorem that is the result of resolution has a default generation of 1 plus the sum of the generations of the resolved theorems. Its actual generation will be the minimum of its default generation, and the generations of any previous theorems it subsumes. In any resolution attempt at least one of the two theorems will be from the original goals assumptions, or be derived from an earlier resolution in the evaluation. Duplicates and pure specialisations will be discarded.

Resolution will be attempted on subterms reached through outer universal quantification, and propositional connectives, by specialising the outer quantifications, and by type instantiation on the input theorems thms where necessary and possible.

The resulting list of theorems will have all the thms removed, and all the old assumptions removed. MAP.EVERY thmtac is then applied to the new theorems, and then to the goal. As a special case, \( \vdash F \) is checked for, before any further processing. If present it will be used to prove the goal.

Uses  On its own, or in combination with some canonicalisation of the input theorems.

Errors

| 67003 | The limit, \( \geq 0 \), must be a positive integer |
| 67004 | No resolution occurred |

SML

val basic_resolve_rule: TERM → THM → THM → THM;

Description  basic_resolve_rule subterm pos neg attempts to resolve two theorems that have a common subterm, subterm, occurring “positively” in pos and “negatively” in neg.

\[
\begin{align*}
\Gamma \vdash P \ [\text{subterm}] \\
\Delta \vdash N \ [\text{subterm}]
\end{align*}
\]

\[
\text{simplify} \ (\Gamma, \Delta \vdash P[F] \lor N[T])
\]

\[
\text{basic_resolve_rule} \ 	ext{subterm}
\]

Where simplify carries out the simplifications in the predicate calculus where an argument is the constant \( \neg T \) or \( \neg F \), plus a few others.

Errors

| 3031 | ?0 is not of type \( \lnot:BOOL \) |
| 67009 | ?0 is not a subterm of ?1 |

SML

val basic_res_extract : RES_DB_TYPE → THM list;

Description  This is the extraction function for the basic resolution tool based on prim_res_rule. It does no more than return the fourth item of the RES_DB_TYPE tuple.
val basic_res_next_to_process : BASIC_RES_TYPE list -> BASIC_RES_TYPE list;

Description: This takes as the next fragment to process the first fragment which comes from a theorem that subsumed some pre-existing one, and failing that the next one on the list of fragments.

val basic_res_post :
  (THM -> THM -> int) ->
  (THM list * int) * RES_DB_TYPE ->
  (RES_DB_TYPE * bool);

Description: This is the post processor for the basic resolution tool based on prim_res_rule. The results will be split into their respective conjuncts (if any). Then basic_res_post subsum ((res, gen), data) will test each member of res, checking for the conclusion T or F, and then against each member of the theorem list of data. In checking one theorem against another it will use subsum - discarding the new theorem if the result is 1, and discarding (with tidying up of data) the original if the result is 2, or keeping both (except for discards from further tests) if the result is 0, or any other value bar 1 and 2. gen is the default “generation” of the new theorems, except that the fragments for each new theorem will have the minimum generation number of this default generation, and the generation of any theorem in data it subsumes.

val basic_res_pre : THM list -> THM list -> RES_DB_TYPE;

Description: This is the preprocessor for the basic resolution tool based on prim_res_rule. The first argument is the set of support theorems, the second argument is the rest of the input theorems. Each theorem will be fragmented, and each fragment added to the appropriate list (i.e. to the third list of the result if in the set of support, and the first list if otherwise). The final theorem list part of the result, dbdata, is just the appending of the first list of theorems to the second.

val basic_res Resolver : Unification.SSUBS -> int ->
  BASIC_RES_TYPE -> BASIC_RES_TYPE -> THM list * int;

Description: This is the resolver for the basic resolution tool based on prim_res_rule. Resolution seeks to find sufficient term specialisation and type instantiation on both terms to make one of the two term fragments the negation of the other, using term_unify. The resolution will not be attempted if the result would involve more resolutions than the “generations” limit. If this can be done then the two original theorems are specialised and instantiated in the same manner and the term fragment cancelled by basic_resolve_rule, and the result returned as a singleton list, paired with the default generation of the result. Prior to being returned, any allowed universal quantification will be added back in. In the basic resolution tool the generality of a list of theorems is unnecessary.

The SUBS argument is a “scratchpad” for the type unifier. The function keeps track of the number of resolutions used to create the result.

Errors:
67001 Neither argument is in the set of support
67002 Cannot resolve the two arguments
67008 term_unify succeeded on ?0 and ?1 but failed to resolve ?2 and ?3

Message is a variant on 67002, included for diagnostic purposes. It will be removed in a more stable product.
### 7.6. First Order Resolution

#### basic_res_rule

| val basic_res_rule : int -> THM list -> THM list ->
| \text{THM list} |

**Description**  
`basic_res_rule` \text{limit sos rest} will resolve the theorems in the set of support and the rest against each other until only theorems with default generation past \text{limit} can be derived, or until \( \vdash F \) is derived, or until no further resolution can be done. A input theorem’s generation is 0, and a theorem that is the result of resolution has a default generation of 1 plus the sum of the generations of the resolved theorems. Its actual generation will be the minimum of its default generation, and the generations of any previous theorems it subsumes. In any resolution attempt at least one of the two theorems will belong to the set of support, or be derived from an earlier resolution in the evaluation. Resolution will be attempted on subterms reached through outer universal quantification, and propositional connectives, by specialising the outer quantifications, and by type instantiation where necessary and allowed. Duplicates and pure specialisations in the resulting list will be discarded.

If any of the input theorems have \( \lceil F \rceil \) as a conclusion then that theorem is returned as a singleton list.

**Uses**  
On its own, or in combination with some canonicalisation of the input theorems.

**Errors**

- 67003 *The limit, \(?0, must be a positive integer*  
- 67004 *No resolution occurred*

---

#### basic_res_subsumption

| val basic_res_subsumption : THM -> THM -> int |

**Description**  
This returns 1 if the conclusion of the first theorem equals the second’s, or is a less general form than the second (i.e. could be produced only by specialising and type instantiating the second theorem). It returns 2 if the second theorem’s conclusion is a less general form than the first, and otherwise returns 0.
**val** basic_res_tac1 : int -> THM list -> TACTIC;

**Description** basic_res_tac1 limit thms (seqasms, conc) will take the theorems gained by asm-rule’ing the assumptions and thms as inputs. These theorems will be resolved against each other until only theorems with default generation past limit can be derived, or until \( \vdash F \) is derived, or until no further resolution can be done. An assumption’s or input theorem’s generation is 0, and a theorem that is the result of resolution has a default generation of 1 plus the sum of the generations of the resolved theorems. Its actual generation will be the minimum of its default generation, and the generations of any previous theorems it subsumes. In any resolution attempt at least one of the two theorems will be from the original goals assumptions, or be derived from an earlier resolution in the evaluation. Duplicates and pure specialisations will be discarded.

Resolution will be attempted on subterms reached through outer universal quantification, and propositional connectives, by specialising the outer quantifications, and by type instantiation on the input theorems thms where necessary and possible.

The resulting list of theorems will have all the thms removed, and all the old assumptions removed. MAP_EVERY strip_asm_tac is then applied to the new theorems, and then to the goal. As a special case, \( \vdash F \) is checked for, before any further processing. If present it will be used to prove the goal.

**Uses** On its own, or in combination with some canonicalisation of the input theorems.

**Errors**

67003  The limit, \(?0, must be a positive integer
67004  No resolution occurred

**val** basic_res_tac2 : int -> THM list -> TACTIC;

**Description** basic_res_tac2 limit thms (seqasms, conc) will first strip the negated goal into the assumption list. This uses strip_tac, except that the negation is pushed through all the outer universals. The assumptions derived from this will become the set of support, the pre-existing assumptions and the input thms will be the rest of the theorems. These theorems will be resolved against each other until only theorems with default generation past limit can be derived, or until \( \vdash F \) is derived, or until no further resolution can be done. An assumption’s or input theorem’s generation is 0, and a theorem that is the result of resolution has a default generation of 1 plus the sum of the generations of the resolved theorems. Its actual generation will be the minimum of its default generation, and the generations of any previous theorems it subsumes. In any resolution attempt at least one of the two theorems will be assumed fragments from the stripped goal, or be derived from an earlier resolution in upon those fragments. Duplicates and pure specialisations will be discarded.

Resolution will be attempted on subterms reached through outer universal quantification, and propositional connectives, by specialising the outer quantifications, and by type instantiation on the input theorems thms where necessary and possible.

The tactic will fail unless the resulting list of theorems contains \( \vdash F \). If present it will be used to prove the goal.

**Errors**

67003  The limit, \(?0, must be a positive integer
67004  No resolution occurred
67014  Failed to prove goal
7.6. First Order Resolution

SML

```sml
val basic_res_tac3 : int -> THM list -> TACTIC;
```

Description  

`basic_res_tac3 limit thms (seqasms, conc)` will take the theorems gained by `asm_rule`'ing the assumptions and `thms` as inputs. These theorems will be resolved against each other until only theorems with default generation past `limit` can be derived, or until `\vdash F` is derived, or until no further resolution can be done. An assumption’s or input theorem’s generation is 0, and a theorem that is the result of resolution has a default generation of 1 plus the sum of the generations of the resolved theorems. Its actual generation will be the minimum of its default generation, and the generations of any previous theorems it subsumes. In any resolution attempt at least one of the two theorems will be from the original goals assumptions, or be derived from an earlier resolution in the evaluation. Duplicates and pure specialisations will be discarded.

Resolution will be attempted on subterms reached through outer universal quantification, and propositional connectives, by specialising the outer quantifications, and by type instantiation on the input theorems `thms` where necessary and possible.

The tactic will fail unless the resulting list of theorems contains `\vdash F`. If present it will be used to prove the goal.

Errors  

67003 The limit, ?0, must be a positive integer
67004 No resolution occurred
67014 Failed to prove goal

SML

```sml
val basic_res_tac4 : int -> int list -> int list ->
    THM list -> THM list -> TACTIC;
```

Description  

`basic_res_tac4 limit sos rest sos_thms rest_thms (seqasms, conc)` will take the theorems gained by `asm_rule`'ing the numbered assumptions and `thms` as inputs. The “set of support” theorems will be those assumptions noted in the `sos` and those theorems in `sos_thms`, and “the rest” will be those assumptions noted in the `rest`, as well as `rest_thms`. These theorems will be resolved against each other until only theorems with default generation past `limit` can be derived, or until `\vdash F` is derived, or until no further resolution can be done. An assumption’s or input theorem’s generation is 0, and a theorem that is the result of resolution has a default generation of 1 plus the sum of the generations of the resolved theorems. Its actual generation will be the minimum of its default generation, and the generations of any previous theorems it subsumes. In any resolution attempt at least one of the two theorems will be from the set of support, or be derived from an earlier resolution in the evaluation. Duplicates and pure specialisations will be discarded.

Resolution will be attempted on subterms reached through outer universal quantification, and propositional connectives, by specialising the outer quantifications, and by type instantiation on the input theorems `thms` where necessary and possible.

The resulting list of theorems will have all the `thms` removed, and all the old assumptions removed. `MAP_EVERY strip_asm_tac` is then applied to the new theorems, and then to the goal. As a special case, `\vdash F` is checked for, before any further processing. If present it will be used to prove the goal.

Uses  

On its own, or in combination with some canonicalisation of the input theorems.

Errors  

67003 The limit, ?0, must be a positive integer
67004 No resolution occurred
9303 the index ?0 is out of range

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
val basic_res_tac : int -> THM list -> TACTIC;

**Description**  
`basic_res_tac limit thms (seqasms, conc)` will first strip the negated goal into the assumption list. This uses `strip_tac`, except that the negation is pushed through all the outer universals. The assumptions derived from this will become the set of support, the pre-existing assumptions and the input `thms` will be the rest of the theorems. These theorems will be resolved against each other until only theorems with default generation past `limit` can be derived, or until `... ⊢ F` is derived, or until no further resolution can be done. An assumption’s or input theorem’s generation is 0, and a theorem that is the result of resolution has a default generation of 1 plus the sum of the generations of the resolved theorems. Its actual generation will be the minimum of its default generation, and the generations of any previous theorems it subsumes. In any resolution attempt at least one of the two theorems will be assumed fragments from the stripped goal, or be derived from an earlier resolution in upon those fragments. Duplicates and pure specialisations will be discarded.

Resolution will be attempted on subterms reached through outer universal quantification, and propositional connectives, by specialising the outer quantifications, and by type instantiation on the input theorems `thms` where necessary and possible.

The resulting list of theorems will have all the `thms` removed, all the theorems derived from stripping and negating the goal removed, and all the old assumptions removed. `MAP_EVERY strip_asm_tac` is then applied to the new theorems, and then to the goal. As a special case, `... ⊢ F` is checked for, before any further processing. If present it will be used to prove the goal.

**Uses**  
On its own, or in combination with some canonicalisation of the input theorems.

**Errors**
- 67003 The limit, ?0, must be a positive integer
- 67004 No resolution occurred
7.6. First Order Resolution

SML

val prim_res_rule : 
  (THM list -> THM list -> (a list * a list * a list * b)) -> (* preprocessor *)
  (a -> a -> c) -> (* the resolver function *)
  (c * (a list * a list * a list * b)) ->
    ((a list * a list * a list * b) * bool) -> (* postprocessor *)
  (a list -> a list) -> (* next item to process *)
  (a list * a list * b -> THM list) -> (* extract results *)
  THM list -> (* input set of support theorems *)
  THM list -> (* input other theorems *)
  THM list; (* final outcome *)

Description  

prim_res_rule prep reso postp next extract limit sos rest works as follows:

- If any of the input theorems have "\( F \)" as a conclusion then that theorem is returned as a singleton list.
- Evaluate prep sos rest, and set (against, tried, toprocess, dbdata) to this.
- Attempt resolutions, choosing the head of toprocess against the head of against. Commonly, the head of toprocess should be the first fragment from the set of support, against is all the non-set of support fragments, plus the head of toprocess, and tried is empty.
- The resolver will usually return a list of theorems, and perhaps some further data. When a resolution attempt returns a list of theorems, res, (resolution failures should not occur, just []), evaluate postp (res, (against, tried, toprocess, dbdata)) to extract a new (against, tried, toprocess, dbdata), and halt. It is up to the postprocessor to move the head of against either to tried or just thrown away.
- If halt is true (e.g. have proved \( \vdash F \)), or the toprocess list is empty then return as a result of the call extract (against, tried, toprocess, dbdata).
- If halt is false, then continue with the new data. If against is [] then the head of toprocess is dropped, and the new list of things to process generated by next (tl toprocess), the new head of this cons’d to done and against is set to done reversed, and then done set to [].

Errors

67004  No resolution occurred
67010  Postprocessor corrupted processing
val term_unify : Unification.SSUBS -> (TYPE list) -> (TERM list) ->
(TERM * TERM list * TYPE list) *
(TERM * TERM list * TYPE list) ->
((TYPE * TYPE) list * (TERM * TERM) list) *
((TYPE * TYPE) list * (TERM * TERM) list);

Description  This is a method of unifying two subterms in the context of limitations on both
type instantiation and term specialisation. The SUBS argument is a “scratchpad” for the type
unification function, based on Unification.unify. The initial type list is a list of type variables to
avoid in generating new names, and the initial term list a list of term variables to likewise avoid.
The other two input arguments are each a tuple of: a term to unify, a list of variables in the term
that may be specialised, and a list of types for which instantiation is allowed. If the two terms
can be unified then the function returns two tuples, referring to each of the two input tuples.
Each tuple is a list of type instantiations and a list of term specialisations, which pair the original
before type instantiation, and the result, type instantiated.

Errors  
3007  ?0 is not a term variable
3019  ?0 is not a type variable
67005 Cannot unify ?0 and ?1
67006 Cannot unify ?0 and ?1 as cannot specialise ?2
67012 Cannot unify ?0 and ?1 as would cause a loop

As as errors of Unification.unify.
### 7.7 Proof Contexts

#### SML

```sml
signature ProofContext = sig
```

**Description** This provides the basic tools for handling equational and proof contexts. To keep them short, the names in the structure are heavily abbreviated. The abbreviations used are:

| pc(s)       | proof context(s) |
| rw          | rewriting        |
| cs          | constant specification |
| ∃           | existential theorem prover |
| pr          | prove_tac and related tools |
| sg          | goal stripping |
| st          | theorem stripping |
| cd          | clausal definition |
| vs          | variable structure |
| ad          | application data |
| net         | discrimination net |
| eqn_cxt     | equational context |
| nd          | dictionary of discrimination nets (and sources) |
| canon(s)    | theorem canonicalisation |
| mmp         | matching mp rule |
| eqn         | equation matcher |

```sml
(* proof context key "initial" *)
```

**Description** This is the initial proof context, formed with empty lists and other default values. It thus has no default rewriting or stripping theorems. The rewriting canonicalisation is the identity. The automated existence prover fails on any input. The matching modus ponens rule is Nil.

```sml
type EQN_CXT;
```

**Description** This is the type of equational contexts. An equational context is a list of conversions, each paired with term index. It represents a statement of how to rewrite a term to result in an equational theorem, guided by the outermost form of the term to be rewritten, which is matched against the term index of each conversion. It is used to create a single conversion via `eqn_cxt_conv` (q.v.).

A theorem may be converted into a member of an equational context by `thm_eqn_cxt`. A pre-existing conversion may be converted by determining the term index that matches at least all terms that the conversion must work on (see `net_enter` for details), and pair it with the conversion.

```sml
type EQN_CXT = (TERM * CONV) list;
```

Note that equational contexts can be merged by appending. An equational context may be transformed into a conversion discrimination net by `make_net` or `list_net_enter`(q.v.).
\textbf{asm\_prove\_tac : \textit{THM list} $\rightarrow$ \textit{TACTIC};}

\textbf{Description}  This tactic is an automatic proof procedure appropriate to the current proof context.

At the point of applying this tactic to its theorems it will access the current setting of proof context field \texttt{pr\_tac}, apply it to the theorem list immediately, and then to the goal when available (i.e. the result is partially evaluated with only the list of theorems).

\begin{align*}
\text{Tactic} & \quad \frac{}{\{ \Gamma \} \ t} \\
& \quad \text{asm\_prove\_tac} \\
& \quad \text{thms}
\end{align*}

\textbf{See Also} \texttt{PC\_T1} to defer accessing the proof context until application to the goal; \texttt{prove\_tac} for the form that does not react to the presence of assumptions.

\textbf{Errors} 51021 The current proof context was created in theory ?0 at a point now either not in scope, deleted or modified and as the proof context setting.

\textbf{asm\_prove\_∃\_tac : \textit{TACTIC};}

\textbf{Description}  This tactic is an automatic proof procedure for existential proofs, appropriate to the current proof context.

At the point of applying this tactic to a goal it will access the current setting of proof context field \texttt{prove\_∃}, apply it to the goal using \texttt{conv\_tac}.

\begin{align*}
\text{Tactic} & \quad \frac{}{\{ \Gamma \} \ t} \\
& \quad \text{asm\_prove\_∃\_tac} \\
& \quad \text{thms}
\end{align*}

\textbf{See Also} \texttt{prove\_∃\_tac} that does not react to any assumptions that are present.

\textbf{Errors} 51021 The current proof context was created in theory ?0 at a point now either not in scope, deleted or modified and as the proof context setting.

\textbf{commit\_pc : \textit{string} $\rightarrow$ \textit{unit};}

\textbf{Description}  This commits a record of the proof context database, preventing further change, and allowing it to be used in the creation of further records. The context must be loadable at the point of committing (i.e. was created at a point now in scope), and after committal the proof context can only be loaded at a point when the point of committal is in scope, rather than the point of its initial creation (i.e. doing \texttt{new\_pc}).

\textbf{Errors} 51010 There is no proof context with key ?0  
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified  
51016 Proof context ?0 has been committed
val current_ad_mmp_rule : unit -> (THM -> THM -> THM) OPT;

Description  This function returns the application data of the current proof context for the matching modus ponens rule as used by tools such as forward_chain_rule.

See Also  set_mmp_rule for user data.

Errors
51021 The current proof context was created in theory ?0 at a point now either not in scope, deleted or modified

val current_ad_pr_conv : unit -> (THM list -> CONV) OPT;

Description  These functions returns the application data of the current proof context to the proof contexts for prove_conv.

See Also  set_pr_conv for user data.

Errors
51021 The current proof context was created in theory ?0 at a point now either not in scope, deleted or modified

val current_ad_pr_tac : unit -> (THM list -> TACTIC) OPT;

Description  This function returns the application data of the current proof context for prove_tac.

See Also  set_pr_tac for user data.

Errors
51021 The current proof context was created in theory ?0 at a point now either not in scope, deleted or modified

val current_ad_rw_eqm_rule : unit -> (THM -> TERM * CONV) OPT;

Description  This function returns the application data of the current proof context for the equation matcher as used by the rewriting tools.

See Also  set_rw_eqm_rule for user data.

Errors
51021 The current proof context was created in theory ?0 at a point now either not in scope, deleted or modified
val delete_pc_fields : string list -> string -> unit;

Description  delete_pc_fields fields key empties (sets to the value of proof context “initial”) the named fields, fields of the proof context with key key. If any field is divided into subfields, this deletion includes deleting the subfields of the field gained from merging in other proof contexts, as well as the proof context’s “own” subfield.

Valid field names are:

"rw_eqn_cxt" , "rw_canons" , "st_eqn_cxt" , "sc_eqn_cxt" ,
"cs_∃_convs" , "∃_cd_thms" , "∃_vs_thms" , "pr_tac" , "pr_conv" ,
"nd_entries" , "mmp_rule"

Errors
51010  There is no proof context with key ?0
51014  Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified
51016  Proof context ?0 has been committed
51019  There is no field called ?0

val delete_pc : string -> unit;

Description  This deletes a record from the proof context database. The record with key “initial” may not be deleted.

Errors
51010  There is no proof context with key ?0
51012  Initial proof context may not be deleted

val eqn_cxt_conv : EQN_CXT -> CONV;

Description  This function creates a single conversion from an equational context. This is done via make_net and net_lookup(q,v). There is a CHANGED_C wrapped around each conversion in the equational context.

Errors
51005  Equational context gave no conversions that succeeded for ?0
7.7. Proof Contexts

SML

```sml
val EXTEND_PC_C1 : string -> (a -> CONV) -> a -> CONV;
val EXTEND_PCS_C1 : string list -> (a -> CONV) -> a -> CONV;
```

**Description**  
*EXTEND_PC_C* context *conv arg* will apply conversion *conv arg* in the proof context obtained by merging the proof context with key *context* into the current proof context. The named context is used as it is at the point of applying the rule to the argument. The *pr_tac*, *pr_conv* and *mpp_rule* fields are taken from the named proof context. This is done via pushing and popping on the proof context stack.

*EXTEND_PCS_C1* takes a list of proof contexts instead, merged as if by, e.g. *push_extend_pcs*.

**See Also**  
*PC_C*

**Errors**

51010  There is no proof context with key ?0
51014  Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified

and as the errors of the conversion. The previous proof context is restored, even if the conversion fails.

SML

```sml
val EXTEND_PC_C : string -> CONV -> CONV;
val EXTEND_PCS_C : string list -> CONV -> CONV;
```

**Description**  
*EXTEND_PC_C* context *conv* will apply conversion *conv* to a term in the proof context obtained by merging the proof context with key *context* into the current proof context. The named context is used as it is at the point of applying the conversion to a term. The *pr_tac*, *pr_conv* and *mpp_rule* fields are taken from the named proof context. This is done via pushing and popping on the proof context stack.

*EXTEND_PCS_C* takes a list of proof contexts instead, merged as if by, e.g. *push_extend_pcs*.

Note that when using this functions that the standard rewriting functions (obvious candidates for this function) access the current proof context at the point of being given their theorem list argument: see *EXTEND_PC_C1* for a method of avoiding this.

**Errors**

51010  There is no proof context with key ?0
51014  Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified

and as the errors of the conversion. The previous proof context is restored, even if the conversion fails.
val extend\_pc\_rule1 : string \rightarrow ('a \rightarrow 'b \rightarrow \text{THM}) \rightarrow 'a \rightarrow 'b \rightarrow \text{THM};
val extend\_pcs\_rule1 : string list \rightarrow ('a \rightarrow 'b \rightarrow \text{THM}) \rightarrow 'a \rightarrow 'b \rightarrow \text{THM};

**Description**

`extend\_pc\_rule1 context rule arg1 arg2` will apply rule `rule arg1` to `arg2` in the proof context obtained by merging the proof context with key `context` into the current proof context. The named context is used as it is at the point of applying the rule to the argument. The `pr\_tac`, `pr\_conv` and `mpp\_rule` fields are taken from the named proof context. This is done via pushing and popping on the proof context stack.

`extend\_pcs\_rule1` takes a list of proof contexts instead, merged as if by, e.g. `push\_extend\_pcs`.

**See Also**

`pc\_rule`

**Errors**

51010  There is no proof context with key ?0
51014  Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified

and as the errors of the rule. The previous proof context is restored, even if the rule fails.

val extend\_pc\_rule : string \rightarrow ('a \rightarrow \text{THM}) \rightarrow ('a \rightarrow \text{THM});
val extend\_pcs\_rule : string list \rightarrow ('a \rightarrow \text{THM}) \rightarrow ('a \rightarrow \text{THM});

**Description**

`extend\_pc\_rule context rule` will apply rule `rule` to its argument in the proof context obtained by merging the proof context with key `context` into the current proof context. The named context is used as it is at the point of applying the rule to the argument. The `pr\_tac`, `pr\_conv` and `mpp\_rule` fields are taken from the named proof context. This is done via pushing and popping on the proof context stack.

`extend\_pcs\_rule` takes a list of proof contexts instead, merged as if by, e.g. `extend\_merge\_pcs`.

Note that when using this functions that the standard rewriting functions (obvious candidates for this function) access the current proof context at the point of being given their theorem list argument: see `extend\_pc\_rule1` for a method of avoiding this.

**Errors**

51010  There is no proof context with key ?0
51014  Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified

and as the errors of the rule. The previous proof context is restored, even if the rule fails.
7.7. Proof Contexts

SML

| val EXTEND_PC_T1 : string -> 'a -> TACTIC -> 'a -> TACTIC; |
| val EXTEND_PCS_T1 : string list -> 'a -> TACTIC -> 'a -> TACTIC; |

Description EXTEND_PC_T1 context tac arg will apply tactic tac arg to a goal, and evaluate the proof, in the proof context obtained by merging the proof context with key context into the current proof context. The named context is used as it is at the point of applying the rule to the argument. The pr_tac, pr_conv and mpp_rule fields are taken from the named proof context. This is done via pushing and popping on the proof context stack.

EXTEND_PCS_T1 takes a list of proof contexts instead, merged as if by, e.g. push_extend_pcs.

See Also PC_T

Errors

51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified

and as the errors of the tactic. The previous proof context is restored, even if the tactic application or proof fails.

SML

| val EXTEND_PC_T : string -> TACTIC -> TACTIC; |
| val EXTEND_PCS_T : string list -> TACTIC -> TACTIC; |

Description EXTEND_PC_T context tac will apply tactic tac to a goal, and evaluate its proof, in the proof context obtained by merging the proof context with key context into the current proof context. The named context is used as it is at the point of applying the tactic to a goal. The pr_tac, pr_conv and mpp_rule fields are taken from the named proof context. This is done via pushing and popping on the proof context stack.

EXTEND_PCS_T_T takes a list of proof contexts instead, merged as if by, e.g. push_extend_pcs

Note that when using this functions that the standard rewriting functions (obvious candidates for this function) access the current proof context at the point of being given their theorem list argument: see EXTEND_PC_T1 for a method of avoiding this.

Errors

51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified

and as the errors of the tactic. The previous proof context is restored, even if the tactic application or proof fails.
val force_delete_theory : string -> unit;

Description force_delete_theory thy attempts to delete theory thy and all its descendants. If
thy is in scope, then the function will change the current theory to the first theory that it can in
the list returned by get_parents thy; (there may be none, in which case the function fails). It will
then determine whether thy and its descendants can all be deleted: in particular it checks that
none of them are locked (see lock_theory) or are a read-only ancestor.

The function indicates:

- whether the current theory has been deleted, and if so states the new current theory,
- the list of theories that have been deleted (unless this is just the requested theory, and is
  also not the current theory).

Further, all proof contexts created in now deleted theories will also be deleted (but the current
proof context will remain unchanged).

Errors

51002 Cannot open any of the parent theories, ?,0, of the named theory, ?1
51003 Will not be able to delete theories ?,0, so no deletions made
51004 Unexpectedly unable to delete any of ?,0
51006 Cannot open the parent theory, ?,0, of the named theory, ?1
51007 Will not be able to delete theory ?,0, so no deletions made
51008 Named theory, ?,0, has no parents

Error 51004 will be raised by error rather than fail, as it shouldn’t happen.

val get_current_pc : unit -> (string list * string);

Description Returns the key(s) of the entries from which the current proof context was copied,
and the theory in which the single proof context was created. If the theory has since been placed
out of scope, deleted or if the definition level becomes deleted, e.g. because an axiom or definition
has been deleted, then this will output

["context name","theory name (out of scope, deleted, or modified)"]

Note that the key may no longer access a proof context in the database identical to the current
proof context.

Merged proof contexts upon the stack (from push_merge_pcs and set_merge_pcs) will have the
list of names of the constituent proof contexts, singleton contexts will have singleton lists.

See Also get_stack_pcs

val get_pcs : unit -> (string * string) list;

Description This lists the names of the proof contexts held in the proof context database, and
the theory that was current at their time of creation. If the theory has since been deleted or if
the definition level becomes deleted, e.g. because an axiom or definition has been deleted, then
this will output ("context name","theory name (out of scope, deleted, or modified)"

See Also get_stack_pcs, get_current_pc.
7.7. Proof Contexts

SML

```sml
val get_stack_pcs : unit -> (string list * string) list;
```

**Description** This lists the keys of the proof contexts held in the proof context stack, and the theory that was current at their time of creation. If a proof context is pushed onto the stack by, e.g. `push_pc`, the “keys” will be the singleton list of the name of the source proof context. If a proof context is pushed onto the stack by, e.g. `push_merge_pcs`, the “keys” will be the list of the names of the source proof contexts. If the theory has since been deleted or if the definition level becomes deleted, e.g. because an axiom or definition has been deleted, then this will output

```sml
|(["context name"],"theory name (out of scope, deleted, or modified)"
```

The head of the list returned is the current proof context, as also displayed by `get_current_pc`.

SML

```sml
val merge_pcs : string list -> string -> unit;
```

**Description** `merge_pcs keys tokey` takes a list of committed proof contexts named by `keys`, and merges their fields into proof context `tokey`'s fields, discarding duplicates. For each field that has subfields the lists of subfields from each proof context are appended, discarding subfields with duplicate keys, and if a field is not divided into subfields, then the proof contexts fields are appended, discarding duplicates. The `pr_conv`, `pr_tac` and `mmp_rule` fields take the value of the last proof context in the list that has the field set.

Failure to extract any proof context for merging will result in the proof context `tokey` being unchanged.

**See Also** `merge_pc_fields`, `delete_pc_fields`

**Errors**

51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified
51016 Proof context ?0 has been committed
51017 Proof context ?0 has not been committed
val merge_pc_fields : {context:string, fields:string list} list -> string -> unit;

Description merge_pc_fields fields tokey merges the fields noted for each committed proof context in fields into proof context tokey's fields, discarding duplicates. Merging for each field that has subfields the lists of subfields is appending the proof contexts fields, discarding subfields with duplicate keys, and if a field is not divided into subfields, then the proof contexts fields are appended, discarding duplicates. Each of the pr_conv, pr_tac and mmp_rule fields take the value from the last proof context whose list of field names includes that field and which has the field set.

Failure to extract any proof context for merging will result in the proof context tokey being unchanged.

Valid field names are:

"rw_eqn_cxt","rw_canons","st_eqn_cxt","sc_eqn_cxt",
"cs_∃_convs","∃_cd_thms","∃_vs_thms","pr_tac","pr_conv",
"nd_entries","mmp_rule"

See Also delete_pc_fields and merge_pcs, which used together in a particular order can give the same functionality as this function.

Errors
51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified
51016 Proof context ?0 has been committed
51017 Proof context ?0 has not been committed
51019 There is no field called ?0

val new_pc : string -> unit;

Description new_pc new creates a new record in the proof context database, with key new. The fields of the proof context are set to default values. A note will be made of the current theory, and its current definition level at the time of creation, and an error will be raised if an attempt is made to push the new proof context (see push_pc) when that theory is not in scope, or when the definition level has been recorded as deleted. The definition level will be recorded as deleted if, e.g., some definition or axiom that was in scope in the original theory has since been deleted.

One responsibility of the creator of a proof context is to ensure that the theorems used within, or created by, the new context are also in scope: this is not automatically checked.

Errors
51011 There is already a proof context with key ?0
7.7. Proof Contexts

Description

PC_C context conv arg will apply conversion conv arg in the proof context with key context, using the named context as it is at the point of applying the conversion to a term. This is done via pushing and popping on the proof context stack.

MERGE_PCS_C1 takes a list of proof contexts instead, merged as if by, e.g. push_merge_pcs.

See Also

PC_C

Errors

51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified

and as the errors of the conversion. The previous proof context is restored, even if the conversion fails.

Description

PC_C context conv will apply conversion conv to a term in the proof context with key context, using the named context as it is at the point of applying the conversion to a term. This is done via pushing and popping on the proof context stack.

MERGE_PCS_C takes a list of proof contexts instead, merged as if by, e.g. push_merge_pcs

Note that when using this functions that the standard rewriting conversions (obvious candidates for this function) access the current proof context at the point of being given their theorem list argument: see PC_C1 for a method of avoiding this.

Errors

51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified

and as the errors of the conversion. The previous proof context is restored, even if the conversion fails.

Description

pc_rule context rule arg1 arg2 will apply rule rule arg1 to arg2 in the proof context with key context, using the named context as it is at the point of applying the rule to argument arg2. This is done via pushing and popping on the proof context stack.

merge_pcs_rule1 takes a list of proof contexts instead, merged as if by, e.g. push_merge_pcs.

See Also

pc_rule

Errors

51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified

and as the errors of the rule. The previous proof context is restored, even if the rule fails.
val pc_rule : string -> ('a -> THM) -> ('a -> THM);
val merge_pcs_rule : string list -> ('a -> THM) -> ('a -> THM);

Description  pc_rule context rule will apply rule rule to its argument in the proof context with
key context, using the named context as it is at the point of applying the rule to the argument.
This is done via pushing and popping on the proof context stack.

merge_pcs_rule takes a list of proof contexts instead, merged as if by, e.g. push_merge_pcs

Note that when using this functions that the standard rewriting functions (obvious candidates
for this function) access the current proof context at the point of being given their theorem list
argument: see pc_rule1 for a method of avoiding this.

Errors

51010  There is no proof context with key ?0
51014  Proof context ?0 was created in theory ?1 at a
   point now either not in scope, deleted or modified

and as the errors of the rule. The previous proof context is restored, even if the rule fails.

val PC_T1 : string -> ('a -> TACTIC) -> 'a -> TACTIC;
val MERGE_PCS_T1 : string list -> ('a -> TACTIC) -> 'a -> TACTIC;

Description  PC_T1 context tac arg will apply tactic tac arg to a goal, and evaluate the proof,
in the proof context with key context, using at both times the named context as it is at the point
of applying the tactic to a goal. This is done via pushing and popping on the proof context stack.

MERGE_PCS_T1 takes a list of proof contexts instead, merged as if by, e.g. push_merge_pcs.

See Also  PC_T

Errors

51010  There is no proof context with key ?0
51014  Proof context ?0 was created in theory ?1 at a
   point now either not in scope, deleted or modified

and as the errors of the tactic. The previous proof context is restored, even if the tactic application
or proof fails.

val PC_T : string -> TACTIC -> TACTIC;
val MERGE_PCS_T : string list -> TACTIC -> TACTIC;

Description  PC_T context tac will apply tactic tac to a goal, and evaluate its proof, in the
proof context with key context, using at both times the named context as it is at the point
of applying the tactic to a goal. This is done via pushing and popping on the proof context stack.

PCS_MERGE_T takes a list of proof contexts instead, merged as if by, e.g. push_merge_pcs

Note that when using this functions that the standard rewriting functions (obvious candidates
for this function) access the current proof context at the point of being given their theorem list
argument: see PC_T1 for a method of avoiding this.

Errors

51010  There is no proof context with key ?0
51014  Proof context ?0 was created in theory ?1 at a
   point now either not in scope, deleted or modified

and as the errors of the tactic. The previous proof context is restored, even if the tactic application
or proof fails.
7.7. Proof Contexts

**SML**

```sml
val pending_push_merge_pcs : string list -> unit -> unit;
val pending_push_extend_pcs : string list -> unit -> unit;
```

**Description**  
`pending_push_merge_pcs` takes a snapshot of the result of merging the named proof contexts, and returns a function that, when applied to () stacks the previous proof context, and and sets the current proof context of the system to this snapshot.

`pending_push_extend_pcs` takes a snapshot of the result of merging the named proof contexts with the current proof context and then behaves just like `pending_push_merge_pcs`.

Merged proof contexts upon the stack will have `current_ad_names` giving the list of names of the constituent proof contexts, singleton contexts will have singleton lists. The proof contexts used need not have been committed. The `pr_conv`, `pr_tac` and `mmp_rule` fields take the value of the last proof context in the list that has the field set.

The proof context must be in scope both at the time of the snapshot, and at the time of pushing on the stack.

This provides a method of being independent of changes to uncommitted proof contexts, or proof context deletions.

**See Also**  
push_merge_pc

**Errors**

- `51010` There is no proof context with key `?0`
- `51014` Proof context `?0` was created in theory `?1` at a point now either not in scope, deleted or modified
- `51020` Must be at least one key in list

---

**SML**

```sml
val pending_push_pc : string -> unit -> unit;
val pending_push_extend_pc : string -> unit -> unit;
```

**Description**  
`pending_push_pc` takes a snapshot of the named proof context, and returns a function that, when applied to () : unit stacks the previous “current” proof context, and sets the current proof context of the system to this snapshot.

`pending_push_extend_pc` takes a snapshot of the result of merging the named proof context with the current proof context and then behaves just like `pending_push_merge_pc`.

The proof context must be in scope both at the time of the snapshot, and at the time of pushing on the stack.

This provides a method of being independent of changes to uncommitted proof contexts, or proof context deletions.

**See Also**  
push_pc

**Errors**

- `51010` There is no proof context with key `?0`
- `51014` Proof context `?0` was created in theory `?1` at a point now either not in scope, deleted or modified


### pending_reset_pc_database

```sml
val pending_reset_pc_database : unit -> unit -> unit;
```

**Description**  
This function, applied to () takes a snapshot of the proof context database, and returns a function that, if applied to () will restore the proof context database to the snapshot.

This function is particularly useful in initialising child databases, and in conjunction with `pending_reset_pc_stack` and `pending_reset_pc_evaluators`.

Note that a named proof context on the proof context stack is never taken as more than an echo of the item with that name (if any) of proof context database, and this function in particular, though not alone, is responsible for the possible differences.

### pending_reset_pc_stack

```sml
val pending_reset_pc_stack : unit -> unit -> unit;
```

**Description**  
This function, applied to () takes a snapshot of the proof context stack, and returns a function that, if applied to () will restore the proof context stack to the snapshot.

**Uses**  
This function is particularly useful in initialising child databases, and in conjunction with `pending_reset_pc_database` and `pending_reset_pc_evaluators`.

### pending_reset_pc_evaluators

```sml
val pending_reset_pc_evaluators : unit -> unit -> unit;
```

**Description**  
This function, applied to () takes a snapshot of the proof context evaluators (e.g. the one set by `pp/set_eval_ad_\_\_vs_thms`, and returns a function that, if applied to () will restore the proof context evaluators to the snapshot.

**Uses**  
This function is particularly useful in initialising child databases, and in conjunction with `pending_reset_pc_database` and `pending_reset_pc_stack`.

### pop_pc

```sml
val pop_pc : unit -> unit;
```

**Description**  
This function unstacks the top of the proof context stack, and sets the current proof context of the system to it. There will always be a current proof context, though it may be the trivial “initial” proof context.

This function may make an out of scope proof context the current proof context.

**See Also**  
`push_pc`, `set_pc`, `push_merge_pcs`, `set_merge_pcs`

**Errors**  

51001 The proof context stack is empty

### pp/set_eval_ad rw_net

```sml
val pp/set_eval_ad_rw_net : (EQN_CXT -> CONV NET) -> unit;

val current_ad_rw_net : unit -> CONV NET;
```

**Description**  
These functions provide the interface to the initial conversion net for rewriting (see e.g. `rewrite_tac`) held in the application data of a proof context. The first sets the evaluator, the second extracts the field in the current proof context.

**See Also**  
`set_rw_eqn_cxt` for the associated user data.

**Errors**  

51021 The current proof context was created in theory ?0 at a point now either not in scope, deleted or modified

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
7.7. Proof Contexts

SML

val pp'set_eval_ad_rw_canon : ((THM -> THM list) list -> (THM -> THM list))
  -> unit;
val current_ad_rw_canon : unit -> THM -> THM list;

Description  These functions provide the interface to the canonicalisation function applied to
rewriting theorems (see e.g. rewrite_tac) held in the application data of a proof context. The
proof context is accessed after providing the theorem. The first sets the evaluator, the second
extracts the field in the current proof context.

See Also  set_rw.canons for the associated user data.

Errors
51021  The current proof context was created in theory ?0 at a
  point now either not in scope, deleted or modified

SML

val pp'set_eval_ad_st_conv : (EQN.CXT -> CONV) -> unit;
val current_ad_st_conv : unit -> CONV;

Description  These functions provide the interface to the conversion for stripping theorems into
the assumption list (see e.g. strip_tac) held in the application data of a proof context. The proof
context is accessed before provision of a term. The first sets the evaluator, the second extracts
the field in the current proof context.

See Also  set_st_conv for the associated user data.

Errors
51021  The current proof context was created in theory ?0 at a
  point now either not in scope, deleted or modified

SML

val pp'set_eval_ad_sc_conv : (EQN.CXT -> CONV) -> unit;
val current_ad_sc_conv : unit -> CONV;

Description  These functions provide the interface to the conversion for stripping goal conclusions
(see e.g. strip_tac) held in the application data of a proof context. The proof context is
accessed before provision of a term. The first sets the evaluator, the second extracts the field in
the current proof context.

See Also  set_sg_conv for the associated user data.
These functions provide the interface to the additional dictionary of discrimination nets held in the application data of a list of proof contexts.

The application data is generated by taking, for each key in at least one of the dictionaries in the appropriate subfields of the proof context, the appended lists of all the entries for that key in any of the subfields of the proof context. To this is applied the evaluator set by \texttt{pp\,\prime\,set\_eval\_ad\_nd\_net} first applied to the dictionary key. The result is used as an entry, using the same dictionary key, in the resulting dictionary of nets. The default evaluator will just use \texttt{make\_net} on each list of sources.

\texttt{current\_ad\_nd\_net key} returns the net indexed by the key \texttt{key} in the current proof context. If no entry exists it returns the empty net \texttt{empty\_net}. Note that the returned net can be viewed as something of type \texttt{EQN\_CXT}, and made into a conversion by \texttt{eqn\_cxt\_conv}.

\textbf{Uses} For extending the proof context mechanisms. Though available to the end user, and indeed intended for use by the sophisticated user, the proof context mechanisms (as opposed to proof contexts) should be extended under ICL direction.

\textbf{See Also} \texttt{set\_nd\_entry} for the associated user data.

These functions provide the interface to the existence prover for constant specifications (see \texttt{const\_spec}) held in the application data of a proof context. The proof context is accessed before provision of a term. The first sets the evaluator, the second extracts the field in the current proof context.

\textbf{See Also} \texttt{set\_cs\_\exists\_rule} for the associated user data.

\textbf{Errors} 51015 No automated existence prover in the current proof context succeeds 51021 The current proof context was created in theory \texttt{?0} at a point now either not in scope, deleted or modified

These functions provide the interface to the clausal definition theorem information for the existence prover \texttt{prove\_\exists\_conv}. See \texttt{evaluate\_\exists\_cd\_thms} for details upon the form of the information. The first sets the evaluator, the second extracts the field in the current proof context.

\textbf{See Also} \texttt{set\_\exists\_cd\_thms} for the associated user data.

\textbf{Errors} 51021 The current proof context was created in theory \texttt{?0} at a point now either not in scope, deleted or modified
7.7. Proof Contexts

```sml
val pp/set_eval_ad_∃_vs_thms : ((string * (TERM list * THM)) list ->
  (string * (TERM list * THM)) list) -> unit;
val current_ad_∃_vs_thms : unit ->
  (string * (TERM list * THM)) list;
```

**Description** These functions provide the interface to the application data variable structure information for the existence prover prove_∃_conv. The first sets the evaluator, the second extracts the field in the current proof context.

**See Also** set_∃_vs_thms for user data.

**Errors**

51021 The current proof context was created in theory ?0 at a point now either not in scope, deleted or modified

```sml
val prove_conv : THM list -> CONV;
```

**Description** This conversion is an automatic proof procedure appropriate to the current proof context.

At the point of applying this conversion to its theorems it will access the current setting of proof context field pr_conv, applying the result to the theorem list immediately, and then to the term when available (i.e. the result is partially evaluated with only the list of theorems).

**Conversion**

```
current_ad_pr_conv () thms \(\vdash t\) →
prove_conv thms \(\vdash t\)
```

**See Also** PC_C1 to defer accessing the proof context until application to the term.

**Errors**

51021 The current proof context was created in theory ?0 at a point now either not in scope, deleted or modified and as the proof context setting.

```sml
val prove_rule : THM list -> TERM -> THM;
```

**Description** This rule is an automatic proof procedure appropriate to the current proof context.

At the point of applying this rule to its theorem list it will access the current setting of proof context field pr_conv, apply it to the theorem list immediately, and then to the term when available (i.e. the result is partially evaluated with only the list of theorems), and then, if the resulting theorem is \(\vdash \text{term} \leftrightarrow T\) (with no assumptions) where term is \(\alpha\)-convertible to term’, then apply \(\leftrightarrow_t\)elim, and otherwise fail.

**Rule**

```
\vdash tm →
prove_rule thms \(\vdash tm\)
```

**See Also** pc_rule1 to defer accessing the proof context until application to the term.

**Errors**

51021 The current proof context was created in theory ?0 at a point now either not in scope, deleted or modified

51022 Result of applying conversion to ?0, which was ?1,
  not of form: \(\vdash input \leftrightarrow T\)

and as the proof context setting.
| val prove_∃_conv : CONV; |

**Description**  This conversion is an automatic proof procedure for existential proofs, appropriate to the current proof context.

At the point of applying this conversion to a term it will access the current setting of proof context field cs_∃_conv, apply it to the theorem list, and then to the term.

The resulting theorem is not checked as having its L.H.S. being the input term.

<table>
<thead>
<tr>
<th>Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_ad_cs_∃_conv () ⊢ t</td>
</tr>
<tr>
<td>prove_∃_conv ⊢ t</td>
</tr>
</tbody>
</table>

**Errors**

51021 The current proof context was created in theory ?0 at a point now either not in scope, deleted or modified and as the proof context setting.

---

| val prove_∃_rule : TERM → THM; |

**Description**  This rule is an automatic proof procedure for existential proofs, appropriate to the current proof context.

At the point of applying this rule to a term term it will access the current setting of proof context field cs_∃_conv, apply it to the term, and then, if the resulting theorem is ⊢ term' ⇔ T (with no assumptions) where term is α-equivalent to term', then apply ⇔ t_elim, and otherwise fail.

<table>
<thead>
<tr>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊢ tm</td>
</tr>
<tr>
<td>prove_∃_rule ⊢ tm'</td>
</tr>
</tbody>
</table>

**Errors**

51021 The current proof context was created in theory ?0 at a point now either not in scope, deleted or modified 51022 Result of applying conversion to ?0, which was ?1, not of form: ‘⊢ input ⇔ T’ and as the proof context setting.
\begin{verbatim}
val push_extend_pcs : string list -> unit;
val set_extend_pcs : string list -> unit;

Description push_extend_pcs stacks the previous “current” proof context, and and then merges the proof contexts with the given keys into the current proof context. set_extend_pcs merges the proof contexts with the given keys into the previous current proof context without changing the stack.

Merged proof contexts upon the stack will have current_ad_names giving the list of names of the constituent proof contexts, singleton contexts will have singleton lists. The proof contexts used need not have been committed.

The pr_conv, pr_tac and mmp_rule fields take the value of the last proof context in the list that has the field set.

The current proof context is accessed by the functions prefixed current_ad_, and by get_current_pc.

See Also pop_pc, push_merge_pcs, set_merge_pcs
\end{verbatim}

\begin{verbatim}
val push_extend_pc : string -> unit;
val set_extend_pc : string -> unit;

Description push_extend_pc stacks the previous “current” proof context, and and then merges the proof context with the given key into the current proof context. set_extend_pcs merges the proof context with the given key into the current proof context without changing the stack.

Merged proof contexts upon the stack will have current_ad_names giving the list of names of the constituent proof contexts. The proof context used need not have been committed.

The pr_conv, pr_tac and mmp_rule fields take the value from the named proof context.

The current proof context is accessed by the functions prefixed current_ad_, and by get_current_pc.

See Also pop_pc, push_pc, set_pc
\end{verbatim}
val push_merge_pcs : string list -> unit;
val set_merge_pcs : string list -> unit;

**Description**  
`push_merge_pcs` stacks the previous “current” proof context, and and sets the current proof context of the system to the merge of the proof contexts with the given keys.  
`set_merge_pcs` discards the previous “current” proof context, and and sets the current proof context of the system to the merge of the proof contexts with the given keys.  
Merged proof contexts upon the stack will have `current_ad_names` giving the list of names of the constituent proof contexts, singleton contexts will have singleton lists. The proof contexts used need not have been committed.

The `pr_conv`, `pr_tac` and `mmp_rule` fields take the value of the last proof context in the list that has the field set.

The current proof context is accessed by the functions prefixed `current_ad_`, and by `get_current_pc`.

**See Also**  
`pop_pc`, `push_pc`, `set_pc`

**Errors**
- 51010  There is no proof context with key ?0
- 51014  Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified
- 51020  Must be at least one key in list

val push_pc : string -> unit;
val set_pc : string -> unit;

**Description**  
`push_pc` stacks the previous “current” proof context, and and sets the current proof context of the system to the proof context with the given key.  
`set_pc` discards the previous “current” proof context, and and sets the current proof context of the system to the proof context with the given key.

The current proof context is accessed by the functions prefixed `current_ad_`, and by `get_current_pc`.

**See Also**  
`pending_push_pc`, `pop_pc`, `push_merge_pcs`, `set_merge_pcs`

**Errors**
- 51010  There is no proof context with key ?0
- 51014  Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified
SML

val set_cs∃convs : (CONV list) ->
  string -> unit;
val get_cs∃convs : string ->
  (((CONV list) * string) list);

Description These functions provide the interface to the existence provers for constant specifications (see const_spec) held in the user data of a proof context. Under the initial evaluator, the existence proving conversion supplied by current_cs∃conv will have each of the conversions tried, in the reverse order of their entry, being applied to the RHS of the result of the previous successful application, or the initial term to which the conversion was applied, until the RHS is `⌜T⌝`, or no conversions remain.

Example If get_cs∃convs of the current proof context returns

`[((conv1, conv2),"pc1"),(conv3, conv4),"pc2")]

Then current_ad_cs∃conv will return

```
conv4 AND OR C conv3 AND OR C conv2 AND OR C conv1
```

“setting” overwrites the subfield whose key is the proof context’s name, “getting” returns the entire field (which pairs data with proof context names).

Errors

51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified
51016 Proof context ?0 has been committed

SML

val set_mmp_rule : (THM -> THM -> THM) -> string -> unit;
val mmp_rule : string -> (THM -> THM -> THM) OPT;

Description These functions provide the interface to the proof contexts for the matching modus ponens rule as used by tools such as forward_chain_rule. Note that setting overwrites all previous data in this field, including from merged in proof contexts. Merged proof contexts take their value for this field from the last proof context in the list that has the field set.

Errors

51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified
51016 Proof context ?0 has been committed
**SML**

```sml
val set_nd_entry : string -> \( \text{TERM} * (\text{TERM} \to \text{THM}) \) list -> string -> unit;
val get_nd_entry : string -> string ->
    \( \text{((TERM} * (\text{TERM} \to \text{THM})\) list} \) * string list;
```

**Description**  These functions provide the interface to the additional dictionary of sources for discrimination nets held in the user data of a proof context. The dictionary is actually a list of subfields of the proof context, indexed by source proof context name, each subfield being a dictionary in its own right. You “set” a single dictionary entry of the subfield indexed by the proof context’s name (creating a new entry if necessary). You “get” the dictionaries for all the subfields.

**Errors**

- 51010  There is no proof context with key ?0
- 51014  Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified
- 51016  Proof context ?0 has been committed

**SML**

```sml
val set_pr_conv : \( \text{THM list} \to \text{CONV} \) -> string -> unit;
val get_pr_conv : string -> \( \text{THM list} \to \text{CONV} \);
val get_pr_conv1 : string -> \( \text{THM list} \to \text{CONV} \) OPT;
```

**Description**  These functions provide the interface to the proof contexts for prove_conv. Note that setting overwrites all previous data in this field, including from merged in proof contexts. If the field has not been set, get_pr_conv returns a function mapping any list of theorems to fail_conv and get_pr_conv1 returns Nil. Merged proof contexts take their value for this field from the last proof context in the list that has the field set.

Note that when using these functions that the standard rewriting functions (obvious candidates for inclusion in automatic proof) access the current proof context at the point of being given their theorem list argument.

**Errors**

- 51010  There is no proof context with key ?0
- 51014  Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified
- 51016  Proof context ?0 has been committed
### Proof Contexts

**SML**

```sml
val set_pr_tac : (THM list -> TACTIC) -> string -> unit;
val get_pr_tac : string -> (THM list -> TACTIC);
val get_pr_tac1 : string -> (THM list -> TACTIC) OPT;
```

**Description** These functions provide the interface to the proof contexts for `prove_tac`. Note that setting overwrites all previous data in this field, including from merged in proof contexts. If the field has not been set, `get_pr_tac` returns a function mapping any list of theorems to `fail_tac` and `get_pr_tac1` returns `Nil`. Merged proof contexts take their value for this field from the last proof context in the list that has this field set.

When `asm_prove_tac` is applied to its theorem list argument the system will evaluate this by applying the value set by `set_pr_tac` for the current proof context to that argument. The provided values for `set_pr_tac` can interpret their theorem list arguments as they wish (e.g. as a set of rewrite theorems, or as theorems to resolve against) - no interpretation is forced upon this argument.

Note that when using these functions that the standard rewriting functions (obvious candidates for inclusion in automatic proof) access the current proof context at the point of being given their theorem list argument.

**Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>51010</td>
<td>There is no proof context with key ?0</td>
</tr>
<tr>
<td>51014</td>
<td>Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified</td>
</tr>
<tr>
<td>51016</td>
<td>Proof context ?0 has been committed</td>
</tr>
</tbody>
</table>

**SML**

```sml
val set_rw_canons : (THM -> THM list) list ->
                   string -> unit;
val get_rw_canons : string -> ((THM -> THM list) list * string) list;
```

**Description** These functions provide the interface to the individual canonicalisation functions used to create the canonicalisation function applied to rewriting theorems (see e.g. `rewrite_tac`) held in the user data of a proof context.

“setting” overwrites the subfield whose key is the proof context’s name, “getting” returns the entire field (which pairs data with proof context names).

**Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>51010</td>
<td>There is no proof context with key ?0</td>
</tr>
<tr>
<td>51014</td>
<td>Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified</td>
</tr>
<tr>
<td>51016</td>
<td>Proof context ?0 has been committed</td>
</tr>
</tbody>
</table>
val set_rw_eqm_rule : (THM -> TERM * CONV) -> string -> unit;
val get_rw_eqm_rule : string -> (THM -> TERM * CONV) OPT;

Description  These functions provide the interface to the proof contexts for the equation matcher as used by the rewriting tools. Note that setting overwrites all previous data in this field, including from merged in proof contexts. Merged proof contexts take their value for this field from the last proof context in the list that has the field set.

Errors
51010  There is no proof context with key ?0
51014  Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified
51016  Proof context ?0 has been committed

val set_rw_eqn_cxt : EQU_CXT -> string -> unit;
val get_rw_eqn_cxt : string -> (EQU_CXT * string) list;
val add_rw_thms : THM list -> string -> unit;

Description  These functions provide the interface to the equational context for rewriting (see e.g. rewrite_tac) held in the user data of a proof context. “setting” overwrites the subfield whose key is the proof context’s name, “getting” returns the entire field (which pairs data with proof context names). “adding” processes its theorems by first canonicalising according to the current proof context’s canonicalisation function, and then with thm_eqn_cxt and then adds them into the subfield whose key is the proof context’s name.

Errors
51010  There is no proof context with key ?0
51014  Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified
51016  Proof context ?0 has been committed

val set_sc_eqn_cxt : EQU_CXT -> string -> unit;
val get_sc_eqn_cxt : string -> (EQU_CXT * string) list;
val add_sc_thms : THM list -> string -> unit;

Description  These functions provide the interface to the equational context for stripping goal conclusions (see e.g. strip_tac) held in the user data of a proof context. “setting” overwrites the subfield whose key is the proof context’s name, “getting” returns the entire field (which pairs data with proof context names). “adding” processes its theorems by first canonicalising according to the current proof context’s canonicalisation function, and then with thm_eqn_cxt and then adds them into the subfield whose key is the proof context’s name.

Errors
51010  There is no proof context with key ?0
51014  Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified
51016  Proof context ?0 has been committed
### Proof Contexts

<table>
<thead>
<tr>
<th>SML</th>
<th>val set_st_eqn_cxt : EQN_CTX -&gt; string -&gt; unit;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>val get_st_eqn_cxt : string -&gt; (EQN_CTX * string)list;</td>
</tr>
<tr>
<td></td>
<td>val add_st_thms : THM list -&gt; string -&gt; unit;</td>
</tr>
</tbody>
</table>

#### Description
These functions provide the interface to the equational context for stripping theorems into the assumption list (see e.g. `strip_tac`) held in the user data of a proof context. “setting” overwrites the subfield whose key is the proof context’s name, “getting” returns the entire field (which pairs data with proof context names). “adding” processes its theorems by first canonicalising according to the current proof context’s canonicalisation function, and then with `thm_eqn_ctxt` and then adds them into the subfield whose key is the proof context’s name.

#### Errors

| 51010 | There is no proof context with key ?0 |
| 51014 | Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified |
| 51016 | Proof context ?0 has been committed |

<table>
<thead>
<tr>
<th>SML</th>
<th>val set_∃_cd_thms : THM list -&gt; string -&gt; unit;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>val get_∃_cd_thms : string -&gt; THM list;</td>
</tr>
<tr>
<td></td>
<td>val add_∃_cd_thms : THM list -&gt; string -&gt; unit;</td>
</tr>
</tbody>
</table>

#### Description
These functions provide the interface to the unevaluated clausal definition theorems held for the existence prover `prove_∃_conv`. There are no subfields to this field, so “setting” overwrites the field with the proof context’s name, “getting” returns the field. “adding” unions its theorem list with the proof contexts field.

#### See Also
See `evaluate_∃_cd_thms` for details upon the form of the theorems.

#### Errors

| 51010 | There is no proof context with key ?0 |
| 51014 | Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified |
| 51016 | Proof context ?0 has been committed |
val set_∃_vs_thms : (string * (TERM list * THM)) list → string → unit;
val get_∃_vs_thms : string → (string * (TERM list * THM)) list;

Description  These functions provide the interface to the variable structure information for the
existence prover prove_∃_conv. An individual entry in the list gives a method of handling an
extended variable structure. It consists of the name of the constructor; a list of functions that
each field of the constructor, and a theorem that states how the extraction functions
extract from a data construction, and that the data constructor may be applied to the extracted
values to regain the original value. For instance, for pairs the information is:

```
("", ",
(["Fst"],"Snd"),
\forall x y p .
Fst (x, y) = x \land Snd (x, y) = y \land
(Fst p, Snd p) = p)
```

There are no subfields to this field, so “setting” overwrites the field with the proof context’s name,
“getting” returns the field.

Errors
51010  There is no proof context with key ?0
51014  Proof context ?0 was created in theory ?1 at a
      point now either not in scope, deleted or modified
51016  Proof context ?0 has been committed

val simple_ho_thm_eqn_cxt : THM → (TERM * CONV);

Description  This function is an equation matcher for use by the rewriting tools that uses
higher-order matching. It transforms an equational theorem into a representation of a higher-
order rewrite rule in a form suitable for inclusion in an equational context (EQN_CXT q.v.)

```
\Gamma \vdash \forall x_1 ... \bullet LHS = RHS →
(LHS', simple_eq_match_conv1 \Gamma \vdash \forall x_1 ... \bullet LHS = RHS')
```

Here the pattern term LHS' is derived from LHS by replacing linear patterns (see simple_ho_match) by variables of the same type.

The universal quantifiers must be over simple variables (not patterns) and the higher-order matching is done using simple_ho_match.

See Also  cthm_eqn_cxt which canonicalises the theorem before transformation.

Errors
7095  ?0 is not of the form \( \Gamma \vdash \forall x_1 ... x_n \bullet u = v \) where \( x_i \) are variables
val thm_eqn_cxt : THM -> (TERM * CONV);

**Description**  This function is a simple form of equation matcher for use by the rewriting tools. It transforms an equational theorem into a representation of a first-order rewrite rule in a form suitable for inclusion in an an equational context (EQQN.CXT q.v.)

```
thm_eqn_cxt `\Gamma \vdash x_1 \ldots \bullet \text{LHS} = \text{RHS}` →
(LHS, simple_eq_match_conv1 `\Gamma \vdash x_1 \ldots \bullet \text{LHS} = \text{RHS}`)
```

The universal quantifiers must be over simple variables (not patterns).

**See Also**  cthm_eqn_cxt which canonicalises the theorem before transformation.

**Errors**

7095  `?0 is not of the form `\Gamma \vdash \forall x_1 \ldots x_n \bullet u = v` where `x_i` are variables

---

SML

```
signature ProofContexts1 = sig

**Description**  This signature gives access to two functions used in supplying the first group of proof contexts. Proof contexts themselves have no entry in the signature, however the contexts provided are:

<table>
<thead>
<tr>
<th>Component</th>
<th>Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>simple_abstractions</code></td>
<td>predicates</td>
</tr>
<tr>
<td><code>paired_abstractions</code></td>
<td>predicates1</td>
</tr>
<tr>
<td><code>propositions</code></td>
<td>basic_hol</td>
</tr>
<tr>
<td><code>fun_ext</code></td>
<td>basic_hol1</td>
</tr>
<tr>
<td><code>pair</code></td>
<td>sets_ext</td>
</tr>
<tr>
<td><code>pair1</code></td>
<td>hol</td>
</tr>
<tr>
<td><code>N</code></td>
<td>hol1</td>
</tr>
<tr>
<td><code>N_lit</code></td>
<td></td>
</tr>
<tr>
<td><code>list</code></td>
<td></td>
</tr>
<tr>
<td><code>char</code></td>
<td></td>
</tr>
<tr>
<td><code>sum</code></td>
<td></td>
</tr>
<tr>
<td><code>one</code></td>
<td></td>
</tr>
<tr>
<td><code>combin</code></td>
<td></td>
</tr>
<tr>
<td><code>sets_alg</code></td>
<td></td>
</tr>
<tr>
<td><code>sets_ext</code></td>
<td></td>
</tr>
<tr>
<td><code>basic_prove_∃_conv</code></td>
<td></td>
</tr>
</tbody>
</table>
```

---

SML

```
(* Proof Context: `basic_prove_∃_conv *)

**Description**  A component proof context that adds the function `basic_prove_∃_conv` as an automatic existence prover.

**Contents**  Automatic proof procedures are respectively “always fail tactic”, “always fail conversion”, and `basic_prove_∃_conv`.

**Usage Notes**  Requires theory “basic_hol”, intended to be combined into the merge of any component proof contexts that do not have their own special existence prover. It should usually be the first in the list of proof contexts to be merged together, so that other proof contexts may introduce pre-processors, and then the final default prover is invoked. This is because the standard application of the list of existence prover conversions is defined to be to apply them in a cumulative manner, in reverse order.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
### Chapter 7. PROOF IN HOL

#### Description
A component proof context for handling only simple abstractions in stripping and canonicalisation.

#### Contents
Rewriting:  

- **Stripping theorems:**  
  - `simple \( \neg \) in conv`  

- **Stripping conclusions:**  
  - `simple \( \neg \) in conv`  

- **Rewriting canonicalisation:**  
  - `simple \( \forall \) rewrite canon, simple \( \neg \) rewrite canon`

Automatic proof procedures are respectively `basic_prove_tac`, `basic_prove_conv` and no existence prover.

#### Usage Notes
Not to be used with proof context "paired_abstractions" as their "domains" overlap. It requires theory `basic_hol`.

---

#### Description
A component proof context for handling simple and paired abstractions in stripping and canonicalisation.

#### Contents
Rewriting:  

- **\( \beta \) conv**  

- **Stripping theorems:**  
  - `\( \neg \) in conv, \( \exists \) in conv, \( \forall \) uncurry_conv, \( \exists \) uncurry_conv`  

- **Stripping conclusions:**  
  - `\( \neg \) in conv, \( \forall \) uncurry_conv`  

- **Rewriting canonicalisation:**  
  - `\( \forall \) rewrite canon, \( \neg \) rewrite canon`

Automatic proof procedures are respectively `basic_prove_tac`, `basic_prove_conv` and no existence prover.

#### Usage Notes
Not to be used with proof context "simple_abstractions", as their "domains" overlap. It requires theory `basic_hol`. 
7.7. Proof Contexts

SML

\(* Proof Context: \text{propositions} *)

**Description**  A component proof context for reasoning about propositions.

**Contents**  Rewriting:

- eq\_rewrite\_thm, \(\iff\)\_rewrite\_thm, \(\neg\)\_rewrite\_thm,
- \(\land\)\_rewrite\_thm, \(\lor\)\_rewrite\_thm, \(\Rightarrow\)\_rewrite\_thm,
- if\_rewrite\_thm, \(\forall\)\_rewrite\_thm, \(\exists\)\_rewrite\_thm,
- \(\beta\)\_rewrite\_thm, simple\_\(\beta\)\_conv

Stripping theorems:

- \(\Rightarrow\)\_thm, \(\iff\)\_thm, simple\_\(\exists\)\_conv,
- \(\vdash \forall x \bullet ((x = x) \iff T)\),'
- \(\vdash \forall x \bullet (\neg(x = x) \iff F)\),'
- \(\vdash \forall a t1 t2\bullet (\text{if } a \text{ then } t1 \text{ else } t2) \iff (a \Rightarrow t1) \land (\neg a \Rightarrow t2)\)'

Note these are intended to be used with (simple\_) \(\neg\)\_in\_conv from "paired\_abstractions" or "simple\_abstractions", which covers the cases of an outermost \(\neg\) for each operator.

Stripping conclusions:

- \(\iff\)\_thm,
- \(\vdash \forall x \bullet ((x = x) \iff T)\),'
- \(\vdash \forall x \bullet (\neg(x = x) \iff F)\),'
- \(\vdash \forall a t1 t2\bullet (\text{if } a \text{ then } t1 \text{ else } t2) \iff (a \Rightarrow t1) \land (\neg a \Rightarrow t2)\)'
- \(\vdash \forall a b\bullet (a \lor \neg b) \iff (b \Rightarrow a)\)'
- \(\vdash \forall a b\bullet (\neg a \lor b) \iff a \Rightarrow b\)'
- \(\vdash \forall a b\bullet a \lor b \iff \neg a \Rightarrow b\)'

Note that the above are intended to be used in combination with (simple\_) \(\neg\)\_in\_conv from "paired\_abstractions" or "simple\_abstractions", which covers the cases of an outermost \(\neg\) for each operator.

Rewriting canonicalisation:

- \(\land\)\_rewrite\_canon, f\_rewrite\_canon

Automatic proof procedures are respectively taut\_tac, taut\_conv and basic\_prove\_\(\exists\)\_conv.

**Usage Notes**  Usually used in conjunction with "paired\_abstractions" or "simple\_abstractions", requires theory basic\_hol.
### Proofs in HOL

#### fun_ext

**Description** A component proof context for adding reasoning using functional extensionality.

**Contents**
- **Rewriting**
  - `ext.thm`
- **Stripping Theorems**
  - `ext.thm`
- **Stripping Conclusions**
  - `ext.thm`
- **Rewriting Canonicalisation**

Automatic proof procedures are, respectively, `taut_tac`, `taut_conv` and `basic_prove_∃.conv`.

**Usage Notes** Normally used in conjunction with “propositions”, requires theory `basic_hol`.

#### predicates

**Description** A “mild” complete proof context for reasoning about the predicate calculus, including paired abstractions.

**Contents**
- Proof contexts “basic_prove_∃.conv”, “paired_abstractions” and “propositions”.

Automatic proof procedures are respectively `basic_prove_tac`, `basic_prove_conv` and `basic_prove_∃.conv` (merged in from the proof context of the same name).

**Usage Notes** Requires theory `basic_hol`.

#### predicates1

**Description** An “aggressive” complete proof context for reasoning about the predicate calculus, including paired abstractions and functional extensionality.

**Contents**
- Proof contexts “basic_prove_∃.conv”, “paired_abstractions”, “propositions” and “fun_ext”.

Automatic proof procedures are, respectively, `basic_prove_tac`, `basic_prove_conv` and `basic_prove_∃.conv` (merged in from the proof context of the same name).

**Usage Notes** Requires theory `basic_hol`.

---

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
7.7. Proof Contexts

SML

\((^\ast \text{ Proof Context: 'pair *})\)

**Description**  A “mild” component proof context for theory `pair`.

**Contents**  Rewriting (selected from `pair_clauses`):

\[\vdash \forall x \ y \ a \ b \ p \ fu \ fc\]
\[\bullet \ \text{Fst} \ (x, \ y) = x\]
\[\land \ \text{Snd} \ (x, \ y) = y\]
\[\land \ ((a, \ b) = (x, \ y) \iff a = x \land b = y)\]
\[\land \ (\text{Fst} \ p, \ \text{Snd} \ p) = p\]
\[\land \ \text{Curry} \ fc \ x \ y = fc \ (x, \ y)\]
\[\land \ \text{Uncurry} \ fu \ (x, \ y) = fu \ x \ y\]
\[\land \ \text{Uncurry} \ fu \ p = fu \ (\text{Fst} \ p) \ (\text{Snd} \ p)\]

Stripping theorems:

\[\vdash \forall a \ b \ x \ y \bullet ((a, \ b) = (x, \ y) \iff a = x \land b = y)\]

Stripping conclusions:

\[\vdash \forall a \ b \ x \ y \bullet ((a, \ b) = (x, \ y) \iff a = x \land b = y)\]

Existential variable structures:

\[\vdash \forall x \ y \ p \bullet\]
\[\text{Fst} \ (x, \ y) = x \land\]
\[\text{Snd} \ (x, \ y) = y \land\]
\[ (\text{Fst} \ p, \ \text{Snd} \ p) = p\]

Automatic proof procedures are respectively `basic_prove_tac`, `basic_prove_conv` and no existence prover.

**Usage Notes**  Requires theory `basic_hol`.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
## Chapter 7. PROOF IN HOL

### SML

| (* Proof Context: 'pair1 *) |

**Description** An “aggressive” component proof context for theory `pair`.

**Contents** Rewriting:

\[
\forall a \ b \ p \\
\bullet ((a, b) = p \iff a = Fst p \land b = Snd p) \\
\land (p = (a, b) \iff Fst p = a \land Snd p = b)'
\]

Stripping theorems (selected from `pair_clauses`):

\[
\forall a \ b \ p \\
\bullet ((a, b) = p \iff a = Fst p \land b = Snd p) \\
\land (p = (a, b) \iff Fst p = a \land Snd p = b)'
\]

Stripping conclusions:

\[
\forall a \ b \ p \\
\bullet ((a, b) = p \iff a = Fst p \land b = Snd p) \\
\land (p = (a, b) \iff Fst p = a \land Snd p = b)'
\]

Existential variable structures:

Automatic proof procedures are respectively `basic_prove_tac`, `basic_prove_conv` and no existence prover.

**Usage Notes** Requires theory `basic_hol`, expected to be used in combination with “pair”.

### SML

| (* Proof Context: 'N *) |

**Description** A “mild” component proof context for theory `N`.

**Contents** Rewriting:

\[
\geq_{def}, \ greater_{def}, \ plus_{clauses}, \ times_{clauses}, \\
\leq_{clauses}, \ less_{clauses}, \ minus_{clauses}
\]

Stripping theorems:

\[
\geq_{def}, \ greater_{def}, \ plus_{clauses}, \ times_{clauses}, \\
\leq_{clauses}, \ less_{clauses}, \ minus_{clauses}, \\
\text{and all boolean equations also pushed through } \neg$
\]

Stripping conclusions:

\[
\geq_{def}, \ greater_{def}, \ plus_{clauses}, \ times_{clauses}, \\
\leq_{clauses}, \ less_{clauses}, \ minus_{clauses}, \\
\text{and all boolean equations also pushed through } \neg$
\]

Existential clausal definition theorems:

\[ prim_{rec_thm} \]

Automatic proof procedures are respectively `basic_prove_tac`, `basic_prove_conv` and no existence prover.

**Usage Notes** Requires theory `basic_hol`. 

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
Proof Contexts

(\texttt{Proof Context: }\texttt{\textasciicircum N\_lit }\texttt{*)}

\textbf{Description} A component proof context for theory N, that will, e.g., evaluate any arithmetic expression involving only numeric literals and certain arithmetic operators, namely +, *, -, Div, Mod, \(\leq\), <, >, \(\geq\), and =.

\textbf{Contents} Rewriting:

\begin{itemize}
  \item plus\_conv, times\_conv, minus\_conv, div\_conv,
  \item mod\_conv, \(\leq\)\_conv, less\_conv, greater\_conv,
  \item \(\geq\)\_conv, N\_eq\_conv
\end{itemize}

Stripping theorems:

\begin{itemize}
  \item \(\leq\)\_conv, less\_conv, greater\_conv,
  \item \(\geq\)\_conv, N\_eq\_conv
\end{itemize}

Stripping conclusions:

\begin{itemize}
  \item \(\leq\)\_conv, less\_conv, greater\_conv,
  \item \(\geq\)\_conv, N\_eq\_conv
\end{itemize}

Existential clausal definition theorems:

Automatic proof procedures are respectively basic\_prove\_tac, basic\_prove\_conv and no existence prover.

\textbf{Usage Notes} Requires theory basic\_hol, expected to be used with proof context “\textasciicircum N”. It is separated from it as spotting the application of the conversions is time consuming, and may be known to be irrelevant.
Chapter 7. PROOF IN HOL

(* Proof Context: 'list *)

Description  A component proof context for the theory list.

Contents  Rewriting:

| list_clauses

Stripping theorems:

\[ \forall x_1 x_2 \text{ list1 list2} \]
\[ \neg \text{Cons } x_1 \text{ list1} = [] \]
\[ \land \neg [] = \text{Cons } x_1 \text{ list1} \]
\[ \land (\text{Cons } x_1 \text{ list1} = \text{Cons } x_2 \text{ list2} \iff x_1 = x_2 \land \text{list1} = \text{list2}) \]

Stripping conclusions:

\[ \forall x_1 x_2 \text{ list1 list2} \]
\[ \neg \text{Cons } x_1 \text{ list1} = [] \]
\[ \land \neg [] = \text{Cons } x_1 \text{ list1} \]
\[ \land (\text{Cons } x_1 \text{ list1} = \text{Cons } x_2 \text{ list2} \iff x_1 = x_2 \land \text{list1} = \text{list2}) \]

Existential clausal definition theorems:

| list_prim_rec_thm

Automatic proof procedures are respectively basic_prove_tac, basic_prove_conv and no existence prover.

Usage Notes  Requires theory list.

(* Proof Context: 'char *)

Description  A component proof context for theory char, for reasoning about character and string literals.

Contents  Rewriting:

| char_eq_conv, string_eq_conv

Stripping theorems:

| char_eq_conv, string_eq_conv

Stripping conclusions:

| char_eq_conv, string_eq_conv

Automatic proof procedures are respectively basic_prove_tac, basic_prove_conv and an existence prover preprocessor that rewrites with \( \vdash "" = [] \) which assists using list’s primitive induction on strings.

Usage Notes  Requires theory basic_hol.
### Proof Contexts

<table>
<thead>
<tr>
<th>Proof Context: <code>basic_hol</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Contents</strong></td>
</tr>
<tr>
<td><strong>Usage Notes</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proof Context: <code>basic_hol1</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Contents</strong></td>
</tr>
<tr>
<td><strong>Usage Notes</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proof Context: <code>mmp1</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Usage Notes</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proof Context: <code>mmp2</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Usage Notes</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proof Context: <code>sho_rw</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Usage Notes</strong></td>
</tr>
</tbody>
</table>
A "mild" component proof context for theory *sum*.

**Contents**

**Rewriting:**

\[ \{ *\} \vdash \forall x_1 \ x_2 \ y_1 \ y_2 \ z \]
\[ \bullet (\text{InL} \ x_1 = \text{InL} \ x_2 \iff x_1 = x_2) \]
\[ \land (\text{InR} \ y_1 = \text{InR} \ y_2 \iff y_1 = y_2) \]
\[ \land \neg \text{InL} \ x_1 = \text{InR} \ y_1 \]
\[ \land \neg \text{InR} \ y_1 = \text{InL} \ x_1 \]
\[ \land \text{OutL} (\text{InL} \ x_1) = x_1 \]
\[ \land \text{OutR} (\text{InR} \ y_1) = y_1 \]
\[ \land \text{IsL}(\text{InL} \ x_1) \land \text{IsR}(\text{InR} \ y_1) \]
\[ \land \neg \text{IsL}(\text{InR} \ y_1) \land \neg \text{IsR}(\text{InL} \ x_1) \]

**Stripping theorems:**

\[ \{ *\} \vdash \forall x_1 \ x_2 \ y_1 \ y_2 \ z \]
\[ \bullet (\text{InL} \ x_1 = \text{InL} \ x_2 \iff x_1 = x_2) \]
\[ \land (\text{InR} \ y_1 = \text{InR} \ y_2 \iff y_1 = y_2) \]
\[ \land \neg \text{InL} \ x_1 = \text{InR} \ y_1 \]
\[ \land \neg \text{InR} \ y_1 = \text{InL} \ x_1 \]
\[ \land \text{IsL}(\text{InL} \ x_1) \land \text{IsR}(\text{InR} \ y_1) \]
\[ \land \neg \text{IsL}(\text{InR} \ y_1) \land \neg \text{IsR}(\text{InL} \ x_1) \]

**Stripping conclusions:**

\[ \{ *\} \vdash \forall x_1 \ x_2 \ y_1 \ y_2 \ z \]
\[ \bullet (\text{InL} \ x_1 = \text{InL} \ x_2 \iff x_1 = x_2) \]
\[ \land (\text{InR} \ y_1 = \text{InR} \ y_2 \iff y_1 = y_2) \]
\[ \land \neg \text{InL} \ x_1 = \text{InR} \ y_1 \]
\[ \land \neg \text{InR} \ y_1 = \text{InL} \ x_1 \]
\[ \land \text{IsL}(\text{InL} \ x_1) \land \text{IsR}(\text{InR} \ y_1) \]
\[ \land \neg \text{IsL}(\text{InR} \ y_1) \land \neg \text{IsR}(\text{InL} \ x_1) \]

**Existential clausal definition theorems:**

\[ \{ *\} \vdash f \cdot g \cdot h \cdot (\forall \ x \bullet h \cdot (\text{InL} \ x) = f \ x) \land (\forall \ x \bullet h \cdot (\text{InR} \ x) = g \ x) \]

Automatic proof procedures are respectively `basic_prove_tac`, `basic_prove_conv` and no existence prover.

**Usage Notes**

Requires theory *sum*.
7.7. Proof Contexts

SML

(* Proof Context: 'one *)

Description  A component proof context for theory one

Contents  Rewriting (these both have the problem that their discrimination net entry will match anything):

| one_def, one_fns_thm

Stripping theorems:

\[
\begin{align*}
\forall x y : \text{ONE} \bullet (x = y) & \iff T' \\
\forall x y : 'a \rightarrow \text{ONE} \bullet (x = y) & \iff T'
\end{align*}
\]

and through \neg

Stripping conclusions:

\[
\begin{align*}
\forall x y : \text{ONE} \bullet (x = y) & \iff T' \\
\forall x y : 'a \rightarrow \text{ONE} \bullet (x = y) & \iff T'
\end{align*}
\]

and through \neg

Automatic proof procedures are respectively basic_prove_tac, basic_prove_conv and no existence prover.

Usage Notes  Requires theory one. As when entered into the rewriting net the rewriting theorems will match any term presented to the net, this proof context will slow down rewriting.

SML

(* Proof Context: 'combin *)

Description  A component proof context for theory combin

Contents  Rewriting:

| comb_i_def, comb_k_def, o_def, o_i_thm

Stripping theorems:

Stripping conclusions:

Automatic proof procedures are, respectively, basic_prove_tac, basic_prove_conv and no existence prover.

Usage Notes  Requires theory combin.
证明上下文：'sets_alg'

描述：一个“温和”的组件证明上下文用于理论 set。

内容：重写：
∈_comp_conv, ∈_enum_set_conv, complement_clauses,
∪_clauses, ∩_clauses, set_dif_clauses, ⊕_clauses,
⊆_clauses, ⊂_clauses, union_clauses,
∪_clauses, ∩_clauses, \( P \)_clauses
\( \forall x \ y \bullet \neg x \in \{ \} \)
\( \land x \in \) Universe
\( \land (x \in \{ y \} \iff x = y) \)

去角定理：
∈_comp_conv, ∈_enum_set_conv, ∈_in_clauses
⊆_clauses, ⊂_clauses
这些全部推入通过 ¬

去角结论：
∈_comp_conv, ∈_enum_set_conv, ∈_in_clauses
⊆_clauses, ⊂_clauses
这些全部推入通过 ¬

自动证明程序分别是 basic_prove_tac, basic_prove_conv 和存在性证明预处理器：
TOP_MAP_C (all_∃_uncurry_conv AND_OR_C sets_simple_∃_conv)

说明：该上下文不与证明上下文 "sets_ext" 共同使用，要求理论 sets。

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
### Proof Context: `sets_ext`

**Description** A component proof context for theory `set`, “aggressively” using the extensionality of sets.

**Contents** Rewriting:

| ∈_comp_conv, ∈_enum_set_conv, ∈_in_clauses, sets_ext_clauses |

Stripping theorems:

| ∈_comp_conv, ∈_enum_set_conv, ∈_in_clauses, sets_ext_clauses |
| plus these all pushed in through ¬ |

Stripping conclusions:

| ∈_comp_conv, ∈_enum_set_conv, ∈_in_clauses, sets_ext_clauses |
| plus these all pushed in through ¬ |

Automatic proof procedures are respectively `basic_prove_tac`, `basic_prove_conv` and the existence prover preprocessor:

| TOP_MAP.C (all_∃.uncurry_conv AND OR.C sets.simple_∃.conv) |

The preprocessor causes set membership (∈) to be treated as function application in some cases.

**Usage Notes** Should not be used with proof context “`sets_alg`”, requires theory `sets`.

### Proof Context: `sets_ext1`

**Description** A complete proof context for reasoning about sets within the predicate calculus, “aggressively” using the extensionality of sets.

**Contents** Proof contexts “`sets_ext`” and “`predicates`”.

**Usage Notes** Requires theory `sets`. The proof context “`sets_ext1`” offers a much more useful treatment of sets of pairs.
Chapter 7. PROOF IN HOL

SML

(* Proof Context: 'sets_ext1 *)

Description A component proof context for theory set, including sets of pairs, “aggressively” using the extensionality of sets.

Contents Rewriting:

\[\varepsilon_{\text{comp.conv}}, \varepsilon_{\text{enum.set.conv}}, \varepsilon_{\text{in.clauses}},\]
\[\text{sets.eq.conv}, \subseteq_{\text{conv}}, \subset_{\text{conv}}\]

Stripping theorems:

\[\varepsilon_{\text{comp.conv}}, \varepsilon_{\text{enum.set.conv}}, \varepsilon_{\text{in.clauses}},\]
\[\text{sets.eq.conv}, \subseteq_{\text{conv}}, \subset_{\text{conv}}\]
\[\text{plus these all pushed in through } \neg\]

Stripping conclusions:

\[\varepsilon_{\text{comp.conv}}, \varepsilon_{\text{enum.set.conv}}, \varepsilon_{\text{in.clauses}},\]
\[\text{sets.eq.conv}, \subseteq_{\text{conv}}, \subset_{\text{conv}}\]
\[\text{plus these all pushed in through } \neg\]

Automatic proof procedures are respectively basic_prove_tac, basic_prove_conv and the existence prover preprocessor:

\[\text{TOP.MAP.C (all}_3 \text{uncurry.conv AND OR } C \text{ sets.simple}_3 \text{.conv)}\]

The preprocessor causes set membership (\(\varepsilon\)) to be treated as function application in some cases.

Usage Notes Should not be used with proof context “sets_alg”, requires theory sets.

SML

(* Proof Context: sets_ext *)

Description A complete proof context for reasoning about sets within the predicate calculus, “aggressively” using the extensionality of sets.

Contents Proof contexts “sets_ext” and “predicates”.

Usage Notes Requires theory sets.

SML

(* Proof Context: sets_ext1 *)

Description A complete proof context for reasoning about sets, including sets of pairs, within the predicate calculus, “aggressively” using the extensionality of sets.

Contents Proof contexts “sets_ext1” and “predicates”.

Usage Notes Requires theory sets. The proof context “sets_ext1” offers a much more useful treatment of sets of pairs.

SML

(* Proof Context: hol *)

Description A “mild” complete proof context for the ancestors of theory hol

Contents Proof contexts “basic_hol”, “sum”, “combin”, and “sets_alg”.

Automatic proof procedures are respectively basic_prove_tac, basic_prove_conv and basic_prove_\(\exists\)_conv (merged in from the proof context of the same name).

Usage Notes Requires theory hol.
### Proof Contexts

**SML**

<table>
<thead>
<tr>
<th>(* Proof Context: hol1 *)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Contents</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Usage Notes</strong></td>
</tr>
</tbody>
</table>

**SML**

<table>
<thead>
<tr>
<th>(* Proof Context: hol2 *)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Contents</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Usage Notes</strong></td>
</tr>
</tbody>
</table>
**Description**  This is the conversion used for the automatic proof conversion \((pr\_tac\ field)\) of most supplied proof contexts, and is a reasonable, general-purpose, automatic proof conversion. It will either prove the theorem with the given conclusion, or fail.

In summary it will:

1. Set the term as the goal of the subgoal package.

2. Attempt to rewrite the term with the current default rewrite rules and given theorems.

3. Repeatedly apply \(strip\_tac\) to the goal.

4. Try \(all\_var\_elim\_asm\_tac\) to do variable elimination.

5. Attempt to prove the resulting goals with resolution for up to 3 resolution steps, with goal’s negated conclusion as a resolvant that must be used, and the assumptions as possible other resolvants. This has no effect on any resulting goal if it is unsolved.

6. Attempt to prove the resulting goals with resolution for up to 3 resolution steps amongst just the assumptions. This has no effect on any resulting goal if it is unsolved.

7. If the proof is successful, return \(\vdash \text{term} \iff T\) and otherwise fail.

Note that in the stripping step may result in more than one subgoal, and thus the plural “resulting goals”.

Under the current interface to resolution this equivalent to:

```ml
fun basic_prove_conv thms tm =
  t_intro (tac_proof(([],tm),
    TRY T (rewrite_tac thms) THEN
    REPEAT strip_tac THEN_TRY
    (basic_res_tac2 3 [\vdash \forall x \bullet x = x]
     ORELSE_T basic_res_tac3 3 [\vdash \forall x \bullet x = x])))
```

In the implementation however, partial evaluation with just the theorems is allowed.

**Errors**

76001  Could not prove theorem with conclusion ?0
val basic_prove_tac : THM list -> TACTIC;

Description This is the tactic used for the automated proof tactic (the pr_tac field) of most supplied proof contexts, and is a reasonable, general-purpose, automatic proof tactic.

In summary it will:

1. Try all_var_elim_asm_tac to do variable elimination.
2. Extract the assumption list, rewrite each extracted assumption with the current default rewrite rules and given theorems, and strip the results back into the assumption list.
3. Attempt to rewrite the resulting goal’s conclusions with the current default rewrite rules and given theorems.
4. Again try all_var_elim_asm_tac to do variable elimination.
5. Repeatedly apply strip_tac to the conclusions of the resulting goals.
6. Attempt to prove each of the resulting goals with resolution for up to 3 resolution steps, with goal’s negated conclusion as a resolvant that must be used, and the assumptions as possible other resolvants. This has no effect on any resulting goal if it is unsolved.
7. Attempt to prove each of the resulting goals with resolution for up to 3 resolution steps amongst just the assumptions. This has no effect on any resulting goal if it is unsolved.

Note that either stripping step may result in more than one subgoal, and thus the plural “resulting goals”.

Under the current interface to resolution this is

fun basic_prove_tac thms =
TRY_T (DROP_ASM (TRY_T (TRY_T (DROP_ASM (REPEAT strip_tac THEN TRY
(basic_res_tac2 3 (\[\forall x : x = x\])) ORELSE_T basic_res_tac3 3 (\[\forall x : x = x\]))))))
8.1 Syntactic Manipulations

In the following descriptions of derived term constructors for Z it has been convenient to describe the effects of constructors using Z language quotations. In doing so quotations have sometimes been used which would not in fact be acceptable to the Z parser. The most frequent example of these is in quoting the declaration part of variable binding constructs in Z. The Z parser will not accept such declarations in isolation from the variable binding construct of which they form a part, but the most readable description of the effect of the constructor is obtained if we describe this as if the parser did accept such declarations in isolation.

In practice the best way of obtaining the term corresponding to the declaration part of such a construct is to parse a horizontal schema containing the required declaration part, and then take it apart using the appropriate destructor.

```
SML
signature ZTypesAndTerms = sig

Description
The Z Abstract Machine functions are packaged into this signature.
```

```
SML
datatype BDZ
  = BdzOk of Z_TERM
  | BdzNotZ of int
  | BdzFail of {
    BdzFCode : int,
    BdzFCompc : int,
    BdzFArgc : int
  }

Description
The return value from function basic_dest_z_term. The BdzFail constructor gives information primarily intended for use by the Z pretty printer.

See Also
Function basic_dest_z_term.
```
### Chapter 8. SUPPORT FOR Z

#### Z_TERM =
- \( \text{ZDec} \) of \( \text{TERM list} \) * \( \text{TERM} \)
- \( \text{ZDecl} \) of \( \text{TERM list} \)
- \( \text{ZEq} \) of \( \text{TERM} \) * \( \text{TERM} \)
- \( \text{ZTrue} \)
- \( \text{Z} \neg \) of \( \text{TERM} \)
- \( \text{Z} \wedge \) of \( \text{TERM} \) * \( \text{TERM} \)
- \( \text{Z} \equiv \) of \( \text{TERM} \) * \( \text{TERM} \)
- \( \text{Z} \Rightarrow \) of \( \text{TERM} \) * \( \text{TERM} \)
- \( \text{Z} \exists \) of \( \text{TERM} \) * \( \text{TERM} \)
- \( \text{Z} \exists_1 \) of \( \text{TERM} \) * \( \text{TERM} \)
- \( \text{Z} \forall \) of \( \text{TERM} \) * \( \text{TERM} \)
- \( \text{ZSchemaPred} \) of \( \text{TERM} \) * \( \text{string} \)
- \( \text{ZLVar} \) of \( \text{string} \) * \( \text{TYPE} \) * \( \text{TERM list} \)
- \( \text{ZSetd} \) of \( \text{TYPE} \) * \( \text{TERM list} \)
- \( \text{ZTuple} \) of \( \text{TERM list} \)
- \( \text{ZBinding} \) of \( \text{(string} * \text{TERM)} \) list
- \( \text{Z} \times \) of \( \text{TERM list} \)
- \( \text{ZSel} \) of \( \text{TERM} \) * \( \text{string} \)
- \( \text{ZSel}_s \) of \( \text{TERM} \) * \( \text{string} \)
- \( \text{ZApp} \) of \( \text{TERM} \) * \( \text{TERM} \)
- \( \text{ZLet} \) of \( \text{(string} * \text{TERM)} \) list * \( \text{TERM} \)
- \( \text{ZHSchema} \) of \( \text{TERM} \) * \( \text{TERM} \)
- \( \text{ZDecor} \) of \( \text{TERM} \) * \( \text{string} \)
- \( \text{Zs} \) of \( \text{TERM} \) * \( \text{TERM} \)
- \( \text{ZΔ} \) of \( \text{TERM} \)
- \( \text{ZRename} \) of \( \text{TERM} \) * \( \text{(string} * \text{string)} \) list

**Description**
This datatype corresponds to a version of the abstract syntax of Z in which recursion has been removed and the distinction between declarations, predicates and terms ignored. It is used by the generalised mapping functions \( \text{mk}_\text{Z_TERM}, \text{is}_\text{Z_TERM} \) and \( \text{dest}_\text{Z_TERM} \) (q.v.).

#### Z_TYPE =
- \( \text{ZGivenType} \) of \( \text{string} \)
- \( \text{ZVarType} \) of \( \text{string} \)
- \( \text{ZPowerType} \) of \( \text{TYPE} \)
- \( \text{ZTupleType} \) of \( \text{TYPE list} \)
- \( \text{ZSchemaType} \) of \( \text{(string} * \text{TYPE)} \) list

**Description**
This datatype is a representation of the abstract syntax of Z types. It is used by the generalised mapping functions \( \text{mk}_\text{Z_TYPE}, \text{is}_\text{Z_TYPE} \) and \( \text{dest}_\text{Z_TYPE} \) (q.v.). The operand of \( \text{ZGivenType} \) is the HOL name of the type.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
8.1. Syntactic Manipulations

SML

val basic_dest_z_term : TERM * TERM list -> BDZ;

Description Function basic_dest_z_term does the work of destroying a term to yield its Z structure. The arguments are in the result of applying strip_app to a term.

A call of ‘basic_dest_z_term(strip_app zt)’ will attempt to destroy the Z term zt, if successful (i.e., zt is a valid Z term) then BdzOk is returned with the appropriate Z_TERM value. If zt is not a valid Z term then one of the other BDZ constructors is returned, these include an error code indicating what was wrong with the term. A BdzFail is returned when the term is similar to a Z term (i.e., it has a known constructor but the wrong number of arguments). In this case the BdzFCompc and BdzFArgc fields tell how many component lists and arguments (respectively) are allowed in a well formed Z term. A BdzNotZ is returned when the term is not recognisable as a Z term. In cases where insufficient component lists or arguments are given to a known constructor either BdzFail or BdzNotZ may be returned.

All of the error codes of function dest_z_term may be returned by this function.

See Also Functions: dest_z_term and strip_app; and, datatype BDZ.

SML

val dest_z_name1 : string -> string * string OPT;
val dest_z_name2 : string -> string OPT -> string list list * string OPT;

Description Supplying dest_z_name2 with the result of dest_z_name1 gives the same overall result as dest_z_name q.v. These functions allow the destruction of the component names and projection part to be deferred for efficiency, in case they are not required.

Errors

47000 ?0 is not a Z constant name

SML

val dest_z_name : string -> string * string list list * string OPT;

Description Analyses the names of Z semantic constants, returning the basic name and lists of embedded component names. If the name is a projection, then the projection part is also returned.

Errors

47000 ?0 is not a Z constant name
**val dest_z_term : TERM -> ZTERM**;

**Description** Converts a HOL term, which represents a valid Z term, to the appropriate ZTERM.

**See Also** dest_z_term1 which makes a more careful check, especially of schema constructs.

**Errors**
- 47900 ?0 is not a Z term
- 47901 ?0 is not a Z package
- 47910 ?0 is not a Z simple declaration
- 47911 ?0 is not a Z schema declaration
- 47912 ?0 is not a Z declaration
- 47920 ?0 is not a Z existential quantification
- 47921 ?0 is not a Z unique existential quantification
- 47922 ?0 is not a Z universal quantification
- 47923 ?0 is not a Z schema as a predicate
- 47930 ?0 is not a Z set comprehension
- 47931 ?0 is not a Z θ term
- 47932 ?0 is not a Z function application
- 47936 ?0 is not a Z λ abstraction
- 47937 ?0 is not a Z let expression
- 47940 ?0 is not a Z schema
- 47941 ?0 is not a Z schema existential quantification
- 47942 ?0 is not a Z schema unique existential quantification
- 47943 ?0 is not a Z schema universal quantification

---

**val dest_z_type : TYPE -> ZTYPE**;

**Description** Converts a HOL type, which represents a valid Z type, to the appropriate ZTYPE.

**Errors**
- 47800 ?0 is not a Z type

---

**val gvar_subst : TERM -> (TERM * TERM) list**;

**Description** Given an arbitrary term, \( t \), \( gvar\_subst \) creates a substitution mapping those free variables of \( t \) (in the HOL sense) which have the same names as Z global variables (i.e. HOL constants) in the current scope to the appropriate instances of those global variables (with generic instantiation using \( \text{U} \) as necessary). The resulting substitution may then be used with subst, q.v., to “bind” the term into the current scope.

---

**val is_z_term : TERM -> bool**;

**Description** Tests if a given HOL term is valid Z in its top level structure.

**Uses** Recursively in well-formedness checks.

**See Also** is_z_term1 for a more complete check of top level structure, is_z for a full traversal of the terms structure.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
8.1. Syntactic Manipulations

SML

val is_z_type : TYPE -> bool;

Description Tests if a given HOL type represents a valid Z type.

Uses Recursively in well-formedness checks.

SML

val mk_dollar_quoted_string : string -> string;
val dest_dollar_quoted_string : string -> string;
val is_dollar_quoted_string : string -> bool;

Description The Z parser allows an arbitrary ML character string to be used to form an identifier. These functions implement the encoding used to embed an arbitrary ML string in the name of a Z variable:

Example

mk_dollar_quoted_string"<ext-name>" = "$<ext-name>""  
dest_dollar_quoted_string"$<ext-name>"" = "<ext-name>"  
is_dollar_quoted_string"$"<ext-name>"" = true  
is_dollar_quoted_string""<ext-name>"" = false

Errors

47001 ?0 is not a valid dollar-quoted string

SML

val mk_u : TYPE -> TERM;
val is_u : TERM -> bool;
val dest_u : TERM -> TYPE;

Description These functions create, test for, and destroy terms of the form $\mathbb{U}[Totality]$ which are used by the Z type inferrer to stand for elided generic actual parameters. The type parameter to mk_u and the result of dest_u is the type of the U-term in question.

Errors

47950 ?0 is not of the form $\mathbb{U}[Totality]$  
47951 ?0 is not an instance of $\mathbb{U}a$ SET

SML

val mk_z_app : TERM * TERM -> TERM;
val is_z_app : TERM -> bool;
val dest_z_app : TERM -> TERM * TERM;

Description Z function application. The first argument must be a set of pairs, the second must have the same type as the first elements of the pairs.

Definition

$\text{mk}_z\text{app}(\langle f, a \rangle) = \langle f(a) \rangle$

Errors

47190 ?0 is not a Z function application  
47191 ?0 has the wrong type to be a Z function  
47192 ?0 has the wrong type to be applied to ?1
val mk_z_binding : (string * TERM) list -> TERM;
val is_z_binding : TERM -> bool;
val dest_z_binding : TERM -> (string * TERM) list;

Description  The binding constructor.

Definition  \[ mk_z_binding [("n_1", \langle Z t_1 \rangle),...,("n_n", \langle Z t_n \rangle)] = \langle Z (n_1 \equiv t_1,\ldots,n_n \equiv t_n) \rangle \]

Errors  47151  ?0 is not a Z binding
        47152  Cannot bind more than one value to ?0

val mk_z_decl : TERM list -> TERM;
val is_z_decl : TERM -> bool;
val dest_z_decl : TERM -> TERM list;

Description  Constructor, discriminator and destructor functions for the declaration part of a schema text. Its arguments must be made using \( mk_z_dec \) or \( mk_z_schema_dec \).

Definition  \[ mk_z_decl [\langle Z t_1 \rangle,\ldots,\langle Z t_n \rangle] = \langle Z t_1;\ldots;t_n \rangle \]

Errors  47912  ?0 is not a Z declaration
        3012  ?0 and ?1 do not have the same types

val mk_z_decor_s : TERM * string -> TERM;
val is_z_decor_s : TERM -> bool;
val dest_z_decor_s : TERM -> TERM * string;

Description  Constructor, discriminator and destructor functions for systematic decoration of schemas. The first argument must be a schema, the second a decoration.

Example  \[ mk_z_decor_s[\langle [a,b,c:X | a = b] \rangle,\langle a',b',c':X | a' = b' \rangle \]  Errors  47340  ?0 is not a Z decorated schema
8.1. Syntactic Manipulations 365

SML

val mk_z_dec : TERM list * TERM -> TERM;
val is_z_dec : TERM -> bool;
val dest_z_dec : TERM -> TERM list * TERM;

**Description** Makes a simple declaration of one or more variables of the same type for use in the declaration part of a schema text.

**Definition**

\[ \text{mk}_z\text{dec}(\{\langle z\text{v}_1,\ldots,z\text{v}_n\rangle,\langle z\text{S}\rangle\}) = \langle z\text{v}_1,\ldots,z\text{v}_n:z\text{S}\rangle \]

Where the \( v_i \) and the members of \( S \) must have the same type.

**Uses** May only be used to make arguments for \( \text{mk}_z\text{decl} \).

**Errors**

47060 0 is not a Z set
3012 0 and 1 do not have the same types
3017 An empty list argument is not allowed
47061 0 is not a Z simple declaration

SML

val mk_z_eq : TERM * TERM -> TERM;
val is_z_eq : TERM -> bool;
val dest_z_eq : TERM -> TERM * TERM;

**Description** Equality. For the moment this is the same as HOL equality, but this is likely to change in the future. Both arguments must be of the same type.

**Definition**

\[ \text{mk}_z\text{eq}(\langle z\text{a},z\text{b}\rangle) = \langle z(a = b)\rangle \]

**Errors**

3012 0 and 1 do not have the same types
47220 0 is not a Z equality

SML

val mk_z_false : TERM;
val is_z_false : TERM -> bool;

**Description** The Z constant false. It is the same as the HOL constant \( F \).

SML

val mk_z_float : TERM * TERM * TERM -> TERM;
val is_z_float : TERM -> bool;
val dest_z_float : TERM -> TERM * TERM * TERM;

**Description** Constructor, discriminator and destructor functions for floating point literals. The argument is a triple of terms of type \( \mathbb{Z} \) giving the mantissa, the number of digits after the decimal point and the exponent in that order, i.e., the triple \((x, p, e)\) represents the real number \( x \times 10^{e-p} \).

**Errors**

47107 0 is not a Z floating point literal
47108 0 does not have type \( \mathbb{Z} \)
val mk_z_given_type : string -> TYPE;
val is_z_given_type : TYPE -> bool;
val dest_z_given_type : TYPE -> string

Description These are the constructor, discriminator and destructor functions for the types of
given sets. The type names used by these functions are the HOL names.

Errors

val mk_z_gvar : string * TYPE * TERM list -> TERM;
val is_z_gvar : TERM -> bool;
val dest_z_gvar : TERM -> string * TYPE * TERM list;

Description Constructor, discriminator and destructor functions for global variables. If the
third argument is the empty list, this function is the same as the HOL mk_const function, other-
wise a generic constant is created, the third argument being the generic actual parameters.

Errors

val mk_z_hide : TERM * string list -> TERM;
val is_z_hide : TERM -> bool;
val dest_z_hide : TERM -> TERM * string list;

Description The schema hiding constructor. The first argument must be a schema, the second
is a list of components to be hidden.

Definition

mk_z_hide(⌜S⌝,"c1",...,"cn") = ⌜S \ (c1,...,cn)⌝

Errors

val mk_z_schema : TERM * TERM -> TERM;
val is_z_schema : TERM -> bool;
val dest_z_schema : TERM -> TERM * TERM;

Description The schema constructor. The first argument is a declaration constructed using
mk_z_decl, the second is a predicate.

Definition

mk_z_schema(⌜d⌝,⌜p⌝) = ⌜d \ p⌝

Errors

val mk_z_int : string -> TERM;
val is_z_int : TERM -> bool;
val dest_z_int : TERM -> string;

Description Constructor, discriminator and destructor functions for integer literals. The arg-
ument should be a numeral, the result is the corresponding positive integer.

Errors
8.1. Syntactic Manipulations

SML

```sml
val mk_z_let : (string * TERM) list * TERM -> TERM;
val is_z_let : TERM -> bool;
val dest_z_let : TERM -> (string * TERM) list * TERM;
```

**Description** The let-term constructor. The arguments are list of pairs, each comprising a local variable name and a defining term for that local variable, and a term giving the body of the let-expression.

**Definition**

\[ \text{mk}_z\text{let}([("v", \frac{\delta}{\Delta} dt)], \ldots, \frac{\gamma}{\Gamma} b) = \frac{\xi}{\Xi} \text{let } v \triangleq dt; \ldots \bullet t \]

**Errors**

\[ \texttt{47211 !0 is not a Z let term} \]

---

SML

```sml
val mk_z_lvar : string * TYPE * TERM list -> TERM;
val is_z_lvar : TERM -> bool;
val dest_z_lvar : TERM -> string * TYPE * TERM list;
```

**Description** Constructor, discriminator and destructor functions for local variables. If the third argument is the empty list, this function is the same as the HOL \texttt{mk_var} function, otherwise a generic variable is created, the third argument being the generic actual parameters.

**Errors**

\[ \texttt{47090 !0 is not a Z local variable} \]

---

SML

```sml
val mk_z_power_type : TYPE -> TYPE;
val is_z_power_type : TYPE -> bool;
val dest_z_power_type : TYPE -> TYPE;
```

**Description** Set type constructor.

**Definition**

\[ \text{mk}_z\text{power}\_\text{type}\ (\frac{\Pi}{\Pi} ty) = \frac{\Pi}{\Pi} ty \]

**Errors**

\[ \texttt{47030 !0 is not a Z set type} \]

---

SML

```sml
val mk_z_pre : TERM -> TERM;
val is_z_pre : TERM -> bool;
val dest_z_pre : TERM -> TERM;
```

**Description** The schema precondition constructor. The argument must be a schema.

**Definition**

\[ \text{mk}_z\text{pre} (\frac{\forall }{\forall} S) = \frac{\forall }{\forall} \text{pre } S \]

**Errors**

\[ \texttt{47350 !0 is not a Z schema precondition} \]
val mk_z_rename : TERM * (string * string) list -> TERM;
val is_z_rename : TERM -> bool;
val dest_z_rename : TERM -> TERM * (string * string) list;

Description  The schema renaming construct. Its argument must be a schema.

Definition
\[
\text{mk}_z\text{rename}(S, [("x_1","y_1"),...]) = S[x_1/y_1,...]
\]

Errors
47461 ?0 is not a Z schema renaming
47462 Cannot rename ?0 to more than one name
47463 Cannot rename more than one name to ?0

val mk_z_schema_dec : TERM * string -> TERM;
val is_z_schema_dec : TERM -> bool;
val dest_z_schema_dec : TERM -> TERM * string;

Description  Constructor, discriminator and destructor functions for the components of a schema (the first argument), systematically decorated with the second argument.

Uses  May only be used to make arguments for mk_z_decl.

Errors
47940 ?0 is not a Z schema
47071 ?0 is not a Z schema as a declaration

val mk_z_schema_pred : TERM * string -> TERM;
val is_z_schema_pred : TERM -> bool;
val dest_z_schema_pred : TERM -> TERM * string;

Description  The schema as predicate constructor. The first argument must be a schema, the second is an optional decoration.

Errors
47940 ?0 is not a Z schema
47320 ?0 is not a Z schema as a predicate expression

val mk_z_schema_type : (string * TYPE) list -> TYPE;
val is_z_schema_type : TYPE -> bool;
val dest_z_schema_type : TYPE -> (string * TYPE) list;

Description  Binding type constructor.

Definition
\[
\text{mk}_z\text{schema}\_\text{type}([(c_1,ty_1),...,(c_n,ty_n)]) = [c_1:ty_1 ; ... ; c_n:ty_n]
\]

Errors
47050 ?0 is not a Z binding type
8.1. Syntactic Manipulations

SML:

```sml
val mk_z_sel : TERM * string -> TERM;
val is_z_sel : TERM -> bool;
val dest_z_sel : TERM -> TERM * string;
```

**Description** Selection of a component from a binding. The type of the first argument must be a binding and the second argument must be a component of that type.

**Definition**

\[ \text{mk}_z\text{sel}(\mathcal{Z}S, "c") = \mathcal{Z}S.c \]

**Errors**

47180 ?0 is not a Z selection

---

SML:

```sml
val mk_z_sel : TERM * int -> TERM;
val is_z_sel : TERM -> bool;
val dest_z_sel : TERM -> TERM * int;
```

**Description** Selection of a component from a tuple. The type of the first argument must be a tuple and the second argument must be a component in that tuple.

**Definition**

\[ \text{mk}_z\text{sel}(\mathcal{Z}\text{Tup}, i) = \mathcal{Z}\text{Tup}.i \]

**Errors**

47185 ?0 is not a Z tuple selection

---

SML:

```sml
val mk_z_seta : TERM * TERM * TERM -> TERM;
val is_z_seta : TERM -> bool;
val dest_z_seta : TERM -> TERM * TERM * TERM;
```

**Description** Constructor, discriminator and destructor functions for set comprehension. The three arguments represent the declaration, predicate and body parts of the set comprehension and so must have the appropriate types. In particular, the first argument must be made using \( \text{mk}_z\text{decl} \).

**Definition**

\[ \text{mk}_z\text{seta}(\mathcal{Z}d, \mathcal{Z}p, \mathcal{Z}v) = \{d \mid p \bullet v\} \]

**Errors**

47130 ?0 is not a Z set comprehension

---

SML:

```sml
val mk_z_setd : TYPE * TERM list -> TERM;
val is_z_setd : TERM -> bool;
val dest_z_setd : TERM -> TYPE * TERM list;
```

**Description** Constructor, discriminator and destructor functions for finite set displays. The result is the set made from the terms in the second argument, each of whose types must be the same as the first argument.

**Definition**

\[ \text{mk}_z\text{setd}(ty, [\mathcal{Z}t_1, \ldots, \mathcal{Z}t_n]) = \{t_1, \ldots, t_n\} \]

Where the \( t_i \) all have type \( ty \).

**Errors**

47120 ?0 is not a Z set display
val mk_z_string : string -> TERM;
val is_z_string : TERM -> bool;
val dest_z_string : TERM -> string;

Description Constructor, discriminator and destructor functions for string literals. The argument should be a string, the result is the corresponding string quotation.

Errors 47106 ?0 is not a Z string

val mk_z_term : Z.Term -> TERM;

Description Given any Z.Term, mk_z_term calls the appropriate abstract machine mk function.

val mk_z_true : TERM;
val is_z_true : TERM -> bool;

Description The Z constant true. It is the same as the HOL constant T.

val mk_z_tuple_type : TYPE list -> TYPE;
val is_z_tuple_type : TYPE -> bool;
val dest_z_tuple_type : TYPE -> TYPE list;

Description Cartesian product type constructor.

Definition \(mk_z_{\text{tuple}} [t_1, \ldots, t_n] = t_1 \times \ldots \times t_n\)

Errors 47040 ?0 is not a Z tuple type

val mk_z_tuple : TERM list -> TERM;
val is_z_tuple : TERM -> bool;
val dest_z_tuple : TERM -> TERM list;

Description The tuple constructor.

Definition \(mk_z_{\text{tuple}} [\langle t_1, \ldots, t_n \rangle] = \langle t_1, \ldots, t_n \rangle\)

Errors 47150 ?0 is not a Z tuple

val mk_z_type : Z.Type -> TYPE;

Description Given any Z.Type, mk_z_type calls the appropriate abstract machine mk function.

val mk_z_var_type : string -> TYPE;
val is_z_var_type : TYPE -> bool;
val dest_z_var_type : TYPE -> string;

Description The type of generic parameters.

Errors 47020 ?0 is not a Z type variable
8.1. Syntactic Manipulations

**SML**

```sml
val mk_z_Δ_s : TERM -> TERM;
val is_z_Δ_s : TERM -> bool;
val dest_z_Δ_s : TERM -> TERM;
```

**Description**  The delta constructor. Its argument must be a schema.

**Definition**  \[ \text{mk}_z_s \Delta \frac{\_}{s} \frac{\_}{\_} \frac{\_}{\_} = \frac{\_}{s} \Delta \frac{\_}{\_} \]

**Errors**  

\[ \text{47460} \ ?0 \text{ is not a Z } \Delta \]

**SML**

```sml
val mk_z_∈ : TERM * TERM -> TERM;
val is_z_∈ : TERM -> bool;
val dest_z_∈ : TERM -> TERM * TERM;
```

**Description**  Set membership. The second argument must be a set, whose members have the same type as the first argument.

**Definition**  \[ \text{mk}_z_s \in \frac{\_}{a} \frac{\_}{b} = \frac{\_}{a} (a \in b) \]

**Errors**  

\[ \text{47230} \ ?0 \text{ is not a Z set membership} \]

**SML**

```sml
val mk_z_Ξ_s : TERM -> TERM;
val is_z_Ξ_s : TERM -> bool;
val dest_z_Ξ_s : TERM -> TERM;
```

**Description**  The xi constructor. Its argument must be a schema.

**Definition**  \[ \text{mk}_z_s \Xi \frac{\_}{s} \frac{\_}{\_} = \frac{\_}{s} \Xi \frac{\_}{\_} \]

**Errors**  

\[ \text{47470} \ ?0 \text{ is not a Z } \Xi \]

**SML**

```sml
val mk_z_⇔_s : TERM * TERM -> TERM;
val is_z_⇔_s : TERM -> bool;
val dest_z_⇔_s : TERM -> TERM * TERM;
```

**Description**  The schema equivalence constructor. Both arguments must be schemas.

**Definition**  \[ \text{mk}_z_s \leftrightarrow \frac{\_}{R} \frac{\_}{S} = \frac{\_}{R} \leftrightarrow \frac{\_}{S} \]

**Errors**  

\[ \text{47400} \ ?0 \text{ is not a Z schema if and only if} \]
val mk\_z\_\rightarrow : TERM * TERM -> TERM;
val is\_z\_\rightarrow : TERM -> bool;
val dest\_z\_\rightarrow : TERM -> TERM * TERM;

**Description**  If and only if; the same as HOL $\iff$. Its argument must be $\mathbb{bool}$ type.

**Errors**
3015 ?1 is not of type $\mathbb{BOOL}$
3031 ?0 is not of type $\mathbb{BOOL}$
47280 ?0 is not a Z if and only if

val mk\_z\_\langle\rangle : TYPE * TERM list -> TERM;
val is\_z\_\langle\rangle : TERM -> bool;
val dest\_z\_\langle\rangle : TERM -> TYPE * TERM list;

**Description**  Constructor, discriminator and destructor functions for finite sequences. The result is the sequence made from the terms in the second argument, each of whose types must be the same as the first argument.

**Definition**
$$mk\_z\_\langle\rangle(ty, z_t_1, \ldots, z_t_n) = z_t_1, \ldots, z_t_n)$$

Where the $t_i$ all have type $ty$.

**Errors**
47110 ?0 is not a Z sequence display

val mk\_z\_\wedge\_\langle\rangle : TERM * TERM -> TERM;
val is\_z\_\wedge\_\langle\rangle : TERM -> bool;
val dest\_z\_\wedge\_\langle\rangle : TERM -> TERM * TERM;

**Description**  The schema conjunction constructor. Both arguments must be schemas.

**Definition**
$$mk\_z\_\wedge\_\langle\rangle(z_R, z_S) = z_R \wedge z_S$$

**Errors**
47370 ?0 is not a Z schema conjunction

val mk\_z\_\wedge\_ : TERM * TERM -> TERM;
val is\_z\_\wedge\_ : TERM -> bool;
val dest\_z\_\wedge\_ : TERM -> TERM * TERM;

**Description**  Conjunction; the same as HOL $\wedge$. Its arguments must be $\mathbb{bool}$ type.

**Errors**
3015 ?1 is not of type $\mathbb{BOOL}$
3031 ?0 is not of type $\mathbb{BOOL}$
47250 ?0 is not a Z conjunction
### 8.1. Syntactic Manipulations

<table>
<thead>
<tr>
<th>SML</th>
<th>(\text{val mk_z_∨_s : TERM \ast TERM \rightarrow TERM;})</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(\text{val is_z_∨_s : TERM \rightarrow \text{bool};})</td>
</tr>
<tr>
<td></td>
<td>(\text{val dest_z_∨_s : TERM \rightarrow TERM \ast TERM;})</td>
</tr>
</tbody>
</table>

**Description** The schema disjunction constructor. Both arguments must be schemas.

**Definition**

\[
\text{mk\_z\_∨\_s(⌜R⌝,⌜S⌝)} = ⌜R ∨ S⌝
\]

**Errors**

\[47380 ?0 \text{ is not a Z schema disjunction}\]

<table>
<thead>
<tr>
<th>SML</th>
<th>(\text{val mk_z_¬_s : TERM \rightarrow TERM;})</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(\text{val is_z_¬_s : TERM \rightarrow \text{bool};})</td>
</tr>
<tr>
<td></td>
<td>(\text{val dest_z_¬_s : TERM \rightarrow TERM;})</td>
</tr>
</tbody>
</table>

**Description** The schema negation constructor. The argument must be a schema.

**Definition**

\[
\text{mk\_z\_¬\_s(⌜S⌝)} = ⌜¬S⌝
\]

**Errors**

\[47360 ?0 \text{ is not a Z schema negation}\]

<table>
<thead>
<tr>
<th>SML</th>
<th>(\text{val mk_z_⇒_s : TERM \ast TERM \rightarrow TERM;})</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(\text{val is_z_⇒_s : TERM \rightarrow \text{bool};})</td>
</tr>
<tr>
<td></td>
<td>(\text{val dest_z_⇒_s : TERM \rightarrow TERM \ast TERM;})</td>
</tr>
</tbody>
</table>

**Description** The schema implication constructor. Both arguments must be schemas.

**Definition**

\[
\text{mk\_z\_⇒\_s(⌜R⌝,⌜S⌝)} = ⌜R ⇒ S⌝
\]

**Errors**

\[47390 ?0 \text{ is not a Z schema implication}\]
val mk_z ⇒ : TERM * TERM -> TERM;
val is_z ⇒ : TERM -> bool;
val dest_z ⇒ : TERM -> TERM * TERM;

**Description**  Implication; the same as HOL ⇒. Its arguments must be bool type.

**Errors**

3015 ?1 is not of type `:BOOL`
3031 ?0 is not of type `:BOOL`
47270 ?0 is not a Z implication

val mk_z ∀ : TERM * TERM * TERM -> TERM;
val is_z ∀ : TERM -> bool;
val dest_z ∀ : TERM -> TERM * TERM * TERM;

**Description**  The schema universal quantifier constructor. The arguments must be a declaration (constructed using mk_z decl), a predicate and a schema.

**Definition**

\[ \text{mk}_z \forall (\exists_1 d, \exists_2 p, \exists_3 S) = \exists_1 d \mid p \bullet S \]

**Errors**

47450 ?0 is not a Z schema universal

val mk_z ∃ : TERM * TERM * TERM -> TERM;
val is_z ∃ : TERM -> bool;
val dest_z ∃ : TERM -> TERM * TERM * TERM;

**Description**  Constructor, discriminator and destructor functions for universal quantification. Its arguments must be a declaration (constructed with mk_z decl) and two predicates.

**Definition**

\[ \text{mk}_z \exists (\exists_1 d, \exists_2 p, \exists_3 v) = \exists_1 d \mid p \bullet v \]

**Errors**

47912 ?0 is not a Z declaration
47310 ?0 is not a Z universal quantification

val mk_z ∃_1 : TERM * TERM * TERM -> TERM;
val is_z ∃_1 : TERM -> bool;
val dest_z ∃_1 : TERM -> TERM * TERM * TERM;

**Description**  The schema unique existential quantifier constructor. The arguments must be a declaration (constructed using mk_z decl), a predicate and a schema.

**Definition**

\[ \text{mk}_z \exists_1 (d, p, S) = \exists_1 d \mid p \bullet S \]

**Errors**

47440 ?0 is not a Z schema unique existential
SML
\[ \texttt{val \ mk\_z\_\exists_1} : \text{TERM} \times \text{TERM} \times \text{TERM} \rightarrow \text{TERM}; \]
\[ \texttt{val \ is\_z\_\exists_1} : \text{TERM} \rightarrow \text{bool}; \]
\[ \texttt{val \ dest\_z\_\exists_1} : \text{TERM} \rightarrow \text{TERM} \times \text{TERM} \times \text{TERM}; \]

**Description** Constructor, discriminator and destructor functions for unique existential quantification. Its arguments must be a declaration (constructed with \texttt{mk\_z\_decl}) and two predicates.

**Definition**
\[ \texttt{mk\_z\_\exists_1} (\zeta d'\exists_p,\xi v') = \zeta \exists_1 d | p \bullet v \]

**Errors**
\[ 47912 \ ?0 \ is \ not \ a \ Z \ declaration \]
\[ 47300 \ ?0 \ is \ not \ a \ Z \ unique \ existential \ quantification \]

---

SML
\[ \texttt{val \ mk\_z\_\exists_\Sigma} : \text{TERM} \times \text{TERM} \times \text{TERM} \rightarrow \text{TERM}; \]
\[ \texttt{val \ is\_z\_\exists_\Sigma} : \text{TERM} \rightarrow \text{bool}; \]
\[ \texttt{val \ dest\_z\_\exists_\Sigma} : \text{TERM} \rightarrow \text{TERM} \times \text{TERM} \times \text{TERM}; \]

**Description** The schema existential quantifier constructor. The arguments must be a declaration (constructed using \texttt{mk\_z\_decl}), a predicate and a schema.

**Definition**
\[ \texttt{mk\_z\_\exists_\Sigma} (\zeta d'\exists_p,\xi S) = \zeta \exists d | p \cdot S \]

**Errors**
\[ 47430 \ ?0 \ is \ not \ a \ Z \ schema \ existential \]

---

SML
\[ \texttt{val \ mk\_z\_\exists} : \text{TERM} \times \text{TERM} \times \text{TERM} \rightarrow \text{TERM}; \]
\[ \texttt{val \ is\_z\_\exists} : \text{TERM} \rightarrow \text{bool}; \]
\[ \texttt{val \ dest\_z\_\exists} : \text{TERM} \rightarrow \text{TERM} \times \text{TERM} \times \text{TERM}; \]

**Description** Constructor, discriminator and destructor functions for existential quantification. Its arguments must be a declaration (constructed with \texttt{mk\_z\_decl}) and two predicates.

**Definition**
\[ \texttt{mk\_z\_\exists} (\zeta d'\exists_p,\xi v) = \zeta \exists d | p \bullet v \]

**Errors**
\[ 47912 \ ?0 \ is \ not \ a \ Z \ declaration \]
\[ 47290 \ ?0 \ is \ not \ a \ Z \ existential \ quantification \]

---

SML
\[ \texttt{val \ mk\_z\_\times} : \text{TERM} \rightarrow \text{TERM}; \]
\[ \texttt{val \ is\_z\_\times} : \text{TERM} \rightarrow \text{bool}; \]
\[ \texttt{val \ dest\_z\_\times} : \text{TERM} \rightarrow \text{TERM} \rightarrow \text{TERM} \]

**Description** The cartesian product constructor.

**Definition**
\[ \texttt{mk\_z\_\times} (t_1'\times\ldots\times t_n') = (t_1 \times \ldots \times t_n) \]

**Errors**
\[ 47160 \ ?0 \ is \ not \ a \ Z \ cartesian \ product \]
val mk_z_o9s : TERM * TERM -> TERM;
val is_z_o9s : TERM -> bool;
val dest_z_o9s : TERM -> TERM * TERM;

Description The sequential composition constructor. Its arguments must both be schemas.

Definition
\[ \text{mk}_z(\sigma R \sigma S) = \sigma R \sigma S \]

Errors
\[ 47480 \text{ ?0 is not a Z schema composition} \]

val mk_z_\theta : TERM * string -> TERM;
val is_z_\theta : TERM -> bool;
val dest_z_\theta : TERM -> TERM * string;

Description The theta term constructor. The first argument must be a schema, the second is an optional decoration.

Definition
\[ \text{mk}_z(\sigma S, \theta') = \sigma \theta' S \]

Errors
\[ 47170 \text{ ?0 is not a Z } \theta \text{ term} \]

val mk_z_\lambda : TERM * TERM * TERM -> TERM;
val is_z_\lambda : TERM -> bool;
val dest_z_\lambda : TERM -> TERM * TERM * TERM;

Description The lambda constructor. The arguments are a declaration (constructed using \( \text{mk}_z \_\text{decl} \) q.v.), a predicate and the body of the abstraction.

Definition
\[ \text{mk}_z(\sigma d, \sigma p, \sigma v) = \sigma \lambda d | p \cdot v \]

Errors
\[ 47200 \text{ ?0 is not a Z } \lambda \text{ abstraction} \]
\[ 47201 \text{ ?0, ?1 and ?2 are inconsistent in Z} \]

val mk_z_\mu : TERM * TERM * TERM -> TERM;
val is_z_\mu : TERM -> bool;
val dest_z_\mu : TERM -> TERM * TERM * TERM;

Description The definite description constructor. The arguments are a declaration (constructed using \( \text{mk}_z \_\text{decl} \) q.v.), a predicate and the body of the definite description.

Definition
\[ \text{mk}_z(\sigma d, \sigma p, \sigma v) = \sigma \mu d | p \cdot v \]

Errors
\[ 47210 \text{ ?0 is not a Z } \mu \text{ term} \]
8.1. Syntactic Manipulations

**SML**

\[
\begin{align*}
\text{val } \text{mk\_z\_P} & : \text{TERM} \rightarrow \text{TERM}; \\
\text{val } \text{is\_z\_P} & : \text{TERM} \rightarrow \text{bool}; \\
\text{val } \text{dest\_z\_P} & : \text{TERM} \rightarrow \text{TERM};
\end{align*}
\]

**Description**  The powerset constructor.

**Definition**  \[\text{mk\_z\_P} \ t = \mathcal{P} \ t\]

**Errors**  47140  ?0 is not a Z powerset

\[
\begin{align*}
\text{val } \text{mk\_z\_↾} & : \text{TERM} \ast \text{TERM} \rightarrow \text{TERM}; \\
\text{val } \text{is\_z\_↾} & : \text{TERM} \rightarrow \text{TERM}; \\
\text{val } \text{dest\_z\_↾} & : \text{TERM} \rightarrow \text{TERM} \ast \text{TERM};
\end{align*}
\]

**Description**  The schema projection constructor. Both arguments must be schemas.

**Definition**  \[\text{mk\_z\_↾} (\langle \mathcal{Z} R \rangle, \langle \mathcal{Z} S \rangle) = \langle \mathcal{Z} R \mid S \rangle\]

**Errors**  47410  ?0 is not a Z schema projection
8.2 Reasoning about Predicates

```sml
signature ZPredicateCalculus = sig

Description This provides a set of rules of inference, conversions and tactics sufficient for reasoning about the Z predicate calculus in ProofPower. This structure declares the theory `z_language_ps`, which is also used by structures `ZSetTheory` and `ZSchemaCalculus`.
```

```sml
(* Proof Context: z_predicates *)

Description A complete proof context for handling the requirements of the Z predicates of the Z language (as opposed to the mathematical tool-kit). It is composed of proof contexts "z_predicates" and "z_decl".

Usage Notes It requires theory `z_language_ps`. It is not intended to be mixed with HOL proof contexts.
```
8.2. Reasoning about Predicates

Description
A component proof context for handling the requirements of the Z predicates of the Z language (as opposed to the mathematical tool-kit). It remains purely within the Z language, and thus lacks the features found in proof context “z_decl” which are necessary for a complete treatment of Z predicates. (which may be found in proof context “z_predicates”).

Predicates treated by this proof context are constructs formed from:

This proof context further handles membership of constructs purely constructed from U, generic formals, and Z paragraph markers. The language predicate € is treated with the set constructs that it expresses membership of. Schemas (and especially schema references) as predicates are treated by “zschemas”, except that this proof context will replace an ill-formed “schema as predicate” expression with an explicit membership.

Contents
Rewriting:

Stripping conclusions:

Note that we do not break apart a Z ∀ into HOL quantifiers during conclusion stripping.

Rewriting canonicalisation:

Notice in particular the use of the HOL \( \forall \_\text{rewrite\_canon} \).

Automatic proof procedures are respectively \( \_\text{basic\_prove\_tac} \), \( \_\text{basic\_prove\_conv} \), and the list

Usage Notes
It requires theory \( z\_\text{language\_ps} \). It is not intended to be mixed with HOL proof contexts. Use with proof context “z_decl” to handle declarations properly.
**Chapter 8. SUPPORT FOR Z**

**SML**

(* Proof Context: 'z_decl *)

**Description** A component proof context for handling the requirements of converting Z declarations into their implicit predicates, kept separate from "z_predicates" due to it introducing a small portion of Z library set theoretic reasoning.

The requirement is met by appropriate treatment of:

| set_display ⊆ set_expression |

during stripping.

**Contents**

Rewriting:

Stripping theorems:

| z_setd ⊆ conv, and this pushed in through ¬. |

Stripping conclusions:

| z_setd ⊆ conv, and this pushed in through ¬. |

Notice how this proof context does not use z_setd ⊆ conv for rewriting, but leaves such an effect to the proof context concerned with extensional reasoning about the Z library.

Rewriting canonicalisation:

Automatic proof procedures are respectively z_basic_prove_tac, z_basic_prove_conv, and no existence provers.

**Usage Notes** It requires theory z_language_ps. It is not intended to be mixed with HOL proof contexts. Used with proof context "z_predicates".

**SML**

(* Proof Context: 'z_fc *)

**Description** A component proof context giving a faster but less general automatic proof capability than the one supplied in most other proof contexts for Z. The automatic proof procedures in the proof context are z_fc_prove_tac, z_fc_prove_conv. All other fields are blank.

**Usage Notes** It requires theory z_language_ps.

Note that the way proof contexts are merged by push_merge_pcs is such that to get the faster automatic proof procedures, one should put 'z_fc at the end of the list of proof contexts to be merged. For example, to work in the Z predicate calculus with the faster automatic proof procedures, one might use

| push_merge_pcs["z_predicates", "'z_fc"] |
8.2. Reasoning about Predicates

SML

```sml
(* check_is_z : boolean flag *)
val set_check_is_z : bool -> bool;
```

**Description**  This flag, if true (the default), will cause all Z inference rules and tactics that claim to remain in the Z language to check any terms they change (i.e. assumptions and conclusions) for remaining within the Z language. If any fail then the informational message 41004 is used to output text to the user. If the flag is false, no such checks are made. The checks are computationally expensive, and the results may be excessively verbose if terms are not all Z.

The function sets the flag to a specified value and returns the original value.

**Errors**

| 41004 The following subterms in the result are not in the Z language: ?0 |

SML

```sml
val all_z_∀Intro : THM -> THM;
```

**Description**  This will Z universally quantify all free variables in the conclusion of a theorem, that do not occur in the assumptions. The declaration part will state the variables are of type \( \forall U \), and the predicate part will just be `true`. If no variables can be introduced then the original theorem will be returned.

**Errors**

| 41005 In the result of '?0 the following subterms are not in the Z language: '?1 |

SML

```sml
| val check_is_z_thm : string -> THM -> THM;
| val check_is_z_goal : string -> GOAL -> GOAL;
| val check_is_z_term : string -> TERM -> TERM;
| val CHECK_IS_Z_T : string -> TACTIC -> TACTIC;
| val check_is_z_conv_result : string -> THM -> THM;
```

**Description**  For `check_is_z_thm`, if flag `check_is_z` is true then the conclusion and assumptions of the provided theorem are checked for being within the Z language (except for outermost HOL universal quantification), and informational message 41005 used if not. The string argument is used as the name of the calling function in the error message. If the flag is false then there is no effect. In either case the theorem is passed through unchanged.

`check_is_z_goal` and `check_is_z_term` are analogous. `CHECK_IS_Z_T` checks each of the subgoals a tactic requests.

`check_is_z_conv_result` checks that the RHS of the resulting equational theorem, and any assumptions are within the Z language. This allows the RHS side of the equation to have outer HOL universal quantification, and the LHS not to be Z (e.g. in an Z introduction conversion) without complaint.

**Errors**

| 41005 In the result of '?0 the following subterms are not in the Z language: '?1 |
val dest_z_term1 : TERM -> Z_TERM;

**Description**  This function acts as dest_z_term on terms (i.e. expressions and predicates) in the Z language, but makes additional checks. This is in contrast to dest_z_term whose intended purpose is categorisation and destruction of Z terms with minimal overhead.

The function does not recursively check the constituents of the outermost Z syntactic construction. For example, it does not check that the constituents of a Z decl are individually in the syntactic category `dec`.

**Errors**

| 41002 | Not within the Z language due to subterm ?0 |

val is_z_term1 : TERM -> bool;

**Description**  Tests if a given HOL term is valid Z in its top level structure.

**Uses**  Recursively in well-formedness checks.

**See Also**  is_z_term for a less complete check of top level structure, is_z for a full traversal of the terms structure.

val is_z : TERM -> bool;
val is_all_z_type : TYPE -> bool;

**Description**  If the term (i.e. expression or predicate) or type given is in the range of the Z mapping for a term or type respectively then these functions will return true. They will otherwise return false, unless the only form of incorrectness is that the constituents of a Z syntactic construction are not as required. For example, it does not check that the constituents of a Z decl are individually in the syntactic category `dec`.

The test traverses the provided object by using full_dest_z_term (and dest_z_type for constituent-types) - the test is passed if the entire term can be broken into non-type and non-term parts (i.e. primitives such as strings or integers). Otherwise it will fail with the given error message.

Note that a term is a subterm of itself for these purposes.

**See Also**  is_z_term and is_z_term1.

**Errors**

| 41002 | Not within the Z language due to subterm ?0 |
| 41003 | Not within the Z language due to containing type ?0 |

val not_z_subterms : TERM -> TERM list;

**Description**  This function will return a list (perhaps empty) of all the subterms that prevent a term (i.e. expression or predicate) being within the Z language (by the checks of is_z, q.v.), starting with the rightmost subterm that is not Z. The subterms given will be maximal in the sense that subterms of those given will not be included in the list.
8.2. Reasoning about Predicates

SML

val set_u_simp_eqn_cxt : EQN_CXT -> string -> unit;
val get_u_simp_eqn_cxt : string -> (EQN_CXT * string)list;

Description

set_u_simp_eqn_cxt ec pc_name; sets the “icl’u_simp” entry of the dictionary of nets field of the proof context called “pc_name” to the equational context ec. This means that when this named proof context has been made the current proof context (probably merged with others) it will be “aware” of the equational contexts potential $\Un$ simplifications.

For example, to make the current proof context aware of the $\Un$ simplifications of the (in scope) theory “thy” one would do:

```
new_pc "thy_u_simp_pe";
set_u_simp_eqn_cxt (theory_u_simp_eqn_cxt "thy") "thy_u_simp_pe";
push_merge_pcs ("thy_u_simp_pe" :: other_desired_proof_contexts);
```

One could later update information about the theory (e.g. because new definitions have been added) by:

```
set_u_simp_eqn_cxt (theory_u_simp_eqn_cxt "thy") "thy_u_simp_pe";
set_merge_pcs ("thy_u_simp_pe" :: other_desired_proof_contexts);
set_u_simp_eqn_cxt ex pc_name; extracts the $\Un$ simplification subfields of the named proof context. These subfields are each an equational context paired with its original source proof context name.
```

See Also

u_simp_eqn_cxt, theory_u_simp_eqn_cxt

Errors

51010 There is no proof context with key ?0
51014 Proof context ?0 was created in theory ?1 at a point now either not in scope, deleted or modified
51016 Proof context ?0 has been committed

SML

val theory_u_simp_eqn_cxt : string -> EQN_CXT;

Description

theory_u_simp_eqn_cxt theory_name takes the named theory and checks it for theorems, definitions and axioms that could be used for creating proof context entries used by z_\in u_conv.

A theorem is checked by canonicalising it, and accepting those resulting theorems that are equations between an expression that is not $\Un$, and $\Un$. Those that can be so used are processed by thm_eqn_cxt and then added to the equational context being generated.

Uses

This function is primarily intended for the automatic extraction and processing of the given set and free type definitions of a theory, when building a proof context for a particular theory.

Note that equational contexts can be joined using list append, @.

See Also

u_simp_eqn_cxt, set_u_simp_eqn_cxt

Errors

As the failures of get_defn.
val u_simp_eqn_cxt : THM list -> EQN_CXT;

**Description**  
*u_simp_eqn_cxt* thms takes each member of thms, and checks and then processes it for use in creating proof context entries used by *z∈u_conv*.

The check is that each theorem is canonicalised with the current proof context’s canonicalisation function. For each resulting theorem, if it is a universally quantified equation of sets then it is processed by *thm_eqn_cxt* and added into the created equational context. If it is not equation of sets the theorem is ignored.

**Uses**  
This function is primarily intended to aid the construction of proof contexts containing U simplification material.

Note that equational contexts can be joined using list append, @.

**See Also**  
theory_u_simp_eqn_cxt, set_u_simp_eqn_cxt
**Description** This is the conversion used for the automatic proof conversion (pr\_tac field) of most supplied proof contexts, and is a reasonable, general-purpose, automatic proof conversion. It will either prove the theorem with the given conclusion, or fail.

In summary it will:

1. Set the term as the goal of the subgoal package (or, more exactly, tac\_proof).
2. Attempt to rewrite the term with the current default rewrite rules and given theorems.
3. Repeatedly apply strip\_tac to the goal.
4. Attempt variable elimination, using all\_var\_elim\_asm\_tac.
5. In all resulting goals replace all Z quantifiers by their HOL equivalents in both assumptions and goal.
6. Apply all\_asm\_fc\_tac once to each resulting goal.
7. Attempt to prove the resulting goals with resolution for up to 3 resolution steps, with goal’s negated conclusion as a resolvant that must be used, and the assumptions as possible other resolvants.
8. Attempt to prove the resulting goals with resolution for up to 3 resolution steps amongst just the assumptions.
9. If the proof is successful, return $\vdash \text{term} \iff T$ and otherwise fail.

Note that in the stripping step may result in more than one subgoal, and thus the phrase “resulting goals” is used above.

Under the current interface to resolution this is equivalent to:

```sml
|fun z_basic_prove_conv thms tm =
 ⇐ t_intro {
  tac_proof(([],tm),
   TRY T (rewrite_tac thms) THEN
   REPEAT strip_tac THEN
   TRY T all_var_elim_asm_tac THEN
   (z_quantifiers_elim_tac THEN
   basic_res_tac2 3 [¬ ∀ x • x = x]
   ORELSE T basic_res_tac3 3 [¬ ∀ x • x = x]))
 );

In the implementation however, partial evaluation with just the theorems is allowed.
```

Errors

| 76001 Could not prove theorem with conclusion ?0 |
\begin{verbatim}
val z_basic_prove_tac : THM list -> TACTIC;

Description  This is the tactic used for the automated proof tactic (the \texttt{pr\_tac} field) of most supplied Z proof contexts, and is a reasonable, general-purpose, automatic proof tactic for Z. In summary it will:

1. Attempt variable elimination, using \texttt{all\_var\_elim\_asm\_tac}.
2. Extract the assumption list, rewrite each extracted assumption with the current default rewrite rules and given theorems, and strip the results back into the assumption list.
3. Attempt to rewrite the conclusion of the resulting goal with the current default rewrite rules and given theorems.
4. Repeatedly apply \texttt{strip\_tac} to the conclusions of the resulting goals.
5. Again attempt variable elimination, using \texttt{all\_var\_elim\_asm\_tac}.
6. In all resulting goals replace all Z quantifiers by their HOL equivalents in both assumptions and goal. This has no effect on any resulting goal if it is unsolved.
7. Apply \texttt{all\_asm\_fc\_tac} once to each resulting goal.
8. Attempt to prove each of the resulting goals with resolution for up to 3 resolution steps, with goal’s negated conclusion as a resolvent that must be used, and the assumptions as possible other resolvants. This has no effect on any resulting goal if it is unsolved.
9. Attempt to prove each of the resulting goals with resolution for up to 3 resolution steps amongst just the assumptions. This has no effect on any resulting goal if it is unsolved.

Note that either stripping step may result in more than one subgoal, and thus the plural “resulting goals”.

Under the current interface to resolution this is

\begin{verbatim}
fun z_basic_prove_tac thms =
  TRY_T all_var_elim_asm_tac THEN
  DROP_ASMST (MAP_EVERY (strip_asm_tac o rewrite_rule thms) o rev) THEN
  TRY_T (rewrite_tac thms)) THEN
  REPEAT strip_tac THEN
  TRY_T all_var_elim_asm_tac THEN_TRY
  (z_quantifiers_elim_tac
   THEN basic_res_tac2 3 \[ \forall x \cdot x = x\])
  ORELSE_T basic_res_tac3 3 \[ \forall x \cdot x = x\]);
\end{verbatim}
\end{verbatim}
8.2. Reasoning about Predicates

**SML**

\[
\text{val } Z\_\text{DECL}\_C : \text{CONV } \rightarrow \text{CONV};
\]

**Description**  
\(Z\_\text{DECL}\_C\) applies the supplied conversion to each member of a declaration and returns the conjunction of the results. It fails if its conversion fails on any member of the declaration.

\[
\text{fun } z\_\text{decl\_pred\_conv} = Z\_\text{DECL}\_C \ z\_\text{dec\_pred\_conv};
\]

will convert a valid Z declaration into its implicit Z predicate.

**Errors**

47912  
?0 is not a Z declaration

41012  
Supplied conversion failed on one or more members of ?0

**SML**

\[
\text{val } Z\_\text{DECL\_INTRO}\_C : \text{CONV } \rightarrow \text{CONV};
\]

**Description**  
\(Z\_\text{DECL\_INTRO}\_C\) applies the supplied conversion to each conjunct of a predicate, flattening the conjunctive structure. If this is successful, it attempts to produce a declaration from the results.

\(Z\_\text{DECL\_INTRO}\_C \ z\_\text{pred\_dec\_conv}\) will convert certain Z predicates into Z declarations implicitly containing the predicates, and otherwise will fail.

**Errors**

41013  
?0 not of the form: \(\exists x \text{true}\) or \(\exists c_1 \land \ldots \) where all the \(c_i\) may have the supplied conversion applied

41014  
?0 when converted to ?1 cannot be viewed as a declaration

The conversion fails if the supplied conversion fails on any conjunct, returning the error message of that conversion application.
val z_decl_pred_conv : CONV;

Description  A conversion which rewrites an explicit Z declaration (i.e. a decl) to its implicit predicate. An Z declaration may be found, e.g., as a component of a Z horizontal schema. A declaration consists of a list of components (each a dec), that are individually converted into predicates, and the results conjoined. The predicate implicit in a declaration, $D$, is also sometimes referred to as the “predicate from $D$”.

The function is defined much as if by the following:

Definition  \[
\text{val } z_{\text{decl pred conv}} = Z_{\text{DECL C}} z_{\text{dec pred conv}};
\]

Thus the handling of the individual declarations is as shown in the following examples:

Conversion  \[
\vdash _{ML} \text{decl of } Z[x : X; y, z : Y; S]\iff z_{\text{decl pred conv}} (\text{decl of } Z[x : X; y, z : Y; S])
\]

and

Conversion  \[
\vdash _{ML} \text{decl of } Z[\Rightarrow true] \iff z_{\text{decl pred conv}} (\text{decl of } Z[\Rightarrow true])
\]

Note that a declaration on its own is not a Z expression, though it may be correctly embedded within certain forms of Z expressions.

See Also  z_dec_pred_conv

Errors  [47912 ?0 is not a Z declaration]
### z_dec_pred_conv

**Description**  
A conversion which rewrites a `dec` part of a declaration to its implicit predicate.  
A `decesexp` type of declaration remains unchanged (since `decesexp` and `predsexp` are, in fact, the same thing).

Conversions:

1. \[ \vdash \text{ml} \text{mk}_z \text{dec}([\boxed{x}], [_{x}{X}]) \iff x \in X \]

   \[ z_{-\text{dec}\_pred\_conv} \quad (\text{mk}_z z_{-\text{dec}}([_{x}{X}]), z_{-\text{dec}\_pred\_conv}) \]

2. \[ \vdash \text{ml} \text{mk}_z \text{dec}([_{x}{X}], ..., [_{x}{X}]) \iff \{x_1, ..., \} \subseteq X \]

   \[ z_{-\text{dec}\_pred\_conv} \quad (\text{mk}_z z_{-\text{dec}}([_{x}{X}], ..., [_{x}{X}]), z_{-\text{dec}\_pred\_conv}) \]

3. \[ \vdash S \iff S \]

   \[ z_{-\text{dec}\_pred\_conv} \quad (S, z_{-\text{dec}\_pred\_conv}) \]

where `S` is a schema (here promoted to a predicate). In this last case if the schema as predicate expression is not well-formed `Z` (perhaps because of substitution of variables) the result will be further converted to correct `Z` of the form:

\[ binding \in \text{schema} \]

Note that a declaration on its own is not a `Z` expression, though it may be correctly embedded within certain forms of `Z` expressions.

**See Also**  
`z\_pred\_dec\_conv`

**Errors**

`41010`  
`?0` is not a declaration

### z_fc_prove_conv

**Description**  
This is the automatic proof conversion supplied in the proof context `z\_fc\`. It is based on the automatic proof tactic `z\_fc\_prove\_tac`, q.v., and is defined, in effect as:

\[
\text{fun } z_{-\text{fc}\_prove\_conv} (\text{thms}: \text{THM list}) : \text{CONV} = (\text{fn tm }\Rightarrow \text{t\_intro (tac\_proof(([],tm), z_{-\text{fc}\_prove\_tac\_thms})}))
\]
val \texttt{z\_fc\_prove\_tac} : \texttt{THM list → TACTIC}

Description  The resolution-based proof procedure \texttt{z\_basic\_prove\_tac} supplied as the automatic proof tactic in many of the the proof contexts for \texttt{Z} may be found to be somewhat slow on complex problems. \texttt{z\_fc\_prove\_tac} supplies a less general but quicker alternative based on forward chaining (in the sense of \texttt{fc\_tac}). It is supplied as the automatic proof tactic field in the proof context \texttt{\'z\_fc}. Its effect may be described as follows:

1. Attempt variable elimination, using \texttt{all\_var\_elim\_asm\_tac}.

2. Extract the assumption list, rewrite each assumption as it is extracted with the current default rewrite rules and given theorems, and strip the results back into the assumption list.

3. Attempt to rewrite the conclusions of the resulting goals with the current default rewrite rules and the argument theorems.

4. Apply \texttt{contr\_tac}.

5. Again attempt variable elimination, using \texttt{all\_var\_elim\_asm\_tac}.

6. In all resulting goals replace all \texttt{Z} quantifiers by their \texttt{HOL} equivalents.

7. Apply \texttt{all\_asm\_fc\_tac}.

8. Generate (universally quantified) implications from the assumptions using the canonicalisation function \texttt{fc\_canon1}. The go through three forward chaining passes (in the sense of \texttt{fc\_tac}) using these implications as a starting point. At the end of each pass any generated results are both stripped into the assumptions and processed with \texttt{fc\_canon1} to be passed on as additional implications for the subsequent pass.

For example, the tactic will prove the following goal:

\[
(\forall x1 : Z \bullet x1 \in A \Rightarrow x1 \in B) \land \\
(\forall x1 : U; x2 : U \bullet (x1, x2) \in B \land x \iff (x1, x2) \in B \land x') \land \\
x1 \in A \land \\
x1 \in Z \land \\
(x1, x2) \in x \Rightarrow (x1, x2) \in x'
\]

\]
8.2. Reasoning about Predicates

SML

```sml
val z_gen_pred_elim : TERM list -> THM -> THM;
val z_gen_pred_elim1 : TERM -> THM -> THM;
```

**Description**
Eliminate (some of) the generic formals of a generic predicate for actual values. If possible, the theorem will be type instantiated to allow generic formals to match the types of the supplied TERM list, otherwise the rule fails.

**Rule**

\[ \frac{\Gamma \vdash [X_1,...,X_n] (t[X_1,...,X_n])}{\Gamma \vdash t[t_1,...,t_n]} \]

\( z_{gen\_pred\_elim} \) is just like \( z_{gen\_pred\_elim} \) except that its argument is a term rather than a list of terms. \( z_{gen\_pred\_elim1} \) is equivalent to \( z_{gen\_pred\_elim} \) if the term argument, \( t \), is not a \( Z \) tuple; \( z_{gen\_pred\_elim1} \) if the term argument, \( t \), is not a \( Z \) tuple, \( z_{gen\_pred\_elim1} \) is equivalent to \( z_{gen\_pred\_elim} \). The advantage of \( z_{gen\_pred\_elim1} \) is that in a call such as \( z_{gen\_pred\_elim1} \), the \( Z \) type inferrer can assign a more general type to the occurrences of \( U \) than it does in the call \( z_{gen\_pred\_elim} \).

**Errors**

41033 \( 0 \) is not of the form: \( \forall \Gamma \vdash [X_1,...,X_n] t^\gamma \) where the types of the theorem can be instantiated to allow the types of the generic formals to match the types of the term list.

41034 \( 0 \) is not of the form: \( \forall \Gamma \vdash [X_1,...,X_n] t^\gamma \) where there are sufficient \( X_i \) to match the supplied term list.

SML

```sml
val z_gen_pred_intro : TERM list -> THM -> THM;
```

**Description**
Introduce a list of generic formals. The TERM list argument is of variables. Their types will be ignored, they are replaced by the variables \( \forall \var \ P \ var^\gamma \).

**Rule**

\[ \frac{\Gamma \vdash [X_1,...,X_n]}{\Gamma \vdash [X_1,...,X_n] (t[X_1,...,X_n])} \]

\( z_{gen\_pred\_intro} \)

**Errors**

3007 \( 0 \) is not a term variable

6005 \( 0 \) occurs free in assumption list

SML

```sml
val z_gen_pred_tac : TACTIC;
```

**Description**
A tactic to eliminate generic predicates.

**Tactic**

\[ \{\Gamma\} \ [X_1,...] t \]

\( z_{gen\_pred\_tac} \)

**Errors**

41035 conclusion of goal is not of the form \( \forall [X_1,...] t^\gamma \)
| val  z_gen_pred_u_elim : THM → THM;  |
| Description Substitute U for each of the generic formals of a generic predicate. |
| Rule       | $Γ ⊢ [X_1, X_2, ...] (t[X_1, X_2, ...])$  |
|            | $Γ ⊢ t[U, U, ...]$                        |
| z_gen_pred_u_elim  |

Each occurrence of U is instantiated to the same type as the corresponding generic formal parameter.
SML

|val z_get_spec : TERM --> THM;

**Description** This function returns the specification of a constant, based on its defining theorem and, if one can be found, a consistency theorem. The defining theorem may have been created by Z paragraph processing, `new_axiom`, or a HOL definitional mechanism. This function should be the Z user’s interface to definitional theorems, as `get_spec(q.v.)` is for the HOL user.

`z_get_spec "const"` will find the (first) definition or axiom in scope stored under key “name of const”, in the theory in which the in-scope constant named `const` was defined. A definition will be taken in preference to an axiom in the same theory. `z_get_spec "const t1 t2 ..."` (i.e. a constant applied to an arbitrary number of arguments in HOL) will act as `z_get_spec "const"`. This choice is made in the assumption that a naming convention has been followed that such a definition (or axiom) should be the definition of the constant named `const`. This convention has been followed throughout the implementation of ProofPower. In addition, there can only be one definition of a particular constant in scope (though the conventional key might be used elsewhere, or not at all). If there is no such constant in scope, or no definition with the given key, then the function fails.

If the definitional theorem is of the form:

\[ \vdash ConstSpec \ p \ c \]

(i.e. its introduction requires a consistency assumption) the function will seek for a theorem or axiom stored with key `const` “consistent”, starting at the theory in which the definition was found, and working “out” to the current theory. If conventions have been followed this theorem should be of the form:

\[ \Gamma \vdash Consistent \ p \]

(Ideally there should be no assumptions in the theorem, but the function caters for their presence.) If a theorem of this form is found then the theorem:

\[ \beta \text{rule } \Gamma \vdash p \ c' \]

is formed. If not, then the theorem:

\[ \beta \text{rule } Consistent \ p \vdash p \ c' \]

is formed. In all of the above cases, (i.e. with or without `ConstSpec`), the theorem formed is checked to see whether it is the definition formed from processing a Z paragraph. If so, then the conclusion of the theorem is converted into a predicate (by `z_para_pred_conv`), and then returned as the result of `z_get_spec`. If not, then the theorem is returned without further processing as result of `z_get_spec`.

**Errors**

46005 There is no constant with name `?0` in scope

46006 There is no definition or axiom with key `?0` in the declaration theory of the constant

46009 `?0` is not a constant, or a constant applied to some arguments
\[\textbf{val } \text{z\_intro\_gen\_pred\_tac} : (\text{TERM } \times \text{ TERM}) \text{ list } \rightarrow \text{TACTIC};\]

\textbf{Description} A tactic to introduce a generic predicate as the goal. The term list argument pairs is of a term and a variable (that is appropriate to be a generic formal), with the same set type i.e. the second is of the form \[\text{\{Z\_var : P\_\var\}}\].

\begin{align*}
\text{Tactic} & \quad \Gamma \vdash t[\ldots] \\
\text{\{z\_intro\_gen\_pred\_tac\}} \quad \Gamma \vdash X[\ldots] \quad (t[X],\ldots)
\end{align*}

where either \(t_i\) is the same as \(X_i\), or \(X_i\) does not appear free in the conclusion, \(t[t\ldots]\), of the original goal.

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

\textbf{Errors}
\begin{itemize}
\item 28082 \(0\) does not appear free in the goal
\item 28083 \(0\) appears free in the goal and is not the same as \(\ddagger 1\)
\item 41032 \(0\) is not of the form: \[\text{\{Z\_var : P\_\var\}}\]
\item 41036 \(0\) does not have the same type as \(\ddagger 1\)
\end{itemize}

\[\textbf{val } \text{z\_intro\_\forall\_tac} : \text{TERM } \rightarrow \text{TACTIC};\]

\textbf{Description} Introduce a Z universal with reference to a binding.

\begin{align*}
\text{Tactic} & \quad \Gamma \vdash t[\ldots] \\
\text{\{z\_intro\_\forall\_tac\}} \quad \Gamma \vdash \forall x_1 : U; \ldots \texttt{true}\cdot t[x_1/t,\ldots]
\end{align*}

\textbf{Errors}
\begin{itemize}
\item 41029 \(0\) cannot be interpreted to be of the form: \[\text{\{Z\_var : P\_\var\}}\]
8.2. Reasoning about Predicates

SML

```
val z_para_pred_canon : CANON;
val z_para_pred_conv : CONV;
```

**Description**  This canonicalisation function and conversion change Z paragraphs to Z predicates. This change is also the one `z_get_spec` does, where appropriate.

Some paragraphs entered by the Z parser have “markers” applied to the rest of the theorem body to indicate their origin (i.e. the kind of paragraph). In addition the form of the term is likely to have an explicit declaration as the left conjunct, rather than a “predicate implicit in a declaration”. `z_para_pred_canon` is a canonicalisation function that removes these markers and converts, if present, a left conjunct declaration as `z_decl_pred_conv` would; `z_para_pred_conv` is a conversion that has the equivalent effect.

The following are instances in which markers are used:

- **Constraint Definitions**
- **Free Type Definitions**
- **Given Set Definitions**
- **Axiomatic Definitions**
- **Schema Boxes**
- **Abbreviation Definitions**

**Example**

*If the following is entered:*

```
⟦X, Y⟧ =
⟦\exists X : P(X \times Y)
\exists Ex = {}⟧
```

`z_para_pred_canon` given the defining theorem, returns a singleton list containing:

```
\forall X Y \exists Ex[X, Y] \in P(X \times Y) \land Ex[X, Y] = {}\uparrow
```

Both functions remain within the Z language, though this is not checked, with the caveat on HOL universals representing generic formals.

**Errors**

- `41080` No Z markers found applied to conclusion body of ?0
- `41082` No Z markers found applied to body of ?0
Chapter 8. SUPPORT FOR Z

**SML**

\[\text{val } \text{z} \_ \text{pred} \_ \text{decl} \_ \text{conv} : \text{CONV};\]

**Description** A conversion which, given a predicate comprising a conjunction of the forms recognised by \(\text{z} \_ \text{pred} \_ \text{dec} \_ \text{conv}\), rewrites the predicate as a declaration.

The function is defined much as if by the following:

\[\text{val } \text{z} \_ \text{pred} \_ \text{decl} \_ \text{conv} = \text{Z} \_ \text{DECL} \_ \text{INTRO} \_ \text{C} \text{ z} \_ \text{dec} \_ \text{pred} \_ \text{conv};\]

Thus the handling of the conjuncts is as shown in the following examples:

**Conversion**

\[\vdash \; x \in X \land \{y, z\} \subseteq Y \land S \iff \frac{z \_ \text{pred} \_ \text{decl} \_ \text{conv}}{\frac{x \in X \land \{y, z\} \subseteq Y \land S^\gamma}{\text{z} \_ \text{decl} \_ \text{of} \_ z \_ x \_ \text{decl} \_ \text{conv}}};\]

and

**Conversion**

\[\vdash \; \text{true} \iff \frac{z \_ \text{decl} \_ \text{of} \_ z \_ x \_ \text{decl} \_ \text{conv}}{\frac{\text{true}^\gamma}{z \_ \text{pred} \_ \text{decl} \_ \text{conv}}};\]

**See Also** \(\text{z} \_ \text{decl} \_ \text{pred} \_ \text{conv}\)

**Errors** 41011 ?0 cannot be rewritten to a declaration

---

**SML**

\[\text{val } \text{z} \_ \text{pred} \_ \text{dec} \_ \text{conv} : \text{CONV};\]

**Description** A conversion which, given a certain form of predicate, rewrites the predicate as the \text{dec} component of a declaration. This acts as an inverse to the conversion \(\text{z} \_ \text{dec} \_ \text{pred} \_ \text{conv}\), the four forms recognised being as shown below:

**Conversion**

\[\vdash \; x \in X \iff \frac{z \_ \text{pred} \_ \text{dec} \_ \text{conv}}{\frac{x \in X^\gamma}{\text{z} \_ \text{dec} \_ \text{of} \_ z \_ x \_ \text{dec} \_ \text{conv}}};\]

where the \(x\) must be variable, and

**Conversion**

\[\vdash \{x_1, \ldots\} \subseteq X \iff \frac{z \_ \text{pred} \_ \text{dec} \_ \text{conv}}{\frac{\{x_1, \ldots\} \subseteq X^\gamma}{\text{z} \_ \text{dec} \_ \text{of} \_ z \_ \{x_1, \ldots\} \_ \text{dec} \_ \text{conv}}};\]

where the \(x_i\) must be variables, and

**Conversion**

\[\vdash \; S \iff \frac{z \_ \text{pred} \_ \text{dec} \_ \text{conv}}{\frac{S^\gamma}{z \_ \text{dec} \_ \text{of} \_ z \_ S \_ \text{dec} \_ \text{conv}}};\]

and

**Conversion**

\[\vdash \; (\theta S \in S) \iff \frac{z \_ \text{pred} \_ \text{dec} \_ \text{conv}}{\frac{\theta S \in S^\gamma}{z \_ \text{dec} \_ \text{of} \_ z \_ (\theta S \in S) \_ \text{dec} \_ \text{conv}}};\]

**See Also** \(\text{z} \_ \text{dec} \_ \text{pred} \_ \text{conv}\)

**Errors** 41011 ?0 cannot be rewritten to a declaration
8.2. Reasoning about Predicates

SML

\begin{verbatim}
|val z_push_consistency_goal : TERM \rightarrow unit;

Description z_push_consistency_goal \textcircled{z} const will first determine the specification theorem of const, by executing z_get_spec. The const may either be a constant, or a constant applied to a list of arguments. If this theorem has an assumption, it will then push that specification assumption onto the stack of subgoals (using push_subgoal, q.v.), as a goal with no assumptions. By how z_get_spec is designed, this (single) assumption will be of the form:

\[
\text{⌜Consistent}(\lambda vs[x_1,...,x_n]\bullet p[x_1,...,x_n])\quad
\]

or the consistency has already been proven, and saved, under some assumptions. Only in the former case will the function continue: it will apply a tactic (that may be undone by undo) which rewrites the goal to:

\[
\text{⌜(\[] \quad \text{⌜Z exists } D[x_1,...,x_n]\bullet p[x_1,...,x_n]\quad
\]

where D is a declaration of the variables, x_1,...,x_n representing the existence witnesses of the n constants declared in one paragraph. Otherwise, if the definition involves generic formals:

\[
\text{⌜(\[] \quad \text{⌜Z exists } D[x_1,...,x_n]\bullet p[x_1,...,x_n]\quad
\]

If not, the function fails.

See Also save_consistency_thm to save the result in a conventional manner.
\end{verbatim}

Errors

\begin{verbatim}
\begin{enumerate}
\item 46005 There is no constant with name ?0 in scope
\item 46006 There is no definition or axiom with key ?0 in the declaration theory of the constant
\item 46007 Specification of ?0 is not of the form: 'Consistent (\lambda vs[x_1,...,x_n]\bullet p[x_1,...,x_n]) \vdash ...
\item 46009 ?0 is not a constant, or a constant applied to some arguments
\end{enumerate}
\end{verbatim}

SML

\begin{verbatim}
|val z_quantifiers_elim_tac : TACTIC;

Description This tactic eliminates all Z \forall, \exists and \exists_1 quantifiers in both conclusion and assumptions, in favour of HOL \forall and \exists, using z_\forall_elim_conv2, z_\exists_elim_conv2, z_\exists_1_conv1. All declarations introduced will be converted to their implicit Z predicates, and the following simplifications also done throughout:

\[
\begin{align*}
\text{⌜\{x,y,...\} \subseteq s\quad \rightarrow \quad } & x \in s \land y \in s \land ... \\
\text{⌜x \land true\quad \rightarrow \quad } & x \\
\text{⌜true \land x\quad \rightarrow \quad } & x \\
\text{⌜x \Rightarrow true\quad \rightarrow \quad } & true
\end{align*}
\]

All assumptions will be stripped back into the assumption list, regardless of whether they were modified, using the current proof context.

This is done to prepare for some further processing, such as resolution. The result is unlikely to be in the Z language It has no effect (rather than failing) if there are no conversions to be done.

Uses Intended for implementing automated proof procedures.
\end{verbatim}
Chapter 8. SUPPORT FOR Z

val z_schema_pred_conv1 : CONV;

**Description** Convert an ill-formed schema as a predicate expression into a statement of a binding being a member of the schema. The input expression is ill-formed if it is of the form

\[ Z'\text{SchemaPred bind schema} \]

where `bind` is not equal to \[ \ztheta \text{schema} \].

<table>
<thead>
<tr>
<th>Conversion</th>
<th>[ \vdash Z'\text{SchemaPred bind schema} \iff \text{bind} \in \text{schema} ]</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>z_schema_pred_conv1</code></td>
<td>[ \ztheta \text{Z'SchemaPred bind schema} ]</td>
</tr>
</tbody>
</table>

**Uses** In correcting the results of functions that leave Z because of substituting into the binding portion of a schema as predicate. In particular, in the proof context "Z_predicates".

**Errors**

41018

?0 is not an ill-formed schema as predicate expression

---

val z_setd_c_conv : CONV

**Description** Expand out expressions that state that a set display is a subset of some other set. This is particularly aimed at processing declarations of the form \( x_1, \ldots, x_n : X \).

<table>
<thead>
<tr>
<th>Conversion</th>
<th>[ \z{x_1, \ldots} \subseteq X \iff (x_1 \in X \land \ldots) ]</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>z_setd_c_conv</code></td>
<td>[ \z{x_1, \ldots} \subseteq X ]</td>
</tr>
</tbody>
</table>

and

<table>
<thead>
<tr>
<th>Conversion</th>
<th>[ \z{} \subseteq X \iff \text{true} ]</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>z_setd_c_conv</code></td>
<td>[ \z{} \subseteq X ]</td>
</tr>
</tbody>
</table>

The conversion will all simplify certain subterms involving `true` or terms of the form \( x = x \).

**See Also** `z_setd_e_conv`

**Errors**

41017

?0 is not of the form: \[ \z{x_1, \ldots} \subseteq X \]
8.2. Reasoning about Predicates

SML

```sml
val z_spec_asm_tac : TERM -> TERM -> TACTIC;
val z_spec_nth_asm_tac : int -> TERM -> TACTIC;
```

**Description** These are two methods of specialising a Z universally quantified assumption. Both leave the old assumption in place, and place the instantiated assumption onto the assumption list using `strip_asm_tac`. If the desired behaviour differs from any of those supplied then use `GET_ASM_T` and its cousins, with `z_v_elim`, to create the desired functionality.

**Tactic**

```
\{ \Gamma; \ \forall D[x_1,...] \ | \ P_1 \bullet P_2^{-}\} \ t \quad \rightarrow \quad \{ \Gamma; \ \forall D'[x_1,...] \ | \ P_1' \bullet P_2'^{-}\} \ t
```

where $D'$, $P_1'$ and $P_2'$ are specialised, and if necessary have bound variable renaming done, appropriately. Remains within the Z language (though failure to do this will be reported to be from `z_v_elim`).

**Definition**

```
fun z_spec_asm_tac (asm:TERM) (bind:TERM):TACTIC =
    GET_ASM_T asm (strip_asm_tac o z_v_elim bind);
fun z_spec_nth_asm_tac (n:int) (bind:TERM):TACTIC =
    GET_NTH_ASM_T n (strip_asm_tac o z_v_elim bind);
```

**Errors** As the constituents of the implementing functions (e.g. `GET_ASM_T` and `z_v_elim`).

SML

```sml
val z_term_of_type : TYPE -> TERM;
val z_type_of : TERM -> TERM;
```

**Description** `z_term_of_type ty` is a term denoting the set of all elements of the type `ty`. The term is constructed using `mk_z_P`, `mk_z_X`, `mk_z_h_schema`, given sets, and the relation symbol $\in$ in order to display the structure of the type in a Z-like way. `mk_u` is used when all else fails.

For example:

```
z_term_of_type(type_of $\subseteq$ (1, 2, 3)) = $\subseteq$ Z $\rightarrow$ Z^n
z_term_of_type(type_of $\subseteq$ (a\\in\{1\}, b\\in\{2\}, c\\in\{3\})) = $\subseteq a, b, c : Z^n
```

Note that the quotation in the last example contains an HOL list display, the type of which, namely `\in\{N LIST\}`, lies outside the representation of the Z type system in HOL.

`z_type_of` returns the set of all elements of the (HOL) type of a particular term.

**Definition**

```
val z_type_of = z_term_of_type o type_of;
```
val z ∈ setd_conv : CONV;

Description  A conversion of membership of a Z set display into equality with a member of the set.

\[
\vdash t \in \{ t_1, t_2, \ldots \} = \frac{z \in \text{setd_conv}}{\forall t \in \{ t_1, t_2, \ldots \}}
\]

\[
(t = t_1) \lor (t = t_2) \ldots
\]

See Also  z ∈ setd_conv1

Errors

41015  ?0 is not of the form: \[\forall x \in \{ t_1, \ldots \}\]

41016  ?0 is an ill-formed fragment of the membership of a set display
8.2. Reasoning about Predicates

SML

\[ \text{val } z \in \_ \_ \_ u \text{ conv } : \text{ CONV}; \]

**Description**  Simplifies to true a predicate of the forms: \( \forall x \in S[U] \), \( \forall x \subseteq S[U] \) or a schema as a predicate: \( \forall [a, b : S[U]; c : S'[U]; ...] \), where \( S[U], S'[U], ... \) are structures that can be simplified to \( U \). This uses the application of the built-in simplifications listed below, and conversions held in the “icl’u_simp” entry of the dictionary of nets field of the current proof context (the built-in’s taking precedence).

**Conversion**

\[
\begin{align*}
\forall x \in S[U] & \iff true \\
\forall x \subseteq S[U] & \iff true \\
\forall [a, b : S[U]; c : S'[U]; ...] & \iff true
\end{align*}
\]

The conversion starts with the structure \( S[U] \) above. It will attempt to recursively prove equal to \( U \): the argument to \( P \), the constituent sets of a cartesian tuple, the types of a declaration part of a set abstraction with a true predicate, and the types of a declaration part of a horizontal schema with a true predicate. If it can do so it will then use:

\[
\begin{align*}
\vdash P U & = U \\
\vdash (U \times U \times ...) & = U \\
\vdash \{ \text{lab1 : U; lab2 : U; lab3,lab4 : U; ... } \} & = U \\
\vdash [\text{lab1 : U; lab2 : U; lab3,lab4 : U; ... }] & = U
\end{align*}
\]

to prove the set equal to \( U \). If it cannot complete the above proof it will use the first applicable conversion of the “icl’u_simp” entry of the dictionary of nets field of the current proof context, and then return to attempting to use the built-in algorithm.

If the set has been reduced to \( U \) the conversion will prove the input term true. If the expression cannot be proven the conversion fails.

**Uses**  For stripping in proof contexts, and in eliminating redundant declarations that have been converted to the predicates implicit in them.

**See Also**  \( u \_ \_ \_ simp \_ eqn \_ ext \), \( theory \_ u \_ \_ \_ simp \_ eqn \_ ext \), and \( set \_ u \_ \_ \_ simp \_ eqn \_ ext \) for creating appropriate proof contexts.

**Errors**

\[41061\] cannot prove \(?0 equal to \( \forall x \) true? \]
\[41062\] ?0 is not of the form: \( \forall x \subseteq s \), \( \forall x \in s \) or a schema as a predicate
**SML**

```sml
val z_¬_gen_pred_conv : CONV;
```

**Description** Convert a negated generic predicate (which is not legal Z) into an existentially quantified negation (and therefore into Z).

Conversion

\[ \Gamma \vdash (\neg \neg Z[x_1,\ldots,\bullet \neg pred]) \iff \exists x_1 : U : \ldots \bullet \neg pred \]

**Uses** In stripping for repaired the effects of, e.g., `contr_tac`.

**Errors**

| 41031 | ?0 is not of the form: \( \neg \neg Z[x_1,\ldots] \) pred\^\neg

---

**SML**

```sml
val z_¬_in_conv : CONV;
```

**Description** This is a conversion which moves an outermost negation inside other Z predicate calculus connectives using whichever of the following rules applies:

\[
\begin{align*}
\neg t & = t \\
\neg(t_1 \land t_2) & = \neg t_1 \lor \neg t_2 \\
\neg(t_1 \lor t_2) & = \neg t_1 \land \neg t_2 \\
\neg(t_1 \Rightarrow t_2) & = t_1 \land \neg t_2 \\
\neg(t_1 \Leftrightarrow t_1) & = \text{false} \\
\neg(t_1 \Leftrightarrow t_2) & = (t_1 \land \neg t_2) \lor (t_2 \land \neg t_1) \\
\neg(t_1 = t_1) & = \text{false} \\
\neg(\forall D \mid P \bullet V) & = \exists D \mid P \bullet \neg V \\
\neg(\exists D \mid P \bullet V) & = \forall D \mid P \bullet \neg V \\
\neg(\exists_1 D \mid P \bullet V) & = \forall D | (V \land \forall D' | P' \bullet V' \Rightarrow D = D') \\
\neg \text{true} & = \text{false} \\
\neg \text{false} & = \text{true}
\end{align*}
\]

**Uses** Tactic and conversion programming.

**Errors**

| 47240 | ?0 is not a Z negation
| 28131 | No applicable rules for the term ?0

---

**SML**

```sml
val z_¬_rewrite_canon : THM -> THM list
```

**Description** This is a canonicalisation function used for breaking theorems up into lists of equations for use in rewriting. It performs the following transformations:

\[
\begin{align*}
z_¬_rewrite_canon (\Gamma \vdash \neg(t_1 \lor t_2)) & = (\Gamma \vdash \neg t_1 \land \neg t_2) \\
z_¬_rewrite_canon (\Gamma \vdash \neg \exists D \mid P \bullet V) & = (\Gamma \vdash \forall D \mid P \bullet \neg V) \\
z_¬_rewrite_canon (\Gamma \vdash \neg t) & = (\Gamma \vdash t) \\
z_¬_rewrite_canon (\Gamma \vdash \neg t) & = (\Gamma \vdash t \Leftrightarrow \text{false})
\end{align*}
\]

Remains within the Z language, though this is not checked.

**See Also** `simple_¬_rewrite_canon`, `simple_\forall_rewrite_canon`.

**Errors**

| 26201 | Failed as requested

The area given by the failure will be `fail_canon`.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
8.2. Reasoning about Predicates

SML

val z_¬_∀_conv : CONV;
val z_¬_∃_conv : CONV;

Description  $z_\neg \forall$ conv converts a negated Z universal quantification to a Z existential quantification.

Conversion

$$\vdash \neg(\forall D \mid P_1 \bullet P_2) \iff (\exists D \mid P_1 \bullet \neg P_2)$$

The dual is $z_\neg \exists$ conv:

Conversion

$$\vdash \neg(\exists D \mid P_1 \bullet P_2) \iff (\forall D \mid P_1 \bullet \neg P_2)$$

These two functions remain within the Z language, though this is not checked.

Errors

41050  ?0 not of the form: $z_\neg(\forall D \mid P_1 \bullet P_2)^\gamma$
41051  ?0 not of the form: $z_\neg(\exists D \mid P_1 \bullet P_2)^\gamma$

SML

val z_⇒_rewrite_canon : CANON;

Description  This canonicalisation expects to be passed the canonicalisations of, e.g., a Z universal or the result of a $z_\forall$ elim. These are theorems of the form:

$$\vdash "predicate from D" \land P \Rightarrow V$$

In these cases it is intended to prove and discard "predicate from D" whose conjuncts can be proven true by $z_\in\_u$ conv (q.v.), and a P that is identically true.

In fact, each conjunct of the antecedent of the supplied theorem will have $z_\in\_u$ conv attempted upon it, the resulting antecedent will be rewritten with the theorems

$$\vdash \forall x:U \bullet x \land true \iff x$$
$$\vdash \forall x:U \bullet true \land x \iff x$$

and if the antecedent is thus proven true it will be discarded. Remains within the Z language, though this is not checked.

Errors

41083  ?0 is not of the form: $\Gamma \vdash P \Rightarrow Q$
41084  caused no change with ?0
\textbf{SML}
\begin{verbatim}
val z_\forall_elim_conv1 : CONV;
\end{verbatim}

\textbf{Description}  Turn a Z universally quantified predicate into a HOL universally quantified term, eliminating the declaration part of the original quantification using \(z \in u_{\text{conv}}\). The function fails if the declaration cannot be eliminated.

\textbf{Conversion}
\[
\vdash (\forall D[x_1,\ldots] | P_1 \cdot P_2) \iff \forall \, x_1 \ldots \forall P_1 \Rightarrow P_2 \neg \neg
\]
\(z_\forall\_elim\_conv1\)
\[
\neg \forall \, D[x_1,\ldots] | P_1 \cdot P_2 \neg \neg
\]

The order of the resulting universally quantified variables will be in a sorted order, rather than what the declaration part might suggest.

Simplifications based on \(P_i\) being \textit{true} or \textit{false} will also be applied.

If there are no quantified variables and the declaration is \(D[]\), the HOL universal quantification is not generated.

Remains within the Z language (with the caveat of using outer HOL universal quantification).

\textbf{Uses}  For stripping in proof contexts.

\textbf{See Also}  \(z_\forall\_elim\_conv2\) and \(z_\forall\_elim\_conv\)

\textbf{Errors}
- \texttt{401022}  ?0 is not of the form: \(\forall \forall \, D | P_1 \cdot P_2 \neg \neg\)
- \texttt{401071}  ?0 is of the form: \(\forall \forall \, D | P_1 \cdot P_2 \neg \neg\) but could not eliminate \(D\)
8.2. Reasoning about Predicates

```
val z_\forall_elim_conv2 : CONV;
val z_\forall_intro_conv1 : CONV;
```

**Description**  
\(z_\forall_elim_conv2\) turns a Z universally quantified predicate into a HOL universally quantified term. The result fails to be in the Z language because it contains a declaration used in a position requiring a predicate, which Z does not allow.

\[
\begin{align*}
\vdash (\forall D[x_1,...] | P_1 \cdot P_2) \iff \\
\forall x_1 ... \cdot \exists D[x_1,...] \land \exists P_1 \Rightarrow \exists P_2 \land
\end{align*}
\]

The order of the resulting universally quantified variables will be in a sorted order, rather than what the declaration part might suggest.

\(z_\forall_intro_conv1\) undoes this process, returning a theorem whose RHS should be in the Z language.

\[
\begin{align*}
\vdash \forall x_1 ... \cdot \exists D[x_1,...] \land P_1 \Rightarrow P_2 \iff \\
\forall x_1 ... \cdot \exists D[x_1,...] \land P_1 \Rightarrow P_2
\end{align*}
\]

If there are no quantified variables and the declaration is \(D[]\), the HOL universal quantification is not generated by \(z_\forall_elim_conv2\) nor expected by \(z_\forall_intro_conv1\).

**Uses**  
Used in the Z form of \(\text{strip_tac}\), and handling negations with quantifiers. It will handle paired quantifiers, and quantifiers in any order, so long as the quantifiers and declaration can be matched in names and types.

**See Also**  
\(z_\forall_elim_conv1\), \(z_\forall_elim_conv\), \(z_\forall_intro_conv\)

**Errors**

41022 0 is not of the form: \(\exists \forall D | P_1 \cdot P_2\land\)

41023 ?0 is not of the form: \(\forall x_1 ... \cdot \exists Decl \land P_1 \Rightarrow P_2\land\)

41024 ?0 is not of the form: \(\forall x_1 ... \cdot Decl \land P_1 \Rightarrow P_2\land\)

where the \(\exists x_i\) do not match the declaration.

```
val z_\forall_elim_conv : CONV;
```

**Description**  
Turn a Z universally quantified predicate into a HOL universally quantified term, converting the declaration part of the original quantification into its implicit predicate.

\[
\begin{align*}
\vdash (\forall D[x_1,...] | P_1 \cdot P_2) \iff \\
\forall x_1 ... \cdot \exists P_1 \Rightarrow P_2 \land
\end{align*}
\]

The order of the resulting universally quantified variables will be in a sorted order, rather than what the declaration part might suggest.

If there are no quantified variables and the declaration is \(D[]\), the HOL universal quantification is not generated.

Remains within the Z language (with the caveat of using outer HOL universal quantification).

**Errors**

41022 0 is not of the form: \(\exists \forall D | P_1 \cdot P_2\land\)
\textbf{SML} \[ \text{val } z\_\forall\_\text{elim} : \text{TERM} \rightarrow \text{THM} \rightarrow \text{THM}; \]

\textbf{Description} Specialise the variables introduced by a Z universally quantifier to given values of the right type, the values being taken from a binding.

\textbf{Rule} \[
\frac{
\Gamma \vdash \forall D[x_1, \ldots] \mid P_1[x_1, \ldots] \bullet P_2[x_1, \ldots]
}{
\Gamma \vdash \text{"predicate from } D'[t_1, \ldots]" \\land \ P'_1[t_1, \ldots] \Rightarrow P'_2[t_1, \ldots]
}
\]

\[ z\_\forall\_\text{elim} \quad \zeta(x_1 \equiv t_1, \ldots) \gamma \]

where \( D \) is a declaration that binds the \( x_i \), that is converted to its implicit predicate by \( z\_\text{decl\_pred\_conv} \). The theorem may be type instantiated or require bound variable renaming to allow the specialisation to be valid, thus the priming in the result.

If both the supplied binding and the declaration are recognisably “empty” then the universal quantification will be eliminated.

If instead the theorem’s conclusion has a single universally quantified variable and the theorem can be type instantiated to match the supplied argument, then that supplied argument will be used directly.

\textbf{Rule} \[
\frac{
\Gamma \vdash \forall x:X \mid P_1[x] \bullet P_2[x]
}{
\Gamma \vdash t \in X' \land P'_1[t] \Rightarrow P'_2[t]
}
\]

\[ z\_\forall\_\text{elim} \quad \zeta t \gamma \]

If neither of the above apply then the supplied binding may instead be anything else that has an appropriate binding type. In such cases projection functions will be used:

\textbf{Rule} \[
\frac{
\Gamma \vdash \forall D[x_1, \ldots] \mid P_1[x_1, \ldots] \bullet P_2[x_1, \ldots]
}{
\Gamma \vdash \text{"predicate from } D'[t.x_1, \ldots]" \land \ P'_1[t.x_1, \ldots] \Rightarrow P'_2[t.x_1, \ldots]
}
\]

\[ z\_\forall\_\text{elim} \quad \zeta t \gamma \]

\textbf{See Also} \( z\_\forall\_\text{elim\_conv2} \)

\textbf{Errors} \[ 47310 \ ?0 \text{ is not a } Z \text{ universal quantification} \]
\[ 41021 \ ?0 \text{ cannot be interpreted as a binding that matches } ?1 \]
8.2. Reasoning about Predicates

**val z\_\forall\_intro1 : THM -> THM;**

**Description**  A rule to introduce a Z universal quantification. The variables to be quantified over must not occur free in the assumptions, and are determined from the form of the input theorem.

\[
\frac{\Gamma \vdash "predicate from D" \land P_1 \Rightarrow P_2}{\Gamma \vdash \forall D \mid P_1 \bullet P_2 \quad \text{z\_\forall\_intro1}}
\]

where “predicate from D” is converted to a declaration in which this predicate is implicit by Z\_DECL\_INTRO\_C z\_pred\_dec\_conv.

An arbitrary conjunctive structure is allowed in “D as a predicate”, including repeated bindings of single variables: only the ordering, as opposed to the nesting is significant in the conjunctive structure. The predicate true is converted to the empty declaration.

**See Also**  z\_\forall\_intro for implicit \(x_i \in U\) conjuncts, all\_z\_\forall\_intro, z\_\forall\_intro\_conv1.

**Errors**

- 6005 ?0 occurs free in assumption list
- 41026 ?0 not of the form ‘\(\Gamma \vdash "predicate from D" \land P_1 \Rightarrow P_2\)’
- 41027 ?0 cannot be made into a declaration

---

**val z\_\forall\_intro\_conv : CONV;**

**Description**  z\_\forall\_intro\_conv converts an arbitrary simple HOL universally quantified term into the corresponding Z, returning a theorem whose RHS should be in the Z language (though this is not checked for).

\[
\vdash \forall x_1 \ldots \bullet P \iff (\forall x_1 : U; \ldots \mid \text{true} \bullet P) \quad \text{z\_\forall\_intro\_conv}
\]

This conversion cannot introduce a Z universal quantification with an empty declaration.

**See Also**  z\_\forall\_intro\_conv1

**Errors**

- 41047 ?0 is not of the form: \(\forall x_1 \ldots \bullet P\)
val z_\forall_intro : TERM list \rightarrow THM \rightarrow THM;

**Description** A rule to introduce a Z universal quantification. The variables to be quantified over must not occur free in the assumptions, and are determined from the variables from the supplied list.

\[
\Gamma \vdash P_1 \Rightarrow P_2 \\
\Gamma \vdash \forall x_1:U;... \mid P_1 \mathbin{\&} P_2 \]  

or else:

\[
\Gamma \vdash P \\
\Gamma \vdash \forall x_1:U;... \mid \text{true} \mathbin{\&} P_2 \]  

An arbitrary conjunctive structure is allowed, including repeated bindings of single variables: only the ordering, as opposed to the nesting is significant in the conjunctive structure.

**See Also** z_\forall_intro1 for use without additional \(x_i \in U\), all_z_\forall_intro, z_\forall_intro_conv1.

**Errors**

3007 ?0 is not a term variable
6005 ?0 occurs free in assumption list

---

val z_\forall_inv_conv : CONV;

**Description** Simplifies a Z universal quantification whose predicate or constraint is invariant w.r.t. the free variables bound by the declaration.

\[
\vdash \forall D \mid P_1 \mathbin{\&} P_2 \iff (\forall D \mid P_1 \mathbin{\&} \text{false}) \mathbin{\lor} P_2 \]  

if \(P_2\) has no free variables bound by \(D\), and

\[
\vdash \forall D \mid P_1 \mathbin{\&} P_2 \iff P_1 \Rightarrow (\forall D \mid \text{true} \mathbin{\&} P_2) \]  

if \(P_1\) has no free variables bound by \(D\), and

\[
\vdash \forall D \mid P_1 \mathbin{\&} P_2 \iff P_1 \Rightarrow (\forall D \mid \text{true} \mathbin{\&} \text{false}) \mathbin{\lor} P_2 \]  

if both have no free variables bound by \(D\). The appropriate simplification will be avoided where the predicate \(P_1\), is \(\forall \text{true}\) or the value, \(P_2\) is \(\forall \text{false}\).

**See Also** z_\exists_inv_conv

**Errors**

47310 ?0 is not a Z universal quantification
41025 ?0 is not of the form: \(\forall D \mid P_1 \mathbin{\&} P_2\) where at least one of \(P_1\) or \(P_2\) are unbound by \(D\)
8.2. Reasoning about Predicates 409

val z_∀_rewrite_canon : CANON;

Description  Take a possibly Z universally quantified theorem and make it into, as far as possible, a HOL universally quantified theorem usable for rewriting.

\[
\begin{align*}
\Gamma \vdash \forall D[x_1,\ldots] \mid P_1 \cdot P_2)\forall
\Gamma \vdash \forall x_1 \ldots \bullet
\end{align*}
\]

See \textit{z\_decl\_pred\_conv} for a description of the conversion of a declaration to its implicit predicate. Remains within the Z language (under the relaxation that allows outermost HOL universals), though this is not checked.

See Also  \textit{z\_defn\_canon}

Errors

\textbf{41081}  \(?0 is not of the form: \(\forall D \mid P_1 \bullet P_2)\forall

val z_∀_tac : TACTIC;

Description  Eliminate a Z universal in a goal.

\[
\begin{align*}
\{ \Gamma \} \forall D \mid P \bullet V
\end{align*}
\]

\(D\) is converted to its implicit predicate by \textit{z\_decl\_pred\_conv}. \(D\), \(P\) and \(V\) are primed in the result because the tactic may require some renaming to avoid, e.g. variable capture of the names of free variables in the assumption list.

Errors

\textbf{41030} Conclusion of the goal is not of the form: \(\forall D \mid P \bullet V)\forall

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
val z_elim_conv1 : CONV;

**Description**  Turn a Z existentially quantified predicate into a HOL existentially quantified term, eliminating the declaration part of the original quantification using `z_elim_conv1`. The function fails if the declaration cannot be eliminated.

**Conversion**

\[
\vdash (\exists D[x_1,\ldots] | P_1 \& P_2) \iff \exists x_1 \ldots \exists P_1 \& P_2
\]

The order of the resulting existentially quantified variables will be in a sorted order, rather than what the declaration part might suggest.

Simplifications based on \(P_i\) being `true` or `false` will also be applied.

If there are no quantified variables and the declaration is `D[]`, the HOL existential quantification is not generated.

**Uses**  For stripping in proof contexts.

**See Also**  `z_elim_conv2`, `z_elim_conv`

**Errors**

41042 ?0 is not of the form: \(\exists D | P_1 \& P_2\)

41043 ?0 is of the form: \(\exists D | P_1 \& P_2\), but \(D\) non-trivial
8.2. Reasoning about Predicates

SML

```sml
val _∃_elim_conv2 : CONV;
val _∃_intro_conv1 : CONV;
```

**Description**  
_z._∃_elim_conv2 turns a Z existentially quantified predicate into a HOL existentially quantified term. The result fails to be in the Z language because it contains a declaration used in a position that requires a predicate, which Z does not allow, as well as the outer HOL existential quantification.

**Conversion**  

\[
\begin{align*}
\vdash (\exists D[x_1,...] | P_1 \cdot P_2) & \iff \_∃_elim_conv2 \\
\exists x_1 ... \cdot \exists D[x_1,...] \land \exists P_1 \land \exists P_2
\end{align*}
\]

The order of the resulting existentially quantified variables will be in a sorted order, rather than what the declaration part might suggest.

_z._∃_intro_conv1 undoes this process, returning a theorem whose RHS should be in the Z language (though this is not checked for).

**Conversion**  

\[
\begin{align*}
\vdash \forall x_1 ... \cdot \exists D[x_1,...] \land P_1 \land P_2 \iff \_∃_intro_conv1 \\
\exists x_1 ... \cdot \exists D[x_1,...] \land P_1 \land P_2
\end{align*}
\]

If there are no quantified variables and the declaration is \(D[\]\), the HOL existential quantification is not generated by _z._∃_elim_conv2 nor expected by _z._∃_intro_conv1.

**Uses**  
Used in the Z form of strip_tac, and handling negations with quantifiers.

**See Also**  
_z._∃_elim_conv1, _z._∃_elim_conv and _z._∃_intro_conv

**Errors**  
41044 ?0 is not of the form: \(\exists D | P_1 \cdot P_2\)

41045 ?0 is not of the form: \(\exists x_1 ... \cdot D[x_1,...] \land P_1 \land P_2\)

41041 ?0 is not of the form: \(\forall x_1 ... \cdot D \land P_1 \Rightarrow P_2\)

where the \(\exists x_i\) do not match the declaration.
val z_∃_elim_conv : CONV;

Description  Turn a Z existentially quantified predicate into a HOL existentially quantified term, converting the declaration part of the original quantification into its implicit predicate.

Conversion

\[ \vdash (\exists D[x_1,...] | P_1 \cdot P_2) \iff \exists x_1 \ldots \bullet \exists \text{"predicate from } D[x_1,...]" \land P_1 \land P_2 \neg \neg \]

The order of the resulting existentially quantified variables will be in a sorted order, rather than what the declaration part might suggest.

If there are no quantified variables and the declaration is \([\), the HOL existential quantification is not generated.

The result fails to be within the Z language, but only due to the outer HOL existential quantification.

See Also  z_∃_elim_conv1, z_∃_elim_conv2

Errors

41042  ?0 is not of the form: \[\exists D | P_1 \cdot P_2\neg\neg\]

val z_∃_intro_conv : CONV;

Description  z_∃_intro_conv converts an arbitrary simple HOL existentially quantified term into the corresponding Z, returning a theorem whose RHS should be in the Z language (though this is not checked for).

Conversion

\[ \vdash \exists x_1 \ldots \bullet P \neg \iff (\exists x_1 : \mathbb{U}; ... | \text{true} \bullet P) \]

This conversion cannot introduce a Z existential quantification with an empty declaration.

See Also  z_∃_intro_conv1

Errors

41046  ?0 is not of the form: \[\exists x_1 \ldots \bullet P\neg\]
SML

\[ \text{val } z \_ \exists \_ \text{inv} \_ \text{conv } : \text{CONV}; \]

**Description**  Simplifies a Z existential quantification whose predicate or constraint is invariant w.r.t. the free variables bound by the declaration.

Conversion

\[
\frac{\vdash D \mid P_1 \bullet P_2}{z \_ \exists \_ \text{inv} \_ \text{conv}} \quad \frac{z \_ \exists \_ \text{inv} \_ \text{conv}}{zM D \mid P_1 \bullet P_2} \quad \text{if } P_2 \text{ has no free variables bound by } D, \text{ and}
\]

\[
\frac{\vdash D \mid P_1 \bullet P_2}{z \_ \exists \_ \text{inv} \_ \text{conv}} \quad \frac{z \_ \exists \_ \text{inv} \_ \text{conv}}{zM D \mid P_1 \bullet P_2} \quad \text{if } P_1 \text{ has no free variables bound by } D, \text{ and}
\]

\[
\frac{\vdash D \mid P_1 \bullet P_2}{z \_ \exists \_ \text{inv} \_ \text{conv}} \quad \frac{z \_ \exists \_ \text{inv} \_ \text{conv}}{zM D \mid P_1 \bullet P_2} \quad \text{if both have no free variables bound by } D.
\]

\( P_1 \) nor \( P_2 \) will be “extracted” if identical to \( \text{true} \).

**See Also**  \( z \_ \forall \_ \text{inv} \_ \text{conv} \)

**Errors**

| 47290 | ?0 is not a Z existential quantification |
| 41040 | ?0 is not of the form: \( \exists D \mid P_1 \bullet P_2 \) where at least one of \( P_1 \) or \( P_2 \) are unbound by \( D \) |
There is no natural text to extract from the image.
8.2. Reasoning about Predicates

SML

\[ \text{val } z_\exists 1 \text{._conv : CONV;} \]

Description Converts a Z unique existential quantification to a Z existential quantification.

Conversion

\[
\begin{align*}
\Gamma \vdash (\exists \; D \mid P_1 \bullet P_2) & \iff \\
& (\exists \; D \mid P_1 \bullet P_2 \land \\
& (\forall \; D' \mid P_1' \land P_2' \bullet \\
& \text{"characteristic tuples} \\
& \text{component-wise equal"})
\end{align*}
\]

where the \(P'_i\) are variants of the \(P_i\), to correspond to the difference between \(D\) and \(D'\).

Additional decoration may be introduced as necessary to avoid free variable names capture, while maintaining the same decoration on each component (variable, schema, etc) of the declaration.

Example

\[
\begin{align*}
\Gamma \vdash z_\exists 1 \text{._conv} \\
\vdash (\exists \; [x, y : X ; z : Y] \mid x = x' \bullet z = f \; x'')
\end{align*}
\]

See Also \(z_\exists 1 \text{._intro.conv}\)

Errors

41122 ?0 is not of the form: \(\exists \forall \exists \\mid P_1 \bullet P'_2 \bullet \)

SML

\[ \text{val } z_\exists 1 \text{._intro.conv : CONV;} \]

Description \(z_\exists 1 \text{._intro.conv}\) converts an arbitrary simple HOL unique existentially quantified term into the corresponding Z, returning a theorem whose RHS should be in the Z language.

It can only reason about a single bound variable.

Conversion

\[
\begin{align*}
\Gamma \vdash \forall \exists \; x \bullet P[x] \leftrightarrow \\
(\exists \; x : U \mid \text{true} \bullet P[x])
\end{align*}
\]

This conversion cannot introduce a Z unique existential quantification with an empty declaration.

See Also \(z_\forall 1 \text{._intro.conv}\)

Errors

41048 ?0 is not of the form: \(\forall \exists \mid P_1 \bullet P'_2 \bullet \)

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
Chapter 8. SUPPORT FOR Z

\[ \text{val } z_{\exists_1\text{ tac}} : \text{TERM} \rightarrow \text{TACTIC}; \]

**Description**  Provide a witness for a unique existential.

Tactic  
\[ \frac{\{ \Gamma \} \exists_1 D[x_1,...] \mid P[x_1,...] \rightarrow V[x_1,...]}{\exists_1 \text{ tac } z \equiv t_1,...} \]

\[ z_{\exists_1\text{ tac }} \]

---

\[ \begin{array}{l}
\text{\{ \Gamma \} } \exists_1 D[\dot{x}_1,...] \mid P[\dot{x}_1,...] \rightarrow V[\dot{x}_1,...] \\
\{ \Gamma \} \exists_1 D[t_1,...] \rightarrow \text{a predicate} \land \\
(\forall D' \mid P[x'_1,...] \land V[x'_1,...] \rightarrow \text{characteristic tuples of } D' \\
\text{and witness component-wise equal})
\end{array} \]

**Definition**  
\[ \text{val } z_{\exists_1\text{ tac } \text{ wit}} = \text{conv_tac } z_{\exists_1\text{ conv } \text{ THEN } z_{\exists_1\text{ tac } \text{ wit}}}; \]

**Errors**  
\[ 41123 \text{ Conclusion of goal is not of the form: } z_{\exists_1 D} \mid P_1 \rightarrow P_2 \]
\[ 41021 \text{ ?0 cannot be interpreted as a binding that matches ?1} \]

---

\[ \text{val } \alpha_{\to z_{\text{ conv}}} : \text{CONV}; \]

**Description**  This function will return the equality theorem between a term and one that adjusts all sub-terms that fail to be Z because either:

- The subterm is a decl-style binding, and the type of the binding does not match the names of the variables bound. This is adjusted using the Z renaming construct.

- The subterm is a decl-style binding whose bound variables are not in the canonical ordering that would result from the Z mapping. This is adjusted by reorganising the order of abstractions and arguments.

If a HOL \( \alpha \)-conversion will suffice then that will be used instead.

Subterms that are not covered by these two cases will be traversed and their own subterms checked, regardless of their language.

**NOT YET IMPLEMENTED.**

**See Also**  \( \alpha_{\to z} \)

**Errors**  
\[ 41100 \text{ No adjustment took place} \]
This function will adjust all sub-terms that fail to be Z because either:

- The subterm involves a decl-style binding, and the type of the binding does not match the names of the variables bound. This is adjusted using the Z renaming construct.

- The subterm is a decl-style binding whose bound variables are not in the canonical ordering that would result from the Z mapping. This is adjusted by reorganising the order of abstractions and arguments.

If a HOL α-conversion will suffice then that will be used instead.

Subterms that are not covered by these two cases will be traversed and their own subterms checked, regardless of their language.

NOT YET IMPLEMENTED.

See Also  α_to_z_conv

Errors  

41100  No adjustment took place
8.3 Reasoning about Expressions

```
signature ZExpressions = sig
```

**Description** This provides the rules of inference, conversions and theorems for Z language set theory, tuples and cartesian products in the Z proof support system.

```
(* Proof Context: 'z∈_set_lang *)
```

**Description** A component proof context for handling the membership of expressions created by Z language set operations. It also provides some simplifications.

Set expressions treated by this proof context are constructs formed from:

- `set displays`, `set comprehensions`, `P`, `λ`, `µ`, `application`,
- `sequence displays`

If there was proof context material for string literals, or bag displays, it would perhaps go here.

**Contents**

**Rewriting:**

- `z∈_seta_conv1`, `z∈_setd_conv1`, `z∈_λ_conv`, `z∈_⟨⟩_conv`,
- `z_β_conv if its resulting theorem has no assumptions`.

**Stripping theorems:**

- `z∈_seta_conv1`, `z∈_setd_conv1`, `z∈_λ_conv`, `z∈_⟨⟩_conv`,
- plus these all pushed in through `¬`,
- and `z_β_conv, ∈_C z_β_conv if the resulting theorem has no assumptions`.

**Stripping conclusions:**

- `z∈_seta_conv1`, `z∈_setd_conv1`, `z∈_λ_conv`, `z∈_⟨⟩_conv`,
- plus these all pushed in through `¬`,
- and `z_β_conv, ∈_C z_β_conv if the resulting theorem has no assumptions`.

**Rewriting canonicalisation:**

Automatic proof procedures are respectively `z_basic_prove_tac`, `z_basic_prove_conv`, and no existence prover.

**Usage Notes** It requires theory `z_sets`. It is intended to be used with proof context “`z_predicates`”. It is not intended to be mixed with HOL proof contexts.

**See Also** `z_sets_ext`
8.3. Reasoning about Expressions

SML

(* Proof Context: 'z_sets_ext_lang *)

Description  An aggressive component proof context for handling the manipulation of Z sets by breaking them into predicate calculus, within the Z language. It is intended to always be used in conjunction with "'z_set_lib".

Set expressions treated by this proof context are constructs formed from:

| set displays, set comprehensions, P, λ, μ, application, 
| equality of two set expressions, sequence displays |

Contents

Rewriting:

| z_sets_ext_conv, z ∈ P_conv, z_setd ∈ P_conv, 

Stripping theorems:

| z_sets_ext_conv, z ∈ P_conv, z_setd ∈ P_conv, 

| plus these all pushed in through ¬ |

Stripping conclusions:

| z_sets_ext_conv, z ∈ P_conv, z_setd ∈ P_conv, 

| plus these all pushed in through ¬ |

Rewriting canonicalisation:

Automatic proof procedures are respectively z_basic_prove_tac, z_basic_prove_conv, and no existence prover (2-tuples are handled in proof context "z_predicates").

Usage Notes  It requires theory z_sets. It is intended to always be used in conjunction with "'z_set_lang".

It is not intended to be mixed with HOL proof contexts.

See Also  'z ∈ set
A component proof context for handling the manipulation of Z tuples and cartesian products within the Z language.

Expressions and predicates treated by this proof context are constructs formed from:

\[ \text{(membership of) } \times, \text{ equations of tuple displays, } \]
\[ \text{selection from tuple displays} \]

**Contents**

**Rewriting:**

\[ z \in \times \text{conv}, \]
\[ z \text{tuple-lang_eq.conv, } z \text{sel-lang_conv} \]

**Stripping theorems:**

\[ z \in \times \text{conv}, \ z \text{tuple-lang_eq.conv, } \in C \ z \text{sel-lang_conv, } \]
\[ z \text{sel-lang_conv (for boolean components of tuples)} \]
\[ \text{plus these all pushed in through } \neg \]

**Stripping conclusions:**

\[ z \in \times \text{conv}, \ z \text{tuple-lang_eq.conv, } \in C \ z \text{sel-lang_conv, } \]
\[ z \text{sel-lang_conv (for boolean components of tuples)} \]
\[ \text{plus these all pushed in through } \neg \]

**Stripping also contains the above in negated forms.**

**Rewriting canonicalisation:**

Automatic proof procedures are respectively \( z \text{basic_prove_tac, } z \text{basic_prove_conv} \), and no existence prover (2-tuples are handled in proof context “\( z \text{predicates} \)).

**Usage Notes**

It requires theory \( z \text{sets} \). It is intended to be used with proof context “\( z \text{predicates} \).” It should not be used with “\( z \text{tuples_lang} \).” It is not intended to be mixed with HOL proof contexts.
8.3. Reasoning about Expressions

SML

(* Proof Context: 'z_bindings *)

Description A component proof context for handling the manipulation of Z bindings.

Expressions and predicates treated by this proof context are constructs formed from:

equations of binding displays,
selection from binding displays

Contents

Rewriting:

|\(z\_binding\_eq\_conv2, z\_sel\_s\_conv\)

Stripping theorems:

|\(z\_binding\_eq\_conv2, \in\_C z\_sel\_s\_conv,\)
|\(z\_sel\_s\_conv\) (where component of binding is boolean).
|plus this pushed in through \(\neg\)

Stripping conclusions:

|\(z\_binding\_eq\_conv2, \in\_C z\_sel\_s\_conv,\)
|\(z\_sel\_s\_conv\) (where component of binding is boolean).
|plus this pushed in through \(\neg\)

Rewriting canonicalisation:

Automatic proof procedures are respectively \(z\_basic\_prove\_tac, z\_basic\_prove\_conv\), and no existence prover.

Usage Notes It requires theory \(z\_language\_ps\). It is intended to be used with proof context “z_predicates”. It is not intended to be mixed with HOL proof contexts.

SML

val z_sets_ext_thm : THM;
val z\_\in\_P\_thm1 : THM;
val z_app_thm : THM;
val z_app_eq_thm : THM;
val z\_\in\_app\_thm : THM;

Description The ML bindings of the theorems (other than consistency ones) in theory \(z\_language\_ps\).

SML

val z_app_conv : CONV;

Description A function to convert a Z application into the corresponding \(\mu\) expression (i.e. definite description).

Conversion

\[ \vdash f a = (\mu f\_a :\UU \mid (a\_f\_a) \in f \bullet f\_a) \]
\[ z\_app\_conv \]
\[ (\mu f\_a :\UU \mid (a\_f\_a) \in f \bullet f\_a) \]

Remains within the Z language though this is not checked.

See Also \(z\_app\_thm, z\_app\_eq\_tac\)

Errors

47190 ?0 is not a Z function application

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
val z_app_eq_tac : TACTIC;

Description Reduces a subgoal that states a Z application is equal to something to sufficient conditions for this to be provable. The conditions are not “necessary” only because they ignore the fact that in ProofPower-Z every predicate or expression is equal to itself, and other vacuous variants of this.

Tactic \( \{ \Gamma \} f \ a = v \)

\( \{ \Gamma \} (\forall f \ a : \ U \ | \ (a, f \ a) \in f \bullet f \ a = v) \land (a, v) \in f \)

z_app_eq_tac

If this does not match the pattern of the goal then

Tactic \( \{ \Gamma \} v = f \ a \)

\( \{ \Gamma \} (\forall f \ a : \ U \ | \ (a, f \ a) \in f \bullet f \ a = v) \land (a, v) \in f \)

z_app_eq_tac

will be tried instead. In addition an implicit “\( \iff \) true” will be used if the conclusion of the goal is an application.

See Also z_app_thm, z_app_conv

Errors 42002 Conclusion of goal is not of the form: \( \exists f \ a = v \), \( \exists v = f \ a \) or \( \exists f \ x \)

val z_app_\lambda_rule : TERM \(->\) THM;

Description Given a Z \( \beta \) redex this function will return a theorem stating sufficient conditions for this redex to be proven equal to some arbitrary value.

Rule \( \vdash \exists \forall x : U \bullet (\forall f \ a : \ U \ | \ (\exists D' | P' \bullet charD' = t \land V' = f \ a) \bullet f \ a = x) \land (\exists D' | P' \bullet (charD' = t \land V' = x) \Rightarrow (\lambda D | P \bullet V) t = x \)

Some renaming of bound variables may occur, thus the priming of \( D \), etc.

Errors 42008 \( \exists \) is not of the form: \( \exists (\lambda D | P \bullet V) t \)

val z_bindingd_elim_conv : CONV

Description Given a a binding display, that binds labels to the selection of that label to a single value, return that single value.

Conversion \( \vdash (x_1 = b \cdot x_1, \ldots) = b \)

z_bindingd_elim_conv

Errors 42018 \( \exists \) is not of the form: \( \exists (x_1 = b \cdot x_1, \ldots, x_N = b \cdot x_N) \) where \( N \geq 1 \)
### z-bindingd_intro_conv

**Description**  
Given a value with a binding type, prove it equal to a binding display.

<table>
<thead>
<tr>
<th>Conversion</th>
<th>( \vdash b = (x_1 \equiv b.x_1, \ldots) )</th>
<th>( \text{z-bindingd_intro_conv} )</th>
</tr>
</thead>
</table>

**Errors**  
42017  
?0 does not have a binding type

### z-binding_eq_conv

**Description**  
A conversion for eliminating equations of bindings.

<table>
<thead>
<tr>
<th>Conversion</th>
<th>( \vdash (b_1 = b_2) \Leftrightarrow (b_1.s_1 = b_2.s_1) \land \ldots )</th>
<th>( \text{z-binding_eq_conv} )</th>
</tr>
</thead>
</table>

where \( b_1 \) (and thus \( b_2 \)) has a binding type equal to the type of something of the form \( \mathcal{E}(s_1 \equiv \ldots, s_2 \equiv \ldots) \).

\( \text{z-binding_eq_conv1} \) first applies conversion \( \text{z-binding_eq_conv} \), and then, if either or both of \( b_1 \) and \( b_2 \) are binding constructions it eliminates the projection functions, in a manner similar to \( \text{z-sel_s_conv} \).

\( \text{z-binding_eq_conv2} \) requires both sides to be binding displays or have the empty schema type:

<table>
<thead>
<tr>
<th>Conversion</th>
<th>( \vdash ((l_1 \equiv x_1, \ldots) = (l_1 \equiv y_1, \ldots)) \Leftrightarrow )</th>
<th>( \text{z-binding_eq_conv2} )</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Conversion</th>
<th>( \vdash ((b_1 \oplus [])) = b_2 ) \Leftrightarrow ( \text{true} )</th>
<th>( \text{z-binding_eq_conv2} )</th>
</tr>
</thead>
</table>

**See Also**  
\( \text{z-sel_s_conv}, \text{z-binding_eq_conv3} \)

**Errors**  
42013  
?0 is not of the form \( \mathcal{E}\text{binding} = \text{binding} \)

42021  
?0 is not of the form \( \mathcal{E}b_1 = b_2 \) where \( b_i \) has the form \( \mathcal{E}(x_1 \equiv \ldots, \ldots) \) or has the empty schema
### z_defn_simp_rule

**Description**  
This rule is a method of processing a standard style of specification into a simple rewriting theorem.

\[
\begin{align*}
\Gamma &\vdash x \in (\mathbb{P} y) \land (\forall z: y \cdot z \in x \Leftrightarrow f[z]) \\
\Gamma &\vdash \forall z: y \cdot z \in x \Leftrightarrow z \in y \land f[z]
\end{align*}
\]

The rule will also attempt to preprocess its input with \textit{z_para_pred_conv}. This is on the basis that theorems that are of an appropriate form for this rule are often derived from a Z definition, and this pre-processing is all the processing required to convert the definition to acceptable input. The rule can also handle generic parameters to the theorem.

**Errors**  
42011  ?0 cannot be converted to the form:  
\[
\Gamma \vdash x \in (\mathbb{P} y) \land (\forall z: y \cdot z \in x \Leftrightarrow f[z])
\]

### z_let_conv1

**Description**  
This conversion replaces a let-expression by an equivalent \(\mu\)-expression.

\[
\begin{align*}
\Gamma &\vdash (let \ v1 \hat{=} t1; \ldots \bullet b) = (\mu \ v1 : \{t1\}; \ldots \bullet b) \quad z\_let\_conv1 \\
\Gamma &\vdash \mu \ v1 \hat{=} t1; \ldots \bullet b
\end{align*}
\]

This is mainly intended for use in programming proof procedures. \textit{z_let_conv} may be used simply to expand let-expressions.

**See Also**  
\textit{z_let_conv}

**Errors**  
47211  ?0 is not a Z let term

### z_let_conv

**Description**  
This conversion expands the local definitions in a let-expression.

\[
\begin{align*}
\Gamma &\vdash (let \ v1 \hat{=} t1; \ldots \bullet b) = b[t1/v1, \ldots] \\
\Gamma &\vdash \mu \ v1 \hat{=} t1; \ldots \bullet b
\end{align*}
\]

The conversion will fail with message 42029 given a let-expression such as \[
\mu \ let \ x \hat{=} 42; \ y \hat{=} 99; \ x \hat{=} 43 \bullet x + y
\]
that contains incompatible local definitions.

**See Also**  
\textit{z_let_conv1}

**Errors**  
47211  ?0 is not a Z let term

42029  The local definitions in the let-expression ?0 cannot be expanded

### Z_RANK_C, Z_RANKS_C, Z_LEFT_C, Z_RIGHT_C

**Description**  
\textit{Z_RANK_C} (resp. \textit{Z_RANKS_C}, \textit{Z_LEFT_C}, \textit{Z_RIGHT_C}) applies a conversion to the operand (resp. operands, left operand, right operand) of a Z function application.
8.3. Reasoning about Expressions

```sml
val z_sel_s_conv : CONV;
```

**Description** A conversion for selecting a component from a binding.

Conversion

| ⊢ (n_1 ≡ t_1, ..., n_c = t_c) | z_sel_s_conv | \(n_1 ≡ t_1, ..., n_c = t_c\) |

**See Also** z_binding_eq_conv

Errors

| 42014 | ?0 is not of the form: \(n_1 ≡ t_1, ..., n_c = t_c\) |

```sml
val z_sel_intro_conv : CONV;
```

**Description** This conversion carries out the introduction of a tuple display of tuple selections from the same tuple.

Conversion

| ⊢ t = (t_1, ..., t_n) | z_sel_intro_conv | \(t_1, ..., t_n\) |

Errors

| 42004 | ?0 does not have a Z tuple type |

```sml
val z_sel_lang_conv : CONV;
```

**Description** This conversion carries out the selection from a tuple display.

Conversion

| ⊢ (t_1, ..., t_i, ..., t_n) | z_sel_lang_conv | \(t_1, ..., t_i, ..., t_n\) |

Errors

| 47185 | ?0 is not a Z tuple selection |
| 42006 | ?0 is not of the form \(x_i\) |

```sml
val z_setd_∈_P_conv : CONV
```

**Description** Expand out expressions that state that a set display is a member of a power set.

Conversion

\[ \forall \{x_1, ..., \} \in P X \iff (x_1 \in X \wedge ...) \]

z_setd_∈_P_conv

\[ \forall \{x_1, ..., \} \in P X \]

and

Conversion

\[ \forall \{} \in P X \iff true \]

z_setd_∈_P_conv

\[ \forall \{} \in P X \]

The conversion will all simplify certain subterms involving true or terms of the form \(x = x\).

**See Also** z_setd_≤_conv

Errors

| 42019 | ?0 is not of the form: \(\forall \{x_1, ..., \} \in P X\) |

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
val z_sets_ext_conv : CONV;

Description  Use the extensionality of sets in combination with knowledge about tuples. Given as input an equality of the form \( v = w \) then:

If \( v \) is of type \( \text{ty SET} \) where \( \text{ty} \) is not a tuple type:

\[
\vdash (v = w) \iff \left( \forall \, x_n : \mathbb{U} \bullet x_n \in v \iff x_n \in w \right)
\]

where \( x_n \) is the first variable in the list \( x_1, x_2, \ldots \) that doesn’t appear in \( v \) or \( w \) (free or bound).

If \( w \) is of type \( \text{ty SET} \) where \( \text{ty} \) is an \( n \)-tuple type, or binding type, then sufficient \( x_i \) will be introduced, instead of just \( x_n \), to allow \( x_n \) to be replaced by a construct of bindings and tuples of the \( x_i \), such that no \( x_i \) has a binding or tuple type and appears exactly once in the construct.

Example

\[
\vdash z\text{\_sets\_ext\_conv} \, (r \times [a, b : X] \times x_2) = d \iff (\forall \, x_1 : \mathbb{U}; x_3 : \mathbb{U}; x_4 : \mathbb{U} \bullet (x_1, x_3, x_4) \in r \times [a, b : X] \times x_2 \iff (x_1, x_3, x_4) \in d)
\]

Notice how the introduced universal quantification “skips” \( x_2 \) which is present in the input term.

See Also  \( z\_sets\_ext\_thm \)

Errors  

\[42010\] ?0 is not of the form: \( v = w \) where \( v \) has a set type

val z_string_conv : CONV;
val z\_\in\_string_conv : CONV;

Description  \( z\_string\_conv \) changes a Z string into a Z sequence of HOL characters. It thus does not remain in Z.

\[
\vdash "abc..." = (\langle \langle \langle 'a', 'b', 'c', \ldots \rangle, \ldots \rangle \rangle)
\]

Definition

\[
\text{val } z\_\in\_string\_conv = \in\_C \, z\_string\_conv;
\]

See Also  \( \text{char\_eq\_conv} \) for the equality of HOL characters, \( z\_string\_eq\_conv \) for the equality of Z strings.

Errors  

\[42015\] ?0 is not of the form: \( \"abc...\" \)
SML

val z_tuple_lang_eq_conv : CONV;

Description A conversion for eliminating tuples over equality.

Conversion

\[\Gamma \vdash (t_1, t_2, \ldots) = (u_1, u_2, \ldots) \iff z_{\text{tuple-lang\_eq\_conv}} \hat{\xi}(t_1, t_2, \ldots) = (u_1, u_2, \ldots)\]

Errors

42003 \ ?0 is not of the form: \(\hat{\xi}(x_1, \ldots) = (y_1, \ldots)\)

SML

val z_tuple_lang_intro_conv : CONV;

Description This conversion carries out the elimination of a tuple display of tuple selections from the same tuple.

Conversion

\[\Gamma \vdash (t.1, \ldots, t.n) = t \iff z_{\text{tuple-lang\_intro\_conv}} \hat{\xi}(t.1, \ldots, t.n)\]

where \(n\) is the arity of \(t\).

Errors

42005 \ ?0 is not of the form: \(\hat{\xi}(t.1, \ldots, t.n)\)

SML

val Z_\in_ELIM_C : CONV \rightarrow CONV;

Description \(Z_\in_ELIM_C\) \(cnv\) \(tm\) takes a conversion \(cnv\) that can be applied to set memberships, and a set term \(tm\). The conversion is then modified to make it applicable to the term. The resulting conversion will check to see if its term argument, \(tm\) is a set. If so it will form the term: \(\hat{\xi}x_1 \in \{x | f[x]\}\) (where \(x_1\) is the first variable in \(x_1, x_2, \ldots\) not present in \(tm\)), apply \(cnv\) to the result, gaining some equation:

\[\Gamma \vdash x_1 \in \{x | f[x]\} \iff f[x_1]\]

and then return the theorem

\[\Gamma \vdash \{x_1 : \forall x \in \{x | f[x]\}\}\]

Errors

42027 \ ?0 is not a Z set
42026 unable to convert \?0 to the form: \(\hat{\xi}x \in \{x:U|s\}\)

And as conversion argument upon the membership term, with the error being passed through by the conversional untouched.
SML

\begin{verbatim}
| val z ∈ seta_conv : CONV;
| val z ∈ seta_conv1 : CONV;
\end{verbatim}

**Description**  
A conversion of membership of a Z set abstraction into a Z existential quantification. Bound variables in the existential quantification are renamed as necessary.

\[
\begin{align*}
\vdash (t ∈ \{ D | P \cdot T \}) & \iff (\exists D' | P' \cdot T' = t) \\
\end{align*}
\]

In the case of \(z ∈ seta_conv1\), if \(T\) is a tuple or simple variable then the conversion will attempt to eliminate the existential quantification via the methods of basic_prove_∃.conv. In particular, this attempt should succeed if \(T\) is the characteristic tuple of \(D\).

No simplification is attempted by \(z ∈ seta_conv\)

Renaming of bound variables may be necessary, thus the priming in the RHS.

**Errors**

42001  ?0 is not of the form: \(\exists t \in \text{seta}\) where seta is a set abstraction

---

SML

\begin{verbatim}
| val z ∈ setd_conv1 : CONV;
\end{verbatim}

**Description**  
A conversion proving membership of a Z set display where the member is syntactically identical (up to α-conversion) to a member of the set.

\[
\begin{align*}
\vdash t ∈ \{ ..., t, ... \} & \iff \exists t ∈ \{ ..., t, ... \}
\end{align*}
\]

**See Also**  
\(z ∈ setd_conv\)

**Errors**

42009  ?0 is not of the form: \(\exists t ∈ \{ ..., t, ... \}\)

---

SML

\begin{verbatim}
| val z ∈ ×_conv : CONV;
\end{verbatim}

**Description**  
A conversion for the membership of cartesian products.

\[
\begin{align*}
\vdash t ∈ (T_1 × T_2 × ...) & \iff \exists t ∈ (T_1 × T_2 × ...)
\end{align*}
\]

\(z \cdot sel_1\_conv, \text{q.v., will be attempted on each of the tuple selections.}\)

**See Also**  
\(z \cdot ×\_conv\)

**Errors**

42007  ?0 is not of the form: \(\exists t ∈ (T_1 × T_2 × ...)\)
8.3. Reasoning about Expressions

SML

\[ \text{val } z \in P \text{ conv : CONV}; \]

**Description**  Use \( z \in P \text{ thm1} \) in combination with knowledge about tuples. Given as input a term of the form \( v \in P w \) then:

If \( w \) is of type \( ty \ SET \) where \( ty \) is not a tuple type:

\[ \vdash (v \in P) \iff (\forall xn : U \cdot xn \in v \Rightarrow xn \in w) \]

where \( xn \) is the first variable in the list \( x_1, x_2, \ldots \) that doesn’t appear in \( v \) or \( w \) (free or bound).

If \( w \) is of type \( ty \ SET \) where \( ty \) is an \( n \)-tuple type, or binding type, then sufficient \( x_i \) will be introduced, instead of just \( xn \), to allow \( xn \) to be replaced by a construct of bindings and tuples of the \( x_i \), such that no \( x_i \) has a binding or tuple type and appears exactly once in the construct.

**Example**

\[ z \in P \text{ conv } \langle p \rangle \in P (r \times [a, b : X] \times x_2) = \]

\[ \vdash p \in P (r \times [a, b : X] \times x_2) \]

\[ \iff (\forall x_1 : U; x_3 : U; x_4 : U; x_5 : U)
   \cdot (x_1, (a \equiv x_3, b \equiv x_4), x_5) \in p
   \Rightarrow (x_1, (a \equiv x_3, b \equiv x_4), x_5) \in r \times [a, b : X] \times x_2) \]

Notice how the introduced universal quantification “skips” \( x_2 \) which is present in the input term.

**See Also**  \( z \in P \text{ thm1} \), \( z \in P \text{ thm} \), \( z \subseteq \text{ conv} \)

**Errors** 42016 \( ?0 \) is not of the form \( \forall v \in P w \)

---

SML

\[ \text{val } z \{ \} \text{ conv : CONV}; \]

\[ \text{val } z \in \{ \} \text{ conv : CONV}; \]

**Description**  Convert a sequence display into a set display.

\[ \vdash \langle x_1, \ldots, x_n \rangle = \left\{ (1, x_1), \ldots, (n, x_n) \right\} \]

**Definition**

\[ \text{val } z \in \{ \} \text{ conv } = \in C z \{ \} \text{ conv}; \]

**Errors** 42025 \( ?0 \) is not of the form \( \forall \left\{ \ldots \right\} \)
Chapter 8. SUPPORT FOR Z

val \texttt{z\_\times\_conv} : \texttt{CONV};

**Description** A conversion for eliminating cartesian products.

\[
\vdash (T_1 \times T_2 \times ... = \{t_1:T_1 ; t_2:T_2 ; ...: (t_1, t_2,...)\}
\]

The \(t_i\) used are distinct from any variable names in the \(T_i\).

**See Also** \texttt{z\_\in\_\times\_conv}, which is a faster function, if appropriate.

**Errors** \ref{110727} ?0 is not a Z cartesian product

\[
\texttt{val z\_\beta\_conv} : \texttt{CONV};
\]

**Description** A conversion for a simple Z \(\beta\) redex. The \(\lambda\)-term of the redex must have only a single variable in its declaration part.

\[
\vdash (\lambda x : X | P[x]\bullet V[x]) t = V'[t]
\]

The assumptions will be eliminated if trivial (i.e. if the first assumption can be proven true by \texttt{z\_\in\_u\_conv}, the second if the assumption is just \(\texttt{Z}\texttt{true}\)). Some renaming of bound variables may occur, thus the priming of \(V\).

**Errors** \ref{110727} ?0 is not of the form \(\texttt{Z(\lambda x : X | P\bullet V}) t\)

\[
\texttt{val z\_\lambda\_conv} : \texttt{CONV};
\]

**Description** Convert a Z \(\lambda\) abstraction into a set abstraction.

\[
\vdash (\lambda D | P \bullet V) = \{ D | P \bullet (\text{char}D, V)\}
\]

Where \(\text{char}D\) is the characteristic tuple of \(D\).

**Definition** \texttt{val z\_\in\_\lambda\_conv} = \texttt{\in\_C z\_\lambda\_conv};

**See Also** \texttt{z\_app\_\lambda\_rule}, \texttt{z\_\beta\_tac}

**Errors** \ref{110727} ?0 is not a Z \(\lambda\) abstraction
8.3. Reasoning about Expressions

SML

\[ \text{val \( \text{z\_mu\_rule} : \text{TERM} \rightarrow \text{THM}; \)} \]

**Description**  This rule is given a Z \( \mu \) expression (i.e. a Z definite description), and returns a theorem that states what is required for this \( \mu \) expression to be equal to some value, \( x \). The requirement is that if any value satisfies the schema text of the \( \mu \) expression then it must equal \( x \), and that \( x \) satisfies the schema text of the \( \mu \) expression.

\[
\frac{\vdash \forall x : U \bullet 
(\forall D' \mid P' \bullet V' = x) \land 
(\exists D' \mid P' \bullet V' = x) \Rightarrow 
(\mu D \mid P \bullet V) = x}{\text{z\_mu\_rule}}
\]

The result may require bound variable renaming and thus the priming of \( D \), etc.

Errors

\[ \text{47210 ?}\theta \text{ is not a Z } \mu \text{ term} \]

SML

\[ \text{val \( \text{\_C \_\_conv} : \text{CONV} \rightarrow \text{CONV}; \)} \]

**Description**  \( \_\_C \_\_conv tm \) takes a conversion \( \_\_conv \), that applies to set terms, will check to see if its term argument, \( tm \) is a membership statement. If so, it will apply its conversion to the set. If not it will fail. It does not check that its result remaining in Z (and indeed is applicable to HOL membership terms as well).

**See Also**  \( Z\_\_\_\_\_ELIM\_\_C \)

Errors

\[ \text{42028 ?}\theta \text{ is not of the form } \exists v \in s \land \forall v \in s \]

And as conversion argument upon the set, with the error being passed through by the conversional untouched.
8.4 Reasoning about Schema Expressions

SML

|signature ZSchemaCalculus = sig |

Description This provides the rules of inference for schema calculus in the Z proof support system. The material is implemented within the theory z_language_ps.

SML

|(* Proof Context: 'z_schemas *) |

Description A component proof context for handling the manipulation of Z schemas. It “understands” the membership, or schema as predicate, properties of each of the schema calculus operators. It will replace an appropriate $\exists v \in S^\downarrow$ by a “schema $S$ as predicate”.

Predicates and expressions treated by this proof context are constructs formed from:

- (selection from) horizontal schemas, schemas as predicates, (selection from) $\theta$ expressions,
- $\neg s$, $\land s$, $\lor s$, $\Rightarrow s$, $\Leftrightarrow s$, $\exists_1 s$, $\forall s$, $\exists s$, $\pre s$, $\pres s$, $\hide s$, $\Delta s$, $\Z s$, $\O s$, rename_s,

Contents

Rewriting:

$$(\text{RANDOM}_C z_{\theta\_conv} \text{ THEN } C z_{\sel s\_conv})$$

- which simplifies terms of the form: $\exists (\theta s).\nm^\downarrow$
  - $z_{\theta\_eq\_conv}$, $z_{\theta\_conv1}$,
  - $z_{\in}\neg s\_conv$, $z_{\in}\land s\_conv$, $z_{\in}\lor s\_conv$,
  - $z_{\exists s}\Rightarrow s\_conv$, $z_{\exists s}\Leftrightarrow s\_conv$, $z_{\exists s}\exists_1 s\_conv$,
  - $z_{\exists_1 s}\exists_1 s\_conv$, $z_{\exists s}\h\_schema\_conv$, $z_{\exists s}\pre s\_conv$, $z_{\exists s}\pres s\_conv$,
  - $z_{\exists s}\hide s\_conv$, $z_{\exists s}\Delta s\_conv$, $z_{\exists s}\Z s\_conv$,
  - $z_{\exists s}\O s\_conv$, $z_{\exists s}\rename s\_conv$, $z_{\schema\_pred\_intro\_conv}$

Stripping theorems and conclusions:

$$(\text{RANDOM}_C z_{\theta\_conv} \text{ THEN } C z_{\sel s\_conv})$$

- which simplifies boolean terms of the form: $\exists (\theta s).\nm^\downarrow$
  - $\in C (\text{RANDOM}_C z_{\theta\_conv} \text{ THEN } C z_{\sel s\_conv})$
  - which simplifies terms of the form: $\exists x \in (\theta s).\nm^\downarrow$
  - $z_{\theta\_eq\_conv}$, $z_{\theta\_conv1}$,
  - $z_{\exists s}\neg s\_conv$, $z_{\exists s}\land s\_conv$, $z_{\exists s}\lor s\_conv$,
  - $z_{\exists s}\Rightarrow s\_conv$, $z_{\exists s}\Leftrightarrow s\_conv$, $z_{\exists s}\exists_1 s\_conv$,
  - $z_{\exists_1 s}\exists_1 s\_conv$, $z_{\exists s}\h\_schema\_conv$, $z_{\exists s}\pre s\_conv$, $z_{\exists s}\pres s\_conv$,
  - $z_{\exists s}\hide s\_conv$, $z_{\exists s}\Delta s\_conv$, $z_{\exists s}\Z s\_conv$,
  - $z_{\exists s}\O s\_conv$, $z_{\exists s}\rename s\_conv$, $z_{\schema\_pred\_intro\_conv}$

plus these all pushed in through $\neg$

Rewriting canonicalisation:

Automatic proof procedures are respectively $z\_basic\_prove\_tac$, $z\_basic\_prove\_conv$, and no existence prover.

Usage Notes It requires theory z_language_ps. It is intended to be used with proof context “z_bindings”. It is not intended to be mixed with HOL proof contexts.
8.4. Reasoning about Schema Expressions

SML

val z_decor_s_conv : CONV;
val z ∈_decor_s_conv : CONV;

Description  A conversion which expands a statement of membership to a decorated schema.

Conversion

\[ \vdash v \in (S)' \iff \exists v \in (S)' \]

where the type of \( S \) is

\[ \mathbb{P} \{ [x_1:U; ...] \} \]

\( S \) may be a schema-reference, or (in extended Z) anything of the stated type. Schemas as predicates will be treated as membership statements by this conversion. If \( v \) is a binding display then \( v.x_i \) will be simplified.

Definition

\[ \text{val } z_{\text{decor}_s_{\text{conv}}} = Z_{E_{\text{LIM}_C}} z_{\in_{\text{decor}_s_{\text{conv}}}} \]

Errors

43015 ?0 not of the form: \( \exists v \in ds^\gamma \) where \( ds \) is a decorated schema expression

SML

val z_dec_rename_s_conv : CONV;

Description  This conversion turns an ill-formed schema-as-declaration into a well-formed one using renaming. The ill-formed schemas-as-declarations in question are those of the form

\[ \vdash Z'SchemaDec bind schema^\gamma \]

where \( bind \) is not equal to \( \exists \theta schema^\gamma \).

Conversion

\[ \vdash Z'SchemaDec bind schema \iff schema[y_1/x_1, ..., y_k/x_k] \]

Uses  In correcting the results of functions which produce results outside \( Z \) because of substitution within variable binding constructs.

Errors

43060 ?0 is not an ill-formed schema-as-declaration
val \( z \_ \text{hide}\_\text{s-conv} : \text{CONV} \);
val \( z \in \_{\text{hide-s-conv}} : \text{CONV} \);

**Description** A conversion concerning the schema hiding.

**Conversion**

\[
\vdash S \backslash s (x_1, \ldots) = \bigset{y_1 : \mathcal{U} ; \ldots \mid \exists x_1 : \mathcal{U} ; \ldots \bullet S}^{z \_ \text{hide-s-conv}}
\]

where \( S \) is a schema that has signature variables \( x_1, x_2, \ldots \) and \( y_1, y_2, \ldots \).

**Definition**

\[
\vdash z \in \_{\text{hide-s-conv}} = \in \_{C \ z \_ \text{hide-s-conv}}
\]

Schemas as predicates will be treated as membership statements by this conversion.

**Errors**

43018 \( ?0 \) is not of the form \( \forall S \backslash s (x_1, \ldots)^{z \_ \text{hide-s-conv}} \) where \( S \) is a schema.

val \( z \_ \text{h-schema-conv} : \text{CONV} \);

**Description** A conversion from a horizontal schema to a set comprehension.

**Conversion**

\[
\vdash \{D \mid P\} = \{D \_P \bullet \theta\}_D^{z \_ \text{h-schema-conv}}
\]

See Also \( z \in \_ \text{h-schema-conv} \) and \( z \in \_ \text{h-schema-conv} \), which are more appropriate if the schema expression occurs as a subterm of a membership expression.

**Errors**

43004 \( ?0 \) is not a horizontal schema.

val \( z \_ \text{h-schema-pred-conv} : \text{CONV} \);

**Description** A conversion for eliminating a horizontal schema as a predicate.

**Conversion**

\[
\vdash \mid D \_P \mid \iff "D as Predicate" \_ P \ implies \ P:\_D \_P \bullet \theta D^{z \_ \text{h-schema-pred-conv}}
\]

Projections from bindings, which are likely to be introduced, are automatically expanded out. The user may do so with, e.g.,

\[
\text{MAP}_C \ z \_ \text{sel-s-conv}
\]

The horizontal schema may be decorated.

See Also \( z \_ \text{schema-pred-conv} \) for a more general conversion.

**Errors**

43012 \( ?0 \) is not a horizontal schema as a predicate.
8.4. Reasoning about Schema Expressions

**SML**
\[
\text{val } z\_\text{norm}_\text{h}\_\text{schema}\_\text{conv} : \text{CONV};;
\]

**Description** A conversion for normalising horizontal schemas.

\[
\frac{\vdash[D|P] = [DU|D1 \land P]}{z\_\text{norm}_\text{h}\_\text{schema}\_\text{conv}} \quad \lceil[D|P]\rfloor
\]

D1 is the implicit predicate formed from D by \(z\_\text{decl}_\text{pred}\_\text{conv}\), and then simplified. The simplification is that conjuncts of the predicate that are provable by \(z\_\in\_u\_\text{conv}\), q.v., are proven and then eliminated from D1. DU is the signature formed from the variables bound by D, all of type U.

**Example**
\[
z\_\text{norm}_\text{h}\_\text{schema}\_\text{conv} \lceil[w:W; x,y:X; z:U | p w x y z]\rfloor
\]
\[
= \lceil[w:U; x:U; y:U; z:U | (w \in W \land x \in X \land y \in X) \land p w x y z]\rfloor
\]

Notice how, since \(z \in U\) can be proven by \(z\_\in\_u\_\text{conv}\), it is not included in D1.

**Errors**
\[43004 \ ?0 \text{ is not a horizontal schema}\]

**SML**
\[
\text{val } z\_\text{pre}_\text{s}\_\text{conv} : \text{CONV};;
\]
\[
\text{val } z\_\in\_\text{pre}_\text{s}\_\text{conv} : \text{CONV};;
\]

**Description** Schema precondition elimination.

\[
\frac{\vdash \text{pre } S = [DI | (\exists DF \bullet S1)] }{z\_\text{pre}_\text{s}\_\text{conv}} \quad \lceil\text{pre } S\rfloor
\]

DI is the declaration formed from the unprimed and input ('?') variables of S, given type U. DF is the declaration formed from the primed and output ('!') variables of S, given type U. It is possible for one or both of DI and DF to be the empty declaration. S1 is the schema S as a predicate.

**Definition**
\[
\text{val } z\_\in\_\text{pre}_\text{s}\_\text{conv} = \in \_ C z\_\text{pre}_\text{s}\_\text{conv}
\]

Schemas as predicates will be treated as membership statements by this conversion.

**Errors**
\[43007 \ ?0 \text{ is not a schema precondition}\]
Chapter 8. SUPPORT FOR Z

**val z_rename_s_conv : CONV;**

**val z ∈ rename_s_conv : CONV;**

**Description**  A conversion concerning schema renaming.

\[ \vdash v \in S[x_1/y_1, ...] \iff (y_1 \equiv v.x_1, ..., z_1 \equiv v.z_1, ...) \in S \]

where \( S \) has signature variables \( y_1, ... \) and \( z_1, ... \). Schemas as predicates will be treated as membership statements by this conversion. If \( v \) is a binding display then \( v.x_i \) will be simplified. The conversion will fail with error 43035 if applied to a renaming that renames one component to an already existent, unrenamed, component.

**Definition**  
\[ \text{val } z \in rename_s_conv = \text{Z} \in \text{ELIM } C \text{ z} \in rename_s_conv; \]

**Errors**  
43031 ?0 is not of the form: \( \exists S[x_1/y_1, ...] \) where \( S \) is a schema
43035 ?0 is of the form \( \exists S[...x_i/y_i,...] \) where \( x_i \) is already an unrenamed component of \( S \)

---

**val z_schema_pred_conv : CONV;**

**val z_θ ∈ schema_intro_conv : CONV;**

**Description**  \( z \_schema\_pred\_conv \) is a conversion from a schema as a predicate to the predicate asserting that its \( \theta \)-term is a member of the schema.

\[ \vdash S \iff \theta S \in S \]

\( S \) is any schema as a predicate, including both schema references and horizontal schemas.

\( z \_schema\_pred\_conv \) is an alias for \( z_\theta \in schema\_intro\_conv \).

**See Also**  \( z \_h \_schema\_pred\_conv \) for alternative, \( z \_\theta \_conv \), and \( z_\theta \in schema\_conv \).

**Errors**  
43014 ?0 is not a schema as a predicate

---

**val z_schema_pred_intro_conv : CONV;**

**Description**  This conversion attempts to convert a predicate that is a membership of a schema into a schema as a predicate.

\[ \vdash ((x_1 \equiv x_1, ...) \in S) \iff S \iff \exists (x_1 \equiv x_1, ...) \in S \]

The input term must have a binding display that binds to each label a variable with the label’s name (maintaining decoration).

**Errors**  
43032 ?0 cannot be converted to a schema as a predicate

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
8.4. Reasoning about Schema Expressions

**SML**

```sml
val z_strip_tac : TACTIC;
```

**Description**  
*z_strip_tac* is a general purpose tactic for simplifying away the outermost connective of a Z goal. It first attempts to apply *z_∀_tac*. If that fails it then tries to apply the current proof context’s conclusion stripping conversion, to rewrite the outermost connective in the goal. Failing that it tries to simplify the goal by applying an applicable member of the following collection of tactics (only one could possibly apply):

- `simple_∀_tac`, `∧_tac`, `⇒_T strip_asm_tac`, `t_tac`

Failing either being successful, it tries *concl_in_asm_tac* to prove the goal, and failing that, returns the error message below.

Finally, it will attempt to make the goal a “schema as predicate”, if possible, by using *z_schema_pred_intro_conv*.

Note how new assumptions generated by the tactic are processed using *strip_asm_tac*, which uses the current proof context’s theorem stripping conversion.

*z_strip_tac* may produce several new subgoals, or may prove the goal.

The tactic is defined as:

```sml
val z_strip_tac = (z_∀_tac ORELSE T strip_tac)
  THEN TRY T conv_tac z_schema_pred_intro_conv;
```

**Uses**  
This is the usual way of simplifying a goal involving Z predicate calculus connectives, and other functions “understood” by the current prof context.

**See Also**  
*STRIP_CONCL_T* and *STRIP_THM_THEN* which are used to implement this function. *taut_tac* for an alternative simplifier. *swap_∀_tac* to rearrange the conclusion for tailored stripping.

**Errors**  
28003 *There is no stripping technique for ?0 in the current proof context*

---

**SML**

```sml
val z_Δ̲_s_conv : CONV;
val z_∈_Δ̲_s_conv : CONV;
```

**Description**  
A conversion concerning the delta schemas.

**Conversion**  

\[
\vdash \Delta S = [S; S'] \quad z_\Delta_\in_conv
\]

**Definition**  

```sml
val z_∈_Δ̲_s_conv = ∈_C z_Δ̲_s_conv
```

Schemas as predicates will be treated as membership statements by this conversion.

**Errors**  
43022 *?0 is not of the form: \( \parallel \Delta S \parallel \) where S is a schema*
\textbf{Description} A conversion from a predicate asserting membership of a horizontal schema to an existential quantification.

\[
\begin{array}{c}
\vdash v \in [D|P] \iff \exists D'[P'\theta|D] = v \\
\vdash v \in [D|P]^\triangledown
\end{array}
\]

Bound variable renaming may be necessary, and thus the priming in the RHS of the result. Schemas as predicates will be treated as membership statements by this conversion.

\textbf{See Also} \texttt{z\_\_h\_schema\_conv} for a faster, if more verbose result from simplifying the same category of terms, \texttt{z\_\_h\_schema\_conv} for a horizontal schema term without and outer \texttt{\_\_h\_}. 

\textbf{Errors}

43003 \texttt{v is not of the form } \triangledown v \in [D|P]^\triangledown

43033 Unable to prove \texttt{v equal to something of the form } \exists D'[P'\theta|D] = v^\triangledown

use \texttt{z\_\_h\_schema\_conv} instead, and then work by hand

Error 43033 indicates that there is some sort of variable capture problem preventing the conversion from functioning correctly. As indicated, \texttt{z\_\_h\_schema\_conv} is a conversion that does apply to simplify the input term.

\textbf{Description} A conversion from a predicate asserting membership of a horizontal schema to a predicate.

\[
\begin{array}{c}
\vdash v \in [D|P] \iff D' \land P' \\
\vdash v \in [D|P]^\triangledown
\end{array}
\]

where, if \(D\) declares variables \(x_1, x_2, \ldots\), then \(D'\) is

\texttt{"predicate from } D[x_1 \setminus v.x_1, \ldots]"

as converted by \texttt{z\_\_decl\_pred\_conv}, and \(P'\) is

\texttt{[P|x_1 \setminus v.x_1, \ldots]}

The execution of the conversion may also involve bound variable renaming. If \(v\) is a binding display then \(v.x_1\) will be simplified. Though this conversion gives a rather verbose result, it evaluates faster than \texttt{z\_\_h\_schema\_conv1}, and is probably of more practical value in a proof. Schemas as predicates will be treated as membership statements by this conversion.

\textbf{See Also} \texttt{z\_\_h\_schema\_conv1}

\textbf{Errors}

43003 \texttt{v is not of the form } \triangledown v \in [D|P]^\triangledown
8.4. Reasoning about Schema Expressions

\begin{verbatim}
SML
val z_Ξ_s_conv : CONV;
val z_Ξ_s_conv : CONV;

Description  A conversion concerning Ξ schemas.

Conversion
\[ \vdash \Xi S = \left[ S; S' \mid \theta S = \theta S' \right] \]
\[ \Xi Z \Xi S \]

Definition
\[ \text{val } z_Ξ_s_conv = \Xi C z_Ξ_s_conv \]

Schemas as predicates will be treated as membership statements by this conversion.

Errors
43023  ?0 is not of the form: \[ Z \Xi S \] where S is a schema
\end{verbatim}

\begin{verbatim}
SML
val z_Ξ_s_conv : CONV;
val z_Ξ_s_conv : CONV;

Description  A conversion concerning the membership of a schema bi-implication.

Conversion
\[ \vdash v \in (R \leftrightarrow S) \leftrightarrow \Xi Z v \in (R \leftrightarrow S)^\neg \]
\[ \Xi Z v \in (R \leftrightarrow S)^\neg \]

where R and S are schemas that have signature variables \( r_1, r_2, \ldots \) and \( s_1, s_2, \ldots \) respectively, and

\[ \text{bind1} = (r_1 \equiv v.r_1, \ldots) \]
\[ \text{bind2} = (s_1 \equiv v.s_1, \ldots) \]

Schemas as predicates will be treated as membership statements by this conversion. If \( v \) is a binding display then \( v.x_i \) will be simplified.

Definition
\[ \text{val } z_Ξ_s_conv = Z_Ξ s_ELIM C z_Ξ_s_conv; \]

Errors
43016  ?0 is not of the form: \[ Z v \in (R \leftrightarrow S)^\neg \] where R and S are schemas
\end{verbatim}
Description  A conversion concerning the membership of a schema conjunction.

\[ \vdash v \in (R \land S) \iff z_{\in \land_s \text{conv}} \land v \in (R \land S) \land \]

where \( R \) and \( S \) are schemas that have signature variables \( r_1, r_2, \ldots \) and \( s_1, s_2, \ldots \) respectively, and

\[
\begin{align*}
\text{bind1} &= (r_1 \equiv v.r_1, 
\text{bind2} &= (s_1 \equiv v.s_1, 
\end{align*}
\]

Schemas as predicates will be treated as membership statements by this conversion. If \( v \) is a binding display then \( v.x \) will be simplified.

Definition

\[ val \ z_{\in \land_s \text{conv}} = Z_{\in \text{ELIM}_C} z_{\in \land_s \text{conv}}; \]

Errors

\[ 43001 \ ?0 \ is \ not \ of \ the \ form: \ \forall v \in (R \land S) \land \]

where \( R \) and \( S \) are schemas

Description  A conversion concerning the membership of a schema disjunction.

\[ \vdash v \in (R \lor S) \iff z_{\in \lor_s \text{conv}} \lor v \in (R \lor S) \lor \]

where \( R \) and \( S \) are schemas that have signature variables \( r_1, r_2, \ldots \) and \( s_1, s_2, \ldots \) respectively, and

\[
\begin{align*}
\text{bind1} &= (r_1 \equiv v.r_1, 
\text{bind2} &= (s_1 \equiv v.s_1, 
\end{align*}
\]

Schemas as predicates will be treated as membership statements by this conversion. If \( v \) is a binding display then \( v.x \) will be simplified.

Definition

\[ val \ z_{\in \lor_s \text{conv}} = Z_{\in \text{ELIM}_C} z_{\in \lor_s \text{conv}}; \]

Errors

\[ 43005 \ ?0 \ is \ not \ of \ the \ form: \ \exists v \in (R \lor S) \lor \]

where \( R \) and \( S \) are schemas
8.4. Reasoning about Schema Expressions

### SML
```sml
val z_¬s_conv : CONV;
val z_∈¬s_conv : CONV;
```

**Description**  A conversion concerning the membership of a schema negation.

#### Conversion

\[ \vdash v \in (\neg S) \iff \neg (v \in S) \]

where \( S \) is a schema. Schemas as predicates will be treated as membership statements by this conversion.

#### Definition

\[ \text{where } z \in \neg s \Rightarrow \neg s \Rightarrow \neg \
\]

#### Errors

43017 0 is not of the form: \( \forall v \in (\neg S) \) where \( S \) is a schema

### SML
```sml
val z_⇒s_conv : CONV;
val z_∈⇒s_conv : CONV;
```

**Description**  A conversion concerning the membership of a schema implication.

#### Conversion

\[ \vdash v \in (R \Rightarrow S) \iff (\text{bind1} \in R \Rightarrow \text{bind2} \in S) \]

where \( R \) and \( S \) are schemas that have signature variables \( r_1, r_2, \ldots \) and \( s_1, s_2, \ldots \) respectively, and

\[ \text{bind1} = (r_1 \equiv v.r_1, \ldots) \]
\[ \text{bind2} = (s_1 \equiv v.s_1, \ldots) \]

Schemas as predicates will be treated as membership statements by this conversion. If \( v \) is a binding display then \( v.x_i \) will be simplified.

#### Definition

\[ \text{where } z \in \Rightarrow s \Rightarrow \Rightarrow s \Rightarrow \|
\]

#### Errors

43006 0 is not of the form: \( \exists v \in (R \Rightarrow S) \) where \( R \) and \( S \) are schemas
**Val Z (∀ s) conv** = Z ∈ (∀ s) conv

**Val Z (∀ s) conv** ∈ (∀ s) conv

**Description**  A conversion concerning schema universals.

**Conversion**

\[
\vdash v \in (\exists D \mid P \bullet S) = \forall y : U \bullet ("predicate from D[y.y_1/y_1,...]"
\wedge P[y.y_1/y_1,...]) \Rightarrow (x_1 \equiv v.x_1,...,y_1 \equiv y.y_1,...) \in S
\]

where S is a schema that has signature variables \(x_1, x_2, \ldots\) and \(y_1, y_2, \ldots\) \(D\) a declaration that declares \(y_1, y_2, \ldots\). The “predicate from D” will also have schemas as predicates eliminated in favour of bindings being members of schemas. Schemas as predicates will be treated as membership statements by this conversion. If \(v\) is a binding display then \(v.x_i\) will be simplified.

**Definition**

\[\text{val } Z (\forall s) \text{ conv } = Z \in \text{ELIM } C ~ Z \in (\forall s) \text{ conv};\]

**Errors**

43030 : 0 is not of the form: \(\exists v \in (\forall D \mid P \bullet S)\) where \(S\) is a schema

---

**Val Z (∃ 1 s) conv** = Z ∈ (∃ 1 s) conv

**Val Z (∃ 1 s) conv** ∈ (∃ 1 s) conv

**Description**  A conversion concerning schema unique existentials.

**Conversion**

\[
\vdash v \in (\exists 1 D \mid P \bullet S) = \exists 1 y : U \bullet ("predicate from D[y.y_1/y_1,...]"
\wedge P[y.y_1/y_1,...]) \wedge (x_1 \equiv v.x_1,...,y_1 \equiv y.y_1,...) \in S
\]

where \(S\) is a schema that has signature variables \(x_1, x_2, \ldots\) and \(y_1, y_2, \ldots\) \(D\) a declaration that declares \(y_1, y_2, \ldots\). The “predicate from D” will also have schemas as predicates eliminated in favour of bindings being members of schemas. Schemas as predicates will be treated as membership statements by this conversion. If \(v\) is a binding display then \(v.x_i\) will be simplified.

**Definition**

\[\text{val } Z (∃ 1 s) \text{ conv } = Z \in \text{ELIM } C ~ Z \in (∃ 1 s) \text{ conv};\]

**Errors**

43021 : 0 is not of the form: \(\exists v \in (∃ 1 D \mid P \bullet S)\) where \(S\) is a schema
8.4. Reasoning about Schema Expressions

### Definition

If $S$ is a schema that has signature variables $x_1, x_2, \ldots$ and $y_1, y_2, \ldots$ $D$ a declaration that declares $y_1, y_2, \ldots$. The “predicate from D” will also have schemas as predicates eliminated in favour of bindings being members of schemas. Schemas as predicates will be treated as membership statements by this conversion. If $v$ is a binding display then $v . x_i$ will be simplified.

### Errors

\[43042\] ?0 is not of the form: $\exists v \in (\exists D \mid P \bullet S)^\gamma$ where $S$ is a schema

---

### SML

```sml
val z_∃s_conv : CONV;
val z_∈_∃s_conv : CONV;
```

**Description**  A conversion concerning membership of schema existentials.

\[
\begin{align*}
\vdash v \in (\exists D \mid P \bullet S) &= \exists y : U \bullet ("\text{predicate from } D[y.y_1/y_1,\ldots]" \\
& \quad \land P[y.y_1/y_1,\ldots]) \land \\
& \quad (x_1 \equiv v . x_1, \ldots, y_1 \equiv y . y_1,\ldots) \in S
\end{align*}
\]

where $S$ is a schema that has signature variables $x_1, x_2, \ldots$ and $y_1, y_2, \ldots$. The “predicate from D” will also have schemas as predicates eliminated in favour of bindings being members of schemas. Schemas as predicates will be treated as membership statements by this conversion. If $v$ is a binding display then $v . x_i$ will be simplified.

### Errors

\[43020\] ?0 is not of the form: $\exists v \in (\exists D \mid P \bullet S)^\gamma$ where $S$ is a schema

---

### SML

```sml
val z_∈_∃s_conv = Z_∈_ELIM_C z_∈_∃s_conv;
```

**Description**  A conversion concerning schema sequential composition.

\[
\begin{align*}
\vdash (R \_\_\_\_s S) &= \exists v \in (R \_\_\_\_s S)^\gamma \\
& \exists x_1 : U; \ldots; z_1 : U; \ldots; y_1 : U; \ldots; w_1 : U; \ldots |
\end{align*}
\]

\[
\begin{align*}
&\exists x_1 : U; \ldots; \bullet \\
&\land (v_1 \equiv v . x_1, \ldots, w_1 \equiv w . x_1, \ldots, x_1, \ldots) \in S
\end{align*}
\]

where $R$ and $S$ are schemas with signature variables as follows:

- **$R$**  
  - Unprimed: $x_1, x_2, \ldots$, $(z_{a_1}, z_{a_2}, \ldots)$, $(y_{b_1}, y_{b_2}, \ldots)$
  - Primed: $(x_{c_1}, x_{c_2}, \ldots)$, $(z'_{a_1}, z'_{a_2}, \ldots)$, $(y'_{b_1}, y'_{b_2}, \ldots)$

- **$S$**  
  - Unprimed: $y_1, y_2, \ldots$, $v_1, v_2, \ldots$
  - Primed: $(v'_{g_1}, v'_{g_2}, \ldots)$, $(w'_{g_1}, w'_{g_2}, \ldots)$

and $x_1, x_2, \ldots$ are variables whose names are not used for variables, or as labels for the binding types of $R$ or $S$. The signature variables enclosed in parentheses in the table are not shown in the theorem returned by the conversion but, if present, may result in extra schema declarations of the form $v : U$ and binding maplets of the form $v \equiv v$ where $v$ is e.g. $z_{a_1}$.

### Errors

\[43025\] ?0 is not of the form: $\exists v \in (R \_\_\_\_s S)^\gamma$ where $R$ and $S$ are schemas

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
**z_\theta_conv** conversion from a \( \theta \)-term to the binding constructor for the schema.

\[
\vdash \theta S = (n_1 \cong n_1, n_2 \cong n_2, \ldots) \quad z_\theta_conv \quad \check{\theta}S^\downarrow
\]

\( z_\theta_conv \) is as \( z_\theta_conv \), except that the conversion only succeeds if the \( \theta \) term is ill-formed (i.e. is not \( Z \)).

**Errors**
- 43010 \( ?\theta \) is not a \( \theta \)-term
- 43011 \( ?\theta \) is not an ill-formed \( \theta \)-term

**z_\theta_eq_conv** conversion from an equality of two \( \theta \)-terms, or a \( \theta \) term and a binding display, to an elementwise equality condition.

\[
\vdash ({}_{\theta S} decS = \theta T {}_{\theta T}) \quad z_\theta_eq_conv \quad \check{\theta}S = \theta T^\downarrow
\]

where \( decS \) and \( decT \) are the decoration of the respective schemas. Also:

\[
\vdash (\theta S = (n_1 \cong x_1, \ldots)) \quad z_\theta_eq_conv \quad \check{\theta}S = (n_1 \cong x_1, \ldots)^\downarrow
\]

**Uses** Used in combination with \( z_\text{binding_eq_conv2} \) to give \( \eta \)-terms the same status as binding displays.

**Errors**
- 43034 \( ?\theta \) is not of the form: \( \check{\theta}S = \theta T^\downarrow \) or \( \check{\theta}S = (n_1 \cong x_1, \ldots)^\downarrow \)

**z_\theta_\in_schema_conv** conversion from a predicate asserting that the \( \theta \)-term of a schema is a member of the schema to that schema as a predicate.

\[
\vdash \theta S \in S \leftrightarrow S \quad z_\theta_\in_schema_conv \quad \check{\theta}S \in S^\downarrow
\]

Note that the schemas cannot be decorated, as the type of \( \check{\theta}S^\downarrow \) is the same as the type of \( \check{\theta} S^\downarrow \). Other than that \( S \) may be any schema as a predicate, including schema references and horizontal schemas.

**See Also** \( z_\theta_\in_schema_intro_conv \) and \( z_\text{pred_dec_conv} \), which subsumes this conversion.

**Errors**
- 43002 \( ?\theta \) is not of the form \( \check{\theta}S \in S^\downarrow \) where \( \check{\theta}S^\downarrow \) is an undecorated schema
A conversion concerning the membership of a schema projection.

**Conversion**

\[
\begin{array}{c}
\vdash (R \upharpoonright S) = (R \cap S) \setminus \{x_1, x_2, \ldots\} \\
\Downarrow \{z \in \upharpoonright s \text{conv} \}
\end{array}
\]

where \(R\) and \(S\) are schemas and \(x_1, x_2, \ldots\) are the signature variables of \(R\) that are not signature variables of \(S\).

**Definition**

\[\text{val } z \in \upharpoonright s \text{conv} = \in C z \upharpoonright s \text{conv}\]

Schemas as predicates will be treated as membership statements by this conversion.

**Errors**

\[\text{43019 } ?0 \text{ is not of the form } \Downarrow \{z \in \upharpoonright s S\} \text{ where } R \text{ and } S \text{ are schemas}\]
THEORIES

9.1 Theory Listings

This section contains the listings of each theory.
9.1.1 The Theory \texttt{z_arithmetic_tools}

9.1.1.1 Parents

\texttt{z_numbers}

9.1.1.2 Children

\texttt{z_numbers1 \ z_library}

9.1.1.3 Constants

\texttt{Z_z, Z} \rightarrow \texttt{Z}

9.1.1.4 Definitions

\texttt{Z_z, Z} \vdash \texttt{ConstSpec} \left( \lambda (\texttt{NZ 1}) \right)

\hspace{2cm} \bullet \texttt{Z_z} (\texttt{NZ 1}) = \texttt{NZ 1}

\hspace{2cm} \wedge (\forall i j)

\hspace{4cm} \bullet \texttt{Z_z} (i + j) = \texttt{NZ z} \texttt{NZ z} (i + j)

\hspace{4cm} \wedge (\forall i \bullet \texttt{Z_z} (\sim i) = \texttt{NZ z} \sim (\texttt{NZ z} i))

\hspace{4cm} \wedge (\forall x \bullet \texttt{Z_z} (\texttt{NZ z} x) = x)

\hspace{4cm} \wedge (\forall y \bullet \texttt{Z_z} (\texttt{NZ z} y) = y))

\texttt{Z_z, Z} \vdash \texttt{Consistent} \left( \lambda (\texttt{NZ 1}) \right)

\hspace{2cm} \bullet \texttt{Z_z} (\texttt{NZ 1}) = \texttt{NZ 1}

\hspace{2cm} \wedge (\forall i j)

\hspace{4cm} \bullet \texttt{Z_z} (i + j) = \texttt{NZ z} \texttt{NZ z} (i + j)

\hspace{4cm} \wedge (\forall i \bullet \texttt{Z_z} (\sim i) = \texttt{NZ z} \sim (\texttt{NZ z} i))

\hspace{4cm} \wedge (\forall x \bullet \texttt{Z_z} (\texttt{NZ z} x) = x)

\hspace{4cm} \wedge (\forall y \bullet \texttt{Z_z} (\texttt{NZ z} y) = y))

9.1.1.5 Theorems

\texttt{Z_z, Z} \vdash \texttt{Consistent} \left( \lambda (\texttt{NZ 1}) \right)

\hspace{2cm} \bullet \texttt{Z_z} (\texttt{NZ 1}) = \texttt{NZ 1}

\hspace{2cm} \wedge (\forall i j)

\hspace{4cm} \bullet \texttt{Z_z} (i + j) = \texttt{NZ z} \texttt{NZ z} (i + j)

\hspace{4cm} \wedge (\forall i \bullet \texttt{Z_z} (\sim i) = \texttt{NZ z} \sim (\texttt{NZ z} i))

\hspace{4cm} \wedge (\forall x \bullet \texttt{Z_z} (\texttt{NZ z} x) = x)

\hspace{4cm} \wedge (\forall y \bullet \texttt{Z_z} (\texttt{NZ z} y) = y))

\texttt{Z_z, Z} \vdash \texttt{plus_thm} \forall i j \bullet \texttt{Z_z} (\texttt{NZ i + j}) = \texttt{Z_z} (\texttt{NZ i} + \texttt{Z_z} (\texttt{NZ j}))

\texttt{Z_z, Z} \vdash \texttt{times_thm} \forall i j \bullet \texttt{Z_z} (\texttt{NZ i * j}) = \texttt{Z_z} (\texttt{NZ i} * \texttt{Z_z} (\texttt{NZ j}))
\(\vdash \forall i \cdot z_z(i) - j = z_z i - z_z j\)

\(\vdash \forall i \cdot z_z(i) \sim i = \sim (z_z i)\)

\(\vdash \forall i, j \cdot z_z(z(i + j) = \langle Z_z z i \rangle + \langle Z_z z j \rangle)\)

\(\vdash \forall i, j \cdot z_z(z(i * j) = \langle Z_z z i \rangle * \langle Z_z z j \rangle)\)

\(\vdash \forall i, j \cdot z_z(z(i - j) = \langle Z_z z i \rangle - \langle Z_z z j \rangle)\)

\(\vdash \forall i \cdot z_z(\sim i) = \langle Z_z z i \rangle\)

\(\vdash \forall i, j \cdot z_z(i) = z_z j \iff i = j\)

\(\vdash \forall i, j \cdot z_z(i) = z_z j \iff i = j\)

\(\vdash \forall i, j \cdot \langle z(i, j) \rangle \in \langle Z \leq \rangle \iff z_z i \leq z_z j\)

\(\vdash \forall i, j \cdot \langle z(i, j) \rangle \in \langle Z < \rangle \iff z_z i < z_z j\)
9.1.2 The Z Theory \texttt{z\_bags}

9.1.2.1 Parents

\texttt{z\_sequences}

9.1.2.2 Children

\texttt{z\_library}

9.1.2.3 Global Variables

\begin{itemize}
  \item \texttt{bag X} \quad \mathbb{P} (X \leftrightarrow \mathbb{Z})
  \item \texttt{count[X]} \quad (X \leftrightarrow \mathbb{Z}) \leftrightarrow X \leftrightarrow \mathbb{Z}
  \item \texttt{(_ in _)[X]} \quad X \leftrightarrow X \leftrightarrow \mathbb{Z}
  \item \texttt{(_ \cup _)\[X]} \quad (X \leftrightarrow \mathbb{Z}) \times (X \leftrightarrow \mathbb{Z}) \leftrightarrow X \leftrightarrow \mathbb{Z}
  \item \texttt{items[X]} \quad (Z \leftrightarrow X) \leftrightarrow X \leftrightarrow \mathbb{Z}
  \item \texttt{([ ... ])[X]} \quad (Z \leftrightarrow X) \leftrightarrow X \leftrightarrow \mathbb{Z}
\end{itemize}

9.1.2.4 Fixity

\begin{itemize}
  \item \texttt{fun 0 rightassoc ([]} \quad \texttt{( [ ... ] )}
  \item \texttt{fun 30 leftassoc (_ \cup _)}
  \item \texttt{gen 70 rightassoc (bag _)}
  \item \texttt{rel (_ in _)}
\end{itemize}

9.1.2.5 Axioms

\begin{itemize}
  \item \texttt{count} \quad \vdash [X](\texttt{count[X]} \in \texttt{bag X} \rightarrow X \rightarrow \mathbb{N}
  \quad \wedge (\forall x : X; B : \texttt{bag X} \quad \bullet \texttt{count[X]} B = (\lambda x : X \bullet 0) \oplus B))
  \item \texttt{(_ in _)} \quad \vdash [X](\texttt{(_ in _)[X]} \in X \leftrightarrow \texttt{bag X}
  \quad \wedge (\forall x : X; B : \texttt{bag X} \quad \bullet (x, B) \in (\texttt{(_ in _)[X]} \Leftrightarrow x \in \text{dom } B))
  \item \texttt{(_ \cup _)} \quad \vdash [X](\texttt{(_ \cup _)[X]} \in \texttt{bag X} \times \texttt{bag X} \rightarrow \texttt{bag X}
  \quad \wedge (\forall B, C : \texttt{bag X}; x : X \quad \bullet \texttt{count ((_ \cup _)[X]} (B, C)) x
  \quad = \texttt{count B x} + \texttt{count C x}))
  \item \texttt{items} \quad \vdash [X](\texttt{items[X]} \in \texttt{seq X} \rightarrow \texttt{bag X}
  \quad \wedge (\forall s : \texttt{seq X}; x : X \quad \bullet \texttt{count (items[X]} s) x
  \quad = \# \{ i : \text{dom } s \mid s i = x\})
  \item \texttt{[ ... ]} \quad \vdash [X](\texttt{[ ... ][X]} \in \texttt{seq X} \rightarrow \texttt{bag X}
  \quad \wedge (\texttt{[ ... ][X]} () = \{\})
  \quad \wedge (\forall x : X; s : \texttt{seq X} \quad \bullet (\texttt{[ ... ][X]} ((x) \cap s)
  \quad = (\texttt{[ ... ][X]} s \oplus \{ x \mapsto (\texttt{[ ... ][X]} s x + 1)\}))
\end{itemize}
9.1.2.6 Definitions

\begin{align*}
\text{bag}_\_ & \vdash [X](\text{bag } X = X \rightarrow \mathbb{N}_I)
\end{align*}
9.1.3 The Z Theory \texttt{z}\_functions

9.1.3.1 Parents

\textit{z}\_relations

9.1.3.2 Children

\textit{z}\_functions\_1 \textit{z}\_numbers

9.1.3.3 Global Variables

\begin{align*}
X & \rightarrow Y & \implies (X \leftrightarrow Y) \\
X & \leftrightarrow Y & \implies (X \leftrightarrow Y) \\
X & \rightarrow Y & \implies (X \leftrightarrow Y) \\
X & \leftrightarrow Y & \implies (X \leftrightarrow Y) \\
X & \rightarrow Y & \implies (X \leftrightarrow Y) \\
X & \rightarrow Y & \implies (X \leftrightarrow Y)
\end{align*}

9.1.3.4 Fixity

\texttt{gen} 5 \texttt{rightassoc}

\begin{align*}
(\_ \rightarrow \_) & (\_ \leftrightarrow \_) & (\_ \rightarrow \_) & (\_ \rightarrow \_) & (\_ \rightarrow \_) & (\_ \rightarrow \_)
\end{align*}

9.1.3.5 Definitions

\begin{align*}
\_ & \rightarrow \_ & \vdash [X, Y](X \rightarrow Y) \\
& & = \{ f : X \rightarrow Y \mid \forall x : X; y1, y2 : Y \\
& & \quad \cdot x \mapsto y1 \in f \land x \mapsto y2 \in f \Rightarrow y1 = y2 \}
\end{align*}

\begin{align*}
\_ & \leftrightarrow \_ & \vdash [X, Y](X \leftrightarrow Y) \\
& & = \{ f : X \rightarrow Y \mid \forall x1, x2 : \text{dom } f \cdot f \ x1 = f \ x2 \Rightarrow x1 = x2 \}
\end{align*}

\begin{align*}
\_ & \rightarrow \_ & \vdash [X, Y](X \rightarrow Y = (X \leftrightarrow Y) \cap (X \rightarrow Y)) \\
\_ & \leftrightarrow \_ & \vdash [X, Y](X \rightarrow Y = \{ f : X \rightarrow Y \mid \text{ran } f = Y \}) \\
\_ & \rightarrow \_ & \vdash [X, Y](X \rightarrow Y = (X \rightarrow Y) \cap (X \rightarrow Y)) \\
\_ & \leftrightarrow \_ & \vdash [X, Y](X \rightarrow Y = (X \rightarrow Y) \cap (X \rightarrow Y))
\end{align*}
9.1.3.6 Theorems

$\textbf{z}_-\_\text{thm} \quad \vdash \forall f : U; X : U; Y : U$
$\quad \bullet f \in X \rightarrow Y$
$\quad \iff f \in X \rightarrow Y$
$\quad \land (\forall x : X; y1, y2 : Y$
$\quad \bullet (x, y1) \in f \land (x, y2) \in f \Rightarrow y1 = y2)$

$\textbf{z}_-\_\text{thm1} \quad \vdash \forall f : U; X : U; Y : U$
$\quad \bullet f \in X \rightarrow Y$
$\quad \iff f \in X \rightarrow Y$
$\quad \land (\forall x : U; y1, y2 : U$
$\quad \land x \in X \land y1 \in Y \land y2 \in Y$
$\quad \bullet (x, y1) \in f \land (x, y2) \in f \Rightarrow y1 = y2)$

$\textbf{z}_-\_\text{thm} \quad \vdash \forall f : U; X : U; Y : U$
$\quad \bullet f \in X \rightarrow Y$
$\quad \iff f \in X \rightarrow Y$
$\quad \land (\forall x1, x2 : U$
$\quad \land x1 \in dom f \land x2 \in dom f$
$\quad \bullet f x1 = f x2 \Rightarrow x1 = x2)$

$\textbf{z}_-\_\text{thm} \quad \vdash \forall f : U; X : U; Y : U$
$\quad \bullet f \in X \rightarrow Y$
$\quad \iff f \in X \rightarrow Y$
$\quad \land (\forall x1, x2 : U$
$\quad \land x1 \in dom f \land x2 \in dom f$
$\quad \bullet f x1 = f x2 \Rightarrow x1 = x2)$

$\textbf{z}_-\_\text{thm} \quad \vdash \forall f : U; X : U; Y : U$
$\quad \bullet f \in X \rightarrow Y$
$\quad \iff f \in X \rightarrow Y$
$\quad \land (\forall x1, x2 : U$
$\quad \land x1 \in dom f \land x2 \in dom f$
$\quad \bullet f x1 = f x2 \Rightarrow x1 = x2)$

$\textbf{z}_-\_\text{app\_thm} \quad \vdash \forall X : U; Y : U; f : U; x : U$
$\quad \bullet f \in X \rightarrow Y \land x \in X \Rightarrow f x \in Y \land (x, f x) \in f$

$\textbf{z}_-\_\text{first\_thm} \quad \vdash \forall x : U \bullet x \in \text{first} \iff x.1.1 = x.2$

$\textbf{z}_-\_\text{second\_thm} \quad \vdash \forall x : U \bullet x \in \text{second} \iff x.1.2 = x.2$

$\textbf{z}_-\_\text{app\_\_rel\_thm} \quad \vdash \forall X : U; Y : U \bullet \forall f : X \rightarrow Y; x : X \bullet (x, f x) \in f$

$\textbf{z}_-\_\text{app\_eq\_\_rel\_thm} \quad \vdash \forall X : U; Y : U$
$\quad \bullet \forall f : X \rightarrow Y; x : X; z : U \bullet f x = z \iff (x, z) \in f$

$\textbf{z}_-\_\text{rel\_\_app\_eq\_thm} \quad \vdash \forall X : U; Y : U$
$\quad \bullet \forall f : X \rightarrow Y; x : X; z : U \bullet (x, z) \in f \iff f x = z$
\[ \begin{align*}
\text{z\_\rightarrow\_clauses} & \quad \vdash \forall Y : U \bullet \{ \} \rightarrow Y = \{ \} \wedge Y \rightarrow \{ \} = \{ \} \\
\text{z\_\rightarrow\_clauses} & \quad \vdash \forall Y : U \\
& \quad \bullet \{ \} \rightarrow Y = \{ \} \\
& \quad \wedge Y \rightarrow \{ \} = \{ x : U \mid x = \} \wedge Y = \{ \} \\
\text{z\_\rightarrow\_clauses} & \quad \vdash \forall Y : U \bullet \{ \} \rightarrow Y = \{ \} \wedge Y \rightarrow \{ \} = \{ \} \\
\text{z\_\rightarrow\_clauses} & \quad \vdash \forall Y : U \\
& \quad \bullet \{ \} \rightarrow Y = \{ x : U \mid x = \} \wedge Y = \{ \} \\
& \quad \wedge Y \rightarrow \{ \} = \{ x : U \mid x = \} \wedge Y = \{ \} \\
\text{z\_\rightarrow\_clauses} & \quad \vdash \forall Y : U \\
& \quad \bullet \{ \} \rightarrow Y = \{ x : U \mid x = \} \wedge Y = \{ \} \\
& \quad \wedge Y \rightarrow \{ \} = \{ x : U \mid x = \} \wedge Y = \{ \} \\
\text{z\_fun\_app\_clauses} & \quad \vdash \forall f : U; x : U; y : U; X : U; Y : U \\
& \quad \bullet \ (f \in X \rightarrow Y \\
& \quad \quad \vee f \in X \rightarrow Y \\
& \quad \quad \vee f \in X \rightarrow Y \\
& \quad \quad \vee f \in X \rightarrow Y \\
& \quad \quad \vee f \in X \rightarrow Y \\
& \quad \quad \vee f \in X \rightarrow Y \\
& \quad \quad \wedge (x, y) \in f \\
& \quad \Rightarrow f \ x = y \\
\text{z\_fun\_e\_clauses} & \quad \vdash \forall f : U; x : U; X : U; Y : U \\
& \quad \bullet \ ((f \in X \rightarrow Y \vee f \in X \rightarrow Y \vee f \in X \rightarrow Y \vee f \in X \rightarrow Y) \\
& \quad \quad \wedge x \in X \\
& \quad \quad \Rightarrow f \ x \in Y \\
& \quad \wedge ((f \in X \rightarrow Y \vee f \in X \rightarrow Y) \vee f \in X \rightarrow Y) \\
& \quad \quad \wedge x \in \text{dom} f \\
& \quad \quad \Rightarrow f \ x \in Y \\
\text{z\_fun\_dom\_clauses} & \quad \vdash \forall f : U; X : U; Y : U \\
& \quad \bullet \ (f \in X \rightarrow Y \vee f \in X \rightarrow Y \Rightarrow \text{dom} f \subseteq X) \\
& \quad \wedge (f \in X \rightarrow Y \vee f \in X \rightarrow Y) \Rightarrow \text{dom} f = X \\
\text{z\_fun\_ran\_clauses} & \quad \vdash \forall f : U; X : U; Y : U \\
& \quad \bullet \ (f \in X \rightarrow Y \vee f \in X \rightarrow Y \Rightarrow \text{ran} f \subseteq Y) \\
& \quad \wedge (f \in X \rightarrow Y \vee f \in X \rightarrow Y \Rightarrow \text{ran} f = Y) 
\end{align*} \]
9.1.4 The Z Theory z_functions

9.1.4.1 Parents

\textit{z_functions}

9.1.4.2 Children

\textit{z_numbers}

9.1.4.3 Theorems

\texttt{z\_\oplus\_\rightarrow\_app\_thm}
\[ \vdash \forall f : U; x : U; y : U \cdot (f \oplus \{x \mapsto y\}) \; x = y \]

\texttt{z\_dom\_\oplus\_\rightarrow\_thm}
\[ \vdash \forall f : U; x : U; y : U \]
\[ \bullet \; \text{dom} \; (f \oplus \{x \mapsto y\}) = \text{dom} \; f \cup \{x\} \]

\texttt{z\_\oplus\_\rightarrow\_\in\_\rightarrow\_thm}
\[ \vdash [X, Y]; (\forall f : X \rightarrow Y; x : X; y : Y \cdot f \oplus \{x \mapsto y\} \in X \rightarrow Y) \]

\texttt{z\_\oplus\_\rightarrow\_app\_thm1}
\[ \vdash [X, Y]; (\forall f : X \rightarrow Y; x2 : X; x1 : U; y : U \]
\[ \bullet \; (f \oplus \{x1 \mapsto y\}) \; x2 = f \; x2) \]

\texttt{z\_\less\_\rightarrow\_thm}
\[ \vdash [Y, Z]; (\forall X : U; f : Y \rightarrow Z \]
\[ \bullet \; X \subseteq Y \Rightarrow X \triangleleft f \in X \rightarrow \text{ran} \; (X \triangleleft f)) \]

\texttt{z\_ran\_\less\_\thm}
\[ \vdash [Y, Z]; (\forall X : U; f : Y \rightarrow Z \]
\[ \bullet \; \text{ran} \; (X \triangleleft f) \]
\[ \vdash \{y : U \]
\[ \bullet \; (X, y) \in f \;
\[ \bullet \; \exists x \in X \;
\[ \bullet \; x = X \land y \in Y \]

\texttt{z\_\rightarrow\_ran\_eq\_\rightarrow\_thm}
\[ \vdash \forall A : U; B : U \]
\[ \bullet \; (\exists f : A \rightarrow B \bullet \; \text{ran} \; f = B) \Leftrightarrow (\exists f : A \rightarrow B \bullet \; \text{true}) \]

\texttt{z\_\rightarrow\_ran\_eq\_\rightarrow\_thm}
\[ \vdash \forall A : U; B : U \]
\[ \bullet \; (\exists f : A \rightarrow B \bullet \; \text{ran} \; f = B) \Leftrightarrow (\exists f : A \rightarrow B \bullet \; \text{true}) \]

\texttt{z\_ran\_mono\_thm}
\[ \vdash \forall X : U; Y, Z : U; f : U \]
\[ \bullet \; f \in X \rightarrow Y \land \text{ran} \; f \subseteq Z \]
\[ \bullet \; f \in X \rightarrow Z \]

\texttt{z\_\rightarrow\_thm2}
\[ \vdash \forall A : U; B : U; f : U \]
\[ \bullet \; f \in A \rightarrow B \Leftrightarrow f \in \text{dom} \; f \rightarrow B \land \text{dom} \; f \subseteq A \]
\[
\begin{align*}
z_{\to\_thm} & \vdash \forall A : \mathbb{U}; B : \mathbb{U}; f : \mathbb{U} \\
& \quad \bullet f \in A \to B \iff f \in A \to B \land B \subseteq \text{ran } f \\
z_{\to\_thm} & \vdash |X, \\
& \quad Y|(X \mapsto Y) \\
& \quad = \{ f : X \to Y \\
& \quad \mid \forall x_1, x_2 : \mathbb{U}; y : \mathbb{U} \\
& \quad \bullet (x_1, y) \in f \land (x_2, y) \in f \implies x_1 = x_2 \}\} \\
z_{\to\_dom\_thm} & \vdash \forall A : \mathbb{U}; B : \mathbb{U}; f : \mathbb{U} \bullet f \in A \to B \implies f \in \text{dom } f \to B \\
z_{\to\_thm} & \vdash \forall A : \mathbb{U}; B : \mathbb{U}; f : \mathbb{U} \\
& \quad \bullet f \in A \to B \\
& \quad \iff f \in A \to B \\
& \quad \land (\forall x, y : \mathbb{U}; z : \mathbb{U} \\
& \quad \bullet (x, z) \in f \land (y, z) \in f \implies x = y) \\
z_{\cup\_\to\_thm} & \vdash \forall A : \mathbb{U}; B : \mathbb{U}; C : \mathbb{U}; D : \mathbb{U}; f : \mathbb{U}; g : \mathbb{U} \\
& \quad \bullet f \in A \to B \land g \in C \to D \implies f \cup g \in A \cup C \to B \cup D \\
z_{\text{ran }\cup\_\to\_thm} & \vdash \forall f : \mathbb{U}; g : \mathbb{U} \bullet \text{ran } (f \cup g) = \text{ran } f \cup \text{ran } g \\
z_{\cup\_\to\_thm} & \vdash \forall A : \mathbb{U}; B : \mathbb{U}; C : \mathbb{U}; D : \mathbb{U}; f : \mathbb{U}; g : \mathbb{U} \\
& \quad \bullet f \in A \to B \land g \in C \to D \land A \cap C = \{} \\
& \quad \implies f \cup g \in A \cup C \to B \cup D \\
z_{\cup\_\to\_thm} & \vdash \forall A : \mathbb{U}; B : \mathbb{U}; C : \mathbb{U}; D : \mathbb{U}; f : \mathbb{U}; g : \mathbb{U} \\
& \quad \bullet f \in A \to B \land g \in C \to D \land A \cap C = \{} \\
& \quad \implies f \cup g \in A \cup C \to B \cup D \\
z_{\cup\_\to\_thm} & \vdash \forall A : \mathbb{U}; B : \mathbb{U}; C : \mathbb{U}; D : \mathbb{U}; f : \mathbb{U}; g : \mathbb{U} \\
& \quad \bullet f \in A \to B \land g \in C \to D \land A \cap C = \{} \\
& \quad \implies f \cup g \in A \cup C \to B \cup D \\
z_{\circ\_\to\_thm} & \vdash \forall A : \mathbb{U}; B : \mathbb{U}; C : \mathbb{U}; f : \mathbb{U}; g : \mathbb{U} \\
& \quad \bullet f \in A \to B \land g \in B \to C \\
& \quad \implies g \circ f \in A \to C \\
z_{\circ\_\to\_thm} & \vdash \forall A : \mathbb{U}; B : \mathbb{U}; C : \mathbb{U}; f : \mathbb{U}; g : \mathbb{U} \\
& \quad \bullet f \in A \to B \land g \in B \to C \\
& \quad \implies g \circ f \in A \to C \\
z_{\circ\_\to\_thm} & \vdash \forall A : \mathbb{U}; B : \mathbb{U}; C : \mathbb{U}; f : \mathbb{U}; g : \mathbb{U} \\
& \quad \bullet f \in A \to B \land g \in B \to C \\
& \quad \implies g \circ f \in A \to C \\
z_{\circ\_\to\_thm} & \vdash \forall A : \mathbb{U}; B : \mathbb{U}; C : \mathbb{U}; f : \mathbb{U}; g : \mathbb{U} \\
& \quad \bullet f \in A \to B \land g \in B \to C \\
& \quad \implies g \circ f \in A \to C \\
z_{\text{rel }\text{inv }\to\_\to\_thm} & \vdash \forall A : \mathbb{U}; B : \mathbb{U}; f : \mathbb{U} \bullet f \in A \to B \implies f \sim \in B \to A \\
z_{\text{id }\_thm} & \vdash \forall X : \mathbb{U}; x, y : \mathbb{U} \bullet (x, y) \in \text{id }X \iff x \in X \land x = y \\
z_{\text{id }\_\to\_thm} & \vdash \forall X : \mathbb{U} \bullet \text{id }X \in X \to X \\
z_{\text{simple }\text{swap }\_\to\_thm} & \vdash \forall x, y : \mathbb{U} \bullet \{(x, y), (y, x)\} \in \{x, y\} \implies \{x, y\} \\
z_{\text{swap }\_\to\_thm} & \vdash \forall X : \mathbb{U} \\
& \quad \bullet \forall x, y : X \\
& \quad \bullet \exists g : X \to X \bullet (x, y) \in g \land (y, x) \in g \\
z_{\text{trans }\_\to\_thm} & \vdash \forall X : \mathbb{U} \bullet \forall x, y : X \bullet \exists g : X \to X \bullet (x, y) \in g \\
z_{\text{dom }f \_\to\_f \_\to\_thm} & \vdash \forall A : \mathbb{U}; B : \mathbb{U}; f : \mathbb{U} \\
& \quad \bullet f \in A \iff B \\
& \quad \implies \{x : A; y : B\}
\end{align*}
\]
\[ (x, y) \in f \]
\[ (x, (x, y)) \}
\[ \in dom f \leftrightarrow f \]

\[ \defy z_{\text{dom } f \to f_{\text{thm}}} \]
\[ \forall A : \text{U}; B : \text{U}; f : \text{U} \]
\[ \bullet f \in A \to B \]
\[ \Rightarrow \{ x : A; y : B \]
\[ | \ (x, y) \in f \]
\[ \bullet (x, (x, y)) \}
\[ \in dom f \to f \]

\[ \defy z_{\text{dom } f \to f_{\text{thm}}} \]
\[ \forall A : \text{U}; B : \text{U}; f : \text{U} \]
\[ \bullet f \in A \to B \]
\[ \Rightarrow \{ x : A; y : B \]
\[ | \ (x, y) \in f \]
\[ \bullet (x, (x, y)) \}
\[ \in dom f \to f \]

\[ \defy z_{\text{dom } f \to f_{\text{thm}}} \]
\[ \forall A : \text{U}; B : \text{U}; f : \text{U} \]
\[ \bullet f \in A \to B \]
\[ \Rightarrow \{ x : A; y : B \]
\[ | \ (x, y) \in f \]
\[ \bullet (x, (x, y)) \}
\[ \in dom f \to f \]

\[ \defy z_{\text{\cap } \to f_{\text{thm}}} \]
\[ \forall X : \text{U}; Y : \text{U}; f, g : \text{U} \]
\[ \mid f \in X \leftrightarrow Y \]
\[ \bullet f \cap g \in \text{dom} \ (f \cap g) \leftrightarrow \text{ran} \ (f \cap g) \]

\[ \defy z_{\text{ran } f_{\text{thm}}} \]
\[ \forall X : \text{U}; f : \text{U} \bullet f \in X \leftrightarrow \text{ran} \ f \leftrightarrow f \in X \leftrightarrow \text{U} \]

\[ \defy z_{\text{ran } f_{\text{thm}}} \]
\[ \forall X : \text{U}; f : \text{U} \bullet f \in X \to \text{ran} \ f \leftrightarrow f \in X \to \text{U} \]

\[ \defy z_{\text{\cap } f \to f_{\text{thm}}} \]
\[ \forall X : \text{U}; Y : \text{U}; f, g : \text{U} \]
\[ \mid f \in X \to Y \]
\[ \bullet f \cap g \in \text{dom} \ (f \cap g) \rightarrow \text{ran} \ (f \cap g) \]

\[ \defy z_{\text{\cap } f \to f_{\text{thm}}} \]
\[ \forall X : \text{U}; Y : \text{U}; f, g : \text{U} \]
\[ \mid f \in X \to Y \]
\[ \bullet f \cap g \in \text{dom} \ (f \cap g) \rightarrow \text{ran} \ (f \cap g) \]

\[ \defy z_{\text{\cap } f \to f_{\text{thm}}} \]
\[ \forall X : \text{U}; Y : \text{U}; f, g : \text{U} \]
\[ \mid f \in X \to Y \]
\[ \bullet f \cap g \in \text{dom} \ (f \cap g) \rightarrow \text{ran} \ (f \cap g) \]

\[ \defy z_{\text{\cap } f \to f_{\text{thm}}} \]
\[ \forall X : \text{U}; Y : \text{U}; f, g : \text{U} \]
\[ \mid f \in X \to Y \]
\[ \bullet f \cap g \in \text{dom} \ (f \cap g) \rightarrow \text{ran} \ (f \cap g) \]

\[ \defy z_{\text{\cap } f \to f_{\text{thm}}} \]
\[ \forall X : \text{U}; Y : \text{U}; f, g : \text{U} \]
\[ \mid f \in X \to Y \]
\[ \bullet f \cap g \in \text{dom} \ (f \cap g) \rightarrow \text{ran} \ (f \cap g) \]
\begin{itemize}
  \item \( \forall f : X \rightarrow Y; \ x : U; \ y : U \)
  \item \( (x, y) \in f \Rightarrow f \setminus \{(x, y)\} \in X \setminus \{x\} \rightarrow Y \)
\end{itemize}

z\_\rightarrow\_diff\_singleton\_thm
\begin{center}
\begin{array}{l}
\vdash \forall X : U; \ Y : U \\
\quad \bullet \ \forall f : X \rightarrow Y; \ x : X; \ y : Y \\
\quad \bullet \ (x, y) \in f \Rightarrow f \setminus \{(x, y)\} \in X \setminus \{x\} \rightarrow Y \setminus \{y\}
\end{array}
\end{center}

z\_singleton\_app\_thm
\begin{center}
\begin{array}{l}
\vdash \forall x : U; \ y : U \bullet \{(x, y)\} \ x = y
\end{array}
\end{center}

z\_empty\_\rightarrow\_thm
\begin{center}
\begin{array}{l}
\vdash \forall X : U \bullet (\exists f : \{} \rightarrow X \bullet true) \iff X = \{}
\end{array}
\end{center}

z\_\rightarrow\_empty\_thm
\begin{center}
\begin{array}{l}
\vdash \forall X : U \bullet (\exists f : X \rightarrow \{} \bullet true) \iff X = \{}
\end{array}
\end{center}
9.1.5 The Theory \textit{z\_language}

9.1.5.1 Parents

\begin{itemize}
  \item $\mathbb{Z}$ \textit{hol}
\end{itemize}

9.1.5.2 Children

\begin{itemize}
  \item \textit{z\_language\_ps}
\end{itemize}

9.1.5.3 Notes

This theory is a cache theory; its contents have not been listed.
9.1.6 The Z Theory z_language_ps

9.1.6.1 Parents

\[ z\text{\_language} \]

9.1.6.2 Children

\[ z\text{\_sets} \]

9.1.6.3 Theorems

\[ z\text{\_app\_thm} \vdash \forall a : U; f : U; x : U \]
\[ \quad \bullet (\forall f_a : U \mid (a, f_a) \in f \bullet f_a = x) \land (a, x) \in f \]
\[ \quad \Rightarrow f_a = x \]

\[ z\text{\_sets\_ext\_thm} \vdash \forall x : U; y : U \bullet x = y \Leftrightarrow (\forall z : U \bullet z \in x \Leftrightarrow z \in y) \]

\[ z\text{\_\in\_P\_thm1} \vdash \forall t : U; u : U \bullet t \in P u \Leftrightarrow (\forall z : U \bullet z \in t \Rightarrow z \in u) \]

\[ z\text{\_\in\_app\_thm} \vdash \forall a : U; x : U; f : U \]
\[ \quad \bullet (\exists f_x : U \]
\[ \quad \quad \bullet a \in f_x \]
\[ \quad \quad \land (x, f_x) \in f \]
\[ \quad \quad \land (\forall f_{x1} : U \bullet (x, f_{x1}) \in f \Rightarrow f_{x1} = f_x)) \]
\[ \quad \Rightarrow a \in f_x \]

\[ z\text{\_app\_\in\_thm} \vdash \forall a : U; x : U; f : U \]
\[ \quad \bullet (\exists f_x : U \]
\[ \quad \quad \bullet f_x \in a \]
\[ \quad \quad \land (x, f_x) \in f \]
\[ \quad \quad \land (\forall f_{x1} : U \bullet (x, f_{x1}) \in f \Rightarrow f_{x1} = f_x)) \]
\[ \quad \Rightarrow f_x \in a \]
9.1.7 The Z Theory z_library

9.1.7.1 Parents

\[ z_{\text{sequences1}} \quad z_{\text{arithmetic_tools}} \quad z_{\text{bags}} \]
9.1.8 The Z Theory \textit{z\_numbers}

9.1.8.1 Parents

\textit{z\_functions}

9.1.8.2 Children

\textit{z\_reals z\_sequences}
\textit{z\_numbers1 z\_arithmetic\_tools}

9.1.8.3 Global Variables

\begin{align*}
Z & \rightarrow P \ Z \\
N & \rightarrow P \ Z \\
(\sim \_) & \rightarrow Z \\
(\_ + \_) & \rightarrow Z \\
(\_ - \_) & \rightarrow Z \\
(\_ \ast \_) & \rightarrow Z \\
(\_ \leq \_) & \rightarrow Z \\
(\_ < \_) & \rightarrow Z \\
(\_ \geq \_) & \rightarrow Z \\
(\_ > \_) & \rightarrow Z \\
(abs \_) & \rightarrow Z \\
(\_ \div \_) & \rightarrow Z \\
(\_ \mod \_) & \rightarrow Z \\
N_1 & \rightarrow P \ Z \\
succ & \rightarrow Z \\
iter[X] & \rightarrow (X \rightarrow X) \rightarrow X \rightarrow X \\
(\_ \_\_)[X] & \rightarrow (X \rightarrow X) \times Z \rightarrow X \rightarrow X \\
(\_ .. \_)[X] & \rightarrow Z \times Z \rightarrow P \ Z \\
F X & \rightarrow P (P X) \\
F_1 X & \rightarrow P (P X) \\
\#[X] & \rightarrow P X \rightarrow Z \\
X \rightarrow Y & \rightarrow P (X \rightarrow Y) \\
X \rightarrow Y & \rightarrow P (X \rightarrow Y) \\
\min & \rightarrow P Z \rightarrow Z \\
\max & \rightarrow P Z \rightarrow Z
\end{align*}

9.1.8.4 Fixity

fun 20 \textit{leftassoc} \hspace{1cm} (\_ .. \_)

fun 30 \textit{leftassoc} \hspace{1cm} (\_ + \_)(\_ - \_)

fun 40 \textit{leftassoc} \hspace{1cm} (\_ \div \_)(\_ \mod \_)(\_ \ast \_)

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
fun 50 rightassoc
    (abs _) (∼ _)

fun 70 rightassoc
    (_, _)

gen 5 rightassoc
    (_, ▷ _)

gen 70 rightassoc
    (⌜ integers ⌝) (⌜ I ⌝)

rel
    (_ < _)(_ > _)(_ ≤ _)(_ ≥ _)

9.1.8.5 Axioms

N

∼ _

_ + _ ⊢ ((_ + _) ∈ Z × Z → Z
    ∧ (∼ _) ∈ Z → Z
    ∧ N ∈ P Z
    ∧ (∀ i, j, k : Z
        • i + j + k = i + (j + k)
        ∧ i + j = j + i
        ∧ i + ∼ i = 0
        ∧ i + 0 = i)
    ∧ (∀ h : P Z
        • 1 ∈ h ∧ (∀ i, j : h • i + j ∈ h → ∼ i ∈ h)
        ⇒ h = Z)
    ∧ N = ∩ {s : P Z | 0 ∈ s ∧ {i : s • i + 1} ⊆ s}
    ∧ ∼ 1 ∉ N

z\text{\small int}.\text{\small def}

Z'Int

⊢ (∀ i • ∃ Z'Int (i + 1) = ∃ Z'Int i + 1)

_ − _ ⊢ (_ − _) ∈ Z × Z → Z ∧ (∀ i, j : Z • i − j = i + ∼ j)

_ * _ ⊢ (_ * _) ∈ Z × Z → Z
    ∧ (∀ i, j, k : Z
        • i * j * k = i * (j * k)
        ∧ i * j = j * i
        ∧ i * (j + k) = i * j + i * k
        ∧ 1 * i = i)

_ ≤ _

_ < _

_ ≥ _

_ > _ ⊢ {(_ ≤ _), (_ < _), (_ ≥ _), (_ > _)} ⊆ Z → Z
    ∧ (∀ i, j : Z
        • i ≤ j ⇔ j − i ∈ N
        ∧ (i < j ⇔ i + 1 ≤ j)
        ∧ (i ≥ j ⇔ j ≤ i)
        ∧ (i > j ⇔ j < i))
Chapter 9. THEORIES

9.1.8.6 Definitions

\( Z \)  
\( \vdash Z = \mathbb{U} \)

\( \mathbb{N}_1 \)  
\( \vdash \mathbb{N}_1 = \mathbb{N} \setminus \{0\} \)

\( \mathbb{F} \)  
\( \vdash [X][\mathbb{F}] \)  
\( = \{ S : \mathbb{P}_1 \mathbb{Z} ; m : \mathbb{Z} \} \)  
\( \mid m \in S \wedge (\forall n : S \bullet m \leq n) \)  
\( \bullet S \mapsto m \)  

\( \mathbb{F}_1 \)  
\( \vdash [X][\mathbb{F}_1] X = \mathbb{F} X \setminus \{ \emptyset \} \)

\( \rightarrow \rightarrow \)  
\( \vdash [X, Y] (X \rightarrow \rightarrow Y) = \{ f : X \rightarrow Y \mid \text{dom } f \in \mathbb{F} X \} \)

9.1.8.7 Theorems

\( \text{z_plus_comm_thm} \)  
\( \vdash \forall i, j : \mathbb{U} \bullet i + j = j + i \)

\( \text{z_plus_assoc_thm} \)
9.1. Theory Listings

\[ \vdash \forall i, j, k : \mathbb{U} \cdot i + j + k = i + (j + k) \]

**z_plus_assoc_thm1**

\[ \vdash \forall i, j, k : \mathbb{U} \cdot i + (j + k) = i + j + k \]

**z_plus_order_thm**

\[ \vdash \forall i : \mathbb{U} \]
\[ \bullet \forall j, k : \mathbb{U} \]
\[ \bullet j + i = i + j \]
\[ \wedge i + j + k = i + (j + k) \]
\[ \wedge j + (i + k) = i + (j + k) \]

**z_plus0_thm**

\[ \vdash \forall i : \mathbb{U} \cdot i + 0 = i \wedge 0 + i = i \]

**z_plus_minus_thm**

\[ \vdash \forall i : \mathbb{U} \cdot i + i = 0 \wedge i + i = 0 \]

**z_N_thm**

\[ \vdash N = \bigcap \{ s : \mathbb{U} \mid 0 \in s \wedge \{ i : s \mid i + 1 \} \subseteq s \} \]
\[ \wedge \sim 1 \in N \]

**z_plus_cyclic_group_thm**

\[ \vdash \forall h : \mathbb{U} \]
\[ \bullet I \in h \wedge (\forall i, j : h \cdot i + j \in h \wedge \sim i \in h) \]
\[ \Rightarrow h = \mathbb{U} \]

**z_int_homomorphism_thm**

\[ \vdash (\forall i, j : \mathbb{Z}' \mathbb{I}nt (i + j) ) \Rightarrow (\forall i, j : \mathbb{Z}' \mathbb{I}nt (i + j) ) \]

**z_Z_induction_thm**

\[ \vdash (\forall p \]
\[ \bullet p \vdash Z' \mathbb{I}nt (i + j) \]
\[ \wedge (\forall i \cdot p i \Rightarrow p i) \]
\[ \wedge (\forall i \cdot p i \wedge p j \Rightarrow p i + j) \]
\[ \Rightarrow (\forall m \cdot p m) \]

**z_N_plus1_thm**

\[ \vdash \forall i : \mathbb{N} \cdot i + 1 \in \mathbb{N} \]

**z_0_N_thm**

\[ \vdash 0 \in \mathbb{N} \]

**z_N_induction_thm**

\[ \vdash (\forall p \]
\[ \bullet p \vdash Z' \mathbb{I}nt (i + j) \]
\[ \wedge (\forall i \cdot p i \Rightarrow p i + 1) \]
\[ \Rightarrow (\forall m \cdot p m) \]

**z_N_plus_thm**

\[ \vdash \forall i, j : \mathbb{N} \cdot i + j \in \mathbb{N} \]

**z_Z_eq_thm**

\[ \vdash \forall i, j : \mathbb{U} \cdot i = j \iff i + \sim j = 0 \]

**z_minus_thm**

\[ \vdash \forall i, j : \mathbb{U} \]
\[ \bullet \sim i = i \]
\[ \wedge i + \sim i = 0 \]
\[ \wedge \sim i + i = 0 \]
\[ \wedge (i + j) = \sim i + \sim j \]
\[ \wedge \sim 0 = 0 \]

**z_minus_clauses**

\[ \vdash \forall i : \mathbb{U} \]
\[ \bullet \sim i = i \wedge \sim 0 = 0 \wedge i + \sim i = 0 \wedge i + i = 0 \]

**z_N_cases_thm**

\[ \vdash \forall i : \mathbb{N} \cdot i = 0 \lor (\exists j : \mathbb{N} \cdot i = j + 1) \]

**z_N_¬_thm**

\[ \vdash \forall i : \mathbb{U} \cdot \neg i \in \mathbb{N} \Rightarrow \neg i \in \mathbb{N} \]

**z_Z_cases_thm**

\[ \vdash \forall i : \mathbb{U} \cdot \exists j : \mathbb{N} \cdot i = j \lor i = \sim j \]

**z_N_¬_plus1_thm**

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
\[ \forall i : \mathbb{N} \bullet \neg i + 1 = 0 \]

**z_\mathbb{Z}_\text{cases_thm1}**

\[ \forall i : \mathbb{U} \bullet i \in \mathbb{N} \lor (\exists j : \mathbb{N} \bullet i = \sim (j + 1)) \]

**z_\mathbb{N}_\sim_\text{minus_thm}**

\[ \forall i : \mathbb{N} \bullet i = 0 \lor \neg \sim i \in \mathbb{N} \]

**z_\text{plus_clauses}**

\[ \forall i, j, k : \mathbb{U} \]

\[ (i + k = j + k \leftrightarrow i = j) \]
\[ \land (k + i = j + k \leftrightarrow i = j) \]
\[ \land (i + k = k + j \leftrightarrow i = j) \]
\[ \land (k + i = k \leftrightarrow i = 0) \]
\[ \land (k + i = i \leftrightarrow i = 0) \]
\[ \land (k = k + j \leftrightarrow j = 0) \]
\[ \land (k = j + k \leftrightarrow j = 0) \]
\[ \land i + 0 = i \]
\[ \land 0 + i = i \]
\[ \land \neg \]
\[ 1 = 0 \]
\[ \land \neg \]
\[ 0 = 1 \]

**z_\text{times_comm_thm}**

\[ \forall i, j : \mathbb{U} \bullet i \ast j = j \ast i \]

**z_\text{times_assoc_thm}**

\[ \forall i, j, k : \mathbb{U} \bullet i \ast j \ast k = i \ast (j \ast k) \]

**z_\text{times_assoc_thm1}**

\[ \forall i, j, k : \mathbb{U} \bullet i \ast (j \ast k) = i \ast j \ast k \]

**z_\text{times_order_thm}**

\[ \forall i : \mathbb{U} \]

\[ \bullet \forall j, k : \mathbb{U} \]

\[ \bullet j \ast i = i \ast j \]
\[ \land i \ast j \ast k = i \ast (j \ast k) \]
\[ \land j \ast (i \ast k) = i \ast (j \ast k) \]

**z_\text{times1_thm}**

\[ \forall i : \mathbb{U} \bullet i \ast 1 = i \land 1 \ast i = i \]

**z_\text{times_plus_distrib_thm}**

\[ \forall i, j, k : \mathbb{U} \]

\[ \bullet i \ast (j + k) = i \ast j + i \ast k \]
\[ \land (i + j) \ast k = i \ast k + j \ast k \]

**z_\text{times0_thm}**

\[ \forall i : \mathbb{U} \bullet 0 \ast i = 0 \land i \ast 0 = 0 \]

**z_\text{minus_times_thm}**

\[ \forall i, j : \mathbb{U} \]

\[ \bullet \sim i \ast j = \sim (i \ast j) \]
\[ \land i \ast \sim j = \sim (i \ast j) \]
\[ \land \sim i \ast \sim j = i \ast j \]

**z_\mathbb{N}_\text{times_thm}**

\[ \forall i, j : \mathbb{N} \bullet i \ast j \in \mathbb{N} \]

**z_\text{times_eq_0_thm}**

\[ \forall i, j : \mathbb{U} \bullet i \ast j = 0 \leftrightarrow i = 0 \lor j = 0 \]

**z_\text{times_clauses}**

\[ \forall i, j : \mathbb{U} \]

\[ \bullet 0 \ast i = 0 \land i \ast 0 = 0 \land i \ast 1 = i \land 1 \ast i = i \]
z≤_trans_thm
\[ \forall i, j, k : U \mid i \leq j \land j \leq k \implies i \leq k \]

z_less_trans_thm
\[ \forall i, j, k : U \mid i < j \land j < k \implies i < k \]

z_less_le_trans_thm
\[ \forall i, j, k : U \mid i < j \land j \leq k \implies i < k \]

z<=_less_trans_thm
\[ \forall i, j, k : U \mid i \leq j \land j < k \implies i < k \]

z_minus_N<=_thm
\[ \forall i : U; j : N \cdot i + j \leq i \]

z<=_plus_N_thm
\[ \forall i : U; j : N \cdot i \leq i + j \]

z<=_cases_thm
\[ \forall i, j : U \cdot i \leq j \lor j \leq i \]

z<=_refl_thm
\[ \forall i : U \cdot i \leq i \]

z<=N_thm
\[ \forall i : U \cdot i \in N \iff 0 \leq i \]

z<=_0_thm
\[ \forall i, j : U \cdot i \leq j \iff i + j \leq 0 \]

z<=_antisym_thm
\[ \forall i, j : U \cdot i \leq j \iff j \leq i \]

z<=_antisym_thm
\[ \forall i, j : U \cdot i \leq j \iff j \leq i \]

z<=_clauses
\[ \forall i, j, k : U \]
\[ \cdot (i + k \leq j + k \iff i \leq j) \]
\[ \land (k + i \leq j + k \iff i \leq j) \]
\[ \land (i + k \leq k + j \iff i \leq j) \]
\[ \land (k + i \leq k + j \iff i \leq j) \]
\[ \land (i + k \leq k \iff i \leq 0) \]
\[ \land (k + i \leq k \iff i \leq 0) \]
\[ \land (i \leq k + i \iff 0 \leq k) \]
\[ \land (i \leq k + i \iff 0 \leq k) \]
\[ \land i \leq i \]
\[ \land \neg \]
\[ 1 \leq 0 \]
\[ \land 0 \leq 1 \]

z<=_clauses
\[ \forall i, j, k : U \]
\[ \cdot (i + k < j + k \iff i < j) \]
\[ \land (k + i < j + k \iff i < j) \]
\[ \land (i + k < k + j \iff i < j) \]
\[ \land (k + i < k + j \iff i < j) \]
\[ \land (i + k < k \iff i < 0) \]
\[ \land (k + i < k \iff i < 0) \]
\[ \land (i < k + i \iff 0 < k) \]
\[ \land (i < i + k \iff 0 < k) \]
\[ \land \neg \]
\[ i < i \]
\[ \land 0 < 1 \]
\[ \land \neg \]
\[ 1 < 0 \]

z<=_irrefl_thm
\[ \forall i, j : U \cdot \neg (i < j \land j < i) \]
\[
\begin{align*}
&z_{\text{abs thm}} \vdash \forall i : \mathbb{N} \cdot \text{abs } i = i \land \text{abs } \sim i = i \\
&z_{\text{abs minus thm}} \vdash \forall i : \mathbb{U} \cdot \text{abs } \sim i = \text{abs } i \\
&z_{\text{abs N thm}} \vdash \forall i : \mathbb{U} \cdot \text{abs } i \in \mathbb{N} \\
&z_{\text{abs plus thm}} \vdash \forall i, j : \mathbb{U} \cdot \text{abs } (i \ast j) = \text{abs } i \ast \text{abs } j \\
&z_{\text{abs eq 0 thm}} \vdash \forall i : \mathbb{U} \cdot \text{abs } i = 0 \iff i = 0 \\
&z_{\text{N abs minus thm}} \vdash \forall i, j : \mathbb{N} \mid j \leq i \cdot \text{abs } (i \sim j) \leq i \\
&z_{\leq \text{- induction thm}} \vdash \forall j \ p \hfill (\forall i \cdot \mathcal{E}(j, i) \in \mathcal{E}(\leq, \leq) \land p i \Rightarrow p (i + 1)) \\
&\quad \Rightarrow (\forall m \cdot \mathcal{E}(j, m) \in \mathcal{E}(\leq, \leq) \Rightarrow p m) \\
&z_{\text{less plus 1 thm}} \vdash \forall m, n : \mathbb{U} \cdot m < n + 1 \iff m = n \lor m < n \\
&z_{\text{cov induction thm}} \vdash \forall j \ p \hfill (\forall i \cdot \mathcal{E}(j, i) \in \mathcal{E}(\leq, \leq)) \\
&\quad \land \forall k : \mathbb{Z} \cdot j \leq k \land k < i \Rightarrow p k \Rightarrow p i \\
&\quad \Rightarrow (\forall i \cdot \mathcal{E}(j, i) \in \mathcal{E}(\leq, \leq) \Rightarrow p i) \\
&z_{\text{div mod unique thm}} \vdash \forall i, j, d, r : \mathbb{U} \hfill | \n \n j = 0 \\
&\quad \Rightarrow i = d \ast j + r \land 0 \leq r \land r < \text{abs } j \\
&\quad \iff d = i \text{ div } j \land r = i \text{ mod } j \\
&z_{\leq \text{- less eq thm}} \vdash \forall x, y : \mathbb{U} \cdot x \leq y \iff x < y \lor x = y \\
&z_{\in \text{- N1 thm}} \vdash \forall x : \mathbb{U} \cdot x \in \mathbb{N} \iff 0 < x \\
&z_{\text{F thm}} \vdash \forall X : \mathbb{U} \hfill | \n \n F X \\
&\quad = \{ S : \mathbb{P} X \mid \exists n : \mathbb{N} \cdot \exists f : 1 .. n \rightarrow S \cdot \text{ran } f = S \} \\
&z_{\text{F1 thm}} \vdash \forall X : \mathbb{U} \cdot F_1 X = F X \setminus \{ \emptyset \} \\
&z_{\text{F empty thm}} \vdash F \{ \} = \mathbb{P} \{ \} 
\end{align*}
\]
9.1.9 The Z Theory $z\_\text{numbers}1$

9.1.9.1 Parents

$z\_\text{arithmetic\_tools} \quad z\_\text{numbers} \quad z\_\text{functions}1$

9.1.9.2 Children

$z\_\text{sequences}1$

9.1.9.3 Theorems

$z\_\text{dot\_dot\_clauses}$

$\vdash \forall i, i1, i2, j1, j2 : \mathbb{U}$

- $(i \in i1 .. i2 \iff i1 \leq i \land i \leq i2)$
- $(i1 .. i2 = \emptyset \iff i2 < i1)$
- $(i1 .. i2 \subseteq j1 .. j2 \iff i2 < i1 \lor j1 \leq i1 \land i2 \leq j2)$

$z\_\text{dot\_dot\_plus\_thm}$

$\vdash \forall n, i1, i2 : \mathbb{U}$

- $(\{ i : i1 .. i2 \cdot i + n \} = i1 + n .. i2 + n)$

$z\_\text{less\_cases\_thm}$

$\vdash \forall i, j : \mathbb{U} \cdot i < j \lor i = j \lor j < i$

$z\_\text{\leq\_plus\_thm}$

$\vdash \forall i, j : \mathbb{U} \cdot i \leq j \land j \leq i + 1 \iff j = i \lor j = i + 1$

$z\_\text{dot\_dot\_diff\_thm}$

$\vdash \forall i : \mathbb{N} \cdot (1 .. i + 1) \setminus \{ i + 1 \} = 1 .. i$

$z\_\text{dot\_dot\_or\_thm}$

$\vdash \forall i : \mathbb{N} \cdot (1 .. i) \cup \{ i + 1 \} = 1 .. i + 1$

$z\_\text{dot\_dot\_and\_thm}$

$\vdash \forall i : \mathbb{N} \cdot (1 .. i) \cap \{ i + 1 \} = \{ \}

$z\_\text{empty\_\_F\_thm}$

$\vdash [X](\{\} \in \mathbb{F} X)$

$z\_\text{\_\_F\_or\_singleton\_thm}$

$\vdash [X](\forall x : X ; a : \mathbb{F} X \cdot a \cup \{ x \} \in \mathbb{F} X)$

$z\_\text{F\_thm1}$

$\vdash [X](\mathbb{F} X = \bigcap\{ u : \mathbb{P} \mathbb{P} X \mid \{ \} \in u \land (\forall x : X ; a : u \cdot a \cup \{ x \} \in u)\})$

$z\_\text{F\_induction\_thm}$

$\vdash \forall X \ P$

- $(p \ f\{\}\land$
  - $(\forall x \ a$
  - $p a \land a \in \mathbb{F} X \land x \in X \land \neg x \in a$
  - $\Rightarrow p \ f\{ a \cup \{ x \}\})$
- $\Rightarrow (\forall a \cdot a \in \mathbb{F} X \land p a)$

$z\_\text{F\_F\_thm}$

$\vdash [X](\mathbb{F} X = \mathbb{P} X \cap (\mathbb{F} _))$

$z\_\text{F\_size\_thm}$

$\vdash \forall A : \mathbb{U} ; f : \mathbb{U} ; n : \mathbb{N}$

- $(f \in I .. n \Rightarrow A$
  - $A \in (\mathbb{F} _) \land \# A = n$

$z\_\text{size\_empty\_thm}$
Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030

470 Chapter 9. THEORIES

\[ \{\} \in (F) \land \# \{\} = 0 \]

\[ z \_size \_singleton \_thm \]
\[ \vdash \forall x : U \bullet \{x\} \in (F) \land \# \{x\} = 1 \]

\[ z \_size \_dot \_dot \_thm \]
\[ \vdash \forall n : N \bullet 1 .. n \in (F) \land \# (1 .. n) = n \]

\[ z \_size \_\rightarrow \_thm \]
\[ \vdash X : U ; Y : U ; f : U \]
\[ | f \in X \rightarrow Y \]
\[ \bullet f \in (F) \land \# f = \# (dom f) \]

\[ z \_size \_seq \_thm \]
\[ \vdash X : U ; f : U ; n : N \mid f \in 1 .. n \rightarrow X \bullet \# f = n \]

\[ z \_size \_\cup \_singleton \_thm \]
\[ \vdash \forall a : (F) \bullet x : U \mid \lnot x \in a \bullet \# (a \cup \{x\}) = \# a + 1 \]

\[ z \_F \_\cap \_thm \]
\[ \vdash \forall a, b : U \mid a \in (F) \lor b \in (F) \bullet a \cap b \in (F) \]

\[ z \_size \_\subseteq \_thm \]
\[ \vdash \forall a : (F) \bullet a \setminus b \in (F) \land \# (a \setminus b) + \# (a \cap b) = \# a \]

\[ z \_size \_\uin \_thm \]
\[ \vdash \forall a : (F) \bullet \# a \in N \]

\[ z \_\_ mono \_thm \]
\[ \vdash \forall a : (F) \bullet \exists f : 1 .. a \Rightarrow a \bullet true \]

\[ z \_size \_\cup \_\leq \_thm \]
\[ \vdash \forall a, b : (F) \bullet b : U \mid b \subseteq a \bullet \# b \leq \# a \]

\[ z \_size \_eq \_thm \]
\[ \vdash \forall a, b : (F) \bullet \# (a \cup b) \leq \# a + \# b \]

\[ z \_size \_0 \_thm \]
\[ \vdash \forall a : (F) \bullet \# a = 0 \Rightarrow a = \{\} \]

\[ z \_size \_1 \_thm \]
\[ \vdash \forall a : (F) \bullet \# a = 1 \Rightarrow (\exists x : U \bullet a = \{x\}) \]

\[ z \_size \_pair \_thm \]
\[ \vdash \forall x, y : U \mid \lnot x = y \bullet \{x, y\} \in (F) \land \# \{x, y\} = 2 \]

\[ z \_size \_2 \_thm \]
\[ \vdash \forall a : (F) \bullet \# a = 2 \Leftrightarrow (\exists x, y : U \bullet \lnot x = y \land a = \{x, y\}) \]

\[ z \_size \_\times \_thm \]
\[ \vdash \forall a : (F) ; b : (F) \bullet a \times b \in (F) \land \# (a \times b) = \# a \ast \# b \]

\[ z \_size \_\leq \_1 \_thm \]
\[ \vdash \forall a : (F) \mid \# a \leq 1 \bullet a = \{\} \lor (\exists x : U \bullet a = \{x\}) \]

\[ z \_size \_dot \_dot \_thm \]
\[ \vdash \forall i, j : Z \]
\[ \bullet i .. j \in (F) \]
\[ \land (i \leq j \Rightarrow \# (i .. j) = j + \sim i + 1) \]
\[ \land (j < i \Rightarrow \# (i .. j) = 0) \]

\[ z \_pigeon \_hole \_thm \]
\[ \vdash \forall u : F (F) \mid \# (\cup u) > \# u \bullet \exists a : u \bullet \# a > 1 \]

\[ z \_div \_thm \]
\[ \vdash \forall i, j, k : Z \]
\[ | \neg \]
\[ j = 0 \]
\[ \bullet i \div j = k \]

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
\( \Leftrightarrow (\exists m : \mathbb{Z} \cdot i = k \cdot j + m \land 0 \leq m \land m < \text{abs } j) \)

\[ z_{\text{mod_thm}} \vdash \forall i, j, k : \mathbb{Z} \]
\[ \begin{array}{c}
  \neg j = 0 \\
  \cdot i \mod j = k
\end{array} \]
\( \Leftrightarrow (\exists d : \mathbb{Z} \cdot i = d \cdot j + k \land 0 \leq k \land k < \text{abs } j) \)

\[ z_{\text{abs_pos_thm}} \quad \forall i : \mathbb{Z} \mid 0 < i \bullet \text{abs } i = i \land \text{abs } \sim i = i \]

\[ z_{\text{abs_neg_thm}} \quad \forall i : \mathbb{Z} \mid i < 0 \bullet \text{abs } i = \sim i \land \text{abs } \sim i = \sim i \]

\[ z_{\text{abs_\leq_times_thm}} \quad \forall i, j : \mathbb{Z} \mid \neg i = 0 \land \neg j = 0 \bullet \text{abs } j \leq \text{abs } (i \cdot j) \]

\[ z_{\text{abs_0_less_thm}} \quad \forall i : \mathbb{Z} \mid \neg i = 0 \bullet 0 < \text{abs } i \]

\[ z_{\text{0_less_times_thm}} \quad \forall i, j : \mathbb{Z} \]
\[ \cdot 0 < i \cdot j \Leftrightarrow 0 < i \land 0 < j \lor i < 0 \land j < 0 \]

\[ z_{\text{times_less_0_thm}} \quad \forall i, j : \mathbb{Z} \]
\[ \cdot i \cdot j < 0 \Leftrightarrow 0 < i \land j < 0 \lor i < 0 \land j < 0 \]

\[ z_{\in_{\text{succ_thm}}} \quad \forall i, j \]
\[ \cdot z(i, j)^\gamma \in z^{\text{succ}} \]
\[ \Leftrightarrow z(0, i)^\gamma \in z^{(\leq)} \gamma \land j = z^i + 1 \gamma \]

\[ z_{\text{succ^0_thm}} \quad \text{succ}^0 = \text{id } \mathbb{Z} \]

\[ z_{\text{succ^n_thm}} \quad \forall n : \mathbb{Z} \mid 1 \leq n \bullet \text{succ }^n = \{ m : \mathbb{N} \bullet m \mapsto m + n \} \]

\[ z_{\text{succ^\sim_n_thm}} \quad \forall n : \mathbb{N} \mid 1 \leq n \bullet \text{succ }^{\sim n} = \{ m : \mathbb{N} \bullet m + n \mapsto m \} \]
9.1.10  The Z Theory \( z \)-reals

9.1.10.1  Parents

\[
\mathbb{R} \quad z\text{-numbers}
\]

9.1.10.2  Global Variables

\[
\begin{align*}
\mathbb{R} & \quad P \; \mathbb{R} \\
(\text{abs}_{\mathbb{R}} ) & \quad \mathbb{R} \leftrightarrow \mathbb{R} \\
(\_ \; /_{\mathbb{R}} \; ) & \quad \mathbb{R} \times \mathbb{R} \leftrightarrow \mathbb{R} \\
(\_ \; *_{\mathbb{R}} \; ) & \quad \mathbb{R} \times \mathbb{R} \leftrightarrow \mathbb{R} \\
(\_ \; +_{\mathbb{R}} \; ) & \quad \mathbb{R} \times \mathbb{R} \leftrightarrow \mathbb{R} \\
(\sim_{\mathbb{R}} ) & \quad \mathbb{R} \leftrightarrow \mathbb{R} \\
(\_ \; \leq_{\mathbb{R}} \; ) & \quad \mathbb{R} \leftrightarrow \mathbb{R} \\
(\_ \; <_{\mathbb{R}} \; ) & \quad \mathbb{R} \leftrightarrow \mathbb{R} \\
(\_ \; \geq_{\mathbb{R}} \; ) & \quad \mathbb{R} \leftrightarrow \mathbb{R} \\
(\_ \; >_{\mathbb{R}} \; ) & \quad \mathbb{R} \leftrightarrow \mathbb{R} \\
\text{real} & \quad Z \leftrightarrow \mathbb{R} \\
(\_ \; /_{Z} \; ) & \quad Z \times Z \leftrightarrow \mathbb{R} \\
(\_ \; \wedge_{Z} \; ) & \quad \mathbb{R} \times Z \leftrightarrow \mathbb{R} \\
(\_ \; \cdot_{\mathbb{R}} \; ) & \quad \mathbb{R} \times \mathbb{R} \leftrightarrow P \; \mathbb{R} \\
(\_ \; \text{lb}_{\mathbb{R}} \; ) & \quad \mathbb{R} \leftrightarrow P \; \mathbb{R} \\
\text{glb}_{\mathbb{R}} & \quad P \; \mathbb{R} \leftrightarrow \mathbb{R} \\
(\_ \; \text{ub}_{\mathbb{R}} \; ) & \quad \mathbb{R} \leftrightarrow P \; \mathbb{R} \\
\text{lub}_{\mathbb{R}} & \quad P \; \mathbb{R} \leftrightarrow \mathbb{R} \\
\text{Z/Float} & \quad U
\end{align*}
\]

9.1.10.3  Fixity

fun 20 leftassoc

\[
(\; \cdot_{\mathbb{R}} \; )
\]

fun 30 leftassoc

\[
(\_ \; +_{\mathbb{R}} \; ) \quad (\_ \; -_{\mathbb{R}} \; )
\]

fun 40 leftassoc

\[
(\_ \; *_{\mathbb{R}} \; ) \quad (\_ \; /_{\mathbb{R}} \; ) \quad (\_ \; /_{Z} \; )
\]

fun 50 rightassoc

\[
(\text{abs}_{\mathbb{R}} \; ) \quad (\sim_{\mathbb{R}} \; )
\]

fun 60 rightassoc

\[
(\_ \; \wedge_{Z} \; )
\]

rel

\[
(\_ \; \text{lb}_{\mathbb{R}} \; ) \quad (\_ \; <_{\mathbb{R}} \; ) \quad (\_ \; \leq_{\mathbb{R}} \; ) \quad (\_ \; \geq_{\mathbb{R}} \; ) \quad (\_ \; >_{\mathbb{R}} \; )
\]
9.1.10.4 Axioms

abs$_R$
- \( /R \)
- \( *R \)
- \( +R \)
\( \sim_R \)
- \( \leq_R \)
- \( <_R \)
\[ \vdash ((<_R \cdot) \in R \leftrightarrow R) \]
\[ \land ((\leq_R \cdot) \in R \leftrightarrow R) \]
\[ \land (\sim_R \cdot) \in R \rightarrow R \]
\[ \land ((+_R \cdot) \in R \times R \rightarrow R) \]
\[ \land ((\cdot_R \cdot) \in R \times R \rightarrow R) \]
\[ \land ((/R \cdot) \in R \times R \rightarrow R) \]
\[ \land (abs_R \cdot) \in R \rightarrow R \]
\[ \land (\forall x, y : R \cdot x <_R y \iff \lnot x < y) \]
\[ \land (\forall x, y : R \cdot x \leq_R y \iff \lnot x < y) \]
\[ \land (\forall x : R \cdot \sim_R x = \lnot \sim x) \]
\[ \land (\forall x, y : R \cdot x +_R y = \lnot x + y) \]
\[ \land (\forall x, y : R \cdot x *_R y = \lnot x * y) \]
\[ \land (\forall x, y : R \cdot x /_R y = \lnot x / y) \]
\[ \land (\forall x : R \cdot abs_R x = \lnot Abs x) \]

- \( \neg_R \)
- \( \geq_R \)
- \( >_R \)
\[ \vdash ((>_R \cdot) \in R \leftrightarrow R) \]
\[ \land ((\geq_R \cdot) \in R \leftrightarrow R) \]
\[ \land ((\neg_R \cdot) \in R \times R \rightarrow R) \]
\[ \land (\forall x, y : R \cdot x >_R y \iff y <_R x) \]
\[ \land (\forall x, y : R \cdot x \geq_R y \iff y \leq_R x) \]
\[ \land (\forall x, y : R \cdot x -_R y = x +_R \neg_R y) \]

real
\[ \vdash \text{real} \in Z \rightarrow R \]
\[ \land (\text{real} 1 = \lnot 1.\) \]
\[ (\forall i : Z \cdot \text{real} (\lnot i) = \sim_R \text{real} i) \]
\[ (\forall i, j : Z \cdot \text{real} (i + j) = \text{real} i +_R \text{real} j) \]

- \( /Z \)
\[ \vdash ((/_Z \cdot) \in Z \times Z \rightarrow R \]
\[ (\forall i, j : Z \cdot i /_Z j = \text{real} i /_R \text{real} j) \]

- \( \hat{Z} \)
\[ \vdash ((\hat{Z} \cdot) \in R \times Z \rightarrow R \]
\[ (\forall x \cdot \hat{Z} x \sim Z \sim R \text{Int} m \gamma = x \sim m \gamma) \]
\[ (\forall x \cdot m \bullet \hat{Z} x \sim Z (\sim R \text{Int} (m + 1) \gamma) = 1. \sim x \sim (m + 1) \gamma) \]

- \( \neg_R \)
\[ \vdash ((\neg_R \cdot) \in R \times R \rightarrow P\ R \]
\[ (\forall x, y : R \cdot \bullet x \neg_R y = \{ t : R \mid x \leq_R t \land t \leq_R y \}) \]

- \( lb_R \)
\[ \vdash ((lb_R \cdot) \in R \rightarrow P\ R \]
\[ (\forall r : R, sr : P\ R \cdot r lb_R sr \iff (\forall x : sr \cdot r \leq_R x) \]

- \( glb_R \)
\[ \vdash (glb_R \in P\ R \rightarrow R \]
\[ (\forall sr : P\ R, glb : R \cdot sr \rightarrow glb \in glb_R \]
\[ (\forall lb : R \cdot lb lb_R sr) \]
\[ (\forall lb : R \mid lb lb_R sr \cdot \bullet lb \leq_R glb) \]

- \( ub_R \)
\[ \vdash ((ub_R \cdot) \in R \leftrightarrow P\ R \]

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
\( \land (\forall r : \mathbb{R}; sr : \mathbb{P} \mathbb{R} \cdot r \uparrow_R sr \iff (\forall x : sr \cdot r \geq_R x)) \)

\( \uparrow \) lub\(_R\) \( \vdash \) lub\(_R\) \( \in \mathbb{P} \mathbb{R} \) \( \mapsto \) lub\(_R\) & (\( \forall sr : \mathbb{P} \mathbb{R}; \) lub : \( \mathbb{R} \)

\( \cdot \) lub \( \in \) lub\(_R\) \( \iff \) lub \( \uparrow\) ub\(_R\) sr

\( \land (\forall \) ub : \( \mathbb{R} \) \( \mid \) ub \( \uparrow\) ub\(_R\) sr \( \cdot \) ub \( \geq_R \) lub\(_R\))

### 9.1.10.5 Definitions

\( \mathbb{R} \) \( \vdash \) \( \mathbb{R} = \mathbb{U} \)

\( \mathbb{Z}'\mathbb{F}l\) \( \vdash \) \( \forall \) m p e

\( \cdot \) \( \lceil \) \( \forall \) Z'\( \mathbb{F}l\) \( m p e \rceil \)

\( = \) \( \llbracket \) \( \exists \) real \( m \) \( \ast \) \( \mathbb{R} \) \( \ast \) real \( 10 \)

\( \sim \) \( e \) \( \rceil \)

### 9.1.10.6 Theorems

\( z_\mathbb{R}\) _unbounded_below_thm_ \( \vdash \) \( \forall x : \mathbb{R} \cdot \exists y : \mathbb{R} \cdot y <_R x \)

\( z_\mathbb{R}\) _unbounded_above_thm_ \( \vdash \) \( \forall x : \mathbb{R} \cdot \exists y : \mathbb{R} \cdot x <_R y \)

\( z_\mathbb{R}\) _less_irrefl_thm_ \( \vdash \) \( \forall x : \mathbb{R} \cdot \neg x <_R x \)

\( z_\mathbb{R}\) _less_antisym_thm_ \( \vdash \) \( \forall x, y : \mathbb{R} \cdot \neg (x <_R y \land y <_R x) \)

\( z_\mathbb{R}\) _less_trans_thm_ \( \vdash \) \( \forall x, y, z : \mathbb{R} \cdot x <_R y \land y <_R z \Rightarrow x <_R z \)

\( z_\mathbb{R}\) _less_cases_thm_ \( \vdash \) \( \forall x, y : \mathbb{R} \cdot x <_R y \lor x = y \lor y <_R x \)

\( z_\mathbb{R}\) _less_cases_thm_ \( \vdash \) \( \forall x, y : \mathbb{R} \cdot x <_R y \lor y <_R x \)

\( z_\mathbb{R}\) _less_trans_thm_ \( \vdash \) \( \forall x, y : \mathbb{R} \cdot x \leq_R y \lor y <_R x \)

\( z_\mathbb{R}\) _eq_cases_thm_ \( \vdash \) \( \forall x, y : \mathbb{R} \cdot x = y \iff x \leq_R y \land y \leq_R x \)

\( z_\mathbb{R}\) _less_antisym_thm_ \( \vdash \) \( \forall x, y : \mathbb{R} \cdot x \leq_R y \land y \leq_R x \Rightarrow x = y \)

\( z_\mathbb{R}\) _less_trans_thm_ \( \vdash \) \( \forall x, y, z : \mathbb{R} \cdot x <_R y \land y \leq_R z \Rightarrow x <_R z \)

\( z_\mathbb{R}\) _less_trans_thm_ \( \vdash \) \( \forall x, y, z : \mathbb{R} \cdot x <_R y \land y <_R z \Rightarrow x <_R z \)

\( z_\mathbb{R}\) _less_refl_thm_ \( \vdash \) \( \forall x, y : \mathbb{R} \cdot x \leq_R y \land y \leq_R x \Rightarrow x = y \)

\( z_\mathbb{R}\) _less_trans_thm_ \( \vdash \) \( \forall x, y, z : \mathbb{R} \cdot x \leq_R y \land y \leq_R z \Rightarrow x \leq_R z \)

\( z_\mathbb{R}\) _less_not_less_thm_ \( \vdash \) \( \forall x, y : \mathbb{R} \cdot x \leq_R y \iff \neg y <_R x \)

\( z_\mathbb{R}\) _less_not_less_thm_ \( \vdash \) \( \forall x, y : \mathbb{R} \cdot \neg x \leq_R y \iff y <_R x \)

\( z_\mathbb{R}\) _less_not_eq_thm_ \( \vdash \) \( \forall x, y : \mathbb{R} \cdot x <_R y \iff \neg x = y \)
9.1. Theory Listings

\[ \vdash \forall x, y : \mathbb{R} \cdot x <_R y \Rightarrow (\exists z : \mathbb{R} \cdot x <_R z \land z <_R y) \]

\text{z.R_complete_thm}

\[ \vdash \forall A : \mathbb{P} \mathbb{R} \]
\[ \quad \cdot \neg A = \{\} \land (\exists b : \mathbb{R} \cdot \forall x : \mathbb{R} \cdot x \in A \Rightarrow x \leq_R b) \]
\[ \quad \Rightarrow (\exists s : \mathbb{R} \]
\[ \quad \quad \cdot (\forall x : \mathbb{R} \cdot x \in A \Rightarrow x \leq_R s) \]
\[ \quad \quad \land (\forall b : \mathbb{R} \]
\[ \quad \quad \quad \cdot (\forall x : \mathbb{R} \cdot x \in A \Rightarrow x \leq_R b) \Rightarrow s \leq_R b) \]

\text{z.R_plus_assoc_thm}

\[ \vdash \forall x, y, z : \mathbb{R} \cdot x +_R y +_R z = x +_R (y +_R z) \]

\text{z.R_plus_assoc_thm1}

\[ \vdash \forall x, y, z : \mathbb{R} \cdot x +_R (y +_R z) = x +_R y +_R z \]

\text{z.R_plus_comm_thm}

\[ \vdash \forall x, y : \mathbb{R} \cdot x +_R y = y +_R x \]

\text{z.R_plus_unit_thm}

\[ \vdash \forall x : \mathbb{R} \cdot x +_R \text{real } 0 = x \]

\text{z.R_plus_mono_thm}

\[ \vdash \forall x, y, z : \mathbb{R} \cdot y <_R z \Rightarrow x +_R y <_R x +_R z \]

\text{z.R_plus_mono_thm1}

\[ \vdash \forall x, y, z : \mathbb{R} \cdot y <_R z \Rightarrow y +_R x <_R z +_R x \]

\text{z.R_plus_mono_thm2}

\[ \vdash \forall x, y, z : \mathbb{R} \cdot y <_R z \Rightarrow x +_R y <_R z +_R x \]

\text{z.R_plus_0_thm}

\[ \vdash \forall x : \mathbb{R} \cdot x +_R \text{real } 0 = x \land \text{real } 0 +_R x = x \]

\text{z.R_plus_order_thm}

\[ \vdash \forall x, y, z : \mathbb{R} \]
\[ \quad \cdot y +_R x = x +_R y \]
\[ \quad \land x +_R y +_R z = x +_R (y +_R z) \]
\[ \quad \land y +_R (x +_R z) = x +_R (y +_R z) \]

\text{z.R_plus_minus_thm}

\[ \vdash \forall x : \mathbb{R} \cdot x +_R \sim_R x = \text{real } 0 \land \sim_R x +_R x = \text{real } 0 \]

\text{z.R_eq_thm}

\[ \vdash \forall x, y : \mathbb{R} \cdot x = y \iff x +_R \sim_R y = \text{real } 0 \]

\text{z.R_minus_clauses}

\[ \vdash \forall x, y : \mathbb{R} \]
\[ \quad \cdot \sim_R \sim_R x = x \]
\[ \quad \land x +_R \sim_R x = \text{real } 0 \]
\[ \quad \land \sim_R x +_R x = \text{real } 0 \]
\[ \quad \land \sim_R (x +_R y) = \sim_R x +_R \sim_R y \]
\[ \quad \land \sim_R \text{real } 0 = \text{real } 0 \]

\text{z.R_minus_eq_thm}

\[ \vdash \forall x, y : \mathbb{R} \cdot \sim_R x = \sim_R y \iff x = y \]

\text{z.R_plus_clauses}

\[ \vdash \forall x, y, z : \mathbb{R} \]
\[ \quad \cdot (x +_R z = y +_R z \iff x = y) \]
\[ \quad \land (z +_R x = y +_R z \iff x = y) \]
\[ \quad \land (x +_R z = z +_R y \iff x = y) \]
\[ \quad \land (x +_R z = z \iff x = \text{real } 0) \]
\[ \quad \land (z +_R x = z \iff x = \text{real } 0) \]
\[ \quad \land (z +_R y \iff y = \text{real } 0) \]
\( \therefore (z = y +_R z \iff y = \text{real } 0) \)
\( \therefore x +_R \text{real } 0 = x \)
\( \therefore \text{real } 0 +_R x = x \)
\( \therefore \)
\( \therefore \text{real } 1 = \text{real } 0 \)
\( \therefore \)
\( \therefore \text{real } 0 = \text{real } 1 \)

\textit{z}\textsubscript{\(\mathbb{R}\)}\textunderscore less\textunderscore clauses
\[
\vdash \forall x, y, z : \mathbb{R} \\
\therefore (x +_R z <_R y +_R z \iff x <_R y) \\
\therefore (z +_R x <_R y +_R z \iff x <_R y) \\
\therefore (x +_R z <_R z +_R y \iff x <_R y) \\
\therefore (z +_R x <_R z +_R y \iff x <_R y) \\
\therefore (x +_R z <_R z \iff x \leq_R \text{real } 0) \\
\therefore (z +_R x <_R z \iff x \leq_R \text{real } 0) \\
\therefore (x <_R z +_R x \iff \text{real } 0 \leq_R z) \\
\therefore (x \leq_R z +_R x \iff \text{real } 0 \leq_R z) \\
\therefore x \leq_R x \\
\therefore \text{real } 0 \leq_R \text{real } 1 \\
\therefore \)
\( \therefore \text{real } 1 \leq_R \text{real } 0 \)

\textit{z}\textsubscript{\(\mathbb{R}\)}\textunderscore \leq\textunderscore clauses
\[
\vdash \forall x, y, z : \mathbb{R} \\
\therefore (x +_R z \leq_R y +_R z \iff x \leq_R y) \\
\therefore (z +_R x \leq_R y +_R z \iff x \leq_R y) \\
\therefore (x +_R z \leq_R z +_R y \iff x \leq_R y) \\
\therefore (z +_R x \leq_R z +_R y \iff x \leq_R y) \\
\therefore (x +_R z \leq_R z \iff x \leq_R \text{real } 0) \\
\therefore (z +_R x \leq_R z \iff x \leq_R \text{real } 0) \\
\therefore (x \leq_R z +_R x \iff \text{real } 0 \leq_R z) \\
\therefore (x \leq_R z +_R x \iff \text{real } 0 \leq_R z) \\
\therefore x \leq_R x \\
\therefore \text{real } 0 \leq_R \text{real } 1 \\
\therefore \)
\( \therefore \text{real } 1 \leq_R \text{real } 0 \)

\textit{z}\textsubscript{\(\mathbb{R}\)}\textunderscore times\textunderscore assoc\textunderscore thm
\[
\vdash \forall x, y, z : \mathbb{R} \\
\therefore x *_R y *_R z = x *_R (y *_R z) \\
\]

\textit{z}\textsubscript{\(\mathbb{R}\)}\textunderscore times\textunderscore comm\textunderscore thm
\[
\vdash \forall x, y : \mathbb{R} \\
\therefore x *_R y = y *_R x \\
\]

\textit{z}\textsubscript{\(\mathbb{R}\)}\textunderscore times\textunderscore unit\textunderscore thm
\[
\vdash \forall x : \mathbb{R} \\
\therefore x *_R \text{real } 1 = x \\
\]

\textit{z}\textsubscript{\(\mathbb{R}\)}\textunderscore 0\textunderscore less\textunderscore 0\textunderscore less\textunderscore times\textunderscore thm
\[
\vdash \forall x, y : \mathbb{R} \\
\therefore \text{real } 0 <_R x \land \text{real } 0 <_R y \Rightarrow \text{real } 0 <_R x *_R y \\
\]

\textit{z}\textsubscript{\(\mathbb{R}\)}\textunderscore times\textunderscore assoc\textunderscore thm1
\[
\vdash \forall x, y, z : \mathbb{R} \\
\therefore x *_R (y *_R z) = x *_R y *_R z \\
\]

\textit{z}\textsubscript{\(\mathbb{R}\)}\textunderscore times\textunderscore plus\textunderscore distrib\textunderscore thm
\[
\vdash \forall x, y, z : \mathbb{R} \\
\therefore x *_R (y +_R z) = x *_R y +_R x *_R z \\
\therefore (x +_R y) *_R z = x *_R z +_R y *_R z \\
\]
9.1. Theory Listings

\[ \begin{align*}
\text{z} & \times \text{order_thm} \\
& \vdash \forall x, y, z : \mathbb{R} \\
& \quad \bullet y \times_R x = x \times_R y \\
& \quad \land x \times_R y \times_R z = x \times_R (y \times_R z) \\
& \quad \land y \times_R (x \times_R z) = x \times_R (y \times_R z) \\
\text{z} & \times \text{clauses} \\
& \vdash \forall x : \mathbb{R} \\
& \quad \bullet \text{real 0} \times_R x = \text{real 0} \\
& \quad \land x \times_R \text{real 0} = \text{real 0} \\
& \quad \land x \times_R \text{real 1} = x \\
& \quad \land \text{real 1} \times_R x = x \\
\text{z} & \times \text{over_clauses} \\
& \vdash \forall y, z : \mathbb{R} \implies z = \text{real 0} \Rightarrow y \times_R z / R z = y \\
& \quad \land (\forall x, y, z : \mathbb{R} \\
& \quad \bullet \neg z = \text{real 0} \Rightarrow x \times_R y / R z = x \times_R (y / R z) \\
\text{z} & \times \text{float_thm} \\
& \vdash \forall m, p, e : \mathbb{Z} \\
& \quad \bullet \text{ceil} (Z! \text{Float} m p e) \\
& \quad = \text{real m} \times_R \text{real 10} \times_Z (e + \sim p) \\
\end{align*} \]
9.1.11 The Z Theory z_relations

9.1.11.1 Parents

\textit{z_sets}

9.1.11.2 Children

\textit{z_functions}

9.1.11.3 Global Variables

\begin{align*}
(\_ & \mapsto \_)[X, Y] & \quad X \times Y \leftrightarrow X \times Y \\
\text{ran}[X, Y] & \quad (X \leftrightarrow Y) \leftrightarrow \mathbb{P} Y \\
\text{dom}[X, Y] & \quad (X \leftrightarrow Y) \leftrightarrow \mathbb{P} X \\
id X & \quad X \leftrightarrow X \\
(\_ \circ \_)[X, Y, Z] & \quad (Y \leftrightarrow Z) \times (X \leftrightarrow Y) \leftrightarrow X \leftrightarrow Z \\
(\_ \odot \_)[X, Y, Z] & \quad (X \leftrightarrow Y) \times (Y \leftrightarrow Z) \leftrightarrow X \leftrightarrow Z \\
(\_ \triangleright \_)[X, Y] & \quad (X \leftrightarrow Y) \times \mathbb{P} Y \leftrightarrow X \leftrightarrow Y \\
(\_ \triangleleft \_)[X, Y] & \quad \mathbb{P} X \times (X \leftrightarrow Y) \leftrightarrow X \leftrightarrow Y \\
(\_ \trianglerighteq \_)[X, Y] & \quad (X \leftrightarrow Y) \times \mathbb{P} Y \leftrightarrow X \leftrightarrow Y \\
(\_ \trianglerightlefteq \_)[X, Y] & \quad \mathbb{P} X \times (X \leftrightarrow Y) \leftrightarrow X \leftrightarrow Y \\
(\_ \lnot \_)[X, Y] & \quad (X \leftrightarrow Y) \leftrightarrow Y \leftrightarrow X \\
(\_ \lnot \_)[X, Y] & \quad (X \leftrightarrow Y) \times \mathbb{P} X \leftrightarrow \mathbb{P} Y \\
(\_ \rightarrow \_)[X] & \quad (X \leftrightarrow X) \leftrightarrow X \leftrightarrow X \\
(\_ \times \_)[X] & \quad (X \leftrightarrow X) \leftrightarrow X \leftrightarrow X \\
(\_ \oplus \_)[X, Y] & \quad (X \leftrightarrow Y) \times (X \leftrightarrow Y) \leftrightarrow X \leftrightarrow Y
\end{align*}

9.1.11.4 Fixity

fun 10 leftassoc

(\_ \mapsto \_)

fun 40 leftassoc

(\_ \circ \_) \quad (\_ \odot \_)

fun 50 leftassoc

(\_ \oplus \_)

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
9.1. Theory Listings

fun 60 leftassoc
( _) (∘ _) (∘ _)

fun 65 rightassoc
( _) (∘ _) (∘ _)

fun 70 rightassoc
( _ | _ ) ( _ ) (∘ _ ) (∘ _ )

gen 70 rightassoc
(id _)

9.1.11.5 Axioms

⊢ [X, Y][(_ → _)[X, Y] ∈ X × Y → X × Y ∧ (∀ x : X; y : Y • (_ → _)[X, Y] (x, y) = (x, y))]

ran dom
⊢ [X, Y][(_ ◦ _)[X, Y] ∈ (X ↔ Y) → P X ∧ ran[X, Y] [X ↔ Y] → P Y] ∧ (∀ R : X ↔ Y • dom[X, Y] R = {x : X; y : Y | x ↔ y ∈ R • x} ∧ ran[X, Y] R} = {x : X; y : Y | x ↔ y ∈ R • y})]

― o _
― ◦ _
⊢ [X, Y, Z][(_ ◦ _)[X, Y, Z] ∈ (X ↔ Y) × (Y ↔ Z) → X ↔ Z ∧ (∀ R : X ↔ Y; S : Y ↔ Z • (_ ◦ _)[X, Y, Z] (R, S) = (_ ◦ _)[X, Y, Z] (S, R) ∧ (_ ◦ _)[X, Y, Z] (S, R) = {x : X; y : Y; z : Z | x ↔ y ∈ R ∧ y ↔ z ∈ S • x ↔ z})]

― ▽ _
― ▽ _
⊢ [X, Y][(_ ▽ _)[X, Y] ∈ P X × (X ↔ Y) → X ↔ Y ∧ (∀ S : P X; R : X ↔ Y • (_ ▽ _)[X, Y] (S, R) = {x : X; y : Y | x ∈ S ∧ x ↔ y ∈ R • x ↔ y}) ∧ (∀ R : X ↔ Y; T : P Y • (_ ▽ _)[X, Y] (R, T)
\[
\begin{align*}
\{x : X; y : Y & \mid x \mapsto y \in R \land y \in T \\
\bullet x \mapsto y}\}
\end{align*}
\]

\[
\begin{align*}
\forall & \quad [X, Y]\{[X, Y] \in \mathcal{P}X \times (X \leftrightarrow Y) \rightarrow X \leftrightarrow Y \\
\land \quad (\forall R : X \leftrightarrow Y) \\
\bullet \quad [X, Y] (R) \\
= \{x : X; y : Y & \mid x \notin S \land x \mapsto y \in R \\
\bullet x \mapsto y}\}
\end{align*}
\]

\[
\begin{align*}
\exists & \quad [X, Y]\{[X, Y] \in (X \leftrightarrow Y) \rightarrow (X \leftrightarrow Y) \\
\land \quad (\forall R : X \leftrightarrow Y; T : \mathcal{P}Y) \\
\bullet \quad [X, Y] (R, T) \\
= \{x : X; y : Y & \mid x \mapsto y \in R \land y \notin T \\
\bullet x \mapsto y}\}
\end{align*}
\]

\[
\begin{align*}
\forall & \quad [X, Y]\{[X, Y] \in (X \leftrightarrow Y) \times \mathcal{P}X \rightarrow \mathcal{P}Y \\
\land \quad (\forall R : X \leftrightarrow Y; S : \mathcal{P}X) \\
\bullet \quad [X, Y] (R, S) \\
= \{x : X; y : Y & \mid x \in S \land x \mapsto y \in R \\
\bullet y \mapsto x}\}
\end{align*}
\]

\[
\begin{align*}
\exists & \quad [X]\{[X] \subseteq (X \leftrightarrow X) \rightarrow X \leftrightarrow X \\
\land \quad (\forall R : X \leftrightarrow X) \\
\bullet \quad [X] (R) \\
= \cap \{Q : X \leftrightarrow X & \mid R \subseteq Q \land Q \not\subseteq Q\} \\
\land \quad (\exists *)[X] R \\
= \cap \{Q : X \leftrightarrow X & \mid id X \subseteq Q \\
\land R \subseteq Q \\
\land Q \not\subseteq Q\})
\end{align*}
\]

\[
\begin{align*}
\forall & \quad [X, Y]\{[X, Y] \in (X \leftrightarrow Y) \times (X \leftrightarrow Y) \rightarrow X \leftrightarrow Y \\
\land \quad (\forall f, g : X \leftrightarrow Y) \\
\bullet \quad [X, Y] (f, g) = \text{dom} g \not\subseteq f \cup g\}
\end{align*}
\]

9.11.6 Definitions

\[
\text{id} \quad [X](id X = \{x : X \bullet x \mapsto x\})
\]
9.1.11.7 Theorems

\( z_{\leftarrow \text{thm}} \quad \vdash \forall X : U; Y : U \cdot X \leftrightarrow Y = \text{id} (X \times Y) \)

\( z_{\rightarrow \text{thm}} \quad \vdash \forall x : U; y : U \cdot x \rightarrow y = (x, y) \)

\( z_{\text{dom_thm}} \quad \vdash \forall z : U; R : U \cdot z \in \text{dom} R \iff (\exists y : U \cdot (z, y) \in R) \)

\( z_{\text{ran_thm}} \quad \vdash \forall z : U; R : U \cdot z \in \text{ran} R \iff (\exists x : U \cdot (x, z) \in R) \)

\( z_{\text{id_thm}} \quad \vdash \forall X : U \cdot \text{id} X = \{ x : U \mid x \in X \cdot (x, x) \} \)

\( z_{\equiv \text{thm}} \quad \vdash \forall R : U; S : U \cdot R \equiv S = S \circ R \)

\( z_{\circ \text{thm}} \quad \vdash \forall x : U; S : U; R : U \cdot x \in S \circ R \iff (\exists y : U \cdot (x.1, y) \in R \land (y, x.2) \in S) \)

\( z_{\langle \text{thm}} \quad \vdash \forall x : U; \quad R : U \cdot x \in S \iff x.1 \in S \land x \in R \land x.2 \in S \)

\( z_{\rightarrow \text{thm}} \quad \vdash \forall x : U; \quad S : U \cdot \forall x \in S \iff \neg x.1 \in S \land x \in R \land \neg x.2 \in S \)

\( z_{\text{rel_inv_thm}} \quad \vdash \forall x : U; \quad R : U \cdot x \in R \iff (x.2, x.1) \in R \)

\( z_{\text{trans_closure_thm}} \quad \vdash \forall x : U; \quad S : U \cdot y \in R \iff (\exists x : U \cdot x \in S \land (x, y) \in R) \)

\( z_{\text{trans_closure_thm}} \quad \vdash \forall R : U \cdot R^* = \bigcap \{ Q : U \mid \text{id} \subseteq Q \land R \subseteq Q \land Q \subseteq Q \} \)

\( z_{\text{reflex_trans_closure_thm}} \quad \vdash \forall x : U; \quad R \cdot \bigcap \{ Q : U \mid (\text{id} \cdot R) \subseteq Q \land R \subseteq Q \land Q \subseteq Q \} \)

\( z_{\oplus \text{thm}} \quad \vdash \forall f : U; \quad g : U \cdot f \oplus g = \text{dom} g \iff f \cup g \)

\( z_{\rightarrow \text{clauses}} \quad \vdash \forall X : U \cdot X \leftrightarrow \{\} = \{\} \land \{\} \iff X = \{\} \)

\( z_{\text{dom_clauses}} \quad \vdash \forall a : U; \quad b : U \cdot \text{dom} U = U \land \text{dom} \{\} = \{\} \land \text{dom} \{a \mapsto b\} = \{a\} \land \text{dom} \{(a, b)\} = \{a\} \)

\( z_{\text{ran_clauses}} \quad \vdash \forall a : U; \quad b : U \cdot \text{ran} U = U \land \text{ran} \{\} = \{\} \land \text{ran} \{a \mapsto b\} = \{b\} \land \text{ran} \{(a, b)\} = \{b\} \)

\( z_{\text{id_clauses}} \quad \vdash \text{id} \{\} = \{\} \)

\( z_{\equiv \text{clauses}} \quad \vdash \forall R : U \cdot R \equiv \{\} = \{\} \land \{\} \equiv R = \{\} \land U \equiv U = U \)

\( z_{\circ \text{clauses}} \quad \vdash \forall R : U; \quad S : U \cdot U \circ R = R \land \{\} \circ R = \{\} \land U \circ U = U \)

\( z_{\langle \text{clauses}} \quad \vdash \forall R : U; \quad S : U \cdot U \langle R = R \land \{\} \langle R = \{\} \land S \langle \{\} = \{\} \)

\( z_{\rightarrow \text{clauses}} \quad \vdash \forall R : U; \quad S : U \)
\begin{itemize}
  \item \( R \triangleright U = R \land \{ \} \triangleright S = \{ \} \land R \triangleright \{ \} = \{ \} \)
  \begin{align*}
z_{\triangleleft \mbox{ clauses}} & \quad \vdash \forall R : U; S : U \\
\end{align*}
  \item \( U \triangleleft R = \{ \} \land \{ \} \triangleleft R = R \land S \triangleleft \{ \} = \{ \} \)
  \begin{align*}
z_{\triangleright \mbox{ clauses}} & \quad \vdash \forall R : U; S : U \\
\end{align*}
  \item \( R \triangleright U = \{ \} \land \{ \} \triangleright S = \{ \} \land R \triangleright \{ \} = R \)
  \begin{align*}
\end{align*}
\end{itemize}
\begin{align*}
z_{\mbox{ rel inv clauses}} & \quad \vdash U \sim = U \land \{ \} \sim = \{ \} \\
\end{align*}
\begin{align*}
z_{\mbox{ rel image clauses}} & \quad \vdash \forall R : U; S : U \bullet R \parallel \{ \} = \{ \} \land \{ \} \parallel S = \{ \} \\
\end{align*}
\begin{align*}
z_{\mbox{ trans closure clauses}} & \quad \vdash U + = U \land \{ \} + = \{ \} \\
\end{align*}
\begin{align*}
z_{\mbox{ reflex closure clauses}} & \quad \vdash U * = U \land \{ \} * = (id \ .) \\
\end{align*}
\begin{align*}
z_{\mbox{ \oplus clauses}} & \quad \vdash \forall f : U \bullet f \oplus \{ \} = f \land \{ \} \oplus f = f \land f \oplus U = U \\
\end{align*}
9.1.12 The Z Theory z_sequences

9.1.12.1 Parents

z_numbers

9.1.12.2 Children

z_sequences1 z_bags

9.1.12.3 Global Variables

\begin{align*}
s\text{eq} & \, X & \equiv \, P (Z \leftrightarrow X) \\
seq & \, X & \equiv \, P (Z \leftrightarrow X) \\
is\text{eq} & \, X & \equiv \, P (Z \leftrightarrow X) \\
(\_ \, \cup \, \_)[X] & \equiv \, (Z \leftrightarrow X) \times (Z \leftrightarrow X) \leftrightarrow Z \leftrightarrow X \\
h\text{ead} & \, [X] & \equiv \, (Z \leftrightarrow X) \leftrightarrow X \\
l\text{ast} & \, [X] & \equiv \, (Z \leftrightarrow X) \leftrightarrow X \\
t\text{ail} & \, [X] & \equiv \, (Z \leftrightarrow X) \leftrightarrow Z \leftrightarrow X \\
f\text{ront} & \, [X] & \equiv \, (Z \leftrightarrow X) \leftrightarrow Z \leftrightarrow X \\
\text{rev} & \, [X] & \equiv \, (Z \leftrightarrow X) \leftrightarrow Z \leftrightarrow X \\
\text{squash} & \, [X] & \equiv \, (Z \leftrightarrow X) \leftrightarrow Z \leftrightarrow X \\
(\_ \, \setminus \_)[X] & \equiv \, P Z \times (Z \leftrightarrow X) \leftrightarrow Z \leftrightarrow X \\
(\_ \, \setminus \_)[X] & \equiv \, (Z \leftrightarrow X) \times P X \leftrightarrow Z \leftrightarrow X \\
(\_ \, \text{disjoint} \_)[I, X] & \equiv \, P (I \leftrightarrow P X) \\
(\_ \, \text{partition} \_)[I, X] & \equiv \, (I \leftrightarrow P X) \leftrightarrow P X
\end{align*}

9.1.12.4 Fixity

\begin{align*}
\text{fun} & \, 30 \, \text{leftassoc} \\
(\_ \, \cup \_)
\end{align*}

\begin{align*}
\text{fun} & \, 40 \, \text{leftassoc} \\
(\_ \, \setminus \_)
\end{align*}

\begin{align*}
\text{fun} & \, 45 \, \text{rightassoc} \\
(\_ \, \setminus \_)
\end{align*}

\begin{align*}
\text{gen} & \, 70 \, \text{rightassoc} \\
(\text{iseq} \_)(\text{seq} \_)(\text{seq}_1 \_)
\end{align*}

\begin{align*}
\text{rel} & \equiv \, (\text{disjoint} \_)(\_ \, \text{partition} \_)
\end{align*}
9.1.12.5 Axioms


\[
\begin{align*}
\vdash [X][(\lnot \, \rightarrow)\, [X]] & \in \text{seq } X \times \text{seq } X \rightarrow \text{seq } X \\
& \land (\forall \ s, \ t : \text{seq } X \\
& \quad \cdot (\lnot \, \rightarrow)\, [X] (s, t) \\
& \quad \quad = s \cup \{ n : \text{dom } t \cdot n + \# s \leftrightarrow t \ n\})
\end{align*}
\]

head


\[
\begin{align*}
\vdash [X] & \text{(head}[X] \in \text{seq}_1 \, X \rightarrow X \\
& \land (\forall \ s : \text{seq}_1 \, X \cdot \text{head}[X] \ s = s \ 1))
\end{align*}
\]

last


\[
\begin{align*}
\vdash [X] & \text{(last}[X] \in \text{seq}_1 \, X \rightarrow X \\
& \land (\forall \ s : \text{seq}_1 \, X \cdot \text{last}[X] \ s = s \ # s))
\end{align*}
\]

tail


\[
\begin{align*}
\vdash [X] & \text{(tail}[X] \in \text{seq}_1 \, X \rightarrow \text{seq } X \\
& \land (\forall \ s : \text{seq}_1 \, X \\
& \quad \cdot \text{tail}[X] \ s = (\lambda \ n : 1 \ldots \ # s - 1 \bullet s \ (n + 1)))
\end{align*}
\]

front


\[
\begin{align*}
\vdash [X] & \text{(front}[X] \in \text{seq}_1 \, X \rightarrow \text{seq } X \\
& \land (\forall \ s : \text{seq}_1 \, X \\
& \quad \cdot \text{front}[X] \ s = (1 \ldots \ # s - 1) \triangleleft s))
\end{align*}
\]

rev


\[
\begin{align*}
\vdash [X] & \text{(rev}[X] \in \text{seq } X \rightarrow \text{seq } X \\
& \land (\forall \ s : \text{seq } X \\
& \quad \cdot \text{rev}[X] \ s = (\lambda \ n : \text{dom } s \bullet s \ (\# s - n + 1)))
\end{align*}
\]

squash


\[
\begin{align*}
\vdash [X] & \text{(squash}[X] \in (\text{seq } X \rightarrow X) \rightarrow \text{seq } X \\
& \land (\forall \ f : X \rightarrow X \\
& \quad \cdot \text{squash}[X] \ f \\
& \quad \quad = \{ p : f \\
& \quad \quad \quad \cdot \# \{ i : \text{dom } f \mid i \leq p.1 \} \leftrightarrow p.2\})
\end{align*}
\]

\[
\land (\forall a : \text{seq } X \\
& \quad \cdot (\lnot \, \rightarrow)\, [X] (a, s) = \text{squash} \ (a \triangleleft s))
\]

\[
\land (\forall s : \text{seq } X; a : \text{seq } X \\
& \quad \cdot (\lnot \, \rightarrow)\, [X] (s, a) = \text{squash} \ (s \triangleright a))
\]

\[
\land (\forall s : \text{seq } X \\
& \quad \cdot \text{\diameter} / [X] \ ⟨s\rangle = \langle\rangle)
\]

\[
\land (\forall s : \text{seq } X \\
& \quad \cdot \text{\diameter} / [X] \ (s) = s)
\]

\[
\land (\forall q, r : \text{seq } X \\
& \quad \cdot \text{\diameter} / [X] \ (q \triangleleft r) = \text{\diameter} / [X] \ q \triangleleft \text{\diameter} / [X] \ r)
\]

disjoint


\[
\begin{align*}
\vdash [I, \ X][(\text{disjoint } \rightarrow)\, [I, \ X]] & \in \text{P } (\text{seq } X \rightarrow \text{seq } X) \\
& \land (\forall S : I \rightarrow \text{P } X \\
& \quad \cdot \text{disjoint } S \in \text{disjoint } S \ [I, \ X] \\
& \quad \leftrightarrow (\forall i, j : \text{dom } S \mid i \neq j \bullet S \ i \cap S \ j = \emptyset))
\end{align*}
\]

\[
\begin{align*}
\vdash [I, \ X][(\text{partition } \rightarrow)\, [I, \ X]] & \in (I \rightarrow \text{P } X) \leftrightarrow \text{P } X \\
& \land (\forall S : I \rightarrow \text{P } X; \ T : \text{P } X \\
& \quad \cdot (S, T) \in \text{partition } S \ [I, \ X] \\
& \quad \leftrightarrow \text{disjoint } S \land \bigcup \{ i : \text{dom } S \bullet S \ i \} = T))
\end{align*}
\]

9.1.12.6 Definitions

seq


\[
\begin{align*}
\vdash [X] & \text{(seq } X = \{ f : \text{N } \rightarrow X \mid \text{dom } f = 1 \ldots \ # f\})
\end{align*}
\]

seq1


\[
\begin{align*}
\vdash [X] & \text{(seq}_1 \ X = \{ f : \text{seq } X \mid \# f > 0\})
\end{align*}
\]
iseq _ ⊢ [X](iseq X = seq X \cap (\mathbb{N} \mapsto X))
9.1.13 The Z Theory $z$-sequences

9.1.13.1 Parents

$z$-sequences $z$-numbers

9.1.13.2 Children

$z$-library

9.1.13.3 Theorems

$z_{\text{seq\_thm}}$ \quad \vdash \forall X : \mathbb{U} \bullet \text{seq } X = \bigcup \{ n : \mathbb{N} \bullet 1 .. n \to X \}

$z_{\text{prim\_seq\_induction\_thm}}$ \quad \vdash \forall X \ p

\quad \bullet \ p \ Z \{ \} \wedge (\forall x \ n \ s \bullet x \in X \wedge n \in \mathbb{N} \wedge s \in \mathbb{N} \to X \wedge p \ s \Rightarrow p \ s \cup \{ (n + 1, x) \}) \Rightarrow (\forall x \bullet s \in \mathbb{N} \to \text{seq } X \Rightarrow p \ s) \}

$z_{\text{seq\_thm1}}$ \quad \vdash \forall X : \mathbb{U}; n : \mathbb{U}

\quad \bullet \ \text{seq } X = \{ s : \mathbb{U} \bullet \exists n : \mathbb{N} \bullet s \in 1 .. n \to X \}

$z_{\text{size\_seq\_thm1}}$ \quad \vdash \forall X : \mathbb{U}; s : \mathbb{U}; n : \mathbb{N}

\quad \bullet \ s \in \text{seq } X \wedge \# s = n \Leftrightarrow s \in 1 .. n \to X

$z_{\text{size\_seq\_thm2}}$ \quad \vdash \forall n : \mathbb{N}; s : (\text{seq } _{-}) \bullet \# s = n \Leftrightarrow \text{dom } s = 1 .. n

$z_{\text{size\_seq\_N\_thm}}$ \quad \vdash \forall s : (\text{seq } _{-}) \bullet \# s = n \Leftrightarrow s \in \mathbb{N}

$z_{\text{singleton\_seq\_thm}}$ \quad \vdash \forall x : \mathbb{U}

\quad \bullet \langle x \rangle \in (\text{seq } _{-})

\quad \wedge \text{dom } \langle x \rangle = \{ 1 \}

\quad \wedge \text{ran } \langle x \rangle = \{ x \}

\quad \wedge \{ x \} \ 1 = x

$z_{\text{seq\_u\_thm}}$ \quad \vdash \forall X : \mathbb{U} \bullet \forall s : \text{seq } X \bullet s \in (\text{seq } _{-})

$z_{\text{\_\_thm}}$ \quad \vdash \forall X, Y : \mathbb{U}

\quad \bullet \forall s : \text{seq } X; t : \text{seq } Y

\quad \bullet s \land t = s \cup \{ n : \text{dom } t \bullet n + \# s \mapsto t n \}

$z_{\text{\_\_\_\_seq\_thm}}$ \quad \vdash \forall X, Y : \mathbb{U} \bullet \forall s : \text{seq } X; t : \text{seq } Y \bullet s \land t \in (\text{seq } _{-})

$z_{\text{\_\_\_\_seq\_thm1}}$ \quad \vdash \forall s : (\text{seq } _{-}); t : (\text{seq } _{-}) \bullet s \land t \in (\text{seq } _{-})

$z_{\text{\_\_def\_thm}}$ \quad \vdash \forall i : \mathbb{U}; t : (\text{seq } _{-})

\quad \bullet \{ n : \text{dom } t \bullet n + i \mapsto t n \}

\quad = \{ n : \mathbb{U}; x : \mathbb{U} \mid (n, x) \in t \}

\quad \bullet \{ n + i, x \}

$z_{\text{\_\_singleton\_thm}}$
\[ \forall x \in X \wedge s \in \mathbb{F}_{X} \wedge p \Rightarrow p \mathbb{F}_{X}(x) \wedge p \Rightarrow (\forall s \cdot s \in \mathbb{F}_{X} \Rightarrow p) \wedge (\forall s \cdot s \in \mathbb{F}_{X} \Rightarrow p) \]

\[ \forall s : (seq \_); \ x : U \cdot s \cap \langle x \rangle = s \cup \{(\# s + 1, x)\} \]

9.1. Theory Listings

487
\[\vdash \forall \ l \ n \\
\quad \bullet \ Z'\mathbb{NumList} (l, n) \equiv \\
\quad \quad \quad \quad = \{i : \mathbb{U}; \ x : \mathbb{U} | (i, x) \in \{Z'\} l\} \equiv \\
\quad \quad \quad \quad \bullet (i + Z'\mathbb{Int} n, x)\]
9.1 Theory Listings

9.1.14 The Z Theory z_sets

9.1.14.1 Parents

z_language_ps

9.1.14.2 Children

z_relations

9.1.14.3 Global Variables

\[
\begin{align*}
X \leftrightarrow Y & \quad \mathcal{P} (X \leftrightarrow Y) \\
X \rightarrow Y & \quad \mathcal{P} (X \leftrightarrow Y) \\
(\_ \not\in \_)[X] & \quad X \leftrightarrow \mathcal{P} X \\
(\_ \neq \_)[X] & \quad X \leftrightarrow X \\
\emptyset[X] & \quad \mathcal{P} X \\
(\_ \subset \_)[X] & \quad \mathcal{P} X \leftrightarrow \mathcal{P} X \\
\mathcal{P}_1[X] & \quad \mathcal{P} (\mathcal{P} X) \\
(\_ \cup \_)[X] & \quad \mathcal{P} X \times \mathcal{P} X \leftrightarrow \mathcal{P} X \\
(\_ \cap \_)[X] & \quad \mathcal{P} X \times \mathcal{P} X \leftrightarrow \mathcal{P} X \\
(\_ \setminus \_)[X] & \quad \mathcal{P} X \times \mathcal{P} X \leftrightarrow \mathcal{P} X \\
\mathcal{U}[X] & \quad \mathcal{P} (\mathcal{P} X) \leftrightarrow \mathcal{P} X \\
\cap[X] & \quad \mathcal{P} (\mathcal{P} X) \leftrightarrow \mathcal{P} X \\
second[X, Y] & \quad X \times Y \leftrightarrow Y \\
first[X, Y] & \quad X \times Y \leftrightarrow X \\
(if \_? then \_! else \_!)[X] & \quad \mathcal{B} \times X \times X \leftrightarrow X \\
(\_ \oplus \_)[X] & \quad X \times \mathcal{P} X \leftrightarrow X \\
\Pi[X] & \quad \mathcal{B} \leftrightarrow \mathcal{B} \\
(\ll \_! \gg)[X] & \quad X \leftrightarrow X \\
(\_ \_ \_ \_)[X, Y] & \quad \mathcal{P} (X \times (X \leftrightarrow Y) \times Y)
\end{align*}
\]

9.1.14.4 Fixity

fun 0 rightassoc

\[
\begin{align*}
(if \_? then \_! else \_!) (\_ \oplus \_) \\
(\ll \_! \gg) & \quad (\Pi[X])
\end{align*}
\]

fun 25 leftassoc

\[
(\_ \oplus \_)
\]

fun 30 leftassoc

\[
(\_ \setminus \_) (\_ \cup \_)
\]

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
fun 40 leftassoc
  \((\_ \cap \_\)\)

gen 5 rightassoc
  \((\_ \leftrightarrow \_\) \(\_ \rightarrow \_\)\)

gen 70 rightassoc
  \((\mathbb{P} \ _\)\)

rel
  \((\_ \not\in \_)(\_ \subset \_)(\_ \not\neq \_)(\_ \underline{\_ \_ \_} \_\)

9.1.14.5 Axioms

\(- \not\in _d\)
\(- \not\neq _d\)
\(\vdash [X__((\_ \not\neq \_)[X] \in X \leftrightarrow X\]
\(- \not\in _d\)\)
\(\wedge \ (\_ \not\in _d)[X] \in X \leftrightarrow \mathbb{P} \ X\)
\(\wedge \ (\forall \ x, y : X \bullet (x, y) \in (\_ \not\neq \_)[X] \leftrightarrow \neg \ x = y\)
\(\wedge \ (\forall \ x : X ; S : \mathbb{P} \ X\]
\(\bullet (x, S) \in (\_ \not\in \_)[X] \leftrightarrow \neg \ x \in S))\)
\(- \subset _d\) \(\vdash [X__((\_ \subset \_)[X] \in \mathbb{P} \ X \leftrightarrow \mathbb{P} \ X\]
\(\wedge \ (\forall \ S, T : \mathbb{P} \ X\]
\(\bullet (S, T) \in (\_ \subset \_)[X] \leftrightarrow S \subseteq T \wedge S \neq T))\)
\(- \cup _d\)
\(- \cap _d\)
\(- \setminus _d\)
\(- \ominus _d\)
\(\vdash [X\{(\_ \cup _d)[X], (\_ \cap _d)[X], (\_ \setminus _d)[X], (\_ \ominus _d)[X]\}
\(\subseteq \mathbb{P} \ X \times \mathbb{P} \ X \rightarrow \mathbb{P} \ X\)
\(\wedge \ (\forall \ S, T : \mathbb{P} \ X\]
\(\bullet (\_ \cup _d)[X] (S, T) = \{x : X \mid x \in S \vee x \in T\}
\wedge \ (\_ \cap _d)[X] (S, T) = \{x : X \mid x \in S \wedge x \in T\}
\wedge \ (\_ \setminus _d)[X] (S, T) = \{x : X \mid x \in S \wedge x \not\in T\}
\wedge \ (\_ \ominus _d)[X] (S, T)
\(= \{x : X\)
\(\mid \neg \)
\(\ (x \in S \leftrightarrow x \in T))))\)

\(\cup\)
\(\cap\)
\(\setminus\)
\(\ominus\)
\(\vdash [X\{\{\cup[X], \cap[X]\} \subseteq \mathbb{P} \ (\mathbb{P} \ X) \rightarrow \mathbb{P} \ X\]
\(\wedge \ (\forall \ A : \mathbb{P} \ \mathbb{P} \ X\]
\(\bullet \cup[X] A = \{x : X \mid \exists \ S : A \bullet x \in S\}
\wedge \cap[X] A = \{x : X \mid \forall \ S : A \bullet x \in S\}))\)

second
first
\(\vdash [X,
\ Y]\{(\text{first}[X, Y] \in X \times Y \rightarrow X
\wedge \text{second}[X, Y] \in X \times Y \rightarrow Y)\]
\(\wedge \ (\forall \ x : X ; y : Y\]
\(\bullet \text{first}[X, Y] (x, y) = x
\wedge \text{second}[X, Y] (x, y) = y))\)

if _? then _! else _!
\(\vdash [X]\{(\text{if } _? \text{ then } _! \text{ else } _!)\}[X] \in \mathbb{B} \times X \times X \rightarrow X\]
\(\wedge \ (\forall \ x, y : X\)
9.1. Theory Listings

- (if ? then ! else !)[X] (true, x, y) = x
  ∨ (if ? then ! else !)[X] (false, x, y) = y

Π ⊢ [X]((⊥) X) ∈ X → H X → X ∧ (⊥) X = first
Π ⊢ (Π ⊢ ?) ∈ B → B ∧ (∀ x : B • Π x = x)
≪ ⊢ [X]((≪ ⊢)) [X] ∈ X → X
  ∧ (∀ x : X • (≪ ⊢)[X] x = x)
- ⊢ [X, Y]((⊥) X, Y) ∈ P (X × P (X × Y) × Y)
  ∧ (∀ x : X; R : P (X × Y); y : Y
  • (x, R, y) ∈ (⊥) X, Y) ⇐ (x, y) ∈ R)

9.1.14.6 Definitions

- ⊢ [X, Y] (X ⊢ Y = P (X × Y))
- ⊢ [X, Y] (X = Y = P (X × Y))
- ⊢ [X, Y] (X ⊢ Y = P (X × Y))

9.1.14.7 Theorems

z.≠_thm ⊢ ∀ x : U; y : U • x ≠ y ⇐ ¬ x = y
z.≤_thm ⊢ ∀ x : U; S : U • x ∈ S ⇐ ¬ x ∈ S
z.∅_thm ⊢ ∀ x : U; y : U • x ∈ y ⇐ ¬ x ∈ ∅
z.∅_thm1 ⊢ ∅ = {∅}
z.P1_thm ⊢ ∀ X : U; P X = S : P X | S ≠ ∅
z.u_and_thm ⊢ ∀ x : U; y : U • x ∈ y ⇐ ¬ x ∈ t
z.u_or_thm ⊢ ∀ x : U; y : U • x ∈ y ⇐ ¬ x ∈ t
z.u_and_thm1 ⊢ ∀ x : U; y : U • x ∈ y ⇐ ¬ x ∈ t
z.u_or_thm1 ⊢ ∀ x : U; y : U • x ∈ y ⇐ ¬ x ∈ t
z.first_thm ⊢ ∀ x : U; first x = x.1
z.second_thm ⊢ ∀ x : U • second x = x.2
z.if_thm ⊢ ∀ x : U; y : U • if true then x else y = x
  ∧ if false then x else y = y
z.guilems_thm ⊢ ∀ x : U • ≪ x ≫ = x
\[
\begin{align*}
\text{z_underlining_brackets_thm} & \quad \vdash \forall x : U; R : U; y : U \bullet x \in R \iff (x, y) \in R \\
\text{z_U_clauses} & \quad \vdash \forall a : U \\
& \quad \quad \bullet a \cup \{\} = a \\
& \quad \quad \wedge \{\} \cup a = a \\
& \quad \quad \wedge a \cup U = U \\
& \quad \quad \wedge U \cup a = U \\
& \quad \quad \wedge a \cup a = a \\
\text{z_n_clauses} & \quad \vdash \forall a : U \\
& \quad \quad \bullet a \cap \{\} = \{\} \\
& \quad \quad \wedge \{\} \cap a = \{\} \\
& \quad \quad \wedge a \cap U = a \\
& \quad \quad \wedge U \cap a = a \\
& \quad \quad \wedge a \cap a = a \\
\text{z_set_dif_clauses} & \quad \vdash \forall a : U \\
& \quad \quad \bullet a \setminus \{\} = a \\
& \quad \quad \wedge \{\} \setminus a = \{\} \\
& \quad \quad \wedge a \setminus U = \{\} \\
& \quad \quad \wedge a \setminus a = \{\} \\
\text{z_O_clauses} & \quad \vdash \forall a : U \\
& \quad \quad \bullet a \ominus \{\} = a \\
& \quad \quad \wedge \{\} \ominus a = a \\
& \quad \quad \wedge a \ominus U = U \setminus a \\
& \quad \quad \wedge U \ominus a = U \setminus a \\
& \quad \quad \wedge a \ominus a = \{\} \\
\text{z_CL_clauses} & \quad \vdash \forall a : U \\
& \quad \quad \bullet a \subseteq a \land \{\} \subseteq a \land a \subseteq U \\
\text{z_C_clauses} & \quad \vdash \forall a : U \bullet \neg a \subset a \land \neg a \subset \{\} \land \{\} \subset U \\
\text{z_n_clauses} & \quad \vdash \forall \{\} = U \land \{\} = U \\
\text{z_U_clauses} & \quad \vdash \forall \{\} = \{\} \wedge U \cup U = U \\
\text{z_P_clauses} & \quad \vdash \forall a : U \bullet P \{\} = \{\} \wedge P \cup a \subseteq a \wedge \{\} \subseteq P a \\
\text{z_P1_clauses} & \quad \vdash \forall a : U \\
& \quad \quad \bullet P \{\} = \{\} \land (a \in P a \iff a \neq \{\}) \land \neg \{\} \in P a \\
\text{z_Cll_clauses} & \quad \vdash \forall a : U \bullet a \times \{\} = \{\} \times a = \{\} \times U = U \\
\text{z_sets_ext_clauses} & \quad \vdash \forall a : U; s \in t \\
& \quad \quad \bullet (s = t \iff (\forall x : U \bullet x \in s \iff x \in t)) \\
& \quad \quad \wedge (s \subseteq t \iff (\forall x : U \bullet x \in s \Rightarrow x \in t)) \\
& \quad \quad \wedge (s \subseteq t \iff (\exists y : U \bullet y \in t \land \neg y \in s))
\end{align*}
\]
9.2 Theory Related ML Values

This section contains various theory related ML values (e.g. the value of theorems bound to ML names, or special tactics of proof contexts associated with the theory). Where a theorem or definition is bound to an ML name the value of the theorem is to be found in the theory listing, only the ML name is given below.

9.2.1 Z Sets

SML

\[
\text{signature } \text{ZSets} = \text{sig}
\]

Description This provides the Z library sets material. It creates the theory \textit{z_sets}.

SML

\[
(∗ \text{ Proof Context: } 'z\_\in\_set\_lib ∗)
\]

Description A component proof context for handling the membership of expressions created by Z set operations of the Z library.

Predicates and expressions treated by this proof context are constructs formed from:

\[
\cap, \cup, \cap, \cup, \setminus, \varnothing, P_1, \varnothing
\]

Contents

Rewriting:

Stripping theorems:

Stripping conclusions:

All three of the above have theorems concerning the membership (\(\in\)) of terms generated by the following operators:

\[
\cap, \cup, \cap, \cup, \setminus, \varnothing, P_1, \varnothing
\]

Stripping also contains the above in negated forms.

Rewriting canonicalisation:

\(U\) simplification has the definition of \(↔\) added.

Automatic proof procedures are respectively \textit{z_basic_prove_tac}, \textit{z_basic_prove_conv}, and no existence prover.

Usage Notes It requires theory \textit{z_sets}. It is intended to be used with proof context \textit{"z_set-lang"} and \textit{"z_normal"}. It is not intended to be mixed with HOL proof contexts.

See Also \textit{"z_sets_ext_lib"}
SML

(* Proof Context: 'z_normal *)

**Description** A component proof context for normalising certain constructs of the Z library. The normalisation is done to fix on, in each case, one of two possible equivalent representations of the same concept. These constructs are:

- \( x \neq y \) normalised to \( \neg(x = y) \)
- \( x \notin y \) normalised to \( \neg(x \in y) \)
- \( \emptyset \) normalised to \( \{\} \)
- \( x \in P \) normalised to \( x \subseteq y \)
- if true then \( x \) else \( y \) normalised to \( x \)
- if false then \( x \) else \( y \) normalised to \( y \)

**Contents**

Rewriting:

- \( z \in \mathcal{P}_\text{thm} \), \( z \emptyset_\text{thm} \), \( z \notin_\text{thm} \), \( z \neq_\text{thm} \), \( z \text{if}_\text{thm} \)

Stripping theorems:

- \( z \in \mathcal{P}_\text{thm} \), \( z \emptyset_\text{thm} \), \( z \notin_\text{thm} \), \( z \neq_\text{thm} \), \( z \text{if}_\text{thm} \)
- and these all pushed through \( \neg \)

Stripping conclusions:

- \( z \in \mathcal{P}_\text{thm} \), \( z \emptyset_\text{thm} \), \( z \notin_\text{thm} \), \( z \neq_\text{thm} \), \( z \text{if}_\text{thm} \)
- and these all pushed through \( \neg \)

Rewriting canonicalisation:

U simplification has the definition of \( \leftrightarrow \) added.

Automatic proof procedures are respectively \( z \text{basic_prove_tac} \), \( z \text{basic_prove_conv} \), and no existence prover.

**Usage Notes** It requires theory \( z \text{sets} \). It is intended to be used with proof contexts "'z_set_lib" or "'z_set_alg"."
9.2. Theory Related ML Values

SML

(* Proof Context: 'z_sets_alg *)

**Description**  A component proof context for handling algebraic reasoning of expressions created by Z set operations of the Z library.

Predicates and expressions treated by this proof context are constructs formed from:
\[\in, \cap, \cup, \setminus, \subseteq, \subset, \ast, \bigcup, \bigcap, \mathbb{P}, \{D \mid \text{false} \cdot V\}, \times\]

**Contents**

Rewriting:

\[z_\cup\_clauses, z_\cap\_clauses, z_\setdiff\_clauses, z_\Theta\_clauses,\]
\[z_\subseteq\_clauses, z_\subset\_clauses, z_\cup\_clauses, z_\cap\_clauses,\]
\[z_\mathbb{P}\_clauses, z_\mathbb{P}_1\_clauses, z_\text{seta}_{\text{false}}\_conv,\]
\[z_\times\_clauses\]

Stripping theorems:

\[z_\cup\_clauses, z_\cap\_clauses, z_\setdiff\_clauses, z_\Theta\_clauses,\]
\[z_\subseteq\_clauses, z_\subset\_clauses, z_\cup\_clauses, z_\cap\_clauses,\]
\[z_\mathbb{P}\_clauses, z_\mathbb{P}_1\_clauses, z_\text{seta}_{\text{false}}\_conv,\]
\[z_\times\_clauses\]

*as necessary converted to membership statements by \(\in\_C\),
And all of this pushed through \(\neg\)

Stripping conclusions:

\[z_\cup\_clauses, z_\cap\_clauses, z_\setdiff\_clauses, z_\Theta\_clauses,\]
\[z_\subseteq\_clauses, z_\subset\_clauses, z_\cup\_clauses, z_\cap\_clauses,\]
\[z_\mathbb{P}\_clauses, z_\mathbb{P}_1\_clauses, z_\text{seta}_{\text{false}}\_conv,\]
\[z_\times\_clauses\]

*as necessary converted to membership statements by \(\in\_C\),
And all of this pushed through \(\neg\)

Rewriting canonicalisation:

Automatic proof procedures are respectively \(z\_\text{basic\_prove\_tac}, z\_\text{basic\_prove\_conv}\), and no existence prover.

**Usage Notes** It requires theory \(z\_\text{sets}\). It is intended to usable with proof context \(\"z\_\in\_set\_lib\"\), and always with \(\"z\_normal\"\). The proof context ensures that its simplifications will be attempted before more general rules concerned membership of set operators are used.

It is not intended to be mixed with HOL proof contexts.
Description   An aggressive component proof context for handling the manipulation of Z set expressions, by breaking them into predicate calculus, within the Z library.

Predicates treated by this proof context are constructs formed from:

\[ \subseteq, \subset \]

Contents

Rewriting:

\[ z_{\subseteq_{\text{conv}}}, z_{\subseteq_{\text{thm}}}, z_{\text{setd}_{\subseteq_{\text{conv}}}} \]

Stripping theorems:

\[ z_{\subseteq_{\text{conv}}}, z_{\subseteq_{\text{thm}}}, z_{\text{setd}_{\subseteq_{\text{conv}}}}, \]

plus these all pushed in through \( \neg \)

Stripping conclusions:

\[ z_{\subseteq_{\text{conv}}}, z_{\subseteq_{\text{thm}}}, z_{\text{setd}_{\subseteq_{\text{conv}}}}, \]

plus these all pushed in through \( \neg \)

In all of the above \( z_{\text{setd}_{\subseteq_{\text{conv}}}} \), which does the conversion:

\[ \{x_1, x_2, \ldots \} \subseteq y \implies x_1 \in y \land x_2 \in y \land \ldots \]

is used, where applicable, in preference to \( z_{\subseteq_{\text{conv}}} \), which, in the simplest cases, does the conversion:

\[ p \subseteq q \implies \forall x_1 \bullet x_1 \in p \Rightarrow xx_1 \in q \]

Rewriting canonicalisation:

Automatic proof procedures are respectively \( z_{\text{basic}_\text{prove_tac}}, z_{\text{basic}_\text{prove_conv}} \), and no existence prover.

Usage Notes   It requires theory \( z_{\text{sets}} \). It is intended to always be used in conjunction with "\( z_{\text{set_lib}} \)" and "\( z_{\text{set_ext_lang}} \)". If used with "\( z_{\text{sets_alg}} \)" then the simplification in that proof context will take precedence over the extensionality effects of this proof context.

It is not intended to be mixed with HOL proof contexts.

See Also   "\( z_{\in_{\text{set_lib}}} \)"
9.2. Theory Related ML Values

SML
\[
\begin{align*}
\text{val } & \text{mk}_z \subseteq : (\text{TERM} \times \text{TERM}) \rightarrow \text{TERM}; \\
\text{val } & \text{dest}_z \subseteq : \text{TERM} \rightarrow (\text{TERM} \times \text{TERM}); \\
\text{val } & \text{is}_z \subseteq : \text{TERM} \rightarrow \text{bool};
\end{align*}
\]

Description  Constructor, destructor and discriminator functions for Z subset terms.

Errors  
78006  ?0 is not of the form $\exists x a \subseteq s$
78004  ?0 and ?1 do not have the same types
78007  ?0 does not have a Z set type

SML
\[
\begin{align*}
\text{val } & \text{z.seta}\_false\_conv : \text{CONV};
\end{align*}
\]

Description  Simplifies a Z set abstraction whose predicate is false.

Conversion  
\[
\begin{align*}
\vdash \{ D \mid \text{false} \cdot P \} = \{ \} \\
\forall \{ D \mid \text{false} \cdot P \}^\gamma
\end{align*}
\]

Errors  
78002  ?0 is not of the form \( \forall \{ D \mid \text{false} \cdot P \}^\gamma \)

SML
\[
\begin{align*}
\text{val } & \text{z}\_\subseteq\_conv : \text{CONV};
\end{align*}
\]

Description  Use \( z\_\subseteq\_thm \) in combination with knowledge about tuples. Given as input an equality of the form \( v \subseteq w \) then:

If \( w \) is of type \( ty \) \( SET \) where \( ty \) is not a tuple type:

Conversion  
\[
\begin{align*}
\vdash (v \subseteq w) \iff (\forall xn : U \cdot xn \in v \Rightarrow xn \in w) \\
\forall \{ v \subseteq w \}^\gamma
\end{align*}
\]

where \( xn \) is the first variable in the list \( x1, x2,... \) that doesn’t appear in \( v \) or \( w \) (free or bound).

If \( w \) is of type \( ty \) \( SET \) where \( ty \) is an \( n \)-tuple type, or binding type, then sufficient \( x\_i \) will be introduced, instead of just \( xn \), to allow \( xn \) to be replaced by a construct of bindings and tuples of the \( x\_i \), such that no \( x\_i \) has a binding or tuple type and appears exactly once in the construct.

Example  
\[
\begin{align*}
\vdash p \subseteq r \times [a, b : X] \times x2 \gamma = \\
\iff (\forall x1 : U; x3 : U; x4 : U; x5 : U \\
\quad \cdot (x1, (a \equiv x3, b \equiv x4), x5) \in p \\
\quad \Rightarrow (x1, (a \equiv x3, b \equiv x4), x5) \in r \times [a, b : X] \times x2)
\end{align*}
\]

Notice how the introduced universal quantification “skips” \( x2 \) which is present in the input term.

See Also  \( z\_\subseteq\_thm \), \( z\_\in\_P\_\subseteq\_conv \).

Errors  
78001  ?0 is not of the form $\exists v \subseteq w^\gamma$
These are the ML bindings of the definitions of built-in global variables that support the use of the ProofPower-Z language.

**Description** These are the ML bindings of the definitions of the theory $z$-sets.
9.2. Theory Related ML Values

SML

val z_≠_thm: THM;
val z_∉_thm: THM;
val z_Ø_thm: THM;
val z_⊆_thm: THM;
val z_⊆_thm1: THM;
val z_⊂_thm: THM;
val z_∈_thm: THM;
val z_∅_thm: THM;
val z_⊆_thm: THM;
val z_∈_P_thm: THM;
val z_P1_thm: THM;
val z_∪_thm: THM;
val z_∩_thm: THM;
val z_set_dif_thm: THM;
val z_⊙_thm: THM;
val z_∪_thm: THM;
val z_∩_thm: THM;
val z_first_thm: THM;
val z_second_thm: THM;
val z_∪_clauses: THM;
val z_∩_clauses: THM;
val z_set_dif_clauses: THM;
val z_⊙_clauses: THM;
val z_⊆_clauses: THM;
val z_⊂_clauses: THM;
val z_∪_clauses: THM;
val z_P1_clauses: THM;
val z_×_clauses: THM;
val z_if_thm: THM;
val z_guillemets_thm: THM;
val z_underlining_brackets_thm: THM;
val z_sets_ext_clauses: THM;

Description These are the ML bindings of the theorems of the theory z_sets.

9.2.2 Z Relations

SML

signature ZRelations = sig

Description This provides the basic proof support for the Z library relations. It creates the theory z_relations.
Description A component proof context for handling the membership of Z relations created by Z library operations.

Predicates treated by this proof context are constructs formed from:

\[ \mapsto, \oplus, -^+, -^-, -^+, -^-, \{ - \}, \bowtie, \prec, \succ, \preceq, \succsim, \precsim, o, \circ, id, ran, dom, \leftrightarrow \]

Contents

Rewriting:

\[ z \mapsto \text{thm} \]

Stripping theorems:

Stripping conclusions:

All three of the above also have theorems concerning the membership of terms generated by the following operators:

\[ \oplus, -^+, -^-, -^+, -^-, \{ - \}, \bowtie, \prec, \succ, \preceq, \succsim, \precsim, o, \circ, id, ran, dom, \leftrightarrow \]

Stripping also contains the above in negated forms.

Rewriting canonicalisation:

Automatic proof procedures are respectively \texttt{z-basic-prove-tac}, \texttt{z-basic-prove-conv}, and no existence prover.

Usage Notes It requires theory \texttt{z-relations}. It is intended to be used with proof contexts "\texttt{z-sets-ext}" and "\texttt{z-sets-alg}". It is not intended to be mixed with HOL proof contexts.
\*\* Proof Context: 'z\_rel\_alg *

**Description** A component proof context for the simplification of Z relations created by Z library operations.

Predicates treated by this proof context are constructs formed from:
\[ \oplus, -\oplus, -^+, -^*, -^\sim, -\{\}, \geq, \leq, <, \]
\[ o, \land, id, ran, dom, \leftrightarrow \]

**Contents**

**Rewriting:**
\[ z\_\leftrightarrow\_clauses, z\_dom\_clauses, z\_ran\_clauses, z\_id\_clauses, \]
\[ z\_\ominus\_clauses, z\_o\_clauses, z\_<\_clauses, z\>_\_clauses, \]
\[ z\_<\_clauses, z\>_\_clauses, z\_rel\_inv\_clauses, z\_rel\_image\_clauses, \]
\[ z\_trans\_closure\_clauses, z\_reflex\_closure\_clauses, \]
\[ z\_\oplus\_clauses \]

**Stripping theorems:**
\[ z\_\leftrightarrow\_clauses, z\_dom\_clauses, z\_ran\_clauses, z\_id\_clauses, \]
\[ z\_\ominus\_clauses, z\_o\_clauses, z\_<\_clauses, z\>_\_clauses, \]
\[ z\_<\_clauses, z\>_\_clauses, z\_rel\_inv\_clauses, z\_rel\_image\_clauses, \]
\[ z\_trans\_closure\_clauses, z\_reflex\_closure\_clauses, \]
\[ z\_\oplus\_clauses \]

Expressed as memberships, as necessary, using \[ \in \_C \]

All also pushed through \[ \neg \]

**Stripping conclusions:**
\[ z\_\leftrightarrow\_clauses, z\_dom\_clauses, z\_ran\_clauses, z\_id\_clauses, \]
\[ z\_\ominus\_clauses, z\_o\_clauses, z\_<\_clauses, z\>_\_clauses, \]
\[ z\_<\_clauses, z\>_\_clauses, z\_rel\_inv\_clauses, z\_rel\_image\_clauses, \]
\[ z\_trans\_closure\_clauses, z\_reflex\_closure\_clauses, \]
\[ z\_\oplus\_clauses \]

Expressed as memberships, as necessary, using \[ \in \_C \]

All also pushed through \[ \neg \]

**Rewriting canonicalisation:**

Automatic proof procedures are respectively \[ z\_basic\_prove\_tac, z\_basic\_prove\_conv, \] and no existence prover.

**Usage Notes** It requires theory \[ z\_relations \]. It is intended to be used with proof contexts \"z\_sets\_ext\" and \"z\_sets\_alg\". There are clashes of effects if merged with \"z\_\in\_rel\", resolved in favour of \"z\_\in\_rel\", though the resulting merge has its uses. It is not intended to be mixed with HOL proof contexts.
Proof Context: 'z_tuples *

Description A component proof context for handling the manipulation of Z tuples and cartesian products within the Z language and library.

Expressions and predicates treated by this proof context are constructs formed from:

\[(\text{membership of}) \times, \text{equations of tuple displays}, \]
\[\text{selection from tuple displays, first, second, } \mapsto\]

Contents

Rewriting:

| z\_\in\_\times\_\conv, 
| z\_\tuple\_eq\_\conv, z\_sel\_\conv, 
| z\_\text{second}\_\thm, z\_\text{first}\_\thm

Stripping theorems:

| z\_\in\_\times\_\conv, 
| z\_\tuple\_eq\_\conv, \in\_C z\_sel\_\conv, 
| z\_sel\_\conv \text{(where component of tuple is boolean),} 
| plus these all pushed in through \neg

Stripping conclusions:

| z\_\in\_\times\_\conv, 
| z\_\tuple\_eq\_\conv, \in\_C z\_sel\_\conv, 
| z\_sel\_\conv \text{(where component of tuple is boolean),} 
| plus these all pushed in through \neg

Stripping also contains the above in negated forms.

Rewriting canonicalisation:

Automatic proof procedures are respectively \_basic\_prove\_tac, \_basic\_prove\_conv, and no existence prover (1-tuples and 2-tuples are handled in proof context “z_predicates”).

Usage Notes It requires theory z\_relations. It is intended to be used with proof contexts “z\_sets\_ext” and “z\_sets\_alg”. It should not be used with \_z\_tuples\_lang. It is not intended to be mixed with HOL proof contexts.
9.2. Theory Related ML Values

SML

(* Proof Context: 'z_elementwise_eq *)

Description  A aggressive component proof context for forcing the elementwise comparison of any two items of tuple or binding types.

Predicates and expressions treated by this proof context are:

\[ x = y \quad \text{where } x \text{ has a tuple type} \]
\[ x = y \quad \text{where } x \text{ has the type of a binding display} \]

Contents

Rewriting:

\[ z\_binding\_eq\_conv3 \quad z\_tuple\_eq\_conv1 \]

Stripping theorems:

\[ z\_binding\_eq\_conv3 \quad z\_tuple\_eq\_conv1 \]

plus these all pushed in through \( \neg \)

Stripping conclusions:

\[ z\_binding\_eq\_conv3 \quad z\_tuple\_eq\_conv1 \]

plus these all pushed in through \( \neg \)

Rewriting canonicalisation:

Automatic proof procedures are respectively \( z\_basic\_prove\_tac \), \( z\_basic\_prove\_conv \), and no existence prover.

Usage Notes  It requires theory \( z\_relations \). It is intended to be used with proof context "\( z\_language \)". It is not intended to be mixed with HOL proof contexts.

SML

(* Proof Context: z_language *)

Description  A mild complete proof context for reasoning in the Z language. It will also do some minor pieces of Z Library reasoning - in particular, it “understands” maplets and \( \subseteq \).

It consists of the merge of the proof contexts:

"\( z\_predicates \)"
"\( z\_\in\_set\_lang \)"
"\( z\_bindings \)"
"\( z\_schemas \)"
"\( z\_tuples \)"

Usage Notes  It requires theory \( z\_relations \) (rather than \( z\_language\_ps \) as one might expect). This is because we wish to provide a proof context that can be added to to provide Library reasoning facilities. This means that we cannot use the Z language proof context "\( z\_tuples\_lang \)" or "\( asthisisincompatiblewith\)"\( z\_tuples \)" its library extension. This is why this proof context understands maplets, which are Z Library contracts.
(* Proof Context: z_language_ext *)

**Description** An aggressive complete proof context for reasoning in the Z language. It uses the extensionality of sets, and will also decompose any equality of objects of schema or tuple type into a pairwise equality clause. It will also do some minor pieces of Z Library reasoning - in particular, it "understands" maplets and $\subseteq$.

It consists of the merge of the proof contexts:

\[
\text{"z predicates",} \\
\text{"z$\in$set_lang",} \\
\text{"z_sets_ext_lang",} \\
\text{"z bindings",} \\
\text{"z schemas",} \\
\text{"z tuples",} \\
\text{"z_elementwise_eq"}
\]

**Usage Notes** It requires theory $z_relations$ (rather than $z_language_ps$ as one might expect). This is because we wish to provide a proof context that can be added to to provide Library reasoning facilities. This means that we cannot use the Z language proof context "$z_tuples_lang$", as this is incompatible with "$z$ tuples", its library extension. This is why this proof context understands maplets, which are Z Library constructs.

(* Proof Context: z_sets_ext *)

**Description** An aggressive complete proof context for handling the manipulation of Z set expressions, by breaking them into predicate calculus.

It consists of the merge of the proof contexts:

\[
\text{"z_language_ext",} \\
\text{"z$\in$set_lib",} \\
\text{"z_sets_ext_lib",} \\
\text{"z_normal"}
\]

**Usage Notes** It requires theory $z_relations$.

It is not intended to be mixed with HOL proof contexts or "$z$ sets alg", which offers an alternative approach to reasoning about sets.

(* Proof Context: z_sets_alg *)

**Description** A mild complete proof context for handling the manipulation of Z set expressions, by algebraic reasoning and knowledge of the set membership of the set operators.

It consists of the merge of the proof contexts:

\[
\text{"z_language",} \\
\text{"z$\in$set_lib",} \\
\text{"z_sets_alg",} \\
\text{"z_normal"}
\]

**Usage Notes** It requires theory $z_relations$. The proof context ensures that its simplifications will be attempted before more general rules concerned membership of set operators are used (including extensionality rules).

It is not intended to be mixed with HOL proof contexts.
9.2. Theory Related ML Values

Description
An aggressive complete proof context for reasoning about Z relations. When stripping or rewriting it attempts to reduce any predicate concerning relational constructs to predicate calculus. As a side effect set constructs are also reduced to predicate calculus. The proof context is a merge of:

- $z$ sets ext — extensional reasoning about sets
- $z \in$ rel — membership of relational constructs
- $z$ rel alg — simplifications of relational constructs

It requires the theory “z_relations”.

ML bindings of the theorems (other than consistency ones) in theory z_relations.
**z\_binding\_eq\_conv3**

**Description** A conversion for eliminating equations of bindings to an elementwise equality clause. In general this does:

\[
\vdash (b_1 = b_2) \iff (b_1.s_1 = b_2.s_1) \land (b_1.s_2 = b_2.s_2) \land ...
\]

\[\text{z}\_\text{binding}\_\text{eq}\_\text{conv3} \quad \frac{}{z \_ b_1 = b_2} \]

However, it will expand on either side \(\theta\)-terms into binding displays, and also use \(z\_sel\_conv\) on selections from binding displays (whether from \(\theta\)-terms or otherwise).

**Errors**

\[\text{42013} \ ?0 \ \text{is not of the form} \ z \_ \text{binding} = \text{binding} \]

---

**z\_sel\_conv**

**Description** This conversion carries out the selection from a tuple display.

\[
\vdash (t_1,...,t_i,...,t_n).i = t_i \\
\]

\[\text{z}\_\text{sel}\_\text{conv} \quad \frac{}{z \_ (t_1,...,t_i,...,t_n).i} \]

\(x \mapsto y\) will be treated as a 2-tuple.

**See Also** \(z\_sel\_lang\_conv\)

**Errors**

\[\text{47185} \ ?0 \ \text{is not a Z tuple selection} \]

\[\text{42006} \ ?0 \ \text{is not of the form} \ z \_ (x...).i \]

---

**z\_tuple\_eq\_conv**

**Description** A conversion for eliminating tuples over equality.

\[
\vdash (t_1,t_2,...) = (u_1,u_2,...) \iff \ (t_1 = u_1) \land (t_2 = u_2) \land ...
\]

\[\text{z}\_\text{tuple}\_\text{eq}\_\text{conv} \quad \frac{}{z \_ (t_1,t_2,...) = (u_1,u_2,...)} \]

\(x \mapsto y\) will be treated as a 2-tuple.

**See Also** \(z\_\text{tuple}\_\text{lang}\_eq\_conv\)

**Errors**

\[\text{42003} \ ?0 \ \text{is not of the form} \ z \_ (x1,...) = (y1,...) \]
9.2. Theory Related ML Values

<table>
<thead>
<tr>
<th>val z_tuple_eq_conv1 : CONV;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong>  A conversion for eliminating tuples over equality to an elementwise equality clause.</td>
</tr>
</tbody>
</table>
| Conversion  \[
\vdash (t1 = t2) \iff (t1.1 = t2.1 \land ...) \quad \text{z_tuple_eq_conv} \\
\quad \frac{\exists t1 = t2^\top}{z1.t1 = t2^\top}
\]

This will then use \(z\_sel\_conv\) to eliminate explicit tuples. \(x \mapsto y\) will be treated as a 2-tuple.

| See Also  \(z\_tuple\_lang\_eq\_conv\) |

| Errors  \[83001\] ?0 is not of the form: \(\exists\)tuple1 = tuple2^\top |

<table>
<thead>
<tr>
<th>val z_tuple_intro_conv : CONV;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong>  This conversion carries out the elimination of a tuple display of tuple selections from the same tuple.</td>
</tr>
</tbody>
</table>
| Conversion  \[
\vdash (t.1,,,n) = t \quad \text{z_tuple_intro_conv} \\
\quad \frac{\exists t.1,,,n}{\exists (t.1,,,n)^\top}
\]

where \(n\) is the arity of \(t\). \(x \mapsto y\) will be treated as a 2-tuple.

| See Also  \(z\_tuple\_lang\_intro\_conv\) |

| Errors  \[42005\] ?0 is not of the form: \(\exists\)\((t.1,,,n)^\top\) |

| val z_rdef : THM; |
| val z_dom_def : THM; |
| val z_ran_def : THM; |
| val z_id_def : THM; |
| val z<_def : THM; |
| val z>_def : THM; |
| val z>=_def : THM; |
| val z<=_def : THM; |
| val z@_def : THM; |
| val z_rel_inv_def : THM; |
| val z_rel_image_def : THM; |
| val z_tc_def : THM; |
| val zrtc_def : THM; |
| val z⊕_def : THM; |

| **Description**  These are the definitions of the theory \(z\_relations\). |

9.2.3 Z Functions

<table>
<thead>
<tr>
<th>signature ZFunctions = sig</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong>  This provides the basic proof support for the Z library functions. It creates the theory (z_functions).</td>
</tr>
</tbody>
</table>
Proof Context: 'z⊆fun *)

Description  A component proof context for handling the membership of Z functions created by Z library operations. Expressions and predicates treated by this proof context are constructs formed from:

\[ \mapsto, \mapsto, \mapsto, \mapsto, \mapsto, \mapsto, \mapsto \]

Contents

Rewriting:

Stripping theorems:

Stripping conclusions:

All three of the above also have theorems concerning the membership of terms generated by the following operators:

\[ \mapsto, \mapsto, \mapsto, \mapsto, \mapsto, \mapsto, \mapsto \]

Stripping also contains the above in negated forms.

Rewriting canonicalisation:

Automatic proof procedures are respectively \texttt{z_basic_prove_tac}, \texttt{z_basic_prove_conv}, and no existence prover.

Usage Notes  It requires theory \texttt{z_sets}. It is intended to be used with proof context \texttt{"z⊆rel"}. It is not intended to be mixed with HOL proof contexts.
9.2. Theory Related ML Values

**SML**

(* Proof Context: 'z_fun_alg *)

**Description**  A component proof context for handling the simplification of Z functions created by Z library operations. Expressions and predicates treated by this proof context are constructs formed from:

\[ \rightarrow, \mapsto, \rightarrow\mapsto, \mapsto\rightarrow, \mapsto\rightarrow \]

**Contents**

Rewriting:

\[ z\rightarrow\rightarrow\cdash c l a u s e s, z\mapsto\rightarrow\cdash c l a u s e s, z\rightarrow\rightarrow\cdash c l a u s e s, z\rightarrow\rightarrow\cdash c l a u s e s, z\rightarrow\rightarrow\cdash c l a u s e s \]

Stripping theorems:

\[ z\rightarrow\rightarrow\cdash c l a u s e s, z\mapsto\rightarrow\cdash c l a u s e s, z\rightarrow\rightarrow\cdash c l a u s e s, z\rightarrow\rightarrow\cdash c l a u s e s, z\rightarrow\rightarrow\cdash c l a u s e s \]

Expressed as membership statements as necessary, using \( \in C \).

All also pushed through \( \neg \).

Stripping conclusions:

\[ z\rightarrow\rightarrow\cdash c l a u s e s, z\mapsto\rightarrow\cdash c l a u s e s, z\rightarrow\rightarrow\cdash c l a u s e s, z\rightarrow\rightarrow\cdash c l a u s e s, z\rightarrow\rightarrow\cdash c l a u s e s \]

Expressed as membership statements as necessary, using \( \in C \).

All also pushed through \( \neg \).

Rewriting canonicalisation:

Automatic proof procedures are respectively \( z\_basic\_prove\_tac, z\_basic\_prove\_conv\), and no existence prover.

**Usage Notes**  It requires theory \( z\_sets \). The proof context ensures that its simplifications will be attempted before more general rules concerned membership of set operators are used (including extensionality rules).

It is not intended to be mixed with HOL proof contexts.

**SML**

(* Proof Context: z_fun_ext *)

**Description**  An aggressive complete proof context for reasoning about Z functions. When stripping or rewriting it attempts to reduce any predicate concerning function constructs to predicate calculus. As a side effect relational and set constructs are also reduced to predicate calculus.

The proof context is a merge of:

\[ z\_rel\_ext \text{ } - \text{ extensional reasoning about relations (and sets)} \]

\[ z\_\in\_fun \text{ } - \text{ membership of function constructs} \]

\[ z\_fun\_alg \text{ } - \text{ simplifications of function constructs} \]

It requires the theory “z_functions”.

**SML**

\[ \text{val } z\rightarrow\rightarrow\cdash \text{def : THM}; \quad \text{val } z\rightarrow\rightarrow\cdash \text{def : THM}; \]

\[ \text{val } z\rightarrow\rightarrow\cdash \text{def : THM}; \quad \text{val } z\rightarrow\rightarrow\cdash \text{def : THM}; \]

\[ \text{val } z\rightarrow\rightarrow\cdash \text{def : THM}; \quad \text{val } z\rightarrow\rightarrow\cdash \text{def : THM}; \]

**Description**  These are the ML bindings of the defining theorems in the theory \( z\_functions \).
val \( z \mapsto \text{thm} : \text{THM} \);
val \( z \mapsto \text{thm}1 : \text{THM} \);
val \( z \mapsto \text{thm}2 : \text{THM} \);
val \( z \mapsto \text{app_thm} : \text{THM} \);
val \( z \mapsto \text{clauses} : \text{THM} \);
val \( z \mapsto \text{clauses}1 : \text{THM} \);
val \( z \mapsto \text{clauses}2 : \text{THM} \);
val \( z \mapsto \text{clauses}3 : \text{THM} \);
val \( z \mapsto \text{app_clauses} : \text{THM} \);
val \( z \mapsto \text{fun_clauses} : \text{THM} \);
val \( z \mapsto \text{fun_dom_clauses} : \text{THM} \);
val \( z \mapsto \text{fun_ran_clauses} : \text{THM} \);

**Description**  These are the ML bindings of the theorems in the theory \( z \_\text{functions} \).

### 9.2.4 Z Numbers

```sml
signature ZNumbers = sig
```

**Description**  This provides the basic proof support for the Z library relations. It creates the theory \( z \_\text{numbers} \).
9.2. Theory Related ML Values

SML

(* Proof Context: 'z_numbers *)

Description  A component proof context for handling the basic arithmetic operations for Z.
Expressions and predicates treated by this proof context are constructs formed from:
| +, *, −, abs, div, mod, Z, ≤, ≥, >, =, N |

Contents

Rewriting:

| z_plus_conv, z_times_conv, z_subtract_minus_conv |
| z_abs_conv, z_div_conv, z_mod_conv |
| z_Z_eq_conv, z_leq_conv, z_less_conv |
| z_geq_leq_conv, z_greater_less_conv, z_in_N_conv |
| z_plus_clauses, z_minus_clauses, z_leq_clauses |
| z_less_clauses, z_not_leq_thm, z_not_less_thm, |
| z_in_N1_thm, simple_z_dot_conv, z_in_dot_dot_conv |

Stripping theorems:

| z_Z_eq_conv, z_leq_conv, z_less_conv |
| z_geq_leq_conv, z_greater_less_conv, z_in_N_conv |
| z_plus_clauses, z_minus_clauses, z_leq_clauses |
| z_less_clauses, z_in_N1_thm, z_in_dot_dot_conv |
| and all the above pushed through ¬ |

| z_not_leq_thm, z_not_less_thm, z_leq_conv, z_less_conv |

Stripping conclusions: as for stripping theorems.

Rewriting canonicalisation: blank.

U-simplification:

\[ |- Z = \mathbb{U} \]

Automatic proof procedures: z_basic_prove_tac, z_basic_prove_conv.

Automatic existence prover: blank.

See Also  Proof context 'z_numbers1
Description  A component proof context for handling the basic arithmetic operations for Z. It is distinct from 'z.numbers' by its normalising all inequalities to ≤.

Expressions and predicates treated by this proof context are constructs formed from:
| +, *, −, abs, div, mod, Z, ≤, <, ≥, >, =, N |

Contents

Rewriting:

| z.plus.conv, z.times.conv, z.subtract.minus.conv |
| z.abs.conv, z.div.conv, z.mod.conv |
| z.Z.eq.conv, z.≤.conv, z.less.conv |
| z.≥.≤.conv, z.greater.less.conv, z.∈.N.conv |
| z.plus.clauses, z.minus.clauses, z.≤.clauses |
| z.less.clauses, z.¬.less.thm, |
| z.∈.N.1.thm, z.simple.dot.dot.conv, z.∈.dot.dot.conv, |
| conv.rule (ONCE_MAP_C eq.sym.conv) z.¬.≤.thm |

The final conversion to < to ≤ will only occur if no other rewriting applies.

Stripping theorems:

| z.Z.eq.conv, z.≤.conv, z.less.conv |
| z.≥.≤.conv, z.greater.less.conv, z.∈.N.conv |
| z.plus.clauses, z.minus.clauses, z.≤.clauses |
| z.less.clauses, z.∈.N.1.thm, z.∈.dot.dot.conv |
| and all the above pushed through ¬ |
| z.¬.less.thm, z.≤.conv, z.less.conv, |
| conv.rule (ONCE_MAP_C eq.sym.conv) z.¬.≤.thm |

Stripping conclusions: as for stripping theorems.

Rewriting canonicalisation: blank.

U-simplification:

| |- Z = U |

Automatic proof procedures: z.basic.prove_tac, z.basic.prove.conv.

Automatic existence prover: blank.
9.2. Theory Related ML Values

SML

val dest_z_le : TERM -> TERM * TERM;
val dest_z_ge : TERM -> TERM * TERM;
val dest_z_abs : TERM -> TERM;
val dest_z_div : TERM -> TERM * TERM;
val dest_z_greater : TERM -> TERM * TERM;
val dest_z_less : TERM -> TERM * TERM;
val dest_z_minus : TERM -> TERM;
val dest_z_mod : TERM -> TERM * TERM;
val dest_z_plus : TERM -> TERM * TERM;
val dest_z_signed_int : TERM -> INTEGER;
val dest_z_subtract : TERM -> TERM * TERM;
val dest_z_times : TERM -> TERM * TERM;

Description These are derived destructor functions for the Z basic arithmetic operations. An optionally signed integer literal, signed_int, is taken to be either a numeric literal or the result of applying (~) to a numeric literal. The other constructors correspond directly to the arithmetic operations of the theory z_numbers with alphabetic names assigned to give a valid ML name as needed (greater :, less :, minus: ~, plus : +, subtract : -, times : *).

As usual, there are also corresponding discriminator (is ...) and constructor functions (mk ...). For programming convenience, dest_z_signed_int returns 0 and is_z_signed_int returns true when applied to ~0, but mk_z_signed_int cannot be used to construct such a term.

Errors

86101 ?0 is not of the form \( \frac{i}{j} \)
86102 ?0 is not of the form \( \frac{i}{j} \)
86103 ?0 is not of the form \( \frac{i}{j} \)
86104 ?0 is not of the form \( \frac{i}{j} \)
86105 ?0 is not of the form \( \frac{i}{j} \)
86106 ?0 is not of the form \( \frac{i}{j} \)
86107 ?0 is not of the form \( \frac{i}{j} \)
86108 ?0 is not of the form \( \frac{i}{j} \)
86109 ?0 is not of the form \( \frac{i}{j} \)
86110 ?0 is not an optionally signed integer literal
86111 ?0 is not of the form \( \frac{i}{j} \)
86112 ?0 is not of the form \( \frac{i}{j} \)

SML

val is_z_le : TERM -> bool;
val is_z_ge : TERM -> bool;
val is_z_abs : TERM -> bool;
val is_z_div : TERM -> bool;
val is_z_greater : TERM -> bool;
val is_z_less : TERM -> bool;
val is_z_minus : TERM -> bool;
val is_z_mod : TERM -> bool;
val is_z_plus : TERM -> bool;
val is_z_signed_int : TERM -> bool;
val is_z_subtract : TERM -> bool;
val is_z_times : TERM -> bool;

Description These are derived discriminator functions for the Z basic arithmetic operations. See the documentation for the destructor functions (dest_z_plus etc.) for more information.
val mk_z_\leq : TERM * TERM -> TERM;
val mk_z_\geq : TERM * TERM -> TERM;
val mk_z_abs : TERM -> TERM;
val mk_z_div : TERM * TERM -> TERM;
val mk_z_greater : TERM * TERM -> TERM;
val mk_z_less : TERM * TERM -> TERM;
val mk_z_minus : TERM -> TERM;
val mk_z_mod : TERM * TERM -> TERM;
val mk_z_plus : TERM * TERM -> TERM;
val mk_z_signed_int : INTEGER -> TERM;
val mk_z_subtract : TERM * TERM -> TERM;
val mk_z_times : TERM * TERM -> TERM;

Description
These are derived constructor functions for the Z basic arithmetic operations. See
the documentation for the destructor functions (dest_z_plus etc.) for more information.

Errors
86201 ?0 does not have type Z

val z_cov_induction_tac : TERM -> TACTIC

Description
A course of values induction tactic for a subset of the integers. To prove \( j \leq x \Rightarrow t \),
it suffices to prove \( t[i/x] \) on the assumptions that \( j \leq i \) and \( \forall k \cdot j \leq k \land k < i \Rightarrow t[k/x] \).

(Course of values induction is sometimes called complete induction.) The term argument must
appear free in the conclusion of the goal. It must also appear once, and only once, in the
assumptions, in an assumption of the form \( j \leq x \).

\[
\begin{align*}
\{ \Gamma, j \leq x \} & \quad t[x] \\
\{ \Gamma, j \leq x \} & \quad t[j/x] ;
\end{align*}
\]

See Also
z_Z_cases_thm, z_intro_\forall_tac, z_Z_induction_tac,
z_Z_induction_tac, z_\leq_induction_tac

Errors
As for z_\leq_induction_tac.
Description  The first of these two conversions simplifies certain *dots* terms, the second, given a membership of a *dots* expression, first tries the simplifications, and whether or not that succeeds, expands the membership.

\[
\begin{align*}
\text{Conversion} & \quad \vdash (x \ldots x) = \{x\} \\
& \quad \text{\texttt{z.simple.dot.dot.conv} \quad \text{\texttt{\{}x \ldots x\texttt{\}}} }
\end{align*}
\]

and

\[
\begin{align*}
\text{Conversion} & \quad \vdash (n1 \ldots n2) = \{} \\
& \quad \text{\texttt{z.simple.dot.dot.conv} \quad \text{\texttt{\{}n1 \ldots n2\texttt{\}}} }
\end{align*}
\]

where \( n1 \) is a numeric literal less than the numeric literal \( n2 \).

\[
\begin{align*}
\text{Conversion} & \quad \vdash x \in y \ldots y \Leftrightarrow x = y \\
& \quad \text{\texttt{z.\in.dot.dot.conv} \quad \text{\texttt{\{}x \in y \ldots y\texttt{\}}} }
\end{align*}
\]

\[
\begin{align*}
\text{Conversion} & \quad \vdash x \in n1 \ldots n2 \Leftrightarrow \text{false} \\
& \quad \text{\texttt{z.\in.dot.dot.conv} \quad \text{\texttt{\{}x \in n1 \ldots n2\texttt{\}}} }
\end{align*}
\]

where \( n1 \) is a numeric literal less than the numeric literal \( n2 \).

\[
\begin{align*}
\text{Conversion} & \quad \vdash x \in \text{low} \ldots \text{high} \Leftrightarrow \text{low} \leq x \land x \leq \text{high} \\
& \quad \text{\texttt{z.\in.dot.dot.conv} \quad \text{\texttt{\{}x \in \text{low} \ldots \text{high}\texttt{\}}} }
\end{align*}
\]

See Also  \texttt{z.dot.dot.conv}

Errors

86001  ?0 is not of the form: \texttt{\{}low \ldots high\texttt{\}} where the expression can be simplified
86002  ?0 is not of the form: \texttt{\{}x \in low \ldots high\texttt{\}}
Description These are the ML value bindings for the theorems saved in the theory \textit{z\_numbers}.
9.2. Theory Related ML Values

SML
define val \( z_{\leq\text{induction tac}} : \text{TERM} \rightarrow \text{TACTIC} \)

**Description** An induction tactic for a subset of the integers. To prove \( j \leq x \Rightarrow t \), it suffices to prove \( t[j/x] \) and to prove \( t[x+1/x] \) on the assumptions \( t \) and \( j \leq x \). The term argument must be a variable of type \( \Gamma : \mathbb{Z} \) and must appear free in the conclusion of the goal. It must also appear once, and only once in the assumptions, in an assumption of the form \( j \leq x \).

Tactic
\[
\frac{\{ \Gamma, j \leq x \} \ t[x]}{\{ \Gamma, j \leq x \} \ t[j/x] ; \text{strip} \ \{t[x], j \leq x, \Gamma\} \ t[x+1]} \ z_{\leq\text{induction tac}} \ \frac{\Gamma}{x : \mathbb{Z}}
\]

**See Also** \( z_{\mathbb{Z}\text{cases thm}}, z_{\text{intro} \forall \text{tac}}, z_{\mathbb{N}\text{induction tac}} \), \( z_{\mathbb{Z}\text{induction tac}}, z_{\text{cov} \text{induction tac}} \)

**Errors**
- 86401 \(?0 is not a variable of type \( \Gamma : \mathbb{Z} \)\)
- 86402 A term of the form \( \forall j \leq i \) where \( i \) is the induction variable could not be found in the assumptions
- 86403 \(?0 appears free in more than one assumption of the goal\)
- 86404 \(?0 does not appear free in the conclusions of the goal\)

SML
define val \( z_{\mathbb{N}\text{induction tac}} : \text{TACTIC} \)

**Description** This tactic implements induction over the natural numbers in \( \mathbb{Z} \): to prove \( x \in \mathbb{N} \Rightarrow t \), it suffices to prove \( t[0/x] \) and to prove \( t[x+1/x] \) on the assumption that \( t \). The conclusion of the goal must have the form \( x \in \mathbb{N} \Rightarrow t \).

Tactic
\[
\frac{\{ \Gamma \} \ x \in \mathbb{N} \Rightarrow t}{\{ \Gamma \} \ t[0/x] ; \text{strip} \{t, \Gamma\} \ t[x+1/x]} \ z_{\mathbb{N}\text{induction tac}}
\]

**See Also** \( z_{\mathbb{Z}\text{cases thm}}, z_{\text{intro} \forall \text{tac}}, z_{\mathbb{Z}\text{induction tac}}, z_{\leq\text{induction tac}}, z_{\text{cov} \text{induction tac}} \)

**Errors** As for \( \text{gen\_induction\_tac1} \).
These conversions are used to perform evaluation of arithmetic expressions involving numeric literal operands. The normal interface to the conversion is via the proof context 'z.numbers' and other proof contexts which contain it.

The first block above gives conversions to evaluate expressions of the form $i \ op \ j$ where $i$ and $j$ are numeric literals and $\ op$ is one of + or *. The second block gives conversions to transform terms of the form $i - j$, $i > j$, $i \geq j$ and $i \in \mathbb{N}$ into $i + j$, $j < i$, $j \leq i$ and $0 \leq i$ respectively. The third block give conversions which evaluate expressions of the form $i \ op \ j$ or $\text{abs} \ i$, where $\ op$ is one of +, *, div, mod, $\leq$, $<$, or $=$, and where $i$ and $j$ are signed integer literals (i.e., either numeric literals or of the form $\sim k$, where $k$ is a numeric literal). Thus the second block of conversions transform expressions of the form $i - j$, $i > j$, $i \geq j$ and $i \in \mathbb{N}$ into a form which can be evaluated by the conversions in the third block if $i$ and $j$ are signed literals.

Errors

86301 \( ?0 \) is not of the form \( \?\!1 \) where \( \mathbb{Z}i \) and \( \mathbb{Z}j \) are numeric literals
86302 \( ?0 \) is not of the form \( \?\!1 \)
86303 \( ?0 \) is not of the form \( \?\!1 \) where \( \mathbb{Z}i \) and \( \mathbb{Z}j \) are optionally signed numeric literals

These are the ML bindings of the definitions of the theory $z.numbers$. 

Description

SML

\begin{verbatim}
| val z.N_plus_conv : CONV; | val z.N_times_conv : CONV;
| val z.N_subtract_minus_conv : CONV; | val z.N_greater_less_conv : CONV;
| val z.N_geq_leq_conv : CONV; | val z.N_eq_conv : CONV;
| val z.N_plus_conv : CONV; | val z.N_times_conv : CONV;
| val z.N_abs_conv : CONV; | val z.N_div_conv : CONV;
| val z.N_mod_conv : CONV; | val z.N_leq_conv : CONV;
| val z.N_less_conv : CONV | val z.N_eq_conv : CONV |
\end{verbatim}

86301 \( ?0 \) is not of the form \( \?\!1 \) where \( \mathbb{Z}i \) and \( \mathbb{Z}j \) are numeric literals
86302 \( ?0 \) is not of the form \( \?\!1 \)
86303 \( ?0 \) is not of the form \( \?\!1 \) where \( \mathbb{Z}i \) and \( \mathbb{Z}j \) are optionally signed numeric literals

\begin{verbatim}
| val z.Z_def : THM; | val z.N_def : THM;
| val z.arith_def : THM; | val z.inequality_def : THM;
| val z.N1_def : THM; | val z.succ_def : THM;
| val z.iter_def : THM; | val z.dot_dot_def : THM;
| val z.F_def : THM; | val z.F1_def : THM;
| val z.hash_def : THM; | val z.\sim{\rightarrow}\sim{\rightarrow} : THM;
| val z.min_def : THM; | val z.min_def : THM;
| val z.max_def : THM; | val z.min_def : THM; |
\end{verbatim}

Description

These are the ML bindings of the definitions of the theory $z.numbers$. 

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
9.2. Theory Related ML Values

val \texttt{z\_Z\_induction\_tac} : \texttt{TERM \rightarrow TACTIC}

\textbf{Description} An induction-like tactic for the integers, based on the fact that any subset of the integers containing 1 and closed under negation and addition must contain every integer.

\textbf{Tactic} \[
\frac{\{ \Gamma \} \ t \quad \{ \Gamma \} \ t[i/x] ; \quad \text{strip}(t[i/x], \Gamma) \ t[\sim i/x] ; \quad \text{strip}(t[i/x] \land t[j/x], \Gamma) \ t[i+j/x]}{\ z\_Z\_induction\_tac \ \check{\ z\_x}}
\]

\textbf{See Also} \texttt{z\_Z\_cases\_thm}, \texttt{z\_intro\_\forall\_tac}, \texttt{z\_N\_induction\_tac}, \texttt{z\_\le\_induction\_tac}, \texttt{z\_cov\_induction\_tac}

\textbf{Errors} As for \texttt{gen\_induction\_tac}.

9.2.5 \texttt{Z} Arithmetic Proof Support

\textbf{signature} \texttt{ZArithmeticTools} = \textit{sig}

\textbf{Description} This is the signature of a structure containing arithmetic and an automatic linear arithmetic prover for the integers in \texttt{Z}.

\begin{verbatim}
(* Proof Context: z_lin_arith *)
(* Proof Context: z_lin_arith1 *)

\textbf{Description} \texttt{"z\_lin\_arith"} is a proof context whose main purpose is to supply a decision procedure for problems in linear arithmetic in \texttt{Z}. \texttt{"z\_lin\_arith1"} differs from it only by using "\texttt{\_numbers1}". The proof context provides an interface to the proof context \texttt{\_lin\_arith} which provides the analogous facilities for the HOL integers.

\textbf{Contents} The proof context is the result of merging \texttt{z\_predicates}, \texttt{"\_numbers(1)} and \texttt{\_lin\_arith}.

\textbf{Examples} \texttt{PC\_T1\"z\_lin\_arith\"prove\_tac[]} will prove any of the following goals:

\begin{itemize}
  \item \([\[], \forall a, b, c:Z a \leq b \land (a + b < c + a) \Rightarrow a < c]\)
  \item \([\[], \forall a, b, c:Z a \geq b \land \sim b < c \Rightarrow a \geq c]\)
  \item \([\[], \forall a, b, c:Z a + 2*b < 2*a \Rightarrow b + b < a]\)
  \item \([\[], \forall x, y:Z \sim (2*x + y = 4 \land 4*x + 2*y = 7)\]
\end{itemize}

In the following example, an induction reduces the problem to linear arithmetic, and then the automatic proof tactic in \texttt{z\_lin\_arith} completes the proof.

\begin{verbatim}
set_goal([], \forall b:N (b + 1)*(b + 1) > 0); a(PC\_T1 "z\_library" REPEAT strip_tac); a( z\_\le\_induction\_tac \check{\ b} THEN PC\_T1 "z\_lin\_arith" \_asm\_prove\_tac[]));
pop_thm();
\end{verbatim}

\textbf{See Also} \texttt{\_lin\_arith}

\textbf{Errors} The errors reported by the automatic proof tactic are as for \texttt{\_lin\_arith}.
\begin{verbatim}
(val z_anf_conv : CONV;

Description  z_anf_conv is a conversion which proves theorems of the form \( \vdash t1 = t2 \) where
\( t1 \) is a Z expression formed from atoms of type \( \mathbb{Z} \) and \( t2 \) is in what we may call additive normal
form, i.e. it has the form: \( t1 + t2 + ... \), where the \( t_i \) have the form \( s_1 * s_2 * ... \) where the
\( s_i \) are atoms. Here, by atom we mean an expression which is not of the form \( t1 + t2 + ... \) or
\( s_1 * s_2 * ... \).

The summands \( t_i \) and, within them, the factors \( s_j \) are given in increasing order with respect to
the ordering on terms analogous to that given by the function \( \text{z_term_order} \), q.v. Arithmetic
computation is carried out on atoms to ensure that at most one of the summands is a numeric
literal and that, within each summand, at most one factor is a numeric literal. Any literal appears
at the beginning of its factor or summand and addition of \( 0 \) or multiplication by \( 1 \) is simplified
out. Any signs are moved to the first factor in each summand.

The conversion fails with error number 106010 if there no changes can be made to the term.

Errors  106010?0 is not of type \( \vdash \mathbb{Z} \) or is already in additive normal form
\end{verbatim}
9.2. Theory Related ML Values

Description  In the theory \textit{z\_arithmetic\_tools}, isomorphisms, \textit{z\_Z} and \textit{Z\_z}, are defined between the \textit{Z} integers and the HOL integers. These may be used to translate an arithmetic problem in \textit{Z} into one in HOL. These conversions implement this translation and its inverse.

The operators handled by the conversions are $+$, $\ast$, $\sim$ and $\neg$.

The translation to HOL is carried out according to the following scheme:

\[
\begin{align*}
\text{\textit{z\_Z\_conv}} : & \quad \text{CONV} \\
\text{\textit{Z\_z\_conv}} : & \quad \text{CONV} \\
\end{align*}
\]

\[
\begin{align*}
& \text{\textit{z\_Z\_plus}} : \quad \text{THM} \\
& \text{\textit{Z\_z\_plus}} : \quad \text{THM} \\
& \text{\textit{z\_Z\_times}} : \quad \text{THM} \\
& \text{\textit{Z\_z\_times}} : \quad \text{THM} \\
& \text{\textit{z\_Z\_subtract}} : \quad \text{THM} \\
& \text{\textit{Z\_z\_subtract}} : \quad \text{THM} \\
& \text{\textit{z\_Z\_minus}} : \quad \text{THM} \\
& \text{\textit{Z\_z\_minus}} : \quad \text{THM} \\
& \text{\textit{z\_Z\_one}} : \quad \text{THM} \\
& \text{\textit{Z\_z\_one}} : \quad \text{THM} \\
& \text{\textit{z\_Z\_less}} : \quad \text{THM} \\
& \text{\textit{Z\_z\_less}} : \quad \text{THM} \\
\end{align*}
\]

These are the Standard ML bindings for the theorems saved in the theory \textit{z\_arithmetic\_tools} which provides isomorphisms between the ring of integers in HOL and the ring of integers in \textit{Z}. The main purpose of this theory is to allow proof procedures for HOL integers to be adapted to work with \textit{Z}. The most common way of using these isomorphisms is via the proof context \textit{z\_lin\_arith}, q.v.
9.2.6 Z Sequences

**SML**

```sml
signature ZSequences = sig

Description  This provides the basic proof support for the Z library sequences. It creates the theory z.sequences.
```

```
val z_seq_def : THM;
val z_seq1_def : THM;
val z_iseq_def : THM;
val z__def : THM;
val z_head_def : THM;
val z_last_def : THM;
val z_tail_def : THM;
val z_front_def : THM;
val z_rev_def : THM;
val z_squash_def : THM;
val z_\_def : THM;
val z_\_/def : THM;
val z_disjoint_def : THM;
val z_partition_def : THM;
```

Description  These are the ML bindings of the definitions of the theory of z.sequences.

9.2.7 Z Finiteness and Sequences

**SML**

```sml
signature ZFunctions1 = sig

Description  This provides additional proof support for the Z library functions. It creates the theory z.functions1.
```

```sml
signature ZNumbers1 = sig

Description  This provides additional proof support for the Z library functions. It creates the theory z.functions1.
```

```sml
signature ZSequences1 = sig

Description  This provides additional proof support for the Z library sequences. It creates the theory z.sequences1.
```
9.2. Theory Related ML Values

**Description** These are the ML bindings of the theorems in the theory \textit{z.numbers1}.

```
SML
val z_dot_dot_clauses : THM;
val z_less_cases_thm : THM;
val z_dot_dot_diff_thm : THM;
val z_dot_dot_inter_thm : THM;
val z_F_inter_singleton_thm : THM;
val z_F_induction_thm : THM;
val z_F_size_thm1 : THM;
val z_size_empty_thm : THM;
val z_size_dot_dot_thm : THM;
val z_size_seq_thm : THM;
val z_F_inter_thm : THM;
val z_size_diff_thm : THM;
val z_size_mono_thm : THM;
val z_size_eq_thm : THM;
val z_size_1_thm : THM;
val z_size_pair_thm : THM;
val z_size_less_1_thm : THM;
val z_pigeon_hole_thm : THM;
val z_div_thm : THM;
val z_abs_pos_thm : THM;
val z_abs_le_times_thm : THM;
val z_0_less_times_thm : THM;
val z_in_succ_thm : THM;
val z_suc0_thm : THM;
```

**SML**

\[ \text{val } z \text{._dot.dot.conv : CONV;} \]

**Description** This conversion expands a range between two integer literals into a set display:

**Example**

\[ \text{z\_dot\_dot\_conv } 1 .. 5 \Rightarrow \text{ gives} \]

\[ \vdash 1 .. 5 = \{1, 2, 3, 4, 5\} \]

**Errors**

\[ 107002?0 \text{ is not of the form } \frac{a}{b} \Rightarrow \text{ where } \frac{a}{b} \text{ and } \frac{a}{b} \text{ are integer literals} \]
val _seqd_app_conv : CONV;
val _size_seqd_conv : CONV;
val _seqd_eq_conv : CONV;

**Description**  Conversions for sequence displays.

_z_seqd_app_conv applies to terms of the form \( sm \), where \( s \) is a sequence display and \( m \) is a numeric literal.

_z_size_seqd_conv

**Description**  applies to terms of the form \#s, where \( s \) is a sequence display.

_z_seqd_eq_conv

**Description**  applies to terms of the form \( s_1 = s_2 \), where \( s_1 \) and \( s_2 \) are sequence displays.

**Errors**

107011?0 is not of the form \( \langle t1, ... \rangle \)
107012?0 is not a positive integer literal
107013?0 is not a valid index for the sequence \( ?1 \)
107020?0 is not of the form \( \langle u1, ... \rangle \)
107021?0 is not of the form \( \langle \#(t1, ...) \rangle \)

val _seq_induction_tac : TERM -> TACTIC;
val _seq_induction_tac1 : TERM -> TACTIC;

**Description**  Induction tactics for Z sequences. To prove \( s \in \text{seq } A \Rightarrow t \), it suffices to prove \( t[\langle \rangle / s] \) and to prove \( t[s \cap \langle x \rangle / s] \) on the assumptions \( t, s \in \text{seq } A \) and \( x \in A \). The term argument must be a variable of the same type as a Z sequence and must appear free in the conclusion of the goal. It must also appear once, and only once, in an assumption of the form \( s \in \text{seq } A \).

Tactic

\[
\begin{align*}
\{ \Gamma, s \in \text{seq } A \} & \quad t[s] \\
\{ \Gamma \} & \quad t[\langle \rangle / s] \\
\text{strip } \{ t, s \in \text{seq } A, x \in A, \Gamma \} & \quad t[s \cap \langle x \rangle / s]
\end{align*}
\]

**Errors**

107031 A term of the form \( \langle s \rangle \in \text{seq } A \) where \( s \) is the induction variable could not be found in the assumptions
107032?0 is not a variable
9.2. Theory Related ML Values

SML

val z_size_dot_dot_conv : CONV;

Description  This conversion will calculate the size of a range between two integer literals, including the empty range case when the end of the range is less than the start.

Example

\[\text{z_size_dot_dot_conv } \langle 1 \ldots 5 \rangle \rightarrow \text{gives} \quad \# (1 \ldots 5) = 5\]

\[\text{z_size_dot_dot_conv } \langle 10 \ldots 1 \rangle \rightarrow \text{gives} \quad \# (10 \ldots 1) = 0\]

Errors

107001?0 is not of the form \(\langle a \ldots b \rangle\) where \(\langle a \rangle\) and \(\langle b \rangle\) are integer literals

SML

val z_empty_thm : THM;
val z_dom_thm : THM;
val z_dom_seq_thm : THM;
val z_assoc_thm : THM;
val z_one_one_thm : THM;
val z_el_seq_thm : THM;
val z_el_seq_thm1 : THM;
val z_seq_x_thm : THM;
val z_singleton_thm : THM;
val z_seq_thm : THM;
val z_seq_thm1 : THM;
val z_seq Singleton_thm : THM;
val z_seq u_thm : THM;
val z_seq_seq_thm : THM;
val z_seq_seq_length_thm : THM;
val z_size_seq_length_thm : THM;
val z_size_seq_thm1 : THM;
val z_size_seq_thm2 : THM;
val z_size_singleton_seq_thm : THM;
val z_seqd_eq_thm : THM;

Description  These are the ML bindings of the theorems in the theory \(z\_sequences1\).
Lemma 9.2.8 Z Bags

Errors

A term of the form \( \exists s \in F A \) where \( s \) is the induction variable could not be found in the assumptions

These are the ML bindings of the theorems in the theory \( z \cdot \text{functions1} \).

9.2.8 Z Bags

These provide the basic proof support for the Z library bags. It creates the theory \( z \cdot \text{bags} \).

\( \begin{align*}
\textbf{Description} & \quad \text{This provides the basic proof support for the Z library bags. It creates the theory } z \cdot \text{bags.}
\end{align*} \)
9.2. Theory Related ML Values

**signature ZLibrary = sig**

**Description** This provides a “marker” theory, indicating the “top” of the Z library theories. It creates the theory `z_library`.

As a side effect, loading this structure will set the current theory to `z_library`, the current proof context to “z_library”, and tidies the subgoal package and proof context stacks.

**SML (∗ Proof Context: z_library ∗)**

**Description** A mild complete proof context for handling the manipulation of Z language and library expressions and predicates. Its contents are chosen to be “uncontroversial”. That is, any effect is considered to be “almost always the correct thing”.

It consists of the merge of the proof contexts:

- "z_sets_alg", – simplification of set constructs, and Z language
- "z_rel_alg", – simplification of relational constructs
- "z_fun_alg", – simplification of function constructs
- "z_numbers" – simplification of numeric constructs

**Usage Notes** It requires theory `z_bags`.

It is not intended to be mixed with HOL proof contexts or “z_library_ext”, which offers an aggressive approach.

**SML (∗ Proof Context: z_library_ext ∗)**

**Description** A aggressive complete proof context for handling the manipulation of Z language and library expressions and predicates. Its purpose is to strip or rewrite its input into the Z predicate calculus.

It consists of the merge of the proof contexts:

- "z_fun_ext", – extensional reasoning about functions (and relations and sets)
- "z_numbers" – simplification of numeric constructs

**Usage Notes** It requires theory `z_bags`.

It is not intended to be mixed with HOL proof contexts or “z_library_ext”, which offers an aggressive approach.
SML
(* Proof Context: z_library1 *)

Description  A mild complete proof context for handling the manipulation of Z language and library expressions and predicates. Its contents are chosen to be “uncontroversial”. That is, any effect is considered to be “almost always the correct thing”.

It differs from z_library only in using z_numbers1.

It consists of the merge of the proof contexts:

"z_sets_alg",  ¬ simplification of set constructs, and Z language
"z_rel_alg",  ¬ simplification of relational constructs
"z_fun_alg",  ¬ simplification of function constructs
"z_numbers1"  ¬ simplification of numeric constructs

Usage Notes  It requires theory z_bags.

It is not intended to be mixed with HOL proof contexts or “z_library_ext”, which offers an aggressive approach.

SML
(* Proof Context: z_library1_ext *)

Description  A aggressive complete proof context for handling the manipulation of Z language and library expressions and predicates. Its purpose is to strip or rewrite its input into the Z predicate calculus.

It differs from z_library only in using z_numbers1.

It consists of the merge of the proof contexts:

"z_fun_ext",  ¬ extensional reasoning about functions (and relations and sets)
"z_numbers1"  ¬ simplification of numeric constructs

Usage Notes  It requires theory z_bags.

It is not intended to be mixed with HOL proof contexts or “z_library_ext”, which offers an aggressive approach.

SML
val z_bag_def : THM;  val z_count_def : THM;
val z_in_def : THM;  val z_def : THM;
val z_items_def : THM;

Description  These are the definitions of the Z bag theory.
9.2.9 Z Reals

SML

(* Proof Context: 'z_reals *)

Description A component proof context for handling the basic arithmetic operations for real numbers in Z.

Expressions and predicates treated by this proof context are constructs formed from:


Contents

Rewriting:

\[ z\_R\_plus\_conv, z\_R\_times\_conv, z\_R\_subtract\_conv \]
\[ z\_R\_abs\_conv, z\_R\_div\_conv, z\_R\_mod\_conv \]
\[ z\_R\_eq\_conv, z\_R\_\leq\_conv, z\_R\_less\_conv \]
\[ z\_R\_\geq\_conv, z\_R\_greater\_conv, \]
\[ z\_R\_plus\_clauses, z\_R\_minus\_clauses, z\_R\_\leq\_clauses \]
\[ z\_R\_less\_clauses, z\_R\_lit\_norm\_conv \]

Stripping theorems:

\[ z\_R\_eq\_conv, z\_R\_\leq\_conv, z\_R\_less\_conv \]
\[ z\_R\_\geq\_conv, z\_R\_greater\_conv, \]
\[ z\_R\_plus\_clauses, z\_R\_minus\_clauses, z\_R\_\leq\_clauses \]
\[ z\_R\_less\_clauses, \]
\[ and \ all \ the \ above \ pushed \ through \ \neg \]

Stripping conclusions: as for stripping theorems.

Rewriting canonicalisation: blank.

Automatic proof procedures: \textit{z_basic_prove_tac}, \textit{z_basic_prove_conv}.

Automatic existence prover: blank.

SML

(* Proof Context: z\_R\_lin_arith *)

Description This is a component proof context whose main purpose is to supply a decision procedure for problems in linear arithmetic for the real numbers in Z.

Contents The rewriting components converts Z real arithmetic expressions into equivalent HOL ones and the automatic proof tactic then uses the HOL linear arithmetic proof context to attempt the proof.
Description

These are derived destructor functions for the Z basic arithmetic operations. An optionally signed integer literal, \texttt{signed\_int}, is taken to be either a numeric literal or the result of applying \((\sim\cdot)\) to a numeric literal. The other constructors correspond directly to the arithmetic operations of the theory \texttt{z\_numbers} with alphabetic names assigned to give a valid ML name as needed (\texttt{greater} \texttt{\_\_\_}, \texttt{less} \texttt{\_\_\_}, \texttt{minus} \texttt{\_\_\_}, \texttt{plus} \texttt{\_\_\_}, \texttt{subtract} \texttt{\_\_\_}, \texttt{times} \texttt{\_\_\_}).

As usual, there are also corresponding discriminator (\texttt{is\_\_\_\_}) and constructor functions (\texttt{mk\_\_\_\_}).

\begin{verbatim}
SML
val dest_z_R_\leq : TERM -> TERM * TERM;
val dest_z_R_\geq : TERM -> TERM * TERM;
val dest_z_R_z_exp : TERM -> TERM * TERM;
val dest_z_R_abs : TERM -> TERM;
val dest_z_R_frac : TERM -> TERM * TERM;
val dest_z_R_greater : TERM -> TERM * TERM;
val dest_z_R_less : TERM -> TERM * TERM;
val dest_z_R_minus : TERM -> TERM;
val dest_z_R_over : TERM -> TERM * TERM;
val dest_z_R_plus : TERM -> TERM * TERM;
val dest_z_R_real : TERM -> TERM;
val dest_z_R_subtract : TERM -> TERM * TERM;
val dest_z_R_times : TERM -> TERM * TERM;

val is_z_R_\leq : TERM -> bool;
val is_z_R_\geq : TERM -> bool;
val is_z_R_z_exp : TERM -> bool;
val is_z_R_abs : TERM -> bool;
val is_z_R_frac : TERM -> bool;
val is_z_R_greater : TERM -> bool;
val is_z_R_less : TERM -> bool;
val is_z_R_minus : TERM -> bool;
val is_z_R_over : TERM -> bool;
val is_z_R_plus : TERM -> bool;
val is_z_R_real : TERM -> bool;
val is_z_R_subtract : TERM -> bool;
val is_z_R_times : TERM -> bool;
\end{verbatim}

\textbf{Errors}

\begin{verbatim}
117101?0 is not of the form \(\frac{z}{R}x \leq_R y\)
117102?0 is not of the form \(\frac{z}{R}x \geq_R y\)
117103?0 is not of the form \(\frac{z}{R}abs_R x\)
117104?0 is not of the form \(\frac{z}{R}x /_R y\)
117105?0 is not of the form \(\frac{z}{R}x >_R y\)
117106?0 is not of the form \(\frac{z}{R}x <_R y\)
117107?0 is not of the form \(\frac{z}{R}x \sim_R x\)
117109?0 is not of the form \(\frac{z}{R}x +_R y\)
117110?0 is not of the form \(\frac{z}{R}x /_Z y\)
117111?0 is not of the form \(\frac{z}{R}x -_R y\)
117112?0 is not of the form \(\frac{z}{R}x *_R y\)
117113?0 is not of the form \(\frac{z}{R}real x\)
\end{verbatim}

\textbf{Description}

These are derived discriminator functions for the Z basic arithmetic operations. See the documentation for the destructor functions (\texttt{dest\_z\_plus} etc.) for more information.
9.2. Theory Related ML Values

SML

\begin{verbatim}
val mk_z_R_≤ : TERM * TERM -> TERM;
val mk_z_R_≥ : TERM * TERM -> TERM;
val mk_z_R_Z_exp : TERM * TERM -> TERM;
val mk_z_R_abs : TERM -> TERM;
val mk_z_R_frac : TERM * TERM -> TERM;
val mk_z_R_greater : TERM * TERM -> TERM;
val mk_z_R_less : TERM * TERM -> TERM;
val mk_z_R_over : TERM * TERM -> TERM;
val mk_z_R_plus : TERM * TERM -> TERM;
val mk_z_R_real : TERM -> TERM;
val mk_z_R_subtract : TERM * TERM -> TERM;
val mk_z_R_times : TERM * TERM -> TERM;
\end{verbatim}

Description These are derived constructor functions for the Z basic arithmetic operations. See the documentation for the destructor functions (dest_z_plus etc.) for more information.

Errors

117201?0 does not have type \( \mathbb{R} \)

SML

\begin{verbatim}
val z_float_conv : CONV;
\end{verbatim}

Description The conversion \( z\_float\_conv \) converts a floating point literal into a normalised real literal form.

Errors

117006?0 is not a Z floating point literal

SML

\begin{verbatim}
val z_R_complete_thm : THM;
val z_R_unbounded_above_thm : THM;
val z_R_unbounded_below_thm : THM;
val z_R_less_antisyms_thm : THM;
val z_R_less_cases_thm : THM;
val z_R_less_clauses : THM;
val z_R_less_dense_thm : THM;
val z_R_less_irrefl_thm : THM;
val z_R_less_thm : THM;
val z_R_less_trans_thm : THM;
\end{verbatim}

Description These are ML bindings for the theorems that characterise the ordering relation \( _{\leq_R} \) on the real numbers.
These are ML bindings for theorems that deal with the equality and ordering relations.

*z_R_eq_≤_thm*: THM;

*z_R_eq_thm*: THM;

*z_R_less_≤_trans_thm*: THM;

*z_R_less_¬_eq_thm*: THM;

*z_R_≤_¬_less_thm*: THM;

*z_R_≤_antisym_thm*: THM;

*z_R_≤_cases_thm*: THM;

*z_R_≤_clauses*: THM;

*z_R_≤_less_cases_thm*: THM;

*z_R_≤_less_trans_thm*: THM;

*z_R_≤_refl_thm*: THM;

*z_R_≤_thm*: THM;

*z_R_≤_trans_thm*: THM;

*z_R_¬_≤_thm*: THM;

*z_R_¬_less_thm*: THM;

*z_R_0_less_0_less_times_thm*: THM;

*z_R_greater_thm*: THM;

*z_R_≥_thm*: THM;

Description These are ML bindings for theorems that deal with the equality and ordering relations.

*z_R_eval_conv*: CONV;  

*z_R_EVAL_C*: CONV → CONV;

Description *z_R_eval_conv* computes theorems of the form \( \Gamma \vdash t1 = t2 \) where \( t1 \) is an expression made up from rational literals (see *z_R_plus_conv*) using real addition, subtraction, multiplication, division, reciprocal, absolute value and unary negation. \( t2 \) will be an optionally signed rational literal in normal form. The conversion fails if the expression cannot be evaluated (e.g., because it contains variables).

*z_R_EVAL_C_conv* is similar to *R_eval_conv* but it also applies *conv* to any subterm that cannot be evaluated using the conversions for the arithmetic operations listed above. E.g., *z_R_EVAL_C z_R_Z_exp_conv* will evaluate expressions involving the usual arithmetic operations and also exponentiation of rational literals by natural number literals.

Errors 117020?0 cannot be evaluated

*z_R_lin_arith_prove_conv*: THM list → CONV;

*z_R_lin_arith_prove_tac*: THM list → TACTIC;

Description This conversion and tactic implement the linear arithmetic decision procedure for real numbers. The usual interface to these is via the proof context *z_reals*, q.v.
9.2. Theory Related ML Values

SML

val z\_\_R\.minus\_clauses : THM;
val z\_\_R\.minus\_eq\_thm : THM;
val z\_\_R\.minus\_thm : THM;
val z\_\_R\.plus\_0\_thm : THM;
val z\_\_R\.plus\_assoc\_thm : THM;
val z\_\_R\.plus\_assoc\_thm1 : THM;
val z\_\_R\.plus\_eq\_thm : THM;
val z\_\_R\.plus\_minus\_thm : THM;
val z\_\_R\.plus\_mono\_thm : THM;
val z\_\_R\.plus\_mono\_thm1 : THM;
val z\_\_R\.plus\_mono\_thm2 : THM;
val z\_\_R\.plus\_order\_thm : THM;
val z\_\_R\.plus\_thm : THM;
val z\_\_R\.plus\_unit\_thm : THM;
val z\_\_R\.subtract\_thm : THM;

Description  ML bindings for theorems about addition, unary minus and subtraction for the real numbers.

SML

val z\_\_R\.real\_NR\_thm : THM;
val z\_\_R\.real\_0\_thm : THM;
val z\_float\_thm : THM;

Description  ML bindings for theorems concerning Z integer and floating point real literals.

SML

val z\_\_R\.times\_assoc\_thm : THM;
val z\_\_R\.times\_assoc\_thm1 : THM;
val z\_\_R\.times\_clauses : THM;
val z\_\_R\.times\_comm\_thm : THM;
val z\_\_R\.times\_order\_thm : THM;
val z\_\_R\.times\_plus\_distrib\_thm : THM;
val z\_\_R\.times\_thm : THM;
val z\_\_R\.times\_unit\_thm : THM;
val z\_\_R\.over\_thm : THM;
val z\_\_R\.over\_clauses : THM;

Description  ML bindings for theorems about multiplication and division of real numbers.
val z\(_R\leq\)conv : CONV; (* _ ≤\(_R\)_ *)
val z\(_R\)eqconv : CONV; (* _ = \(_R\)_ *)
val z\(_R\)less_conv : CONV; (* _ <\(_R\)_ *)
val z\(_R\)minus_conv : CONV; (* _ ∼\(_R\)_ *)
val z\(_R\)over_conv : CONV; (* _ /\(_R\)_ *)
val z\(_R\)plus_conv : CONV; (* _ +\(_R\)_ *)
val z\(_R\)times_conv : CONV; (* _ *\(_R\)_ *)
val z\(_R\)Zexp_conv : CONV; (* _ \(\hat{Z}\)_ *)
val z\(_R\)abs_conv : CONV; (* abs\(_R\)_ *)
val z\(_R\)greater_conv : CONV; (* _ >\(_R\)_ *)
val z\(_R\)ge_conv : CONV; (* _ ≥ \(_R\)_ *)
val z\(_R\)subtract_conv : CONV; (* _ -\(_R\)_ *)
val z\(_R\)lit_norm_conv : CONV;

val z\(_R\)lit_conv : CONV; val z\(_R\)lit_conv1 : CONV;

Description These are conversions for carrying out real arithmetic computation. The first and second blocks of conversions deal with expressions of the form \(c \ op \ d\), where \(c\) and \(d\) are real literal expressions (see below) and where \(op\) is the operator given in the ML comment alongside the conversion above. The conversions in the first block actually carry out the computation to give a theorem \(c \ op \ d = e\) or \(c \ op \ d \equiv v\) where \(e\) and \(v\) are a real literal expression or a truth value as appropriate.

The conversions in the second block rewrite their argument in terms of the operators supported by the conversions in the first block.

The conversion \(z\(_R\)lit_norm_conv\) normalises real literal expressions, i.e., expressions of either of the forms \(\text{real } i\) or \(i \ /\(_Z\) j\), where \(i\) and \(j\) are optionally signed integer literals. The conversion puts the result in a normal form, where the sign if any is moved to the outside, where \(\text{real}\) is used whenever possible and where if the form \(i \ /\(_Z\) j\) has to be used, \(i\) and \(j\) are taken to be coprime. This conversion fails if its argument cannot be normalised or is already in the normal form.

The final two conversions \(z\(_R\)lit_conv\) and \(z\(_R\)lit_conv1\) convert to and from \(Z\) and HOL real literal expressions.

Errors

117001?: 0 is not a \(Z\) real fraction with integer literal operands
117002?: 0 is not an HOL real fraction with literal operands
117003?: 0 is not of the form \(?1\) where \(x\) and \(y\) are real literal expressions
117004?: 0 is not of the form \(?1\) where \(x\) is a real literal expression
117005?: 0 is not of the form \(\hat{Z} \ i\) where \(x\) is a real literal expression and \(i\) is an integer literal
Description  ML bindings for the definitions in the theory of real numbers.
REFERENCES


<table>
<thead>
<tr>
<th>Expression</th>
<th>Line Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>( -. )[X]</code></td>
<td>462</td>
<td><code>all_subterm_tt</code></td>
</tr>
<tr>
<td><code>( ~ )[X, Y]</code></td>
<td>478</td>
<td><code>ALL_VAR_ELIM_ASM_T1</code></td>
</tr>
<tr>
<td><code>( [ . ] )[X]</code></td>
<td>450</td>
<td><code>all_var_elim_asm_tac1</code></td>
</tr>
<tr>
<td><code>( &lt;= ! ][X]</code></td>
<td>489</td>
<td><code>all_var_elim_asm_tac</code></td>
</tr>
<tr>
<td><code>(H ?)</code></td>
<td>489</td>
<td><code>ALL_VAR_ELIM_ASM_T</code></td>
</tr>
<tr>
<td><code>( ~ - )</code></td>
<td>462</td>
<td><code>all_\beta_conv</code></td>
</tr>
<tr>
<td><code>**</code></td>
<td>28</td>
<td></td>
</tr>
<tr>
<td><code>= :</code></td>
<td>81</td>
<td><code>all_\beta_tac</code></td>
</tr>
<tr>
<td><code>= #</code></td>
<td>132</td>
<td><code>all_\epsilon_t</code></td>
</tr>
<tr>
<td><code>= $</code></td>
<td>80</td>
<td><code>ALL_\epsilon_T</code></td>
</tr>
<tr>
<td><code>@*</code></td>
<td>39</td>
<td><code>all_3_uncurry_cone</code></td>
</tr>
<tr>
<td><code>@+</code></td>
<td>39</td>
<td><code>all_Narb_elim</code></td>
</tr>
<tr>
<td><code>@-</code></td>
<td>39</td>
<td><code>all_Nelim</code></td>
</tr>
<tr>
<td><code>@ &lt; =</code></td>
<td>39</td>
<td><code>all_N_intro</code></td>
</tr>
<tr>
<td><code>@ &lt;=</code></td>
<td>39</td>
<td><code>all_N_intro</code></td>
</tr>
<tr>
<td><code>@ &gt; =</code></td>
<td>39</td>
<td><code>All_N_C</code></td>
</tr>
<tr>
<td><code>@ &gt;</code></td>
<td>39</td>
<td><code>All_N_C</code></td>
</tr>
<tr>
<td><code>@ @</code></td>
<td>39</td>
<td><code>All_\Rightarrow_intro</code></td>
</tr>
<tr>
<td><code>@ ~</code></td>
<td>39</td>
<td><code>All</code></td>
</tr>
<tr>
<td><code>abandon_reader_writer</code></td>
<td>59</td>
<td><code>AND_OR_C</code></td>
</tr>
<tr>
<td><code>abs</code></td>
<td>464</td>
<td><code>ANDF.C</code></td>
</tr>
<tr>
<td><code>abs_</code></td>
<td>473</td>
<td></td>
</tr>
<tr>
<td><code>accept_tac</code></td>
<td>231</td>
<td><code>antec_tac</code></td>
</tr>
<tr>
<td><code>add_error_codes</code></td>
<td>59</td>
<td><code>any_submatch_tt</code></td>
</tr>
<tr>
<td><code>add_error_code</code></td>
<td>59</td>
<td><code>any_substring_tt</code></td>
</tr>
<tr>
<td><code>add_general_reader</code></td>
<td>59</td>
<td><code>any</code></td>
</tr>
<tr>
<td><code>add_named_reader</code></td>
<td>59</td>
<td><code>apply_tactic</code></td>
</tr>
<tr>
<td><code>add_new_symbols</code></td>
<td>60</td>
<td></td>
</tr>
<tr>
<td><code>add_rw_thms</code></td>
<td>338</td>
<td><code>app_any_rule</code></td>
</tr>
<tr>
<td><code>add_sc_thms</code></td>
<td>338</td>
<td><code>APP.C</code></td>
</tr>
<tr>
<td><code>add_specific_reader</code></td>
<td>59</td>
<td><code>app_fun_rule</code></td>
</tr>
<tr>
<td><code>add_st_thms</code></td>
<td>339</td>
<td><code>app_if_cone</code></td>
</tr>
<tr>
<td><code>add_3_cd_thms</code></td>
<td>339</td>
<td><code>app</code></td>
</tr>
<tr>
<td><code>advance</code></td>
<td>58</td>
<td><code>area_of</code></td>
</tr>
<tr>
<td><code>all_asm_ante_tac</code></td>
<td>231</td>
<td><code>array</code></td>
</tr>
<tr>
<td><code>ALL_ASM_FC_T1</code></td>
<td>247</td>
<td><code>array</code></td>
</tr>
<tr>
<td><code>all_asm_fc_tac</code></td>
<td>245</td>
<td></td>
</tr>
<tr>
<td><code>ALL_ASM_FC</code></td>
<td>246</td>
<td><code>ask_at_terminal</code></td>
</tr>
<tr>
<td><code>ALL_ASM_FORWARD_CHAIN_T1</code></td>
<td>247</td>
<td><code>asm</code></td>
</tr>
<tr>
<td><code>all_asm_forward_chain_tac</code></td>
<td>245</td>
<td><code>asm_ante_tac</code></td>
</tr>
<tr>
<td><code>ALL_ASM_FORWARDCHAIN_T</code></td>
<td>246</td>
<td><code>asm_elim</code></td>
</tr>
<tr>
<td><code>all_different</code></td>
<td>20</td>
<td><code>ASM_FC_T</code></td>
</tr>
<tr>
<td><code>all_distinct</code></td>
<td>20</td>
<td><code>ASM_FORWARDCHAIN_T1</code></td>
</tr>
<tr>
<td><code>ALL_FC_T1</code></td>
<td>247</td>
<td><code>asm_fc_tac</code></td>
</tr>
<tr>
<td><code>all_fc_tac</code></td>
<td>245</td>
<td><code>asm_forward_chain_tac</code></td>
</tr>
<tr>
<td><code>ALL_FC</code></td>
<td>246</td>
<td><code>ASM_FORWARDCHAIN_T</code></td>
</tr>
<tr>
<td><code>ALL_FORWARDCHAIN_T1</code></td>
<td>247</td>
<td><code>asm_inst_term_rule</code></td>
</tr>
<tr>
<td><code>all_forward_chain_tac</code></td>
<td>245</td>
<td><code>asm_inst_type_rule</code></td>
</tr>
<tr>
<td><code>ALL_FORWARDCHAIN_T</code></td>
<td>246</td>
<td></td>
</tr>
<tr>
<td><code>all_simple_\beta_conv</code></td>
<td>154</td>
<td><code>asm_intro</code></td>
</tr>
<tr>
<td><code>all_simple_\beta_rule</code></td>
<td>154</td>
<td><code>ASM_PROP_EQ_T</code></td>
</tr>
<tr>
<td><code>ALL_SIMPLE_\beta_C</code></td>
<td>154</td>
<td><code>asm_prove_tac</code></td>
</tr>
<tr>
<td><code>ALL_SIMPLE_\forall_C</code></td>
<td>153</td>
<td><code>asm_prove_\exists_tac</code></td>
</tr>
<tr>
<td><code>all_simple_\forall_elim</code></td>
<td>154</td>
<td><code>asm_rewrite_rule</code></td>
</tr>
<tr>
<td><code>all_submatch_tt</code></td>
<td>152</td>
<td><code>asm_rewrite_tac</code></td>
</tr>
<tr>
<td><code>all_substring_tt</code></td>
<td>152</td>
<td><code>asm_rewrite_thm_tac</code></td>
</tr>
<tr>
<td><code>asm_rule</code></td>
<td>159</td>
<td></td>
</tr>
<tr>
<td>Index Item</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>BASIC_RESOLUTION_TI</td>
<td>307</td>
<td></td>
</tr>
<tr>
<td>BASIC_RESOLUTION_T</td>
<td>306</td>
<td></td>
</tr>
<tr>
<td>basic_resolve_rule</td>
<td>307</td>
<td></td>
</tr>
<tr>
<td>basic_res_extract</td>
<td>307</td>
<td></td>
</tr>
<tr>
<td>basic_res_next_to_process</td>
<td>308</td>
<td></td>
</tr>
<tr>
<td>basic_res_post</td>
<td>308</td>
<td></td>
</tr>
<tr>
<td>basic_res_pre</td>
<td>308</td>
<td></td>
</tr>
<tr>
<td>basic_res_solver</td>
<td>308</td>
<td></td>
</tr>
<tr>
<td>basic_res_rule</td>
<td>309</td>
<td></td>
</tr>
<tr>
<td>basic_res_subsumption</td>
<td>309</td>
<td></td>
</tr>
<tr>
<td>basic_res_tac1</td>
<td>310</td>
<td></td>
</tr>
<tr>
<td>basic_res_tac2</td>
<td>310</td>
<td></td>
</tr>
<tr>
<td>basic_res_tac3</td>
<td>311</td>
<td></td>
</tr>
<tr>
<td>basic_res_tac4</td>
<td>311</td>
<td></td>
</tr>
<tr>
<td>BASIC_RES_TYPE</td>
<td>305</td>
<td></td>
</tr>
<tr>
<td>be_tac</td>
<td>235</td>
<td></td>
</tr>
<tr>
<td>be_thm_tac</td>
<td>236</td>
<td></td>
</tr>
<tr>
<td>BdzFail</td>
<td>359</td>
<td></td>
</tr>
<tr>
<td>BdzFArgc</td>
<td>359</td>
<td></td>
</tr>
<tr>
<td>BdzFCode</td>
<td>359</td>
<td></td>
</tr>
<tr>
<td>BdzFCompc</td>
<td>359</td>
<td></td>
</tr>
<tr>
<td>BdzNonZ</td>
<td>359</td>
<td></td>
</tr>
<tr>
<td>BdzOk</td>
<td>359</td>
<td></td>
</tr>
<tr>
<td>BTags</td>
<td>359</td>
<td></td>
</tr>
<tr>
<td>before_kernel_state_change</td>
<td>133</td>
<td></td>
</tr>
<tr>
<td>BINDER_C</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>Binder</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>binbool_op</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>BOOL</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>CANON</td>
<td>153</td>
<td></td>
</tr>
<tr>
<td>can_input</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>CASES_T2</td>
<td>237</td>
<td></td>
</tr>
<tr>
<td>cases_tac</td>
<td>237</td>
<td></td>
</tr>
<tr>
<td>CASES_T</td>
<td>237</td>
<td></td>
</tr>
<tr>
<td>CHANGED_C</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>CHANGED_T</td>
<td>237</td>
<td></td>
</tr>
<tr>
<td>CharacterUtilities</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>char_conv</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>Char</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>CHAR</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>CHECKPOINT</td>
<td>132</td>
<td></td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest</td>
<td>80</td>
</tr>
<tr>
<td>delete</td>
<td>87</td>
</tr>
<tr>
<td>DeleteConst</td>
<td>131</td>
</tr>
<tr>
<td>DeleteTheory</td>
<td>364</td>
</tr>
<tr>
<td>DeleteType</td>
<td>365</td>
</tr>
<tr>
<td>delete_all_conjectures</td>
<td>366</td>
</tr>
<tr>
<td>delete_axiom</td>
<td>367</td>
</tr>
<tr>
<td>delete_conjecture</td>
<td>370</td>
</tr>
<tr>
<td>delete_const</td>
<td>366</td>
</tr>
<tr>
<td>delete_pc_fields</td>
<td>368</td>
</tr>
<tr>
<td>delete_pc</td>
<td>368</td>
</tr>
<tr>
<td>delete_theory</td>
<td>368</td>
</tr>
<tr>
<td>delete_thm</td>
<td>369</td>
</tr>
<tr>
<td>delete_to_level</td>
<td>369</td>
</tr>
<tr>
<td>delete</td>
<td>370</td>
</tr>
<tr>
<td>DerivedRules1</td>
<td>372</td>
</tr>
<tr>
<td>DerivedRules2</td>
<td>373</td>
</tr>
<tr>
<td>dest_app</td>
<td>374</td>
</tr>
<tr>
<td>dest_allocator</td>
<td>375</td>
</tr>
<tr>
<td>dest_binder</td>
<td>376</td>
</tr>
<tr>
<td>dest_bin_op</td>
<td>376</td>
</tr>
<tr>
<td>dest_char</td>
<td>377</td>
</tr>
<tr>
<td>dest_cons</td>
<td>373</td>
</tr>
<tr>
<td>dest_ctype</td>
<td>374</td>
</tr>
<tr>
<td>dest_dollar_quoted_string</td>
<td>375</td>
</tr>
<tr>
<td>dest_empty_list</td>
<td>376</td>
</tr>
<tr>
<td>dest_enum_set</td>
<td>377</td>
</tr>
<tr>
<td>dest_eq</td>
<td>378</td>
</tr>
<tr>
<td>dest_float</td>
<td>378</td>
</tr>
<tr>
<td>dest_if</td>
<td>378</td>
</tr>
<tr>
<td>dest_let</td>
<td>379</td>
</tr>
<tr>
<td>dest_list</td>
<td>379</td>
</tr>
<tr>
<td>dest_mon_op</td>
<td>380</td>
</tr>
<tr>
<td>dest_mult</td>
<td>380</td>
</tr>
<tr>
<td>dest_pair</td>
<td>381</td>
</tr>
<tr>
<td>dest_set_comp</td>
<td>381</td>
</tr>
<tr>
<td>dest_simple_bound</td>
<td>382</td>
</tr>
<tr>
<td>DEST_SIMPLE_TERM</td>
<td>383</td>
</tr>
<tr>
<td>DEST_SIMPLE_TYPE</td>
<td>384</td>
</tr>
<tr>
<td>dest_simple</td>
<td>384</td>
</tr>
<tr>
<td>dest_simple_∃</td>
<td>385</td>
</tr>
<tr>
<td>dest_simple_∃</td>
<td>386</td>
</tr>
<tr>
<td>dest_simple_∀</td>
<td>387</td>
</tr>
<tr>
<td>dest_simple_∀</td>
<td>388</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
<table>
<thead>
<tr>
<th>Function</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>eq_trans_rule</td>
<td>165</td>
</tr>
<tr>
<td>Error</td>
<td>16</td>
</tr>
<tr>
<td>error</td>
<td>17</td>
</tr>
<tr>
<td>Error</td>
<td>70</td>
</tr>
<tr>
<td>EVERY_CAN</td>
<td>165</td>
</tr>
<tr>
<td>EVERY_C</td>
<td>165</td>
</tr>
<tr>
<td>EVERY_TTCL</td>
<td>242</td>
</tr>
<tr>
<td>EVERY_T</td>
<td>243</td>
</tr>
<tr>
<td>exit</td>
<td>50</td>
</tr>
<tr>
<td>expand_symbol</td>
<td>60</td>
</tr>
<tr>
<td>expand_type_abbrev</td>
<td>124</td>
</tr>
<tr>
<td>ExtendedIO</td>
<td>41</td>
</tr>
<tr>
<td>EXTEND_PCS_C1</td>
<td>319</td>
</tr>
<tr>
<td>EXTEND_PCS_C</td>
<td>319</td>
</tr>
<tr>
<td>extend_pcs_rule1</td>
<td>320</td>
</tr>
<tr>
<td>extend_pcs_rule</td>
<td>320</td>
</tr>
<tr>
<td>EXTEND_PCS_T1</td>
<td>321</td>
</tr>
<tr>
<td>EXTEND_PCS_T</td>
<td>321</td>
</tr>
<tr>
<td>EXTEND_PC_C1</td>
<td>319</td>
</tr>
<tr>
<td>EXTEND_PC_C</td>
<td>319</td>
</tr>
<tr>
<td>extend_pc_rule1</td>
<td>320</td>
</tr>
<tr>
<td>extend_pc_rule</td>
<td>320</td>
</tr>
<tr>
<td>EXTEND_PC_T1</td>
<td>321</td>
</tr>
<tr>
<td>EXTEND_PC_T</td>
<td>321</td>
</tr>
<tr>
<td>ext_rule</td>
<td>166</td>
</tr>
<tr>
<td>e_delete</td>
<td>33</td>
</tr>
<tr>
<td>e_dict_of_ec dict</td>
<td>40</td>
</tr>
<tr>
<td>E_DICT</td>
<td>33</td>
</tr>
<tr>
<td>e_enter</td>
<td>33</td>
</tr>
<tr>
<td>e_extend</td>
<td>33</td>
</tr>
<tr>
<td>e_flatten</td>
<td>34</td>
</tr>
<tr>
<td>e_get_key</td>
<td>33</td>
</tr>
<tr>
<td>e_key_delete</td>
<td>33</td>
</tr>
<tr>
<td>e_key_delete</td>
<td>33</td>
</tr>
<tr>
<td>e_key_extend</td>
<td>33</td>
</tr>
<tr>
<td>e_key_lookup</td>
<td>33</td>
</tr>
<tr>
<td>E_KEY</td>
<td>33</td>
</tr>
<tr>
<td>e_lookup</td>
<td>34</td>
</tr>
<tr>
<td>e_merge</td>
<td>34</td>
</tr>
<tr>
<td>e_stats</td>
<td>34</td>
</tr>
<tr>
<td>fail_can</td>
<td>166</td>
</tr>
<tr>
<td>fail_conv</td>
<td>166</td>
</tr>
<tr>
<td>fail_tac</td>
<td>243</td>
</tr>
<tr>
<td>FAIL_THEN</td>
<td>243</td>
</tr>
<tr>
<td>fail_with_canon</td>
<td>166</td>
</tr>
<tr>
<td>fail_with_conv</td>
<td>166</td>
</tr>
<tr>
<td>fail_with_tac</td>
<td>243</td>
</tr>
<tr>
<td>FAIL_WITH_THEN</td>
<td>243</td>
</tr>
<tr>
<td>Fail</td>
<td>16</td>
</tr>
<tr>
<td>fail</td>
<td>17</td>
</tr>
<tr>
<td>fc_canon1</td>
<td>168</td>
</tr>
<tr>
<td>fc_canon</td>
<td>168</td>
</tr>
<tr>
<td>FC_CAN</td>
<td>167</td>
</tr>
<tr>
<td>fc_rule</td>
<td>169</td>
</tr>
<tr>
<td>FC_TI</td>
<td>247</td>
</tr>
<tr>
<td>fc_tac</td>
<td>245</td>
</tr>
<tr>
<td>FC_T</td>
<td>246</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTEx-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_const_language</code></td>
<td>125</td>
</tr>
<tr>
<td><code>get_const_theory</code></td>
<td>139</td>
</tr>
<tr>
<td><code>get_const_type</code></td>
<td>139</td>
</tr>
<tr>
<td><code>get_controls</code></td>
<td>46</td>
</tr>
<tr>
<td><code>get_cs_3_convs</code></td>
<td>335</td>
</tr>
<tr>
<td><code>get_curly_braces</code></td>
<td>62</td>
</tr>
<tr>
<td><code>get_current_language</code></td>
<td>125</td>
</tr>
<tr>
<td><code>get_current_pc</code></td>
<td>322</td>
</tr>
<tr>
<td><code>get_current_terminators</code></td>
<td>125</td>
</tr>
<tr>
<td><code>get_current_theory_name</code></td>
<td>139</td>
</tr>
<tr>
<td><code>get_current_theory_status</code></td>
<td>139</td>
</tr>
<tr>
<td><code>get_defns</code></td>
<td>140</td>
</tr>
<tr>
<td><code>get_defn_dict</code></td>
<td>140</td>
</tr>
<tr>
<td><code>get_defn</code></td>
<td>140</td>
</tr>
<tr>
<td><code>get_descendants</code></td>
<td>140</td>
</tr>
<tr>
<td><code>get_env</code></td>
<td>41</td>
</tr>
<tr>
<td><code>get_error_messages</code></td>
<td>17</td>
</tr>
<tr>
<td><code>get_error_message</code></td>
<td>17</td>
</tr>
<tr>
<td><code>GET_FILTER_ASM_T</code></td>
<td>250</td>
</tr>
<tr>
<td><code>get_fidelity</code></td>
<td>125</td>
</tr>
<tr>
<td><code>get_flags</code></td>
<td>46</td>
</tr>
<tr>
<td><code>get_HOL_any</code></td>
<td>62</td>
</tr>
<tr>
<td><code>get_init_funs</code></td>
<td>48</td>
</tr>
<tr>
<td><code>get_int_controls</code></td>
<td>46</td>
</tr>
<tr>
<td><code>get_int_control</code></td>
<td>46</td>
</tr>
<tr>
<td><code>get_language</code></td>
<td>126</td>
</tr>
<tr>
<td><code>get_left_infixes</code></td>
<td>126</td>
</tr>
<tr>
<td><code>get_line_length</code></td>
<td>67</td>
</tr>
<tr>
<td><code>get_message_text</code></td>
<td>18</td>
</tr>
<tr>
<td><code>get_message</code></td>
<td>18</td>
</tr>
<tr>
<td><code>get_ML_any</code></td>
<td>62</td>
</tr>
<tr>
<td><code>get_ML_string</code></td>
<td>63</td>
</tr>
<tr>
<td><code>get_mmp_rule</code></td>
<td>335</td>
</tr>
<tr>
<td><code>get_ml_entry</code></td>
<td>336</td>
</tr>
<tr>
<td><code>get_nonfixes</code></td>
<td>126</td>
</tr>
<tr>
<td><code>GET_NTH_ASM_T</code></td>
<td>250</td>
</tr>
<tr>
<td><code>get_percent_name</code></td>
<td>63</td>
</tr>
<tr>
<td><code>get_postfixes</code></td>
<td>126</td>
</tr>
<tr>
<td><code>get_prefixes</code></td>
<td>126</td>
</tr>
<tr>
<td><code>get_prime_string</code></td>
<td>63</td>
</tr>
<tr>
<td><code>get_proved_conjectures</code></td>
<td>149</td>
</tr>
<tr>
<td><code>get_pr_conv</code></td>
<td>336</td>
</tr>
<tr>
<td><code>get_pr_conv1</code></td>
<td>336</td>
</tr>
<tr>
<td><code>get_pr_tac</code></td>
<td>337</td>
</tr>
<tr>
<td><code>get_pr_tac1</code></td>
<td>337</td>
</tr>
<tr>
<td><code>get_right_infixes</code></td>
<td>126</td>
</tr>
<tr>
<td><code>get_round_braces</code></td>
<td>62</td>
</tr>
<tr>
<td><code>get_RW_canons</code></td>
<td>337</td>
</tr>
<tr>
<td><code>get_RW_eqm_rule</code></td>
<td>338</td>
</tr>
<tr>
<td><code>get_RW_eqm_ext</code></td>
<td>338</td>
</tr>
<tr>
<td><code>get_save_funs</code></td>
<td>48</td>
</tr>
<tr>
<td><code>get_sc_eqm_ext</code></td>
<td>338</td>
</tr>
<tr>
<td><code>get_shell_var</code></td>
<td>48</td>
</tr>
<tr>
<td><code>get_stack_pcs</code></td>
<td>323</td>
</tr>
<tr>
<td><code>get_stats</code></td>
<td>53</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
| get_string_controls         | 46 | HTIn.                                      | 70 |
| get_string_control          | 46 | HTLbrace                                   | 70 |
| get_st_eqn_ctx              | 339| HTLbrack                                  | 70 |
| get_terminators             | 126| HTLet                                     | 70 |
| get_theory_info             | 141| HTLsbrack                                 | 70 |
| get_theory_names            | 140| HTName                                   | 70 |
| get_theory_status           | 140| HTNumLit                                | 70 |
| get_theory                  | 141| HTPostOp                                | 70 |
| get_thms                    | 141| HTPreOp                                | 70 |
| get_thm_dict                | 141| HTRbrace                                 | 70 |
| get_thm                     | 141| HTRbrack                                | 70 |
| get_types                   | 141| HTRsbrack                                 | 70 |
| get_type_abbrevs            | 127| HTSemi                                   | 70 |
| get_type_arity              | 141| HTString                                 | 70 |
| get_type_info               | 127| HTTthen                                | 70 |
| get_type_keys               | 141| HTVert                                  | 70 |
| get_type_theory             | 142| iabs                                   | 39 |
| get_undeclared_aliases      | 127| id_w                                   | 39 |
| get_undeclared_terminators  | 127| id_canon                              | 170 |
| get_undeclared_type_abbrevs | 127| id_conv                                | 170 |
| get_unproved_conjectures    | 149| id_tac                                 | 250 |
| get_user_datum              | 142| ID_THEN                                 | 250 |
| get_use_extended_chars_flag | 63 | if_rewrite_thm                         | 164 |
| get_u_simp_eqn_ctx          | 383| IF_T2                                 | 251 |
| get_variant_suffix          | 90 | if_app_conv                           | 170 |
| get_\_cd_thms               | 339| if_then_elim                          | 251 |
| get_\_es_thms               | 340| if_intro                               | 171 |
| gbh                         | 472| if_rewrite_thm                      | 164 |
| gbh                         | 473| IF_T2                                 | 251 |
| GOAL            | 222| IF_THEN2                               | 251 |
| GOAL            | 231| IF_THEN3                               | 251 |
| grab                        | 22 | IF_THEN6                               | 171 |
| gvar_subst                  | 362| IF_THEN3                               | 251 |
| hd                          | 22 | IF_THEN6                               | 251 |
| head[X]                    | 483| IF_T4                                   | 252 |
| head                       | 484| IF_? then _! else _!               | 490 |
| hol                         | 355| Ignore                                 | 56 |
| hol                         | 355| ilformed_rewrite_warning               | 153 |
| HOLReaderWriter            | 55 | ilmod                                  | 39 |
| HOLSystem                   | 48 | Infix                                  | 69 |
| HOL_lab_prod_reader         | 63 | Initialisation                         | 48 |
| hol_list                    | 7  | initial_e_dict           | 34 |
| HOL_reader                  | 64 | initial_oc_dict         | 40 |
| HOL_TOKEN                   | 70 | initial_rw_canon       | 172 |
| hol                         | 7  | initial_s_dict            | 32 |
| hol                         | 354| initial                   | 315 |
| HTAnd                       | 70 | init_stats                         | 53 |
| HTAtm                       | 70 | init                                 | 48 |
| HTAtTy                      | 70 | input_line                        | 41 |
| HTBinder                    | 70 | input                                 | 41 |
| HTBlob                      | 70 | INPUT                                 | 70 |
| HTCChar                     | 70 | insert                                | 22 |
| HTColon                     | 70 | instream                             | 41 |
| HTElse                      | 70 | inst_term_rule            | 172 |
| HTEos                       | 70 | inst_type_rule          | 173 |
| HTIf                        | 70 | inst_type                         | 90 |
| HTInOp                      | 70 | inst                                | 90 |

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
<table>
<thead>
<tr>
<th>INDEX</th>
<th>549</th>
</tr>
</thead>
<tbody>
<tr>
<td>list.mk_simple_λ</td>
<td>96</td>
</tr>
<tr>
<td>list.mk_c</td>
<td>98</td>
</tr>
<tr>
<td>list.mk_ɔ</td>
<td>98</td>
</tr>
<tr>
<td>list.mk_V</td>
<td>98</td>
</tr>
<tr>
<td>list.mk_λ</td>
<td>97</td>
</tr>
<tr>
<td>list.mk_V</td>
<td>97</td>
</tr>
<tr>
<td>list.mk=&gt;</td>
<td>98</td>
</tr>
<tr>
<td>list.mk=&gt;</td>
<td>98</td>
</tr>
<tr>
<td>listmek</td>
<td>98</td>
</tr>
<tr>
<td>listmek_.enter</td>
<td>117</td>
</tr>
<tr>
<td>listmek_merge</td>
<td>40</td>
</tr>
<tr>
<td>listmek_overwrite</td>
<td>24</td>
</tr>
<tr>
<td>listmek_possible</td>
<td>67</td>
</tr>
<tr>
<td>listmek_save_thm</td>
<td>142</td>
</tr>
<tr>
<td>listmek_simple_ɔ_intro</td>
<td>174</td>
</tr>
<tr>
<td>listmek_simple_ɔ_tac</td>
<td>255</td>
</tr>
<tr>
<td>listmek_simple_v_elim</td>
<td>174</td>
</tr>
<tr>
<td>listmek_simple_v_intro</td>
<td>174</td>
</tr>
<tr>
<td>listmek_spec_asm_tac</td>
<td>270</td>
</tr>
<tr>
<td>LIST_SPEC_ASM.T</td>
<td>271</td>
</tr>
<tr>
<td>listmek_spec_nth_asm_tac</td>
<td>270</td>
</tr>
<tr>
<td>LIST_SPEC_NTH_ASM.T</td>
<td>271</td>
</tr>
<tr>
<td>listmek_swap_asm_concl_tac</td>
<td>256</td>
</tr>
<tr>
<td>LIST_SWAP_ASM_CONCL.T</td>
<td>256</td>
</tr>
<tr>
<td>listmek_swap_nth_asm_concl_tac</td>
<td>256</td>
</tr>
<tr>
<td>LIST_SWAP_ASM_CONCL.T</td>
<td>256</td>
</tr>
<tr>
<td>listmek_term_union</td>
<td>99</td>
</tr>
<tr>
<td>listmek_variant</td>
<td>99</td>
</tr>
<tr>
<td>listmek_variant</td>
<td>99</td>
</tr>
<tr>
<td>listmek_ɔ_elim</td>
<td>175</td>
</tr>
<tr>
<td>listmek_ɔ_intro</td>
<td>175</td>
</tr>
<tr>
<td>listmek_ɔ_intro</td>
<td>174</td>
</tr>
<tr>
<td>load_files</td>
<td>40</td>
</tr>
<tr>
<td>local_error</td>
<td>64</td>
</tr>
<tr>
<td>local_warn</td>
<td>64</td>
</tr>
<tr>
<td>Look Theory</td>
<td>131</td>
</tr>
<tr>
<td>lookahead</td>
<td>41</td>
</tr>
<tr>
<td>look at next</td>
<td>58</td>
</tr>
<tr>
<td>look_up_general_reader</td>
<td>64</td>
</tr>
<tr>
<td>look_up_named_reader</td>
<td>64</td>
</tr>
<tr>
<td>look_up_specific_reader</td>
<td>64</td>
</tr>
<tr>
<td>LSADNestedStructure</td>
<td>75</td>
</tr>
<tr>
<td>LSADSection</td>
<td>75</td>
</tr>
<tr>
<td>LSADStrings</td>
<td>75</td>
</tr>
<tr>
<td>LSADString</td>
<td>75</td>
</tr>
<tr>
<td>LSADTable</td>
<td>75</td>
</tr>
<tr>
<td>LSADTables</td>
<td>75</td>
</tr>
<tr>
<td>LSADThms</td>
<td>75</td>
</tr>
<tr>
<td>LSADTypes</td>
<td>75</td>
</tr>
<tr>
<td>LSAliases</td>
<td>75</td>
</tr>
<tr>
<td>LSAXioms</td>
<td>75</td>
</tr>
<tr>
<td>LSBanner</td>
<td>75</td>
</tr>
<tr>
<td>mk_simple_bind</td>
<td>103</td>
</tr>
<tr>
<td>----------------</td>
<td>-----</td>
</tr>
<tr>
<td>mk_simple_term</td>
<td>104</td>
</tr>
<tr>
<td>mk_simple_type</td>
<td>104</td>
</tr>
<tr>
<td>mk_simple_</td>
<td></td>
</tr>
<tr>
<td>mk_simple_bind</td>
<td>105</td>
</tr>
<tr>
<td>mk_simple_</td>
<td></td>
</tr>
<tr>
<td>mk_simple_bind</td>
<td>105</td>
</tr>
<tr>
<td>mk_simple_</td>
<td></td>
</tr>
<tr>
<td>mk_simple_bind</td>
<td>105</td>
</tr>
<tr>
<td>mk_simple_</td>
<td></td>
</tr>
<tr>
<td>mk_simple_bind</td>
<td>105</td>
</tr>
<tr>
<td>mk_string</td>
<td>105</td>
</tr>
<tr>
<td>mk_term</td>
<td>105</td>
</tr>
<tr>
<td>mk_t</td>
<td>105</td>
</tr>
<tr>
<td>mk_u</td>
<td>363</td>
</tr>
<tr>
<td>mk_vartype</td>
<td>105</td>
</tr>
<tr>
<td>mk_var</td>
<td>106</td>
</tr>
<tr>
<td>mk_z_abs</td>
<td>514</td>
</tr>
<tr>
<td>mk_z_app</td>
<td>363</td>
</tr>
<tr>
<td>mk_z_binding</td>
<td>364</td>
</tr>
<tr>
<td>mk_z_dec</td>
<td>364</td>
</tr>
<tr>
<td>mk_z_dec</td>
<td>364</td>
</tr>
<tr>
<td>mk_z_div</td>
<td>514</td>
</tr>
<tr>
<td>mk_z_eq</td>
<td>514</td>
</tr>
<tr>
<td>mk_z_false</td>
<td>514</td>
</tr>
<tr>
<td>mk_z_float</td>
<td>365</td>
</tr>
<tr>
<td>mk_z_given_type</td>
<td>366</td>
</tr>
<tr>
<td>mk_z_greater</td>
<td>514</td>
</tr>
<tr>
<td>mk_z_gvar</td>
<td>366</td>
</tr>
<tr>
<td>mk_z_hide</td>
<td>366</td>
</tr>
<tr>
<td>mk_z_h_schema</td>
<td>366</td>
</tr>
<tr>
<td>mk_z_if</td>
<td>496</td>
</tr>
<tr>
<td>mk_z_int</td>
<td>366</td>
</tr>
<tr>
<td>mk_z_less</td>
<td>514</td>
</tr>
<tr>
<td>mk_z_let</td>
<td>367</td>
</tr>
<tr>
<td>mk_z_lvar</td>
<td>367</td>
</tr>
<tr>
<td>mk_z_minus</td>
<td>514</td>
</tr>
<tr>
<td>mk_z_mod</td>
<td>514</td>
</tr>
<tr>
<td>mk_z_plus</td>
<td>514</td>
</tr>
<tr>
<td>mk_z_power_type</td>
<td>367</td>
</tr>
<tr>
<td>mk_z_pres</td>
<td>367</td>
</tr>
<tr>
<td>mk_z_real</td>
<td>531</td>
</tr>
<tr>
<td>mk_z_rename</td>
<td>368</td>
</tr>
<tr>
<td>mk_z_schema_dec</td>
<td>368</td>
</tr>
<tr>
<td>mk_z_schema_pred</td>
<td>368</td>
</tr>
<tr>
<td>mk_z_schema_type</td>
<td>368</td>
</tr>
<tr>
<td>mk_z_select</td>
<td>369</td>
</tr>
<tr>
<td>mk_z_set</td>
<td>369</td>
</tr>
<tr>
<td>mk_z_seta</td>
<td>369</td>
</tr>
<tr>
<td>mk_z_setd</td>
<td>369</td>
</tr>
<tr>
<td>mk_z_string</td>
<td>370</td>
</tr>
<tr>
<td>mk_z_subtract</td>
<td>514</td>
</tr>
<tr>
<td>mk_z_term</td>
<td>370</td>
</tr>
<tr>
<td>mk_z_times</td>
<td>514</td>
</tr>
<tr>
<td>mk_z_true</td>
<td>370</td>
</tr>
<tr>
<td>mk_z_tuple_type</td>
<td>370</td>
</tr>
<tr>
<td>mk_z_tuple</td>
<td>370</td>
</tr>
<tr>
<td>mk_z_var_type</td>
<td>370</td>
</tr>
<tr>
<td>mk_z_()</td>
<td>372</td>
</tr>
<tr>
<td>named_quotation</td>
<td></td>
</tr>
<tr>
<td>pp_set_eval_ad__3_cd_thms</td>
<td>pure_asm_rewrite_thm_tac</td>
</tr>
<tr>
<td>pp_set_eval_ad__4_vs_thms</td>
<td>pure_once_asm_rewrite_rule</td>
</tr>
<tr>
<td>pp_theory_hierarchy</td>
<td>pure_once_asm_rewrite_tac</td>
</tr>
<tr>
<td>pp_TS</td>
<td>pure_asm_rewrite_thm_tac</td>
</tr>
<tr>
<td>pp_TypesAndTerms</td>
<td>pure_once_rewrite_conv</td>
</tr>
<tr>
<td>PPArray</td>
<td>pure_once_rewrite_conv</td>
</tr>
<tr>
<td>PPString</td>
<td>pure_once_rewrite_tac</td>
</tr>
<tr>
<td>PPVector</td>
<td>pure_once_rewrite_thm_tac</td>
</tr>
<tr>
<td>pp_formal_depth</td>
<td>pure_rewrite_conv</td>
</tr>
<tr>
<td>pp_init</td>
<td>pure_rewrite_rule</td>
</tr>
<tr>
<td>pp_list</td>
<td>pure_rewrite_thm_tac</td>
</tr>
<tr>
<td>pp_make_database</td>
<td>push_back</td>
</tr>
<tr>
<td>pp_print_assumptions</td>
<td>push_extend_pcs</td>
</tr>
<tr>
<td>pp_top_level_depth</td>
<td>push_extend_pc</td>
</tr>
<tr>
<td>pp</td>
<td>push_goal_state_thm</td>
</tr>
<tr>
<td>predicates1</td>
<td>push_goal</td>
</tr>
<tr>
<td>predicates</td>
<td>push_merge_pcs</td>
</tr>
<tr>
<td>Prefix</td>
<td>push_pc</td>
</tr>
<tr>
<td>present</td>
<td>quantifier</td>
</tr>
<tr>
<td>PrettyNames</td>
<td>quit</td>
</tr>
<tr>
<td>PrettyPrinter</td>
<td>RANTS_C</td>
</tr>
<tr>
<td>PRETTY_NAME</td>
<td>RAND_C</td>
</tr>
<tr>
<td>prim_res_rule</td>
<td>ran([X, Y])</td>
</tr>
<tr>
<td>prim_rewrite_conv</td>
<td>ran</td>
</tr>
<tr>
<td>prim_rewrite_rule</td>
<td>rassoc1</td>
</tr>
<tr>
<td>prim_rewrite_tac</td>
<td>rassoc2</td>
</tr>
<tr>
<td>prim_succ_conv</td>
<td>rassoc3</td>
</tr>
<tr>
<td>print_banner</td>
<td>rassoc4</td>
</tr>
<tr>
<td>print_current_goal</td>
<td>rassoc5</td>
</tr>
<tr>
<td>print_goal</td>
<td>ReaderWriterSupport</td>
</tr>
<tr>
<td>print_stats</td>
<td>ReaderWriter</td>
</tr>
<tr>
<td>print_status</td>
<td>Reader_Writer</td>
</tr>
<tr>
<td>print_theory</td>
<td>Reader_ENV</td>
</tr>
<tr>
<td>Pro^lizing</td>
<td>Reader_FUNCTION</td>
</tr>
<tr>
<td>profiling</td>
<td>Reader_FUNCTION</td>
</tr>
<tr>
<td>prof</td>
<td>read_stopwatch</td>
</tr>
<tr>
<td>prompt1</td>
<td>read_symbol</td>
</tr>
<tr>
<td>prompt2</td>
<td>PROP_EQ_T</td>
</tr>
<tr>
<td>ProofContexts1</td>
<td>real</td>
</tr>
<tr>
<td>ProofContext</td>
<td>real</td>
</tr>
<tr>
<td>PROOF</td>
<td>redo</td>
</tr>
<tr>
<td>PropositionalEquality</td>
<td>refl_conv</td>
</tr>
<tr>
<td>prop_eq_pair</td>
<td>rel</td>
</tr>
<tr>
<td>prop_eq_prove_tac</td>
<td>rel</td>
</tr>
<tr>
<td>prop_eq_rule</td>
<td>rel</td>
</tr>
<tr>
<td>PROP_EQ_T</td>
<td>rel</td>
</tr>
<tr>
<td>pro^p_eq</td>
<td>rel</td>
</tr>
<tr>
<td>prove_asm_rule</td>
<td>rel</td>
</tr>
<tr>
<td>prove_conv</td>
<td>rel</td>
</tr>
<tr>
<td>prove_rule</td>
<td>rename_tac</td>
</tr>
<tr>
<td>prove_tac</td>
<td>rename</td>
</tr>
<tr>
<td>prove_thm</td>
<td>REPEAT_C1</td>
</tr>
<tr>
<td>prove__3_conv</td>
<td>REPEAT_CAN</td>
</tr>
<tr>
<td>prove__3_rule</td>
<td>REPEAT_C</td>
</tr>
<tr>
<td>prove__3_tac</td>
<td>REPEAT_MAP_C</td>
</tr>
<tr>
<td>pure_asm_rewrite_rule</td>
<td>REPEAT_N_T</td>
</tr>
<tr>
<td>pure_asm_rewrite_tac</td>
<td>REPEAT_N</td>
</tr>
<tr>
<td>INDEX</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>swap_nth_asm_concl_tac</td>
<td>275</td>
</tr>
<tr>
<td>SWAP_NTH_ASM_CONCL_T</td>
<td>276</td>
</tr>
<tr>
<td>swap_v_tac</td>
<td>276</td>
</tr>
<tr>
<td>swap</td>
<td>29</td>
</tr>
<tr>
<td>switch</td>
<td>29</td>
</tr>
<tr>
<td>SymbolTable</td>
<td>122</td>
</tr>
<tr>
<td>SYMBOL</td>
<td>57</td>
</tr>
<tr>
<td>SymCharacter</td>
<td>57</td>
</tr>
<tr>
<td>SymDoublePercent</td>
<td>57</td>
</tr>
<tr>
<td>SymEndOfFile</td>
<td>57</td>
</tr>
<tr>
<td>SymKnown</td>
<td>57</td>
</tr>
<tr>
<td>SymUnknownChar</td>
<td>57</td>
</tr>
<tr>
<td>SymUnknownKw</td>
<td>57</td>
</tr>
<tr>
<td>SymWhite</td>
<td>57</td>
</tr>
<tr>
<td>SystemControl</td>
<td>46</td>
</tr>
<tr>
<td>system</td>
<td>41</td>
</tr>
<tr>
<td>s_delete</td>
<td>32</td>
</tr>
<tr>
<td>S_DICT</td>
<td>19</td>
</tr>
<tr>
<td>s_enter</td>
<td>32</td>
</tr>
<tr>
<td>s_extend</td>
<td>32</td>
</tr>
<tr>
<td>s_lookup</td>
<td>32</td>
</tr>
<tr>
<td>s_merge</td>
<td>32</td>
</tr>
<tr>
<td>S</td>
<td>30</td>
</tr>
<tr>
<td>Tactics1</td>
<td>231</td>
</tr>
<tr>
<td>Tactics2</td>
<td>231</td>
</tr>
<tr>
<td>Tactics3</td>
<td>231</td>
</tr>
<tr>
<td>tactic_subgoal_warning</td>
<td>222</td>
</tr>
<tr>
<td>TACTIC</td>
<td>231</td>
</tr>
<tr>
<td>tap_proof</td>
<td>277</td>
</tr>
<tr>
<td>tail[X]</td>
<td>483</td>
</tr>
<tr>
<td>tail</td>
<td>484</td>
</tr>
<tr>
<td>TAll</td>
<td>151</td>
</tr>
<tr>
<td>TAny</td>
<td>151</td>
</tr>
<tr>
<td>taut_cone</td>
<td>277</td>
</tr>
<tr>
<td>taut_rule</td>
<td>278</td>
</tr>
<tr>
<td>taut_tac</td>
<td>278</td>
</tr>
<tr>
<td>term_any</td>
<td>113</td>
</tr>
<tr>
<td>term_const</td>
<td>113</td>
</tr>
<tr>
<td>term_diff</td>
<td>113</td>
</tr>
<tr>
<td>term_fail</td>
<td>113</td>
</tr>
<tr>
<td>term_fold</td>
<td>113</td>
</tr>
<tr>
<td>term_grab</td>
<td>113</td>
</tr>
<tr>
<td>term_less</td>
<td>114</td>
</tr>
<tr>
<td>term_map</td>
<td>114</td>
</tr>
<tr>
<td>term_match</td>
<td>114</td>
</tr>
<tr>
<td>term_mem</td>
<td>114</td>
</tr>
<tr>
<td>term_order</td>
<td>302</td>
</tr>
<tr>
<td>term_letcons</td>
<td>114</td>
</tr>
<tr>
<td>term_types</td>
<td>114</td>
</tr>
<tr>
<td>term_tvars</td>
<td>114</td>
</tr>
<tr>
<td>term_unify</td>
<td>314</td>
</tr>
<tr>
<td>term_union</td>
<td>114</td>
</tr>
<tr>
<td>term_vars</td>
<td>114</td>
</tr>
<tr>
<td>Term</td>
<td>70</td>
</tr>
<tr>
<td>TERM</td>
<td>80</td>
</tr>
<tr>
<td>TEST</td>
<td>151</td>
</tr>
<tr>
<td>tczdi</td>
<td>13</td>
</tr>
<tr>
<td>Term</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>TSDel ected</td>
<td>119</td>
</tr>
<tr>
<td>TSLocked</td>
<td>119</td>
</tr>
<tr>
<td>TSNor mal</td>
<td>119</td>
</tr>
<tr>
<td>TTAxion</td>
<td>151</td>
</tr>
<tr>
<td>TTDfn</td>
<td>151</td>
</tr>
<tr>
<td>TTSaved</td>
<td>151</td>
</tr>
<tr>
<td>TypesAndTerms</td>
<td>79</td>
</tr>
<tr>
<td>type_any</td>
<td>115</td>
</tr>
<tr>
<td>type_fail</td>
<td>115</td>
</tr>
<tr>
<td>type_map</td>
<td>115</td>
</tr>
<tr>
<td>type_match1</td>
<td>115</td>
</tr>
<tr>
<td>type_match</td>
<td>115</td>
</tr>
<tr>
<td>type_of</td>
<td>115</td>
</tr>
<tr>
<td>type_order</td>
<td>303</td>
</tr>
<tr>
<td>type_tycons</td>
<td>116</td>
</tr>
<tr>
<td>type_tys av</td>
<td>116</td>
</tr>
<tr>
<td>Type</td>
<td>70</td>
</tr>
<tr>
<td>TYPE</td>
<td>80</td>
</tr>
<tr>
<td>t_tac</td>
<td>280</td>
</tr>
<tr>
<td>t_thm</td>
<td>199</td>
</tr>
<tr>
<td>UD_Int</td>
<td>119</td>
</tr>
<tr>
<td>UD_Str  ing</td>
<td>119</td>
</tr>
<tr>
<td>UD_Term</td>
<td>119</td>
</tr>
<tr>
<td>UD_Type</td>
<td>119</td>
</tr>
<tr>
<td>uindex</td>
<td>36</td>
</tr>
<tr>
<td>uncurry</td>
<td>38</td>
</tr>
<tr>
<td>undeclare_ alias</td>
<td>128</td>
</tr>
<tr>
<td>undeclare_ termina tor</td>
<td>128</td>
</tr>
<tr>
<td>undeclare_ type_ abbrev</td>
<td>128</td>
</tr>
<tr>
<td>undisch_ rule</td>
<td>199</td>
</tr>
<tr>
<td>undo_buffer_ length</td>
<td>222</td>
</tr>
<tr>
<td>undo</td>
<td>230</td>
</tr>
<tr>
<td>union</td>
<td>27</td>
</tr>
<tr>
<td>UnlockTheory</td>
<td>131</td>
</tr>
<tr>
<td>unlock_ theory</td>
<td>148</td>
</tr>
<tr>
<td>update</td>
<td>37</td>
</tr>
<tr>
<td>update</td>
<td>38</td>
</tr>
<tr>
<td>user_ banner</td>
<td>49</td>
</tr>
<tr>
<td>USER_DATA</td>
<td>121</td>
</tr>
<tr>
<td>USER_DATUM</td>
<td>119</td>
</tr>
<tr>
<td>use_ extended_ chars</td>
<td>56</td>
</tr>
<tr>
<td>use_file</td>
<td>66</td>
</tr>
<tr>
<td>use_file_non_ stop_ mode</td>
<td>56</td>
</tr>
<tr>
<td>use_file</td>
<td>66</td>
</tr>
<tr>
<td>use_string</td>
<td>66</td>
</tr>
<tr>
<td>use_terminal</td>
<td>66</td>
</tr>
<tr>
<td>UtilitySharedTypes</td>
<td>19</td>
</tr>
<tr>
<td>u_simp_ eqn_ ext</td>
<td>384</td>
</tr>
<tr>
<td>valid_thm</td>
<td>148</td>
</tr>
<tr>
<td>Value</td>
<td>19</td>
</tr>
<tr>
<td>variant</td>
<td>116</td>
</tr>
<tr>
<td>varstruct_variant</td>
<td>200</td>
</tr>
<tr>
<td>Varype</td>
<td>79</td>
</tr>
<tr>
<td>var_elim_ asm_ tac</td>
<td>281</td>
</tr>
<tr>
<td>VAR_ELIM_ASM_T</td>
<td>281</td>
</tr>
<tr>
<td>var_elim_nth_asm_ tac</td>
<td>281</td>
</tr>
<tr>
<td>VAR_ELIM_NTH_ASM_T</td>
<td>281</td>
</tr>
<tr>
<td>ZSchemaCalculus</td>
<td>432</td>
</tr>
<tr>
<td>ZSchemaDec</td>
<td>360</td>
</tr>
<tr>
<td>ZSchemaPred</td>
<td>360</td>
</tr>
<tr>
<td>ZSchemaType</td>
<td>360</td>
</tr>
<tr>
<td>ZSel</td>
<td>360</td>
</tr>
<tr>
<td>ZSel_1</td>
<td>360</td>
</tr>
<tr>
<td>ZSequences</td>
<td>522</td>
</tr>
<tr>
<td>ZSequences</td>
<td>522</td>
</tr>
<tr>
<td>ZSet</td>
<td>360</td>
</tr>
<tr>
<td>ZSetds</td>
<td>493</td>
</tr>
<tr>
<td>ZString</td>
<td>360</td>
</tr>
<tr>
<td>ZTrue</td>
<td>360</td>
</tr>
<tr>
<td>ZTupleType</td>
<td>360</td>
</tr>
<tr>
<td>ZTuple</td>
<td>360</td>
</tr>
<tr>
<td>ZTypesAndTerms</td>
<td>359</td>
</tr>
<tr>
<td>ZVarType</td>
<td>360</td>
</tr>
<tr>
<td>z_0_less_times_thm</td>
<td>471</td>
</tr>
<tr>
<td>z_0_less_times_thm</td>
<td>523</td>
</tr>
<tr>
<td>z_0_N_thm</td>
<td>465</td>
</tr>
<tr>
<td>z_0_N_thm</td>
<td>516</td>
</tr>
<tr>
<td>z_abs_0_less_thm</td>
<td>471</td>
</tr>
<tr>
<td>z_abs_0_less_thm</td>
<td>523</td>
</tr>
<tr>
<td>z_abs_conv</td>
<td>518</td>
</tr>
<tr>
<td>z_abs_eq_0_thm</td>
<td>468</td>
</tr>
<tr>
<td>z_abs_eq_0_thm</td>
<td>516</td>
</tr>
<tr>
<td>z_abs_minus_thm</td>
<td>468</td>
</tr>
<tr>
<td>z_abs_minus_thm</td>
<td>516</td>
</tr>
<tr>
<td>z_abs_neg_thm</td>
<td>471</td>
</tr>
<tr>
<td>z_abs_neg_thm</td>
<td>523</td>
</tr>
<tr>
<td>z_abs_plus_thm</td>
<td>468</td>
</tr>
<tr>
<td>z_abs_plus_thm</td>
<td>516</td>
</tr>
<tr>
<td>z_abs_pos_thm</td>
<td>471</td>
</tr>
<tr>
<td>z_abs_pos_thm</td>
<td>523</td>
</tr>
<tr>
<td>z_abs_thm</td>
<td>468</td>
</tr>
<tr>
<td>z_abs_thm</td>
<td>516</td>
</tr>
<tr>
<td>z_abs_times_thm</td>
<td>468</td>
</tr>
<tr>
<td>z_abs_times_thm</td>
<td>516</td>
</tr>
<tr>
<td>z_abs_N_thm</td>
<td>468</td>
</tr>
<tr>
<td>z_abs_N_thm</td>
<td>516</td>
</tr>
<tr>
<td>z_abs_less_times_thm</td>
<td>471</td>
</tr>
<tr>
<td>z_abs_less_times_thm</td>
<td>523</td>
</tr>
<tr>
<td>z_and_conv</td>
<td>520</td>
</tr>
<tr>
<td>z_app_conv</td>
<td>421</td>
</tr>
<tr>
<td>z_app_eq_tac</td>
<td>422</td>
</tr>
<tr>
<td>z_app_thm</td>
<td>460</td>
</tr>
<tr>
<td>z_app_¬_thm</td>
<td>460</td>
</tr>
<tr>
<td>z_app_¬_thm</td>
<td>422</td>
</tr>
<tr>
<td>z_arith_def</td>
<td>518</td>
</tr>
<tr>
<td>z_bag_def</td>
<td>528</td>
</tr>
<tr>
<td>z_basic_prove_conv</td>
<td>385</td>
</tr>
<tr>
<td>z_basic_prove_tac</td>
<td>386</td>
</tr>
<tr>
<td>z_bindingelim_conv</td>
<td>422</td>
</tr>
<tr>
<td>z_bindingelim_conv</td>
<td>423</td>
</tr>
<tr>
<td>z_binding_intro_conv</td>
<td>423</td>
</tr>
<tr>
<td>z_binding_eq_cone1</td>
<td>423</td>
</tr>
<tr>
<td>z_binding_eq_cone2</td>
<td>423</td>
</tr>
<tr>
<td>z_binding_eq_cone3</td>
<td>506</td>
</tr>
<tr>
<td>z_binding_eq_cone</td>
<td>423</td>
</tr>
</tbody>
</table>
z_first_thm ................................................. 499  z_less_trans_thm ........................................ 467
z_float_conv ........................................... 531  z_less_trans_thm ........................................ 516
z_float_thm ............................................. 477  z_less_Zless_thm ......................................... 449
z_float_thm ............................................. 533  z_less_Zless_thm ......................................... 521
z_front_def ............................................. 522  z_less_trans_thm ......................................... 467
z_fun_app_clauses ....................................... 545  z_less_trans_thm ......................................... 516
z_fun_app_clauses ....................................... 510  z_le_conv .............................................. 424
z_fun_dom_clauses ....................................... 510  z_le_conv .............................................. 424
z_fun_dom_clauses ....................................... 509  z_library1_ext .......................................... 528
z_fun_ext ............................................... 509  z_library .............................................. 528
z_fun_ran_clauses ....................................... 454  z_library_ext .......................................... 527
z_fun_ran_clauses ....................................... 510  z_lin_arith ............................................ 519
z_func_def ............................................... 454  z_lin_arith ............................................ 519
zfunc_def ............................................... 510  z_max_def ............................................... 518
z_gen_pred_elim ......................................... 391  z_minus_clauses ......................................... 465
z_gen_pred_elim ......................................... 391  z_minus_clauses ......................................... 516
z_gen_pred_intro ........................................ 391  z_minus_thm ............................................ 465
z_gen_pred_tac .......................................... 392  z_minus_thm ............................................ 516
z_gen_pred_u_elim ....................................... 392  z_minus_thm ............................................ 516
z_get_spec ............................................... 393  z_minus_times_thm ..................................... 466
z_gre <_less < 0 .......................................... 458  z_minus_times_thm ..................................... 516
z_guile_less_thm ....................................... 491  z_minus_Nless_thm ..................................... 467
z_guilethms_thm ........................................ 499  z_minus_Nless_thm ..................................... 516
z_hash_def ............................................... 518  z_min_def ............................................... 518
z_head_def .............................................. 522  z_mod_conv ............................................ 518
z_hide_conv ............................................. 434  z_mod_thm .............................................. 471
z_h_schema_converter ................................... 434  z_mod_thm .............................................. 523
z_h_schema_pred_converter ............................. 434  z_norm_h_schema_converter ............................. 435
z_id_clauses ............................................ 481  z_num_list_thm ......................................... 488
z_id_def ................................................ 507  z_output_thm .......................................... 78
z_id_thm ................................................ 456  z_output_thm .......................................... 78
z_id_thml ................................................ 526  z_para_pred_canon .................................... 395
z_id_thm ................................................ 526  z_para_pred_conv ..................................... 395
z_id_thm ................................................ 481  z_partition_def ........................................ 522
z_id_rh_thm ............................................. 456  z_plus0 ............................................... 516
z_id_rhm ................................................ 526  z_pigeon_hole_thm ...................................... 470
z_if_thm ................................................ 491  z_pigeon_hole_thm ...................................... 523
z_if_thm ................................................ 499  z_plus ............................................... 465
z_inequality_def ....................................... 518  z_plus0 ............................................... 516
z_intro_gen_pred_tac .................................. 394  z_plus_assoc_thm ...................................... 465
z_intro_v_tac .......................................... 394  z_plus_assoc_thm ...................................... 516
z_int_homomorphism_thm ................................ 465  z_plus_assoc ........................................... 464
z_int_homomorphism_thm ................................ 516  z_plus_assoc ........................................... 516
z_in_def ................................................ 528  z_plus_clauses ......................................... 466
z_iseq_def .............................................. 522  z_plus ............................................... 516
z_items_def ............................................. 528  z_plus_comm_thm ....................................... 464
z_iter_def .............................................. 518  z_plus_comm_thm ....................................... 516
z_language_ext ......................................... 504  z_plus_conv ............................................ 518
z_language .............................................. 503  z_plus_cyclic_group_thm ............................... 465
z_last_def .............................................. 522  z_plus_cyclic_group_thm ............................... 516
Z_LEFT .................................................. 424  z_plus_minus_thm ...................................... 465
z_less_cases_thm ....................................... 469  z_plus_minus_thm ...................................... 516
z_less_cases_thm ....................................... 523  z_plus_order_thm ...................................... 465
z_less_clauses ......................................... 467  z_plus_order_thm ...................................... 516
z_less_clauses ......................................... 516  z_predicates ........................................... 378
z_less_conv ............................................. 518  z_pred_decl_conv ...................................... 396
z_less_irrefl_thm ...................................... 467  z_pred_decl_conv ...................................... 396
z_less_irrefl_thm ...................................... 516  z_pred_conv ............................................ 435
z_less_plus_thm ....................................... 468  z_prime_seq_induction_thm ............................ 486

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
<table>
<thead>
<tr>
<th>Entry</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>z_prim_seq_induction_thm</td>
<td>525</td>
</tr>
<tr>
<td>z_print_fixity</td>
<td>78</td>
</tr>
<tr>
<td>z_print_theory</td>
<td>78</td>
</tr>
<tr>
<td>z_push_consistency_goal</td>
<td>397</td>
</tr>
<tr>
<td>z_quantifiers_elim_tac</td>
<td>397</td>
</tr>
<tr>
<td>Z_RANDS_C</td>
<td>424</td>
</tr>
<tr>
<td>Z_RAND_C</td>
<td>424</td>
</tr>
<tr>
<td>z_rand_clauses</td>
<td>481</td>
</tr>
<tr>
<td>z_rand_def</td>
<td>507</td>
</tr>
<tr>
<td>z_rand_mono_thm</td>
<td>455</td>
</tr>
<tr>
<td>z_rand_mono_thm</td>
<td>526</td>
</tr>
<tr>
<td>z_rand_seqd_thm</td>
<td>488</td>
</tr>
<tr>
<td>z_rand_seqd_thm</td>
<td>525</td>
</tr>
<tr>
<td>z_rand_thm</td>
<td>481</td>
</tr>
<tr>
<td>z_rand_u_thm</td>
<td>456</td>
</tr>
<tr>
<td>z_rand_u_thm</td>
<td>526</td>
</tr>
<tr>
<td>z_rand_&lt;thm</td>
<td>455</td>
</tr>
<tr>
<td>z_reflex_closure_clauses</td>
<td>482</td>
</tr>
<tr>
<td>z_reflex_trans_closure_thm</td>
<td>481</td>
</tr>
<tr>
<td>z_ref_ext</td>
<td>505</td>
</tr>
<tr>
<td>z_rel_image_clauses</td>
<td>482</td>
</tr>
<tr>
<td>z_rel_image_def</td>
<td>507</td>
</tr>
<tr>
<td>z_rel_image_thm</td>
<td>481</td>
</tr>
<tr>
<td>z_rel_inv_clauses</td>
<td>482</td>
</tr>
<tr>
<td>z_rel_inv_def</td>
<td>507</td>
</tr>
<tr>
<td>z_rel_inv_thm</td>
<td>481</td>
</tr>
<tr>
<td>z_rel_inv&lt;thm</td>
<td>456</td>
</tr>
<tr>
<td>z_rename_conv</td>
<td>436</td>
</tr>
<tr>
<td>z_ren_def</td>
<td>522</td>
</tr>
<tr>
<td>Z_RIGHT_C</td>
<td>424</td>
</tr>
<tr>
<td>z_rte_def</td>
<td>507</td>
</tr>
<tr>
<td>z_schema_pred_conv</td>
<td>398</td>
</tr>
<tr>
<td>z_schema_pred</td>
<td>436</td>
</tr>
<tr>
<td>z_schema_pred_intro_conv</td>
<td>436</td>
</tr>
<tr>
<td>z_second_def</td>
<td>498</td>
</tr>
<tr>
<td>z_second_thm</td>
<td>491</td>
</tr>
<tr>
<td>z_second_thm</td>
<td>499</td>
</tr>
<tr>
<td>z_sel_conv</td>
<td>425</td>
</tr>
<tr>
<td>z_sel_conv</td>
<td>506</td>
</tr>
<tr>
<td>z_sel_intro_conv</td>
<td>425</td>
</tr>
<tr>
<td>z_sel_lang_conv</td>
<td>425</td>
</tr>
<tr>
<td>z_seqd_app_conv</td>
<td>524</td>
</tr>
<tr>
<td>z_seqd_eq_conv</td>
<td>524</td>
</tr>
<tr>
<td>z_seqd_eq_thm</td>
<td>488</td>
</tr>
<tr>
<td>z_seqd_eq&lt;thm</td>
<td>525</td>
</tr>
<tr>
<td>z_seqd&lt;seq_thm</td>
<td>488</td>
</tr>
<tr>
<td>z_seqd&lt;seq_thm</td>
<td>525</td>
</tr>
<tr>
<td>z_seqd&lt;rw_thm</td>
<td>488</td>
</tr>
<tr>
<td>z_seqd&lt;rw_thm</td>
<td>525</td>
</tr>
<tr>
<td>z_seqd&lt;thm</td>
<td>488</td>
</tr>
<tr>
<td>z_seqd&lt;thm</td>
<td>525</td>
</tr>
<tr>
<td>z_seqd&lt;()_clauses</td>
<td>488</td>
</tr>
<tr>
<td>z_seqd&lt;()_clauses</td>
<td>525</td>
</tr>
<tr>
<td>z_seq_cases_thm</td>
<td>487</td>
</tr>
<tr>
<td>z_seq_cases_thm</td>
<td>525</td>
</tr>
<tr>
<td>z_size&lt;mono_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size&lt;mono_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_diff_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_diff_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size&lt;2_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size&lt;2_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_empty_thm</td>
<td>469</td>
</tr>
<tr>
<td>z_size_empty_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_diff_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_eq_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_eq_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size&lt;mono_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size&lt;mono_thm</td>
<td>470</td>
</tr>
</tbody>
</table>
$\sim_R$ ........................................... 473
<table>
<thead>
<tr>
<th>KEYWORD INDEX</th>
<th>577</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>C1</td>
</tr>
<tr>
<td>REPEAT</td>
<td>C1</td>
</tr>
<tr>
<td>SUB</td>
<td>C1</td>
</tr>
<tr>
<td>clear</td>
<td></td>
</tr>
<tr>
<td>_compactification</td>
<td></td>
</tr>
<tr>
<td>get_compactification</td>
<td></td>
</tr>
<tr>
<td>set_compactification</td>
<td></td>
</tr>
<tr>
<td>ZPredicate</td>
<td></td>
</tr>
<tr>
<td>ZSchema</td>
<td></td>
</tr>
<tr>
<td>EVERY</td>
<td>CAN</td>
</tr>
<tr>
<td>FC</td>
<td>CAN</td>
</tr>
<tr>
<td>FC&lt;=&gt;</td>
<td>CAN</td>
</tr>
<tr>
<td>FIRST</td>
<td>CAN</td>
</tr>
<tr>
<td>FORWARD_CHAIN</td>
<td>CAN</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>THEN</td>
<td>CAN</td>
</tr>
<tr>
<td>THEN_LIST</td>
<td>CAN</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SymUnknown</td>
<td>Char</td>
</tr>
<tr>
<td>SymUnknown</td>
<td>Character</td>
</tr>
<tr>
<td>SymUnknown</td>
<td>CharacterUtilities</td>
</tr>
<tr>
<td>get_use_extended</td>
<td>chars_flag</td>
</tr>
<tr>
<td>use_extended</td>
<td>chars</td>
</tr>
<tr>
<td>get_use_extended</td>
<td>check_asm_tac</td>
</tr>
<tr>
<td>check_is_z</td>
<td>381</td>
</tr>
<tr>
<td>check_is_z_conv_result</td>
<td>381</td>
</tr>
<tr>
<td>check_is_z_goal</td>
<td>381</td>
</tr>
<tr>
<td>set_</td>
<td>check_is_z</td>
</tr>
<tr>
<td>CHECK_IS_Z_T</td>
<td>381</td>
</tr>
<tr>
<td>check_is_z_term</td>
<td>381</td>
</tr>
<tr>
<td>check_is_z_thm</td>
<td>381</td>
</tr>
<tr>
<td>CHECKPOINT</td>
<td>132</td>
</tr>
<tr>
<td>checkpoint</td>
<td>132</td>
</tr>
<tr>
<td>get_</td>
<td>children</td>
</tr>
<tr>
<td>NAME_CLASS</td>
<td>Children</td>
</tr>
<tr>
<td>z_dom</td>
<td>clauses</td>
</tr>
<tr>
<td>z_dot_dot</td>
<td>clauses</td>
</tr>
<tr>
<td>z_dot_dot</td>
<td>clauses</td>
</tr>
<tr>
<td>z_fun_app</td>
<td>clauses</td>
</tr>
<tr>
<td>z_fun_app</td>
<td>clauses</td>
</tr>
<tr>
<td>z_fun_dom</td>
<td>clauses</td>
</tr>
<tr>
<td>z_fun_dom</td>
<td>clauses</td>
</tr>
<tr>
<td>z_fun_ran</td>
<td>clauses</td>
</tr>
<tr>
<td>z_fun_ran</td>
<td>clauses</td>
</tr>
<tr>
<td>z_fun_\in</td>
<td>clauses</td>
</tr>
<tr>
<td>z_id</td>
<td>clauses</td>
</tr>
<tr>
<td>z_less</td>
<td>clauses</td>
</tr>
<tr>
<td>z_less</td>
<td>clauses</td>
</tr>
<tr>
<td>z_minus</td>
<td>clauses</td>
</tr>
<tr>
<td>z_minus</td>
<td>clauses</td>
</tr>
<tr>
<td>z_plus</td>
<td>clauses</td>
</tr>
<tr>
<td>z_plus</td>
<td>clauses</td>
</tr>
<tr>
<td>z_ran</td>
<td>clauses</td>
</tr>
<tr>
<td>z_reflex_closure</td>
<td>clauses</td>
</tr>
<tr>
<td>z_rel_image</td>
<td>clauses</td>
</tr>
<tr>
<td>z_rel_inv</td>
<td>clauses</td>
</tr>
<tr>
<td>z_seq_\bigcap (____)</td>
<td>clauses</td>
</tr>
<tr>
<td>z_seq_\bigcap (____)</td>
<td>clauses</td>
</tr>
<tr>
<td>z_set_diff</td>
<td>clauses</td>
</tr>
<tr>
<td>z_set_diff</td>
<td>clauses</td>
</tr>
<tr>
<td>z_set_ext</td>
<td>clauses</td>
</tr>
<tr>
<td>z_set_ext</td>
<td>clauses</td>
</tr>
<tr>
<td>z_times</td>
<td>clauses</td>
</tr>
<tr>
<td>z_trans_closure</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\leq</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\leq</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>z_\to</td>
<td>clauses</td>
</tr>
<tr>
<td>adds_error</td>
<td>code</td>
</tr>
<tr>
<td>BdzF</td>
<td>Code</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
add_error | codes 59 | Const 79
HT | colon 70 | D 80
' | combin 351 | Delete 131
| combiners 30 | delete const 134
| combine 20 | dest const 82
| comm_thm 464 | get const info 125
| plus 516 | is const 91
| thm 466 | key dest const 95
| times 516 | key mk const 95
| R_thm 475 | get const keys 139
| plus 533 | declare const language 122
| comm_thm 476 | get const language 125
| R_thm 533 | mk const 100
| thm 533 | New Const 131
| complete thm 52 | get const theory 139
| compactification cache 133 | get const type 139
| compactification_cache 130 | context 222
| compactification_cache 130 | Proof Contexts 341
| compactification_mask 130 | contr rule 161
| BdzF Comp 359 | CONTR_T 239
| z_R complete_thm 475 | i contr tac 253
| z_R complete_thm 531 | get_int control 46
| concl 133 | get_int control 46
| strip concl_conv 273 | new_int control 46
| concl_in_asm_tac 238 | new_string control 46
| LIST_SWAP_ASM CONCL_T 256 | reset_int control 47
| LIST_SWAP_NTH | reset_string control 47
| .ASM CONCL_T 256 | set_int control 47
| STRIP CONCL_T 274 | set_string control 47
| SWAP_ASM CONCL_T 276 | pending_reset control state 47
| SWAP_NTH_ASM CONCL_T 276 | System Control 46
| list_swap_asm concl_tac 256 | get controls 46
| list_swap_nth_asm concl_tac 256 | get_int controls 46
| strip concl_tac 274 | get_string controls 46
| swap_asm concl_tac 275 | reset controls 47
| swap_nth_asm concl_tac 275 | reset_int controls 47
| COND_C 161 | reset_string controls 47
| COND_T 238 | set controls 47
| cond_thm 161 | set_int controls 47
| delete conjecture 150 | set_string controls 47
| get conjecture 150 | CONV 119
| is_proved conjecture 149 | 'basic_prove conv 341
| new conjecture 150 | all_simple conv 154
| delete_all conjectures 150 | all_uncarry conv 156
| get conjectures 150 | all_\_uncarry conv 156
| get_proved conjectures 149 | all_\_ conv 157
| get_unproved conjectures 149 | anf conv 297
| z_push consistency_goal 397 | app if conv 158
| Z_z consistent 448 | conjecture conv 12
| Z_z consistent 448 | basic_prove conv 356
### KEYWORD INDEX

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>z_binding</td>
<td>506</td>
</tr>
<tr>
<td>get_cs, B</td>
<td>335</td>
</tr>
<tr>
<td>pp_set_eval_ad cs, B</td>
<td>330</td>
</tr>
<tr>
<td>set cs, B</td>
<td>335</td>
</tr>
<tr>
<td>count</td>
<td>53</td>
</tr>
<tr>
<td>count_def</td>
<td>528</td>
</tr>
<tr>
<td>count[X]</td>
<td>450</td>
</tr>
<tr>
<td>cov_induction_tac</td>
<td>514</td>
</tr>
<tr>
<td>cov_induction_thm</td>
<td>468</td>
</tr>
<tr>
<td>cov_induction_thm</td>
<td>516</td>
</tr>
<tr>
<td>current_ad cs, B</td>
<td>330</td>
</tr>
<tr>
<td>get cs, B</td>
<td>335</td>
</tr>
<tr>
<td>pp_set_eval_ad cs, B</td>
<td>330</td>
</tr>
<tr>
<td>set cs, B</td>
<td>335</td>
</tr>
<tr>
<td>cthm_eqn_cxt</td>
<td>161</td>
</tr>
<tr>
<td>Ctype</td>
<td>79</td>
</tr>
<tr>
<td>dest ctyper</td>
<td>82</td>
</tr>
<tr>
<td>is ctyper</td>
<td>91</td>
</tr>
<tr>
<td>key_dest ctyper</td>
<td>95</td>
</tr>
<tr>
<td>key_mk ctyper</td>
<td>95</td>
</tr>
<tr>
<td>mk ctyper</td>
<td>100</td>
</tr>
<tr>
<td>cup</td>
<td>20</td>
</tr>
<tr>
<td>list cup</td>
<td>24</td>
</tr>
<tr>
<td>get curly_braces</td>
<td>62</td>
</tr>
<tr>
<td>current_ad cs, B</td>
<td>330</td>
</tr>
<tr>
<td>current_ad_nmp_rule</td>
<td>317</td>
</tr>
<tr>
<td>current_ad_nd_net</td>
<td>330</td>
</tr>
<tr>
<td>current_ad_pr_conv</td>
<td>317</td>
</tr>
<tr>
<td>current_ad_pr_lac</td>
<td>317</td>
</tr>
<tr>
<td>current_ad_rw_canon</td>
<td>329</td>
</tr>
<tr>
<td>current_ad_rw_eqn</td>
<td>329</td>
</tr>
<tr>
<td>current_ad_rw_eqn_rule</td>
<td>317</td>
</tr>
<tr>
<td>current_ad_rw_net</td>
<td>328</td>
</tr>
<tr>
<td>current_ad_sc_conv</td>
<td>329</td>
</tr>
<tr>
<td>current_ad_st_conv</td>
<td>329</td>
</tr>
<tr>
<td>current_ad_3_cd_thms</td>
<td>330</td>
</tr>
<tr>
<td>current_ad_3_3vs_thms</td>
<td>331</td>
</tr>
<tr>
<td>current_goal</td>
<td>225</td>
</tr>
<tr>
<td>top current_label</td>
<td>228</td>
</tr>
<tr>
<td>get current_language</td>
<td>125</td>
</tr>
<tr>
<td>set current_language</td>
<td>128</td>
</tr>
<tr>
<td>get current_pc</td>
<td>322</td>
</tr>
<tr>
<td>get current_terminators</td>
<td>125</td>
</tr>
<tr>
<td>get current_terminators</td>
<td>125</td>
</tr>
<tr>
<td>get current_theory_name</td>
<td>139</td>
</tr>
<tr>
<td>get current_theory_status</td>
<td>139</td>
</tr>
<tr>
<td>curry</td>
<td>28</td>
</tr>
<tr>
<td>eqn ctxt</td>
<td>318</td>
</tr>
<tr>
<td>cthm_eqn_cxt</td>
<td>161</td>
</tr>
<tr>
<td>EQN CXT</td>
<td>315</td>
</tr>
<tr>
<td>get_rw_eqn ctxt</td>
<td>338</td>
</tr>
<tr>
<td>get_sc_eqn ctxt</td>
<td>338</td>
</tr>
<tr>
<td>get_st_eqn ctxt</td>
<td>339</td>
</tr>
<tr>
<td>get_u_simp_eqn ctxt</td>
<td>383</td>
</tr>
<tr>
<td>set_rw_eqn ctxt</td>
<td>338</td>
</tr>
<tr>
<td>set_sc_eqn ctxt</td>
<td>338</td>
</tr>
<tr>
<td>set_st_eqn ctxt</td>
<td>339</td>
</tr>
<tr>
<td>set_u_simp_eqn ctxt</td>
<td>383</td>
</tr>
<tr>
<td>simple_ho_thm_eqn</td>
<td>340</td>
</tr>
<tr>
<td>theory_u_simp_eqn</td>
<td>383</td>
</tr>
<tr>
<td>thm_eqn</td>
<td>341</td>
</tr>
<tr>
<td>u_simp_eqn</td>
<td>384</td>
</tr>
<tr>
<td>z_plus</td>
<td>465</td>
</tr>
<tr>
<td>cyclic_group_thm</td>
<td>516</td>
</tr>
<tr>
<td>DApp</td>
<td>80</td>
</tr>
<tr>
<td>DChar</td>
<td>80</td>
</tr>
<tr>
<td>DConst</td>
<td>80</td>
</tr>
<tr>
<td>DEEmptyList</td>
<td>80</td>
</tr>
<tr>
<td>DEnumSet</td>
<td>80</td>
</tr>
<tr>
<td>DEq</td>
<td>80</td>
</tr>
<tr>
<td>DFloat</td>
<td>80</td>
</tr>
<tr>
<td>DIf</td>
<td>80</td>
</tr>
<tr>
<td>DLet</td>
<td>80</td>
</tr>
<tr>
<td>DList</td>
<td>80</td>
</tr>
<tr>
<td>DPair</td>
<td>80</td>
</tr>
<tr>
<td>DSetComp</td>
<td>80</td>
</tr>
<tr>
<td>DString</td>
<td>80</td>
</tr>
<tr>
<td>DVar</td>
<td>80</td>
</tr>
<tr>
<td>DATABASE_INFO_TYPE</td>
<td>121</td>
</tr>
<tr>
<td>database_info</td>
<td>48</td>
</tr>
<tr>
<td>database_info</td>
<td>49</td>
</tr>
<tr>
<td>database_info</td>
<td>49</td>
</tr>
<tr>
<td>_TYPE</td>
<td>48</td>
</tr>
<tr>
<td>database</td>
<td>328</td>
</tr>
<tr>
<td>database</td>
<td>9</td>
</tr>
<tr>
<td>datum</td>
<td>142</td>
</tr>
<tr>
<td>Datum</td>
<td>131</td>
</tr>
<tr>
<td>datum</td>
<td>147</td>
</tr>
<tr>
<td>USER</td>
<td>119</td>
</tr>
<tr>
<td>DATA</td>
<td>305</td>
</tr>
<tr>
<td>pending_reset_pc</td>
<td>396</td>
</tr>
<tr>
<td>pp_make</td>
<td>396</td>
</tr>
<tr>
<td>get_user</td>
<td>365</td>
</tr>
<tr>
<td>SetUser</td>
<td>365</td>
</tr>
<tr>
<td>get_user</td>
<td>368</td>
</tr>
<tr>
<td>get_user</td>
<td>368</td>
</tr>
<tr>
<td>get_user</td>
<td>365</td>
</tr>
<tr>
<td>USER</td>
<td>329</td>
</tr>
<tr>
<td>USER</td>
<td>368</td>
</tr>
<tr>
<td>USER</td>
<td>368</td>
</tr>
<tr>
<td>RES</td>
<td>368</td>
</tr>
<tr>
<td>DB_TYPE</td>
<td>366</td>
</tr>
<tr>
<td>dec_conv</td>
<td>369</td>
</tr>
<tr>
<td>dec</td>
<td>365</td>
</tr>
<tr>
<td>dest z</td>
<td>368</td>
</tr>
<tr>
<td>is z</td>
<td>365</td>
</tr>
<tr>
<td>is z_schema</td>
<td>368</td>
</tr>
<tr>
<td>dec</td>
<td>365</td>
</tr>
<tr>
<td>mk z</td>
<td>368</td>
</tr>
<tr>
<td>mk z_schema</td>
<td>368</td>
</tr>
<tr>
<td>mk z_schema</td>
<td>368</td>
</tr>
<tr>
<td>mk z_schema</td>
<td>368</td>
</tr>
<tr>
<td>mk z_schema</td>
<td>368</td>
</tr>
<tr>
<td>mk z_schema</td>
<td>368</td>
</tr>
<tr>
<td>mk z_schema</td>
<td>389</td>
</tr>
<tr>
<td>mk z_schema</td>
<td>389</td>
</tr>
<tr>
<td>mk z_schema</td>
<td>433</td>
</tr>
<tr>
<td>dec</td>
<td>360</td>
</tr>
<tr>
<td>Z</td>
<td>360</td>
</tr>
<tr>
<td>ZSchema</td>
<td>360</td>
</tr>
<tr>
<td>Dec</td>
<td>360</td>
</tr>
<tr>
<td>dec</td>
<td>360</td>
</tr>
<tr>
<td>Z</td>
<td>380</td>
</tr>
<tr>
<td>Z</td>
<td>387</td>
</tr>
<tr>
<td>DEC</td>
<td>387</td>
</tr>
<tr>
<td>Decl</td>
<td>387</td>
</tr>
<tr>
<td>decl</td>
<td>364</td>
</tr>
<tr>
<td>decl</td>
<td>364</td>
</tr>
<tr>
<td>decl</td>
<td>364</td>
</tr>
<tr>
<td>decl</td>
<td>388</td>
</tr>
<tr>
<td>Decl</td>
<td>388</td>
</tr>
<tr>
<td>declare alias</td>
<td>122</td>
</tr>
<tr>
<td>declare binder</td>
<td>122</td>
</tr>
<tr>
<td>declare const</td>
<td>122</td>
</tr>
<tr>
<td>_language</td>
<td>122</td>
</tr>
<tr>
<td>Keyword</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------</td>
</tr>
<tr>
<td>declare_infix</td>
<td>123</td>
</tr>
<tr>
<td>declare_left_infix</td>
<td>123</td>
</tr>
<tr>
<td>declare_nonfix</td>
<td>123</td>
</tr>
<tr>
<td>declare_postfix</td>
<td>123</td>
</tr>
<tr>
<td>declare_prefix</td>
<td>124</td>
</tr>
<tr>
<td>declare_right_infix</td>
<td>123</td>
</tr>
<tr>
<td>declare_terminator</td>
<td>124</td>
</tr>
<tr>
<td>declare_type_abbrev</td>
<td>124</td>
</tr>
<tr>
<td>z_\in</td>
<td>433</td>
</tr>
<tr>
<td>z_\subset</td>
<td>433</td>
</tr>
<tr>
<td>dest_\in</td>
<td>364</td>
</tr>
<tr>
<td>is_\in</td>
<td>364</td>
</tr>
<tr>
<td>mk_\in</td>
<td>364</td>
</tr>
<tr>
<td>Z</td>
<td>360</td>
</tr>
<tr>
<td>z_\circle</td>
<td>486</td>
</tr>
<tr>
<td>z_\circle</td>
<td>525</td>
</tr>
<tr>
<td>z_\arith</td>
<td>518</td>
</tr>
<tr>
<td>z_\bag</td>
<td>528</td>
</tr>
<tr>
<td>z_\count</td>
<td>528</td>
</tr>
<tr>
<td>z_\disjoint</td>
<td>522</td>
</tr>
<tr>
<td>z_\dom</td>
<td>507</td>
</tr>
<tr>
<td>z_\dot_{\cdot \cdot} \dot_{\cdot \cdot} \dot_{\cdot \cdot} \dot_{\cdot \cdot}</td>
<td>518</td>
</tr>
</tbody>
</table>
| z_\first              | 498  | `z_{R_\times \times \times \times} \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \time

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fun</code></td>
<td>28</td>
</tr>
<tr>
<td><code>is</code></td>
<td>365</td>
</tr>
<tr>
<td><code>mk</code></td>
<td>365</td>
</tr>
<tr>
<td><code>Z</code></td>
<td>360</td>
</tr>
<tr>
<td><code>fc</code></td>
<td>380</td>
</tr>
<tr>
<td><code>FC_CAN</code></td>
<td>167</td>
</tr>
<tr>
<td><code>fe_canon</code></td>
<td>168</td>
</tr>
<tr>
<td><code>fe_canon1</code></td>
<td>168</td>
</tr>
<tr>
<td><code>z</code></td>
<td>389</td>
</tr>
<tr>
<td><code>fc_prove_con</code></td>
<td>365</td>
</tr>
<tr>
<td><code>fc_rule</code></td>
<td>169</td>
</tr>
<tr>
<td><code>ALL</code></td>
<td>246</td>
</tr>
<tr>
<td><code>ASM</code></td>
<td>246</td>
</tr>
<tr>
<td><code>D</code></td>
<td>80</td>
</tr>
<tr>
<td><code>dest</code></td>
<td>83</td>
</tr>
<tr>
<td><code>dest_z</code></td>
<td>365</td>
</tr>
<tr>
<td><code>is</code></td>
<td>91</td>
</tr>
<tr>
<td><code>mk</code></td>
<td>101</td>
</tr>
<tr>
<td><code>mk_z</code></td>
<td>365</td>
</tr>
<tr>
<td><code>all</code></td>
<td>245</td>
</tr>
<tr>
<td><code>asm</code></td>
<td>245</td>
</tr>
<tr>
<td><code>string_of</code></td>
<td>39</td>
</tr>
<tr>
<td><code>fields</code></td>
<td>318</td>
</tr>
<tr>
<td><code>fields</code></td>
<td>324</td>
</tr>
<tr>
<td><code>use</code></td>
<td>56</td>
</tr>
<tr>
<td><code>use</code></td>
<td>66</td>
</tr>
<tr>
<td><code>use</code></td>
<td>66</td>
</tr>
<tr>
<td><code>load</code></td>
<td>49</td>
</tr>
<tr>
<td><code>filter</code></td>
<td>21</td>
</tr>
<tr>
<td><code>DROP</code></td>
<td>241</td>
</tr>
<tr>
<td><code>GET</code></td>
<td>250</td>
</tr>
<tr>
<td><code>find</code></td>
<td>21</td>
</tr>
<tr>
<td><code>find_char</code></td>
<td>61</td>
</tr>
<tr>
<td><code>find_name</code></td>
<td>61</td>
</tr>
<tr>
<td><code>find_thm</code></td>
<td>151</td>
</tr>
<tr>
<td><code>gen</code></td>
<td>152</td>
</tr>
<tr>
<td><code>gen</code></td>
<td>152</td>
</tr>
<tr>
<td><code>FIRST</code></td>
<td>244</td>
</tr>
<tr>
<td><code>FIRST_C</code></td>
<td>167</td>
</tr>
<tr>
<td><code>FIRST_CAN</code></td>
<td>166</td>
</tr>
<tr>
<td><code>first_def</code></td>
<td>498</td>
</tr>
<tr>
<td><code>MAP</code></td>
<td>257</td>
</tr>
<tr>
<td><code>MAP</code></td>
<td>244</td>
</tr>
<tr>
<td><code>MAP</code></td>
<td>244</td>
</tr>
<tr>
<td><code>z</code></td>
<td>491</td>
</tr>
<tr>
<td><code>z</code></td>
<td>499</td>
</tr>
<tr>
<td><code>z</code></td>
<td>453</td>
</tr>
<tr>
<td><code>z</code></td>
<td>510</td>
</tr>
<tr>
<td><code>FIRST_TTCL</code></td>
<td>244</td>
</tr>
<tr>
<td><code>FIRST_THM</code></td>
<td>489</td>
</tr>
<tr>
<td><code>FIXITY</code></td>
<td>69</td>
</tr>
<tr>
<td><code>get</code></td>
<td>125</td>
</tr>
<tr>
<td><code>LS</code></td>
<td>75</td>
</tr>
<tr>
<td><code>z_print</code></td>
<td>78</td>
</tr>
</tbody>
</table>

### Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
 KEYWORD INDEX  595

pp' set_database: info 49  string_of: int3  65
THM: INFO_TEST 151  INTEGER  39
THEORY: INFO 120  Integer  39
THM: INFO 151  int_of:  39
ICL DATABASE: INFO_TYPE 48  integer_of_int  39
init 48  integer_of_string  39
new: init_fun 49  integer_order  39
get: init_funs 48  integer  39
pp: intro 74  kernel: interface_diagnostics  142
pp' error: init 19  Kernel  129
init_stats 53  pending_reset_kernel: interface  146
initial 315  intro 22
initial_e_dict 34  intro 381
initial_occ_dict 40  intro 155
initial빰: camon 172  intro 156
initial_s_dict 32  intro 159
Initialisation 48  Z_DECL: INTRO_C  387
input 41  intro_conv 423
INPUT 70  intro_conv 436
can: input 41  intro_conv 425
input_line 41  intro_conv 507
SymEndOf Input 57  intro_conv 427
insert 22  intro_conv 407
inst 90  intro_conv 412
inst_term_rule 172  intro_conv 415
asm: inst_term_rule 159  intro_conv 436
KI InstTermRule 129  intro_convl 405
inst_type 90  intro_convl 411
inst_type_rule 173  intro_gen_pred_tac 394
asm: inst_type_rule 159  intro 171
KI InstTypeRule 129  list_simple: intro 174
is_type: instance 93  list_simple_int 174
istream 41  list_int 175
get: int_control 46  intro_rule 218
new: int_control 46  intro 189
reset: int_control 47  intro 190
set: int_control 47  intro 191
get: int_controls 46  intro_thm 213
reset: int_controls 47  intro 200
z: int_def 463  ⇔: intro 391
z: int_def 518  intro 408
dest_z: int 366  intro 202
dest_z_signed: int 513  intro 204
z: int_homomorphism 465  intro 205
z: int_homomorphism 516  intro 207
integer_of: int 39  \\~: intro 208
is_z: int 366  ⇒\~: intro 211
is_z_signed: int 513  intro: \\forall: intro 252
mk_z: int 366  intro_\\forall: tac 252
mk_z_signed: int 514  intro_\\forall: tac 239
int_of_integer 39  intro_\\forall: tac\dagger 252
string_of: int 31  \\exists_1: intro 214
UD: Int 119  \\exists_1: intro 216
Z: Int 360  intro1 407
Z: Int 463  inv_clauses 242

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
Keyword Index

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>∀</td>
<td>408</td>
</tr>
<tr>
<td>inv_conv</td>
<td>413</td>
</tr>
<tr>
<td>inv_def</td>
<td>507</td>
</tr>
<tr>
<td>inv_thm</td>
<td>481</td>
</tr>
<tr>
<td>inv-&gt;thm</td>
<td>456</td>
</tr>
<tr>
<td>inv--&gt;thm</td>
<td>526</td>
</tr>
<tr>
<td>Io</td>
<td>56</td>
</tr>
<tr>
<td>Basic IO</td>
<td>41</td>
</tr>
<tr>
<td>Extended IO</td>
<td>41</td>
</tr>
<tr>
<td>less</td>
<td>467</td>
</tr>
<tr>
<td>dless</td>
<td>516</td>
</tr>
<tr>
<td>dless</td>
<td>474</td>
</tr>
<tr>
<td>less</td>
<td>531</td>
</tr>
<tr>
<td>all_decimal</td>
<td>31</td>
</tr>
<tr>
<td>all_z_type</td>
<td>382</td>
</tr>
<tr>
<td>app</td>
<td>90</td>
</tr>
<tr>
<td>bin_op</td>
<td>91</td>
</tr>
<tr>
<td>binder</td>
<td>91</td>
</tr>
<tr>
<td>char</td>
<td>91</td>
</tr>
<tr>
<td>const</td>
<td>91</td>
</tr>
<tr>
<td>ctype</td>
<td>91</td>
</tr>
<tr>
<td>dollar_quoted_string</td>
<td>363</td>
</tr>
<tr>
<td>empty_list</td>
<td>91</td>
</tr>
<tr>
<td>enum_set</td>
<td>91</td>
</tr>
<tr>
<td>eq</td>
<td>91</td>
</tr>
<tr>
<td>f</td>
<td>92</td>
</tr>
<tr>
<td>float</td>
<td>91</td>
</tr>
<tr>
<td>free_in</td>
<td>92</td>
</tr>
<tr>
<td>free_var_in</td>
<td>92</td>
</tr>
<tr>
<td>if</td>
<td>92</td>
</tr>
<tr>
<td>let</td>
<td>92</td>
</tr>
<tr>
<td>list</td>
<td>92</td>
</tr>
<tr>
<td>mon_op</td>
<td>92</td>
</tr>
<tr>
<td>Nil</td>
<td>22</td>
</tr>
<tr>
<td>nil</td>
<td>22</td>
</tr>
<tr>
<td>pair</td>
<td>92</td>
</tr>
<tr>
<td>proved_conjecture</td>
<td>149</td>
</tr>
<tr>
<td>same_symbol</td>
<td>64</td>
</tr>
<tr>
<td>set_comp</td>
<td>92</td>
</tr>
<tr>
<td>simple_binder</td>
<td>93</td>
</tr>
<tr>
<td>simple_V</td>
<td>93</td>
</tr>
<tr>
<td>simple_Ξ</td>
<td>93</td>
</tr>
<tr>
<td>simple_Ξ_I</td>
<td>93</td>
</tr>
<tr>
<td>simple_λ</td>
<td>93</td>
</tr>
<tr>
<td>special_char</td>
<td>64</td>
</tr>
<tr>
<td>string</td>
<td>93</td>
</tr>
<tr>
<td>t</td>
<td>93</td>
</tr>
<tr>
<td>term_in</td>
<td>41</td>
</tr>
<tr>
<td>term_out</td>
<td>41</td>
</tr>
<tr>
<td>theory_ancestor</td>
<td>142</td>
</tr>
<tr>
<td>type_abbrev</td>
<td>127</td>
</tr>
<tr>
<td>type_instance</td>
<td>93</td>
</tr>
<tr>
<td>u</td>
<td>363</td>
</tr>
<tr>
<td>var</td>
<td>94</td>
</tr>
<tr>
<td>vartype</td>
<td>93</td>
</tr>
<tr>
<td>white</td>
<td>64</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTerminals</td>
<td>75</td>
</tr>
<tr>
<td>LSDefns</td>
<td>75</td>
</tr>
<tr>
<td>LSFixity</td>
<td>75</td>
</tr>
<tr>
<td>Lisparsities</td>
<td>75</td>
</tr>
<tr>
<td>LSGenerals</td>
<td>75</td>
</tr>
<tr>
<td>LSAttributes</td>
<td>75</td>
</tr>
<tr>
<td>LSStatements</td>
<td>75</td>
</tr>
<tr>
<td>LSExpressions</td>
<td>75</td>
</tr>
<tr>
<td>LSExpressions</td>
<td>75</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
simple_⇒. match_mp_rule1 188 is_z_R. minus 530
⇒. match_mp_rule1 201 mk_z. minus 514
⇒. match_mp_rule1 209 mk_z. minus 531
simple_⇒. match_mp_rule2 188 z. minus_thm 465
⇒. match_mp_rule2 209 z. minus_thm 516
term. match 114 z_abs. minus_thm 468
type. match 115 z_abs. minus_thm 516
type. match1 115 z_plus. minus_thm 516
max 462 z. minus_thm 516
max 464 z_R. minus_thm 533
z. max_def 518 z_R_plus. minus_thm 475
mem 24 z_R_plus. minus_thm 533
term. mem 114 z_N_abs. minus_thm 516
e. merge 35 z_N_abs. minus_thm 516
e. merge 34 z_N. minus_thm 466
list. e. merge 34 z_N. minus_thm 516
list. e. merge 34 z. Z. minus_thm 449
oe. merge 40 z. Z. minus_thm 521
merge_pc_fields 324 Z. Z. minus_thm 449
merge_pcs 323 Z. Z. minus_thm 521
MERGE_PCS_C 325 z. minus_times_thm 466
MERGE_PCS_C1 325 z. minus_times_thm 516
pending_push. merge_pcs 327 z. minus_N. ≤ thm 467
push. merge_pcs 334 z. minus_N. ≤ thm 516
merge_pcs_rule 326 mk_app 99
merge_pcs_rule1 325 list. mk_app 95
set. merge_pcs 334 mk_app_rule 176
MERGE_PCS_T 326 KI MkAppRule 129
MERGE_PCS_T1 326 mk_bin_op 100
s. merge 32 list. mk_bin_op 96
MESSAGE 16 list. mk_binder 99
get. message 18 mk_binder 96
new_error. message 18 mk_const 100
pp/ change_error. message 19 key. mk_const 95
g. messages 18 mk ctype 100
gc. messages 17 mk ctype 95
get_error. messages 17 mk _ dollar _ quoted _ string 363
pending_reset_error. messages 18 _ _ _
set_error. messages 17 _ _ _
Microseconds 54 _ _ _
Middle 56 _ _ _
Milliseconds 54 _ _ _
min 462 _ _ _
min 464 _ _ _
z. min_def 518 _ _ _
z. minus_clauses 465 _ _ _
z. minus_clauses 516 _ _ _
z. minus_clauses 475 _ _ _
z. minus_clauses 533 _ _ _
z. subtract. minus_conv 518 _ _ _
z. minus_conv 534 _ _ _
z. minus_conv 535 _ _ _
dest. z. minus 513 _ _ _
dest. z. minus 530 _ _ _
dest. z. minus 530 _ _ _
z. minus_eq thm 475 _ _ _
z. minus_eq thm 533 _ _ _
is. z. minus 513 _ _ _
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>97</td>
</tr>
<tr>
<td>mk_simple_∃</td>
<td>97</td>
</tr>
<tr>
<td>mk_simple_∃₁</td>
<td>104</td>
</tr>
<tr>
<td>mk_simple_λ</td>
<td>105</td>
</tr>
<tr>
<td>mk_simple_λ</td>
<td>96</td>
</tr>
<tr>
<td>mk_string</td>
<td>105</td>
</tr>
<tr>
<td>mk t</td>
<td>105</td>
</tr>
<tr>
<td>mk_term</td>
<td>363</td>
</tr>
<tr>
<td>mk_u</td>
<td>106</td>
</tr>
<tr>
<td>mk_var</td>
<td>105</td>
</tr>
<tr>
<td>mk_vartype</td>
<td>514</td>
</tr>
<tr>
<td>mk_app</td>
<td>363</td>
</tr>
<tr>
<td>mk_binding</td>
<td>364</td>
</tr>
<tr>
<td>mk_dec</td>
<td>365</td>
</tr>
<tr>
<td>mk_decl</td>
<td>364</td>
</tr>
<tr>
<td>mk_decor</td>
<td>364</td>
</tr>
<tr>
<td>mk_div</td>
<td>514</td>
</tr>
<tr>
<td>mk_eq</td>
<td>365</td>
</tr>
<tr>
<td>mk_false</td>
<td>365</td>
</tr>
<tr>
<td>mk_float</td>
<td>365</td>
</tr>
<tr>
<td>mk_given_type</td>
<td>366</td>
</tr>
<tr>
<td>mk_greater</td>
<td>366</td>
</tr>
<tr>
<td>mk_gvar</td>
<td>514</td>
</tr>
<tr>
<td>mk_h_schema</td>
<td>366</td>
</tr>
<tr>
<td>mk_hide</td>
<td>366</td>
</tr>
<tr>
<td>mk_if</td>
<td>496</td>
</tr>
<tr>
<td>mk_int</td>
<td>366</td>
</tr>
<tr>
<td>mk_less</td>
<td>514</td>
</tr>
<tr>
<td>mk_let</td>
<td>367</td>
</tr>
<tr>
<td>mk_lvar</td>
<td>367</td>
</tr>
<tr>
<td>mk_minus</td>
<td>514</td>
</tr>
<tr>
<td>mk_mod</td>
<td>514</td>
</tr>
<tr>
<td>mk_plus</td>
<td>514</td>
</tr>
<tr>
<td>mk_power_type</td>
<td>367</td>
</tr>
<tr>
<td>mk_pres</td>
<td>367</td>
</tr>
<tr>
<td>mk_real</td>
<td>531</td>
</tr>
<tr>
<td>mk_rename</td>
<td>368</td>
</tr>
<tr>
<td>mk_schema_dec</td>
<td>368</td>
</tr>
<tr>
<td>mk_schema_type</td>
<td>368</td>
</tr>
<tr>
<td>mk_secl</td>
<td>369</td>
</tr>
<tr>
<td>mk_secl₁</td>
<td>369</td>
</tr>
<tr>
<td>mk_seca</td>
<td>369</td>
</tr>
<tr>
<td>mk_secd</td>
<td>369</td>
</tr>
<tr>
<td>mk_string</td>
<td>370</td>
</tr>
<tr>
<td>mk_subtract</td>
<td>370</td>
</tr>
<tr>
<td>mk_term</td>
<td>370</td>
</tr>
<tr>
<td>mk_times</td>
<td>370</td>
</tr>
<tr>
<td>mk_true</td>
<td>370</td>
</tr>
<tr>
<td>mk_tuple</td>
<td>370</td>
</tr>
<tr>
<td>mk_tuple_type</td>
<td>370</td>
</tr>
<tr>
<td>mk_type</td>
<td>370</td>
</tr>
<tr>
<td>mk_var_type</td>
<td>370</td>
</tr>
<tr>
<td>mk_c</td>
<td>497</td>
</tr>
<tr>
<td>mk_r</td>
<td>371</td>
</tr>
<tr>
<td>mk_r</td>
<td>371</td>
</tr>
<tr>
<td>list</td>
<td>371</td>
</tr>
<tr>
<td>mk_r</td>
<td>371</td>
</tr>
<tr>
<td>mk_r</td>
<td>371</td>
</tr>
<tr>
<td>list</td>
<td>371</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUAL USR030
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>z_z</td>
<td>521</td>
</tr>
<tr>
<td>z_x</td>
<td>487</td>
</tr>
<tr>
<td>z_x</td>
<td>521</td>
</tr>
<tr>
<td>z_x</td>
<td>449</td>
</tr>
<tr>
<td>z_x</td>
<td>521</td>
</tr>
<tr>
<td>z_x</td>
<td>449</td>
</tr>
<tr>
<td>bin_bool</td>
<td>81</td>
</tr>
<tr>
<td>dest_bin</td>
<td>82</td>
</tr>
<tr>
<td>dest_mon</td>
<td>84</td>
</tr>
<tr>
<td>HTIn</td>
<td>70</td>
</tr>
<tr>
<td>HTPost</td>
<td>70</td>
</tr>
<tr>
<td>HTPre</td>
<td>70</td>
</tr>
<tr>
<td>is_bin</td>
<td>91</td>
</tr>
<tr>
<td>is_mon</td>
<td>92</td>
</tr>
<tr>
<td>list_mk_bin</td>
<td>96</td>
</tr>
<tr>
<td>mk_bin</td>
<td>100</td>
</tr>
<tr>
<td>mk_mon</td>
<td>103</td>
</tr>
<tr>
<td>strip_bin</td>
<td>110</td>
</tr>
<tr>
<td>open_append</td>
<td>41</td>
</tr>
<tr>
<td>open_in</td>
<td>41</td>
</tr>
<tr>
<td>open_out</td>
<td>41</td>
</tr>
<tr>
<td>OpenTheory</td>
<td>131</td>
</tr>
<tr>
<td>open_theory</td>
<td>146</td>
</tr>
<tr>
<td>OPT</td>
<td>19</td>
</tr>
<tr>
<td>sub</td>
<td>36</td>
</tr>
<tr>
<td>sub</td>
<td>38</td>
</tr>
<tr>
<td>AND</td>
<td>157</td>
</tr>
<tr>
<td>OR</td>
<td>28</td>
</tr>
<tr>
<td>fun</td>
<td>39</td>
</tr>
<tr>
<td>gen_term</td>
<td>300</td>
</tr>
<tr>
<td>integer_term</td>
<td>300</td>
</tr>
<tr>
<td>make_term</td>
<td>302</td>
</tr>
<tr>
<td>term</td>
<td>302</td>
</tr>
<tr>
<td>z_plus</td>
<td>465</td>
</tr>
<tr>
<td>z_plus</td>
<td>516</td>
</tr>
<tr>
<td>z_times</td>
<td>466</td>
</tr>
<tr>
<td>z_times</td>
<td>516</td>
</tr>
<tr>
<td>z_R_plus</td>
<td>475</td>
</tr>
<tr>
<td>z_R_plus</td>
<td>533</td>
</tr>
<tr>
<td>z_R_times</td>
<td>533</td>
</tr>
<tr>
<td>z_R_times</td>
<td>477</td>
</tr>
<tr>
<td>type</td>
<td>303</td>
</tr>
<tr>
<td>translate_for</td>
<td></td>
</tr>
<tr>
<td>output</td>
<td>66</td>
</tr>
<tr>
<td>outputstream</td>
<td>41</td>
</tr>
<tr>
<td>over_clauses</td>
<td>477</td>
</tr>
<tr>
<td>over_clauses</td>
<td>533</td>
</tr>
<tr>
<td>over_conv</td>
<td>534</td>
</tr>
<tr>
<td>over_def</td>
<td>535</td>
</tr>
<tr>
<td>over</td>
<td>530</td>
</tr>
<tr>
<td>over</td>
<td>530</td>
</tr>
<tr>
<td>over</td>
<td>531</td>
</tr>
<tr>
<td>over_thm</td>
<td>533</td>
</tr>
<tr>
<td>overwrite</td>
<td>25</td>
</tr>
<tr>
<td>overwrite</td>
<td>24</td>
</tr>
<tr>
<td>pending_reset_subgoal</td>
<td></td>
</tr>
<tr>
<td>package</td>
<td>224</td>
</tr>
<tr>
<td>package_quiet</td>
<td>221</td>
</tr>
<tr>
<td>package_size</td>
<td>228</td>
</tr>
<tr>
<td>Package</td>
<td>221</td>
</tr>
<tr>
<td>package_ti_context</td>
<td>222</td>
</tr>
<tr>
<td>pair</td>
<td>345</td>
</tr>
<tr>
<td>pair</td>
<td>293</td>
</tr>
<tr>
<td>\lambda</td>
<td>219</td>
</tr>
<tr>
<td>\lambda</td>
<td>80</td>
</tr>
<tr>
<td>dest</td>
<td>84</td>
</tr>
<tr>
<td>pair</td>
<td>294</td>
</tr>
<tr>
<td>pair</td>
<td>92</td>
</tr>
<tr>
<td>pair</td>
<td>103</td>
</tr>
<tr>
<td>pair</td>
<td>293</td>
</tr>
<tr>
<td>pair</td>
<td>258</td>
</tr>
<tr>
<td>pair_thm</td>
<td>470</td>
</tr>
<tr>
<td>pair_thm</td>
<td>523</td>
</tr>
<tr>
<td>pair1</td>
<td>346</td>
</tr>
<tr>
<td>paired_abstractions</td>
<td>342</td>
</tr>
<tr>
<td>para_pred_canon</td>
<td>395</td>
</tr>
<tr>
<td>para_pred_conv</td>
<td>395</td>
</tr>
<tr>
<td>parent</td>
<td>131</td>
</tr>
<tr>
<td>parent</td>
<td>143</td>
</tr>
<tr>
<td>get</td>
<td>140</td>
</tr>
<tr>
<td>parents</td>
<td>75</td>
</tr>
<tr>
<td>LS</td>
<td></td>
</tr>
<tr>
<td>partition</td>
<td>484</td>
</tr>
<tr>
<td>partition</td>
<td>484</td>
</tr>
<tr>
<td>partition</td>
<td>484</td>
</tr>
<tr>
<td>partition</td>
<td>483</td>
</tr>
<tr>
<td>partition_def</td>
<td>522</td>
</tr>
<tr>
<td>pass_on</td>
<td>18</td>
</tr>
<tr>
<td>PC_C</td>
<td>325</td>
</tr>
<tr>
<td>PC_C</td>
<td>319</td>
</tr>
<tr>
<td>PC_C1</td>
<td>325</td>
</tr>
<tr>
<td>EXTEND</td>
<td></td>
</tr>
<tr>
<td>EXTEND</td>
<td></td>
</tr>
<tr>
<td>commit</td>
<td>316</td>
</tr>
<tr>
<td>pending_reset</td>
<td>328</td>
</tr>
<tr>
<td>pending_reset</td>
<td>318</td>
</tr>
<tr>
<td>pending_reset</td>
<td>328</td>
</tr>
<tr>
<td>pending_reset</td>
<td>318</td>
</tr>
<tr>
<td>pending_reset</td>
<td>324</td>
</tr>
<tr>
<td>pending_reset</td>
<td>322</td>
</tr>
<tr>
<td>pending_reset</td>
<td>324</td>
</tr>
<tr>
<td>pending_reset</td>
<td>327</td>
</tr>
<tr>
<td>pending_reset</td>
<td>327</td>
</tr>
<tr>
<td>pending_reset</td>
<td>328</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
<table>
<thead>
<tr>
<th>Key</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z, Z</td>
<td>448</td>
</tr>
<tr>
<td>Z, Z</td>
<td>521</td>
</tr>
<tr>
<td>Z, Z</td>
<td>449</td>
</tr>
<tr>
<td>Z, Z</td>
<td>521</td>
</tr>
<tr>
<td>Z, R</td>
<td>475</td>
</tr>
<tr>
<td>Z, R</td>
<td>533</td>
</tr>
<tr>
<td>Z, N</td>
<td>467</td>
</tr>
<tr>
<td>Z, N</td>
<td>516</td>
</tr>
<tr>
<td>Z, N</td>
<td>465</td>
</tr>
<tr>
<td>Z, N</td>
<td>516</td>
</tr>
<tr>
<td>Z, less</td>
<td>468</td>
</tr>
<tr>
<td>Z, Z, less</td>
<td>469</td>
</tr>
<tr>
<td>Z, Z, less</td>
<td>523</td>
</tr>
<tr>
<td>Z, N, less</td>
<td>465</td>
</tr>
<tr>
<td>Z, N, less</td>
<td>516</td>
</tr>
<tr>
<td>Z, N, less</td>
<td>465</td>
</tr>
<tr>
<td>poly_conv</td>
<td>301</td>
</tr>
<tr>
<td>POP_ASM_T</td>
<td>258</td>
</tr>
<tr>
<td>pop_pc</td>
<td>328</td>
</tr>
<tr>
<td>pop_thm</td>
<td>225</td>
</tr>
<tr>
<td>save</td>
<td>227</td>
</tr>
<tr>
<td>z_abs</td>
<td>471</td>
</tr>
<tr>
<td>z_abs</td>
<td>523</td>
</tr>
<tr>
<td>basic_res</td>
<td>308</td>
</tr>
<tr>
<td>HT</td>
<td>70</td>
</tr>
<tr>
<td>PostOp</td>
<td>69</td>
</tr>
<tr>
<td>declare</td>
<td>123</td>
</tr>
<tr>
<td>get</td>
<td>126</td>
</tr>
<tr>
<td>fun</td>
<td>28</td>
</tr>
<tr>
<td>dest, z</td>
<td>367</td>
</tr>
<tr>
<td>is, z</td>
<td>367</td>
</tr>
<tr>
<td>mk, z</td>
<td>367</td>
</tr>
<tr>
<td>Z</td>
<td>360</td>
</tr>
<tr>
<td>pp</td>
<td>10</td>
</tr>
<tr>
<td>PPArry</td>
<td>42</td>
</tr>
<tr>
<td>pp_format_depth</td>
<td>73</td>
</tr>
<tr>
<td>pp_init</td>
<td>74</td>
</tr>
<tr>
<td>pp_list</td>
<td>8</td>
</tr>
<tr>
<td>pp_make_database</td>
<td>9</td>
</tr>
<tr>
<td>pp_print_assumptions</td>
<td>73</td>
</tr>
<tr>
<td>PPSstring</td>
<td>42</td>
</tr>
<tr>
<td>pp_top_level_depth</td>
<td>73</td>
</tr>
<tr>
<td>PPVector</td>
<td>42</td>
</tr>
<tr>
<td>pp', change_error</td>
<td>19</td>
</tr>
<tr>
<td>_message</td>
<td></td>
</tr>
<tr>
<td>pp', database_info</td>
<td>48</td>
</tr>
<tr>
<td>pp', error_init</td>
<td>19</td>
</tr>
<tr>
<td>pp', reset_database_info</td>
<td>49</td>
</tr>
<tr>
<td>pp', set_banner</td>
<td>49</td>
</tr>
<tr>
<td>pp', set_database_info</td>
<td>49</td>
</tr>
<tr>
<td>pp', set_eval_ad_cs_∃</td>
<td>330</td>
</tr>
<tr>
<td>_convs</td>
<td></td>
</tr>
<tr>
<td>pp', set_eval_ad nd_net</td>
<td>330</td>
</tr>
<tr>
<td>pp', set_eval_ad rw</td>
<td>329</td>
</tr>
<tr>
<td>_canon</td>
<td></td>
</tr>
<tr>
<td>pp', set_eval_ad rw_net</td>
<td>328</td>
</tr>
<tr>
<td>Keyword</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------</td>
</tr>
<tr>
<td>REPEAT_MAP_C</td>
<td>182</td>
</tr>
<tr>
<td>REPEAT_N</td>
<td>261</td>
</tr>
<tr>
<td>REPEAT_N_T</td>
<td>261</td>
</tr>
<tr>
<td>REPEAT_T</td>
<td>262</td>
</tr>
<tr>
<td>REPEAT_TTCL</td>
<td>262</td>
</tr>
<tr>
<td>REPEAT_UNTIL</td>
<td>262</td>
</tr>
<tr>
<td>REPEAT_UNTIL_T</td>
<td>262</td>
</tr>
<tr>
<td>REPEAT_UNTIL_T1</td>
<td>262</td>
</tr>
<tr>
<td>REPEAT_UNTIL1</td>
<td>262</td>
</tr>
<tr>
<td>reset</td>
<td>19</td>
</tr>
<tr>
<td>RES_DB_TYPE</td>
<td>305</td>
</tr>
<tr>
<td>basic_res_extract</td>
<td>307</td>
</tr>
<tr>
<td>basic_res_next_to_process</td>
<td>308</td>
</tr>
<tr>
<td>basic_res_post</td>
<td>308</td>
</tr>
<tr>
<td>basic_res_prev</td>
<td>308</td>
</tr>
<tr>
<td>basic_res_resolver</td>
<td>308</td>
</tr>
<tr>
<td>basic_res_rule</td>
<td>309</td>
</tr>
<tr>
<td>basic_res_subsumption</td>
<td>309</td>
</tr>
<tr>
<td>basic_res_tac</td>
<td>311</td>
</tr>
<tr>
<td>basic_res_tac1</td>
<td>310</td>
</tr>
<tr>
<td>basic_res_tac2</td>
<td>310</td>
</tr>
<tr>
<td>basic_res_tac3</td>
<td>311</td>
</tr>
<tr>
<td>basic_res_tac4</td>
<td>311</td>
</tr>
<tr>
<td>BASIC_RES_TYPE</td>
<td>305</td>
</tr>
<tr>
<td>pending_reset_control_state</td>
<td>47</td>
</tr>
<tr>
<td>pending_reset Controls</td>
<td>47</td>
</tr>
<tr>
<td>pending_reset_error_messages</td>
<td>18</td>
</tr>
<tr>
<td>pending_reset_flag</td>
<td>47</td>
</tr>
<tr>
<td>pending_reset_flags</td>
<td>47</td>
</tr>
<tr>
<td>pending_reset_int_control</td>
<td>47</td>
</tr>
<tr>
<td>pending_reset_int_controls</td>
<td>47</td>
</tr>
<tr>
<td>pending_reset_kernel_interface</td>
<td>146</td>
</tr>
<tr>
<td>pending_reset_pc_database</td>
<td>328</td>
</tr>
<tr>
<td>pending_reset_pc_evaluators</td>
<td>328</td>
</tr>
<tr>
<td>pending_reset_pc_stack</td>
<td>328</td>
</tr>
<tr>
<td>pending_reset_stopwatch</td>
<td>54</td>
</tr>
<tr>
<td>pending_reset_string_control</td>
<td>47</td>
</tr>
<tr>
<td>pending_reset_string_controls</td>
<td>47</td>
</tr>
<tr>
<td>pending_reset_subgoal_package</td>
<td>224</td>
</tr>
<tr>
<td>pending_reset_use_terminal</td>
<td>65</td>
</tr>
<tr>
<td>Resolution</td>
<td>304</td>
</tr>
<tr>
<td>BASIC_RESOLUTION_T</td>
<td>306</td>
</tr>
<tr>
<td>BASIC_RESOLUTION_T1</td>
<td>307</td>
</tr>
<tr>
<td>resolve_alias</td>
<td>127</td>
</tr>
<tr>
<td>basic_resolver</td>
<td>308</td>
</tr>
<tr>
<td>check_is_z_conv</td>
<td>381</td>
</tr>
<tr>
<td>result</td>
<td>381</td>
</tr>
<tr>
<td>rev</td>
<td>484</td>
</tr>
<tr>
<td>z_rev_def</td>
<td>522</td>
</tr>
<tr>
<td>refold</td>
<td>26</td>
</tr>
<tr>
<td>refold[x]</td>
<td>483</td>
</tr>
<tr>
<td>REWRITE_CAN</td>
<td>182</td>
</tr>
<tr>
<td>f_rewrite_canon</td>
<td>203</td>
</tr>
<tr>
<td>simple_z_eq_rewrite_canon</td>
<td>203</td>
</tr>
<tr>
<td>simple_z_eq_rewrite_canon</td>
<td>203</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Value</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>strip_spine</td>
<td>right</td>
<td>KIMKApp Rule</td>
<td>111</td>
</tr>
<tr>
<td>∧_</td>
<td>right_tac</td>
<td>KISimple λ Eq Rule</td>
<td>284</td>
</tr>
<tr>
<td>fun0</td>
<td>rightassoc</td>
<td>KISubst Rule</td>
<td>450</td>
</tr>
<tr>
<td>fun0</td>
<td>rightassoc</td>
<td>KI ⇔ MP Rule</td>
<td>489</td>
</tr>
<tr>
<td>fun15</td>
<td>rightassoc</td>
<td>merge_pcs rule</td>
<td>483</td>
</tr>
<tr>
<td>fun50</td>
<td>rightassoc</td>
<td>mk_app rule</td>
<td>463</td>
</tr>
<tr>
<td>fun50</td>
<td>rightassoc</td>
<td>modus_tollens rule</td>
<td>472</td>
</tr>
<tr>
<td>fun60</td>
<td>rightassoc</td>
<td>once_asm_rewrite rule</td>
<td>472</td>
</tr>
<tr>
<td>fun65</td>
<td>rightassoc</td>
<td>once_rewrite rule</td>
<td>479</td>
</tr>
<tr>
<td>fun70</td>
<td>rightassoc</td>
<td>pc rule</td>
<td>463</td>
</tr>
<tr>
<td>fun70</td>
<td>rightassoc</td>
<td>prim_rewrite rule</td>
<td>479</td>
</tr>
<tr>
<td>gen5</td>
<td>rightassoc</td>
<td>prim_rewrite rule</td>
<td>452</td>
</tr>
<tr>
<td>gen5</td>
<td>rightassoc</td>
<td>prop_eq rule</td>
<td>463</td>
</tr>
<tr>
<td>gen5</td>
<td>rightassoc</td>
<td>prove_rule</td>
<td>490</td>
</tr>
<tr>
<td>gen70</td>
<td>rightassoc</td>
<td>prove_asm_rule</td>
<td>450</td>
</tr>
<tr>
<td>gen70</td>
<td>rightassoc</td>
<td>prove_rule</td>
<td>463</td>
</tr>
<tr>
<td>gen70</td>
<td>rightassoc</td>
<td>pure_asm_rewrite rule</td>
<td>479</td>
</tr>
<tr>
<td>gen70</td>
<td>rightassoc</td>
<td>pure_once_asm rule</td>
<td>483</td>
</tr>
<tr>
<td>gen70</td>
<td>rightassoc</td>
<td>rewrite_rule</td>
<td>490</td>
</tr>
<tr>
<td>rollback</td>
<td>rotate_braces</td>
<td>rotate_rule</td>
<td>62</td>
</tr>
<tr>
<td>get_</td>
<td>round_braces</td>
<td>rotate_rule</td>
<td>264</td>
</tr>
<tr>
<td>list_</td>
<td>roverwrite</td>
<td>set_mmp_rule</td>
<td>26</td>
</tr>
<tr>
<td>list_</td>
<td>roverwrite</td>
<td>set_rw_eqm_rule</td>
<td>24</td>
</tr>
<tr>
<td>HT</td>
<td>Rsbgbrack</td>
<td>simple ⇔ match_mp rule</td>
<td>70</td>
</tr>
<tr>
<td>z_</td>
<td>rte_def</td>
<td>simple ⇒ match_mp rule</td>
<td>507</td>
</tr>
<tr>
<td>all_simple_β</td>
<td>rule</td>
<td>simple_β_eq rule</td>
<td>154</td>
</tr>
<tr>
<td>all_β</td>
<td>rule</td>
<td>simple_β_elim rule</td>
<td>157</td>
</tr>
<tr>
<td>app_arg</td>
<td>rule</td>
<td>simple_λ_eq rule</td>
<td>158</td>
</tr>
<tr>
<td>app_fun</td>
<td>rule</td>
<td>strip_β rule</td>
<td>158</td>
</tr>
<tr>
<td>asm_</td>
<td>rule</td>
<td>strip_β ⇒ rule</td>
<td>160</td>
</tr>
<tr>
<td>asm_inst_term</td>
<td>rule</td>
<td>subst_rule</td>
<td>159</td>
</tr>
<tr>
<td>asm_inst_type</td>
<td>rule</td>
<td>taut_rule</td>
<td>159</td>
</tr>
<tr>
<td>asm_rewrite</td>
<td>rule</td>
<td>undisch_rule</td>
<td>184</td>
</tr>
<tr>
<td>basic_res</td>
<td>rule</td>
<td>z_app_β_rule</td>
<td>309</td>
</tr>
<tr>
<td>basic_resolve</td>
<td>rule</td>
<td>z_defn_β_rule</td>
<td>307</td>
</tr>
<tr>
<td>c_constr</td>
<td>rule</td>
<td>z_β_rule</td>
<td>162</td>
</tr>
<tr>
<td>contr</td>
<td>rule</td>
<td>⇒_match_mp_rule</td>
<td>161</td>
</tr>
<tr>
<td>conv</td>
<td>rule</td>
<td>⇒_mp_rule</td>
<td>161</td>
</tr>
<tr>
<td>current_ad_mmp</td>
<td>rule</td>
<td>∧_⇒_rule</td>
<td>317</td>
</tr>
<tr>
<td>current_ad_rw_eqm</td>
<td>rule</td>
<td>∨_cancel_rule</td>
<td>317</td>
</tr>
<tr>
<td>disc_</td>
<td>rule</td>
<td>¬_eq_sym_rule</td>
<td>162</td>
</tr>
<tr>
<td>eq_sym</td>
<td>rule</td>
<td>⇒_match_mp_rule</td>
<td>165</td>
</tr>
<tr>
<td>eq_trans</td>
<td>rule</td>
<td>⇒_mp_rule</td>
<td>165</td>
</tr>
<tr>
<td>ext</td>
<td>rule</td>
<td>⇒_trans_rule</td>
<td>166</td>
</tr>
<tr>
<td>extend_pc</td>
<td>rule</td>
<td>∃_⇒_rule</td>
<td>320</td>
</tr>
<tr>
<td>extend_pcs</td>
<td>rule</td>
<td>∀_asm_rule</td>
<td>320</td>
</tr>
<tr>
<td>fc</td>
<td>rule</td>
<td>∀_⇒_rule</td>
<td>320</td>
</tr>
<tr>
<td>forward_chain</td>
<td>rule</td>
<td>∃_⇒_asm_rule</td>
<td>169</td>
</tr>
<tr>
<td>get_mmp</td>
<td>rule</td>
<td>β_rule</td>
<td>335</td>
</tr>
<tr>
<td>get_rw_eqm</td>
<td>rule</td>
<td>c_elim_rule</td>
<td>338</td>
</tr>
<tr>
<td>inst_term</td>
<td>rule</td>
<td>c_intro_rule</td>
<td>172</td>
</tr>
<tr>
<td>inst_type</td>
<td>rule</td>
<td>λ_eq_rule</td>
<td>173</td>
</tr>
<tr>
<td>KIAsm</td>
<td>Rule</td>
<td>λ_rule</td>
<td>129</td>
</tr>
<tr>
<td>KIEqSym</td>
<td>Rule</td>
<td>λ_eq_rule</td>
<td>129</td>
</tr>
<tr>
<td>KIEqTrans</td>
<td>Rule</td>
<td>extend_pc_rule</td>
<td>129</td>
</tr>
<tr>
<td>KIIInstTerm</td>
<td>Rule</td>
<td>extend_pcs_rule</td>
<td>129</td>
</tr>
<tr>
<td>KIIInstType</td>
<td>Rule</td>
<td>merge_pcs_rule</td>
<td>129</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_seqd_conv</td>
<td>524</td>
</tr>
<tr>
<td>size_seqd_length_thm</td>
<td>488</td>
</tr>
<tr>
<td>size_seqd_thm</td>
<td>525</td>
</tr>
<tr>
<td>size_seqd_thm1</td>
<td>488</td>
</tr>
<tr>
<td>size_thm</td>
<td>525</td>
</tr>
<tr>
<td>size_thm1</td>
<td>470</td>
</tr>
<tr>
<td>size_thm1</td>
<td>523</td>
</tr>
<tr>
<td>size_w_thm</td>
<td>470</td>
</tr>
<tr>
<td>size_w_thm1</td>
<td>523</td>
</tr>
<tr>
<td>size_X_thm</td>
<td>470</td>
</tr>
<tr>
<td>size_X_thm1</td>
<td>523</td>
</tr>
<tr>
<td>size_L_thm</td>
<td>470</td>
</tr>
<tr>
<td>size_L_thm1</td>
<td>523</td>
</tr>
<tr>
<td>size_L_singleton_thm</td>
<td>470</td>
</tr>
<tr>
<td>size_L_thm</td>
<td>523</td>
</tr>
<tr>
<td>size_L_thm1</td>
<td>470</td>
</tr>
<tr>
<td>size_L_thm1</td>
<td>523</td>
</tr>
<tr>
<td>size_L_thm</td>
<td>487</td>
</tr>
<tr>
<td>size_L_thm1</td>
<td>525</td>
</tr>
<tr>
<td>size_N_thm</td>
<td>470</td>
</tr>
<tr>
<td>size_N_thm</td>
<td>523</td>
</tr>
<tr>
<td>skip_and_look_at_next</td>
<td>58</td>
</tr>
<tr>
<td>skip_comment</td>
<td>65</td>
</tr>
<tr>
<td>SML recognise</td>
<td>65</td>
</tr>
<tr>
<td>SML97BasisLibrary</td>
<td>43</td>
</tr>
<tr>
<td>snd</td>
<td>29</td>
</tr>
<tr>
<td>SOLVED_T</td>
<td>269</td>
</tr>
<tr>
<td>Sort</td>
<td>35</td>
</tr>
<tr>
<td>sort</td>
<td>35</td>
</tr>
<tr>
<td>sort_conv</td>
<td>302</td>
</tr>
<tr>
<td>sorted_listings</td>
<td>75</td>
</tr>
<tr>
<td>SparseArray</td>
<td>36</td>
</tr>
<tr>
<td>SPARSE_ARRAY</td>
<td>36</td>
</tr>
<tr>
<td>SPEC_ASM_T</td>
<td>271</td>
</tr>
<tr>
<td>SPEC_ASM_T</td>
<td>271</td>
</tr>
<tr>
<td>spec_asm_tac</td>
<td>270</td>
</tr>
<tr>
<td>list_spec_asm_tac</td>
<td>270</td>
</tr>
<tr>
<td>z_spec_asm_tac</td>
<td>399</td>
</tr>
<tr>
<td>NewSpec</td>
<td>131</td>
</tr>
<tr>
<td>new_spec</td>
<td>144</td>
</tr>
<tr>
<td>SPEC_NTH_ASM_T</td>
<td>271</td>
</tr>
<tr>
<td>SPEC_NTH_ASM_T</td>
<td>271</td>
</tr>
<tr>
<td>spec_nth_asm_tac</td>
<td>270</td>
</tr>
<tr>
<td>list_spec_nth_asm_tac</td>
<td>270</td>
</tr>
<tr>
<td>z_spec_nth_asm_tac</td>
<td>399</td>
</tr>
<tr>
<td>zget</td>
<td>393</td>
</tr>
<tr>
<td>is_special_char</td>
<td>64</td>
</tr>
<tr>
<td>specific_quotation</td>
<td>61</td>
</tr>
<tr>
<td>add_specific_reader</td>
<td>59</td>
</tr>
<tr>
<td>look_up</td>
<td>specific_reader</td>
</tr>
<tr>
<td>strip</td>
<td>spine_left</td>
</tr>
<tr>
<td>strip</td>
<td>spine_right</td>
</tr>
<tr>
<td>split</td>
<td>26</td>
</tr>
<tr>
<td>split3</td>
<td>26</td>
</tr>
<tr>
<td>squash</td>
<td>484</td>
</tr>
<tr>
<td>squash_def</td>
<td>522</td>
</tr>
<tr>
<td>squash[X]</td>
<td>483</td>
</tr>
<tr>
<td>current_ad</td>
<td>329</td>
</tr>
<tr>
<td>st_conv</td>
<td>329</td>
</tr>
<tr>
<td>get</td>
<td>st_eqn_ext</td>
</tr>
<tr>
<td>set</td>
<td>st_eqn_ext</td>
</tr>
<tr>
<td>add</td>
<td>st_thms</td>
</tr>
<tr>
<td>pending_reset_pc</td>
<td>stack</td>
</tr>
<tr>
<td>pending_reset_control</td>
<td>47</td>
</tr>
<tr>
<td>before_kernel</td>
<td>133</td>
</tr>
<tr>
<td>KERNEL</td>
<td>STATE_CHANGE</td>
</tr>
<tr>
<td>on_kernel</td>
<td>state_change</td>
</tr>
<tr>
<td>GOAL</td>
<td>STATE</td>
</tr>
<tr>
<td>state</td>
<td>47</td>
</tr>
<tr>
<td>state_change</td>
<td>133</td>
</tr>
<tr>
<td>state_change</td>
<td>146</td>
</tr>
<tr>
<td>GOAL</td>
<td>STATE</td>
</tr>
<tr>
<td>state</td>
<td>47</td>
</tr>
<tr>
<td>e</td>
<td>34</td>
</tr>
<tr>
<td>get</td>
<td>53</td>
</tr>
<tr>
<td>set</td>
<td>53</td>
</tr>
<tr>
<td>init</td>
<td>53</td>
</tr>
<tr>
<td>print</td>
<td>53</td>
</tr>
<tr>
<td>status</td>
<td>139</td>
</tr>
<tr>
<td>status</td>
<td>140</td>
</tr>
<tr>
<td>status</td>
<td>50</td>
</tr>
<tr>
<td>status</td>
<td>119</td>
</tr>
<tr>
<td>std_err</td>
<td>41</td>
</tr>
<tr>
<td>std_in</td>
<td>41</td>
</tr>
<tr>
<td>std_out</td>
<td>41</td>
</tr>
<tr>
<td>step_strip_asm_tac</td>
<td>272</td>
</tr>
<tr>
<td>step_strip_tac</td>
<td>272</td>
</tr>
<tr>
<td>stop_mode</td>
<td>56</td>
</tr>
<tr>
<td>use_file_non</td>
<td>46</td>
</tr>
<tr>
<td>read</td>
<td>54</td>
</tr>
<tr>
<td>reset</td>
<td>54</td>
</tr>
<tr>
<td>stopwatch</td>
<td>54</td>
</tr>
<tr>
<td>end_of</td>
<td>41</td>
</tr>
<tr>
<td>String</td>
<td>70</td>
</tr>
<tr>
<td>STRING</td>
<td>109</td>
</tr>
<tr>
<td>get</td>
<td>46</td>
</tr>
<tr>
<td>new</td>
<td>46</td>
</tr>
<tr>
<td>reset</td>
<td>47</td>
</tr>
<tr>
<td>get</td>
<td>47</td>
</tr>
<tr>
<td>set</td>
<td>46</td>
</tr>
<tr>
<td>reset</td>
<td>47</td>
</tr>
<tr>
<td>set</td>
<td>47</td>
</tr>
<tr>
<td>string_conv</td>
<td>194</td>
</tr>
<tr>
<td>StringConv</td>
<td>129</td>
</tr>
<tr>
<td>z</td>
<td>string_conv</td>
</tr>
<tr>
<td>Keyword</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>SucConv</td>
<td>129</td>
</tr>
<tr>
<td>succ_conv</td>
<td>180</td>
</tr>
<tr>
<td>succ</td>
<td>462</td>
</tr>
<tr>
<td>succ</td>
<td>464</td>
</tr>
<tr>
<td>z. succ_def</td>
<td>518</td>
</tr>
<tr>
<td>z. succ_thm</td>
<td>471</td>
</tr>
<tr>
<td>z. succ_thm</td>
<td>523</td>
</tr>
<tr>
<td>z. succ/0_thm</td>
<td>471</td>
</tr>
<tr>
<td>z. succ/0_thm</td>
<td>523</td>
</tr>
<tr>
<td>z. succ/minus n thm</td>
<td>471</td>
</tr>
<tr>
<td>z. succ/minus n thm</td>
<td>523</td>
</tr>
<tr>
<td>z. succ/n thm</td>
<td>471</td>
</tr>
<tr>
<td>z. succ/n thm</td>
<td>523</td>
</tr>
<tr>
<td>get_variant</td>
<td>suffix</td>
</tr>
<tr>
<td>set_variant</td>
<td>suffix</td>
</tr>
<tr>
<td>Lister Support</td>
<td>75</td>
</tr>
<tr>
<td>ReaderWriter Support</td>
<td>55</td>
</tr>
<tr>
<td>swap</td>
<td>29</td>
</tr>
<tr>
<td>SWAP_ASM_CONCL T</td>
<td>276</td>
</tr>
<tr>
<td>SWAP_ASM_CONCL T</td>
<td>256</td>
</tr>
<tr>
<td>swap_asm_concl_tac</td>
<td>275</td>
</tr>
<tr>
<td>swap_asm_concl_tac</td>
<td>256</td>
</tr>
<tr>
<td>SWAP_NTH_ASM CONCL T</td>
<td>276</td>
</tr>
<tr>
<td>SWAP_NTH_ASM CONCL T</td>
<td>256</td>
</tr>
<tr>
<td>swap_nth_asm_concl_tac</td>
<td>275</td>
</tr>
<tr>
<td>swap_nth_asm_concl_tac</td>
<td>256</td>
</tr>
<tr>
<td>z. swap thm</td>
<td>456</td>
</tr>
<tr>
<td>z. swap thm</td>
<td>526</td>
</tr>
<tr>
<td>z. simple swap thm</td>
<td>456</td>
</tr>
<tr>
<td>z. simple swap thm</td>
<td>526</td>
</tr>
<tr>
<td>swap_nth thm</td>
<td>276</td>
</tr>
<tr>
<td>swap_nth thm</td>
<td>256</td>
</tr>
<tr>
<td>eq. sym_asm_tac</td>
<td>242</td>
</tr>
<tr>
<td>eq. sym_conv</td>
<td>164</td>
</tr>
<tr>
<td>eq. sym_conv</td>
<td>57</td>
</tr>
<tr>
<td>eq. sym_double_percent</td>
<td>57</td>
</tr>
<tr>
<td>eq. sym_end_of_input</td>
<td>57</td>
</tr>
<tr>
<td>eq. sym_known</td>
<td>57</td>
</tr>
<tr>
<td>eq. sym_nth_asm_tac</td>
<td>242</td>
</tr>
<tr>
<td>eq. sym_rule</td>
<td>165</td>
</tr>
<tr>
<td>KIEq SymRule</td>
<td>129</td>
</tr>
<tr>
<td>¬ eq. sym_rule</td>
<td>205</td>
</tr>
<tr>
<td>SymUnknownChar</td>
<td>57</td>
</tr>
<tr>
<td>SymUnknownKw</td>
<td>57</td>
</tr>
<tr>
<td>SymWhite</td>
<td>57</td>
</tr>
<tr>
<td>expand symbol</td>
<td>60</td>
</tr>
<tr>
<td>is_same symbol</td>
<td>64</td>
</tr>
<tr>
<td>read symbol</td>
<td>65</td>
</tr>
<tr>
<td>SymbolTable</td>
<td>122</td>
</tr>
<tr>
<td>add_new symbols</td>
<td>60</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
<table>
<thead>
<tr>
<th>KEYWORD INDEX</th>
<th>619</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST_SPEC_NTH</td>
<td>T</td>
</tr>
<tr>
<td>.ASM</td>
<td>T</td>
</tr>
<tr>
<td>LIST_SWAP_ASM</td>
<td>T</td>
</tr>
<tr>
<td>.CONCL</td>
<td></td>
</tr>
<tr>
<td>LIST_SWAP_NTH</td>
<td>T</td>
</tr>
<tr>
<td>.ASM_CONCL</td>
<td>T</td>
</tr>
<tr>
<td>MAP_EVERY</td>
<td>T</td>
</tr>
<tr>
<td>MAP_FIRST</td>
<td>T</td>
</tr>
<tr>
<td>MERGE_PCS</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>105</td>
</tr>
<tr>
<td>mg</td>
<td></td>
</tr>
<tr>
<td>TNone</td>
<td></td>
</tr>
<tr>
<td>ORELSE</td>
<td>T</td>
</tr>
<tr>
<td>PC</td>
<td></td>
</tr>
<tr>
<td>POP_ASM</td>
<td>T</td>
</tr>
<tr>
<td>PROP_EQ</td>
<td></td>
</tr>
<tr>
<td>REPEAT</td>
<td>T</td>
</tr>
<tr>
<td>REPEAT_N</td>
<td>T</td>
</tr>
<tr>
<td>REPEAT_UNTIL</td>
<td>T</td>
</tr>
<tr>
<td>=&gt;</td>
<td></td>
</tr>
<tr>
<td>rotate</td>
<td>T</td>
</tr>
<tr>
<td>rotate</td>
<td></td>
</tr>
<tr>
<td>SPEC_ASM</td>
<td>T</td>
</tr>
<tr>
<td>SPEC_NTH_ASM</td>
<td>T</td>
</tr>
<tr>
<td>STRIP</td>
<td>T</td>
</tr>
<tr>
<td>STRIP_CONCL</td>
<td>T</td>
</tr>
<tr>
<td>SWAP_ASM</td>
<td></td>
</tr>
<tr>
<td>_CONCL</td>
<td>T</td>
</tr>
<tr>
<td>SWAP_NTH_ASM</td>
<td>T</td>
</tr>
<tr>
<td>_CONCL</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>280</td>
</tr>
<tr>
<td>tac</td>
<td></td>
</tr>
<tr>
<td>THEN</td>
<td>T</td>
</tr>
<tr>
<td>THEN_LIST</td>
<td>T</td>
</tr>
<tr>
<td>THEN_TRY</td>
<td>T</td>
</tr>
<tr>
<td>THEN_TRY_LIST</td>
<td>T</td>
</tr>
<tr>
<td>THEN_TRY_LIST</td>
<td></td>
</tr>
<tr>
<td>TOP_ASM</td>
<td>T</td>
</tr>
<tr>
<td>TRY</td>
<td>T</td>
</tr>
<tr>
<td>TRY</td>
<td></td>
</tr>
<tr>
<td>VAR_ELIM_ASM</td>
<td>T</td>
</tr>
<tr>
<td>VAR_ELIM_NTH</td>
<td></td>
</tr>
<tr>
<td>.ASM</td>
<td>T</td>
</tr>
<tr>
<td>=&gt;</td>
<td></td>
</tr>
<tr>
<td>=&gt;</td>
<td></td>
</tr>
<tr>
<td>=&gt;</td>
<td></td>
</tr>
<tr>
<td>=&gt;</td>
<td></td>
</tr>
<tr>
<td>ALL_ASM.FC</td>
<td>T1</td>
</tr>
<tr>
<td>ALL_ASM</td>
<td></td>
</tr>
<tr>
<td>_FORWARD_CHAIN</td>
<td>T1</td>
</tr>
<tr>
<td>ALL.FC</td>
<td>T1</td>
</tr>
<tr>
<td>ALL_FORWARD</td>
<td>T1</td>
</tr>
<tr>
<td>_CHAIN</td>
<td>T1</td>
</tr>
<tr>
<td>ALL_VAR_ELIM</td>
<td>T1</td>
</tr>
<tr>
<td>ALL_VAR_ELIM</td>
<td></td>
</tr>
<tr>
<td>.ASM</td>
<td>T1</td>
</tr>
<tr>
<td>ASM.FC</td>
<td>T1</td>
</tr>
<tr>
<td>ASM.FC</td>
<td></td>
</tr>
<tr>
<td>ASM_FORWARD</td>
<td>T1</td>
</tr>
<tr>
<td>_CHAIN</td>
<td>T1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
**KEYWORD INDEX**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple</td>
<td>621</td>
<td>Term</td>
</tr>
<tr>
<td>z</td>
<td>621</td>
<td>Term</td>
</tr>
<tr>
<td>taut_tac</td>
<td>265</td>
<td>Term</td>
</tr>
<tr>
<td>to_def</td>
<td>507</td>
<td>Term</td>
</tr>
<tr>
<td>TERM</td>
<td>80</td>
<td>Term</td>
</tr>
<tr>
<td>term</td>
<td>133</td>
<td>Term</td>
</tr>
<tr>
<td>term_consts</td>
<td>113</td>
<td>Term</td>
</tr>
<tr>
<td>DEST_TERM</td>
<td>80</td>
<td>Term</td>
</tr>
<tr>
<td>dest_term</td>
<td>86</td>
<td>Term</td>
</tr>
<tr>
<td>DEST_SIMPLETERM</td>
<td>79</td>
<td>Term</td>
</tr>
<tr>
<td>dest_simple</td>
<td>85</td>
<td>Term</td>
</tr>
<tr>
<td>dest_z</td>
<td>362</td>
<td>Term</td>
</tr>
<tr>
<td>format</td>
<td>73</td>
<td>Term</td>
</tr>
<tr>
<td>term</td>
<td>113</td>
<td>Term</td>
</tr>
<tr>
<td>is_term_in</td>
<td>41</td>
<td>Term</td>
</tr>
<tr>
<td>is_z_term</td>
<td>362</td>
<td>Term</td>
</tr>
<tr>
<td>term</td>
<td>114</td>
<td>Term</td>
</tr>
<tr>
<td>term_diff</td>
<td>113</td>
<td>Term</td>
</tr>
<tr>
<td>term_fail</td>
<td>113</td>
<td>Term</td>
</tr>
<tr>
<td>term_fold</td>
<td>113</td>
<td>Term</td>
</tr>
<tr>
<td>format_term</td>
<td>73</td>
<td>Term</td>
</tr>
<tr>
<td>term_grab</td>
<td>113</td>
<td>Term</td>
</tr>
<tr>
<td>term_in</td>
<td>41</td>
<td>Term</td>
</tr>
<tr>
<td>mk_term</td>
<td>105</td>
<td>Term</td>
</tr>
<tr>
<td>mk_simple</td>
<td>104</td>
<td>Term</td>
</tr>
<tr>
<td>mk_z_term</td>
<td>370</td>
<td>Term</td>
</tr>
<tr>
<td>term_of_type</td>
<td>399</td>
<td>Term</td>
</tr>
<tr>
<td>term_order</td>
<td>302</td>
<td>Term</td>
</tr>
<tr>
<td>gen_term_order</td>
<td>300</td>
<td>Term</td>
</tr>
<tr>
<td>make_term_order</td>
<td>300</td>
<td>Term</td>
</tr>
<tr>
<td>gen_term</td>
<td>41</td>
<td>Term</td>
</tr>
<tr>
<td>is_term_out</td>
<td>41</td>
<td>Term</td>
</tr>
<tr>
<td>asm_inst</td>
<td>159</td>
<td>Term</td>
</tr>
<tr>
<td>inst_term_rule</td>
<td>172</td>
<td>Term</td>
</tr>
<tr>
<td>KIInst_termRule</td>
<td>129</td>
<td>Term</td>
</tr>
<tr>
<td>show_term</td>
<td>74</td>
<td>Term</td>
</tr>
<tr>
<td>string_of</td>
<td>109</td>
<td>Term</td>
</tr>
<tr>
<td>term</td>
<td>114</td>
<td>Term</td>
</tr>
<tr>
<td>term_types</td>
<td>114</td>
<td>Term</td>
</tr>
<tr>
<td>term_type</td>
<td>114</td>
<td>Term</td>
</tr>
<tr>
<td>UD_Term</td>
<td>119</td>
<td>Term</td>
</tr>
<tr>
<td>term_unify</td>
<td>314</td>
<td>Term</td>
</tr>
<tr>
<td>term_union</td>
<td>114</td>
<td>Term</td>
</tr>
<tr>
<td>list</td>
<td>99</td>
<td>Term</td>
</tr>
<tr>
<td>term_union</td>
<td>114</td>
<td>Term</td>
</tr>
<tr>
<td>term_vars</td>
<td>114</td>
<td>Term</td>
</tr>
<tr>
<td>Z_TERM</td>
<td>360</td>
<td>Term</td>
</tr>
<tr>
<td>dest_z_term1</td>
<td>382</td>
<td>IF_Z_TERM</td>
</tr>
<tr>
<td>format_term1</td>
<td>73</td>
<td>Term</td>
</tr>
<tr>
<td>is_z_term1</td>
<td>382</td>
<td>Term</td>
</tr>
<tr>
<td>ask_at</td>
<td>60</td>
<td>Terminal</td>
</tr>
<tr>
<td>reset_use</td>
<td>65</td>
<td>gen_find_thm_in</td>
</tr>
<tr>
<td>use</td>
<td>66</td>
<td>theories</td>
</tr>
<tr>
<td>declare</td>
<td>124</td>
<td>theories</td>
</tr>
<tr>
<td>undeclare</td>
<td>128</td>
<td>Delete</td>
</tr>
<tr>
<td>get</td>
<td>126</td>
<td>Delete</td>
</tr>
<tr>
<td>get_current</td>
<td>125</td>
<td>do_in</td>
</tr>
<tr>
<td>get_undeclared</td>
<td>127</td>
<td>Theory</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
duplicate theory 138  modify_goal_state theory 224
force_delete theory 322  pop theory 225
get theory 141  prove theory 260
get_const theory 139  push_goal_state theory 226
get_type theory 142  Save Thm 131
pp theory_hierarchy 50  save theory 146
get theory_info 141  show theory 74
gen theory_list 76  simplify_goal_state theory 228
gen theory_list1 76  string_of theory 147
Lock Theory 131  t theory 199
lock theory 143  asm_rewrite theory_tac 263
get_current theory_name 139  back_chain theory_tac 236
get theory_names 140  bc theory_tac 236
New Theory 131  once_asm_rewrite theory_tac 263
new theory 144  once_rewrite theory_tac 263
Open Theory 131  pure_asm_rewrite theory_tac 263
open theory 146  pure_once_asm theory_tac 263
output theory 77  _rewrite theory_tac 263
print theory 77  pure_once_rewrite theory_tac 263
get theory_status 140  rewrite theory_tac 263
get_current theory_status 139  ⇒ theory_tac 290
theory 147  THM_TAC 231
theory_u_simp_eqn_cxt 383  STRIP theory 147
Unlock Theory 131  top theory 229
unlock theory 148  THEN theory 228
z_output theory 78  top_goal_state theory 263
z_output theory1 76  valid theory 263
z_output theory1 78  z_0_less_times_thm 471
THM 120  z_0_less_times_thm 523
check_is_z_thm 381  z_0_N_thm 465
compact_thm 133  z_0_N_thm 516
cond_thm 161  z_abs_thm 468
Delete Thm 131  z_abs_thm 516
delete_thm 135  z_abs_0_less_thm 471
dest_thm 136  z_abs_0_less_thm 523
get_thm_dict 141  z_abs_eq_0_thm 468
eq_rewrite_thm 164  z_abs_eq_0_thm 516
thm_eqn_cxt 341  z_abs_minus_thm 468
simple_ho_thm_eqn_cxt 340  z_abs_minus_thm 516
f_thm 170  z_abs_neg_thm 471
thm_fail 147  z_abs_neg_thm 523
find_thm 151  z_abs_plus_thm 468
format_thm 73  z_abs_plus_thm 516
get_thm 152  z_abs_pos_thm 471
thm 141  z_abs_pos_thm 523
if_thm 289  z_abs_times_thm 468
if_rewrite_thm 164  z_abs_times_thm 516
gen_find_thm_in_theories 152  z_abs_le_times_thm 471
THM_INFO 151  z_abs_le_times_thm 523
THM_INFO_TEST 151  z_abs_N_thm 468
thm_level 135  z_abs_N_thm 516
ListSave Thm 131  z_app_thm 460
list_save_thm 142  z_app_c_thm 460
KEYWORD INDEX
z F ∩
z F ∩
z F ∪ singleton
z F ∪ singleton
z F P
z F P
z F1
z F1
z a
z a
a
z
assoc
a
z
assoc
a
z
def
z a def
z a one one
z a one one
z a seq x
z a seq x
z a singleton
z a singleton
z a ∈ seq
z a ∈ seq
z a hi
z a hi
z 7→
z N
z N
z N abs minus
z N abs minus
z N cases
z N cases
z N induction
z N induction
z N plus
z N plus
z N plus1
z N plus1
z N times
z N times
z N ¬ minus
z N ¬ minus
z N ¬ plus1
z N ¬ plus1
z ³
z ³
z ³ ran
z ³ ran
z P1
z P1
z C
z C →
z C →
z Z cases
z Z cases
z Z eq

thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm

627
470
523
469
523
469
523
468
516
486
525
487
525
486
525
487
525
487
525
486
525
486
525
487
525
481
465
516
468
516
465
516
465
516
465
516
465
516
466
516
466
516
465
516
453
510
457
526
491
499
481
455
526
465
516
465

z Z eq
z Z induction
z Z induction
z Z minus
z Z minus
z Z one one
z Z one one
z Z plus
z Z plus
z Z subtract
z Z subtract
z Z times
z Z times
z ½
7
z ½
7
⇔
⇔ rewrite
∧
∧ rewrite
∨
∨ rewrite
¬
¬ f
¬ if
¬ rewrite
¬ t
¬ ⇔
¬ ∧
¬ ∨
¬ ¬
¬ ⇒
¬ ∀
¬ ∃
⇒
⇒ rewrite
∀ rewrite
∃ intro
∃ rewrite
∃1
β rewrite
Z z minus
Z z minus
Z z one one
Z z one one
Z z plus
Z z plus
Z z subtract
Z z subtract
Z z times
Z z times
format
z id
z id
z plus assoc
z plus assoc
z seq
z seq
z seq induction

thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm
thm1
thm1
thm1
thm1
thm1
thm1
thm1
thm1

516
465
516
449
521
449
521
448
521
449
521
448
521
453
510
289
164
203
164
205
164
206
289
289
164
206
289
289
289
289
289
207
208
289
164
164
213
164
216
164
449
521
449
521
449
521
449
521
449
521
73
456
526
465
516
486
525
487

c Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
°


thm1

\text{current_ad_\_cd}_{} \text{thsms}

\text{current_ad_\_vs}_{} \text{thsms}

\text{get}_{} \text{thsms}

\text{get_\_cd}_{} \text{thsms}

\text{get_\_vs}_{} \text{thsms}

\text{LS}_{} \text{Thms}

\text{LSAD}_{} \text{Thms}

\text{pp_\_set_eval_ad_\_cd}_{} \text{thsms}

\text{pp_\_set_eval_ad_\_vs}_{} \text{thsms}

\text{set_\_cd}_{} \text{thsms}

\text{set_\_vs}_{} \text{thsms}

\text{subgoal_package}_{} \text{ti_context}

\text{time_app}_{} 54

\text{TIMED}_{} 54

\text{TIMER\_UNITS}_{} 54

\text{times_assoc_thm}_{} 466

\text{times_assoc_thm}_{} 516

\text{times_assoc_thm}_{} 476

\text{times_assoc_thm}_{} 516

\text{times_assoc_thm}_{} 533

\text{times_assoc_thm}_{} 516

\text{times_assoc_thm}_{} 533

\text{times_assoc_thm}_{} 477

\text{times_assoc_thm}_{} 533

\text{times_assoc_thm}_{} 466

\text{times_conw}_{} 518

\text{times_conw}_{} 534

\text{times_conw}_{} 518

\text{times_def}_{} 535

\text{times_subgoal}_{} 513

\text{times_subgoal}_{} 530

\text{times_subgoal}_{} 516

\text{times_subgoal}_{} 513

\text{times_subgoal}_{} 530

\text{times_subgoal}_{} 466

\text{times_order_thm}_{} 516

\text{times_order_thm}_{} 466

\text{times_order_thm}_{} 533

\text{times_plus_distrib}_{} 466

\text{times_plus_distrib}_{} 476

\text{times_plus_distrib}_{} 533

\text{times_plus_distrib}_{} 471

\text{times_plus_distrib}_{} 531

\text{times_plus_distrib}_{} 514

\text{times_plus_distrib}_{} 523

\text{times_plus_distrib}_{} 477

\text{times_plus_distrib}_{} 533

\text{times_plus_distrib}_{} 476

\text{times_plus_distrib}_{} 516

\text{times_plus_distrib}_{} 471

\text{times_plus_distrib}_{} 523

\text{times_plus_distrib}_{} 468

\text{times_plus_distrib}_{} 516

\text{times_plus_distrib}_{} 471

\text{times_plus_distrib}_{} 523

\text{times_plus_distrib}_{} 466

\text{times_plus_distrib}_{} 516

\text{times_plus_distrib}_{} 466

\text{times_plus_distrib}_{} 516

\text{times_plus_distrib}_{} 466

\text{times_plus_distrib}_{} 516

\text{times_plus_distrib}_{} 466

\text{times_plus_distrib}_{} 516

\text{times_plus_distrib}_{} 466

\text{times_plus_distrib}_{} 516

\text{times_plus_distrib}_{} 466

\text{times_plus_distrib}_{} 516

\text{times_plus_distrib}_{} 466

\text{times_plus_distrib}_{} 516

\text{times_plus_distrib}_{} 466

\text{times_plus_distrib}_{} 516

\text{times_plus_distrib}_{} 466
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>z, R, <code>times_thm</code></td>
<td>533</td>
</tr>
<tr>
<td>z, R, <code>0_less_0_less</code>, <code>times_thm</code></td>
<td>476</td>
</tr>
<tr>
<td>z, R, <code>0_less_0_less</code>, <code>times_thm</code></td>
<td>532</td>
</tr>
<tr>
<td>z, N, <code>times_thm</code></td>
<td>466</td>
</tr>
<tr>
<td>z, N, <code>times_thm</code></td>
<td>516</td>
</tr>
<tr>
<td>z, Z, <code>times_thm</code></td>
<td>448</td>
</tr>
<tr>
<td>z, Z, <code>times_thm</code></td>
<td>521</td>
</tr>
<tr>
<td>Z, <code>times_thm</code></td>
<td>449</td>
</tr>
<tr>
<td>Z, <code>times_thm</code></td>
<td>521</td>
</tr>
<tr>
<td>z, R, <code>times_unit_thm</code>, <code>trans_thm</code></td>
<td>533</td>
</tr>
<tr>
<td>z, <code>times0_thm</code></td>
<td>466</td>
</tr>
<tr>
<td>z, <code>times0_thm</code></td>
<td>516</td>
</tr>
<tr>
<td>z, <code>times0_thm</code></td>
<td>448</td>
</tr>
<tr>
<td>z, <code>times0_thm</code></td>
<td>521</td>
</tr>
<tr>
<td><code>trans_for_output</code></td>
<td>66</td>
</tr>
<tr>
<td><code>fun</code></td>
<td>28</td>
</tr>
<tr>
<td><code>z</code></td>
<td>370</td>
</tr>
<tr>
<td><code>mk_z</code></td>
<td>370</td>
</tr>
<tr>
<td><code>True</code></td>
<td>360</td>
</tr>
<tr>
<td><code>TRY</code></td>
<td>280</td>
</tr>
<tr>
<td><code>TRY_C</code></td>
<td>199</td>
</tr>
<tr>
<td><code>TRY_C</code></td>
<td>199</td>
</tr>
<tr>
<td><code>TRY_T</code></td>
<td>280</td>
</tr>
<tr>
<td><code>TRY_T</code></td>
<td>279</td>
</tr>
<tr>
<td><code>TRY</code></td>
<td>279</td>
</tr>
<tr>
<td><code>TRY_TTCL</code></td>
<td>280</td>
</tr>
<tr>
<td><code>TRY_TTCL</code></td>
<td>279</td>
</tr>
<tr>
<td><code>TSAncestor</code></td>
<td>119</td>
</tr>
<tr>
<td><code>TSDeleted</code></td>
<td>119</td>
</tr>
<tr>
<td><code>TSLocked</code></td>
<td>119</td>
</tr>
<tr>
<td><code>TSNormal</code></td>
<td>119</td>
</tr>
<tr>
<td><code>TS</code></td>
<td>221</td>
</tr>
<tr>
<td><code>all_submatch</code></td>
<td>152</td>
</tr>
<tr>
<td><code>all_substring</code></td>
<td>152</td>
</tr>
<tr>
<td><code>all_subterm</code></td>
<td>152</td>
</tr>
<tr>
<td><code>any_submatch</code></td>
<td>152</td>
</tr>
<tr>
<td><code>any_substring</code></td>
<td>152</td>
</tr>
<tr>
<td><code>any_subterm</code></td>
<td>152</td>
</tr>
<tr>
<td><code>TTAxiom</code></td>
<td>151</td>
</tr>
<tr>
<td><code>TTDefn</code></td>
<td>151</td>
</tr>
<tr>
<td><code>TTSaved</code></td>
<td>151</td>
</tr>
<tr>
<td><code>EVERY</code></td>
<td>242</td>
</tr>
<tr>
<td><code>FIRST</code></td>
<td>244</td>
</tr>
<tr>
<td><code>ORELSE</code></td>
<td>257</td>
</tr>
<tr>
<td><code>REPEAT</code></td>
<td>262</td>
</tr>
<tr>
<td><code>THEN</code></td>
<td>279</td>
</tr>
<tr>
<td><code>THEN_TRY</code></td>
<td>279</td>
</tr>
<tr>
<td><code>TRY</code></td>
<td>279</td>
</tr>
<tr>
<td><code>TRY_TTCL</code></td>
<td>280</td>
</tr>
<tr>
<td><code>tuple</code></td>
<td>370</td>
</tr>
<tr>
<td><code>tuple_eq_conv</code></td>
<td>506</td>
</tr>
<tr>
<td><code>tuple_eq_conv1</code></td>
<td>507</td>
</tr>
<tr>
<td><code>tuple_intro_conv</code></td>
<td>507</td>
</tr>
<tr>
<td><code>is_z</code></td>
<td>370</td>
</tr>
<tr>
<td><code>tuple_lang_eq_conv</code></td>
<td>427</td>
</tr>
<tr>
<td><code>tuple_lang_intro_conv</code></td>
<td>427</td>
</tr>
<tr>
<td><code>mk_z</code></td>
<td>370</td>
</tr>
<tr>
<td><code>tuple_type</code></td>
<td>370</td>
</tr>
<tr>
<td><code>is_z</code></td>
<td>370</td>
</tr>
<tr>
<td><code>tuple_type</code></td>
<td>370</td>
</tr>
<tr>
<td><code>mk_z</code></td>
<td>370</td>
</tr>
<tr>
<td><code>tuple_type</code></td>
<td>370</td>
</tr>
<tr>
<td><code>Z</code></td>
<td>360</td>
</tr>
</tbody>
</table>
term var 114 X → Y 462
λ_ varstruct_conv 220 X → Y 462
varstruct_variant 200 X → Y 489
Vartype 79 X → Y 489
dest_ vartype 87 X → Y 452
is_ vartype 93 X → Y 452
mk_ vartype 105 X → Y 452
PP Vector 42 X → Y 452
HT Vector 70 X → Y 452
current_ad_ vs_thms 331 (._o_)[X, Y, Z] 478
get_ vs_thms 340 (._o_)[X, Y, Z] 478
pp' set_eval_ad_ vs_thms 331 (._o_)[X, Y] 478
set_ vs_thms 340 (._o_)[X, Y] 478
warn 52 (._o_)[X, Y] 478
ONCE_MAP_ WARN_C 177 (._o_)[X, Y] 478
local_ warn 64 (._o_)[X, Y] 478
ilformed_rewrite_ warning 153 (._o_)[X, Y] 478
tactic_subgoal_ warning 222 (._o_)[X, Y] 478
which is_ white 64 dom[X, Y] 478
Sym White 57 first[X, Y] 489
fail_ with_canon 166 ran[X, Y] 478
fail_ with_conv 166 second[X, Y] 489
fail_ with_tac 243 z_0_less_times_thm 497
FAIL_ WITH_ THEN 243 z_0_less_times_thm 523
abandon_reader_ writer 59 z_0_N_thm 485
HOLReader Writer 55 z_0_N_thm 516
Reader Writer 55 z_abs_0_less_thm 471
Reader WriterSupport 55 z_abs_0_less_thm 523
bag X 450 z_abs_conv 518
id X 478 dest_ z_abs 513
iseq X 483 z_abs_eq_0_thm 468
seq X 483 z_abs_eq_0_thm 516
seq_ X 483 is_ z_abs 513
z_seq_seq_ x_thm 487 z_abs_minus_thm 468
z_seq_seq_ x_thm 525 z_abs_minus_thm 516
z_singleton_seq_ x_thm 487 z_abs_minus_thm 468
z_singleton_seq_ x_thm 525 z_abs_minus_thm 516
z_ seq_ x_thm 487 z_abs_minus_thm 468
z_ seq_ x_thm 525 z_abs_minus_thm 516
z_ partition_ [I_ X] 483 z_abs_times_thm 468
ZApp 360
(disjoint_ [I_ X] 483 z_abs_times_thm 516
xpp 11 dest_ z_app 363
X → Y 452
z_app 363
z_app conv 421
z_app_eq_tac 422
is_ z_app 363
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZFloat</td>
<td>360</td>
</tr>
<tr>
<td>z_float_conv</td>
<td>531</td>
</tr>
<tr>
<td>dest. z_float</td>
<td>365</td>
</tr>
<tr>
<td>is. z_float</td>
<td>365</td>
</tr>
<tr>
<td>mk. z_float</td>
<td>365</td>
</tr>
<tr>
<td>z_float_thm</td>
<td>477</td>
</tr>
<tr>
<td>z_float_thm</td>
<td>533</td>
</tr>
<tr>
<td>z_front_def</td>
<td>522</td>
</tr>
<tr>
<td>, z_fun_alg</td>
<td>509</td>
</tr>
<tr>
<td>, z_fun_app_clauses</td>
<td>454</td>
</tr>
<tr>
<td>, z_fun_app_clauses</td>
<td>510</td>
</tr>
<tr>
<td>, z_fun_dom_clauses</td>
<td>454</td>
</tr>
<tr>
<td>, z_fun_dom_clauses</td>
<td>510</td>
</tr>
<tr>
<td>, z_fun_ext</td>
<td>509</td>
</tr>
<tr>
<td>, z_fun_run_clauses</td>
<td>454</td>
</tr>
<tr>
<td>, z_fun_run_clauses</td>
<td>510</td>
</tr>
<tr>
<td>, z_fun_e_clauses</td>
<td>454</td>
</tr>
<tr>
<td>, z_fun_e_clauses</td>
<td>510</td>
</tr>
<tr>
<td>ZFunctions</td>
<td>507</td>
</tr>
<tr>
<td>ZFunctions1</td>
<td>522</td>
</tr>
<tr>
<td>dest. z_given_type</td>
<td>366</td>
</tr>
<tr>
<td>is. z_given_type</td>
<td>366</td>
</tr>
<tr>
<td>mk. z_given_type</td>
<td>366</td>
</tr>
<tr>
<td>check_is. z_greater</td>
<td>513</td>
</tr>
<tr>
<td>is. z_greater</td>
<td>513</td>
</tr>
<tr>
<td>mk. z_greater</td>
<td>513</td>
</tr>
<tr>
<td>, z_greater_less_conv</td>
<td>518</td>
</tr>
<tr>
<td>dest. z_greater</td>
<td>518</td>
</tr>
<tr>
<td>is. z_greater</td>
<td>518</td>
</tr>
<tr>
<td>mk. z_greater</td>
<td>518</td>
</tr>
<tr>
<td>, z_guillems_thm</td>
<td>491</td>
</tr>
<tr>
<td>, z_guillems_thm</td>
<td>499</td>
</tr>
<tr>
<td>dest. z_guar</td>
<td>366</td>
</tr>
<tr>
<td>is. z_guar</td>
<td>366</td>
</tr>
<tr>
<td>mk. z_guar</td>
<td>366</td>
</tr>
<tr>
<td>, z_h_schema_conv</td>
<td>434</td>
</tr>
<tr>
<td>dest. z_h_schema</td>
<td>366</td>
</tr>
<tr>
<td>is. z_h_schema</td>
<td>366</td>
</tr>
<tr>
<td>mk. z_h_schema</td>
<td>366</td>
</tr>
<tr>
<td>, z_h_schema_pred</td>
<td>366</td>
</tr>
<tr>
<td>, z_h_schema_pred</td>
<td>366</td>
</tr>
<tr>
<td>, conv</td>
<td>434</td>
</tr>
<tr>
<td>, hash_def</td>
<td>518</td>
</tr>
<tr>
<td>, head_def</td>
<td>522</td>
</tr>
<tr>
<td>ZHide. z_hide</td>
<td>360</td>
</tr>
<tr>
<td>, z_hide_conv</td>
<td>434</td>
</tr>
<tr>
<td>dest. z_hide</td>
<td>366</td>
</tr>
<tr>
<td>is. z_hide</td>
<td>366</td>
</tr>
<tr>
<td>mk. z_hide</td>
<td>366</td>
</tr>
<tr>
<td>, id_clauses</td>
<td>481</td>
</tr>
<tr>
<td>, id_def</td>
<td>507</td>
</tr>
<tr>
<td>, id_thm</td>
<td>481</td>
</tr>
<tr>
<td>, id_thm1</td>
<td>456</td>
</tr>
<tr>
<td>, id_thm1</td>
<td>526</td>
</tr>
<tr>
<td>z.id_{→,thm}</td>
<td>456</td>
</tr>
<tr>
<td>z.id_{→,thm}</td>
<td>526</td>
</tr>
<tr>
<td>dest. z_if</td>
<td>496</td>
</tr>
<tr>
<td>is. z_if</td>
<td>496</td>
</tr>
<tr>
<td>mk. z_if</td>
<td>496</td>
</tr>
<tr>
<td>, z_if_thm</td>
<td>491</td>
</tr>
<tr>
<td>, z_if_thm</td>
<td>499</td>
</tr>
<tr>
<td>, in_def</td>
<td>528</td>
</tr>
<tr>
<td>, inequality_def</td>
<td>518</td>
</tr>
<tr>
<td>ZInt</td>
<td>360</td>
</tr>
<tr>
<td>dest. z_int_homomorphism</td>
<td>366</td>
</tr>
<tr>
<td>, z_int_homomorphism</td>
<td>465</td>
</tr>
<tr>
<td>, z_int_homomorphism</td>
<td>516</td>
</tr>
<tr>
<td>is. z_int</td>
<td>366</td>
</tr>
<tr>
<td>mk. z_int</td>
<td>366</td>
</tr>
<tr>
<td>, z_intro_gen_pred_tac</td>
<td>394</td>
</tr>
<tr>
<td>, z_intro_Y_tac</td>
<td>394</td>
</tr>
<tr>
<td>is. z</td>
<td>382</td>
</tr>
<tr>
<td>, iseq_def</td>
<td>522</td>
</tr>
<tr>
<td>, items_def</td>
<td>528</td>
</tr>
<tr>
<td>, iter_def</td>
<td>518</td>
</tr>
<tr>
<td>, language</td>
<td>503</td>
</tr>
<tr>
<td>, language_ext</td>
<td>504</td>
</tr>
<tr>
<td>, last_def</td>
<td>522</td>
</tr>
<tr>
<td>Z.LEFT_C</td>
<td>424</td>
</tr>
<tr>
<td>, z_less_cases_thm</td>
<td>469</td>
</tr>
<tr>
<td>, z_less_cases_thm</td>
<td>523</td>
</tr>
<tr>
<td>, z_less_clauses</td>
<td>467</td>
</tr>
<tr>
<td>, z_less_clauses</td>
<td>516</td>
</tr>
<tr>
<td>, z_less_conv</td>
<td>518</td>
</tr>
<tr>
<td>dest. z_less</td>
<td>513</td>
</tr>
<tr>
<td>is. z_less</td>
<td>513</td>
</tr>
<tr>
<td>mk. z_less</td>
<td>514</td>
</tr>
<tr>
<td>, z_less_plus1_thm</td>
<td>468</td>
</tr>
<tr>
<td>, z_less_trans_thm</td>
<td>467</td>
</tr>
<tr>
<td>, z_less_trans_thm</td>
<td>516</td>
</tr>
<tr>
<td>, z_less_trans_thm</td>
<td>467</td>
</tr>
<tr>
<td>, z_less_&lt;=_trans_thm</td>
<td>516</td>
</tr>
<tr>
<td>, z_less_&lt;=_trans_thm</td>
<td>449</td>
</tr>
<tr>
<td>, z_less_Zless_thm</td>
<td>521</td>
</tr>
<tr>
<td>ZLet</td>
<td>360</td>
</tr>
<tr>
<td>, z_let_conv</td>
<td>424</td>
</tr>
<tr>
<td>, z_let_conv1</td>
<td>424</td>
</tr>
<tr>
<td>dest. z_let</td>
<td>367</td>
</tr>
<tr>
<td>is. z_let</td>
<td>367</td>
</tr>
<tr>
<td>mk. z_let</td>
<td>367</td>
</tr>
<tr>
<td>, ZLibrary</td>
<td>527</td>
</tr>
<tr>
<td>, z_library</td>
<td>527</td>
</tr>
<tr>
<td>, z_library_ext</td>
<td>527</td>
</tr>
<tr>
<td>, z_library1</td>
<td>528</td>
</tr>
<tr>
<td>, z_library1_ext</td>
<td>528</td>
</tr>
<tr>
<td>, z_lin_arith</td>
<td>519</td>
</tr>
<tr>
<td>, z_lin_arith</td>
<td>520</td>
</tr>
<tr>
<td>, z_lin_arith1</td>
<td>519</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>z_simple_swap ↦ thm</td>
<td>526</td>
</tr>
<tr>
<td>z_singleton_app_thm</td>
<td>458</td>
</tr>
<tr>
<td>z_singleton_app_thm</td>
<td>526</td>
</tr>
<tr>
<td>z_singleton_seq_thm</td>
<td>486</td>
</tr>
<tr>
<td>z_singleton_seq_thm</td>
<td>525</td>
</tr>
<tr>
<td>z_singleton_seq_x_thm</td>
<td>487</td>
</tr>
<tr>
<td>z_singleton_seq_x_thm</td>
<td>525</td>
</tr>
<tr>
<td>z_size_0_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_0_thm</td>
<td>523</td>
</tr>
<tr>
<td>z_size_1_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_1_thm</td>
<td>523</td>
</tr>
<tr>
<td>z_size_2_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_2_thm</td>
<td>523</td>
</tr>
<tr>
<td>z_size_diff_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_diff_thm</td>
<td>523</td>
</tr>
<tr>
<td>z_size_dot_conv</td>
<td>525</td>
</tr>
<tr>
<td>z_size_dot_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_dot_thm</td>
<td>523</td>
</tr>
<tr>
<td>z_size_dot_thm1</td>
<td>470</td>
</tr>
<tr>
<td>z_size_dot_thm1</td>
<td>523</td>
</tr>
<tr>
<td>z_size_empty_thm</td>
<td>469</td>
</tr>
<tr>
<td>z_size_empty_thm</td>
<td>523</td>
</tr>
<tr>
<td>z_size_eq_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_eq_thm</td>
<td>523</td>
</tr>
<tr>
<td>z_size_mono_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_mono_thm</td>
<td>523</td>
</tr>
<tr>
<td>z_size_pair_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_pair_thm</td>
<td>523</td>
</tr>
<tr>
<td>z_size_seq_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_seq_thm</td>
<td>523</td>
</tr>
<tr>
<td>z_size_seq_thm1</td>
<td>486</td>
</tr>
<tr>
<td>z_size_seq_thm1</td>
<td>525</td>
</tr>
<tr>
<td>z_size_seq_thm2</td>
<td>486</td>
</tr>
<tr>
<td>z_size_seq_thm2</td>
<td>525</td>
</tr>
<tr>
<td>z_size_seq_N_thm</td>
<td>486</td>
</tr>
<tr>
<td>z_size_seq_N_thm</td>
<td>525</td>
</tr>
<tr>
<td>z_size_seq_conv</td>
<td>524</td>
</tr>
<tr>
<td>z_size_seq_length_thm</td>
<td>488</td>
</tr>
<tr>
<td>z_size_seq_length_thm</td>
<td>525</td>
</tr>
<tr>
<td>z_size_seq_thm</td>
<td>488</td>
</tr>
<tr>
<td>z_size_seq_thm</td>
<td>525</td>
</tr>
<tr>
<td>z_size_singleton_seq_thm</td>
<td>487</td>
</tr>
<tr>
<td>z_size_singleton_seq_thm</td>
<td>525</td>
</tr>
<tr>
<td>z_size_singleton_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_singleton_thm</td>
<td>523</td>
</tr>
<tr>
<td>z_size_+_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_+_thm</td>
<td>523</td>
</tr>
<tr>
<td>z_size_×_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_×_thm</td>
<td>523</td>
</tr>
<tr>
<td>z_size_≤_1_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_≤_1_thm</td>
<td>523</td>
</tr>
<tr>
<td>z_size_∪_singleton_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_∪_singleton_thm</td>
<td>523</td>
</tr>
<tr>
<td>z_size_∪_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_∪_thm</td>
<td>523</td>
</tr>
<tr>
<td>z_size_≤_thm</td>
<td>470</td>
</tr>
<tr>
<td>z_size_≤_thm</td>
<td>523</td>
</tr>
<tr>
<td>Z_SIZE</td>
<td>367</td>
</tr>
<tr>
<td>dest</td>
<td>370</td>
</tr>
<tr>
<td>is</td>
<td>370</td>
</tr>
<tr>
<td>mk</td>
<td>370</td>
</tr>
<tr>
<td>not</td>
<td>437</td>
</tr>
<tr>
<td>dest</td>
<td>382</td>
</tr>
<tr>
<td>is</td>
<td>513</td>
</tr>
<tr>
<td>is</td>
<td>513</td>
</tr>
<tr>
<td>Z</td>
<td>518</td>
</tr>
<tr>
<td>Z</td>
<td>514</td>
</tr>
<tr>
<td>Z</td>
<td>449</td>
</tr>
<tr>
<td>Z</td>
<td>521</td>
</tr>
<tr>
<td>Z</td>
<td>518</td>
</tr>
<tr>
<td>Z</td>
<td>471</td>
</tr>
<tr>
<td>Z</td>
<td>523</td>
</tr>
<tr>
<td>Z</td>
<td>471</td>
</tr>
<tr>
<td>Z</td>
<td>523</td>
</tr>
<tr>
<td>Z</td>
<td>471</td>
</tr>
<tr>
<td>Z</td>
<td>523</td>
</tr>
<tr>
<td>Z</td>
<td>456</td>
</tr>
<tr>
<td>Z</td>
<td>526</td>
</tr>
<tr>
<td>CHECK_IS</td>
<td>381</td>
</tr>
<tr>
<td>Z_T</td>
<td>522</td>
</tr>
<tr>
<td>z_fail_def</td>
<td>507</td>
</tr>
<tr>
<td>Z_TERM</td>
<td>360</td>
</tr>
<tr>
<td>basic_dest</td>
<td>361</td>
</tr>
<tr>
<td>check_is</td>
<td>381</td>
</tr>
<tr>
<td>dest</td>
<td>362</td>
</tr>
<tr>
<td>is</td>
<td>362</td>
</tr>
<tr>
<td>mk</td>
<td>370</td>
</tr>
<tr>
<td>dest</td>
<td>399</td>
</tr>
<tr>
<td>is</td>
<td>382</td>
</tr>
<tr>
<td>is</td>
<td>382</td>
</tr>
<tr>
<td>check_is</td>
<td>381</td>
</tr>
<tr>
<td>z_times_assoc_thm</td>
<td>466</td>
</tr>
<tr>
<td>z_times_assoc_thm</td>
<td>516</td>
</tr>
<tr>
<td>z_times_assoc_thm1</td>
<td>466</td>
</tr>
<tr>
<td>z_times_assoc_thm1</td>
<td>516</td>
</tr>
<tr>
<td>z_times_clauses</td>
<td>466</td>
</tr>
<tr>
<td>z_times_clauses</td>
<td>516</td>
</tr>
<tr>
<td>z_times_clauses</td>
<td>516</td>
</tr>
<tr>
<td>z_times_comm_thm</td>
<td>466</td>
</tr>
</tbody>
</table>
644

KEYWORD INDEX

dest
is
mk

z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z
z

a singleton thm1
a thm
a thm
a ∈ seq thm
a ∈ seq thm
a ∈ seq thm1
a ∈ seq thm1
a hi thm
a hi thm
» def
7→ def
7 thm
→
N abs minus thm
N abs minus thm
N cases thm
N cases thm
N def
N induction tac
N induction thm
N induction thm
N plus conv
N plus thm
N plus thm
N plus1 thm
N plus1 thm
N thm
N thm
N times conv
N times thm
N times thm
N ¬ minus thm
N ¬ minus thm
N ¬ plus1 thm
N ¬ plus1 thm
N1 def
³ clauses
³ clauses
³ def
³ ran thm
³ ran thm
³ thm
³ thm
³ thm1
³ thm1
P clauses
P clauses

P
P
P
P1 clauses
P1 clauses
P1 def
P1 thm
P1 thm
C clauses
C def

525
486
525
486
525
486
525
487
525
522
507
481
468
516
465
516
518
517
465
516
518
465
516
465
516
465
516
518
466
516
466
516
465
516
518
454
510
509
457
526
453
510
456
526
492
499
377
377
377
492
499
498
491
499
481
507

dest
is
mk

Z

z C thm
z C → thm
z C → thm
z ¹ def
z ¹s conv
z ¹s
z ¹s
z ¹s
z
z Z
z Z cases thm
z Z cases thm
z Z cases thm1
z Z cases thm1
z Z consistent
z Z conv
z Z def
z Z def
z Z eq conv
z Z eq thm
z Z eq thm
z Z induction tac
z Z induction thm
z Z induction thm
z Z minus thm
z Z minus thm
z Z one one thm
z Z one one thm
z Z plus thm
z Z plus thm
z Z subtract thm
z Z subtract thm
z Z times thm
z Z times thm
z ½
7 clauses
z ½
7 clauses
z ½
7 def
z ½
7 thm
z ½
7 thm
z ½
7 thm1
z ½
7 thm1
Z 0 Float
Z 0 Float
z 0 guillemets def
z 0 if def
Z 0 Int
z 0 int def
z 0 int def
z 0 underlining brackets
def
z 0 Π def
zed
zed list
zero
ZGVar
ZHSchema
zip
ZLVar

481
455
526
522
445
377
377
377
448
448
465
516
466
516
448
521
518
521
518
465
516
519
465
516
449
521
449
521
448
521
449
521
448
521
454
510
509
453
510
456
526
472
474
498
498
463
463
518
498
498
7
11
39
360
360
27
360

c Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030
°


KEYWORD INDEX
<table>
<thead>
<tr>
<th>mk₂₂₀</th>
<th>s</th>
<th>377</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td></td>
<td>462</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>464</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>465</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>516</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>466</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>516</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>448</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>521</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>518</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>521</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>518</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>465</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>516</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>521</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>531</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>519</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>465</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>516</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>449</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>521</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>449</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>521</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>449</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>521</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>449</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>521</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>449</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>521</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>449</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>521</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>450</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>450</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>450</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>450</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>452</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - Z REFERENCE MANUALUSR030