ProofPower

HOL 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
5.3 Higher-order Matching .................................................. 120

6 THE MANAGEMENT OF THEORIES AND THEOREMS .................. 121
6.1 Standard ML Type Definitions ........................................... 121
6.2 Symbol Table ............................................................. 124
6.3 The Kernel Interface ..................................................... 131
6.4 Conjectures Database .................................................... 151
6.5 Theorem Finder ........................................................... 153

7 SPECIFICATION IN HOL .................................................... 155
7.1 Constant Specification .................................................... 155
7.2 Automatic Existence Proofs .............................................. 160
7.3 Product Types ............................................................. 167

8 PROOF IN HOL .............................................................. 173
8.1 General Inference Rules .................................................. 173
8.2 Subgoal Package ......................................................... 241
8.3 General Tactics and Tacticals ......................................... 251
8.4 Propositional Equational Reasoning ................................... 313
8.5 Algebraic Normalisation .................................................. 317
8.6 First Order Resolution ................................................... 324
8.7 Quantifier Elimination Toolkit ......................................... 335
8.8 Proof Contexts ............................................................ 338

9 THEORIES ................................................................. 381
9.1 Theory Listings ........................................................... 381
9.1.1 The Theory basic_hol ............................................... 382
9.1.2 The Theory bin_rel ................................................... 383
9.1.3 The Theory char ....................................................... 388
9.1.4 The Theory combin .................................................... 389
9.1.5 The Theory dyadic .................................................... 390
9.1.6 The Theory fin_set .................................................... 393
9.1.7 The Theory fun_rel .................................................... 396
9.1.8 The Theory fun_rel_thms ............................................ 398
9.1.9 The Theory hol ........................................................ 399
9.1.10 The Theory init ....................................................... 400
9.1.11 The Theory list ....................................................... 401
9.1.12 The Theory log ....................................................... 404
9.1.13 The Theory min ....................................................... 406
9.1.14 The Theory misc ..................................................... 407
9.1.15 The Theory one ........................................................ 410
9.1.16 The Theory orders ................................................... 411
9.1.17 The Theory pair ...................................................... 416
9.1.18 The Theory seq ....................................................... 418
9.1.19 The Theory sets ....................................................... 420
9.1.20 The Theory set_thms ............................................... 424
9.1.21 The Theory sum ...................................................... 425
9.1.22 The Theory R ......................................................... 427
9.1.23 The Theory N ......................................................... 445
9.1.24 The Theory Z ........................................................ 450
9.2 Theory Related ML Values .............................................. 458
9.2.1 Basic Definitions And Axioms .................................... 458
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 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.
UNIX INTERFACES

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

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.
Description  

PP_list is used to obtain selected information from a ProofPower database.

In the first form of use, where a list of one or more theory names is specified, PP_list uses 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 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 output_theory is used. Otherwise it will attempt to use a function named:

<language in lower case>_output_theory

and only if that doesn’t exist will it use output_theory. All but the first language will be ignored.

If the -l lang option is given then it will take the language code of all theories given to be 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 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, 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 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  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  zed_list, hol_list, pp, pp_make_database
pp_make_database \[\text{[-c][-v][-f]} [-p \text{parentdatabase}][\#\text{parenttheory}]\text{ newdatabase}[\#\text{cachetheory}]\]

**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 clashes 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

[pp −d database[#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, theoryname gives the name of a theory to be made the current theory at the start of the session. If theoryname is not specified, then current theory will be set to the theory current when the database was last saved by 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 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 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., pp −d hol −− −h 32000. The environment variable PP_COMPILER 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 PPLINELENGTH, if set, determines the initial value of the string control line_length. This gives the line length used by various listing facilities, e.g., print_theory and output_theory. In interactive use, the xpp interface will set 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
**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

---

**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`
SML
conv ascii [-r] [-K] [-k keyword_file_name] <filename> ...
conv extended [-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(1) or cp(1) 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

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

\[\copyright\] Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUAL USR029
**doctex**

- **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 caller’s 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`

---

**texdvi**

- **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`
PROGRAMMING UTILITIES
## 2.1 Error Management

**SML**

```sml
signature BasicError = sig

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

**SML**

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

Description These exception 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**

```sml
type MESSAGE

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

Uses Obscure debugging situations.
```

**SML**

```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**

```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

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

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

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.
**SML**

|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 * |

---

**SML**

|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

---

**SML**

|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.

---

**SML**

|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.

---

**SML**

|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

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
2.3. Lists

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

SML

```sml
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

SML

```sml
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

SML

```sml
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**

```
hd (a :: x) = a
tl (a :: x) = x
```

**Errors**

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

SML

```sml
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

SML

```sml
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 \(a = b\).

SML

```sml
val is_Nil : 'a OPT -> bool
```

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

**Definition**

```
is_Nil Nil = true
| is_Nil _ = false
```

SML

```sml
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

val lassoc1 : ("a * "a) list -> "a -> "a;

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

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

SML

val lassoc2 : ("a * 'b) list -> ('a -> 'b) -> "a -> 'b;

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

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

SML

val lassoc3 : ("a * 'b) list -> "a -> 'b;

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

Errors

1005 No such value in association list

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

SML

val lassoc4 : ("a * 'b) list -> 'b -> "a -> 'b;

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

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

SML

val lassoc5 : ("a * 'b) list -> "a -> 'b OPT;

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

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

SML

val length : 'a list -> int;

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

SML

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

Description list less what is the list obtained by deleting all members of \( list \) which are equal to \( what \).

See Also drop
**Chapter 2. PROGRAMMING UTILITIES**

SML

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

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

**Definition**

\[
\text{list_cup} \ [\text{list0, list1, ..., listn}] = \\
\text{list0} \ \text{cup} \ (\text{list1} \ \text{cup} \ ... (\text{listn} \ \text{cup} \ []))
\]

**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**

\[
\text{fun list\_overwrite olist = (} \\
\quad \text{fold (fn} (l1, l2) => l2 overwrite l1) \ olist \ \text{alist}
\]

**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**

\[
\text{fun list\_roverwrite olist = (} \\
\quad \text{fold (fn} (l1, l2) => l2 roverwrite l1) \ olist \ \text{alist}
\]

**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**

\[
\text{list\_union eq} \ [\text{list0, list1, ..., listn}] = \\
\quad \text{union eq} \ \text{list0} \ (\text{union eq} \ \text{list1} \ (\text{... (union eq listn \ []) ...}))
\]

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

**See Also** contains, present
2.3. Lists

SML

```sml
val nth : int -> 'a list -> 'a;
```

**Description** Return the n-th element of a list. The head of the list is the 0-th element.

**Errors**

| 1009 | Index past ends of list |

SML

```sml
val overwrite : ("a * 'b) list * ("a * 'b) -> ("a * 'b) list;
```

**Description** alist overwrite (a, b) gives the list in which the first pair in alist that has the left item a 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 tail of alist.

**See Also** roverwrite, list_overwrite

SML

```sml
val present : ('a * 'a) -> bool) -> 'a -> 'a list -> bool;
```

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

**See Also** contains, mem

SML

```sml
val rassoc1 : ("a * "a) list -> "a -> "a;
```

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

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

SML

```sml
val rassoc2 : ('a * "b) list -> ("b -> 'a) -> "b -> 'a;
```

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

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

SML

```sml
val rassoc3 : ('a * "b) list -> "b -> 'a;
```

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

**Errors**

| 1005 | No such value in association list |

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

SML

```sml
val rassoc4 : ('a * "b) list -> 'a -> "b -> 'a;
```

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

**See Also** lassocN and rassocN, where N = 1 ... 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 -> '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

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

**Description** `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 `union eq list1 list2` is the list obtained by prepending those elements of `list1` not already present in `list2`, to the list `list2`. Presence for `x` in the list being created being that there is a member, `y`, of the list being created with `eq(x, y) = true`. If `list1` contains duplicates then all but the rightmost will be eliminated, but those in `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**

```sml
union eq (list1 @ [a]) list2 = union eq list1 (    if present eq a list2    then list2    else (a :: list2)) | union eq [] list2 = list2
```

**See Also** `cup, list_union`

---

**Synopsis**

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

**Definition**

```sml
val which : (('a * 'a) -> bool) -> 'a list -> int OPT;
```

**See Also** `combine`

**Errors**

1008 List lengths differ

---

**Synopsis**

**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** `combine`

**Errors**

1008 List lengths differ

---

**Synopsis**

**Description** `l1 <= l2` is true iff. every member of `l1` is also a member of `l2`. `l1 = l2` is true iff. the set of members of `l1` is equal to the set of members of `l2`.

**See Also** `subset`
2.4 Functions

**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 un curry

**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**

\[(f \text{ fun_and } g) x = f x \text{ andalso } g x\]
\[(f \text{ fun_or } g) x = f x \text{ orelse } g x\]
\[(\text{fun_not } f) x = \text{not}(f x)\]
\[\text{fun_true } x = \text{true}\]
\[\text{fun_false } x = \text{false}\]

**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

\[\lambda x\bullet (f(f(...(f(x))...))\]

where f appears n times.

**Errors**

\[1010\] First argument must not be negative
2.4. Functions

val repeat : (unit -> 'a) -> unit;
val iterate : ('a -> 'a) -> 'a -> 'a;

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.

fun repeat f = (f (); repeat f) handle (Fail _) => ()
fun iterate f a = (iterate f (f a)) handle (Fail _) => a

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

val swap : 'a * 'b -> 'b * 'a;
Description  swap interchanges the elements of a pair: swap(a, b) = (b, a).

val switch : ('a -> 'b -> 'c) -> 'b -> 'a -> 'c;
Description  switch f a b gives f b a.

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

[SML]

```
signature Combinators = sig

val I : 'a -> 'a

val K : 'a -> 'b -> 'a

val S : ('a -> 'b -> 'c) -> ('a -> 'b) -> 'a -> 'c
```

**Description**
- Holds the three combinators $S$, $K$, $I$.
- The identity combinator: $I \, x = x$.
- The deletion combinator: $K \, x \, y = x$.
- The duplication combinator: $S \, f \, g \, a = (f \, a)(g \, a)$. 
2.6 Characters

SML

|signature CharacterUtilities = sig |

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

SML

|val is_all_decimal : string -> bool; |

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

SML

|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"

SML

|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**

```ml
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**

```ml
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**

```ml
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**

```ml
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**

```ml
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**

```ml
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**

```ml
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 sExtend error, but will be replicated in the result if found in the second list.
```
2.8 Efficient Dictionary

SML
signature EfficientDictionary = 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

|type 'a E_DICT;

Description The type of efficient dictionaries.

SML

|type E_KEY;

val e_get_key : string -> E_KEY;

val e_key_lookup : E_KEY -> 'a E_DICT -> 'a OPT

val e_key_enter : E_KEY -> 'a -> 'a E_DICT -> 'a E_DICT;

val e_key_extend : E_KEY -> 'a -> 'a E_DICT -> 'a E_DICT;

val e_key_delete : E_KEY -> 'a E_DICT -> 'a E_DICT;

val string_of_e_key : E_KEY -> string;

Description The abstract data type E_KEY represents the hash-keys used in the internals of the efficient dictionary (E_DICT) access functions. e_get_key computes the hash-key for a given string. This may then be used as an argument to the functions e_key_lookup, e_key_enter, e_key_extend and 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. string_of_key is the left inverse of 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

|val e_delete : string -> 'a E_DICT -> 'a E_DICT;

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

SML

|val e_enter : string -> 'a -> 'a E_DICT -> 'a E_DICT;

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

SML

|val e_extend : string -> 'a -> 'a E_DICT -> 'a E_DICT;

Description e_extend implements extension by a singleton function, that is to say it is like e_enter. e_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.

Errors
1014 ?0 is already in dictionary
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.

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.

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.

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).

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.

val list_e_enter : 'a E.DICT -> 'a S.DICT -> 'a E.DICT;
Description    list_e_merge 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.

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 Order-preserving Efficient Dictionary

SML

```sml
type 'a OE.DICT;
val initial_oe_dict : 'a OE.DICT;
val oe_lookup : string -> 'a OE.DICT -> 'a OPT
val oe_enter : string -> 'a -> '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 -> 'a OE.DICT -> 'a OE.DICT;
val oe_key_extend : E.KEY -> 'a -> '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.
Chapter 2. PROGRAMMING UTILITIES

2.10 Orderings

**SML**

\[
\text{type } \texttt{′a ORDER} = \texttt{′a -> ′a -> int};
\]

**Description** This is a polymorphic type used by several functions (e.g., sort) to represent various kinds of ordering, e.g., linear orders or quasi-orders.

If \( f \) is an ordering in this sense, then elements \( x \) and \( y \) are taken to be in order (\( x \prec y \)), equivalent (\( x \simeq y \)) or out of order (\( x \succ y \)) according as \( f x y \) is negative, zero or positive. So, for example, the usual ordering of the integers corresponds to \( f x y = x - y \).

**SML**

\[
\text{val ascii_order : string ORDER;}
\]

**Description** This function gives the ordering of character string by numeric character code. The integer result is appropriate for use in sort (q.v.).

**See Also** strin_order which is a slightly less efficient but order alphabetic characters in a more intuitive way.

**SML**

\[
\text{val induced_order : (′a -> ′b) * ′b ORDER -> ′a ORDER;}
\]

**Synopsis** Given an ordering \( r \) on elements of type \( ′b \) and a function \( f \) of type \( ′a -> ′b \), \( \text{induced_order}(f, r) \) is the ordering of the type \( ′a \) induced by \( f \) and \( r \), i.e., is the ordering obtained by mapping elements of type \( ′a \) to the type \( ′b \) using \( f \) and then comparing the results with \( r \).

**SML**

\[
\text{val int_order : int ORDER;}
\]

**Description** This function gives the usual ordering on the standard ML integers.

**SML**

\[
\text{val lexicographic : ′a ORDER -> ′a list ORDER;}
\]

**Synopsis** This function returns the lexicographic ordering function induced from an ordering of items of type \( ′a \) on items of type \( ′a \) list.

**Description** Lexicographic ordering may be defined as follows:

Let us presume some ordering \( < \) on the type of variables \( a_i \) and \( b_i \). Then

\[
[a_1, \ldots, a_n] < _\text{lexicographic} [b_1, \ldots, b_m]
\]

If, and only if, either

\[
n < _\text{int} m \land \forall i : 1..n \bullet a_i = b_i
\]

Or

\[
\exists j : 1..n \bullet (\forall i : 1..(j - 1) \bullet a_i = b_i) \land (a_j < b_j)
\]

The resulting ordering is appropriate for use in sort (q.v.).

list_order is a synonym for lexicographic.

**See Also** string_order, pair_order, sort, list_order.

**SML**

\[
\text{val no_order : ′a ORDER;}
\]

**Description** This function is the ordering which makes any two elements compare equivalent.
Synopsis

Several functions such as `sort` are parameterised by ordering functions, \( f \), say, such that, \( f \ x \ y \) is negative, zero or positive result according as \( x < y \), \( x = y \) or \( x > y \) with respect to some total ordering. The above functions help to construct new ordering functions from old.

Given orderings on the component types of a binary product type, `pair_order` gives the ordering on the product type that orders first on the left component and then on the right to break ties.

Given an ordering on the element type of a list type, `list_order` gives the lexicographic ordering on the list type. It is a synonym for `lexicographic`, q.v.

`inv_list_order` is like `list_order` but gives the inverse lexicographic order (equivalent to reversing both operand lists but more efficiently implemented).

Given an ordering on any type, `opt_order` (resp. `opt_order1`) gives an ordering on the type of optional values of that type. `Nil` is taken as greater (resp. less) than any supplied value, `Value x`.

Given an ordering on any type, `rev_order` gives the corresponding reverse ordering.

See Also `string_order`, `lexicographic`, `sort`

Description

This function gives a more civilised ordering on strings than the one given by the ASCII codes. It collates upper-case and lower-case letters together in the (increasing) order ‘A’, ‘a’, . . . , ‘Z’, ‘z’ and all other characters after ‘z’ ordered by numeric character code. The integer result is appropriate for use in `sort` (q.v.).

See Also `lexicographic`, `ascii_order`.

Synopsis `THEN_O` is an infix operator of precedence 4. Given orderings \( r \) and \( s \) on elements of type ‘\( a \)’, \( r \ THEN_O \ s \) is the ordering that first compares elements using \( r \) and accepts the comparison unless the elements compare equivalent, in which case they are compared using \( s \). This is equivalent to `induced_order(diag, pair_order \ r \ s)` where \( diag \ x = (x, x) \).
2.11 Sorting

SML

signature Sort = sig

include Order;

Description This provides an efficient sort utility package. For historical reasons it includes the structure Order.

SML

val sort : 'a ORDER -> 'a list -> 'a list
val merge : 'a ORDER -> 'a list -> '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, \( \text{ilist} \) in ascending order:

\[
\text{sort (curry (op \(-\))) ilist}
\]

or

\[
\text{sort int\_order ilist}
\]

See Also For convenient ways of constructing orderings, see, e.g. string\_order and list\_order.
2.12 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).
### 2.13 Dynamic Arrays

#### SML signature
```
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
```
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`.

#### Errors
- 1101 Cell with index ?0 is empty
- 1303 Index ?0 is out of range

#### 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 `~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).
2.14 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 labs : INTEGER -> INTEGER;
val @< : INTEGER * INTEGER -> bool;
val @> : INTEGER * INTEGER -> bool;
val @<= : INTEGER * INTEGER -> bool;
val @>= : INTEGER * INTEGER -> bool;
val integer_of_string : string -> INTEGER;
val @@ : 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

1. the divisor is zero
2. an empty string is not a valid decimal number
3. the string ‘?0’ is not a valid decimal number
4. the conversion would overflow
2.15 Compatibility with SML’90

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.
SML signature PPArray = sig
   exception Subscript
   type '_a array
   val arrayoflist: '_a list -> '_a array
   val array: int * '_a -> '_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).

SML 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).

SML 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

signature SystemControl = sig

Description This is the signature of the structure SystemControl.

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

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

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.
3.1. System Control

SML
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.

SML
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.

SML
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

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

SML
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) -> unit

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.

SML
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.
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

**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.

**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.

**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.

**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.

**val pp'set_banner : string OPT -> string;**

(* string control: system_banner; default — see below *)

(* string control: user_banner; default — "" *)

**Description** pp'set_banner (Value banner) will change the core part of the system banner to banner, returning the old value. pp'set_banner Nil just returns the current value. (The value is held in the string control system_banner and can also be changed using set_string_control or read get_string_control).

The messages below gives the banner, which has elements which may be changed by setting the string controls system_banner and user_banner. Message 36050 is printed first with system_banner as the insertion (?0) followed by message 36051 with insertions giving the latest copyright year (?0) and the user_banner (?1). If it is not empty, user_banner should begin with a newline character.

Message 36000 gives the value for system_banner set in the HOL database, the insertion being the version string taken from the variable system_version defined by the make file.

**Errors**

36000 ProofPower ?0 [HOL Database]
36050 === ?0
36051 === Copyright (C) Lemma 1 Ltd. 2000−?0?1

**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.
| SML | val pp\textunderscore theory\_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. |
### 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**  
- 36010 *The database name has not been set*  
- 36017 *STATE WAS FOUND TO BE INCONSISTENT: state should not be saved*  

**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

signature Warning = 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

val comment : string -> int -> (unit -> string) list -> 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

val warn : string -> int -> (unit -> string) list -> 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 warn 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 warn</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 warn</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
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**

```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**

```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 71) after '?2'
- 5006 Unknown extended character '?'0' (decimal 71)
- 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**

```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 ‘’ (i.e., code 81) and ‘%’ (37).
Chapter 4. INPUT AND OUTPUT

SML

(* RW_diagnostics – integer control declared by new_int_control, default: 0 *)

Description For reader writer diagnostic purposes.

SML

(* use_extended_chars – boolean flag declared by new_flag, default: true *)

Description 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.

SML

(* use_file_non_stop_mode – boolean flag declared by new_flag, default: false *)

Description 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.

SML

datatype NAME_CLASS

| = Simple
|   | Starting of (READER_ENV -> (string * bool))
|   |   -> string -> bool -> string list
|   |   -> string list) * string
|   | Middle of string
|   | Ending of string
|   | Ignore
|   | Invalid;

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
  | SymEndOfFile;

Description   SymKnown indicates a symbol declared via add_new_symbols, if a keyword was read the string hold 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.

SymEndOfFile 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.
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
```
(["fn", "lambda"], Value "\lambda", Simple),
```

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.

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

```
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 “␣” denotes aspace 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**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5033</td>
<td>Reader already present for language ‘?0’</td>
</tr>
<tr>
<td>5034</td>
<td>Improper reader name ‘?0’</td>
</tr>
<tr>
<td>5035</td>
<td>Improper reader name ‘?0’ and ‘?1’</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>SML</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>val find_name : string -&gt; PRETTY_NAME OPT</code></td>
</tr>
<tr>
<td><code>val find_char : string -&gt; PRETTY_NAME OPT</code></td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>SML</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>val general_quotation : READER_ENV</code></td>
</tr>
<tr>
<td><code>-&gt; (string * bool) (* Start of quotation symbol *)</code></td>
</tr>
<tr>
<td><code>-&gt; string (* Opening characters *)</code></td>
</tr>
<tr>
<td><code>-&gt; bool (* Context, true =&gt; in Standard ML *)</code></td>
</tr>
<tr>
<td><code>-&gt; string list (* Left hand context *)</code></td>
</tr>
<tr>
<td><code>-&gt; string list;</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>val specific_quotation : READER_ENV</code></td>
</tr>
<tr>
<td><code>-&gt; (string * bool) (* Start of quotation symbol *)</code></td>
</tr>
<tr>
<td><code>-&gt; string (* Opening characters *)</code></td>
</tr>
<tr>
<td><code>-&gt; bool (* Context, true =&gt; in Standard ML *)</code></td>
</tr>
<tr>
<td><code>-&gt; string list (* Left hand context *)</code></td>
</tr>
<tr>
<td><code>-&gt; string list;</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>val named_quotation : READER_ENV</code></td>
</tr>
<tr>
<td><code>-&gt; (string * bool) (* Start of quotation symbol *)</code></td>
</tr>
<tr>
<td><code>-&gt; string (* Opening characters *)</code></td>
</tr>
<tr>
<td><code>-&gt; bool (* Context, true =&gt; in Standard ML *)</code></td>
</tr>
<tr>
<td><code>-&gt; string list (* Left hand context *)</code></td>
</tr>
<tr>
<td><code>-&gt; string list;</code></td>
</tr>
</tbody>
</table>

**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 |
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’

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`.

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 * PrettyNamesPRETTY_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.
Chapter 4. INPUT AND OUTPUT

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.

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.).

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

**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.

**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).

**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.

**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%context% 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’

**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

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029


**Chapter 4. INPUT AND OUTPUT**

**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

SML

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.

SML

(* 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

SML

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 concatenation 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 ["\(\mathfrak{f}\),\(\mathfrak{x}\),\(\mathfrak{a} \land \mathfrak{b}\)] "", "

val it = "\(\mathfrak{f}\), \(\mathfrak{x}\), \(\mathfrak{a} \land \mathfrak{b}\)" : string

SML

val get_line_length : unit -> int

Description  Returns current output line length.

See Also  set_line_length

SML

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.

SML

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
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
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

**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.

---

**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

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

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 |

| (* Flag pp_print_top_level_depth : integer control, default -1 *) |
| (* Flag pp_print_format_depth : integer control, default -1 *) |

Description These control the depth to which HOL types and terms are printed. Control pp_print_top_level_depth applies to values printed as part of Standard ML top-level expressions. Control pp_print_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**

```
signature Lister = sig
```

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

**SML**

```
signature ListerSupport = sig
datatype LISTER_SECTION =
| LSBanner   | LSParents  | LSChildren |
| LSCOnsts   | LSAliases  | LSUndeclaredAliases |
| LSTypes    | LSTypeAbbrevs | LSUndeclaredTypeAbbrevs |
| LSFixity   | LSTerminators | LSUndeclaredTerminators |
| LSXioms    | LSDefs     | LSThms     |
LSTrailer
| LSADString of string -> (string list * string) |
| LSADStrings of string -> (string list * string list) |
| LSADThms of string -> (string list * THM) 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_list* and *gen_theory_list1* 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**

```
(* 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 '70' must be greater than zero
```

**See Also**  *output_theory*
val gen_theory_list : LISTER_SECTION list \rightarrow
  
  \{
    print: (string \rightarrow string) \rightarrow string \rightarrow unit,
    out: (string \rightarrow string) \rightarrow \{theory: string, out_file: string\} \rightarrow unit,
    out1: (string \rightarrow string) \rightarrow \{theory: string, out_file: string\} \rightarrow unit\};

val gen_theory_list1 : LISTER_SECTION list \rightarrow
  
  \{
    print: (string \rightarrow string) \rightarrow string \rightarrow unit,
    out: (string \rightarrow string) \rightarrow \{theory: string, out_file: string\} \rightarrow unit,
    out1: (string \rightarrow string) \rightarrow \{theory: string, out_file: string\} \rightarrow unit\};

end (* of structure ListerSupport *) (* of structure ListerSupport *);

Description  The functions ListerSupport.gen_theory_list and
ListerSupport.gen_theory_list1 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_list. This argument is a list of elements of
type LISTER_SECTION, q.v.

The integer control listing_indent and 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_list1 is just like gen_theory_list except that it does not check whether the theory
exists or whether it is in scope.

val output_theory1 : \{theory:string, out_file:string\} \rightarrow 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 (?)
33102  the theory ?0 does not exist
4.5. Theory Lister

SML

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**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>33050</td>
<td>The theory <code>?0</code> is not in scope</td>
</tr>
<tr>
<td>33051</td>
<td>There is no theory called <code>?0</code></td>
</tr>
<tr>
<td>33101</td>
<td>i/o failure on file <code>?0</code> (?1)</td>
</tr>
<tr>
<td>33102</td>
<td>the theory <code>?0</code> does not exist</td>
</tr>
</tbody>
</table>

SML

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**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>33050</td>
<td>The theory <code>?0</code> is not in scope</td>
</tr>
<tr>
<td>33051</td>
<td>There is no theory called <code>?0</code></td>
</tr>
</tbody>
</table>

**See Also**  
`output_theory output_theory1`
Chapter 5

HOL TYPES AND TERMS

5.1 Syntactic Manipulations

It should be noted that the functions documented in this section are drawn from two signatures, TypesAndTerms and pp' 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 pp' TypesAndTerms.xxx. It is the intention that users should not access objects by names of the form pp' 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 =
  | Vartype of string
  | Ctype 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
  | Simpleλ 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.
SML
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 |
De 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

SML
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.

SML
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.).

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

**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
\[
\\text{val bin_bool_op : string} \rightarrow \text{TYPE} \rightarrow \text{TYPE} \rightarrow \text{TERM};
\]

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

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

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

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

**See Also** Theory “min”.

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

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

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

**See Also** Theory “char”.

SML
\[
\text{val dest_app} : \text{TERM} \rightarrow (\text{TERM} \times \text{TERM});
\]

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

**Definition**
\[
\begin{align*}
\text{dest_app} \gamma f t &= (\gamma f, \gamma t) \\
\text{dest_app} \gamma \forall x \cdot t &= (\gamma \forall x \cdot \gamma t)
\end{align*}
\]

**Errors**
\[
3010 \ ?0 \text{ is not of form: } \gamma t1 t2
\]

SML
\[
\text{val dest_binder} : \text{string} \rightarrow \text{int} \rightarrow \text{string} \rightarrow \text{TERM} \rightarrow \text{TERM} \times \text{TERM};
\]

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

**Definition**
\[
\begin{align*}
\text{dest_binder area msg binder_nms} \gamma \text{binder}(\lambda \text{varstruct} \cdot \text{body}) &= (\gamma \text{varstruct}, \gamma \text{body})
\end{align*}
\]

where *binder* is a constant whose name is *binder_nms*. The *varstruct* may be any allowed variable structure.

**See Also** dest_simple_binder

**Failure** If the term cannot be destroyed, then the error will be from *area*, with a message indexed by *msg*.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
\begin{verbatim}
val dest_bin_op : string -> int -> string -> TERM -> (TERM * TERM);

Description dest_bin_op area msg rator_nm term first assumes that term is of the form \((\text{rator } t_1 \ t_2)\), where \text{rator} is a constant with name \text{rator_nm}, and attempts to return the pair \((t_1, t_2)\).

Example

\begin{verbatim}
dest_bin_op "dest\_\&" 4032 "\&" \(\triangleright a \ \& b \triangleright = (\triangleright a \triangleright, \triangleright b)\)
\end{verbatim}

If the function fails it will fail with message msg, area area and with the string form of term.
\end{verbatim}

val dest_char : TERM -> string;

Description Destroy a character literal.

Example

\begin{verbatim}
dest_char \(\triangleright 'a'\) = "a"
\end{verbatim}

Errors

\begin{verbatim}
3024 ?0 is not a character literal
\end{verbatim}

val dest_const : TERM -> (string * TYPE);

Description This destroys a constant into its name and type.

Errors

\begin{verbatim}
3009 ?0 is not a constant
\end{verbatim}

val dest CType : TYPE -> string * TYPE list;

Description Extract the components of a compound type.

Definition

\begin{verbatim}
dest CType \(\triangleright (ty_1,ty_2,\ldots)tc\triangleright = (\triangleright tc\triangleright, [\triangleright ty_1\triangleright\triangleright,\triangleright ty_2\triangleright,\ldots])
dest CType \(\triangleright ty\ ) tc\triangleright = (\triangleright tc\triangleright, [\triangleright ty\triangleright])
dest CType \(\triangleright tc\ ) tc\triangleright = (\triangleright tc\triangleright, [])
\end{verbatim}

Errors

\begin{verbatim}
3001 ?0 is not a compound type
\end{verbatim}

val dest_Empty_list : TERM -> TYPE;

Description A derived term destructor function for empty lists.

Definition

\begin{verbatim}
dest Empty_list \(\triangleright \emptyset ty\ ) LIST\triangleright = \triangleright ty\triangleright
\end{verbatim}

Errors

\begin{verbatim}
4034 ?0 is not of form: \(\triangleright \emptyset \triangleright\)
\end{verbatim}

val dest_Enum_set : TERM -> (TERM list);

Description A derived term destructor function for enumerated sets.

Definition

\begin{verbatim}
dest Enum_set \(\triangleright \{ a; b; \ldots\}\triangleright = [\triangleright a\triangleright\triangleright, \triangleright b\triangleright, \ldots]
\end{verbatim}

Errors

\begin{verbatim}
4011 ?0 is not of form: \(\triangleright \{ t_1, \ldots\} \triangleright\)
\end{verbatim}

5.1. Syntactic Manipulations

SML

\[ \text{val dest_eq : TERM \rightarrow (TERM \times TERM)}; \]

**Description**  A derived term destructor function for equations.

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

**Errors**  
3014  ?0 is not of form: \( \Gamma t = u \gamma \)

SML

\[ \text{val dest_float : TERM \rightarrow INTEGER \times INTEGER \times INTEGER}; \]

**Description**  Destroy a floating point literal.

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

where \( x \) is the natural number with decimal representation \( XXYY \), \( p \) 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

\[ \text{val dest_f : TERM \rightarrow unit}; \]

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

**Errors**  
4037  ?0 is not: \( \Gamma F \gamma \)

SML

\[ \text{val dest_if : TERM \rightarrow (TERM \times TERM \times TERM)}; \]

**Description**  Destroy a conditional.

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

**Errors**  
4006  ?0 is not of form: \( \Gamma if \ c \ then \ y \ else \ n \gamma \)
Chapter 5. HOL TYPES AND TERMS

SML

val dest_list : TERM -> (TERM list * TERM);

Description A derived term destructor function for list-terms. See mk_list 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.

Example

dest_list (mk_list([]), term)) will actually fail (unless term is already a list-term), as apply mk_list to ([], term)) will just return term.

SML

val dest_mon_op : string -> int -> string -> TERM -> TERM;

Description dest_mon_op area msg rator_nm term assumes that term is of the form "rator t", where rator is a constant with name rator_nm, and the function attempts to return t.

Example

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

SML

val dest_multi_¬ : TERM -> (int * TERM);

Description dest_multi_¬ t will strip ¬ from t, returning the number of times, as well as the result. It will return (0, t) if t is either not boolean, or has no negations.

Example

dest_multi_¬ (¬(¬ T)) = (2, ¬¬)

SML

val dest_pair : TERM -> (TERM * TERM);

Description A derived term destructor function for pairs.

Example

dest_pair (t1, t2) = (¬¬t1, ¬¬t2)
5.1. Syntactic Manipulations

SML
\textbf{val} dest_set_comp : TERM \rightarrow (TERM * TERM);

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

Example
\[\text{dest_set_comp} \mapsto \{ x \mid x > 5 \} = (\forall x, x > 5)\]

**Errors**
4013 ?0 is not of form: \{v | p\}

SML
\textbf{val} dest_simple_binder : string \rightarrow int \rightarrow string \rightarrow TERM \rightarrow TERM * TERM;

**Description** Executing dest_simple_binder area msg binder_nm \mapsto binder(\lambda \text{var} \bullet \text{body})\), where binder is a constant with the name binder_nm, will give (\forall \text{var}, \forall \text{body}).

Example
\[\text{dest_simple_binder} "\text{dest_simple}_\forall" 3032 "\forall" \forall x \bullet t = (\forall x, \forall t)\]

**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
\textbf{val} dest_simple_term : TERM \rightarrow 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
\textbf{val} dest_simple_type : TYPE \rightarrow DEST_SIMPLE_TYPE;

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

SML
\textbf{val} dest_simple_\forall : TERM \rightarrow (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
\[\text{dest_simple}_\forall \forall \text{var} \bullet \text{body} = (\forall \text{var}, \forall \text{body})\]

**See Also** dest_\forall

**Errors**
3032 ?0 is not of form: \forall \text{var} \bullet \text{body}

SML
\textbf{val} dest_simple_\exists_1 : TERM \rightarrow (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
\[\text{dest_simple}_\exists_1 \exists_1 \text{var} \bullet \text{body} = (\exists_1 \text{var}, \exists_1 \text{body})\]

**Errors**
4019 ?0 is not of form: \exists_1 \text{var} \bullet t

**See Also** dest_\exists_1

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
Chapter 5. HOL TYPES AND TERMS

SML
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⌝

SML
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⌝

SML
val dest_string : TERM -> string;

Description Destroy a string literal.

Example
dest_string ⌜"abc"⌝ = "abc"

Errors
3025 ?0 is not a string literal

SML
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

SML
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>Definition</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>val dest_vartype : TYPE -&gt; string;</code></td>
<td>Extract the name of a type variable.</td>
<td><code>3019</code> <code>?0</code> is not a type variable, <code>3027</code> STRING STORE ERROR: cannot translate internal id (?0) to string</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>Definition</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>val dest_var : TERM -&gt; (string * TYPE);</code></td>
<td>This destroys a term variable into its name and type.</td>
<td><code>3007</code> <code>?0</code> is not a term variable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>Definition</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>val dest_∅ : TERM -&gt; TYPE;</code></td>
<td>A derived term destructor function for empty enumerated sets.</td>
<td><code>4035</code> <code>?0</code> is not of form: <code>⌜∅⌝</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>Definition</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>val dest_⇔ : TERM -&gt; (TERM * TERM);</code></td>
<td>A derived term destructor function for bi-implications. N.B. this may be successfully applied to boolean equalities.</td>
<td><code>4031</code> <code>?0</code> is not of form: <code>⌜t1 ⇔ t2⌝</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>Definition</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>val dest→type : TYPE -&gt; (TYPE * TYPE);</code></td>
<td>Extract the two constituent types of a function type.</td>
<td><code>3022</code> <code>?0</code> is not of form: <code>⌜ty1 → ty2⌝</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>Definition</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>val dest∧ : TERM -&gt; (TERM * TERM);</code></td>
<td>A derived term destructor function for conjunctions.</td>
<td><code>4032</code> <code>?0</code> is not of form: <code>⌜t1 ∧ t2⌝</code></td>
</tr>
</tbody>
</table>
Chapter 5. HOL TYPES AND TERMS

val dest_\lor : TERM \to (TERM \times TERM);

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

\[
\text{dest}_\lor \triangleright t_1 \lor t_2 = (\triangleright t_1, \triangleright t_2)
\]

\textbf{Errors}

\[4027 \text{ ?0 is not of form: } \triangleright t_1 \lor t_2\]

val dest_\lnot : TERM \to TERM;

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

\[
\text{dest}_\lnot \triangleright \lnot t = \triangleright t
\]

\textbf{Errors}

\[4029 \text{ ?0 is not of form: } \lnot t\]

val dest_\Rightarrow : TERM \to (TERM \times TERM);

\textbf{Description}  A derived term destructor function for implications, returning the antecedent and consequent.

\[
\text{dest}_\Rightarrow \triangleright a \Rightarrow b = (\triangleright a, \triangleright b)
\]

\textbf{Errors}

\[3016 \text{ ?0 is not of form: } \triangleright t \Rightarrow u\]

val dest_\forall : TERM \to (TERM \times TERM);

\textbf{Description}  A derived term destructor function for \forall-terms. It may destroy a paired abstraction \forall-term, being the inverse of mk_\forall.

\[
\text{dest}_\forall \triangleright \forall \text{ varstruct} \cdot \text{body} = (\triangleright \text{ varstruct}, \triangleright \text{body})
\]

\textbf{Errors}

\[4017 \text{ ?0 is not of form: } \forall \text{ vs} \cdot t\]

val dest_\exists_1 : TERM \to (TERM \times TERM);

\textbf{Description}  A derived term destructor function for \exists_1-terms. It may destroy paired abstraction \exists_1-terms, being the inverse of mk_\exists_1.

\[
\text{dest}_\exists_1 \triangleright \exists_1 \text{ varstruct} \cdot \text{body} = (\triangleright \text{ varstruct}, \triangleright \text{body})
\]

\textbf{Errors}

\[4021 \text{ ?0 is not of form: } \exists_1 \text{ vs} \cdot t\]

\textbf{See Also}  dest_simple_\exists_1
5.1. Syntactic Manipulations

SML
val dest_∃ : TERM -> (TERM * TERM);

Description A derived term destructor function for ∃-terms. It may destroy paired abstraction
∃-terms, being the inverse of mk_∃.

Definition
dest_∃⌜∃ varstruct• body⌝ = (⌜varstruct⌝,⌜body⌝)

Errors
4020  ?0 is not of form: ∃ vs• t

See Also dest_simple_∃

SML
val dest×_type : TYPE -> (TYPE * TYPE)

Description dest×_type⌜ty_1 ty_2⌝ returns (⌜ty_1⌝,⌜ty_2⌝).

Errors
4018  ?0 is not of the form: : ty1 × ty2

SML
val dest_ε : TERM -> (TERM * TERM);

Description A derived term destructor function for ε-terms.

Definition
dest_ε⌜ε varstruct• body⌝ = (⌜varstruct⌝,⌜body⌝)

Errors
4023  ?0 is not of form: ε vs• t

SML
val dest_λ : TERM -> (TERM * TERM);

Description Destroys a λ-abstraction. It can destroy paired λ-abstractions, being an inverse
of mk_λ.

Definition
dest_λ⌜λ varstruct• body⌝ = (⌜varstruct⌝,⌜body⌝)

See Also dest_simple_λ

Errors
4002  ?0 is not of form: λ vs• t

Further details of the errors will be given, before the above exceptions are raised.

SML
val dest_N : TERM -> INTEGER;

Description Destroy a numeric literal.

Example
dest_N⌜5⌝ = 5;

Errors
3026  ?0 is not a numeric literal

SML
val equality : TYPE -> TYPE -> TERM;

Description Returns the constant⌜$ =⌝ 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.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
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

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.

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.).

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

val inst: TERM list -> (TYPE * TYPE) list -> TERM -> TERM;

Description inst avlist sl1st term instantiates the type variables of term with the associated types found in sl1st. An element of sl1st 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)

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

\[ \textbf{val is\_binder} : \text{string} \to \text{TERM} \to \text{bool}; \]

\textbf{Description} \ is\_binder binder\_nm tm is true only when tm is of the form \( \rhd \text{binder}(\lambda \text{vs} \cdot \text{body}) \rhd \), where binder is a constant whose name is binder\_nm, and vs an allowed variable structure, and false otherwise. It cannot raise an exception.

\textbf{See Also} is\_simple\_binder

SML

\[ \textbf{val is\_bin\_op} : \text{string} \to \text{TERM} \to \text{bool}; \]

\textbf{Description} \ is\_bin\_op rator\_nm term returns true iff. term is of the form \( \rhd \text{rator t1 t2} \rhd \), and rator is a constant with name rator\_nm. It cannot raise an exception.

\textbf{Example} \ \textbf{is\_bin\_op} "\&" \( \rhd a \ \& \ b \rhd \) = true

SML

\[ \textbf{val is\_char} : \text{TERM} \to \text{bool}; \]

\textbf{Description} \ Return true only when the term is a character literal (e.g. \( \rhd \text{\textquotesingle}a\text{\textquotesingle} \rhd \)), and false otherwise: no exceptions can be raised.

SML

\[ \textbf{val is\_const} : \text{TERM} \to \text{bool}; \]

\textbf{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

\[ \textbf{val is\_ctype} : \text{TYPE} \to \text{bool}; \]

\textbf{Description} \ Return true only when the type 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

\[ \textbf{val is\_empty\_list} : \text{TERM} \to \text{bool}; \]

\textbf{Description} \ Return true only when the term is an empty list-term, \( \rhd [] \rhd \), and false otherwise: no exceptions can be raised.

SML

\[ \textbf{val is\_enum\_set} : \text{TERM} \to \text{bool}; \]

\textbf{Description} \ Return true only when the term is an enumerated set (i.e. of form \( \rhd \{a; b; \ldots\} \rhd \)), and false otherwise: no exceptions can be raised.

SML

\[ \textbf{val is\_eq} : \text{TERM} \to \text{bool}; \]

\textbf{Description} \ Return true only when the term is an equation (i.e. of form \( \rhd a = b \rhd \) or \( \rhd a \IFF b \rhd \)), and false otherwise: no exceptions can be raised.

SML

\[ \textbf{val is\_float} : \text{TERM} \to \text{bool}; \]

\textbf{Description} \ Return true when the term is a floating point literal. and false otherwise: no exceptions are raised.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
SML
\texttt{val is\_free\_in : TERM \rightarrow TERM \rightarrow bool;}

**Description** is\_free\_in \textit{v} term returns true iff. there is a free occurrence of \textit{v} in term. It will raise an exception if the first argument is not a term variable.

**Errors**
\texttt{3007 \ ?0 is not a term variable}

SML
\texttt{val is\_free\_var\_in : (string \ast TYPE) \rightarrow TERM \rightarrow bool;}

**Description** 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.

SML
\texttt{val is\_f : TERM \rightarrow bool;}

**Description** Return true only when the term is \textit{⌜F: BOOL⌝}, and false otherwise: no exceptions can be raised.

SML
\texttt{val is\_if : TERM \rightarrow bool;}

**Description** Return true only when the term is a conditional (i.e. of form \textit{⌜if a then b else c⌝}), and false otherwise: no exceptions can be raised.

SML
\texttt{val is\_let : TERM \rightarrow bool;}

**Description** Return true only when the term is a \textit{let}-term (i.e. of form \textit{⌜let x = y in z⌝}), and false otherwise: no exceptions can be raised.

SML
\texttt{val is\_list : TERM \rightarrow bool;}

**Description** Return true only when the term is a list-term (i.e. of form \textit{⌜[a; b; ...]⌝}), and false otherwise: no exceptions can be raised.

SML
\texttt{val is\_mon\_op : string \rightarrow TERM \rightarrow bool;}

**Description** is\_mon\_op \textit{rator\_nm} term returns true iff. term is of the form \textit{rator t}, where \textit{rator} is a constant with name \textit{rator\_nm}. It cannot raise an exception.

**Example**
\texttt{is\_mon\_op "\neg" \textit{⌜\neg t⌝} = \textit{⌜t⌝}}

SML
\texttt{val is\_pair : TERM \rightarrow bool;}

**Description** Return true only when the term is a pair (i.e. of the form \textit{⌜(a, b)⌝}), and false otherwise: no exceptions can be raised.

SML
\texttt{val is\_set\_comp : TERM \rightarrow bool;}

**Description** Return true only when the term is a set comprehension (i.e. of form \textit{⌜\{v \mid p\}⌝}), and false otherwise: no exceptions can be raised.
5.1. Syntactic Manipulations

SML

\textbf{val} \textit{is\_simple\_binder} : \textit{string} $\rightarrow$ \textit{TERM} $\rightarrow$ bool;

\textbf{Description} \textit{is\_simple\_binder binder\_nm term} returns true iff. argument term is of the form $\text{⌜} binder(\lambda \textit{var} \bullet \textit{body})\text{⌝}$, where \textit{binder} is a constant with the name \textit{binder\_nm}.

\textbf{See Also} \textit{is\_binder}

\begin{verbatim}
/val is\_simple\_\forall : TERM $\rightarrow$ bool;
\end{verbatim}

\textbf{Description} A derived term test for simple $\forall$-terms (i.e. of form $\text{⌜} \forall \textit{x} \bullet \textit{t} \text{⌝}$), not formed with paired abstractions.

\textbf{See Also} \textit{is\_\forall}

\begin{verbatim}
/val is\_simple\_\exists\_1 : TERM $\rightarrow$ bool;
\end{verbatim}

\textbf{Description} Return true only when the term is a $\exists\_1$-term (i.e. of form $\text{⌜} \exists\_1 \textit{x} \bullet \textit{t} \text{⌝}$), formed only by simple abstraction, and false otherwise: no exceptions can be raised.

\textbf{See Also} \textit{is\_\exists\_1}

\begin{verbatim}
/val is\_simple\_\exists : TERM $\rightarrow$ bool;
\end{verbatim}

\textbf{Description} A derived term test for $\exists$-terms (i.e. of form $\text{⌜} \exists \textit{x} \bullet \textit{t} \text{⌝}$), not formed with paired abstractions.

\textbf{See Also} \textit{is\_\exists}

\begin{verbatim}
/val is\_simple\_\lambda : TERM $\rightarrow$ bool;
\end{verbatim}

\textbf{Description} Is the term a simple $\lambda$-abstraction (i.e. of form $\text{⌜} \lambda \textit{x} \bullet \textit{t} \text{⌝}$).

\textbf{See Also} \textit{is\_\lambda}

\begin{verbatim}
/val is\_string : TERM $\rightarrow$ bool;
\end{verbatim}

\textbf{Description} Return true only when the term is a string literal (e.g. \text{⌜}"abc\text{⌝"}), and false otherwise: no exceptions can be raised.

\begin{verbatim}
/val is\_type\_instance : TYPE $\rightarrow$ TYPE $\rightarrow$ bool;
\end{verbatim}

\textbf{Description} \textit{is\_type\_instance ty\_1 ty\_2} returns true iff \textit{ty\_1} is an instance of \textit{ty\_2}. It cannot raise an exception.

\begin{verbatim}
/val is\_t : TERM $\rightarrow$ bool;
\end{verbatim}

\textbf{Description} Return true only when the term is $\text{⌜} \textit{T : BOOL} \text{⌝}$, and false otherwise: no exceptions can be raised.

\begin{verbatim}
/val is\_vartype : TYPE $\rightarrow$ bool;
\end{verbatim}

\textbf{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.
| SML          | val is_var : TERM -> bool;                                      |
| Description | Return true only when the term is a variable, and false otherwise: no exceptions can be raised. |
| SML          | val is_empty : TERM -> bool;                                    |
| Description | Return true only when the term is an empty enumerated set, \(\emptyset\), and false otherwise: no exceptions can be raised. |
| SML          | val is_biconditional : TERM -> bool;                           |
| Description | Return true only when the term is a bi-implication (i.e. of form \(\forall a \leftrightarrow b\)), and false otherwise: no exceptions can be raised. N.B. this may be successfully applied to boolean equations. |
| SML          | val is_function_type : TYPE -> bool;                           |
| Description | Return true only when the type is a function type, i.e. of form \(\forall ty1 \rightarrow ty2\), and false otherwise: no exceptions can be raised. |
| SML          | val is_conjunction : TERM -> bool;                             |
| Description | Return true only when the term is a conjunction (i.e. of form \(\forall a \land b\)), and false otherwise: no exceptions can be raised. |
| SML          | val is_disjunction : TERM -> bool;                             |
| Description | Return true only when the term is a disjunction (i.e. of form \(\forall a \lor b\)), and false otherwise: no exceptions can be raised. |
| SML          | val is_negation : TERM -> bool;                                |
| Description | Return true only when the term is a negation (i.e. of form \(\forall \neg x\)), and false otherwise: no exceptions can be raised. |
| SML          | val is_implication : TERM -> bool;                             |
| Description | Return true only when the term is an implication (i.e. of form \(\forall a \Rightarrow b\)), and false otherwise: no exceptions can be raised. |
| SML          | val is_\forall : TERM -> bool;                                 |
| Description | Return true only when the term is a \(\forall\)-term (i.e. of form \(\forall vs \bullet t\)), possibly formed with paired abstraction, and false otherwise: no exceptions can be raised. |
| See Also    | is_simple_\forall |
| SML          | val is_\exists_1 : TERM -> bool;                               |
| Description | Return true only when the term is a \(\exists_1\)-term (i.e. of form \(\exists_1 vs \bullet t\)), possibly formed with paired abstraction, and false otherwise: no exceptions can be raised. |
| See Also    | is_simple_\exists_1 |
5.1. Syntactic Manipulations

SML

\[ \text{val is}_\exists : \text{TERM} \rightarrow \text{bool}; \]

**Description**  Return true only when the term is a $\exists$-term (i.e. of form $\exists vs \cdot t$), possibly formed with paired abstraction, and false otherwise: no exceptions can be raised.

**See Also**  \(\text{is}_\text{simple}_\exists\)

SML

\[ \text{val is}_\times\_\text{type} : \text{TYPE} \rightarrow \text{bool}; \]

**Description**  Return true only when the type is a pair type, i.e. of the form: $\forall : \text{ty}_1 \times \text{ty}_2 \forall$, and false otherwise: no exceptions can be raised.

SML

\[ \text{val is}_\epsilon : \text{TERM} \rightarrow \text{bool}; \]

**Description**  Return true only when the term is a $\epsilon$-term (i.e. of form $\epsilon vs \cdot t$), possibly formed with paired abstraction, and false otherwise: no exceptions can be raised.

SML

\[ \text{val is}_\lambda : \text{TERM} \rightarrow \text{bool}; \]

**Description**  This function returns true iff. the term is of the form $\lambda vs \cdot t$. It cannot raise exceptions.

**See Also**  \(\text{is}_\text{simple}_\lambda\)

SML

\[ \text{val is}_N : \text{TERM} \rightarrow \text{bool}; \]

**Description**  Return true only when the term is a numeric literal (e.g. $\forall 5 \forall$), and false otherwise: no exceptions can be raised.

SML

\[ \begin{align*}
\text{val \ key\_mk\_const} & : (E\_KEY * \text{TYPE}) \rightarrow \text{TERM}; \\
\text{val \ key\_dest\_const} & : \text{TERM} \rightarrow E\_KEY * \text{TYPE};
\end{align*} \]

**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

\[ \begin{align*}
\text{val \ key\_mk\_ctype} & : E\_KEY * \text{TYPE list} \rightarrow \text{TYPE}; \\
\text{val \ key\_dest\_ctype} & : \text{TYPE} \rightarrow E\_KEY * \text{TYPE list};
\end{align*} \]

**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

\[ \begin{align*}
\text{val \ list\_mk\_app} & : (\text{TERM} * \text{TERM list}) \rightarrow \text{TERM};
\end{align*} \]

**Description**  Applies a function to multiple arguments.

**Definition**  \[
\text{list\_mk\_app} (\forall t \forall, [\forall t1 \forall, \forall t2 \forall, \forall t3 \forall, ...]) = \forall t \ t1 \ t2 \ t3 \ ... \forall
\]

**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 ($\gamma vs$, $\gamma b$) makes an abstraction $\gamma bind vs \bullet b$, then

\[ list\_mk\_binder\ maker\ ([\gamma vs\_1, \gamma vs\_2, \ldots, \gamma body]) \]

returns $\gamma bind vs\_1 \bullet bind vs\_2 \bullet \ldots \bullet body$ 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

\[ list\_mk\_bin\_op\ area\ msg\ \Lambda\_fun\ \gamma [\gamma a, \gamma b \land c, \gamma d] = \gamma a \land ((\gamma b \land c) \land d) \]

where $\Lambda\_fun$ takes two (dummy) arguments and returns $\gamma$\$\Lambda$.

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

\[ list\_mk\_let\ ([[[\gamma x, \gamma 1]], [[\gamma y, \gamma 2]], \gamma x + y]] = \gamma let\ x = 1\ in\ let\ y = 2\ in\ x + y \]

val list_mk_simple_\lambda : (TERM list * TERM) -> TERM;

Description

$\lambda$-abstract a list of variables from a term.

Definition

\[ list\_mk\_simple\_\lambda\ ([\gamma x1, \gamma x2, \ldots, \gamma t]) = \gamma \lambda\ x1\ x2\ \ldots\ \bullet\ t \]

This function will be implemented using mk_simple_\lambda (q.v), not mk_\lambda.

See Also

list_mk_\lambda

Failure

May give rise to the error message from mk_simple_\lambda.
### 5.1. Syntactic Manipulations

| SML | val list.mk.simple.∀ : TERM list * TERM -> TERM; |
| --- | --- | --- |
| **Description** | Universally quantify a term with a list of variables. |
| **Definition** | $\text{list.mk.simple.∀} ([\gamma x1,\gamma x2,\gamma \ldots], \gamma body) = \forall x1 x2 \ldots \bullet body$ |
| This uses $\text{mk.simple.∀} (q.v)$ to generate its result. Note that giving an empty list paired with a non-boolean will return that term, rather than fail. |
| **See Also** | $\text{list.mk.∀}$ |
| **Failure** | This may give $\text{mk.simple.∀}$ error messages. |

| SML | val list.mk.simple.∃ : TERM list * TERM -> TERM; |
| --- | --- | --- |
| **Description** | Existentially quantify a term with a list of variables. |
| **Definition** | $\text{list.mk.simple.∃} ([\gamma x1,\gamma x2,\gamma \ldots], \gamma body) = \exists x1 x2 \ldots \bullet body$ |
| This uses $\text{mk.simple.∃} (q.v)$ to generate its result. Note that giving an empty list paired with a non-boolean will return that term, rather than fail. |
| **See Also** | $\text{list.mk.∃}$ |
| **Failure** | This may give $\text{mk.simple.∃}$ error messages. |

| SML | val list.mk→type : TYPE list -> TYPE; |
| --- | --- | --- |
| **Description** | Create the type of a multi-argument function. |
| **Definition** | $\text{list.mk→type} [\nu ty1,\nu ty2,\nu\ldots,\nu tyn] = \nu ty1 \rightarrow \nu ty2 \rightarrow \nu\ldots \rightarrow tyn$ |
| The supplied list may not be empty. |
| **Errors** | 3017 An empty list argument is not allowed |

| SML | val list.mk.∧ : TERM list -> TERM; |
| --- | --- | --- |
| **Description** | Conjoin a list of terms: |
| **Definition** | $\text{list.mk.∧} [\nu a,\nu b,\nu c,\nu \ldots] = \nu a \land \nu b \land \nu c \ldots$ |
| **Errors** | 3017 An empty list argument is not allowed |
| 3031 ?0 is not of type $\nu:BOOL$ |

| SML | val list.mk.∨ : TERM list -> TERM; |
| --- | --- | --- |
| **Description** | A function to make a disjunction of a list of terms. |
| **Definition** | $\text{list.mk.∨} [\nu a,\nu b,\nu c,\nu \ldots] = \nu a \lor \nu b \lor \nu c \ldots$ |
| **Errors** | 3017 An empty list argument is not allowed |
| 3031 ?0 is not of type $\nu:BOOL$ |
val list.mk ⇒ : TERM list −> TERM;

**Description**   Makes a multiple implication term, using \( mk ⇒ (q.v. \).)

**Definition**   \( \text{list.mk⇒ } [\iota t1, \iota t2, ..., \iota tn] = \iota t1 ⇒ \iota t2 ⇒ ... ⇒ \iota tn \)

Note that giving a singleton list containing a non-boolean will return that term, rather than fail.

**Errors**

\[ 3015 \quad ?1 \text{ is not of type } \iota : BOOL \]
\[ 3017 \quad \text{An empty list argument is not allowed} \]
\[ 3031 \quad ?0 \text{ is not of type } \iota : BOOL \]

val list.mk ∀ : TERM list * TERM −> TERM;

**Description**   Repeatedly universally quantify a term.

**Definition**   \( \text{list.mk∀ } ([\iota a, \iota b, \iota c, ...], \iota body) = \forall a b c ... • body \)

This uses \( mk ∀ \) to generate its result.

**Failure**   This may give the errors of \( mk ∀ \).

val list.mk ∃ : TERM list * TERM −> TERM;

**Description**   Repeatedly existentially quantify a term.

**Definition**   \( \text{list.mk∃ } ([\iota a, \iota b, \iota c, ...], \iota body) = \exists a b c ... • body \)

This uses \( mk ∃ \) to generate its result.

**Failure**   This may give the errors of \( mk ∃ \).

val list.mk ϵ : TERM list * TERM −> TERM;

**Description**   Repeatedly apply \( ϵ \) to a term.

**Definition**   \( \text{list.mkϵ } ([\iota a, \iota b, \iota c, ...], \iota body) = ϵ a b c ... • body \)

**Failure**   This may give the errors of \( mk ϵ \).

val list.mk λ : (TERM list * TERM) −> TERM;

**Description**   Repeatedly \( λ \)-abstract from a term.

**Definition**   \( \text{list.mkλ } ([\iota a, \iota b, \iota c, ...], \iota body) = \lambda a b c ... • body \)

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 \( mk λ \).
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**

\[ \text{mk_app}(\text{⌜}f\text{⌝}, \text{⌜}t\text{⌝}) = \text{⌜}f\text{⌜}t\text{⌝} \]

Note that many derived term constructs, e.g. all quantifications, are also applications. Thus

**Example**

\[ \text{mk_app}(\text{⌜}$\forall$\text{⌜, \⌜\lambda x \cdot t\text{⌝} = \forall x \cdot t\text{⌝} \]

**Errors**

3005  Cannot apply ?0 to ?1 as types are incompatible

3006  Type of ?0 not of form $\text{⌜}: ty1 \rightarrow ty2\text{⌝}$

SML

val mk_binder : string -> int -> (TYPE -> TYPE -> TERM) -> (TERM * TERM) -> TERM;

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

**Definition**

\[ \text{mk_binder area msg binder_n}(\text{⌜}\text{varstruct}\text{⌝}, \text{⌜}\text{body}\text{⌝}) = \]

\[ \text{⌜}\text{binder'}(\lambda \text{varstruct} \bullet \text{body})\text{⌝} = \]

\[ \text{⌜}\text{binder'} \text{varstruct} \bullet \text{body}\text{⌝} \]

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 (t₁, t₂) attempts to form \( \langle t₁ \ rator \ t₂ \rangle \). `rator' is gained by applying `rator_fn` to the types of \( t₁ \) and \( t₂ \).

Example

```
val mk_bin_op = mk_bin_op "mk∧" 3031 3015 (fn _ => fn _ =>> \( \langle a, b \rangle \) = \( \langle a ∧ b \rangle \)
```

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

```
val mk_char = "a" = \( \langle a \rangle \)
```

Errors

```
3023 String ?0 is not a single character
```

val mk_const : (string * TYPE) -> TERM;

Description This produces a constant.

Definition

```
val mk_const = ("c", \[⌜:ty1⌝, ⌜:ty2⌝, ⋮]) = \( :\langle ty1, ty2, ⋮ \rangle c \)
```

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 `=`, `⇒`, `∀`, and `∃`.

Errors

```
3002 Type of constant with name "=" must be of form: \( ty1 \rightarrow ty1 \rightarrow BOOL \)
3003 Type of constant with name "⇒" must be of form: \( BOOL \rightarrow BOOL \rightarrow BOOL \)
3004 Type of constant with name ?0 must be of form: \( (ty1 \rightarrow BOOL) \rightarrow BOOL \)
```

val mkctype : 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

```
val mkctype ("tc", ⌜:ty1\, ty2\, ⋮\}) = \( :\langle ty1, ty2, ⋮ \rangle tc \)
```

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
5.1. Syntactic Manipulations

**SML**

```sml
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**  
\[ \text{mk\_empty\_list } \triangleright: \text{ty} \rightarrow \text{ty LIST} \]

**See Also**  
`mk_list`

**SML**

```sml
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 \( \text{TY} \) is \( \text{TY SET} \). If the term list is empty the function will fail (see `mk\_\nothing`). The set must be of terms with the same HOL type.

**Definition**  
\[ \text{mk\_enum\_set } [\triangleright: a, b, ...] = \triangleright: \{ a; b; ...\} \]

**Errors**  
3012 ?0 and ?1 do not have the same types  
3017 An empty list argument is not allowed

**SML**

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

**Description**  
A derived term constructor function for generating equations.

**Definition**  
\[ \text{mk\_eq } (\triangleright: a, b) = \triangleright: a = b \]  
\[ \text{mk\_eq } (\triangleright: a:BOOL, b:BOOL) = \triangleright: a \iff b \]

**Errors**  
3012 ?0 and ?1 do not have the same types

**SML**

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

**Description**  
Make a floating point literal.

**Definition**  
\[ \text{mk\_float } (\triangleright: x, 0, \triangleright: 0) = \triangleright: XX. \]  
\[ \text{mk\_float } (\triangleright: x, p, \triangleright: 0) = \triangleright: XX.YY. \]  
\[ \text{mk\_float } (\triangleright: x, p, z) = \triangleright: XX.YYeZZ. \]

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**  
4041 the mantissa of a HOL floating point literal must be non-negative

**SML**

```sml
val mk_f : TERM;
```

**Description**  
The term \( \triangleright: F : BOOL \).
SML

val mk_if : (TERM * TERM * TERM) -> TERM;

**Description** Make a conditional.

**Definition**
mk_if (⌜c⌝, ⌜y⌝, ⌜n⌝) = ⌜if c then y else n⌝

**Errors**
3012 ?0 and ?1 do not have the same types
3031 ?0 is not of type "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**
mk_list ([⌜x⌝, ⌜y⌝], ⌜x + y⌝) = ⌜let x = 1 in x + 1⌝
mk_list ([⌜f (x,y)⌝, ⌜(1,2)⌝], ⌜x + y⌝) = ⌜let f = λ (x,y) • (1,2) in x + y⌝

**Errors**
3012 ?0 and ?1 do not have the same types
4007 ?0 is not a well-formed LHS for mk_list

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 mk_empty_list). The list must be of terms with the same HOL type.

**Definition**
mk_list [⌜a⌝, ⌜b⌝, ...] = ⌜[a; b; ...]⌝

**Errors**
3012 ?0 and ?1 do not have the same types
3017 An empty list argument is not allowed
5.1. Syntactic Manipulations

SML

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.

SML

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

SML

val mk_pair : (TERM * TERM) -> TERM;

Description A derived term constructor function for generating pairs.

Definition

mk_pair(⌜t1⌟, ⌜t2⌟) = ⌜(t1, t2)⌟

SML

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 ¬¬:BOOL
4016 ?0 is not an allowed variable structure

SML

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(λvar • 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, 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.
SML

```sml
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

SML

```sml
val mk_simple_type : DEST_SIMPLE_TYPE -> TYPE;
```

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

SML

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

**Description**  A derived term constructor function for generating simple `\forall`-terms.

**Definition**

\[
\text{mk\_simple\_All}(⌜\text{var}⌝, ⌜\text{body}⌝) = ⌜\forall \text{ var} \cdot \text{body}⌝
\]

\text{var} must be a term variable.

**See Also**  mk_\forall

**Errors**
- 3007  `?0` is not a term variable
- 3015  `?1` is not of type `⌜:BOOL⌝`

SML

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

**Description**  A derived term constructor function for generating simply abstracted `\exists`-I-terms.

**Definition**

\[
\text{mk\_simple\_Exists}(⌜\text{var}⌝, ⌜\text{body}⌝) = ⌜\exists_i \text{ var} \cdot \text{body}⌝
\]

\text{var} must be a variable.

**Errors**
- 3007  `?0` is not a term variable
- 3015  `?1` is not of type `⌜:BOOL⌝`

**See Also**  mk_\exists
5.1. Syntactic Manipulations

SML

\[ \text{val mk\_simple\_∃} : (\text{TERM} \times \text{TERM}) \rightarrow \text{TERM}; \]

**Description**  
A derived term constructor function for generating simple $\exists$-terms.

**Definition**  
\[ \text{mk\_simple\_∃} (⌜\text{var}⌝, ⌜\text{body}⌝) = ⌜\exists \text{ var} \bullet \text{body}⌝ \]

*var* must be a term variable.

**See Also**  
\text{mk\_∃}

**Errors**
\[ 3007 \text{ ?0 is not a term variable} \]
\[ 3015 \text{ ?1 is not of type } \vdash \text{: BOOL} \]

SML

\[ \text{val mk\_simple\_λ} : (\text{TERM} \times \text{TERM}) \rightarrow \text{TERM}; \]

**Description**  
This produces a simple $\lambda$-abstraction. It may only abstract variables.

**Definition**  
\[ \text{mk\_simple\_λ} (⌜\text{v}⌝, ⌜\text{t}⌝) = ⌜\lambda \text{ v} \bullet \text{t}⌝ \]

**See Also**  
\text{mk\_λ}

**Errors**
\[ 3007 \text{ ?0 is not a term variable} \]

SML

\[ \text{val mk\_string} : \text{string} \rightarrow \text{TERM}; \]

**Description**  
Construct a string literal.

**Example**  
\[ \text{mk\_string } "\text{abc}" = ⌜"\text{abc}"⌝ \]

SML

\[ \text{val mk\_term} : \text{DEST\_TERM} \rightarrow \text{TERM} \]

**Description**  
Create a term from a derived term. It is an inverse to \text{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. \text{mk\_const} and \text{mk\_∀}), with what checks they use.

**Failure**  
This function will fail with the same messages as the appropriate term constructor functions.

SML

\[ \text{val mk\_t} : \text{TERM}; \]

**Description**  
The term $⌜T:\text{BOOL}⌝$.

SML

\[ \text{val mk\_vartype} : \text{string} \rightarrow \text{TYPE}; \]

**Description**  
Create a HOL type variable from a string:

**Definition**  
\[ \text{mk\_vartype } "\text{tv}" = ⌜'tv⌝ \]
### mk_var

**Description:** This produces a term variable. The function makes no checks against the declaration of the subtypes of the type supplied.

**Definition:**

\[
\text{mk_var}(v; ty) = \downarrow v : ty^\gamma
\]

### mk_empty

**Description:** A derived term constructor function for generating an empty (enumerated) set with elements of a given type.

**Definition:**

\[
\text{mk_empty} \downarrow t; ty^\gamma = \downarrow \emptyset : ty \text{ SET}^\gamma
\]

**See Also:** mk_enum_set

### mk_iff

**Description:** A derived term constructor function for generating bi-implications.

**Definition:**

\[
\text{mk_iff}(t_1; t_2) = \downarrow t_1 \iff t_2^\gamma
\]

### mk_type

**Description:** Create a function type from two types. A function type is just a kind of compound type.

**Definition:**

\[
\text{mk_type}(\downarrow ty_1^\gamma, \downarrow ty_2^\gamma) = \downarrow ty_1 \rightarrow ty_2^\gamma
\]

### mk_conj

**Description:** A derived term constructor function for generating conjunctions.

**Definition:**

\[
\text{mk_conj}(t_1; t_2) = \downarrow t_1 \land t_2^\gamma
\]

### mk_disj

**Description:** A derived term constructor function for generating disjunctions.

**Definition:**

\[
\text{mk_disj}(t_1; t_2) = \downarrow t_1 \lor t_2^\gamma
\]
### 5.1. Syntactic Manipulations

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mk_¬</code> : <code>TERM</code> $\rightarrow$ <code>TERM</code>;</td>
<td>A derived term constructor function for generating negations.</td>
<td><code>3031</code></td>
</tr>
<tr>
<td><code>mk_⇒</code> : `(TERM * TERM) $\rightarrow$ TERM;</td>
<td>A derived term constructor function for generating implications.</td>
<td><code>3015</code></td>
</tr>
<tr>
<td></td>
<td>It takes two arguments: the antecedent and the consequent.</td>
<td><code>3031</code></td>
</tr>
<tr>
<td><code>mk_∀</code> : `(TERM * TERM) $\rightarrow$ TERM;</td>
<td>A derived term constructor function for generating <code>∀</code>-terms.</td>
<td><code>3015</code></td>
</tr>
<tr>
<td><code>mk_∃</code> : `(TERM * TERM) $\rightarrow$ TERM;</td>
<td>A derived term constructor function for generating <code>∃</code>-terms.</td>
<td><code>3015</code></td>
</tr>
</tbody>
</table>

**See Also**
- `mk_simple_∀`
- `mk_∃.1`
SML | val mk_∃ : (TERM * TERM) -> TERM;

Description  A derived term constructor function for generating ∃-terms.

Definition | mk_∃ (⌜varstruct⌝, ▼body) = ▼∃ varstruct • body

describe varstruct may be any allowed variable structure.

Errors
| 3015 ?1 is not of type ▼:BOOL
| 4016 ?0 is not an allowed variable structure

See Also  mk_simple_∃

SML | val mk_×_type : (TYPE * TYPE) -> TYPE

Description  mk_×_type (⌜:ty_1⌝, ▼:ty_2) returns a pair type: ▼:ty_1 × ty_2.

SML | val mk_ϵ : (TERM * TERM) -> TERM;

Description  A derived term constructor function for generating ϵ-terms.

Definition | mk_ϵ (⌜varstruct⌝, ▼body) = ▼ϵ varstruct • body

describe varstruct may be any allowed variable structure.

Errors
| 3015 ?1 is not of type ▼:BOOL
| 4016 ?0 is not an allowed variable structure

SML | val mk_λ : TERM * TERM -> TERM

Description  This creates a λ-abstraction of an allowed variable structure from a term.

Example
| mk_λ (⌜x⌝, ▼x + y) = ▼λ x • x + y
| mk_λ (⌜(x, y)⌝, ▼x + y) = ▼λ (x, y) • x + y
| mk_λ (⌜((x1, x2), (y1, y2))⌝, ▼x2 + y2) = ▼λ ((x1, x2), (y1, y2)) • x2 + y2

See Also  mk_simple_λ

Errors
| 4016 ?0 is not an allowed variable structure

SML | val mk_ℕ : INTEGER -> TERM;

Description  Construct a numeric literal: the argument may not be negative.

Example
| mk_ℕ 5 = ▼5

Errors
| 3021 ?0 should be 0 or positive

SML | 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 | val rename : (string * TYPE) -> string -> TERM -> 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 | val set_variant_suffix : string -> 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 | val string_of_term : TERM -> string; |
| Description | This returns a display of a term in the form of a string, with no inserted new lines, suitable for use with diag_string and fail. |
| See Also | format_term is a formatted string display of a term. |

| SML | val string_of_type : TYPE -> string; |
| Description | This returns a display of a type in the form of a string, with no inserted new lines, suitable for use with diag_string and fail. |
| See Also | format_type is a formatted string display of a type. |

| SML | val string_variant : string list -> string -> string; |
| Description | string_variant vlist name returns a string that is a 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 variant if term variables are already destroyed, and their names and types are directly accessible. |
| See Also | variant |

| SML | val STRING : TYPE; |
| Description | This is the HOL type of strings, a type abbreviation for lists of objects of type CHAR. |
| Definition | val STRING = "CHAR LIST"; |
| See Also | Theory “char”. |

| SML | val strip_app : TERM -> TERM * 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 | strip_app "t t1 t2 t3 ..." = ("t", ["t1", "t2", "t3", ...]) |
| | strip_app "T" = ("T", []) |

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUAL USR029
val strip_binder : string -> TERM -> TERM list * TERM;

Description   strip_binder binder applied to
               \(\langle \lambda v_1 \bullet \text{binder}(\lambda v_2 \bullet \ldots \bullet \text{body}) \ldots \rangle\)\n
will return
               \[\langle \text{\(v_1\)}, \text{\(v_2\)}, \ldots, \text{\(\text{body}\)}\rangle\]

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 } "\wedge" \langle a \wedge (b \wedge c) \wedge d \rangle = \langle \text{\(a\)}, \text{\(b \wedge c\)}, \text{\(d\)}\rangle\]

val strip_leaves : ('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, dest_\&), recursively descend the results of destruction down both branches, destroying until failure.

Example

\[\text{strip_leaves dest_\& } \langle a \& (b \& c) \& d \rangle = \langle \text{\(a\)}, \text{\(b \& c\)}, \text{\(d\)}\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{\(x = 1\) in let } y = 2 \text{ in } x+y \rangle = \langle \langle \text{\(x = 1\)}, \langle \text{\(y = 2\)}\rangle \rangle, \text{\(x+y\)}\rangle\]

val strip_simple_binder : string -> TERM -> TERM list * TERM;

Description   strip_simple_binder binder applied to
               \(\langle \lambda v_1 \bullet \text{binder}(\lambda v_2 \bullet \ldots \bullet \text{body}) \ldots \rangle\)

will return
               \[\langle \text{\(v_1\)}, \text{\(v_2\)}, \ldots, \text{\(\text{body}\)}\rangle\]

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 | val strip_simple_\forall : TERM \rightarrow (TERM list \times TERM); |
| Description | Strip a multiply universally simply quantified term. |
| Definition | \([strip_simple_\forall \forall a b c \ldots \bullet body^{\gamma} = \[\gamma a^{\gamma}, \gamma b^{\gamma}, \gamma c^{\gamma}, \ldots\], \gamma body^{\gamma}]\) |

| SML | val strip_simple_\exists : TERM \rightarrow (TERM list \times TERM); |
| Description | Strip a repeatedly existentially simply quantified term. |
| Definition | \([strip_simple_\exists \exists a b c \ldots \bullet body^{\gamma} = \[\gamma a^{\gamma}, \gamma b^{\gamma}, \gamma c^{\gamma}, \ldots\], \gamma body^{\gamma}]\) |

| SML | val strip_spine_left : ('a \rightarrow 'a * '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_\land), recursively descend the left results of destruction, destroying until failure. |
| Example | \([strip_spine_left dest_\land \land (a \land b) \land c \land d^{\gamma} = \[\gamma a^{\gamma}, \gamma b^{\gamma}, \gamma c^{\gamma}, \gamma d^{\gamma}\]]\) |

| SML | val strip_spine_right : ('a \rightarrow 'a * '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_\land), recursively descend the right results of destruction, destroying until failure. |
| See Also | strip_bin_op for stripping terms formed by binary (constant_op) term operators. |
| Example | \([strip_spine_left dest_\land \land (a \land b) \land c \land d^{\gamma} = \[\gamma a^{\gamma}, \gamma b^{\gamma}, \gamma c^{\gamma}, \gamma d^{\gamma}\]]\) |

| SML | val strip_\rightarrow_type : TYPE \rightarrow TYPE list; |
| Description | Strip the type of a multi-argument function into its constituent types, only descending into the right hand result of dest_\rightarrow_type. |
| Definition | \([strip_\rightarrow_type \rightarrow ty1 \rightarrow ... \rightarrow ty_n^{\gamma} = \[\gamma ty1^{\gamma}, \gamma ty2^{\gamma}, \gamma ty3^{\gamma}, \gamma ty_n^{\gamma}\]]\) |

<p>| SML | val strip_\land : TERM \rightarrow TERM list |
| Description | Break a term into its constituent conjuncts, descending recursively only to the right. |
| Example | ([strip_\land \land a \land (b \land c) \land d^{\gamma} = [\gamma a^{\gamma}, \gamma b^{\gamma}, \gamma c^{\gamma}, \gamma d^{\gamma}]]) |</p>
<table>
<thead>
<tr>
<th>SML</th>
<th>[ val \text{strip} \lor : \text{TERM} \to \text{TERM list} ]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Break a term into its constituent disjuncts, descending recursively only to the right.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>[ \text{strip} \lor \Gamma a \lor (b \lor c) \lor d \lor = [\Gamma a \lor , \Gamma b \lor , \Gamma c \lor , \Gamma d \lor ] ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>[ val \text{strip} \Rightarrow : \text{TERM} \to \text{TERM list}; ]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Strip a multiple implication into a list of antecedents appended to the singleton list of the innermost consequent.</td>
</tr>
<tr>
<td><strong>Definition</strong></td>
<td>[ \text{strip} \Rightarrow \Gamma t1 \Rightarrow t2 \Rightarrow ... \Rightarrow tn \lor = [\Gamma t1 \lor , \Gamma t2 \lor , ... , \Gamma tn \lor ] ]</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>Note that stripping a non-boolean will result in a singleton list containing that term, not a fail.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>[ val \text{strip} \forall : \text{TERM} \to (\text{TERM list} \times \text{TERM}); ]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Strip a multiply universally quantified term (perhaps with paired abstractions).</td>
</tr>
<tr>
<td><strong>Definition</strong></td>
<td>[ \text{strip} \forall \Gamma \forall a b c ... \cdot \text{body} \lor = [\Gamma a \lor , \Gamma b \lor , \Gamma c \lor , ... , \Gamma \text{body} \lor ] ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>[ val \text{strip} \exists : \text{TERM} \to (\text{TERM list} \times \text{TERM}); ]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Strip a repeatedly existentially quantified term, possibly formed with paired abstractions.</td>
</tr>
<tr>
<td><strong>Definition</strong></td>
<td>[ \text{strip} \exists \Gamma \exists a b c ... \cdot \text{body} \lor = [\Gamma a \lor , \Gamma b \lor , \Gamma c \lor , ... , \Gamma \text{body} \lor ] ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>[ val \text{strip} \epsilon : \text{TERM} \to (\text{TERM list} \times \text{TERM}); ]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Strip multiple (\epsilon)’s.</td>
</tr>
<tr>
<td><strong>Definition</strong></td>
<td>[ \text{strip} \epsilon \Gamma \epsilon a b c ... \cdot \text{body} \lor = [\Gamma a \lor , \Gamma b \lor , \Gamma c \lor , ... , \Gamma \text{body} \lor ] ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>[ val \text{strip} \lambda : \text{TERM} \to (\text{TERM list} \times \text{TERM}); ]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Strip a multiple (\lambda)-abstraction.</td>
</tr>
<tr>
<td><strong>Definition</strong></td>
<td>[ \text{strip} \lambda \Gamma \lambda a b c ... \cdot \text{body} \lor = [\Gamma a \lor , \Gamma b \lor , \Gamma c \lor , ... , \Gamma \text{body} \lor ] ]</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>This uses \text{dest} \lambda (q.v.) rather than \text{dest_simple} \lambda.</td>
</tr>
</tbody>
</table>
5.1. Syntactic Manipulations

val subst : (TERM * TERM) list --> TERM --> TERM;

Description subst [(t₁₁, u₁₁), (t₁₂, u₁₂), ...] t returns the term formed from t by parallel substitution of the t₁ᵢ for the u₁ᵢ. The uᵢ can be variables or arbitrary terms but only “free” occurrences of a uᵢ will be changed (i.e., only occurrences in which no free variable of uᵢ becomes a bound variable in t). Bound variables in t are renamed as necessary to prevent bound variable capture.

If some uᵢ appears more than once in the substitution list, say uᵢ = uⱼ for i < j, then the later pair (tⱼ, uⱼ) is ignored.

subst does not perform type instantiation: each tᵢ must have the same type as the corresponding uᵢ.

Definition subst [(⌜t₁⌝, ⌜u₁⌝), (⌜t₂⌝, ⌜u₂⌝), ...] ⌜t⌝ = ⌜t[⌜t₁/u₁, t₂/u₂, ...]⌝

See Also var_subst

Errors 3012 0 and 1 do not have the same types

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.

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)

val term_diff : (TERM list * TERM list) --> TERM list;

Description Remove any terms in the first list that are α-convertible to any in the second. An infix function.

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.

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.

val term_grab : (TERM list * TERM) --> TERM list;

Description If the given term is not α-convertible to any member of the list, then add it to the list. An infix function.
\begin{verbatim}
val term_less : (TERM list * TERM) -> TERM list;

Description Remove any terms in the list that are \(\alpha\)-convertible to the given term. An infix function.

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.

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 \(\alpha\)-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

val term_mem : (TERM * TERM list) -> bool;

Description Is the given term \(\alpha\)-convertible to any term in the list? An infix function.

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).

val term_types : TERM -> TYPE list;

Description Gives a list of all the types of constants, variables or \(\lambda\)-abstraction variables within the term argument.

val term_tyvars : TERM -> string list;

Description Returns the list of type variable names present in types present within a term.

val term_union : (TERM list * TERM list) -> TERM list;

Description Take the union of two term lists viewed as sets, removing any \(\alpha\)-convertible duplicates. An infix function.

See Also \(\text{union}\) for precise ordering of result.

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.
\end{verbatim}
5.1. Syntactic Manipulations

SML

\[ \text{val type\_any : (TYPE -> bool) -> TYPE -> bool;} \]

**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.

SML

\[ \text{val type\_fail : string -> int -> TYPE list -> 'a;} \]

**Description** `type\_fail area msg ty1` 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.

SML

\[ \text{val type\_map : (TYPE -> TYPE) -> TYPE -> TYPE;} \]

**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.

SML

\[ \text{val type\_match1 : (TYPE * TYPE) list -> TYPE -> TYPE -> (TYPE * TYPE)list;} \]

**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.

\[ \text{type\_match1[[(\forall'\ b \rightarrow \ N \implies \forall'\ a \rightarrow \ 'b\ ^\forall'\ a)] \implies (\forall'\ b \rightarrow \ N \implies \forall'\ a \rightarrow \ 'b\ ^\forall'\ a)];} \]

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 \text{ ?0 is not a type instance of ?1 in the supplied context} \\
3019 \text{ ?0 is not a type variable}

SML

\[ \text{val type\_match : TYPE -> TYPE -> (TYPE * TYPE)list;} \]

**Description** `type\_match ty1 ty2` attempts to match `ty1` with `ty2`, i.e., to determine if `ty1` can be obtained from `ty2` 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:

\[ \text{type\_match \forall(\forall'\ a \rightarrow \ N) \rightarrow \forall'\ a \rightarrow \forall'\ a = [\forall'\ a \rightarrow \ N, \forall'\ a];} \]

**See Also** `type\_match1`, `inst\_type`

**Errors**

3053 \text{ ?0 is not a type instance of ?1}

SML

\[ \text{val type\_of : TERM -> TYPE;} \]

**Description** This gives the HOL type of a term.
The page contains code snippets and descriptions of functions in the SML language. Here is a summary of the content:

- **type_tycons**: Returns a list of names of type constructors, and the arity of their use, within a type.

- **type_tyvars**: Returns the list of type variable names present in a type.

- **variant**: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).

- **var_subst**: 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.

- **~=$**: An infix equality test that returns true only when its two term arguments are α-convertible, and false otherwise: no exceptions can be raised. Equality of terms is gained by using =$.

- **N**: This is the HOL type of the natural numbers, 0, 1, ....

The page also includes notes on errors and see also references for further reading.
5.2 Discrimination Nets

**SML**

```sml
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.

  type 'a NET;

  Description This is the type of a discrimination net, its type parameter being the type of values that are handled by the net.

  val empty_net : 'a NET;

  Description This is the starting discrimination net, which returns an empty list of values, regardless of term form.

  val list_net_enter : (TERM * 'a) list => ('a NET) => ('a NET);

  Description This enters a list of values and indexing terms into a discrimination net, returning the resulting net.

  val make_net : (TERM * 'a) list => ('a NET);

  Description This enters a list of values and indexing terms into an empty discrimination net, returning the resulting net.

  val net_enter : (TERM * 'a) => ('a NET) => ('a NET);

  Description This enters a value and its indexing term into a discrimination net, returning the resulting net.

  val net_lookup : ('a NET) => TERM => ('a list);

  Description 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 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 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 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 net_lookup term matches the index term then the second entered value will be returned before the first in the list of matches.
```
5.3 Higher-order Matching

SML

```sml
val simple_ho_match : TERM -> TERM ->
  (TYPE * TYPE) list * (TERM * TERM)list;
```

Description  
`simple_ho_match` implements the higher-order matching algorithm of Miller and Nipkow. `term_match t p` attempts to find type and term instantiations under which `p` becomes $\alpha \beta \eta$-equivalent to `t`. The algorithm is a decision procedure for this problem in the case when `p` is a linear pattern, i.e., modulo, $\eta$-equivalence, `p` is contains no $\beta$-redexes and for each free variable `v` of `p` there is an `n` such that each occurrence of `v` is in a subterm of the form `v x_1 \ldots x_n` where the `x_i` are distinct bound variables and where this subterm does not itself occur as the function in a function application (i.e., it is either the argument of a function application or the body of a $\lambda$-abstraction). The implementation does not check that `p` is linear pattern but will always fail rather than produce a pair of instantiations that do not make `p $\alpha \beta \eta$-equivalent to `t`.

For example,

```sml
simple_ho_match $\forall x \cdot x < 1$ $\forall x : 'a \cdot P x$ =
  ([(⌜$\forall x : 'a \cdot x < 1$, $\forall P : N \to BOOL$)\])
```

```sml
simple_ho_match $\forall x \cdot x < 1$ $\forall x : 'a \cdot P x y$ =
  ([(⌜$\forall x : 'a \cdot x < 1$, $\forall P : N \to 'b \to BOOL$)\])
```

Trivial associations (i.e. `(x, x)`) are omitted from the association lists returned by `ho_match`.

This function is “simple” because although it does work with paired abstractions, it cannot match terms where the pairing structure is not identical: e.g., it can match the pattern $\lambda (x, y) \cdot G x y$ to the target $\lambda (m, n) \cdot m + n$, it cannot match it to $\lambda p \cdot \text{Fst } p$.

Errors

120001?0 is not a higher-order instance of ?1
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 \( \alpha \)-conversion). Though it would be type correct, we conventionally do not use this type name for other functions of type \( \ldots \rightarrow TERM \rightarrow THM \).

Definition  
\[
type CONV = TERM \rightarrow THM;
\]

SML
type SEQ;

Description  This is the type of sequents, consisting of a list of assumptions and a conclusion.

Definition  
\[
type SEQ = (TERM \ list) \ast TERM;
\]

=\# provides a strict equality test on sequents, \( \sim \)\# provides an equality test on the sequents up to \( \alpha \)-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} OEDICT</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} OEDICT</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 OEDICT</td>
<td>A dictionary of axioms.</td>
</tr>
<tr>
<td>defn_dict</td>
<td>THM OEDICT</td>
<td>A dictionary of definitions.</td>
</tr>
<tr>
<td>thm_dict</td>
<td>THM OEDICT</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. 

\[\vdash\] provides a strict equality test on the conclusion and assumptions of theorems. 
\[\sim=\vdash\] provides an equality test on the conclusion and assumptions of theorems up to \(\alpha\)-convertibility and order of assumptions.
6.1. Standard ML Type Definitions

**SML**

```
type USER_DATA;
```

**Description**  This is the type of a store for objects of type `USER_DATUM`. It is implemented as:

```
ML type USER_DATA = USER_DATUM S_DICT;
```

**Uses**  Within the type `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

SML

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

SML

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 ','

SML

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
### SML

```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 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 ","
val declare_prefix : (int * string) -> unit;

Description  declare_prefix (p, s) declares 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 ",",

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_terminator : string -> unit

Description  declare_terminator 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.

Errors
20101 The string ?0 is not a valid terminator. Terminators must start with a symbolic character, must not contain spaces, and must not end with underscore, λ or γ
20102 The string ?0 is already declared as a terminator

val declare_type_abbrev : (string * string list * TYPE) -> unit;

Description  declare_type_abbrev (s, [α_1, . . . , α_k], τ) declares (α_1, . . . , α_k)s as a type abbreviation for the type τ. 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.

Errors
20401 The identifier ?0 is already declared as a type abbreviation
20402 The identifier ?0 is already declared as a type constructor
20407 The formal parameter list ?0 contains duplicate type variable names
20408 The string ?0 is not an identifier

val expand_type_abbrev : (string * TYPE list) -> TYPE;

Description  expand_type_abbrev s, [τ_1, . . . , τ_k] is the expansion of the type abbreviation s with respect to the arguments [τ_1, . . . , τ_k].

Errors
20404 The identifier ?0 is not declared as a type abbreviation
20405 The type abbreviation ?0 should have ?1 argument not ?2
20406 The type abbreviation ?0 should have ?1 arguments not ?2

val get_aliases : string -> (string * TERM) list;

Description  get_aliases 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.

Errors
20601 There is no theory called ?0

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
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 antiunifier 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.
The SML functions for managing theories and theorems include:

- **get_language**: 
  `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

- **get_left_infixes** and **get_right_infixes**: 
  `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

- **get_nonfixes**: 
  `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

- **get_postfixes** and **get_prefixes**: 
  `val get_postfixes : string -> (int * string) list;`
  `val get_prefixes : 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

- **get_terminators**: 
  `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

- **get_type_abbrev**: 
  `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
6.2. Symbol Table

SML

| 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 ?0

SML

| 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.

SML

| 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 ?0

SML

| 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 ?0

SML

| 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 ?0

SML

| val is_type_abbrev : string -> bool; |

Description  is_type_abbrev s returns true iff. s is declared as a type abbreviation

SML

| val resolve_alias : (string * TYPE) -> TERM; |

Description  resolve_alias(s, τ) returns a term of the form mk_const(c, τ) 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 τ’ where τ’ is an instance of τ. If s is an alias then c is taken from the alias declaration for s in which the aliased constant has a type τ’ which can be instantiated to τ. If more than one such declaration is applicable the most recent one is used.

Errors  20304 The identifier ?0 is not a valid constant name (or alias) at this type
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
The Kernel Interface

**Signature**

```
signature KernelInterface = sig
```

**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**

```
6013  ?0 is ill-formed in current theory: type name ?1 is not declared
6014  ?0 is ill-formed in current theory: type name ?1 does not have arity used
6015  ?0 is ill-formed in current theory: constant name ?1 not declared
6038  ?0 is ill-formed in current theory: constant name ?1 cannot have type used
```

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.

**Datatype**

```
datatype KERNEL_INFERENCE =
  | KISubstRule of (THM * TERM) list * TERM * THM * THM
  | Kisimple\EqRule of TERM * THM * THM
  | KIIInstTypeRule of (TYPE * TYPE) list * THM * THM
  | KI⇒Intro of TERM * THM * THM
  | KI⇒Elim of THM * THM * THM
  | KIAsmRule of TERM * THM
  | KIReflConv of TERM * THM
  | KISimple\βConv of TERM * THM
  | KISucConv of TERM * THM
  | KIStringConv of TERM * THM
  | KIEqSymRule of THM * THM
  | KIListSimple\∀Elim of TERM list * THM * THM
  | KIEqTransRule of THM * THM * THM
  | KIMkAppRule of THM * THM * THM
  | KI⇔MPRule of THM * THM * THM
  | KISimple\∀Intro of TERM list * THM * THM
  | KIInstTermRule of (TERM * TERM) list * 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.
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>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Suppress compactification in new_axiom</td>
</tr>
<tr>
<td>2</td>
<td>Suppress compactification in new_const</td>
</tr>
<tr>
<td>4</td>
<td>Suppress compactification in new_type_defn</td>
</tr>
<tr>
<td>8</td>
<td>Suppress compactification in new_spec</td>
</tr>
<tr>
<td>16</td>
<td>Suppress compactification in save_thm</td>
</tr>
<tr>
<td>32</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.
6.3. The Kernel Interface

```sml
datatype KERNEL_STATE_CHANGE = OpenTheory of string * ((string list) * (string list))
| DeleteTheory of string
| NewTheory of string
| NewParent of string * (string list)
| LockTheory of string
| UnlockTheory of string
| DuplicateTheory of string * string
| SaveThm of string * THM
| ListSaveThm of string list * THM
| DeleteConst of TERM
| DeleteType of string
| DeleteAxiom of string
| DeleteThm of string
| NewAxiom of (string list * TERM)*THM
| NewConst of string * TYPE
| NewType of string * int
| SimpleNewDefn of (string list * string * TERM) * THM
| NewTypeDefn of (string list * string * (string list) * THM) * THM
| NewSpec of (string list * int * THM) * THM
| SetUserDatum of string * USER_DATUM;
```

**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 ("", [], `asm_rule mk_t`).

<table>
<thead>
<tr>
<th>Operation</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>open_theory</td>
<td>(thy, (inthys, 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>new_parent</td>
<td>(thy, 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>SimpleNewDefn</td>
<td>(arg, thm)</td>
<td><code>arg</code> gives the argument to the operation. <code>thm</code> is the new defining theorem.</td>
</tr>
<tr>
<td>NewTypeDefn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NewSpec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NewAxiom</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SEE ALSO** on_kernel_state_change, before_kernel_state_change
type CHECKPOINT;
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_const 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 α-convertibility and order of assumptions. =|# provides a strict equality test on sequents, ~|=|# provides an equality test on the sequents up to α-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

**val before_kernel_state_change :** *(KERNEL_STATE_CHANGE \(\rightarrow\) unit) \(\rightarrow\) 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 \(\rightarrow\) TYPE;
val compact_term : TERM \(\rightarrow\) TERM;
val compact_thm : THM \(\rightarrow\) THM;
```

**Description**  
These functions compactify type, term and theorem values, currently by commoning 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 \(\rightarrow\) 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

\begin{verbatim}
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
\begin{itemize}
\item 12035  Theory ?0 is not present in the current hierarchy
\item 6037  Theory ?0 is locked
\item 6069  Theory ?0 is in scope
\item 6071  Theory ?0 is a read–only ancestor
\item 6076  Theory ?0 has child theories
\end{itemize}
\end{verbatim}

SML

\begin{verbatim}
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
\begin{itemize}
\item 6037  Theory ?0 is locked
\item 6046  Key ?0 is not used for a theorem in theory ?1
\item 6071  Theory ?0 is a read–only ancestor
\end{itemize}
\end{verbatim}

SML

\begin{verbatim}
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.
\end{verbatim}
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
6.3. The Kernel Interface

\[\text{val do_in_theory : string \_\rightarrow \ ('a \_\rightarrow \ 'b) \_\rightarrow \ 'a \_\rightarrow \ 'b;}\]

**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:

\[\text{open_theory \ new_theory \ delete_theory}\]

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.
Chapter 6. THE MANAGEMENT OF THEORIES AND THEOREMS

SML

```sml
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.

SML

```sml
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

SML

```sml
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

SML

```sml
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
### 6.3. The Kernel Interface

**SML**

```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**

```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**

```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**

```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**

```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**

```sml
val get_current_theory_name : unit -> string;
```

**Description**  
Returns the name of the current theory.

**SML**

```sml
val get_current_theory_status : unit -> THEORY_STATUS;
```

**Description**  
This returns the current theory’s status.
Chapter 6. THE MANAGEMENT OF THEORIES AND THEOREMS

```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

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

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

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

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

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

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
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

val lock_theory : string -> unit;

**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

val new_axiom : (string list * TERM) -> THM

**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

val new_const : (string * TYPE) -> TERM;

**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

val new_parent : string -> unit;

**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
SML
val new_spec : (string list * int * THM) -> THM;

Description  new_spec (keylist, ndef, \( \vdash \exists x_1, \ldots, x_n \cdot p[x_1, \ldots, x_n] \)) 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
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

SML
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

val new_type_defn : 
   (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:

\[ \exists x : \text{type} \bullet p x \]

with no assumptions. The defining theorem will then be of the form:

\[ \exists f : \text{typars name} \rightarrow \text{type} \bullet \]

\[ \text{TypeDefn (p: type \rightarrow BOOL) f} \]

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: ‘\[ \exists x \bullet p x \]’
6055 ?0 is not of the form: ‘\[ \exists x \bullet p y \] 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, 
   type variables being: ?1
6071 Theory ?0 is a read—only ancestor
6079 ?0 repeated in type parameter list
6080 ?0 is not of the form: ‘\[ \exists x \bullet p y \] where \[ x \] equals \[ y \]’

SML

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

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
148 Chapter 6. THE MANAGEMENT OF THEORIES AND THEOREMS

SML

val on_kernel_state_change : (KERNEL_STATE_CHANGE -> unit) -> unit

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

SML

val open_theory : string -> unit;

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.

SML

val pending_reset_kernel_interface : unit -> unit -> unit;

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.

SML

val save_thm : (string * THM) -> 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
6.3. The Kernel Interface

| SML | val set_user_datum : (string * USER_DATUM) -> 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 |

| SML | val simple_new_defn : (string list * string * TERM) -> 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 |

| SML | val string_of_thm : THM -> 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. |

| SML | val thm_fail : string -> int -> THM list -> '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. |

| SML | val thm_theory : THM -> 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 |
val unlock_theory : string -> unit;

**Description**  unlock_theory thy 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

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

```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

- 20601  There is no theory called ?0
- 103101 This theorem does not prove the conjecture stored under key ?0
- 103102 The theorem with key ?0 does not prove this conjecture
- 103103 Theory ?0 is not in scope
- 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).
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.

dele_conjecture key deletes the conjecture stored in the current theory under key key. It returns the deleted conjecture.

dele_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 ⌜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.

```sml
find_thm [⌜x + y : R⌝, ⌜x * y : R⌝];
```

**See Also** `gen_find_thm`
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 α-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 have a conclusion that does not contain any natural number additions.

|gen_find_thm(TAll[any_substring_tt["plus", "minus"], no_subterm_tt["$+\mathbb{N} \to \mathbb{N} \to \mathbb{N}"]]);

See Also  find_thm
SPECIFICATION IN HOL

7.1 Constant Specification

<table>
<thead>
<tr>
<th>SML</th>
<th>signature ConstantSpecification = sig</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Description</strong> This is the signature of a structure supporting specification of constants.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>val consistent_def: THM;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>val const_spec_def: THM;</td>
</tr>
<tr>
<td></td>
<td>val const_spec_thm: THM;</td>
</tr>
<tr>
<td></td>
<td><strong>Description</strong> These theorems are ML variables bound to the definition of Consistent, the definition of ConstSpec and the theorem saved with key const_spec_thm in theory “basic_hol”.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>val consistent_def = ⊢ ∀ p • Consistent p ⇔ ∃ x • p x</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>val const_spec_def = ⊢ ∀ p c • ConstSpec p c ⇔ (Consistent p ⇒ p c)</td>
</tr>
<tr>
<td></td>
<td>val const_spec_thm = ⊢ ∀ p • ∃ x : 'a • ConstSpec p x</td>
</tr>
</tbody>
</table>
val const_spec : string list * TERM list * TERM -> THM;

**Description**  
`const_spec` is used to introduce new constants that satisfy some predicate.  
`const_spec (keys, varstructs, predicate)` declares constants matching the names and types of those in `varstructs`, with the property that, if `predicate` holds for some list of values, then it holds for the constants in question, with the definition saved under each of `keys`. In addition, if the existence prover held in the current proof context, and accessed by `current.cs∃conv`, can prove the predicate is satisfied for some witness (i.e. `⇔ T`), the specification will instead be made without the caveat. In more detail, if the list of free variables in `varstructs` is `c_1 ... c_n`, and the predicate is of the form `P[ c_1 ... c_n ] : BOOL`, then the specification of constants `c_1 ... c_n` (with same names and types as the variables) will be:

\[ \vdash \text{ConstSpec} (\lambda (x_1, ..., x_n) \cdot P[x_1,...,x_n]) (c_1,...,cn) \]

where each `x_i` is a variant (see `list_variant`) of `c_i`, not present in the original list of variables, or the predicate, and the `c_i` become constants. However, if we can automatically prove:  
\[ \exists c_1 ... c_n \cdot P[c_1,...,cn] \]  
then instead the specification will be:  
\[ \vdash P [c_1,...,cn] \]. There are a number of caveats on the production of the specification: there must be no duplicates in `varstructs`; all the constant’s types must have the same set of type variables; there must be no type variables in the body of `predicate` other than those already in the constant’s types; and there must be at least one key.

**See Also**  
`HOL_axiomatic_recogniser`, `get_spec`.

**Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3031</td>
<td><code>?0</code> is not of type <code>⌜BOOL⌝</code></td>
</tr>
<tr>
<td>6031</td>
<td>Key list may not be empty</td>
</tr>
<tr>
<td>6037</td>
<td>Theory <code>?0</code> is locked</td>
</tr>
<tr>
<td>6044</td>
<td>Must define at least one constant</td>
</tr>
<tr>
<td>6049</td>
<td>There is a constant called <code>?0</code> already in scope</td>
</tr>
<tr>
<td>6051</td>
<td>Key <code>?0</code> has already been used for a definition in theory <code>?1</code></td>
</tr>
<tr>
<td>6056</td>
<td><code>?0</code> is a free variable in <code>?1</code></td>
</tr>
<tr>
<td>6062</td>
<td><code>?0</code> are free variables in <code>?1</code></td>
</tr>
<tr>
<td>6061</td>
<td>the body of <code>?0</code> contains type variables not found in type of constants to be defined, the variables being: <code>?1</code></td>
</tr>
<tr>
<td>6063</td>
<td>There is a constant called <code>?0</code> in the descendants of the current theory</td>
</tr>
<tr>
<td>6071</td>
<td>Theory <code>?0</code> is a read-only ancestor</td>
</tr>
<tr>
<td>6081</td>
<td>Sets of type variables in <code>?0</code> and <code>?1</code> differ</td>
</tr>
<tr>
<td>46003</td>
<td>Variable name <code>?0</code> is repeated in varstructs</td>
</tr>
<tr>
<td>46004</td>
<td><code>?0</code> is not a varstruct</td>
</tr>
</tbody>
</table>
7.1. Constant Specification

SML

\[
\text{val get_spec : TERM -> THM;}
\]

**Description** `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. It it cannot find such a theorem it will determine the list of languages of the constant (e.g. “HOL” and “Z”). It will then attempt to strip a language name plus a prime, in a case-insensitive manner, from the constant name. If there is one such stripped name it will then search in the same theory for a definition or axiom using this stripped name as a key. If there is no such name it will fail with message 46005, or if more than one such stripped name with matching theorem (a very rare circumstance), it will fail with message 46013. If there is no such constant in scope then the function fails.

\[
\text{get_spec } \texttt{const t1 t2 ...'} \quad (\text{i.e. a constant applied to an arbitrary number of arguments}) \quad \text{will act as get_spec } \texttt{const}.
\]

If ICL conventions have been followed, then a definition (or axiom) found as above should be the definition of the constant named `const`. 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). Further, under such conventions, two constants of different languages but with the same name should not be saved in the same theory.

If the definitional theorem is not of the form:

\[
\models \text{ConstSpec } p \ c
\]

then the theorem is returned un-processed - if properly saved then it either comes from (simple_)
new_defn, new_spec, or predicate `p` was automatically proven consistent by `const_spec`.

Otherwise, 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:

\[
\Gamma \models \text{Consistent } p
\]

(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 function returns:

\[
\beta \text{ rule } '\Gamma \models p \ c'
\]

If not, then the function returns a theorem of the form:

\[
\beta \text{ rule } '\text{Consistent } p \models p \ c'
\]

No language-based key name stripping will be attempted for consistency theorem names.

**See Also** `push_consistency_goal` to set the appropriate consistency goal, \((\Gamma, "\text{Consistent } p")\), for a given constant name.

**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
46013 ?0 has more than one possible definition: ?1
val HOL_const_recogniser : string * string * Lex.INPUT list * string
  -> THM;

Description From the argument (which will be supplied by the reader/writer) this function
will derive a list of varstructs, and a predicate. These will then be passed to const_spec (q.v.)
which will attempt the requested specification. The keys under which the result will be saved are
just the names of the variables concerned.

Uses As an interface between the HOL reader/writer and const_spec - it is not intended for
direct use. It is used to process text of the form

Example Constant Specification
dcls
pred

Errors

46000  Input not of right form

Errors from const_spec may also occur: the messages area of origin will not be changed.

val push_consistency_goal : TERM -> unit;

Description push_consistency_goal "const" will first determine the specification theorem of
const, by executing 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
get_spec is designed, this (single) assumption will be of the form

⌜Consistent (λ vs[x1,...,xn]•p[x1,...,xn])⌝

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 precisely:

⌜([], "∃ vs[x1,...,xn]•p[x1,...,xn]"

If not, the function fails.

See Also save_consistency_thm to save the result in a conventional manner.

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
46007 Specification of ?0 is not of the form: ‘Consistent (λ vs[x1,...,xn]•p[x1,...,xn]) ⊢ ...
46009 ?0 is not a constant, or a constant applied to some arguments
7.1. Constant Specification

SML

\[\text{val \ save\_consistency\_thm : TERM \rightarrow \ THM \rightarrow \ THM;}\]

**Description**  
`save\_consistency\_thm const thm` expects `thm` to be of the form

\[\Gamma \vdash \text{Consistent} (\lambda \ vs[x1,\ldots,xn]\bullet p[x1,\ldots,xn])\]

where `vs` forms a varstruct from the simple variables `x_i`. The predicate proven consistent is then expected to be the same as predicate in the `const\_spec` definition of `const`. If so, the theorem will be saved under the keys `c1\_consistent`, ..., `cn\_consistent`, where the `c_i` are the names of all the constants defined in the definition in question. The `const` may either be a constant, or a constant applied to a list of arguments.

Ideally there should be no assumptions in the theorem, but the function caters for their presence.

**See Also**  
`push\_consistency\_goal` to set a goal suitable for saving with this theorem.

**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
- **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
- **46008** `?0` is not of the form: `⌜\text{Consistent} (\lambda \ vs[x1,\ldots,xn]\bullet p[x1,\ldots,xn])\⌝`
- **46009** `?0` is not a constant, or a constant applied to some arguments
- **46011** `?0` does not match definition of `?1`
- **46012** `?0` is not of the form: `⌜\text{ConstSpec} p c\⌝`

If even one key causes a failure, the theorem will not be saved at all (the behaviour of `list\_save\_thm`).
7.2 Automatic Existence Proofs

SML

signature ExistenceProofs = sig

Description This is the signature of a structure supporting automatic existence proofs.

SML

val basic_prove_∃_conv : CONV;

Description This conversion uses its internal mechanisms, and material stored in the current proof context, to attempt to prove a “simpler” term equal to its argument, which must be of the form \( \exists \ldots \). It will simplify by:

- Converting paired existential quantifiers into simple ones (see all_∃_uncurry_conv).
- Removing existential and universal quantifiers not used in the body of their quantification.
- Distributing existential quantifiers over \( \land \) and \( \lor \) as far as it is able (see simple_∃_∧_conv and simple_∃_∨_conv).
- Eliminating an existential quantifier that is equated to a term within the body of the quantification. Implicit equations (e.g. an implicit \( \Leftrightarrow T \) or by \( \neg \)) are also handled (see simple_∃_equation_conv for caveats).
- “Pulling out” universal quantifiers through existential quantifiers if the existential is a function applied in all its instances to the universally quantified variables (see simple_∃_∀_conv1), or variable structures formed by data constructors accessed by current_ad_∃_vs_thms in the current proof context. Redistribution of universals and data constructions may be done to allow this simplification to apply (see ∀_∧_conv).
- Uncurrying existential quantifiers that are curried functions.
- Simplifying existential quantifiers that match the pattern of use of recursive functions held by current_ad_∃_cd_thms in the current proof context.
- Traversing subterms reached from the outside of the term through (perhaps paired) existential and universal quantifiers, and \( \land \) or \( \lor \) operators.

It will repeat its simplification attempts until it can go no further.

\[
\vdash (\exists \text{decls}\bullet \text{pred}) \Leftrightarrow \text{simpler}
\]

Uses To implement basic_prove_∃_rule and basic_prove_∃_tac (q.v), as an appropriate argument for set_cs_∃_conv, and as a stand-alone tool.

Errors
58001 ?0 is not of the form: \( \exists \text{decls}\bullet \text{pred} \)
58002 Failed to simplify ?0

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
7.2. Automatic Existence Proofs

SML
val basic_prove_∃_rule : TERM -> THM;

Description This will attempt to simplify its argument to $T$, by using basic_prove_∃_conv (q.v.). If it succeeds, it returns the theorem with $\Leftrightarrow T$ stripped off.

Rule

$\vdash \exists \mathit{decls} \bullet \mathit{pred}$

$\textit{basic}\_\textit{prove}\_\exists\_\textit{rule}$

$\exists \mathit{decls} \bullet \mathit{pred}^\gamma$

See Also basic_prove_∃_conv, basic_prove_∃_tac

Errors

58001 ?0 is not of the form: $\exists \mathit{decls} \bullet \mathit{pred}^\gamma$
58002 Failed to simplify ?0
58003 Failed to prove ?0

We distinguish between no simplification, and failing to prove a term, to indicate whether basic_prove_∃_conv may be of initial use in a manual proof of the term.

SML
val basic_prove_∃_tac : TACTIC;

Description This will attempt to prove its goal, by using basic_prove_∃_conv (q.v.). If basic_prove_∃_conv fails, or only partially succeeds, then the tactic will fail.

Tactic

$\{ \Gamma \} \exists \mathit{decls} \bullet \mathit{pred}$

$\textit{basic}\_\textit{prove}\_\exists\_\textit{tac}$

See Also basic_prove_∃_conv, basic_prove_∃_rule

Errors

58004 Goal is not of the form: $\{ \Gamma \} \exists \mathit{decls} \bullet \mathit{pred}$
58005 Failed to simplify goal
58006 Failed to prove goal

We distinguish between no simplification, and failing to prove a term, to indicate whether basic_prove_∃_conv may be of initial use in a manual proof of the term.
val evaluate : THM -> (TERM list * int list * TYPE * (TERM list) list * THM);

Description This gives the sophisticated user direct access to the standard internal mechanism set by \( \text{pp' set eval ad \_ cd \_ thm} \). Its input theorem is of the “standard” form of a primitive recursion theorem or pattern matching theorem:

\[
\forall f n1 f n2 \ldots \exists f \bullet (\forall a11 \ldots \bullet f (dc1[a11, \ldots]) = fn1 (patt11[f,a11, \ldots]) \ldots) \land \\
(\forall a21 \ldots \bullet f (dc2[a21, \ldots]) = fn1 (patt21[f,a21, \ldots]) \ldots) \land \\
\ldots
\]

Compare this with:

\[
\text{prim rec thm} = \vdash \forall z s \exists 1 f \bullet (f 0 = z) \land (\forall n \bullet f (n + 1) = s (f n) n)
\]

The \( fn_i \) (in the example: \( z \) and \( s \)) are the new functions which express the behaviour of each data constructor. \( f \) is the function defined by clausal definitions, note that unique existence is not necessary. The \( a_{ij} \) (\( n \) in the second conjunct) are the free variables of the data construction. The \( dc_i \) (\( n + 1 \) and \( 0 \)) are data constructions, as in the example given these do not need to be the original data constructors of a type, just applications of constants and variables. The \( patt_{ij} \) (\( f n, n \)) are the arguments to the new functions. They may involve uses of the clausally defined function.

The result of this function is: the list of data constructions, a list of free variable counts in each data construction, the type of \( f \), a list of free variables in each data construction, and a theorem of the form:

\[
\vdash \forall \text{pred1 } \ldots \\
(\exists f \bullet (\forall a11 \ldots \bullet \text{pred1 } (dc1[\ldots]) (patt1[\ldots]) \ldots) \ldots) \Leftrightarrow \\
(\forall z11 \ldots x1n1 \exists y1 \bullet \text{pred1 } y11 x11 \ldots) \ldots \lor \\
(\text{pp' TS}(\exists f \bullet (\forall a11 \ldots \bullet \text{pred1 } (dc1[\ldots]) (patt1[\ldots]) \ldots) \ldots))
\]

This is in a convenient form for proving the existence of clausally defined functions, in particular recursive functions within \textit{basic \_ prove \_ \_ conv}. The \( \text{pred}_i, x_{ij} \) and \( y_{ij} \) are generated by \textit{gen \_ vars}. \textit{pp' TS} is an identity function, with defining theorem in theory “misc” of key “pp' ts \_ def”.

Errors
58007 ?0 is not of the form: ‘\( \vdash \forall \ldots \exists f \bullet \ldots \)’ or ‘\( \vdash \forall \ldots \exists 1 f \bullet \ldots \)’
58008 ?0 has a conjunct not of the form: ‘\( \forall \ldots \bullet f \ dc = \ldots \)’
58009 ?0 has a conjunct not of the form: ‘\( \forall \ldots \bullet f \ dc = \ldots \)’
where \( f \) is the function whose existence is to be proven
58021 Failed to prove new theorem based on ?0
58023 ?0 has a conjunct not of the form: ‘\( \forall \ldots \bullet f \ dc = P \ldots \)’
58024 ?0 has the form: ‘\( \vdash \forall \ldots fni \ldots \bullet B \)’ where fni does not appear in any of the expected positions in \( B \)
7.2. Automatic Existence Proofs

SML

```sml
val simple_existent_eqn : CONV;
```

**Description**  This conversion eliminates a simple existential quantifier that is equated to a term within the body of the quantification. The term equated with must not contain the existential quantifier, nor may it have free variables that are bound in the body of the existential quantifier, other than in an outer existential. The equation must also be “reached” only through existential and universal quantifications, \( \land \). Implicit equations (e.g. \( x \) being an implicit \( x \iff T \) or by \( \neg y \) giving an implicit \( y \iff F \) are also handled.

**Conversion**

\[
\vdash (\exists f \cdot P [(f = \text{tm}), f]) \iff P [T, \text{tm}]
\]

Some simplification will also take place if any of the following apply where \( T \) replaced the selected equation:

\[
[\land\_\text{rewrite\_thm}, \exists\_\text{rewrite\_thm}, \forall\_\text{rewrite\_thm},
\text{eq\_rewrite\_thm}, \Rightarrow\_\text{rewrite\_thm}, \neg\_\text{rewrite\_thm}]
\]

**Example**

\[
simple_existent_eqn \implies \exists f \cdot f = x \iff \vdash (\exists f \cdot f = x) \iff T
\]

\[
simple_existent_eqn \implies \exists f \cdot x = f \iff \vdash (\exists f \cdot x = f) \iff T
\]

\[
simple_existent_eqn \implies \exists f \cdot (f = x) \land P f \iff \vdash (\exists f \cdot (f = x) \land P f) \iff P x
\]

\[
simple_existent_eqn \implies \exists f \cdot g \cdot (f = g) \land P f \iff \vdash (\exists f \cdot g \cdot (f = g) \land P f) \iff \exists g \cdot P g
\]

\[
simple_existent_eqn \implies \exists f \cdot f \land P f \iff \vdash (\exists f \cdot f \land P f) \iff P T
\]

\[
simple_existent_eqn \implies \exists f \cdot \neg f \land P f \iff \vdash (\exists f \cdot \neg f \land P f) \iff P F
\]

**See Also**  basic_prove_exists_conv for a more powerful existence prover.

**Errors**

3034  ?0 is not of form: \( \exists \text{ var } \cdot \text{ body} \)

58013  Cannot find an equation in ?0 to simplify

58022  Could not prove ?0 by selected witness ?1
val simple\_\exists\_\land\_conv: CONV;

**Description** This conversion will push simple existential quantifiers into a term body that consists of nested conjunctions when not all the conjuncts have all of the quantified variables free in them. It will prove the input equal to a term formed by conjoining the list of existentially quantified groupings of conjuncts, grouped by possession of a quantified variable free in each of the group. It may not necessarily form the “most pushed in” form with each conjunct minimally bound, but will do some pushing in if any is available.

**Conversion**

\[
(\exists x_1 \ldots x_n \ldots \land P[y_1 \ldots y_m] \land \ldots) 
\iff
\text{simple}_\exists \land \text{conv}
\]

\[
\text{simple}_\exists \land \text{conv}
\vdash \exists x_1 \ldots x_n \ldots \land P[y_1 \ldots y_m] \land \ldots
\]

where the \( y_i \) are a subset of the \( x_j \).

**Example**

\[
\text{simple}_\exists \land \text{conv}
\vdash \exists x \bullet P[x] \land Q \iff
\vdash (\exists x \bullet P[x]) \land Q
\]

\[
\text{simple}_\exists \land \text{conv}
\vdash \exists x \bullet P \land Q[x] \iff
\vdash (\exists x \bullet P[x]) \land Q[x]
\]

\[
\text{simple}_\exists \land \text{conv}
\vdash \exists x y \bullet P[x] \land Q[y] \iff
\vdash (\exists x y \bullet P[x] \land Q[y]) \iff (\exists x \bullet Q[y]) \land (\exists x \bullet P[x])
\]

\[
\text{simple}_\exists \land \text{conv}
\vdash \exists x y z \bullet P[x, z] \land Q[y, z] \iff
\vdash (\exists x y z \bullet P[x, z] \land Q[y, z]) \iff
\exists z \bullet (\exists y \bullet Q[y, z]) \land (\exists x \bullet P[x, z])
\]

The original conjunctive structure and ordering is lost.

**See Also** \( \exists\_\text{uncurry\_conv} \) to uncurry any paired \( \exists \)-abstractions. \textit{simple}_\exists \lor \_conv.

**Errors**

58012 ?0 is not of the form: \( \exists x_1 \ldots \bullet a \land b \)

58014 All conjuncts of ?0 have all existentially quantified variables free within them.
7.2. Automatic Existence Proofs

SML

val simple_∃∨.conv: CONV;

**Description** This conversion will push simple existential quantifiers into a term body that consists of nested disjunctions. It will prove the input equal to a term formed by disjoining the list of disjuncts, existentially quantified by any of the original quantifiers that are free in that disjunct.

**Conversion**

\[
\vdash (\exists x_1 \ldots x_n \cdot P[y_1 \ldots y_m] \lor Q[z_1 \ldots z_o]) \\
\iff \\
(\exists y_1 \ldots y_m \cdot P[y_1 \ldots y_m]) \lor \\
(\exists z_1 \ldots z_o \cdot P[z_1 \ldots z_o])
\]

where \(\{y_1 \ldots y_m\} \subseteq \{x_1 \ldots x_n\}\), and \(\{z_1 \ldots z_o\} \subseteq \{x_1 \ldots x_n\}\). The conversion will actually distribute over an arbitrary disjunctive structure, though the result is “flattened”.

**Example**

\[
\vdash (\exists x y \cdot f x \lor g y) \\
\iff (\exists x \cdot f x) \lor (\exists y \cdot g y)
\]

\[
\vdash (\exists x y \cdot f x \lor g y \lor h x y \lor i) \\
\iff \\
(\exists x \cdot f x) \\
\lor (\exists y \cdot g y) \\
\lor (\exists x \cdot h x y) \\
\lor i
\]

**See Also** \(∃\_uncurry\_conv\) to uncurry any paired \(∃\)-abstractions. \(simple_∃\_∧\_conv\).

**Errors**

58010 ?0 is not of the form: \(\forall x_1 \ldots \cdot a \lor b\)

58015 Unexpected feature of ?0 caused failure
val \_\_\_\_conv: CONV;

**Description**  This conversion will push universal quantifiers into a term body that consists of nested conjunctions. It will prove the input equal to a term formed by conjoining the list of conjuncts, universally quantified by any of the original quantifiers that are free in that conjunct.

**Conversion**

\[ \vdash (\forall x_1 \ldots x_n \cdot P[y_1 \ldots y_m] \land Q[z_1 \ldots z_o]) \iff (\forall y_1 \ldots y_m \cdot P[y_1 \ldots y_m]) \land (\forall z_1 \ldots z_o \cdot P[z_1 \ldots z_o]) \]

where \( \{y_1 \ldots y_m\} \subseteq \{x_1 \ldots x_n\} \), and \( \{z_1 \ldots z_o\} \subseteq \{x_1 \ldots x_n\} \). The conversion will actually distribute over an arbitrary conjunctive structure, though the result is “flattened”. The conversion will work with paired universal quantifiers, and will still return a result even if it only acts as all\_\_\_\_uncurry\_\_conv (q.v.), and does not redistribute the quantifiers any further.

**Example**

\[ \forall\_\_\_\_conv \forall x \ y \cdot f \ x \land g \ y \downarrow = \]

\[ \vdash (\forall x \ y \cdot f \ x \land g \ y) \iff (\forall x \cdot f \ x) \land (\forall y \cdot g \ y) \]

\[ \forall\_\_\_\_conv \forall x \ y \cdot f \ x \land g \ y \land h \ x \ y \land i \downarrow = \]

\[ \vdash (\forall x \ y \cdot f \ x \land g \ y \land h \ x \ y \land i) \iff (\forall x \cdot f \ x) \land (\forall y \cdot g \ y) \land (\forall x \ y \cdot h \ x \ y) \land i \]

**See Also**  \_\_\_\_uncurry\_\_conv to uncurry any paired \_\_\_\_abstractions.

**Errors**  \_\_\_\_uncurry\_\_conv is not of the form: \( \forall x_1 \ldots \cdot a \land b \downarrow \)
7.3 Product Types

SML

\texttt{signature ProductTypes = sig}

\textbf{Description} This is the signature of a structure supporting the introduction of HOL product types.

SML

\texttt{val cached_labelled_product_rule : int -> THM;}

\textbf{Description} \texttt{cached_labelled_product_rule n} proves a theorem about an n-tuple, where \( n \) is some positive integer. This states that if some type is isomorphic to an n-tuple (an iterated binary product) then a constructor, and projection functions exist for this type. The theorem will be stored in the current cache theory (see \texttt{get_cache_theories}) under the key “\texttt{nCachedLabelledProduct}”, unless \( n = 1 \), when it is “built-it”. However, if the theorem desired is already in a cache theory in scope, then it just returns the stored existence theorem.

For storing the theorem current cache theory will be found by \texttt{force_get_cache_theory}. If no writable cache theory is in scope this will attempt to use \texttt{new_parent} to bring one in scope, and failing that will declare the current theory to be a cache theory.

\textbf{Example}

\begin{verbatim}
> cached_labelled_product_rule 2;
\end{verbatim}

\begin{verbatim}
\( \exists \text{abs \ rep} \bullet (\forall a \bullet \text{abs}(\text{rep} a) = a) \land (\forall r \bullet \text{rep}(\text{abs} r) = r)) \Rightarrow \\
\exists \text{2Tuple} \bullet \\
\exists \text{Lab2_1} \text{Lab2_2} \bullet \\
\forall t \bullet (\forall (x1:1) (x2:2)\bullet \\
\text{Lab2_1}(\text{2Tuple} x1 x2) = x1 \land \\
\text{Lab2_2}(\text{2Tuple} x1 x2) = x2) \land \\
(\text{2Tuple}(\text{Lab2_1} t) (\text{Lab2_2} t) = t)
\end{verbatim}

\textbf{See Also} \texttt{cached_unlabelled_product_rule}

\textbf{Errors}

59001 Argument ?.0 should have been positive
59002 There is no in scope, writable, cache theory (current theory is ?.0)
59003 Failed to prove ?0th labelled cached theorem
59008 Unable to bring a writable cache theory into scope
or create a new cache theory (current theory is ?.0)
59017 Current theory (?0) does not have theory basic_hol as an ancestor

Also as the appropriate failures of \texttt{save_thm}. Failure 59003 suggests a corrupted cached theorem.
val cached_unlabelled_product_rule : int -> THM;

Description  cached_unlabelled_product_rule n proves a theorem about an n-tuple, where n is some positive integer. This states that if some type is isomorphic to an n-tuple (an iterated binary product) then a constructor with appropriate properties exists for this type. The theorem will be stored in the current cache theory (see get_cache_theories) under the key “nCachedUnlabelledProduct”, unless n = 1, when it is “built-it”. However, if the theorem desired is already in a cache theory in scope, then it just returns the stored existence theorem.

For storing the theorem the current cache theory will be found by force_get_cache_theory. If no writable cache theory is in scope this will attempt to use new_parent to bring one in scope, and failing that will declare the current theory to be a cache theory.

Example  > cached_unlabelled_product_rule 2;

\[ \exists \text{abs rep} \bullet (\forall a \bullet \text{abs(rep a)} = a) \land (\forall r \bullet \text{rep(abs r)} = r) \]
\[ \Rightarrow \exists 2\text{Tuple} \bullet \\
\quad (\forall x_1 x_2 \bullet \forall y_1 y_2 \bullet \\
\quad (2\text{Tuple} x_1 x_2 = 2\text{Tuple} y_1 y_2) \iff (x_1 = y_1) \land (x_2 = y_2)) \]
\[ \land \\
\quad (\forall t \bullet \exists x_1 x_2 \bullet t = 2\text{Tuple} x_1 x_2) \]

Errors  59001 Argument ?0 should have been positive
59002 There is no in scope, writable, cache theory (current theory is ?0)
59004 Failed to prove ?0th unlabelled cached theorem
59008 Unable to bring a writable cache theory into scope
or create a new cache theory (current theory is ?0)
59017 Current theory (?0) does not have theory basic_hol as an ancestor

See Also  cached_labelled_product_rule Also as the appropriate failures of save_thm. Failure 59004 suggests a corrupted cached theorem, but only for certain indicates an otherwise uncaught error.
7.3. Product Types

SML

val HOL_lab_prod_recogniser : string * string * string * Lex.INPUT list * string
–> THM;

Description This function is called by the reader writer and should not be called by a user
directly. The first and second arguments are the start symbol for the recogniser and the lan-
guage name. The third argument is the name of the type, which appears as the argument. to
HOLABPROD. The fourth argument is the HOL term represented in a form understood by
the lexical analyser. The last argument is the terminating symbol, and should always be a “■”.
The name of the type and the parsed input is passed to labelled_product_spec (q.v.) which will
attempt the requested specification. The keys under which the result will be saved are just the
names of the variables concerned.

This may also save a theorem concerning tuples to the current cache theory (see get_cache_theory).

Uses As an interface between the HOL reader/writer and labelled_product_spec - it is not in-
tended for direct use. It is used to process text of the form, e.g.,

Example Labelled Product Specification

HOLABPROD TNAME
dcl dcl;
dcl dcl; de

Errors

59000 Input not of right form

Errors from labelled_product_spec may also occur: the messages area of origin will not be changed.
val labelled_product_spec: {tyname:string, tykey:string, conname:string, constkeys:string list, labels:(string*TYPE)list, tyvars:(TYPE list)OPT} -> THM;

Description  This function introduces a labelled product type, and its constructor and projection functions. Its argument’s fields are:

tyname  The name of the new type.

tykey  The key under which the defining theorem for the type will be saved.

conname  The name of the single constructor of the new type. It will be a curried function, from each of the label types, in turn, to the new type.

constkeys  The keys under which the defining theorem for the constructor conname and projection functions will be saved.

labels  The list of the names and types of the labelled fields in the new type. A projection function will be defined for each label, a single defining theorem for all will be saved under the keys constkeys. It is this defining theorem that is the result of the function call.

tyvars  If this field is Nil then the type variables in the new type will be ordered as their occurrence in labels, otherwise as stated by this parameter.

If the appropriate tuple theorem is not already in scope a call of cached_labelled_product_rule (length labels) will also be made as a side effect.

Example

|:> labelled_product_spec {tyname="PAIR", tykey="PAIR", conname="Comma", constkeys="Comma","First","Second","pair_ops_def"], labels=[("First","f"),("Second","s")], tyvars=Valuel"f","s"};

will cause  \( \exists f : (s, f) \text{PAIR} \to (f \times s) \bullet \text{TypeDefn} (\lambda x \bullet T) f \)

to be saved under the key "PAIR", and the following

\[ \forall t \bullet \forall (x1,f) (x2,s) \bullet \]
\[ \text{First(Comma x1 x2)} = x1 \land \]
\[ \text{Second(Comma x1 x2)} = x2 \land \]
\[ \text{Comma(First t) (Second t)} = t \]

under keys "Comma", "First", "Second" and "pair_ops_def", and returned as the result

Errors

59010  Must be at least one label in list
59014  Failed to introduce type and operators for ?0
59015  Some labels are given types containing type variables not found in supplied list of type variables, being: ?0
59018  May not use key ?0 for both type and constructor definition

Also as the appropriate errors of cached_labelled_product_rule, new_type_defn and new_specification. Error 59014 suggests a corrupted cache theorem.
7.3. Product Types

SML

| val set_cache_theories : string list -> string list; |
| val get_cache_theory : unit -> string; |
| val force_get_cache_theory : unit -> string; |
| val get_cache_theories : unit -> string list; |
| val get_valid_cache_theories : unit -> string list; |

Description These functions provide the interface to the list of cache theories. `set_cache_theories` stores an uninterpreted list of strings, that are intended to be theory names, returning the previous setting.

The current cache theory is the first string of the cache theory list which is a theory name that is in scope and writable (not locked, an ancestor, or below sealing by `pp/seal_hierarchy`). This is the cache theory that should be written to as a cache. This is accessed by `get_cache_theory` which will fail if there is no such theory.

`force_get_cache_theory` will search for the current cache theory as `get_cache_theory`. If that fails, then it will attempt to create one by trying `new_parent` on each of the writable cache theories that are out of scope. If that fails, then it will attempt to create a new cache theory named `cache'N` (where `N` is a sequence number). If a new theory is created, its parent will be the theory named at the head of the list of cache theories and it will become a parent of the current theory.

`get_cache_theories` returns the uninterpreted list of strings last set as the cache theories by `set_cache_theories`.

The current cache theories are the names in the cache theory list of in-scope theories (that need not be writable): this function may return the empty list. This list is accessed by `get_valid_cache_theories`.

The scope and existence of the cache theories is determined at the point of applying the function names to ().

These functions support a method of working using cache theories, where only a subset of the ancestors of the current theory are searched for particular portions of contents, and these things are only stored in the current cache theory. This will speed the search for these items by such functions as `cached_labelled_product_rule`.

Errors

| 59002 | There is no in scope, writable, cache theory (current theory is ?0) |
| 59008 | Unable to bring a writable cache theory into scope or create a new cache theory (current theory is ?0) |
| 59020 | A new cache theory ?0 has been created |
val unlabelled_product_spec : \{tyname:string, tykey:string, conname:string, conkeys:string list, tyi:TYPE list, tyvars:(TYPE list)OPT\} \rightarrow THM;

Description  This function introduces an unlabelled product type. Its argument's fields are:

tyname  The name of the new type.
tykey  The key under which the defining theorem for the type will be saved.
conname  The name of the single constructor of the new type. It will be a curried function, from each of the types tyi, in turn, to the new type.
conkeys  The keys under which the defining theorem for the constructor conname will be saved.
tyi  The list of the types of the fields in the new type.
tyvars  If this field is Nil then the type variables in the new type will be ordered as their occurrence in labels, otherwise as stated by this parameter.

If the appropriate tuple theorem is not already in scope a call of cached_unlabelled_product_rule (length tyi) will also be made as a side effect. This is as part of the speeding up of introducing the new type.

Example

\[
\begin{align*}
\text{unlabelled_product_spec } \{\text{tyname=":UPAIR"}, \text{tykey=":UPAIR"}, \text{conname=":UComma"}, \\
\text{conkeys=} ["UComma","ucomma_def"], \text{tyi=}[⌜:\'f\', \'s\⌝], \text{tyvars=}\text{Nil}\};
\end{align*}
\]

will cause

\[
\begin{align*}
\exists f : (\'f, \'s) \text{ UPAIR} \rightarrow (\'f \times \'s) \bullet \text{TypeDefn } (\lambda x \bullet T) f \\
\rightarrow (\forall (x1:\'f) (x2:\'s)\bullet \forall y1 y2\bullet \\
(UComma x1 x2 = UComma y1 y2) \Rightarrow (x1 = y1) \land (x2 = y2)) \\
\land (\forall t \bullet \exists x1 x2 \bullet t = UComma x1 x2)
\end{align*}
\]

to be saved under the keys "UComma" and "ucomma_def", and returned as the result.

See Also  labelled_product_spec

Errors

59011  There must be at least one type in list
59014  Failed to introduce type and operators for ?0
59016  The supplied types contain type variables not in the supplied list of type variables, being: ?0
59018  May not use key ?0 for both type and constructor definition.

Also as the appropriate errors of cached_unlabelled_product_rule and new_type_defn. Error 59014 suggests a corrupted cache theory.
8.1 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, ORELSE_CAN, REPEAT_CAN, FIRST_CAN, EVERY_CAN as combinators, fail_can and id_can as building blocks for the combinators.

SML

|val ALL_SIMPLE_V_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.
val all_simple_\(\forall\)_elim : THM \rightarrow THM;

**Description**  Specialises all the simple universally quantified variables in a theorem:

\[
\begin{align*}
\Gamma \vdash \forall x_1 \ldots x_n \bullet t[x_1, \ldots, x_n] & \quad \text{all_simple_\(\forall\)_elim} \\
\Gamma \vdash t[x_1', \ldots, x_n'] &
\end{align*}
\]

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.

val ALL_SIMPLE_\(\exists\) C : CONV \rightarrow CONV;

**Description**  This conversional applies its conversion argument to the body of a repeated simple existential quantification.

**Errors**  As the failure of the conversion argument.

val all_simple_\(\beta\)_conv : 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.

\[
\begin{align*}
\vdash t = t' & \quad \text{all_simple_\(\beta\)_conv} \\
\Gamma \vdash t' &
\end{align*}
\]

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 \(\beta\)_conv.

**Errors**  \(\text{7020 } ?0 \text{ contains no } \beta-\text{redexes}\)

val all_simple_\(\beta\)_rule : THM \rightarrow THM;

**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.

\[
\begin{align*}
\Gamma \vdash t & \quad \text{all_simple_\(\beta\)_rule} \\
\Gamma \vdash t' &
\end{align*}
\]

t' is \(t\) with all \(\beta\)-redexes reduced.

**Errors**  \(\text{7020 } ?0 \text{ contains no } \beta-\text{redexes}\)
8.1. General Inference Rules

**SML**
\[
\text{val } \text{ALL}_\land \text{C } : \text{CONV } \rightarrow \text{CONV}; \\
\text{val } \text{ALL}_\lor \text{C } : \text{CONV } \rightarrow \text{CONV};
\]

**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:
\[
\vdash \forall \ t \cdot (T \land t \iff t) \land (t \land T \iff t) \land \neg (F \land t) \land \neg (t \land F) \land (t \land t \iff t)
\]

**Errors** As the failure of the conversion argument.

**SML**
\[
\text{val } \text{all}_\Rightarrow \text{intro } : \text{THM } \rightarrow \text{THM};
\]

**Description** Discharge all members of assumption list using \( \Rightarrow \) intro.

<table>
<thead>
<tr>
<th>Rule</th>
<th>{t_1, ..., t_n} \vdash t</th>
<th>\vdash t_1 \Rightarrow ... \Rightarrow t_n \Rightarrow t</th>
<th>\text{all}_\Rightarrow \text{intro}</th>
</tr>
</thead>
</table>

**SML**
\[
\text{val } \text{all}_\forall \text{arb elim } : \text{THM } \rightarrow \text{THM};
\]

**Description** Specialise all the quantifiers of a possibly universally quantified theorem with a machine generated variables or variable structures.

<table>
<thead>
<tr>
<th>Rule</th>
<th>( \Gamma \vdash \forall \ vs_1[x_1,y_1,...] \ vs_2[x_2,y_2,...] ...\bullet p[x_1,y_1,...,x_2,y_2,...] )</th>
<th>( \Gamma \vdash p[x_1',y_1',...,x_2',y_2',...] )</th>
<th>\text{\textbackslash_\forall \text{arb elim} }</th>
</tr>
</thead>
</table>

where \( x_i', y_i', \) etc, are not variables (free or bound) in \( p \) or \( \Gamma \), created by \text{gen}vars\text{(q,v)}.

**See Also** \text{all}_\forall \text{elim}

**SML**
\[
\text{val } \text{all}_\forall \text{elim } : \text{THM } \rightarrow \text{THM};
\]

**Description** Specialises all the outer universal quantifications in a theorem:

<table>
<thead>
<tr>
<th>Rule</th>
<th>( \Gamma \vdash \forall \ x_1 \ldots \ x_n \bullet t[x_1, ..., x_n] )</th>
<th>( \Gamma \vdash t[x_1', ..., x_n'] )</th>
<th>\text{all}_\forall \text{elim}</th>
</tr>
</thead>
</table>

where \( x_1', \ldots, x_n' \) are renamed from \( x_1, \ldots, x_n \) as necessary to avoid name clashes with free variables in the assumption list.

**See Also** \text{all}_\forall \text{arb elim} which is faster, though the results are slightly opaque. \text{list}_\forall \text{elim}.
| **val all\_\forall\_intro : THM \rightarrow THM;** |
| **Description** | Generalises all the free variables (other than those in the assumption list) in a theorem:
| **Rule** | \[ \Gamma \vdash t \quad \Gamma \vdash \forall x1 \ldots xn \cdot t \quad all\_\forall\_intro \]
| where | \[ x1,\ldots,xn \] are all the free variables of \( t \). The function introduces variables in their order of occurrence, so:
| **Example** | \[ all\_\forall\_intro (\vdash a \lor b) = \vdash \forall a b \cdot a \lor b \]

| **val all\_\forall\_uncurry\_conv : CONV;** |
| **Description** | Apply \( all\_\forall\_uncurry\_conv \) (q.v) to the outer universal quantifications of a term, flattening those binders.
| **Conversion** | \[ \Gamma \vdash (\forall vs1[x,y,...] vs2[x,y,...] ... \cdot f[x1,y1,\ldots,x2,y2,\ldots]) \equiv (\forall x1 y1 ... x2 y2 ... \cdot f[x1,y1,\ldots,x2,y2,\ldots]) \quad all\_\forall\_uncurry\_conv \]
| where | \( vs_i[x_i,y_i,\ldots] \) are variable structures at least one of which must not be a simple variable, built from variables \( x_i,y_i,\ldots \).
| **Errors** | 27041 0 is not of the form: \( \forall \ldots (x,y) \ldots f \)

| **val all\_\exists\_uncurry\_conv : CONV;** |
| **Description** | Apply \( all\_\exists\_uncurry\_conv \) (q.v) to the outer existential quantifications of a term, flattening those binders.
| **Conversion** | \[ \Gamma \vdash (\exists vs1[x,y,...] vs2[x,y,...] ... \cdot f[x1,y1,\ldots,x2,y2,\ldots]) \equiv (\exists x1 y1 ... x2 y2 ... \cdot f[x1,y1,\ldots,x2,y2,\ldots]) \quad all\_\exists\_uncurry\_conv \]
| where | \( vs[x,y,\ldots] \) are variable structures with variables \( x,y,\ldots \), 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: \( \exists \ldots (x,y) \ldots f \)
8.1. General Inference Rules

val all_β_conv : CONV;

**Description**  A conversion to eliminate all instances of β redexes, including paired abstraction redexes, in a term, regardless of nesting, or even that the β redex was created as the result of an earlier reduction in the conversion’s evaluation.

\[ \Gamma \vdash t = t' \]

Rule

\[ all_β_conv \]

\[ t' \]

\[ t' \] is \( t \) with all β 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 β_conv.

**See Also**  all_simple_β_conv which only handles simple β-redexes, but does a faster traversal if that is all that is required. all_β_rule.

**Errors**  27049 ?0 contains no β-redexes

val all_β_rule : THM -> THM;

**Description**  Eliminate all instances of β redexes, including paired abstraction redexes, in the conclusion of a theorem, regardless of nesting, or even that the β redex was created as the result of an earlier reduction in the rule’s evaluation.

\[ \Gamma \vdash t \]

Rule

\[ \Gamma \vdash t' \]

\[ all_β_rule \]

\[ t' \]

\[ t' \] is \( t \) with all β-redexes reduced.

**See Also**  all_β_conv for the conversion. all_simple_β_rule which only handles simple β-redexes, but does a faster traversal if that is all that is required.

**Errors**  27049 ?0 contains no β-redexes

val AND_OR_C : (CONV * CONV) -> 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).
val app_arg_rule : TERM -> THM -> THM;

Description  Apply both sides of an equational theorem to an argument.

\[ \Gamma \vdash f = g \quad \Rightarrow \quad \Gamma \vdash f \; x = g \; x \]

Errors

6020  ?0 is not of the form: ' \( \Gamma \vdash t1 = t2 \)'
7025  Sides of equation may not be applied to term

val APP_C : (CONV * CONV) -> CONV;

Description  Apply one conversion to the operator of a combination, and a second to the operand.

\[ \vdash f \; a = f' \; a' \]

where \( c1 \) gives \( \vdash f = f' \), and \( c2 \) gives \( \vdash a = a' \).

Errors

3010  ?0 is not of form: ' \( t1 \; t2 \)'
7110  Results of conversions, ?0 and ?1, ill-formed or cannot be combined

Also as the failure of the conversions.

val app_fun_rule : TERM -> THM -> THM ;

Description  Apply a function to both sides of an equational theorem.

\[ \Gamma \vdash a = b \quad \Rightarrow \quad \Gamma \vdash f \; a = f \; b \]

Errors

6020  ?0 is not of the form: ' \( \Gamma \vdash t1 = t2 \)'
7024  ?0 may not be applied to each side of equation

val app_if_conv : CONV;

Description  Move a function application into a conditional.

\[ \vdash f (if \; a \; then \; b \; else \; c) = \; \quad \Rightarrow \quad (if \; a \; then \; f \; b \; else \; f \; c) \]

Errors

7098  ?0 is not of the form: ' \( f (if \; a \; then \; b \; else \; c) \)'

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
### 8.1. General Inference Rules

#### SML
code
```sml
val asm_elim : TERM -> THM -> THM -> THM;
```

description
Eliminate an assumption with reference to contradictory assumption lists.

table
<table>
<thead>
<tr>
<th>Rule</th>
<th>asm_elim</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \Gamma_1, a' \vdash t; \Gamma_2, \neg a'' \vdash t' ]</td>
<td>[ \Gamma_1 \cup \Gamma_2 \vdash t ] [ \vdash a ]</td>
</tr>
</tbody>
</table>

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 \( \vdash \) BOOL
3029 ?0 and ?1 are not of the form: \[ \vdash \Gamma_1, a a' \vdash t \] and \[ \vdash \Gamma_2, \neg a a a' \vdash t a' \] where \( \vdash t \) and \( \vdash t a \) are \( \alpha \)-convertible

#### SML
code
```sml
val asm_inst_term_rule : (TERM * TERM) list -> THM -> THM;
```

description
Parallel instantiation of term variables within a theorem’s conclusion and assumptions to some other values.

table
<table>
<thead>
<tr>
<th>Rule</th>
<th>asm_inst_term_rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \Gamma \vdash t[x_1, ..., x_n] ]</td>
<td>[ \Gamma' \vdash t[t_1, ..., t_n] ] [ \vdash [t_1, ..., t_n] ]</td>
</tr>
</tbody>
</table>

\( \vdash [t_1, ..., t_n] \) will instantiate each term variable in \( t \) with its associated term.

see also
inst_term_rule

errors
3007 ?0 is not a term variable
6027 Types of element (?0, ?1) in term association list differ

#### SML
code
```sml
val asm_inst_type_rule : (TYPE * TYPE) list -> THM -> THM;
```

description
Parallel instantiation of some of the type variables of both the conclusion and assumptions of a theorem.

table
<table>
<thead>
<tr>
<th>Rule</th>
<th>asm_inst_type_rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \Gamma \vdash t[v_1, ..., v_n] ]</td>
<td>[ \Gamma' \vdash t[\sigma_1, ..., \sigma_n] ] [ \vdash [\sigma_1, v_1, ..., \sigma_n, v_n] ]</td>
</tr>
</tbody>
</table>

\( \vdash [\sigma_1, v_1, ..., \sigma_n, v_n] \) will instantiate each type variable in \( t \) with its associated type.

asm_inst_type_rule talist thm will instantiate each type variable in talist with its associated type. It will decorate free variables that would become identified with other variables by their types becoming the same and the names originally being the same. \( \alpha \)-convertible duplicate assumptions will be eliminated.

see also
inst_type_rule

errors
3019 ?0 is not a type variable

#### SML
code
```sml
val asm_intro : TERM -> THM -> THM;
```

description
Introduce a new assumption to an existing theorem.

table
<table>
<thead>
<tr>
<th>Rule</th>
<th>asm_intro</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \Gamma \vdash t_2 ]</td>
<td>[ \Gamma \cup {t_1} \vdash t_2 ] [ \vdash t_1 ]</td>
</tr>
</tbody>
</table>

errors
3031 ?0 is not of type \( \vdash \) BOOL
val asm_rule : TERM $\rightarrow$ THM;

Description  “A term is true on the assumption that it is true.”

\[
\begin{array}{c}
\text{Rule} \\
\hline
\text{asm_rule} \\
\end{array}
\]

\[
\begin{array}{c}
t \vdash t \\
\end{array}
\]

A primitive inference rule.

Errors

3031 \text{ ?}0 \text{ is not of type } \vdash \text{BOOL}^\top

val BINDER.C : CONV $\rightarrow$ CONV;

Description  Apply a conversion to the body of a binder term:

\[
\begin{array}{c}
\text{Rule} \\
\hline
\text{BINDER.C} \\
\end{array}
\]

\[
\begin{array}{c}
(B \ x \ p[x]) = (B \ x \ pa[x]) \\
\end{array}
\]

where \(c\) \(p[x]\) gives \(\vdash p[x] = pa[x]^{\top}\), and \(B\) is a binder.

Errors

27035 \text{ ?}0 \text{ is not of the form } \vdash B \ x \ p[x]^{\top}\text{ where } \vdash B^{\top}\text{ is a binder}

7104 \text{ Result of conversion, } ?0, \text{ 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 \text{ Conversion failed to cause a change}

7104 \text{ Result of conversion, } ?0, \text{ 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 \textit{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 CHAR.

\[
\begin{array}{c}
\text{Rule} \\
\hline
\text{char.conv} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\textit{mk_char}}("c")^{\top} = \\
\text{AbsChar \textit{mk_ord} "c"^{\top}}
\end{array}
\]

A primitive inference rule(axiom schemata).

See Also  \textit{mk_char}

Errors

3024 \text{ ?}0 \text{ is not a character literal}
8.1. General Inference Rules

SML

val COND_C : \langle \text{TERM} \rightarrow \text{bool} \rangle \rightarrow \text{CONV} \rightarrow \text{CONV} \rightarrow \text{CONV}\

Description \ \text{COND}_C \ \text{pred} \ \text{cnv1} \ \text{cnv2} \ \text{tm} \ \text{will be, if the term predicate} \ \text{pred} \ \text{applied to} \ \text{tm} \ \text{is true, then} \ \text{cnv1} \ \text{tm} \ \text{and otherwise the} \ \text{cnv2} \ \text{tm}.

Errors \ \text{As the failure of the predicate or either conversion.}

SML

val cond_thm : \text{THM};

Description \ \text{A convenient variant of the definition of the conditional.}

\[
\begin{align*}
\vdash & \forall \ a \ t1 \ t2 \bullet (a \ \text{then} \ t1 \ \text{else} \ t2) = \\
& (\epsilon \ x \bullet ((a \Rightarrow T) \Rightarrow x = t1) \wedge ((a \Rightarrow F) \Rightarrow x = t2))
\end{align*}
\]

SML

val contr_rule : \text{TERM} \rightarrow \text{THM} \rightarrow \text{THM};

Description \ \text{Intuitionistic contradiction rule:}

\[
\begin{array}{c}
\Gamma \vdash F \\
\hline
\Gamma \vdash t
\end{array}
\]

\[
\Gamma \vdash F
\]

\[
\vdash t
\]

\[
\text{contr_rule}
\]

Errors

\begin{align*}
7001 & \ \text{?0 is not of form:} \ '\Gamma \vdash F' \\
3031 & \ \text{?0 is not of type} \ \Gamma : \text{BOOL}
\end{align*}

SML

val conv_rule : \text{CONV} \rightarrow \text{THM} \rightarrow \text{THM};

Description \ \text{Apply a conversion to the conclusion of a theorem, and do} \ \Rightarrow \ \text{modus ponens between the original theorem and the result of the conversion}

\[
\begin{array}{c}
\Gamma 1 \vdash t \\
\hline
\Gamma 1 \cup \Gamma 2 \vdash t'
\end{array}
\]

\[
\text{conv_rule}
\]

\[
\text{where} \ c \ \text{t gives} \ \Gamma 2 \vdash t \Rightarrow t'.
\]

Errors

\begin{align*}
7104 & \ \text{Result of conversion, ?0, ill-formed}
\end{align*}

Also as the failure of the conversion upon the conclusion of the theorem.

SML

val cthm_eqn_ctxt : \text{CANON} \rightarrow \text{THM} \rightarrow \text{EQUATION_CONTEXT};

Description \ \text{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 \text{thm_eqn_ctxt} \ (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 \text{thm_eqn_ctxt} \ will be discarded.}

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUAL USR029
val c_contr_rule : TERM -> THM -> THM;

**Description** Classical contradiction rule:

\[
\frac{\Gamma, \neg t' \vdash F}{F \vdash t} \\
\text{c_contr_rule}
\]

Note that the argument is the unnegated form of what must be present in the assumption list for success. Works up to α-conversion.

**Errors**
- 7001 ?0 is not of form: ‘Γ ⊨ F’
- 3031 ?0 is not of type ‘:BOOL’
- 7003 Negation of ?0 is not in assumption list

val disch_rule : TERM -> THM -> THM;

**Description** Prove an implicative theorem, removing, if α-convertibly present, the antecedent of the implication from the assumption list, and failing if it is not present.

\[
\frac{\Gamma, t1' \vdash t2}{\Gamma \vdash t1 \Rightarrow t2} \\
\text{disch_rule}
\]

**See Also** ⇒_intro (which does not fail if term not in assumption list)

**Errors**
- 7031 ?0 not α–convertibly present in assumption list

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.

\[
\frac{\Gamma \vdash t = v[i1,...,in]}{\text{eq_match_conv1}}
\]

where ‘u[i1,...,in]’ is α-convertible to ‘t’. 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**
- 27003 ?0 is not of the form ‘Γ ⊨ \forall x1 ... xn• u = v’
- 7076 Could not match term ?0 to LHS of theorem ?1

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUAL USR029
8.1. 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

\[
\Gamma \vdash t = v' \quad \text{(eq_match_conv)} \quad (\Gamma \vdash \forall \ldots \cdot u = v)
\]

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
These are some of the default list of theorems used by those rewriting rules, conversions and tactics whose names do not begin with ‘pure_’:

\[\forall x \cdot (x = x) \Leftrightarrow T\]

\[\forall t \cdot (T \Leftrightarrow t) \land ((t \Leftrightarrow T) = t) \land ((F \Leftrightarrow t) = (\neg t)) \land (t \Leftrightarrow F) = (\neg t)\]

\[\forall t \cdot (\neg\neg t) = t \land ((\neg t) = F) \land (\neg F) = T\]

\[\forall t \cdot (T \land t) \land ((t \land T) = t) \land (\neg (F \land t)) \land (\neg (t \land F)) \land (t \land t) = t\]

\[\forall t \cdot (T \lor t) \land ((t \lor T) = t) \land ((t \lor F) = T) \land (t \Rightarrow F) = (\neg t)\]

\[\forall t \cdot (T \Rightarrow t) = t \land ((t \Rightarrow T) = T) \land (t \Rightarrow t) = T\]

\[\forall t \cdot (\forall x \cdot t) = t\]

\[\forall t \cdot (\exists x \cdot t) = t\]

\[\forall t \cdot (\lambda x \cdot t1) t2 = t1\]

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`.

---

Symmetry of equality:

\[\vdash (t1 = t2) \Leftrightarrow (t2 = t1)\]

**See Also** `eq_sym_rule`

**Errors**

3014  ?0 is not of form: "t = u"
8.1. General Inference Rules

SML

|val eq_sym_rule : THM -> THM;

Description Symmetry of equality:

\[
\begin{array}{c}
\Gamma \vdash t_1 = t_2 \\
\Gamma \vdash t_2 = t_1
\end{array}
\]

eq_sym_rule

A built-in inference rule.

See Also eq_sym_conv

Errors

6020 ?0 is not of the form: ‘\(\Gamma \vdash t_1 = t_2\)’

SML

|val eq_trans_rule : THM -> THM -> THM;

Description Transitivity of equality:

\[
\begin{array}{c}
\Gamma_1 \vdash t_1 = t_2; \Gamma_2 \vdash t_2' = t_3 \\
\Gamma_1 \cup \Gamma_2 \vdash t_1 = t_3
\end{array}
\]

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 ‘\(\Gamma_2 \vdash t_2a = t_3\)’

where \(\langle t_2\rangle\) and \(\langle t_2a\rangle\) are \(\alpha\)-convertible

SML

|val EVERY_CAN : CANON list -> CANON

Description EVERY_CAN is a canonicalisation function combinator which combines the elements of its argument using THEN_CAN:

\[\text{EVERY\_CAN } [\text{can1, can2, ...}] = \text{can1 THEN\_CAN can2 THEN\_CAN ...}\]

See Also CANON

SML

|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.
\begin{verbatim}
val ext_rule : THM -> THM;

Description  Extensionality of functions in ICL HOL.

\begin{center}
\begin{align*}
\Gamma \vdash f & = g \\
\Gamma \vdash \forall x \cdot f \ x & = g \ x
\end{align*}
\end{center}

where \( x \) is a machine-generated variable of appropriate type, not found free in the equational theorem.

Errors
\begin{itemize}
\item 6020  ?0 is not of the form: ‘\( \Gamma \vdash t1 = t2 \)’
\item 7026  ?0 is not an equation of functions
\end{itemize}
\end{verbatim}

\begin{verbatim}
val fail_canon    : CANON

Description  This is a canonicalisation function which always fails. It is the identity for \texttt{ORELSE\_CAN}.

See Also \texttt{CANON}

Errors
\begin{itemize}
\item 26201  Failed as requested
\end{itemize}
\end{verbatim}

\begin{verbatim}
val fail_conv    : CONV;

Description  This conversion always fails.

Errors
\begin{itemize}
\item 7061  Failed as requested
\end{itemize}
\end{verbatim}

\begin{verbatim}
val fail_with_canon : string -> int -> (unit -> string) list -> CANON

Description  This is a canonicalisation function which always fails by passing its arguments to \texttt{fail} (q.v.).

See Also \texttt{fail\_can}
\end{verbatim}

\begin{verbatim}
val fail_with_conv : string -> CONV;

Description  This conversion always fails, with the error message being its string argument.

Errors
\begin{itemize}
\item 7075  ?0
\end{itemize}
\end{verbatim}

\begin{verbatim}
val FIRST\_CAN    : CANON list -> CANON

Description  \texttt{FIRST\_CAN} is a canonicalisation function combinator which combines the elements of its argument using \texttt{ORELSE\_CAN}:

\begin{center}
\texttt{FIRST\_CAN [can1, can2, ...]} = \texttt{can1 ORELSE\_CAN can2 ORELSE\_CAN ...}
\end{center}

See Also \texttt{CANON}

Errors
\begin{itemize}
\item 26202  the list of canonicalisation functions is empty
\end{itemize}
\end{verbatim}
### 8.1. General Inference Rules

#### `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

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7103</td>
<td>List may not be empty</td>
</tr>
</tbody>
</table>

or as the failure of the last conversion.

#### `val FORWARD_CHAIN_CAN : CANON list -> CANON;`

**Description** `FORWARD_CHAIN_CAN`, which has the alias `FC_CAN`, is a parameterised variant of `fc_canon`. Given a list of canonicalisation functions `cans`, `FC_CAN cans` behaves as `fc_canon` would do if the line

| |- A | FIRST_C cans A |

were inserted at the beginning of the table of transformations given in the description of `fc_canon`.

For example, `fc_canon1`, q.v., is the same as:

| |- FC_CAN ((fn (x, y) => [x,y]) o || elim); |

**Uses** In tactic programming, or, occasionally interactively, typically in circumstances where neither `fc_canon` nor `fc_canon1` is able to generate enough implications.
Lemma 8. PROOF IN HOL

The asymmetry in the rules is deliberate. E.g., they derive \( A \Rightarrow B \Rightarrow A \Rightarrow C \) 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, forward\_chain\_tac, FC\_CAN}

```sml
val forward\_chain\_canon : \text{THM} \rightarrow \text{THM list};
val fc\_canon : \text{THM} \rightarrow \text{THM list};
val forward\_chain\_canon1 : \text{THM} \rightarrow \text{THM list};
val fc\_canon1 : \text{THM} \rightarrow \text{THM list};
```

**Description** \textit{forward\_chain\_canon} is a canonicalisation function which uses a theorem to generate a list of implications. (\textit{fc\_canon} is an alias for \textit{forward\_chain\_canon}.) It may be used for constructing rules and tactics in conjunction with \textit{forward\_chain\_rule}. An example of such a tactic is \textit{forward\_chain\_tac}. \textit{forward\_chain\_canon1}, which has alias \textit{fc\_canon1}, is just like \textit{fc\_canon} except for its treatment of bi-implications. The effects of \textit{fc\_canon} and \textit{fc\_canon1} are shown schematically in the following table (which only shows assumptions relevant to the process):

| \( \vdash A \land B \) | \( \vdash A \land B \) |
| \( \vdash \forall x \rightarrow A \) | \( \vdash A \land B \) |
| \( \vdash A \rightarrow B \) | \( \vdash A \rightarrow B \) |
| \( \vdash T \rightarrow B \) | \( \vdash T \rightarrow B \) |
| \( A \vdash \neg A \rightarrow B \) | \( A \vdash \neg A \rightarrow B \) |
| \( F \vdash B \) | \( F \vdash B \) |
| \( A \vdash B \) | \( A \vdash B \) |
| \( A \vdash B \) | \( A \vdash B \) |
| \( T \) | \( T \) |
| \( A \) | \( A \) |

The intension here is that is that the first applicable transformation is applied repeatedly until no further change is possible. The resulting theorems are then universally quantified over all of the free variables in their conclusions which were not free in the original theorem. In the table, \textit{st} and \textit{sc} stand for attempts to apply the theorem and conclusion stripping conversions in the current proof context (as returned by \textit{current\_ad\_st\_conv} and \textit{current\_ad\_sc\_conv}). If the stripping conversions fail then \textit{st} and \textit{sc} have no effect. \( x' \) denotes a variable name derived from \( x \) and chosen to avoid variable capture problems. \textit{xf} stands for a nested recursive application of the transformation process.

In the transformations involving \( \Rightarrow\_\text{intro} \) the implication is only introduced if the antecedent is in the assumptions. So, for example, \( A \Rightarrow B \Rightarrow A \Rightarrow C \) 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.).
8.1. General Inference Rules

**SML**

```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 \( \Rightarrow \text{match\_mp\_rule2} \) if \( \text{current\_ad\_mmp\_rule}() \) returns \text{Nil}. (\text{fc\_rule} is an alias for \text{forward\_chain\_rule}.) \text{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 \text{imps} and one from \text{ants}. As a special case, if any theorem to be returned is determined to have \( \lceil F \rfloor \) as its conclusion, the first such found will be returned as a singleton list. In order to work well in conjunction with \text{fc\_canon} and \text{fc\_tac} the theorems returned by the matching modus ponens rule are transformed as follows:

1. Theorems of the form: \( \vdash \forall x_1 \ldots \bullet t_1 \Rightarrow t_2 \Rightarrow \cdots \Rightarrow \neg t_k \Rightarrow F \) have their final implication changed to \( t_k \).
2. Theorems of the form: \( \vdash \forall x_1 \ldots \bullet t_1 \Rightarrow t_2 \Rightarrow \cdots \Rightarrow t_k \Rightarrow F \) have their final implication changed to \( \Rightarrow \neg t_k \).
3. All theorems are universally quantified over all the variables which appear free in their conclusions but not in their assumptions (using \text{all\_\forall\_intro}).

Note that when the matching modus ponens rule is either \( \Rightarrow \text{match\_mp\_rule2} \) or \( \Rightarrow \text{match\_mp\_rule1} \), there is some control over the number of results generated, since variables which appear free in \text{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**  \text{forward\_chain\_tac}, \text{forward\_chain\_canon}.

**SML**

```sml
val FORWARD_CHAIN_CAN : CANON list -> CANON;
val FC_CAN : CANON list -> CANON;
```

**Description**  These are just like \text{FORWARD\_CHAIN\_CAN}, q.v., except that they do not break up bi-implications. Thus, given a list of canonicalisation functions \text{cans}, \text{FC\_C\_CAN cans} behaves as \text{fc\_canon} would do if the line

\[ \vdash A \quad \Rightarrow \quad \text{FIRST\_CAN cans A} \]

were inserted at the beginning of the table of transformations given in the description of \text{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 \text{fc\_C\_CAN} is not able to generate enough implications.
Description: `forward_chain⇔canon` is a canonicalisation function very similar to `forward_chain_canon`, q.v. The difference is that `forward_chain⇔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⇔canon rewrite_tac []` can instantiate an assumption of the form $\forall x_1 x_2 \ldots \cdot A \Rightarrow B \Rightarrow (C \iff D)$ and use the result to rewrite instances of $C$.

See Also: `FC_T1`, `ALL_FC_T1` etc.

---

Description: “Not False” is true.

```
| f_thm : THM;
| Theorem ⊢ ¬ F
```

```
| id_canon : CANON
| Description: This is the identity for the canonicalisation function combinator `THEN_CAN`:
| id_canon thm = [thm]
| See Also: CANON
```

```
| 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
```

```
| if_app_conv : CONV;
| Description: Move a function application out of a conditional.
| Conversion
| ⊢ (if a then f b else f' c) = \[ if_app_conv \]
| \[ \gamma (if a then f b else f' c) \]
| \[ f(if a then b else c) \]
| where $f$ and $f'$ are $\alpha$-convertible, and $f$ is used on the RHS of the resulting equational theorem
| Errors
| 7037 ?0 is not of the form: $\gamma (if a then (f b) else (g c))$
| 7038 ?0 is not of the form: $\gamma (if a then (f b) else (fa c))$
| where $\gamma f$ and $\gamma fa$ are $\alpha$-convertible
```
### 8.1. General Inference Rules

#### Val if\_else\_elim : THM → THM;

**Description**  
Give the dependence of the `else` branch of a conditional upon the condition.

<table>
<thead>
<tr>
<th>Rule</th>
<th>[Γ ⊢ \text{if } tc \text{ then } tt \text{ else } te]</th>
<th>[Γ ⊢ \neg tc \Rightarrow te]</th>
<th>if_else_elim</th>
</tr>
</thead>
</table>

**Errors**  
7012  
?0 is not of the form: ‘Γ ⊢ if tc then tt else te’

#### Val if\_intro : TERM → THM → THM → THM;

**Description**  
Introduce a conditional, based on the assumptions of two theorems.

<table>
<thead>
<tr>
<th>Rule</th>
<th>[Γ_1, a ⊢ tt; \ Γ_2, \neg a' ⊢ et]</th>
<th>[Γ_1 \cup Γ_2 ⊢ \text{if } a \text{ then } tt \text{ else } et]</th>
<th>if_intro</th>
</tr>
</thead>
</table>

where `a` and `a'` are α-converible. Actually, the assumptions may be missing, and the rule still works.

**Example**  
\[(⊢ x = tt), (⊢ x = te)\]  
\[\vdash \text{if } a \text{ then } (x = tt) \text{ else } (x = te)\]  
\[\vdash x = \text{if } a \text{ then } tt \text{ else } te\]  
\[\vdash x = \text{if } a \text{ then } tt \text{ else } te\]  

**Errors**  
3031  
?0 is not of type ⊤:BOOL

#### Val if\_then\_elim : THM → THM;

**Description**  
Give the dependence of the `then` branch of a conditional upon the condition.

<table>
<thead>
<tr>
<th>Rule</th>
<th>[Γ ⊢ \text{if } tc \text{ then } tt \text{ else } te]</th>
<th>[Γ ⊢ tc \Rightarrow tt]</th>
<th>if_then_elim</th>
</tr>
</thead>
</table>

**Errors**  
7012  
?0 is not of the form: ‘Γ ⊢ if tc then tt else te’

---

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
val initial_rw_canon : CANON;

**Description**  This is the initial rewrite canonicalisation function, defined as

```sml
val initial_rw_canon = REWRITE_CAN(READ_CAN(REPEAT_CAN(FIRST_CAN [simple_∀ rewrite_canon, ∧_rewrite_canon, simple_¬ 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;
2. dividing conjunctive theorems into their conjuncts;
3. changing $\Gamma \vdash \neg(t1 \lor t2)$ to $\neg t1 \land \neg t2$;
4. changing $\Gamma \vdash \neg \exists x \bullet t$ to $\forall x \bullet \neg t$;
5. changing $\Gamma \vdash \neg \neg t$ to $t$;
6. changing $\Gamma \vdash \neg t$ to $t \iff F$;
7. changing $\Gamma \vdash F$ to $\Gamma \vdash \forall x \bullet x$;
8. if none of the above apply, changing $\Gamma \vdash t$ to $\Gamma \vdash t = T$.

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.

---

val inst_term_rule : (TERM * TERM) list -> THM -> THM;

**Description**  Parallel instantiation of term variables within a theorem’s conclusion to some other values.

```sml
Rule
\[ \Gamma \vdash t[x_1, \ldots, x_n] \quad \text{inst_term_rule} \quad \Gamma \vdash t[t_1, \ldots, t_n] \quad [..., (\tau t^i, \tau x^i), ...] \]
```

A built-in inference rule.

**See Also**  asm_inst_term_rule

**Errors**

- 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
8.1. General Inference Rules

SML

|val inst_type_rule : (TYPE * TYPE) list -> THM -> THM;

Description
Parallel instantiation of some of the type variables of the conclusion of a theorem.

\[
\Gamma \vdash t[tyv_1,...,tyv_n] \quad inst_type_rule \quad (\sigma_1, tyv_1), ..., (\sigma_n,tyv_n)
\]

\(inst_type_rule\) will instantiate each type variable in \(talist\) 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 \(asm\_inst\_type\_rule\).

A primitive inference rule.

See Also
\(asm\_inst\_type\_rule\) for something that also works on type variables in the assumption list.

Errors

3019 ?\(\theta\) is not a type variable
6006 Trying to instantiate type variable ?\(\theta\), which occurs in assumption list

SML

|val LEFT_C : CONV -> CONV;

Description
Apply a conversion to the first operand of a binary operator:

\[
\Gamma \vdash f \ a \ b = f \ a' \ b \quad LEFT_C \quad (c : CONV)
\]

where \(c\ a\) gives \(\vdash a = a'\). \(f\) may itself be a function application.

Errors

3013 ?\(\theta\) is not of form:\(\vdash f \ a \ b\)
7104 Result of conversion, ?\(\theta\), ill-formed

Also as the failure of the conversion.

SML

|val let_conv : CONV;

Description
Eliminate an outermost \(let... and ... in ...\) construct.

\[
\Gamma \vdash (let \ vs1[x_1,y_1,..] = tI \\
and ... and \ vsn[xn,yn,..] = tn \\
in \ t[x_1,...,xn,..] \\
= t[t_1x,...,t_1y,...,tnx,tny,...]) \quad let_conv \\
\quad \vdash let \ vs1[x_1,y_1,..] = tI \quad t_1 \ and ... \ vsn[xn,yn,..] = tn \quad in \ t[x_1,...,xn,..] \quad \vdash
\]

Where the \(t_2ix\) is the component of \(t_{2i}\) matching \(x_{2i}\) when \(t_{2i}\) matches \(vs_{2i}[x_{2i},y_{2i},..]\).

Errors

4009 ?\(\theta\) is not of form:\(\vdash let ... in ...\)

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
val list_simple_\forall_elim : TERM list \rightarrow THM \rightarrow THM;

**Description** Generalised \forall elimination.

\[
\frac{\Gamma \vdash \forall x_1 \ldots x_n \cdot t[x_1, \ldots, x_n]}{\Gamma \vdash t[I^\forall, \ldots, I^n]} \quad \text{list_simple_\forall_elim}
\]

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 \cdot t’ where the ‘\forall’ are ?1 variables

val list_simple_\forall_intro : TERM list \rightarrow THM \rightarrow THM;

**Description** Generalised simple \forall introduction.

\[
\frac{\Gamma \vdash t[x_1, \ldots, x_n]}{\Gamma \vdash \forall x_1 \ldots x_n \cdot t[x_1, \ldots, x_n]} \quad \text{list_simple_\forall_intro}
\]

**See Also** \forall_intro

**Errors** Same messages as simple_\forall_intro.

val list_simple_\exists_intro : TERM list \rightarrow TERM \rightarrow THM \rightarrow 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.

\[
\frac{\Gamma \vdash t[I^\exists, \ldots, I^n]}{\Gamma \vdash \exists x_1 x_2 \ldots \cdot t[x_1,x_2,\ldots]} \quad \text{list_simple_\exists_intro}
\]

**Errors**

- 7047 ?0 cannot be matched to conclusion of theorem ?1

val list_\land_intro : THM list \rightarrow THM;

**Description** Conjoin a list of theorems.

\[
\frac{\left[\Gamma 1 \vdash t_1, \ldots, \Gamma n \vdash t_n\right]}{\Gamma 1 \land \ldots \land \Gamma n \vdash t_1 \land \ldots \land t_n} \quad \text{list_\land_intro}
\]

**Errors**

- 7107 List may not be empty
### list\_\forall\_elim

**Description** Generalised \(\forall\) elimination. Specialise a universally quantified theorem with given values, instantiating the types of the theorem as necessary.

\[
\begin{array}{c|c}
\Gamma \vdash \forall x_1 \ldots x_n \cdot t[x_1, \ldots, x_n] & \text{list}\_\forall\_elim \\
t'[t_1, \ldots, t_n] & \Gamma \vdash [t_1^\gamma, \ldots, t_n^\gamma]
\end{array}
\]

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 \(\text{Fst}\) and \(\text{Snd}\) as necessary to match the structure of \([t_i^\gamma]\).

Note that due to the type instantiation this function is somewhat more that a fold of \(\forall\_elim\).

**See Also** \(\forall\_elim\), all \(\forall\_elim\).

**Errors**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>27014</td>
<td>(0) is not of the form: (\Gamma \vdash \forall vs_1 \ldots vs_i \cdot t') where (i \geq ?1)</td>
</tr>
<tr>
<td>27015</td>
<td>(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)</td>
</tr>
<tr>
<td>27016</td>
<td>(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</td>
</tr>
</tbody>
</table>

### list\_\forall\_intro

**Description** Generalised \(\forall\) introduction.

\[
\begin{array}{c|c}
\Gamma \vdash t[x_1, \ldots, x_n] & \text{list}\_\forall\_intro \\
\Gamma \vdash \forall x_1 \ldots x_n \cdot t[x_1, \ldots, x_n] & \Gamma \vdash [x_1^\gamma, \ldots, x_n^\gamma]
\end{array}
\]

**See Also** \(\forall\_intro\), all \(\forall\_intro\).

**Errors** Same messages as \(\forall\_intro\).

### 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**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>7005</td>
<td>Conversion fails on term and all its subterms</td>
</tr>
</tbody>
</table>
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.

\[
\frac{\Gamma_1 \vdash u_1 = u_2; \Gamma_2 \vdash v_1 = v_2}{\Gamma_1 \cup \Gamma_2 \vdash u_1 v_1 = u_2 v_2}
\]

*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: ‘\(\Gamma \vdash t_1 = t_2\)’
- 6023 ?0 and ?1 are not of the form: ‘\(\Gamma_1 \vdash u_1 = u_2\)’ and ‘\(\Gamma_2 \vdash v_1 = v_2\)’
  where \(\Gamma u_1\) can be functionally applied to \(\Gamma v_1\)

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).

\[
\frac{\Gamma_1 \vdash t_1 \Rightarrow t_2; \Gamma_2 \vdash \neg t_2'}{\Gamma_1 \cup \Gamma_2 \vdash \neg t_1}
\]

*modus_tollens_rule*

where \(t_2\) and \(t_2'\) are \(\alpha\)-convertible.

**Errors**

- 7040 ?0 is not of the form: ‘\(\Gamma \vdash t_1 \Rightarrow t_2\)’
- 7051 ?0 and ?1 are not of the form: ‘\(\Gamma_1 \vdash t_1 \Rightarrow t_2\)’ and ‘\(\Gamma_2 \vdash \neg t_2'\)’
  where \(\Gamma t_2\) and \(\Gamma t_2'\) 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
8.1. General Inference Rules

val ONCE_MAP_WARN_C : string -> CONV -> CONV;

**Description**  This is an equivalent to \texttt{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 \texttt{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.

val ORELSE\_CAN : (CANON * CANON) -> CANON

**Description**  \texttt{ORELSE\_CAN} is a canonicalisation function combinator written as an infix operator. \((\text{can1 ORELSE\_CAN can2})\text{thm}\) is the same \text{can1 thm} unless evaluation of \text{can1 thm} fails in which case it is the same as \text{can2 thm}.

**See Also**  \texttt{CANON}

val ORELSE\_C : (CONV * CONV) -> CONV;

**Description**  Attempt to apply one conversion, and if that fails, try the second one.

\[
\frac{
\Gamma \vdash t = t' \\
\text{(c1: CONV) ORELSE\_C (c2: CONV)}
}{
\Gamma \vdash t = t'
}
\]

where \(c1 t\) returns \(\Gamma \vdash t = t'\), or \(c1\) fails, and \(c2 t\) returns \(\Gamma \vdash t = t'\).

**See Also**  \texttt{FIRST\_C} (the iterated version of this function), \texttt{THEN\_C}, \texttt{AND\_OR\_C}, and \texttt{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.

\[
\frac{\Gamma \vdash \texttt{ML.mk} \_\texttt{N} \ m \gamma + \texttt{ML.mk} \_\texttt{N} \ n \gamma = \texttt{ML.mk} \_\texttt{N}(m + n) \gamma}{\Gamma}
\]

**Uses**  For doing fast arithmetic proofs.

**Errors**

6085  ?0 is not of the form: \( \texttt{ML.mk} \_\texttt{N} \ m \gamma + \texttt{ML.mk} \_\texttt{N} \ n \gamma \)
8.1. General Inference Rules

**SML**

```sml
val prim_rewrite_conv = CONV_NET -> CANON -> (THM -> TERM * CONV) OPT ->
(CONV -> CONV) -> EQN_CXT -> THM list -> CONV;
```

**Description**  The primitive rewrite conversion.

<table>
<thead>
<tr>
<th>Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

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 \( \text{with thms} \) 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. \( \text{thm eqn ctx} \) is used if \( \text{Nil} \) is supplied.

- **traverse** This is a conversional, which defines the traversal of term \( t \) by the rewriting conversion derived from \( \text{prim rewrite conv} \)’s other arguments.

- **with_eqn_ctx** 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 \( \text{canon} \) 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.

\( \text{prim rewrite conv} \) 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 \( \text{with eqn ctx} \) need not be incurred in calls with the same \( \text{with eqn ctx} \) but different \( \text{with thms} \).
val prim_rewrite_rule = CONV NET \rightarrow CANON \rightarrow (THM \rightarrow TERM \ast CONV) OPT \rightarrow
(CONV \rightarrow CONV) \rightarrow EQN_CXT \rightarrow THM list \rightarrow THM \rightarrow 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:

\[
\begin{array}{c}
\Gamma \vdash t \\
\Gamma \cup \Gamma_1 \vdash t'
\end{array}
\]

where \( t' \) is the result of rewriting \( t \) in the manner prescribed by the arguments, and \( \Gamma_1 \) 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.

\[
\begin{array}{c}
\vdash \text{mk}_\mathbb{N}(m+1)^\uparrow = \text{Suc}_{\text{mk}_\mathbb{N} m}^\uparrow
\end{array}
\]

\[
\begin{array}{c}
\vdash \text{mk}_\mathbb{N} 0^\uparrow = \text{Zero}
\end{array}
\]

Errors

3026  ?0 is not a numeric literal

See Also  mk_\mathbb{N}, suc_conv

val prove_asm_rule = THM \rightarrow THM \rightarrow THM;

Description  Eliminate an assumption with reference to a the assumption being a conclusion
of a theorem.

\[
\begin{array}{c}
\Gamma_1 \vdash t_1; \Gamma_2, t_1 \vdash t_2
\end{array}
\]

\[
\begin{array}{c}
\Gamma_1 \cup \Gamma_2 \vdash t_2
\end{array}
\]

This will in fact work even if the assumption is not present.
8.1. General Inference Rules

**SML**

```sml
val RANDS_C : CONV -> CONV;
```

**Description**  Apply a conversion to each of the arguments of a function

**Rule**

\[
\vdash f\ a\ \ldots\ z = f\ a'\ldots z' \\
\]

where \(c\ a\) gives \(\vdash a = a'\), etc. The function \(f\) may have no arguments in which case \(\text{refl}\_\text{conv}\ f\) is returned.

**Errors**

- 7104 Result of conversion, ?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:

**Rule**

\[
\vdash f\ a = f\ a' \\
\]

where \(c\ a\) gives \(\vdash a = a'\).

**Errors**

- 3010 ?0 is not of form: \(\langle t1 \ t2\rangle\)
- 7104 Result of conversion, ?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:

**Rule**

\[
\vdash f\ a = f'\ a \\
\]

where \(c\ f\) gives \(\vdash f = f'\).

**Errors**

- 3010 ?0 is not of form: \(\langle t1 \ t2\rangle\)
- 7104 Result of conversion, ?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.

**Rule**

\[
\vdash t = t \\
\]

A primitive inference rule.
val REPEAT_C1 : CONV -> CONV;

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:

\[
\text{fun REPEAT_C1 (c:CONV) = (c THEN_TRY_C REPEAT_C1 c)}
\]

Errors  As the error of the conversion if it cannot be applied at least once.

val REPEAT_CAN : CANON -> CANON

Description  REPEAT_CAN is a canonicalisation function combinator which repeatedly applies its argument until it fails:

\[
\text{REPEAT_CAN can thm = ((can THEN_CAN REPEAT_CAN can) ORELSE_CAN id_can) thm}
\]

See Also  CANON

val REPEAT_C : CONV -> CONV;

Description  Repeatedly apply a conversion to a term. To be more precise, the functionality is equivalent that of the following definition:

\[
\text{fun REPEAT_C (c:CONV) = (c THEN_C (REPEAT_C c)) ORELSE_C refl_conv}
\]

See Also  REPEAT_C1

val REPEAT_MAP_C : CONV -> CONV;

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

val REWRITE_CAN : CANON -> CANON;

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., \texttt{commit_pc}) the list of rewrite canonicalisations in the argument (see \texttt{get_rw_canons}), \texttt{arg}, will be converted to a single canonicalisation in the result by:

\[
\text{REWRITE_CAN (REPEAT_CAN(FIRST_CAN (arg @ [\Rightarrow t_rewrite_canon])))}
\]
8.1. General Inference Rules

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.).

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_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 rule is "asm" then the theorems rewritten with will include the canonicalised asm_rule 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.).
8.1. General Inference Rules

SML

val SIMPLE_BINDER_C : CONV → CONV;

**Description**  Apply a conversion to the body of a simple binder term:

\[
\frac{\Gamma \vdash (B \, x \mapsto p[x]) = (B \, x \mapsto p'[x])}{\Gamma \vdash B \, x \mapsto p}
\]

where \( c \, p[x] \) gives \( \vdash p[x] = p'[x] \), and \( B \) is a binder.

**Errors**

7059 ?0 is not of the form: \( \Gamma \vdash B \, x \mapsto p \) where \( \Gamma \vdash B \) is a binder and \( \Gamma \vdash x \) a variable

7104 Result of conversion, (?0, ill-formed)

Also as the failure of the conversion.

SML

val simple_eq_match_conv : THM → 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.

\[
\frac{\Gamma \vdash \forall \cdots \bullet u = v}{\Gamma' \vdash t = v'}
\]

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
SML

\texttt{val simple\_eq\_match\_conv1 : THM \rightarrow CONV ;}

\textbf{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.

\textit{Conversion}

\[
\begin{array}{c}
\Gamma \vdash t = v[t_1,...,t_n] \\
simple\_eq\_match\_conv1 \\
(\Gamma \vdash \forall x_1 \ldots x_n \cdot u[x_1,...,x_n] = v[x_1,...,x_n]) \\
\end{array}
\]

where \(\bar{u}[t_1,...,t_n]\) is \(\alpha\)-convertible to \(\bar{v}\). 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.

\textbf{Errors}

\begin{itemize}
    \item 7095  \textit{?0} is not of the form \(\Gamma \vdash \forall x_1 \ldots x_n \cdot u = v\) where \(\bar{xi}\) are variables
    \item 7076  Could not match term \(\textit{?0}\) to LHS of theorem \(\textit{?1}\)
\end{itemize}

SML

\texttt{val simple\_ho\_eq\_match\_conv : THM \rightarrow CONV}

\textbf{Description}  This conversion is like \texttt{simple\_eq\_match\_conv} but uses higher-order matching. It uses \texttt{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).

\textit{Conversion}

\[
\begin{array}{c}
\Gamma' \vdash t = v' \\
simple\_ho\_eq\_match\_conv \\
(\Gamma' \vdash \forall \ldots \cdot u = v) \\
\end{array}
\]

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 \(t\).

\textbf{Errors}

\begin{itemize}
    \item 7095  \textit{?0} is not of the form \(\Gamma' \vdash \forall x_1 \ldots x_n \cdot u = v'\) where \(\bar{xi}\) are variables
    \item 7076  Could not match term \(\textit{?0}\) to LHS of theorem \(\textit{?1}\)
\end{itemize}
8.1. 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.

\[
\Gamma \vdash t = v'
\]

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 \( \Gamma \vdash \forall x_1 \ldots x_n \bullet u = v' \) where \( \Gamma \vdash \forall x_1 \ldots x_n \bullet u = v' \)

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 \(\Leftrightarrow\).

\[
\frac{\Gamma \vdash \forall x_1 \ldots \bullet t_1 \Leftrightarrow t_2; \; \Gamma_2 \vdash t_1'}{\Gamma_1' \cup \Gamma_2 \vdash t_2'}
\]

\(\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 \(\Rightarrow\_elim\) (Modus Ponens on \(\Rightarrow\)), simple_\(\Rightarrow\)\_match_mp_rule

Errors

7044 Cannot match \(?0\) and \(?1\)

7046 \( ?0 \) is not of the form \( \Gamma \vdash \forall x_1 \ldots x_n \bullet u \Leftrightarrow v' \)

SML

val simple_\(\Leftrightarrow\)\_match_mp_rule1 : THM -> THM -> THM;

Description A matching Modus Ponens for \(\Leftrightarrow\) that doesn’t affect assumption lists.

\[
\frac{\Gamma \vdash \forall x_1 \ldots \bullet t_1 \Leftrightarrow t_2; \; \Gamma_2 \vdash t_1'}{\Gamma_1 \cup \Gamma_2 \vdash t_2'}
\]

\(\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 \(\Rightarrow\_elim\) (Modus Ponens on \(\Rightarrow\)), simple_\(\Rightarrow\)\_match_mp_rule1

Errors

7044 Cannot match \(?0\) and \(?1\)

7046 \( ?0 \) is not of the form \( \Gamma \vdash \forall x_1 \ldots x_n \bullet u \Leftrightarrow v' \)
val simple ⇒ match_mp_rule : THM ⇒ THM ⇒ THM;

Description  A matching Modus Ponens rule for an implicative theorem.

\[ \Gamma_1 \vdash \forall x_1 \ldots \bullet t_1 \Rightarrow t_2; \ \Gamma_2 \vdash t_1' \]
\[ \Gamma_1 \cup \Gamma_2 \vdash t_2' \]

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  simple ⇒ match_mp_rule1, simple ⇒ match_mp_rule2

Errors  7044  Cannot match ?0 and ?1
7045  ?0 is not of the form ‘\( \Gamma \vdash \forall x_1 \ldots x_n \bullet u \Rightarrow v \)’

val simple ⇒ match_mp_rule1 : THM ⇒ THM ⇒ THM;
val simple ⇒ match_mp_rule2 : THM ⇒ THM ⇒ THM;

Description  Two variants on a matching Modus Ponens rule for an implicative theorem.

\[ \Gamma_1 \vdash \forall x_1 \ldots \bullet t_1 \Rightarrow t_2; \ \Gamma_2 \vdash t_1' \]
\[ \Gamma_1 \cup \Gamma_2 \vdash t_2' \]

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 \( t_2 \).

\( \text{simple} ⇒ \text{match}_\text{mp}_\text{rule2} \) is just like \( \text{simple} ⇒ \text{match}_\text{mp}_\text{rule1} \) 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.

See Also  \( \text{simple} ⇒ \text{match}_\text{mp}_\text{rule} \)

Errors  7044  Cannot match ?0 and ?1
7045  ?0 is not of the form ‘\( \Gamma \vdash \forall x_1 \ldots x_n \bullet u \Rightarrow v \)’

val simple ∀elim : TERM ⇒ THM ⇒ THM;

Description  Instantiate a universally quantified variable to a given value.

\[ \Gamma \vdash \forall x \bullet t_2[x] \]
\[ \Gamma \vdash t_2'[t_1] \]

where \( t_2' \) is renamed from \( t_2 \) to prevent bound variable capture, and \( x \) is a variable.

Errors  3012  ?0 and ?1 do not have the same types
7039  ?0 is not of the form ‘\( \Gamma \vdash \forall x \bullet t \)’ where ‘\( x \)’ is a variable
8.1. General Inference Rules

SML

val simple_\forall_intro : TERM \to THM \to THM;

Description Introduce a simple universally quantified theorem.

\[
\begin{array}{c}
\Gamma \vdash t \\
\hline
\Gamma \vdash \forall x \cdot t
\end{array}
\]

simple_\forall_intro \quad \forall_x \exists_y \left( P[x,y] \right) \iff \left( \exists y \cdot \forall x \cdot P[x,y] \right)

A built-in inference rule.

See Also \ \forall_intro

Errors

3007 \ ?0 is not a term variable
6005 \ ?0 occurs free in assumption list

SML

val simple_\exists_conv : CONV;

Description Swap the order of a simple \forall and \exists:

\[
\begin{array}{c}
\Gamma \vdash (\forall x \cdot \exists y \cdot P[x,y]) \iff \\
(\exists y' \cdot \forall x \cdot P[x, y'])
\end{array}
\]

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_\exists_elim : TERM \to THM \to THM \to THM;

Description Eliminate an existential quantifier.

\[
\begin{array}{c}
\Gamma \vdash \exists x \cdot t_1[x]; \ \Gamma_2, t_1[y] \vdash t_2 \\
\hline
\Gamma \cup \Gamma_2 \vdash t_2
\end{array}
\]

simple_\exists_elim \quad \forall \exists \Rightarrow \forall

where \( y \) must be variable which is not present elsewhere in the second theorem, nor in the conclusion of the first. \( t_1[y] \) 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
val simple_∃_intro : TERM -> THM -> THM;

Description Introduce an existential quantifier by reference to a witness.

\[ \Gamma \vdash t_1[t_2] \quad \Rightarrow \quad \Gamma \vdash \exists x \cdot t_1[x] \]

where \( \exists x \) is a variable.

Errors

3034 ?0 is not of form: \( \exists \ var \cdot body \)

7047 ?0 cannot be matched to conclusion of theorem ?1

val simple_∃∀_conv: CONV;

Description Swap the order of a simple \( \exists \) and \( ∀ \):

\[ \vdash (\exists x \cdot \forall y \cdot P[x,y]) \iff (\forall y \cdot \exists x \cdot P[x,y]) \]

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: \( \exists x \cdot \forall y \cdot P[x,y] \)

val simple_∃∀_conv1: CONV;

Description Swap the order of a simple \( \exists \) and \( ∀ \), where the first variable is always applied to the second:

\[ (\exists f \cdot \forall x \cdot P[f \; x, \; x]) \iff (\forall x \cdot \exists f' \cdot P[f', \; x]) \]

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: \( \exists f \cdot \forall x \cdot P[f; x, x] \)

val simple_∃ϵ_conv : CONV;

Description Give that \( ϵ \) of a predicate satisfies the predicate by reference to an \( \exists \) construct.

\[ \Gamma \vdash (\exists x \cdot p[x]) \iff p[ϵ \cdot x \cdot p \; x] \]

See Also \( ϵ \_ϵ\_rule \)

Errors

3034 ?0 is not of form: \( \exists \ var \cdot body \)
8.1. General Inference Rules

| val simple.∃_ε_rule : THM -> THM; |
| Description | Give that ε of a predicate satisfies the predicate by reference to an ∃ construct. It can properly handle paired existence. |
| Rule | \[ \Gamma \vdash \exists x \cdot p[x] \quad \Gamma' \vdash p[\varepsilon] \cdot p[x] \] (simple.∃_ε_rule) |
| See Also | ∃_ε_conv |
| Errors | 7092 ?0 is not of the form: 'Γ ⊢ \exists x \cdot p[x]' |

| val simple.∃_1_elim : THM -> THM; |
| Description | Express a ∃_1 in terms of ∃ and a uniqueness property. |
| Rule | \[ \Gamma \vdash \exists_1 x \cdot P[x] \quad \Gamma' \vdash \exists x \cdot P[x] \land \forall y \cdot P[y] \Rightarrow y = x \] (simple.∃_1_elim) |
| Errors | 7015 ?0 not of the form: 'Γ ⊢ \exists_1 x \cdot P[x]' |

| val simple.∃_1_intro : THM -> THM; |
| Description | Introduce ∃_1 by reference to a witness, and a uniqueness theorem. |
| Rule | \[ \Gamma_1 \vdash P'[t'] \quad \Gamma_2 \vdash \forall x \cdot P'[x] \Rightarrow x = t \] \[ \Gamma_1 \cup \Gamma_2 \vdash \exists_1 x \cdot P[x] \] (simple.∃_1_intro) |
| Where | P' is α-converible to P, and t' is α-converible to t. Notice that for the resulting theorem we take the bound variable name, x, and the form of the predicate, P, from the second theorem. |
| Errors | 7066 ?0 not of the form: 'Γ ⊢ \forall x \cdot P[x] \Rightarrow x = t' |
| 7067 ?0 and ?1 are not of the form: 'Γ ⊢ Pa[ta]' and 'Γ ⊢ \forall x \cdot P[x] \Rightarrow x = t' where 'Γ Pa' and 'Γ P', 'Γ ta' and 'Γ t' are α-converible |

| val simple.α_conv : string -> CONV; |
| Description | Rename a bound variable name, as a conversion. This only works with simple abstractions. |
| Rule | \[ \vdash (\lambda x \cdot t[x]) = (\lambda v \cdot t[v]) \] (simple.α_conv) |
| Errors | 3011 ?0 is not of form: 'Γ \vdash \lambda var \cdot t' |
| 7035 Cannot rename bound variable ?0 to ?1 as this would cause variable capture |
### simple β conv

**Description**
Apply a β-reduction to a simple abstraction.

\[
\begin{align*}
\Gamma \vdash (\lambda x \cdot t_1[x]) t_2 & = t_1[t_2] \\
\text{simple β conv} & \quad \Gamma (\lambda x \cdot t_1[x]) t_2 \gamma
\end{align*}
\]

A primitive inference rule.

**See Also**
β conv

**Errors**
6012 ?0 is not of the form: \(\Gamma (\lambda x \cdot t_1[x]) t_2 \gamma\) where \(\gamma\) is a variable

### simple β η conv

**Description**
If \(t\) is any term, \(\text{simple β η conv} \ t\) is a conversion which will prove all theorems of the form \(\Gamma \vdash t = s\) where \(t\) and \(s\) are simply αβη-equivalent, i.e., can be reduced to α-equivalent normal forms by β- and η-reduction involving only simple (rather than paired) λ-abstractions.

**Errors**
7131 ?0 and ?1 are not simply αβη-equivalent

### simple β η norm conv

**Description**
This conversion eliminates all simple β- and η-redexes from a term giving the βη-normal form. It does not eliminate β- and η-redexes involving abstraction over pairs. It fails if the term is already in normal form.

**Errors**
7130 ?0 contains no simple β- or η-redexes

### simple ε elim rule

**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.

\[
\begin{align*}
\Gamma_1 \vdash t' (\$ \epsilon \ t'') & ; \\
\Gamma_2, t x \vdash s & ; \\
\Gamma_1 \cup \Gamma_2 \vdash s
\end{align*}
\]

where \(t, t'\) and \(t''\) are α-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: ‘\(\Gamma \vdash t(\$ \epsilon \ t)\)’
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
8.1. General Inference Rules

\[
\text{val SIMPLE}_\lambda\text{C} : \text{CONV} \to \text{CONV};
\]

**Description**  Apply a conversion to the body of a simple abstraction:

\[
\frac{\Gamma \vdash (\lambda x \cdot p[x]) = (\lambda x \cdot p'[x])}{\text{SIMPLE}_\lambda\text{C}}
\]

\[
\begin{array}{l}
(c : \text{CONV}) \\
\Gamma \vdash \lambda x \cdot p^\perp
\end{array}
\]

where \( c \, p[x] \) gives \( \vdash p[x] = p'[x] \).

**See Also**  SIMPLE\_BINDER\_C

**Errors**

- 3011 ?0 is not of form: \( \Gamma \vdash \lambda \text{var} \cdot t^\perp \)
- 7104 Result of conversion, ?0, ill-formed

Also as the failure of the conversion.

\[
\text{val simple}_\lambda\text{eq\_rule} : \text{TERM} \to \text{THM} \to \text{THM};
\]

**Description**  Given an equational theorem, return the equation formed by abstracting the term argument (which must be a variable) from both sides.

\[
\frac{\Gamma \vdash t_1[x] = t_2[x]}{\Gamma \vdash (\lambda x \cdot t_1[x]) = (\lambda x \cdot t_2[x])}
\]

\[
\begin{array}{l}
simple_\lambda\text{eq\_rule} \\
\Gamma \vdash x^\perp
\end{array}
\]

A primitive inference rule.

**See Also**  \( \lambda \)\_eq\_rule

**Errors**

- 3007 ?0 is not a term variable
- 6005 ?0 occurs free in assumption list
- 6020 ?0 is not of the form: \( \Gamma \vdash t_1 = t_2^t \)
val string_conv : CONV;

**Description**  This function defines the constants with names starting with "", and type CHAR LIST (an abbreviation of CHAR LIST). A string literal constant is indicated by the constant name starting with a double quote("), as well as being of type 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 CHAR.

\[
\vdash \text{string_conv} \ (\text{mk_string} \ "c...") = \text{Cons} \ (\text{mk_char} \ "c") \text{string_conv} \ (\text{mk_string} \ "...")
\]

Or:

\[
\vdash \text{string_conv} \ (\text{mk_string} \ "") = \text{Nil}
\]

A primitive inference rule(axiom schemata).

**See Also**  mk_string

**Errors**  3025  ?0 is not a string literal

val strip_∧_rule : THM -> THM list;

**Description**  Break a theorem into conjuncts as far as possible.

\[
\Gamma \vdash t \quad \Rightarrow \quad [\Gamma \vdash t_1, ..., \Gamma \vdash t_n]
\]

where \( t \) can be formed from the \( t_i \) by \( \land \_intro \) alone, with no duplication, exception or reordering.

**Example**

\[
\vdash (a \land b) \land (a \land c \land d) \\
= \quad \vdash a', \vdash b', \vdash a', \vdash c', \vdash d'
\]

val strip_⇒_rule : THM -> THM;

**Description**  Repeatedly apply undisch_rule:

\[
\Gamma \vdash t_1 \Rightarrow ... \Rightarrow t_n \Rightarrow t \\
\Gamma \cup \{t_1, ..., t_n\} \vdash t
\]

\[\text{strip}_\Rightarrow \_\_rule\]
8.1. General Inference Rules

SML

val subst_conv : (THM * TERM) list -> TERM -> CONV;

Description  Substitution of equational theorems according to a template.

Conversion

\[
\frac{\Gamma_1 \cup \ldots \Gamma_n \vdash t[...;t_i,...] = t'[...;t_i',...]}{\text{subst_conv} \ \ [\ldots(\Gamma_i \vdash t_i = t_i', \Gamma_i \vdash x_i''), \ldots] \ \ r[t[...;x_i,...]]\vdash r[t'[...;t_i',...]]}
\]

\(\text{subst_conv} \ [(\text{thm}_1, x_1), \ldots, (\text{thm}_n, x_n)]\) 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

\[
\text{subst_conv} \ [(\vdash p = q', \Gamma x_1\gamma), (\vdash r = s', \Gamma x_2\gamma)]
\]

\[
(\forall y \cdot f x_1 r y + g x_2 p = h y\gamma)
\]

\[
(\forall x \cdot f p r x + g r p = h x\gamma)
\]

\[
\vdash (\forall x \cdot f p r x + g r p = h x) \leftrightarrow
\forall y \cdot f q r y + g s p = h y\gamma
\]

See Also  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: \(\vdash \Gamma t_1 = t_2\gamma\)

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.

\[
\frac{\Gamma \vdash t_1 = t_1', \ldots, \Gamma \vdash t_n = t_n'}{\Gamma \vdash t[t_1, \ldots, t_n]} \quad \text{subst_rule}
\]

\(\text{subst_rule} \ [(\text{thm}_1, x_1), \ldots, (\text{thm}_n, x_n)]\) 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  \(\text{subst_rule} \ [(\vdash p = q', \Gamma x_1'), (\vdash r = s', \Gamma x_2')]\)

\((\forall y \cdot f x_1 r y + g x_2 p = h y')\)

\((\vdash \forall x \cdot f p r x + g r p = h x')\)

=  \(\vdash \forall y \cdot f q r y + g s p = h y'\)

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: \(\vdash \Gamma t_1 = t_2'\)  
6029  Substitution list contains entry (?0, ?1) where the type of the variable differs from the type of the LHS of the theorem
8.1. General Inference Rules

\begin{verbatim}
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
\text{mk_app} have failures, then the offending term will be \text{refl_conved} instead). Thus:

\begin{align*}
\text{SUB_C1 cvn var} &= \text{fail_cvn var} \\
\text{SUB_C1 cvn const} &= \text{fail_cvn const} \\
\text{SUB_C1 cvn } (f \ x) &= \Gamma \vdash f \ x = f' \ x' \\
\text{where } \text{cvn } f &= \Gamma_1 \vdash f = f' \\
\text{and } \text{cvn } x &= \Gamma_2 \vdash x = x' \\
\text{and } \Gamma &= \Gamma_1 \cup \Gamma_2 \\
\text{SUB_C1 cvn } (\lambda x \bullet t) &= \Gamma \vdash (\lambda x \bullet t) = (\lambda x \bullet t') \\
\text{where } \text{cvn } t &= \Gamma \vdash t = t'
\end{align*}

\text{Errors}
7104  Result of conversion, \texttt{?0}, ill-formed
7105  \texttt{?0} has no constituents

There may be failure messages from the conversions.
\end{verbatim}

\begin{verbatim}
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:

\begin{align*}
\text{SUB_C cvn var} &= \text{refl_cvn var} \\
\text{SUB_C cvn const} &= \text{refl_cvn const} \\
\text{SUB_C cvn } (f \ x) &= \Gamma \vdash f \ x = f' \ x' \\
\text{where } \text{cvn } f &= \Gamma_1 \vdash f = f' \\
\text{and } \text{cvn } x &= \Gamma_2 \vdash x = x' \\
\text{and } \Gamma &= \Gamma_1 \cup \Gamma_2 \\
\text{SUB_C cvn } (\lambda x \bullet t) &= \Gamma \vdash (\lambda x \bullet t) = (\lambda x \bullet t') \\
\text{where } \text{cvn } t &= \Gamma \vdash t = t'
\end{align*}

\text{See Also}  \text{SUB_C1}
\end{verbatim}
\begin{verbatim}sml
val suc_conv : CONV;

Description  This conversion gives the definition schema for non-zero natural number literals.
\end{verbatim}

\begin{verbatim}
\begin{align*}
\tau \vdash \text{ML} (\text{mk\_N}(m+1)) = \text{Suc } \text{ML} (\text{mk\_N} m)
\end{align*}
\end{verbatim}

The conversion fails if given 0.

\begin{verbatim}
\begin{align*}
3026 \ ?0 \ is \ not \ a \ numeric \ literal \\
7100 \ ?0 \ must \ be \ numeric \ literal > 0
\end{align*}
\end{verbatim}

\textbf{See Also}  \texttt{mk\_N, prim\_suc\_conv}

\begin{verbatim}sml
val THEN\_CAN : (CANON * CANON) \rightarrow\ CANON

Description  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{verbatim}

\textbf{See Also}  \texttt{CANON}

\begin{verbatim}sml
val THEN\_C : (CONV * CONV) \rightarrow\ CONV;

Description  Combine the effect of two successful conversions.
\end{verbatim}

\begin{verbatim}
\begin{align*}
\Gamma \vdash t = t'' \\
(c1: \text{CONV}) \text{THEN\_C} (c2: \text{CONV}) \\
\tau \vdash t'
\end{align*}
\end{verbatim}

where \(c1\) returns \(\Gamma \vdash t = t''\), \(c2\) returns \(\Gamma \vdash t' = t''\), \(t\) and \(t''\) are \(\alpha\)-convertible and \(\Gamma\) equals \(\Gamma_1 \cup \Gamma_2\).

\textbf{See Also}  \texttt{EVERY\_C} (the iterated version of this function), as well as \texttt{THEN\_TRY\_C}, \texttt{AND\_OR\_C}, and \texttt{ORELSE\_C}

\begin{verbatim}
\begin{align*}
7101 \ Result \ of \ first \ conversion, \ ?0, \ not \ an \ equational \ theorem \\
7102 \ LHS \ (if \ any) \ of \ result \ of \ second \ conversion, \ ?0, \ not \ \alpha-convertible \ to \ RHS \ of \ first, \ ?1
\end{align*}
\end{verbatim}

\textbf{Errors}  If any, as the failures of \(c1\) and \(c2\) applied to \(t\) and \(t'\) respectively.

\begin{verbatim}sml
val THEN\_LIST\_CAN : (CANON * CANON list) \rightarrow\ CANON

Description  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{verbatim}

\textbf{See Also}  \texttt{CANON}

\begin{verbatim}
\begin{align*}
26204 \ wrong \ number \ of \ canonicalisation \ functions \ in \ the \ list
\end{align*}
\end{verbatim}
### 8.1. General Inference Rules

<table>
<thead>
<tr>
<th>SML</th>
<th>val THEN TRY C : (CONV * CONV) --&gt; CONV;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>See Also</strong></td>
<td>THEN C, AND OR C, ORELSE C</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>As the failure of c1.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>val TOP MAP C : CONV --&gt; CONV;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>TOP 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 the conversion. 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 TOP 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.</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>7005 Conversion fails on term and all its subterms</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>val TRY C : CONV --&gt; CONV;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Attempt to apply a conversion, and if it fails, apply refl conv.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>val t_thm : THM;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>“True” is true.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>val undisch_rule : THM --&gt; THM ;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Undischarge the antecedent of an implicative theorem into the assumption list.</td>
</tr>
<tr>
<td><strong>Rule</strong></td>
<td>Γ ⊢ a ⇒ b</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>7011 ?0 is not of the form: ‘Γ ⊢ a ⇒ b’</td>
</tr>
</tbody>
</table>
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  

\[ \Gamma \vdash t[x,y,...] \quad \Gamma \vdash \exists vs[x,y,...]\bullet t[x,y,...] \quad v_\exists_intro \quad \Gamma' \vdash vs[x,y,...]\]

where \( \Gamma' \vdash vs[x,y,...]\) is a varstruct built from variables \( \Gamma x \), \( \Gamma y \), etc, which may contain duplicates.

Uses  If the functionality is sufficient, this is superior in efficiency to both \( \exists \) intro and simple_∃_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  

\[ \Gamma \vdash t1 \iff t2 \quad \Gamma \vdash t1 \implies t2; \quad \Gamma \vdash t2 \implies t1 \quad \iff\_elim \]

Errors  7062  ?0 is not of the form: \( \Gamma \vdash t1 \iff t2 \)

7064  ?0 and ?1 are not of the form: \( \Gamma \vdash t1 \implies t2; \quad \Gamma \vdash t2a \implies t1a \)

where \( \Gamma t1 \), \( \Gamma t1a \), \( \Gamma t2 \), and \( \Gamma t2a \), are \( \alpha \)-convertible

val ⇔_intro : THM -> THM -> THM;

Description  Join two implicative theorems into an bi-implicative theorem.

Rule  

\[ \Gamma1 \vdash t1 \implies t2; \quad \Gamma2 \vdash t1' \implies t2' \quad \implies\_intro \quad \Gamma1 \cup \Gamma2 \vdash t1 \iff t2 \]

where \( t1 \) and \( t1' \) are \( \alpha \)-convertible, as are \( t2 \) and \( t2' \).

Errors  7040  ?0 is not of the form: \( \Gamma \vdash t1 \implies t2 \)

7064  ?0 and ?1 are not of the form: \( \Gamma \vdash t1 \implies t2; \quad \Gamma \vdash t2a \implies t1a \)

where \( \Gamma t1 \), \( \Gamma t1a \), \( \Gamma t2 \), and \( \Gamma t2a \), are \( \alpha \)-convertible

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
8.1. General Inference Rules

**SML**

\[
\text{val} \equiv \texttt{match_mp_rule} : \text{THM} \rightarrow \text{THM} \rightarrow \text{THM};
\]

**Description**  
A matching Modus Ponens for \(\equiv\).

**Rule**  
\[
\begin{array}{c}
\Gamma_1 \vdash \forall x_1 \ldots \bullet t_1 \equiv t_2; \quad \Gamma_2 \vdash t_1' \\
\Gamma_1' \cup \Gamma_2 \vdash t_2'
\end{array}
\]

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 \(\Rightarrow\)), \texttt{simple}_\equiv_\texttt{match_mp_rule} \(\Rightarrow\_\text{mp_rule} \equiv \texttt{match_mp_rule1}\)

**Errors**  
7044 Cannot match ?0 and ?1

---

**SML**

\[
\text{val} \equiv \texttt{match_mp_rule1} : \text{THM} \rightarrow \text{THM} \rightarrow \text{THM};
\]

**Description**  
A matching Modus Ponens for \(\equiv\) that doesn’t affect assumption lists.

**Rule**  
\[
\begin{array}{c}
\Gamma_1 \vdash \forall x_1 \ldots \bullet t_1 \equiv t_2; \quad \Gamma_2 \vdash t_1' \\
\Gamma_1' \cup \Gamma_2 \vdash t_2'
\end{array}
\]

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 \(\Rightarrow\)), \texttt{simple}_\equiv_\texttt{match_mp_rule1}\)

**Errors**  
7044 Cannot match ?0 and ?1

---

**SML**

\[
\text{val} \equiv \texttt{mp_rule} : \text{THM} \rightarrow \text{THM} \rightarrow \text{THM};
\]

**Description**  
This is reminiscent of Modus Ponens, but upon bi-implicative theorems.

**Rule**  
\[
\begin{array}{c}
\Gamma_1 \vdash t_1 \equiv t_2; \quad \Gamma_2 \vdash t_1' \\
\Gamma_1 \cup \Gamma_2 \vdash t_2
\end{array}
\]

where \(t_1\) and \(t_1'\) must be \(\alpha\)-convertible.

A built-in inference rule.

**See Also**  
\(\Rightarrow\_\text{elim}\) (true Modus Ponens, on \(\Rightarrow\)), \texttt{match_mp_rule} (a “matching” version of \(\equiv\_\texttt{mp_rule}\))

**Errors**  
6024 ?0 and ?1 are not of the form: \(\Gamma \vdash t_1 \equiv t_2'\) and \(\Gamma \vdash t_1''\) where \(\Gamma t_1\) and \(\Gamma t_1''\) are \(\alpha\)-convertible
6030 ?0 is not of the form: \(\Gamma \vdash t_1 \equiv t_2'\)

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
Chapter 8. PROOF IN HOL

val \_\_t\_elim : THM \rightarrow THM;

**Description** We can always eliminate \( \iff T \).

**Rule**

\[
\frac{\Gamma \vdash t \iff T}{\Gamma \vdash t} \iff\_t\_elim
\]

**Errors**

7106 ?0 not of the form \( \Gamma \vdash t \iff T \)

val \_\_t\_intro : THM \rightarrow THM;

**Description** The conclusion of a theorem is equal to \( T \).

**Rule**

\[
\frac{\Gamma \vdash t}{\Gamma \vdash t \iff T} \iff\_t\_intro
\]

\val \_\_intro : THM \rightarrow THM \rightarrow THM;

**Description** Conjoin two theorems.

**Rule**

\[
\frac{\Gamma_1 \vdash t_1; \Gamma_2 \vdash t_2}{\Gamma_1 \cup \Gamma_2 \vdash t_1 \land t_2} \land\_intro
\]

\val \_\_left\_elim : THM \rightarrow THM;

**Description** Give the left conjunct of a conjunction.

**Rule**

\[
\frac{\Gamma \vdash t_1 \land t_2}{\Gamma \vdash t_1} \land\_left\_elim
\]

**Errors**

7007 ?0 is not of the form \( \Gamma \vdash t_1 \land t_2 \)
8.1. General Inference Rules

SML

val ∧_rewrite_canon : THM -> THM list
val simple_¬_rewrite_canon : THM -> THM list
val ⇔_t_rewrite_canon : THM -> THM list
val f_rewrite_canon : THM -> THM list
val simple_∀_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 perform the following transformations:

| ∧_rewrite_canon          | (Γ ⊢ t1 ∧ t2)       | = Γ ⊢ t1 ; Γ ⊢ t2 |
| simple_¬_rewrite_canon  | (Γ ⊢ ¬(t1 ∨ t2))   | = (Γ ⊢ ¬t1 ∧ ¬t2) |
| simple_¬_rewrite_canon  | (Γ ⊢ ¬x•t)         | = (Γ ⊢ ∀x•¬t)     |
| simple_¬_rewrite_canon  | (Γ ⊢ ¬¬t)          | = (Γ ⊢ ¬t)        |
| ⇔_t_rewrite_canon      | (Γ ⊢ t1 = t2)      | = < failure >     |
| f_rewrite_canon        | (Γ ⊢ F)            | = (Γ ⊢ ∀x•t)      |

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  ¬_rewrite_canon, ∀_rewrite_canon.

Errors

26203 the conclusion of the theorem is already an equation

SML

val ∧_right_elim : THM -> THM;

Description  Give the right conjunct of a conjunction.

Rule

\[
\begin{array}{c}
Γ ⊢ t1 ∧ t2 \\
\end{array}
\Rightarrow
\begin{array}{c}
Γ ⊢ t2 \\
\end{array}
\]

∧_right_elim

Errors

7007  ?0 is not of the form: 'Γ ⊢ t1 ∧ t2'

SML

val ∧_thm : THM;

Description  Expanded form of definition of ∧

Theorem

\[
\begin{array}{c}
∀ t1 t2 • (t1 ∧ t2) ⇔ \\
(∀ b • (t1 ⇒ t2 ⇒ b) ⇒ b) \\
\end{array}
\]

∧_thm

SML

val ∧_⇒_rule : THM -> THM;

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

\[
\begin{array}{c}
Γ ⊢ (a ∧ b) ⇒ c \\
\end{array}
\Rightarrow
\begin{array}{c}
Γ ⊢ a ⇒ b ⇒ c \\
\end{array}
\]

∧_⇒_rule

Errors

7009  ?0 is not of the form: 'Γ ⊢ (a ∧ b) ⇒ c'


### SML

**val \_\_cancel\_rule : THM \rightarrow THM \rightarrow THM;**

**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)

\[
\begin{array}{c}
\Gamma_1 \vdash t_1 \lor t_2; \Gamma_2 \vdash \neg t_1' \\
\hline
\Gamma_1 \cup \Gamma_2 \vdash t_2
\end{array}
\]

\_\_cancel\_rule

And:

\[
\begin{array}{c}
\Gamma_1 \vdash t_1 \lor t_2; \Gamma_2 \vdash \neg t_2' \\
\hline
\Gamma_1 \cup \Gamma_2 \vdash t_1
\end{array}
\]

\_\_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 \(\Gamma_2 \vdash \neg t_3\)

where \(\Gamma t_3\) is \(\alpha\)-convertible to \(\Gamma t_1\) or \(\Gamma t_2\)

### SML

**val \_\_elim : THM \rightarrow THM \rightarrow THM \rightarrow THM;**

**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.

\[
\begin{array}{c}
\Gamma_1 \vdash t_1 \lor t_2 \\
\Gamma_2, t_1' \vdash t \\
\hline
\Gamma_3, t_2' \vdash t' \\
\Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \vdash t
\end{array}
\]

\_\_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 \(\Gamma t_1\) and \(\Gamma t_1a\), \(\Gamma t_2\) and \(\Gamma t_2a\),

\(\Gamma t_3\) and \(\Gamma t_3a\) are each \(\alpha\)-convertible

### SML

**val \_\_left\_intro : TERM \rightarrow THM \rightarrow THM;**

**Description**  Introduce a disjunct to the left of a theorem’s conclusion.

\[
\begin{array}{c}
\Gamma \vdash b \\
\hline
\Gamma \vdash a \lor b
\end{array}
\]

\_\_left\_intro

\(\Gamma a\)

**Errors**

3031  ?0 is not of type \(\Gamma : BOOL\)
8.1. General Inference Rules

**Standard ML**

\[
\text{val} \ \\ \ \textit{∨\_right\_intro} : \text{TERM} \rightarrow \text{THM} \rightarrow \text{THM};
\]

**Description** Introduce a disjunct to the right of a theorem’s conclusion.

**Rule**

\[
\begin{array}{c}
\Gamma \vdash b \\
\hline
\Gamma \vdash b \lor a
\end{array}
\]

**Errors**

3031 \ ?0 is not of type \(\text{\textasciitilde}{}:\text{BOOL}\)

**Standard ML**

\[
\text{val} \ \ \ \ \textit{∨\_thm} : \text{THM};
\]

**Description** Expanded form of definition of \(\lor\)

**Theorem**

\[
\forall t1\ t2 . (t1 \lor t2) \iff (\forall b . (t1 \Rightarrow b) \Rightarrow (t2 \Rightarrow b) \Rightarrow b)
\]

**Standard ML**

\[
\text{val} \ \ \ \ \textit{¬\_elim} : \text{TERM} \rightarrow \text{THM} \rightarrow \text{THM} \rightarrow \text{THM};
\]

**Description** Given two contradictory theorems with the same assumptions, conclude any other fact from the assumptions: input theorems may be in either order.

**Rule**

\[
\begin{array}{c}
\Gamma1 \vdash a ; \Gamma2 \vdash \neg a \\
\hline
\Gamma1 \cup \Gamma2 \vdash b
\end{array}
\]

**Errors**

3031 \ ?0 is not of type \(\text{\textasciitilde}{}:\text{BOOL}\)

7004 \ ?0 and ?1 are not of the form: \(\Gamma1 \vdash a\) and \(\Gamma2 \vdash \neg a\)

**Standard ML**

\[
\text{val} \ \ \ \ \textit{¬\_eq\_sym\_rule} : \text{THM} \rightarrow \text{THM};
\]

**Description** If \(a\) is not equal to \(b\) then \(b\) is not equal to \(a\).

**Rule**

\[
\begin{array}{c}
\Gamma \vdash \neg(a = b) \\
\hline
\Gamma \vdash \neg(b = a)
\end{array}
\]

**Errors**

7091 \ ?0 is not of form: \(\Gamma \vdash \neg(a = b)\)

**Standard ML**

\[
\text{val} \ \ \ \ \textit{¬\_intro} : \text{TERM} \rightarrow \text{THM} \rightarrow \text{THM} \rightarrow \text{THM};
\]

**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).

**Rule**

\[
\begin{array}{c}
\Gamma1 \vdash b ; \Gamma2 \vdash \neg b \\
\hline
(\Gamma1 \cup \Gamma2) \setminus \{a\} \vdash \neg a
\end{array}
\]

**Errors**

3031 \ ?0 is not of type \(\text{\textasciitilde}{}:\text{BOOL}\)

7004 \ ?0 and ?1 are not of the form: \(\Gamma1 \vdash a\) and \(\Gamma2 \vdash \neg a\)
### Chapter 8. PROOF IN HOL

#### Description

Move \(\neg\) into a \(\forall\) construct.

\[
\Gamma \vdash (\neg (\forall x \cdot t[x])) \iff \exists x \cdot \neg t[x]
\]

This will work with any simple universal quantifier.

Errors

7036  ?0 not of the form: \(\neg (\forall x \cdot t[x])\)

#### Description

Move \(\neg\) into an \(\exists\) construct.

\[
\Gamma \vdash (\neg (\exists x \cdot t[x])) \iff \forall x \cdot \neg t[x]
\]

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

#### Description

“Not t if and only if t is false.”

\[
\forall t \cdot (\neg t) \iff (t \equiv F)
\]

#### Description

Expanded form of definition of \(\neg\):

\[
\forall t \cdot (\neg t) \iff (t \Rightarrow F)
\]

#### Description

“Not true is false”.

\[
\neg T \iff F
\]

#### Description

A double negation is redundant.

\[
\Gamma \vdash (\neg (\neg t)) \iff t
\]

Errors

7022  ?0 is not of the form: \(\neg (\neg t)\)
### 8.1. General Inference Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Gamma \vdash \neg (\neg t)$</td>
<td>A double negation is redundant.</td>
<td>7006</td>
</tr>
<tr>
<td>$\Gamma \vdash t$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### SML

<table>
<thead>
<tr>
<th>val</th>
<th>Description</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg \neg \text{elim}$</td>
<td>A double negation is redundant.</td>
<td>7006</td>
</tr>
<tr>
<td>$\forall \ x \cdot t[x]$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### SML

<table>
<thead>
<tr>
<th>val</th>
<th>Description</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg \neg \text{intro}$</td>
<td>We may always introduce a double negation.</td>
<td>27019</td>
</tr>
<tr>
<td>$\forall \ x \cdot t[x]$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### SML

<table>
<thead>
<tr>
<th>val</th>
<th>Description</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg \forall \text{conv}$</td>
<td>Move $\neg$ into a $\forall$ construct.</td>
<td>27019</td>
</tr>
<tr>
<td>$\exists x \cdot t[x]$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### SML

<table>
<thead>
<tr>
<th>val</th>
<th>Description</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg \exists \text{conv}$</td>
<td>Move $\neg$ into an $\exists$ construct.</td>
<td>27020</td>
</tr>
<tr>
<td>$\exists x \cdot t[x]$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### SML

<table>
<thead>
<tr>
<th>val</th>
<th>Description</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg \forall \text{thm}$</td>
<td>Used in pushing negations through simple universal quantifications.</td>
<td></td>
</tr>
<tr>
<td>$\forall p \cdot \neg \forall p \iff \exists x \cdot \neg p x$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### SML

<table>
<thead>
<tr>
<th>val</th>
<th>Description</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg \exists \text{conv}$</td>
<td>Move $\neg$ into an $\exists$ construct.</td>
<td>27020</td>
</tr>
<tr>
<td>$\exists x \cdot t[x]$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### SML

<table>
<thead>
<tr>
<th>val</th>
<th>Description</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg \exists \text{conv}$</td>
<td>Move $\neg$ into an $\exists$ construct.</td>
<td>27020</td>
</tr>
<tr>
<td>$\exists x \cdot t[x]$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
Theorem

\[ \forall p \cdot \neg \exists p \iff (\forall x \cdot \neg p x) \]

\(\neg \exists\_thm\)

### Description

Used in pushing negations through simple existential quantifications.

### SML

```sml
val \_\_\_thm : THM;
```

### Description

Modus Ponens (which is why we introduce the alias \_\_mp_rule, though \_\_elim is shorter, conventional, and the preferred name).

### Rule

\[
\frac{\Gamma_1 \vdash t_1 \Rightarrow t_2; \Gamma_2 \vdash t_1'}{\Gamma_1 \cup \Gamma_2 \vdash t_2} \quad \Rightarrow\_elim
\]

where \(t_1\) and \(t_1'\) must be \(\alpha\)-convertible. A primitive inference rule.

### See Also

\(\iff\_mp\_rule\) (Modus Ponens on \(\iff\)), \(\iff\_match\_mp\_rule\) (a “matching” version of this function).

### Errors

6010 0 is not of the form: ‘\(\Gamma \vdash t_1 \Rightarrow t_2\)’

6011 0 and 1 are not of the forms: ‘\(\Gamma_1 \vdash t_1 \Rightarrow t_2\)’ and ‘\(\Gamma_2 \vdash t_1'\)’ where ‘\(t_1\)’ and ‘\(t_1'\)’ are \(\alpha\)-convertible

### SML

```sml
val \_\_intro : TERM -> THM -> THM;
```

### Description

Prove an implicative theorem, removing, if \(\alpha\)-convertibly present, the antecedent of the implication from the assumption list.

### Rule

\[
\frac{\Gamma \vdash t_2}{\Gamma - \{t_1\} \vdash t_1 \Rightarrow t_2} \quad \Rightarrow\_intro
\]

A primitive inference rule.

### See Also

\(\disch\_rule\) (which fails if term not in assumption list)

### Errors

3031 0 is not of type ‘\(\:\:BOOL\)’
8.1. General Inference Rules

\[
\text{val} \Rightarrow \text{match_mp_rule} : \text{THM} \rightarrow \text{THM} \rightarrow \text{THM} ;
\]

**Description**  
A matching Modus Ponens rule for an implicative theorem.

\[
\frac{\Gamma_1 \vdash \forall x_1 \ldots \bullet t_1 \Rightarrow t_2 ; \Gamma_2 \vdash t_1'}{\Gamma_1' \cup \Gamma_2 \vdash t_2'} \Rightarrow \text{match_mp_rule}
\]

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

\[
\text{val} \Rightarrow \text{match_mp_rule1 : THM} \rightarrow \text{THM} \rightarrow \text{THM} ;
\]

\[
\text{val} \Rightarrow \text{match_mp_rule2 : THM} \rightarrow \text{THM} \rightarrow \text{THM} ;
\]

**Description**  
Two variants of a matching Modus Ponens rule for an implicative theorem.

\[
\frac{\Gamma_1 \vdash \forall x_1 \ldots \bullet t_1 \Rightarrow t_2 ; \Gamma_2 \vdash t_1'}{\Gamma_1' \cup \Gamma_2 \vdash t_2'} \Rightarrow \text{match_mp_rule}
\]

where \(t1'\) is an instance of \(t1\) under type instantiation and substitution for the \(x_i\) and the free variables of the first theorem, and where \(t2'\) is the corresponding instance of \(t2\). The type instantiations and substitutions are obtained by matching \(t1\) and \(t1'\) using \text{term_match}.

\Rightarrow \text{match_mp_rule2} \text{ is just like } \Rightarrow \text{match_mp_rule1 except that the instantiations and substitutions returned by term_match are extended to replace type variables that do not occur in } t1 \text{ or in } \Gamma 1 \text{ and } x_{-i} \text{ that do not occur free in } t1 \text{ by fresh variables to avoid clashes with each other and with the type variables and free variables of } \Gamma 1 \text{ 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 \(\Gamma \vdash \forall x_1 \ldots \bullet u \Rightarrow v\)

\[
\text{val} \Rightarrow \text{trans_rule} : \text{THM} \rightarrow \text{THM} \rightarrow \text{THM} ;
\]

**Description**  
Transitivity of \(\Rightarrow\).

\[
\frac{\Gamma_1 \vdash t_1 \Rightarrow t_2 ; \Gamma_2 \vdash t_2' \Rightarrow t_3}{\Gamma_1 \cup \Gamma_2 \vdash t_1 \Rightarrow t_3} \Rightarrow \text{trans_rule}
\]

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: \(\Gamma 1 \vdash t_1 \Rightarrow t_2\) and \(\Gamma 2 \vdash t_2 \Rightarrow t_3\)  
where \(\Gamma t_2\) and \(\Gamma t_2\) are \(\alpha\)-convertible
\textbf{\texttt{val \_\_\_rule : THM \to THM}};

\textbf{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*}
\frac{\Gamma \vdash a \Rightarrow b \Rightarrow c}{\Gamma \vdash (a \land b) \Rightarrow c} \Rightarrow \_\_\_rule
\end{align*}

\noindent\textbf{Errors} \(7008\) \(?0\ \text{is not of the form:} \ '\Gamma \vdash a \Rightarrow b \Rightarrow c'\)

\textbf{\texttt{val \_\_arb\_elim : THM \to THM}};

\textbf{Description} Specialise a universally quantified theorem with a machine generated variable or variable structure.

\begin{align*}
\frac{\Gamma \vdash \forall \; vs[x,y,...]\bullet p[x,y,...]}{\Gamma \vdash p[x',y',...]} \forall\_arb\_elim
\end{align*}

where \(x', y',\) etc, are not variables (free or bound) in \(p\) or \(\Gamma\), created by \texttt{gen\_vars}(q.v).

\textbf{See Also} \(\forall\_elim\)

\noindent\textbf{Errors} \(27011\) \(?0\ \text{is not of the form:} \ '\Gamma \vdash \forall \cdot t'\) where \(\langle x\rangle\) is a varstruct

\textbf{\texttt{val \_\_asm\_rule : TERM \to TERM \to THM \to THM}};

\textbf{Description} Generalise an assumption (Left \(\forall\) introduction).

\begin{align*}
\frac{\Gamma, p'[x] \vdash q[x]}{\Gamma, \forall \cdot p'[x] \vdash q[x]} \forall\_asm\_rule
\end{align*}

where \(p\) and \(p'\) are \(\alpha\)-convertible. \(x\) may be free in \(\Gamma\). The function will work even if \(p'[x]\) is not present in the assumption list.

\noindent\textbf{Errors} \(4016\) \(?0\ \text{is not an allowed variable structure}\)

\noindent© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
8.1. General Inference Rules

SML

```sml
val ∀_elim : TERM -> THM -> THM;
```

**Description**
Specialise a universally quantified theorem with a given value, instantiating the type of the theorem as necessary.

**Rule**

\[
\begin{array}{c}
\Gamma \vdash ∀\,x \cdot t_2[x] \\
\Gamma \vdash t_2'[t]\end{array} \quad ∀_elim
\begin{array}{c}
\Gamma \vdash t' \\Gamma
\end{array}
\]

where \(t_2\)' is renamed from \(t_2\) to prevent bound variable capture and possibly type instantiated, and \(x\) is a varstruct, instantiable to the structure of \(t_1\). The value \(t_1\) will be expanded using \(Fst\) and \(Snd\) as necessary to match the structure of \(⌜x⌝\).

**See Also**
list_∀_elim, all_∀_elim.

**Errors**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>27011</td>
<td>?0 is not of the form: ('\Gamma \vdash ∀,x \cdot t') where (⌜x⌝) is a varstruct</td>
</tr>
<tr>
<td>27012</td>
<td>?0 is not of the form: ('\Gamma \vdash ∀,x \cdot t') where the type of (⌜x⌝) is instantiable to the type of ?1</td>
</tr>
<tr>
<td>27013</td>
<td>?0 is not of the form: ('\Gamma \vdash ∀,x \cdot t') where the type of (⌜x⌝) is instantiable to the type of ?1 without instantiating type variables in the assumptions</td>
</tr>
</tbody>
</table>

---

SML

```sml
val ∀_intro : TERM -> THM -> THM;
```

**Description**
Introduce a universally quantified theorem.

**Rule**

\[
\begin{array}{c}
\Gamma \vdash t \\Gamma
\end{array} \quad ∀_intro
\begin{array}{c}
\Gamma \vdash ∀\,x \cdot t \\
\Gamma \vdash t' \\Gamma
\end{array}
\]

Where \(⌜x'⌝\) is an allowed variable structure based on \(⌜x⌝\), but with duplicate variables renamed, the original name being rightmost in the resulting variable structure.

**See Also**
list_∀_intro, all_∀_intro.

**Errors**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4016</td>
<td>?0 is not an allowed variable structure</td>
</tr>
<tr>
<td>6005</td>
<td>?0 occurs free in assumption list</td>
</tr>
</tbody>
</table>
val \_reorder_conv : TERM \rightarrow CONV;

**Description**  Reorder universal quantifications.

\[
\begin{align*}
(\forall y_1 .. y_m \cdot t_2) & \iff (\forall x_1 .. x_n \cdot t_1) \\
(\forall y_1 .. y_m \cdot t_2) & \iff (\forall x_1 .. x_n \cdot t_1)
\end{align*}
\]

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:**
\[
(\forall (x, q) \cdot x \land z) \iff (\forall (y, z) \cdot x \lor q) :\ THM
\]

Note that before more sophisticated attempts, the conversion will try \$\alpha\_\_conv\$ on the two term arguments.

**See Also**  \_\_reorder_conv

**Errors**  27050 Cannot prove equality of \(?0\) and \(?1\)

---

val \_uncurry_conv : CONV;

**Description**  Convert a paired universally quantified term into simple universal quantifications of the same term.

Conversion
\[
\begin{align*}
\Gamma \vdash \forall vs[x, y, \ldots] \cdot f[x, y, \ldots] = \\
\forall x y \ldots \cdot f[x, y, \ldots]
\end{align*}
\]

where \(vs[x, y, \ldots]\) is an allowed variable structure with variables \(x, y, \ldots\). It may not be a simple variable.

**See Also**  \_\_varstruct_conv, all\_\_uncurry_conv.

**Errors**  27038 \(?0\) is not of the form: \(\forall (x, y) \cdot f\)

---

val \_\_\_rule : TERM \rightarrow THM \rightarrow THM;

**Description**  Universally quantify a variable on both sides of an equivalence.

\[
\begin{align*}
\Gamma \vdash p[x] \iff q[x] & \iff (\forall x \cdot p[x]) \iff (\forall x \cdot q[x]) \\
\end{align*}
\]

where \(x\) is a varstruct.

**Errors**  6005 \(?0\) occurs free in assumption list

6020 \(?0\) is not of the form: \(\Gamma \vdash t_1 = t_2\)

7062 \(?0\) is not of the form: \(\Gamma \vdash t_1 \iff t_2\)

4016 \(?0\) is not an allowed variable structure
8.1. General Inference Rules

\[\text{val } \exists\text{._asm.rule : TERM }\rightarrow\text{ TERM }\rightarrow\text{ THM }\rightarrow\text{ THM};\]

**Description**  Existentially quantify an assumption (Left \(\exists\) introduction).

\[\frac{\Gamma, p'[x] \vdash q}{\Gamma, \exists x \cdot p'[x] \vdash q} \exists\text{._asm.rule} \quad \Gamma, \exists x \cdot p'[x] \vdash q \quad \Gamma, p[x] \vdash \nabla^r \]

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 \(\Gamma:\text{BOOL}\nabla^r\)
- 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”.

\[\text{val } \exists\text{._elim : TERM }\rightarrow\text{ THM }\rightarrow\text{ THM }\rightarrow\text{ THM};\]

**Description**  Eliminate an existential quantifier by reference to an arbitrary varstruct satisfying the predicate.

\[\frac{\Gamma 1 \vdash \exists vs[x1,x2,...] \cdot t1[x1,x2,...]; 
\quad \Gamma 2, t1[y1,y2,...] \vdash t2 \quad \exists\text{._elim} \quad \Gamma 1 \cup \Gamma 2 \vdash t2}{\exists vs[y1,y2,...]\nabla} \]

\(t1[y1,y2,...]\) 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:

\[\frac{\Gamma 1 \vdash \exists (p,q) \cdot t1[p,q]; 
\quad \Gamma 2, t1[\text{Fst } x, \text{Snd } x] \vdash t2 \quad \exists\text{._elim} \quad \Gamma 1 \cup \Gamma 2 \vdash t2}{\exists x \nabla} \]

**Errors**
- 27042  ?0 does not match the bound varstruct of ?1
- 27046  ?0 is not of the form ‘\(\Gamma \vdash \exists vs\cdot t\’
- 27051  ?0 has members appearing free in conclusion of ?1
- 27052  ?0 has members appearing free in ?1 other than in assumption ?2

\[\text{val } \exists\text{._intro.thm : THM;}\]

**Description**  Introduction of existential quantification.

\[\frac{\vdash \forall P \; x \cdot P\; x \Rightarrow \exists P}{\exists\text{._intro.thm}} \]

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
**SML**

```sml
val ∃_intro : TERM -> THM -> THM ;
```

**Description**  Introduce an existential quantifier by reference to a witness.

**Rule**  
\[ \Gamma \vdash t \]  
\[ \Gamma \vdash \exists \text{vs}[x', y', \ldots] \bullet t[x, y', \ldots] \]  
\[ ∃ \text{intro} \]  
\[ \exists vs[x, y, \ldots] \bullet t[x, y, \ldots] \]

where `"vs[x, y, \ldots]"` is varstruct built from variables `"x"`, `"y"`, etc, and the `"x'"` are renamed if duplicated inside the varstruct, all but the rightmost being so renamed.

**Errors**  
7047  ?0 cannot be matched to conclusion of theorem ?1

**SML**

```sml
val ∃_reorder_conv : TERM -> CONV ;
```

**Description**  Reorder existential quantifications.

**Rule**  
\[ (\exists y_1 \ldots y_m \bullet t_2) \iff (\exists x_1 \ldots x_n \bullet t_1) \]  
\[ \exists \text{reorder_conv} \]  
\[ \exists y_1 \ldots y_m \bullet t_2 \]

where the `x_1` and `y_1` 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**  
\[ \exists \text{reorder_conv} \exists (x, q) z \bullet x \land z \iff \exists (x, y) \bullet x \land z : \text{THM} \]

**See Also**  `∀_reorder_conv`

**Errors**  
27050  Cannot prove equality of ?0 and ?1

**SML**

```sml
val ∃_uncurry_conv : CONV ;
```

**Description**  Convert a paired existentially quantified term into simple universal quantifications of the same term.

**Conversion**  
\[ \vdash \exists vs[x, y, \ldots] \bullet f[x, y, \ldots] = \exists x \ y \ldots f[x, y, \ldots] \]  
\[ ∃ \text{uncurry_conv} \]  
\[ \exists vs[x, y, \ldots] \bullet f[x, y, \ldots] \]

where `vs[x, y, \ldots]` is an allowed variable structure with variables `x, y, \ldots`. It may not be a simple variable.

**See Also**  `λ_varstruct_conv`, `all_∃_uncurry_conv`, `∀_uncurry_conv`.

**Errors**  
27047  ?0 is not of the form: `"∃ (x,y) \bullet f"`
8.1. General Inference Rules

\[
\begin{align*}
\text{val } \exists_\epsilon_{.\ conv} : & \text{ CONV}; \\
\text{Description } & \text{Give that } \epsilon \text{ of a predicate satisfies the predicate by reference to an } \exists \text{ construct. It can properly handle paired existence.} \\
\text{Rule } & \frac{\Gamma \vdash (\exists x \bullet p(x)) = p(\epsilon x \bullet p x)}{\exists_\epsilon_{.\ conv} \quad \exists x \bullet p[x] \neg}
\end{align*}
\]

If \( x \) is formed by paired then the \( \text{Fst} \) and \( \text{Snd} \) are used to extract the appropriate bits of the \( \epsilon \)-term for distribution in \( p[\epsilon x \bullet p x] \).

\textbf{See Also } \exists_\epsilon_{.\ rule}

\textbf{Errors} 27024 „\( \exists \) is not of the form: ‘\( \Gamma \vdash x \bullet p[x] \)’

\textit{where } \( x \) is a varstruct

\[
\begin{align*}
\text{val } \exists_\epsilon_{.\ rule} : & \text{ THM } \rightarrow \text{ THM}; \\
\text{Description } & \text{Give that } \epsilon \text{ of a predicate satisfies the predicate by reference to an } \exists \text{ construct. It can properly handle paired existence.} \\
\text{Rule } & \frac{\Gamma \vdash \exists x \bullet p[x]}{\exists_\epsilon_{.\ rule} \quad \exists x \bullet p[x]}
\end{align*}
\]

If \( x \) is formed by paired then the \( \text{Fst} \) and \( \text{Snd} \) are used to extract the appropriate bits of the \( \epsilon \)-term for distribution in \( p[\epsilon x \bullet p x] \).

\textbf{See Also } \exists_\epsilon_{.\ conv}

\textbf{Errors} 27024 „\( \exists \) is not of the form: ‘\( \Gamma \vdash x \bullet p[x] \)’

\textit{where } \( x \) is a varstruct

\[
\begin{align*}
\text{val } \exists_1_{.\ conv} : & \text{ CONV}; \\
\text{Description } & \text{This is a conversion which turns a unique existential quantifier into an equivalent existential quantifier} \\
\text{Conversion } & \frac{\vdash (\exists_1 \text{vs}[x1,...] \bullet t[x1,...]) \leftrightarrow}{\exists_1_{.\ conv} \quad \exists_1 \text{vs}[x1,...] \bullet t[x1,...] \neg}
\end{align*}
\]

\( \forall \text{vs}[x1',...], t[x1',...] \Rightarrow \text{vs}[x1',...] = \text{vs}[x1,...] \)

\textbf{Uses } Tactic and conversion programming.

\textbf{See Also } \textit{strip_tac}, \textit{simple}_1_{.\ conv}

\textbf{Errors} 27053 „\( \exists \) is not of the form: ‘\( \exists_1 \text{vs} \bullet t \)’

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
val \exists_1\_elim : \text{THM} \rightarrow \text{THM};

\textbf{Description} \quad \text{Express a } \exists_1 \text{ in terms of } \exists \text{ and a uniqueness property.}

\begin{align*}
\Gamma \vdash \exists_1 \, \forall \, \text{vs}[a,b,...] \bullet P[a,b,...] \\
\Gamma \vdash \exists \, \forall \, \text{vs}[a,b,...] \bullet P[a,b,...] \land \\
\forall \, \text{vs}[a',b',...] \bullet P[x1,x2,...] \\
\Rightarrow \\
\text{vs}[a',b',...] = \text{vs}[a,b,...]
\end{align*}

where the \(a',\) etc, are variants of the \(a.\)

\textbf{Errors} \quad 27022 \quad ?0 \text{ is not of the form: } "\Gamma \vdash \exists_1 \, x \bullet P[x]^t" \quad \text{where } \forall \, \neg x_\neg \text{ is a varstruct}

val \exists_1\_intro : \text{THM} \rightarrow \text{THM} \rightarrow \text{THM};

\textbf{Description} \quad \text{Introduce } \exists_1 \text{ by reference to a witness, and a uniqueness theorem.}

\begin{align*}
\Gamma 1 \vdash P'[t'] \\
\Gamma 2 \vdash \forall \, x \bullet P[x] \Rightarrow x = t \\
\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} \quad 27021 \quad ?0 \text{ and } ?1 \text{ are not of the form: } "\Gamma 1 \vdash Pa[ta]^t\" \quad \text{and} \quad "\Gamma 2 \vdash \forall \, \text{vs}[x,y,...] \bullet P[x,y,...] \Rightarrow \text{vs}[x,y,...] = t\" \quad \text{where } \forall \, \neg Pa_\neg \text{ and } \forall \, \neg P_\neg \text{ and } \forall \, \neg ta_\neg \text{ and } \forall \, \neg t_\neg \text{ are } \alpha\text{-convertible} \quad \text{and } \forall \, \neg x_\neg \text{ is a varstruct} \\
27054 \quad ?0 \text{ not of the form: } "\Gamma \vdash \forall \, \text{vs}[x,y,...] \bullet P[x,y,...] \Rightarrow \text{vs}[x,y,...] = t\"
8.1. General Inference Rules

**val β_conv : CONV;**

**Description**  Apply a β-reduction to an abstraction.

<table>
<thead>
<tr>
<th>Rule</th>
<th>$\Gamma \vdash (\lambda x \cdot t[x])y = t'[y]$</th>
<th>$\beta_{\text{conv}} \Gamma (\lambda x \cdot t1[x])t2\gamma$</th>
</tr>
</thead>
</table>

where $x$ may be any varstruct allowed by the ICL HOL syntax, $y$ is an instance of this structure, and $t'$ is α-convertible to $t$, changed to avoid variable capture.

When the bound variable structure has a pair, where the value applied to does not, then $\text{Fst}$ and $\text{Snd}$ are introduced as necessary, e.g.:

**Example**

$\beta_{\text{conv}} \Gamma (\lambda (x, y) \cdot f x y) \ p \gamma =\Gamma (\lambda (x, y) \cdot f x y) \ p = f (\text{Fst} \ p) (\text{Snd} \ p)$

**See Also**  simple_β_conv, β_rule

**Errors**

27008  ?0 is not of the form: $\Gamma (\lambda x \cdot t1[x])t2\gamma$

where $\Gamma x\gamma$ is a varstruct

**val β_rule : THM --> THM;**

**Description**  An elimination rule for λ, which can handle paired abstractions.

<table>
<thead>
<tr>
<th>Rule</th>
<th>$\Gamma \vdash (\lambda x \cdot t[x]) y$</th>
<th>$\beta_{\text{rule}} \Gamma \vdash t[y]$</th>
</tr>
</thead>
</table>

**See Also**  β_conv

**Errors**

27007  ?0 is not of the form: $\Gamma (\lambda x \cdot t[x]) y\gamma$

where $\Gamma x\gamma$ is a varstruct
### Chapter 8. PROOF IN HOL

#### SML

**val epsilon_elim_rule :** \( \text{TERM} \rightarrow \text{THM} \rightarrow \text{THM} \rightarrow \text{THM}; \)

**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 structure satisfies the predicate.

**Rule**

\[
\begin{align*}
\Gamma_1 \vdash t' \ (\epsilon t'') \\
\Gamma_2, t \ vs \vdash s \\
\Gamma_1 \cup \Gamma_2 \vdash s
\end{align*}
\]  

\( \epsilon \)-elim rule

where \( t, t' \) and \( t'' \) are \( \alpha \)-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**

<table>
<thead>
<tr>
<th>Code</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 is not an allowed variable structure</td>
</tr>
<tr>
<td>2</td>
<td>0 is not of the form: ( \Gamma \vdash t_1 (t_1 t_2) )</td>
</tr>
<tr>
<td>3</td>
<td>0 is not of the same type as choice sub-term of first theorem</td>
</tr>
<tr>
<td>4</td>
<td>0 is repeated in the varstruct ( t )</td>
</tr>
<tr>
<td>5</td>
<td>Arguments 0; 1 and 2 not of the form ( \Gamma_1 \vdash t (\epsilon t) ) and ( \Gamma_2, (t \ vs) \vdash s )</td>
</tr>
<tr>
<td>6</td>
<td>0 has members appearing free in conclusion of ( t )</td>
</tr>
<tr>
<td>7</td>
<td>0 has members appearing free in ( t ) other than in assumption ( t )</td>
</tr>
</tbody>
</table>

#### SML

**val epsilon_intro_rule :** \( \text{THM} \rightarrow \text{THM}; \)

**Description**  Given a theorem whose conclusion is a function application, we know that the “function” is a predicate, and the rule states that \( \epsilon \) of this predicate will satisfy the predicate.

**Rule**

\[
\begin{align*}
\Gamma \vdash t_1 \ t_2 \\
\Gamma \vdash t_1 \ (\epsilon t_1)
\end{align*}
\]  

\( \epsilon \)-intro rule

**Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 is not of the form: ( \Gamma \vdash t_1 t_2 )</td>
</tr>
</tbody>
</table>

#### SML

**val eta_conv :** \( \text{CONV}; \)

**Description**  The rule for \( \eta \) conversion.

\[
\begin{align*}
\vdash (\lambda vs \bullet t \ vs) = t \\
\Gamma \vdash \lambda vs \bullet t \ vs
\end{align*}
\]  

\( \eta \)-conv

where \( t \) contains no free instances of the variables of varstruct \( vs \).

**Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 is not of the form: ( \Gamma \vdash \lambda vs \bullet t \ vs )</td>
</tr>
<tr>
<td>2</td>
<td>where ( vs ) is a varstruct</td>
</tr>
<tr>
<td>3</td>
<td>0 is not of the form: ( \Gamma \vdash \lambda vs \bullet t \ vs )</td>
</tr>
<tr>
<td>4</td>
<td>where ( t ) should not contain ( vs )</td>
</tr>
</tbody>
</table>
### 8.1. General Inference Rules

#### SML
```sml
(val λ_C : CONV -> CONV;
)
```

**Description**  
Apply a conversion to the body of an abstraction:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \vdash (\lambda x \cdot p[x]) = (\lambda x \cdot pa[x]) )</td>
<td>( \lambda_C ) ( (c : CONV) ) ( \vdash \lambda x \cdot p )</td>
</tr>
</tbody>
</table>

where \( c \ p[x] \) gives \( \vdash p[x] = pa[x] \).

---

#### Errors

- **4002**  
  '?0 is not of form: \( \lambda vs \cdot t \)'
- **7104**  
  'Result of conversion, ?0, ill-formed'

Also as the failure of the conversion.

---

#### SML
```sml
(val λ_eq_rule : TERM -> THM -> 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.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \vdash t_1[x] = t_2[x] )</td>
<td>( \lambda_eq_rule ) ( \vdash \lambda x \cdot t )</td>
</tr>
</tbody>
</table>

\( \Gamma \vdash (\lambda x \cdot t_1[x]) = (\lambda x \cdot t_2[x]) \)

---

#### Errors

- **4016**  
  '?0 is not an allowed variable structure'
- **6005**  
  '?0 occurs free in assumption list'
- **6020**  
  '?0 is not of the form: \( \Gamma \vdash t_1 = t_2 \)'

---

#### SML
```sml
(val λ_pair_conv : 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.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \vdash (\lambda v \cdot t) = (\lambda (v_1, v_2) \cdot t'[v_1, v_2]) )</td>
<td>( \lambda_pair_conv ) ( \Gamma \vdash \lambda v : a \times b \cdot t )</td>
</tr>
</tbody>
</table>

\( \lambda_pair_conv \) \( \Gamma \vdash \lambda (v, w) : (a \times b) \times (c \times d) \cdot t \)

and so on.

---

#### Errors

- **27055**  
  'The type of '?0 is not of the form \( \sigma \times \tau \)'

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUAL USR029
### λ-rule

**Description** An introduction rule for $\lambda$:

$$\frac{\Gamma \vdash s[t]}{\Gamma \vdash (\lambda x \cdot s[x]) \ t}$$

where $x$ is a machine generated variable.

### λ-varstruct_conv

**Description** This conversion is a generalisation of $\alpha_{\text{conv}}$ 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.

$$\frac{\Gamma \vdash (\lambda \ vs2[x2,y2,...] \cdot t[x2,y2,...])}{\Gamma \vdash (\lambda \ vs1[x1,y1,...] \cdot t[x1,y1,...])}$$

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,

$$\frac{\Gamma \vdash ((\lambda x \cdot \text{Fst} \ x + \text{Snd} \ x = 1)}{\Gamma \vdash (\lambda (a, b) \cdot a + b = 1)}$$

**See Also** $\alpha_{\text{conv}}$ for a more limited form of renaming.

**Errors**

27050 Cannot prove equality of $?0$ and $?1
8.2 Subgoal Package

**SML**


gSignature 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**

(* pp'TS *)

description The theory will contain a constant named pp'TS, defined by a definition with key “pp'TS”.

definition |- ∀ x • (pp'TS x) ↔ 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**

(* 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.
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.

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.

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.

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*
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, 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
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
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
30026  There is no current goal
30027  There is no assumption ?0 in the current goal
SML

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 $\alpha$-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

SML

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.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUAL USR029
8.2. 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 | ∗) | α ⇒ ¬ b  |
| (∗ | 2 | ∗) | α ⇒         |
| |     | a ⇒         |
| |     | a ⇒ b  |
| (∗ | 1 | ∗) | ¬ b ⇒ a  |
| (∗ | ?| |- | ∗) | α ∨ b  |

where ¬ b ⇒ a  is the first assumption, and the second assumption is too long to fit on one line. Then with no assumptions:

Example

| (∗ | ?| |- | ∗) | α ∨ 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 ""; 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 "". 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
8.2. Subgoal Package

**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.

**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

**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"
val simplify_goal_state_thm : THM -> 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

val subgoal_package_size : 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

val top_current_label : 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

val top_goals : 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

val top_goal_state_thm : 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
8.2. Subgoal Package

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

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

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"

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

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
**SML**

```sml
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
8.3 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
\[
\text{type GOAL} \quad (\ast = \text{SEQ} \ast);
\]
\[
\text{type PROOF} \quad (\ast = \text{THM list} -> \text{THM} \ast);
\]
\[
\text{type TACTIC} \quad (\ast = \text{GOAL} -> (\text{GOAL list} * \text{PROOF}) \ast);
\]

**Description** \text{TACTIC} is the type of tactics. The types \text{GOAL} and \text{PROOF} help to abbreviate its definition.

SML
\[
\text{type THM_TAC} \quad (\ast = \text{THM} \ast);
\]
\[
\text{type THM_TAC} \quad (\ast = \text{THM_TAC} \ast);
\]

**Description** These are the types of theorem tactics and theorem tacticals.

SML
\[
\text{val accept_tac} : \text{THM} -> \text{TACTIC};
\]

**Description** Prove a goal by a theorem which is \(\alpha\)-convertible to it.

\[
\begin{align*}
\text{Tactic} & \quad \{ \Gamma2 \} \ t2 \\
& \quad \text{accept_tac} \\
& \quad \Gamma1 \vdash \ t1
\end{align*}
\]

where \(t1\) and \(t2\) are \(\alpha\)-convertible.

**Errors**
\[
9102 \quad ?0 \text{ is not } \alpha\text{-convertible to the goals conclusion } ?1
\]

SML
\[
\text{val all_asm_ante_tac} : \text{TACTIC};
\]

**Description** Apply \text{asm_ante_tac} to every assumption in turn:

\[
\begin{align*}
\text{Tactic} & \quad \{ \ t1, ..., tn \ \} \ t \\
& \quad \{ \} \ tn \Rightarrow ... \Rightarrow t1 \Rightarrow t
\end{align*}
\]

\(\alpha\)-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**
\[
28055 \quad \text{The conclusion or an assumption of goal does not have type } \langle BOO\rangle
\]
**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: "\(\text{var} = \text{value}\)" or "\(\text{value} = \text{var}\)", 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.

\(\text{all\_var\_elim\_asm\_tac}\) 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 \(\text{var\_elim\_asm\_tac}\)). 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 \(\text{check\_asm\_tac}\), otherwise it will be stripped back into the assumption list by \(\text{strip\_asm\_tac}\). 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 \(\text{REPEAT all\_var\_elim\_asm\_tac}\) not halting. \(\text{ALL\_VAR\_ELIM\_ASM\_T}\) allows the users choice of function to be applied to the modified assumptions, rather than \(\text{strip\_asm\_tac}\).

\(\text{all\_var\_elim\_asm\_tac1}\) works as \(\text{all\_var\_elim\_asm\_tac}\), 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 \(\text{var\_elim\_asm\_tac}\)). \(\text{ALL\_VAR\_ELIM\_ASM\_T1}\) 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 basic_prove_tac.

**See Also** prop_eq_prove_tac for more sophisticated approach to these kinds of problems.

**Errors**

29028 This tactic is unable to eliminate any variable
8.3. General Tactics and Tacticals

SML

val all_β_tac : TACTIC;

Description This tactic will β-reduce all β-redexes in the goal’s conclusion, including those redexes introduced by preceding β-reductions in the same tactic application.

Uses In most proof contexts β-reduction will be a side effect of rewriting; this tactic is intended for cases where rewriting would do “too much”.

See Also all_β_rule, all_β_conv

Errors 27049 ?0 contains no β-redexes

SML

val all_ε_tac : TACTIC;
val ALL_ε_T : (THM --> TACTIC) --> TACTIC;

Description all_ε_tac applies ε_tac to all subterms of the conclusion of the goal of the form εx•t. ALL_ε_T is similar but uses ε_T rather than ε_tac. The effect is to set the corresponding terms of the form ∃x•t as lemmas, and to derive new assumptions of the form t[εx•t/x].

Tactic

{ Γ } t[εx1•t1/y1, ..., εxk•tk/yk] →
{ Γ } ∃x1•t1; ...; { Γ } ∃xk•tk;
strip t1[εx1•t1/x1], ..., strip tk[εxk•tk/xk], Γ } t

ε_tac

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., (∀x•T) = (∀x•T).

SML

val anteu_tac : THM --> TACTIC;

Description Replace a goal with conclusion t2 by t1⇒t2, where the antecedent, t1, of the implication is the conclusion of a theorem:

Tactic

{ Γ2 } t2 →
{ Γ2 } t1 ⇒ t2

ante_tac (Γ1 ⊢ t1)

where the assumptions, Γ1, of the theorem are contained in the assumptions, Γ2 of the goal.

Uses This is often useful if one needs to transform the conclusion of theorem e.g. by rewriting with the assumptions.

See Also asm_tac, strip_asm_tac

Errors 28027 Conclusion of goal does not have type "BOOL"
**asm ante tac**

**Description** Bring a term out of the assumption list into the goal as the antecedent of an implication.

\[
\begin{align*}
\{ \Gamma, t1' \} & \vdash t1 \Rightarrow t2 \\
\{ \Gamma \} & \vdash t1 \Rightarrow t2
\end{align*}
\]

where \( t1 \) and \( t1' \) are \( \alpha \)-convertible. Note that all assumptions \( \alpha \)-convertible with \( t1 \) 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 \( \forall:BOOL \).
8.3. General Tactics and Tacticals

SML

```sml
val back_chain_tac : THM list -> TACTIC;
val bc_tac : THM list -> TACTIC;
```

**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, `t1 ⇒ t2`, or a bi-implication `t1 ⇔ t2` for which the conclusion of the goal is a substitution instance, `t2'` say, of `t2`. If no such theorem is found then stop.
2. If in step 1, an applicable theorem, say `thm`, has been found reduce the goal to the corresponding instance of `t1` (or an existentially quantified version thereof) using `bc_thm_tac`, q.v.
3. Repeatedly apply `∀tac` or `∧tac` until neither of these is applicable.
4. Delete `thm` from the list of theorems and return to step 1.

In step 4, only the first appearance of `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:

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

```
⇒ tac
|Γ " Γ1 ⊢ t1 ∧ (∀x•t2) ⇒ t3,  
Γ2 " Γ4 ⇔ t1, 
Γ3 " Γ5 ⇒ t2, 

{ Γ } t3' ; { Γ } t4' ; { Γ } t5'
```

(Here `t3'` is some substitution instance of `t3` and `t4'` and `t5'` are the corresponding instances of `t4` and `t5`.)

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_thm_tac` (which is used to perform step 2).

**Errors**

```
29012  Theorem ?0 is not of the form 'Γ ⊢ ∀ x1 ... xn • u ⇔ v'
      or 'Γ ⊢ ∀ x1 ... xn • u ⇒ v'
```

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
**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:

\[
\frac{\{ \Gamma \} \ t2'}{\{ \Gamma \} \ t1'} \quad \text{bc_thm_tac} \quad \Gamma I \vdash t1 \Rightarrow t2
\]

\[
\frac{\{ \Gamma \} \ t2'}{\{ \Gamma \} \ t1'} \quad \text{bc_thm_tac} \quad \Gamma I \vdash t1 \Leftrightarrow t2
\]

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,

\[
\frac{\{ \Gamma \} \ a < b}{\{ \Gamma \} \ \exists i \cdot a < i \land i < b} \quad \text{bc_thm_tac} \quad \vdash \forall m \ i \ n \cdot m < i \land i < n \Rightarrow m < n
\]

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).

---

**val** bad_proof : string -> 'a

**Description**  
bad_proof name is equivalent to error name 9001 []). bad_proof is for use in low level tactical programming to report the error situation when the proof generated by a tactic is supplied with the wrong number of arguments. (This will not happen for the usual use of tactics with tac_proof or within the subgoal package):

**Errors**

| 29011 | Conclusion of the goal is not an instance of: ?0 |
| 29012 | Theorem ?0 is not of the form 'Γ ⊢ ∀ x1 ... xn • u ⇔ v' or 'Γ ⊢ ∀ x1 ... xn • u ⇒ v' |

**Uses** Specialised low-level tactic programming.
### 8.3. General Tactics and Tacticals

**SML**

```sml
val CASES_T2 : TERM -> (THM -> TACTIC) ->
               (THM -> TACTIC) -> TACTIC;
```

**Description**

Do a case split on a given boolean term using two tactic generating functions:

\[
\text{CASES}_2(t_1 \ t_\text{tac}_1 \ t_\text{tac}_2 (\{\Gamma\} \ t_2) = t_\text{tac}_1(t_1 \vdash t_1)(\{\Gamma\} \ t_2) ; t_\text{tac}_2(\neg t_1 \vdash \neg t_1)(\{\Gamma\} \ t_2)
\]

**See Also**

`cases_tac`, `\_THEN`, `CASES_T`

**Errors**

28022 ?0 is not boolean

---

**SML**

```sml
val cases_tac : TERM -> TACTIC;
```

**Description**

Do a case split on a given boolean term.

\[
\text{cases}_\text{tac} \ (\{\Gamma\} \ t_2)
\]

**Tactic**

\[
\frac{(\text{strip } t_1, \ \{\Gamma\}) \ t_2}{(\text{strip } \neg t_1, \ \{\Gamma\}) \ t_2}
\]

**See Also**

`CASES_T`, `\_THEN`

**Errors**

28022 ?0 is not boolean

---

**SML**

```sml
val CASES_T : TERM -> (THM -> TACTIC) -> TACTIC;
```

**Description**

Do a case split on a given boolean term using a tactic generating function:

\[
\text{CASES}_T(t_1 \ t_\text{tac} (\{\Gamma\} \ t_2) = t_\text{tac}(t_1 \vdash t_1)(\{\Gamma\} \ t_2) ; t_\text{tac}(\neg t_1 \vdash \neg t_1)(\{\Gamma\} \ t_2)
\]

**See Also**

`cases_tac`, `\_THEN`, `CASES_T2`

**Errors**

28022 ?0 is not boolean

---

**SML**

```sml
val CHANGED_T : TACTIC -> TACTIC;
```

**Description**

`CHANGED_T tac` is a tactic which applies `tac` to the goal and fails if this results in a single subgoal which is \(\alpha\)-convertible to the original goal.

**Uses**

`CHANGED_T` can be a useful way of ensuring termination of, e.g., rewriting tactics.

**Errors**

9601 the tactic did not change the goal
**check_asm_tac**

**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>( \Gamma \vdash t )</td>
<td>proves goal if its conclusion is ( t )</td>
</tr>
<tr>
<td>( \Gamma \vdash T )</td>
<td>as <code>id_tac</code> (i.e. the theorem is discarded)</td>
</tr>
<tr>
<td>( \Gamma \vdash F )</td>
<td>proves goal</td>
</tr>
<tr>
<td>( \Gamma \vdash \neg t )</td>
<td>proves goal if ( t ) in assumptions, else as <code>asm_tac</code></td>
</tr>
<tr>
<td>( \Gamma \vdash t )</td>
<td>proves goal if ( \neg t ) in assumptions, else as <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`.

---

**concl_in_asms_tac**

**Description**

`concl_in_asms_tac` is a tactic which checks whether the conclusion of the goal is also in the assumptions, and if so proves the goal.

\[
\begin{array}{c}
\{ \Gamma, \ t \} \ t' \\
\hline
\end{array}
\]

`concl_in_asms_tac`

where \( t \) and \( t' \) are \( \alpha \)-convertible.

**Uses**

Tactic programming.

**See Also**

`strip_tac`.

**Errors**

28002  Goal does not appear in the assumptions

---

**COND_T**

**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 \( \neg X \) 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.
8.3. General Tactics and Tacticals

**SML**

```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 \( \_ \_ \text{contr_tac} \).

**Tactic**

\[
\begin{array}{c}
\{ \Gamma \} t \\
\{ \text{strip } \neg t, \Gamma \} F \\
\end{array} \quad \text{contr_tac}
\]

**Uses**  Proof by contradiction.

**See Also**  \( \_ \_ \text{strip_tac}, \neg \_ \_ \text{tac} \).

**Errors**

```sml
28027 Conclusion of goal does not have type \( \subseteq \text{BOOL} \)
```

**SML**

```sml
val CONTR_T : (THM -> TACTIC) -> TACTIC;
```

**Description**  A form of proof by contradiction as a tactical. \( \_ \_ \text{CONTR_T thmtac} \) is a tactic which attempts to solve a goal \( (\Gamma, t) \), by applying \( \_ \_ \text{thmtac}(\neg t \vdash \neg t) \) to the goal \( (\Gamma, F) \).

**Tactic**

\[
\begin{array}{c}
\{ \Gamma \} t \\
\text{thmtac } (\neg t \vdash \neg t) (\{ \Gamma \} F) \\
\end{array} \quad \_ \_ \text{CONTR_T thmtac}
\]

**Uses**  Proof by contradiction in combination with a theorem tactic.

**See Also**  \( \_ \_ \text{contr_tac}, \neg \_ \_ \text{T} \).

**Errors**

```sml
28027 Conclusion of goal does not have type \( \subseteq \text{BOOL} \)
```

**SML**

```sml
val conv_tac : CONV -> TACTIC;
```

**Description**  \( \_ \_ \text{conv_tac conv} \) is a tactic which applies the conversion \( \_ \_ \text{conv} \) 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:

**Tactic**

\[
\begin{array}{c}
\{ \Gamma 2 \} t2 \\
\{ \Gamma 2 \} t1 \\
\end{array} \quad \_ \_ \text{asm_tac conv}
\]

where \( \_ \_ \text{conv t2 } = (\Gamma 1 \vdash t2 = t1) \).

**Errors**

```sml
9400 the conversion returned ‘?0’ which is not of the form:
‘... \vdash ?1 \leftrightarrow ...’
```

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUAL USR029
\textbf{SML}\[\text{val } \text{CONV\_THEN} : \text{CONV} \rightarrow \text{THM\_TACTICAL};\]

\textbf{Description} \hspace{1em} \text{CONV\_THEN} \text{ conv thmtac} is a theorem tactic which first uses \text{conv} to transform the conclusion of a theorem and then acts as \text{thmtac}.

\[(\text{CONV\_THEN}) \text{ conv thmtac thm} = \text{thmtac} (\text{conv thm})\]

\textbf{Uses} \hspace{1em} For use in programming theorem tacticals. The function may be partially evaluated with only its conversion, theorem tactic and theorem arguments.

\textbf{Errors} \hspace{1em} 9400 \hspace{0.5em} the conversion returned ‘?0‘ which is not of the form:

\[‘... \vdash ?1 \leftrightarrow ...‘\]

\textbf{SML}\[\text{val discard_tac} : ‘a \rightarrow \text{TACTIC};\]
\[\text{val k_id_tac} : ‘a \rightarrow \text{TACTIC};\]

\textbf{Description} \hspace{1em} A tactic that discards its argument, but otherwise has no effect. \text{k_id_tac} is an alias for \text{discard_tac}.

\textbf{Uses} \hspace{1em} Can be used to remove unwanted assumptions: \text{a \ (POP\_ASM\_T discard_tac)} discards the top-most assumption. This usage of \text{discard_tac} may strengthen the goal. ie it may result in unprovable subgoals even when the original goal was provable.

\textbf{SML}\[\text{val DROP\_ASMS\_T} : (\text{THM list} \rightarrow \text{TACTIC}) \rightarrow \text{TACTIC};\]

\textbf{Description} \hspace{1em} \text{DROP\_ASMS\_T} \text{ thmstac} is a tactic which applies \text{asm\_rule} to each assumption of the subgoal, giving a list of theorems, \text{thms} say, then removes all the assumptions of the goal and then acts as \text{thmstac thms}.

\[\begin{array}{c}
\text{Tactic} \\
\begin{array}{c}
\{ \Gamma \} t \\
\text{thmstac (map asm\_rule } \Gamma \text{) (}\{\} t\text{)}
\end{array}
\end{array}\]

\text{DROP\_ASMS\_T thmstac}

\textbf{Uses} \hspace{1em} To use all the assumptions as theorems.

\textbf{Errors} \hspace{1em} As for \text{thmstac}.

\textbf{SML}\[\text{val DROP\_ASM\_T} : \text{TERM} \rightarrow (\text{THM} \rightarrow \text{TACTIC}) \rightarrow \text{TACTIC};\]

\textbf{Description} \hspace{1em} \text{DROP\_ASM\_T} \text{ asm thmtac} is a tactic which removes \text{asm} from the assumption list and then acts as \text{thmtac(asm\_\_asm)}.

\[\begin{array}{c}
\text{Tactic} \\
\begin{array}{c}
\{ \Gamma, \text{asm'} \} t \\
\text{thmstac (asm \vdash asm) (}\{\Gamma\} t\text{)}
\end{array}
\end{array}\]

\text{DROP\_ASM\_T thmstac} \hspace{1em} \text{r asm'}

where \text{asm} and \text{asm'} are \alpha-convertible.

\textbf{Uses} \hspace{1em} To use an assumption as a theorem

\textbf{Errors} \hspace{1em} 9301 \hspace{0.5em} the term ‘?0‘ is not in the assumption list
### 8.3. General Tactics and Tacticals

| SML | \[\text{val DROP\_FILTER\_ASMS\_T : (TERM -> bool) -> (THM list -> TACTIC) -> TACTIC;}\] |
| Description | \(\text{DROP\_FILTER\_ASMS\_T \ pred \ thmstac}\) is a tactic which applies \text{asm\_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 | \[
\begin{align*}
\{ \Gamma \} \ t \\
\text{thmstac} \ (\text{map asm\_rule} \ (\Gamma \cap \text{pred})) \\
\left( \{ \Gamma \setminus \text{pred} \} \ t \right) \\
\text{DROP\_FILTER\_ASMS\_T} \\
\text{pred} \\
\text{thmstac} \\
\end{align*}
\] |
| Uses | To use all the selected assumptions as theorems. |
| Errors | As for \text{thmstac}. |

| SML | \[\text{val DROP\_NTH\_ASM\_T : int -> (THM -> TACTIC) -> TACTIC;}\] |
| Description | \(\text{DROP\_NTH\_ASM\_T \ i \ thmtac}\) is a tactic which applies \text{asm\_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{thmtac thm}. |
| Assumptions are numbered 1, 2..., so that, e.g., \(\text{DROP\_NTH\_ASM\_T 1}\) is the same as \(\text{POP\_ASM\_T}\) |
| Tactic | \[
\begin{align*}
\{ a_1, \ldots, a_n \} \ t \\
\text{thmtac} \ (\text{asm\_rule} \ [a_i]) \ (\{ \Gamma \setminus a_i \} \ t) \\
\text{DROP\_NTH\_ASM\_T} \\
i \\
\text{thmtac} \\
\end{align*}
\] |
| Uses | To use an assumption as a theorem, treating the assumption list as an array. |
| Errors | \(9303 \quad \text{the index ?0 is out of range}\)|

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
val eq_sym_asm_tac : TERM -> TACTIC;
val eq_sym_nth_asm_tac : int -> TACTIC;

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  \[
\frac{\{ \Gamma_1, t[x = y, p = q, ...], \Gamma_2 \} \text{ cnc}}{\{ \text{strip } t[y = x, q = p, ...], \Gamma_1, \Gamma_2 \} \text{ cnc}} \quad \text{eq_sym_asm_tac}
\]
\[
\frac{\{ t_1, ..., t_{n-1}, t_n[x = y, p = q, ...], t_{n+1}, ..., \} \text{ cnc}}{\{ \text{strip } t_n[y = x, q = p, ...], \quad t_1, ..., t_{n-1}, t_{n+1}, ..., \} \text{ cnc}} \quad \text{eq_sym_nth_asm_tac}
\]

Definition

fun eq_sym_asm_tac asm = DROP_ASM_T asm (strip_asm_tac o conv_rule(ONCE_MAP_C eq_sym_conv));

fun eq_sym_nth_asm_tac n = DROP_NTH_ASM_T n (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>( \forall x y \cdot x \leftrightarrow y ) \( \forall y x \cdot y \leftrightarrow x )</td>
<td></td>
</tr>
<tr>
<td>( f(x = (p = q)) ) ( f((p = q) = x) )</td>
<td></td>
</tr>
<tr>
<td>( x = y \land p = q ) ( x = y, p = q )</td>
<td></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

val EVERY_TTCL : THM_TACTICAL list -> THM_TACTICAL;

Description  EVERY_TTCL is a theorem tactical combinator.

\[
\text{EVERY_TTCL [ttcl1, ttcl2, ...]} = \text{ttcl1 THEN_TTCL ttcl2 THEN_TTCL} ...
\]

EVERY_TTCL [] acts as ID_THEN.

Uses  For use in programming theorem tacticals.
8.3. General Tactics and Tacticals

**EVERY.T** : TACTIC list -> TACTIC;

**EVERY** : TACTIC list -> TACTIC;

**Description**  
EVERY.T tlist is a tactic that applies the head of tlist to its subgoal, and recursively applies the tail of tlist to each resulting subgoal. EVERY is an alias for EVERY.T. EVERY [] is equal to id_tac.

**Example**

EVERY [∀.tac, ∧.tac, ∀.tac]

is equivalent to

∀.tac THEN ∧.tac THEN ∀.tac

**Errors**  
As for the tactics in the list.

**FAIL.T** : TACTIC;

**Description**  
This is the identity for the tactical ORELSE.T

**Uses**  
For constructing larger tactics.

**Errors**

9201 failed as requested

**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 combinator ORELSE.TTCL.

**Uses**  
For use in programming theorem tacticals.

**Errors**

9401 failed as requested

**FAIL.with** : string -> int -> (unit -> string) list -> TACTIC;

**Description**  
fail.with 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** : 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.
**SML**

```sml
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

---

**SML**

```sml
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.

---

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALSER029
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 $\langle F \rangle$ 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 $\forall x \cdot x \in a \Rightarrow x \in b$. 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 \, x, \forall x \cdot p \, x \Rightarrow q \, x, \forall x \cdot q \, x \Rightarrow r \, x \} r \, 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 - HOL REFERENCE MANUALUSR029
val FORWARD_CHAIN_T : 
  (THM list -> TACTIC) -> THM list -> TACTIC;
val FC_T : 
  (THM list -> TACTIC) -> THM list -> TACTIC;
val ALL_FORWARD_CHAIN_T : 
  (THM list -> TACTIC) -> THM list -> TACTIC;
val ALL_FC_T : 
  (THM list -> TACTIC) -> THM list -> TACTIC;
val ASM_FORWARD_CHAIN_T : 
  (THM list -> TACTIC) -> THM list -> TACTIC;
val ASM_FC_T : 
  (THM list -> TACTIC) -> THM list -> TACTIC;
val ALL_ASM_FORWARD_CHAIN_T : 
  (THM list -> TACTIC) -> THM list -> TACTIC;
val ALL_ASM_FC_T : 
  (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 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  fc_tac is the same as: FC_T (MAP_EVERY strip_asm_tac).

To rewrite the goal with the results of the forward inference one could use FC_T rewrite_tac.

See Also  fc_tac, asm_fc_tac, bc_tac, FC_T1.
8.3. General Tactics and Tacticals

SML

```sml
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;
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;
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 \(\vdash t1 \Rightarrow (t2 \iff t3)\) for rewriting, one might use \(FC\_T1\) \(id\_canon\) \(rewrite\_tac\).

**See Also** \(fc\_tac\), \(asm\_fc\_tac\), \(bc\_tac\), \(FC\_T\).

SML

```sml
val f_thm_tac : THM -> TACTIC;
```

**Description**

Prove a goal by using a theorem of the form \(\Gamma \vdash F\).

```
Tactic

\[
\left\{ \begin{array}{c} \Gamma 2 \\ \end{array} \right\} t \quad f\_thm\_tac (\Gamma 1 \vdash F)
\]

where the assumptions, \(\Gamma 1\), of the theorem are contained in the assumptions, \(\Gamma 2\), of the goal.
```

**Errors**

\[28021\ ?0 \ does \ not \ have \ the \ form \ \Gamma \vdash F\]

**Uses**

In tactic programming, to use a theorem which shows that the assumptions are contradictory.

**See Also** \(strip\_asm\_tac\).
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.

\texttt{GEN\_INDUCTION\_T} causes any inductive hypotheses (see below) to be passed to a tactic generating function.

\texttt{gen\_induction\_tac thm} is the same as \texttt{GEN\_INDUCTION\_T thm strip\_asm\_tac}.

The discussion below is for the tactic computed by the call \texttt{GEN\_INDUCTION\_T thm ttac y} applied to a goal with conclusion \( t \).

The induction principle, \( thm \) has the form:

\[ \vdash \forall \ p \cdot a \Rightarrow \forall x \cdot p \ x \]

E.g. the usual principle of induction for the natural numbers:

\[ \vdash \forall \ p \cdot p \ 0 \land (\forall n \cdot p \ n \Rightarrow p \ (n + 1)) \Rightarrow (\forall n \cdot p \ n) \]

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 \( \forall \) 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.

| 29021 | ?0 does not have the form \( \vdash \forall p \cdot a \Rightarrow \forall x \cdot p \ x \) |
| 29023 | The type of ?0 is not an instance of ?1 |
| 29024 | ?0 is not a variable |
| 29025 | ?0 appears free in the assumptions of the goal |
| 29026 | ?0 does not appear free in the conclusion of the goal |
8.3. 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 \( \vdash a' \Rightarrow c \) 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 \( \vdash \forall p : \tau \rightarrow \text{ BOOL} \cdot a \Rightarrow \forall x \cdot t[p/x/b] \)
  (where \( \tau x \) must also appear in \( \tau t \) other than as an argument of \( \tau p \))
- 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`.

**Tactic**

\[
\begin{array}{c}
\{ \Gamma \} t \\
\hline
\text{thmstac} \ (\text{map} \ \text{asm_rule} \ \Gamma) \\
\{ a1, \ldots, an \} t
\end{array}
\]

\[ \text{GET}_{-} \text{ASMS}_{-} \text{T} \]

\[ \text{thmstac} \]

**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`.

**Tactic**

\[
\begin{array}{c}
\{ \Gamma, \text{asm'} \} t \\
\hline
\text{thmtac} \ (\text{asm} \vdash \text{asm'}) \ (\{ \Gamma', \text{asm'} \} t)
\end{array}
\]

\[ \text{GET}_{-} \text{ASM}_{-} \text{T} \]

\[ \text{thmtac} \]

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 - HOL REFERENCE MANUALUSR029
**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**

\[
\frac{\{ \Gamma \} \; t}{thmstac \left( \text{map} \; \text{asm_rule} \left( \{ \Gamma \} \right) \cap \text{pred} \right)}
\]

**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**

\[
\frac{\{ a1, ..., an \} \; t}{\text{thmtac} \left( \text{asm_rule} \left[ a1 \right] \right) \left( \{ \Gamma \} \; t \right)}
\]

**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.*

**Tactic**

\[
\frac{\{ \Gamma \} \; t}{id_tac}
\]

**Uses**  
For constructing larger tactics.

---

**SML**

```sml
val ID_THEN : THM_TACTICAL;
```

**Description**  
This is the identity for the theorem tactical combinator *THEN_TTCL.*

\[
(Id\_THEN) \; \text{thmtac} = \text{thmtac}
\]

**Uses**  
For use in programming theorem tacticals.
8.3. General Tactics and Tacticals

SML

```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.

<table>
<thead>
<tr>
<th>Tactic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF_T2</td>
<td>Reduce a conditional by applying tactic generating functions to the two cases for the selector.</td>
</tr>
</tbody>
</table>

```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.

```sml
val IF_THEN : (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.

See Also  `⇔_T, STRIP_CONCL_T`

Errors  [28071]  Goal is not of the form: \{ Γ \} if a then tt else et

See Also  `strip_tac`

Errors  [28071]  Goal is not of the form: \{ Γ \} if a then tt else et
**val** IF.T : (THM \(\rightarrow\) TACTIC) \(\rightarrow\) TACTIC;

**Description** Reduce a conditional by applying a tactic generating function to the two cases for the selector.

\[
\begin{align*}
\text{Tactic} & \quad \{ \Gamma \} \text{ if } a \text{ then } \text{tt} \text{ else } \text{et} \\
& \quad \text{ttac}\{ a, \Gamma \} \vdash \text{tt} \\
& \quad \text{ttac}\{ \neg a, \Gamma \} \vdash \text{et}
\end{align*}
\]

**IF.T**

**See Also** IF.T2, STRIP_CONCL_T

**Errors**

28071 Goal is not of the form: \(\{ \Gamma \} \text{ if } a \text{ then } \text{tt} \text{ else } \text{et}\)

---

**val** intro\_\_tac : (TERM * TERM) \(\rightarrow\) TACTIC;

**val** intro\_\_tac1 : TERM \(\rightarrow\) TACTIC;

**Description** Sometimes it is helpful to prove a goal by proving a more general conjecture has the goal as a special case. *intro\_\_tac* introduces a universal quantifier into the conclusion of a goal in order to do this.

\[
\begin{align*}
\text{Tactic} & \quad \{ \Gamma \} \ t[t1/x] \\
& \quad \{ \Gamma' \} \ \forall x \bullet t
\end{align*}
\]

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.,

\[
\begin{align*}
\{ \Gamma \} \ 1 + 2 > 0 \Rightarrow \neg 1 + 2 = 0 \\
& \quad \{ \Gamma' \} \ \forall i \bullet i > 0 \Rightarrow \neg i = 0
\end{align*}
\]

intro\_\_tac \((\forall 1+2\gamma, \neg i; \mathbb{N}\gamma)\)

**Example**

**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. *intro\_\_tac1* \(\forall x\gamma\) is equivalent to *intro\_\_tac* \(\forall x\gamma, \forall x\gamma\).

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 \(?1\)
### i_contr_tac

**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**

<table>
<thead>
<tr>
<th>{ Γ } t</th>
<th>{ Γ } F</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i_{\text{contr} _ \text{tac}} )</td>
<td></td>
</tr>
</tbody>
</table>

**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.

### lemma_tac

**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**

<table>
<thead>
<tr>
<th>{ Γ } t2</th>
<th>{ Γ } t1; strip t1, Γ { Γ } t2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{lemma} _ \text{tac} )</td>
<td></td>
</tr>
</tbody>
</table>

**See Also**  
LEMMA_T

**Errors**

9603 the term ?0 is not boolean

### LEMMA_T

**Description**  
LEMMA_T newsg thmtac is a tactic which sets newsg as a new subgoal and applies thmtac(newsg \( \vdash \) newsg) to the original goal.

**Tactic**

<table>
<thead>
<tr>
<th>{ Γ } t</th>
<th>{ Γ } newsg; thmtac(newsg ( \vdash ) newsg) ({ Γ } t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{LEMMA} _ \text{T} )</td>
<td></td>
</tr>
</tbody>
</table>

**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.
val list_asm_ante_tac : TERM list \rightarrow TACTIC;

Description  Repeatedly apply \textit{asm_ante_tac}.

\[
\begin{array}{l}
\text{Rule} \\
\{ \Gamma, t_1, \ldots, t_n \} t \\
\{ \Gamma \} t_1 \Rightarrow \ldots \Rightarrow t_n \Rightarrow t \\
\text{list_asm_ante_tac} \\
\{ \Gamma \} t_1 \Rightarrow \ldots \Rightarrow t_n \Rightarrow t
\end{array}
\]

\(\alpha\)-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  \textit{asm_ante_tac}, \textit{all_asm_ante_tac}

Errors  
\begin{itemize}
\item 28052 Term 0 is not in the assumptions
\item 28055 The conclusion or an assumption of goal does not have type \(\lceil \text{BOOL} \rceil\)
\end{itemize}

val LIST_DROP_ASM_T : TERM list \rightarrow (THM list \rightarrow TACTIC) \rightarrow TACTIC;

Description  \textit{LIST_DROP_ASM_T} \([\text{asm}_1, \text{asm}_2, \ldots]\) \textit{thmtac} is a tactic which removes the \textit{asm}_1, \textit{asm}_2, \ldots from the assumption list and then acts as

\[\text{thmtac}[\langle \text{asm}_1 \vdash \text{asm}_1 \rangle, \langle \text{asm}_2 \vdash \text{asm}_2 \rangle, \ldots]\]

\textit{LIST_DROP_ASM_T}

Tactic

\[
\begin{array}{l}
\{ \Gamma, \text{asm}_1', \ldots \} t \\
\text{thmtac} \left[ \langle \text{asm}_1 \vdash \text{asm}_1 \rangle, \ldots \right] \left( \{ \Gamma \} t \right)
\end{array}
\]

where \textit{asm}_i and \textit{asm}_i' are \(\alpha\)-convertible.

Uses  To use assumptions as theorems

Errors  
\begin{itemize}
\item 9301 the term 0 is not in the assumption list
\end{itemize}

val LIST_DROP_NTH_ASM_T : int list \rightarrow (THM list \rightarrow TACTIC) \rightarrow TACTIC;

Description  \textit{LIST_DROP_NTH_ASM_T} \([i, j, \ldots]\) \textit{thmtac} is a tactic which applies \textit{asm\_rule} to the \(i\)-th, \(j\)-th, etc assumptions of the goal, giving theorems, \textit{thm}_i, \textit{thm}_j, etc, say, and then removes the \textit{asm}_i, \textit{asm}_j from the assumptions and acts as \textit{thmtac} \([\text{thm}_i, \text{thm}_j, \ldots]\).

\textit{DROP_NTH_ASM_T}

Tactic

\[
\begin{array}{l}
\{ a_1, \ldots, a_n \} t \\
\text{thmtac} \left[ \langle \text{asm\_rule} \ [a_i] \rangle \right], \left( \langle \text{asm\_rule} \ [a_j] \rangle \right), \ldots \left( \{ \Gamma \ \backslash \ \{ a_i, a_j, \ldots \} \} t \right)
\end{array}
\]

Uses  To use assumptions as theorems, treating the assumption list as an array.

Errors  
\begin{itemize}
\item 9303 the index 0 is out of range
\end{itemize}
8.3. 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

```
thmtac[(asm_1 \vdash asm_1), (asm_2 \vdash asm_2), ...]
```

**Tactic**

```
{ Γ', asm_1', ..., } t
________________________
thmtac [(asm_1 \vdash asm_1), ...]
      { { Γ', asm'_1', ... } t }
```

where `asm_i` and `asm_i'` are \(\alpha\)-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, ...].`

**Tactic**

```
{ a_i, ..., an } t
________________________
        thmtac [(asm_rule [a_i]), ...]
    {{ { Γ' } } t }
```

**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 interated existential subgoal.

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

```
{ Γ } \exists x_1, x_2, ... • t2[x_1, x_2, ...]
____________________________
{ Γ } t2[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: { Γ } \exists x • t2[x]`

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUAL USR029
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 \([\neg F]\).

Tactic
\[
\begin{align*}
\{ \Gamma \} & t2 & \text{list_swap_asm_concl_tac} \\
\{ \text{strip } \neg t2 \}, \Gamma & - \{ \neg t1, ..., \neg tn \} & [\neg t1, ..., \neg tn]
\end{align*}
\]

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
\[
\begin{align*}
\{ \Gamma \} & t & \text{LIST_SWAP_ASM_CONCL_T} \\
\text{ttac}(\text{asm_rule } \neg t) & \left( \{ \Gamma - \{ \neg tp, ..., \neg tq \} \} - t1 \right) & [\neg tp, ..., \neg tq]
\end{align*}
\]

Tactic
\[
\begin{align*}
\{ \Gamma \} & t & \text{LIST_SWAP_NTH_ASM_CONCL_T} \\
\text{ttac}(\text{asm_rule } \neg t) & \left( \{ \Gamma - \{ \neg tp, ..., \neg tq \} \} - tm \right) & [p, ..., q]
\end{align*}
\]

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 \([\neg F]\).

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 \([\neg F]\)
8.3. 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:

```
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 `ai`, then unused `fj` are ignored and the last `fj` to be used may receive less than `nj` 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`.
\[\text{val } \text{pair\_rw\_canon} : \text{CANON};\]

**Description**  This is the rewrite canonicalisation function for the theory of pairs, defined as

\[
\text{val pair\_rw\_canon} = \\
\text{REWRITE\_CAN} \\
(\text{REPEAT\_CAN}(\text{FIRST\_CAN} [ \forall\_\text{rewrite\_canon}, \wedge\_\text{rewrite\_canon}, \neg\_\text{rewrite\_canon}, f\_\text{rewrite\_canon}, \leftrightarrow_t\_\text{rewrite\_canon}]));
\]

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 \( \vdash \exists v s \bullet t \) to \( \forall v s \bullet \neg t \);
5. changing \( \vdash \neg t \) to \( t \leftrightarrow F \);
6. changing \( \vdash \neg t \) to \( t \leftrightarrow F \);
7. if none of the above apply, changing \( \vdash t \) to \( \vdash t \leftrightarrow T \).

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.

\[\text{val } \text{POP\_ASM\_T} : (\text{THM} \rightarrow \text{TACTIC}) \rightarrow \text{TACTIC};\]

**Description**  \( \text{POP\_ASM\_T } \text{thmtac} \) is a tactic which removes the top entry, \( \text{asm} \) say, from the assumption list and then acts as \( \text{thmtac}(\text{asm} \vdash \text{asm}) \).

\[
\text{Tactic} \\
\{ \text{asm}, \Gamma \} t \\
\text{thmtac} (\text{asm} \vdash \text{asm}) (\{ \Gamma \} t) \\
\text{POP\_ASM\_T} \\
\Gamma \vdash \text{asm} \rightarrow \text{thmtac}
\]

**Uses**  To use an assumption as a theorem

**Errors**  
9302  the assumption list is empty
8.3. General Tactics and Tacticals

SML

```sml
val prim_rewrite_tac : CONV NET -> CANON -> (THM -> TERM * CONV) OPT ->
                      (CONV -> CONV) -> EQN_CXT -> THM list -> 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:

<table>
<thead>
<tr>
<th>Tactic</th>
<th>prim_rewrite_tac</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ Γ } ⊢ t</code></td>
<td><code>prim_rewrite_tac</code></td>
</tr>
<tr>
<td><code>{ Γ } ⊢ t'</code></td>
<td>(initial_net: CONV NET)</td>
</tr>
<tr>
<td></td>
<td>(canon : CANON)</td>
</tr>
<tr>
<td></td>
<td>(epp : (THM -&gt; TERM * CONV) OPT)</td>
</tr>
<tr>
<td></td>
<td>(traverse : CONV -&gt; CONV)</td>
</tr>
<tr>
<td></td>
<td>(with_eqn_cxt : EQN_CXT)</td>
</tr>
<tr>
<td></td>
<td>(with_thms : THM list)</td>
</tr>
</tbody>
</table>

where `t'` is the result of rewriting `t` in the manner prescribed by the arguments.

SML

```sml
val prove_tac : THM list -> 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`.

<table>
<thead>
<tr>
<th>Tactic</th>
<th>prove_tac</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ Γ } t</code></td>
<td><code>prove_tac</code></td>
</tr>
<tr>
<td><code>current_ad_pr_tac()thms({}, t)</code></td>
<td><code>thms</code></td>
</tr>
<tr>
<td>THEN MAP_EVERY check_asm_tac Γ</td>
<td></td>
</tr>
</tbody>
</table>

**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.
val prove_thm : (string * TERM * TACTIC) -> 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 α-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 ⌜:BOOL⌝

See Also  simple_tac_proof, prove_thm.

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  

\[
\begin{align*}
\{ \Gamma \} & \quad t \\
\text{conv_tac} & (\text{current_ad_cs}_\exists_c\text{conv})({\{}}, t) \\
\text{THEN MAP_EVERY} & \text{check_asm_tac} \Gamma \\
\end{align*}
\]

prove_∃_tac thms

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.
8.3. 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 $\alpha$-convertible to its goal”.

For example,

Tactic

```sml
{k = 1}
(\forall i:N\times N\bullet \exists i:N\bullet i = 0)
\land (\forall j:N\times N\bullet (\exists k:N\bullet k = Fst j)
\land \forall j:N\bullet j = k)
\land (\forall \text{apple}\bullet (\exists \text{carrot}'\bullet \text{carrot}' = Fst \text{apple})
\land \forall \text{banana}\bullet \text{banana} = \text{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.
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.

REPEAT_T : TACTIC -> TACTIC;
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.

REPEAT_UNTIL_T1 : (GOAL -> bool) -> TACTIC -> TACTIC;
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

\[\text{REPEAT\_UNTIL1 (is\_or o snd) strip\_tac}\]

will repeatedly apply strip\_tac until all outstanding subgoals have disjunctive conclusions or cause strip\_tac to fail.

REPEAT_UNTIL_T : (TERM -> bool) -> TACTIC -> TACTIC;
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_T1 p tac is a tactic which has no effect.

REPEAT_UNTIL is an alias for REPEAT_UNTIL_T.

Example

\[\text{REPEAT\_UNTIL is\_or strip\_tac}\]

will repeatedly apply strip\_tac until all outstanding subgoals have disjunctive conclusions or cause strip\_tac to fail.
8.3. General Tactics and Tacticals

**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_rule`d assumptions of the goal.

**Errors**

| 26001 | no rewriting occurred |

Also as error 26003 and warning 26002 of `REWRITE_MAP.C` (q.v.).

**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.

**Errors**

| 26002 | rewriting gave ill-formed results on some subterms |
\[ \text{val} \ \text{ROTATE\_T} : \text{int} \to \text{TACTIC} \to \text{TACTIC}; \]

**Description**  \( \text{ROTATE\_T} \ i \ \text{tac} \) is a tactic which first applies \( \text{tac} \) 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 \( \text{tac} \).

Thus if the result of \( \text{tac} \) is:

\[
\begin{array}{c}
\{ \Gamma \} \ t \\
\{ \Gamma_1 \} \ t_1; \ldots; \{ \Gamma_k \} \ t_k
\end{array}
\]

then the result of \( \text{ROTATE\_T} \ i \ \text{tac} \) will be:

\[
\begin{array}{c}
\{ \Gamma \} \ t \\
\{ \Gamma_{i+1} \} \ t_{i+1}; \ldots; \{ \Gamma_k \} \ t_k; \\
\{ \Gamma_1 \} \ t_1; \ldots; \{ \Gamma_i \} \ t_i
\end{array}
\]

**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 \( \text{tac} \).

---

\[ \text{val} \ \text{simple\_tac\_proof} : (\text{GOAL} \times \text{TACTIC}) \to \text{THM}; \]

**Description**  \( \text{simple\_tac\_proof}(\text{gl}, \text{tac}) \) applies the tactic \( \text{tac} \) to the goal \( \text{gl} \), and, if the tactic returns no unsolved subgoals returns the theorem proved by the tactic.

Infelicities in the coding of the tactic may cause the theorem returned to be rather different from the specified goal (in general, a “successful” application of a correctly coded tactic will return a theorem which may require addition of assumptions and \( \alpha \)-conversion to give the desired goal). \( \text{tac\_proof} \) should be used rather than \( \text{simple\_tac\_proof} \) if it is important that the theorem should achieve the goal precisely.

**Uses**  In programming tactics or other proof procedures where speed is important and the extra care taken by \( \text{tac\_proof} \) is not required.

**Errors**  
9501 the tactic returned unsolved subgoals: ?0
9502 evaluation of the tactic failed: ?0
9503 the proof returned by the tactic failed: ?0

**See Also**  \( \text{tac\_proof} \)
8.3. 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:

\[ \land, \lor, \rightarrow, \leftrightarrow, \neg, if \ ... \ then \ ... \ else \]

and which is true for any assignment of truth values to the variables.

Tactic

\[ \{ \Gamma \} \ t \]


**simple_taut_tac**

See Also  *strip_tac*

Errors  

28121  *Conclusion of the goal is not a tautology*

---

SML

val simple_¬in_conv : CONV;

**Description**  This is a conversion which moves negations inside other predicate calculus connectives using whichever of the following rules applies:

\[ \neg t \]
\[ \neg(t1 \land t2) = \neg t1 \lor \neg t2 \]
\[ \neg(t1 \lor t2) = \neg t1 \land \neg t2 \]
\[ \neg(t1 \Rightarrow t2) = t1 \land \neg t2 \]
\[ \neg(t1 \Leftrightarrow t2) = (t1 \land \neg t2) \lor (t2 \land \neg t1) \]
\[ \neg(if \ a \ then \ t1 \ else \ t2) = (if \ a \ then \ \neg t1 \ else \ \neg t2) \]
\[ \neg\forall x \cdot t = \exists x \cdot \neg t \]
\[ \neg\exists x \cdot t = \forall x \cdot \neg t \]
\[ \neg\exists x \cdot \neg t = \forall x \cdot t \land \forall x' \cdot t[x'] \Rightarrow x' = x \]
\[ \neg T = F \]
\[ \neg F = T \]

It does not handle paired quantifiers.

**Uses**  Tactic and conversion programming. The more general \(\neg in\_conv\) is just as efficient as *simple_¬in_conv* in cases where both succeed.

See Also  *strip_tac*

Errors

28131  *No applicable rules for the term ?0*
val simple_¬_in_tac : TACTIC;

Description  This is a tactic which moves negations inside other predicate calculus connectives using the following rules:

\[\neg \neg t \rightarrow t\]
\[-(t1 \land t2) \rightarrow \neg t1 \lor \neg t2\]
\[-(t1 \lor t2) \rightarrow \neg t1 \land \neg t2\]
\[-(t1 \Rightarrow t2) \rightarrow t1 \land \neg t2\]
\[-(t1 \Leftrightarrow t2) \rightarrow (t1 \land \neg t2) \lor (t2 \land \neg t1)\]
\[-\forall x \cdot t \rightarrow \exists x \cdot \neg t\]
\[-\exists x \cdot t \rightarrow \forall x \cdot \neg t\]
\[-\exists_1 x \cdot t \rightarrow \forall x \cdot (t \land \forall x' \cdot t[x'] \Rightarrow x' = x)\]
\[-T \rightarrow F\]
\[-F \rightarrow \text{goal solved}\]

It fails with paired quantifiers.

Uses  The more general \_¬_in_tac is just as efficient as simple_¬_in_tac in cases where both succeed.

See Also  strip_tac, contr_tac, ¬_T, ¬_in_tac

Errors  [28025  No applicable rule for this goal]

val SIMPLE_¬_IN_THEN : THM_TACTICAL;

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:

\[\neg \neg t \rightarrow t\]
\[-(t1 \land t2) \rightarrow \neg t1 \lor \neg t2\]
\[-(t1 \lor t2) \rightarrow \neg t1 \land \neg t2\]
\[-(t1 \Rightarrow t2) \rightarrow t1 \land \neg t2\]
\[-(t1 \Leftrightarrow t2) \rightarrow (t1 \land \neg t2) \lor (t2 \land \neg t1)\]
\[-\forall x \cdot t \rightarrow \exists x \cdot \neg t\]
\[-\exists x \cdot t \rightarrow \forall x \cdot \neg t\]
\[-\exists_1 x \cdot t \rightarrow \forall x \cdot (t \land \forall x' \cdot t[x'] \Rightarrow x' = x)\]
\[-T \rightarrow F\]
\[-F \rightarrow T\]

The function may be partially evaluated with only its theorem tactic and theorem arguments. It fails with paired quantifiers.

Uses  The more general \_IN_THEN is just as efficient as SIMPLE_¬_IN_THEN in cases where both succeed.

See Also  strip_tac, STRIP_THM_THEN

Errors  [28026  No applicable rule for this theorem]
8.3. General Tactics and Tacticals

```sml
val simple_∀_tac : TACTIC;
```

**Description**  Reduce a universally quantified goal. It fails with paired quantifiers.

Tactic

\[
\begin{array}{c}
\{ \Gamma \} \forall x \cdot t[x] \\
\{ \Gamma' \} t[x']
\end{array}
\]

\( \text{simple}_\forall_\text{tac} \)

where \( x' \) is a variant name of \( x \), different from any variable in \( \Gamma \) or \( t \).

**Uses**  Tactic programming. The more general \( \forall_\text{tac} \) is just as efficient as \( \text{simple}_\forall_\text{tac} \) in cases where both succeed.

**See Also**  \( \forall_\text{tac} \)

**Errors**

28081  *Goal is not of the form: \( \{ \Gamma \} \forall x \cdot t[x] \)*

```sml
val simple_∃_tac : TERM -> TACTIC ;
```

**Description**  Provide a witness for an existential subgoal. 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

\[
\begin{array}{c}
\{ \Gamma' \} \exists x \cdot t2[x] \\
\{ \Gamma \} t2[t1]
\end{array}
\]

\( \text{simple}_\exists_\text{tac} \)

where \( t1 \) must have the same type as \( x \).

**Uses**  Tactic programming. The more general \( \exists_\text{tac} \) is just as efficient as \( \text{simple}_\exists_\text{tac} \) in cases where both succeed.

**Errors**

28091  *Goal is not of the form: \( \{ \Gamma \} \exists x \cdot t2[x] \)*

28092  *Term ?0 has the wrong type*
Chapter 8. PROOF IN HOL

SML

|val| SIMPLE.∃.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 \( \Gamma \vdash \exists x \cdot t \). It fails with paired quantifiers.

\[ SIMPLE.∃.THEN \ thmtac \ (\Gamma \vdash \exists x \cdot t) = thmtac \ (\Gamma \vdash t[x'/x]) \]

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 \( \exists .THEN \) is just as efficient as \( SIMPLE.∃.THEN \) in cases where both succeed.

Error 28094 normally arises when \( x' \) is also introduced by the proof of \( ttac \), and occurs during the application of the proof of \( SIMPLE.∃.THEN \). The bound variable \( x' \) should be renamed to something that doesn’t cause this identification of distinct variables, by using rename_tac(q.v.).

See Also  \( \exists .THEN \)

Errors

| 28093 | ?0 is not of the form: ‘\( \Gamma \vdash \exists x \bullet t' \)’ |
| 28094 | Error in proof of SIMPLE.∃.THEN. |
|        | Usually indicates chosen skolem variable ?0 also |
|        | introduced by proof of supplied theorem tactic, |
|        | which gave ‘?1’, and the two became identified: |
|        | use rename_tac to rename original bound variable ?2 |

SML

|val| simple.∃.conv : CONV; |

Description  This is a conversion which turns a unique existential quantifier into an equivalent existential quantifier

\[
\vdash (\exists_1 x \cdot t[x]) \iff (\exists x \cdot t\,[x] \land \forall x' \cdot t\,[x'] \Rightarrow x' = x)
\]

Uses  Tactic and conversion programming. The more general \( \exists.1.conv \) is just as efficient as \( simple.∃.1.conv \) in cases where both succeed.

See Also  strip_tac

Errors

| 4019 | ?0 is not of form: ‘∃_1 v \bullet t' |

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
8.3. General Tactics and Tacticals

**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

\[ \{ \Gamma \} \; \text{simple}_\exists \, x \bullet P[x] \]

\[ \{ \Gamma \} \; P[t]; \]

\[ \{ \Gamma \} \; \forall x' \bullet P[x'] \Rightarrow x' = t \]

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 \( \text{simple}_\exists \_ 1 \_ \text{tac} \) in cases where both succeed.

**Errors**

28101  Goal is not of the form: \( \{ \Gamma \} \; \exists \, x \bullet P[x] \)

28092  Term ?0 has the wrong type

**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 \( \Gamma \vdash \exists \_ 1 \_ x \bullet t \). It fails with paired quantifiers.

\[ \text{SIMPLE}_\exists \_ 1 \_ \text{THEN} \; \text{thmtac} \; (\Gamma \vdash \exists \_ 1 \_ x \bullet t) = \]

\[ \text{thmtac} \; (\Gamma \vdash \exists \_ 1 \_ x \bullet t) = \]

\[ (\Gamma \vdash \text{⌜x'}⌝/x \wedge \forall \text{⌜x''⌝} \bullet P[\text{⌜x''⌝}] \Rightarrow x'' = \text{⌜x'}⌝) \]

where \( \text{⌜x'}⌝ \) and \( \text{⌜x''⌝} \) are distinct variants of \( \text{⌜x⌝} \) 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: \( ^\text{⌜} \Gamma \vdash \exists \_ 1 \_ x \bullet t ^\text{⌝} \)

**SOLVED_T** : TACTIC → TACTIC;

**Description**  SOLVED_T tac is a tactic which applies 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.

SOLVED_T does not check that the proof delivered by the tactic is valid. tac_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**  tac_proof

**Errors**

9602  the tactic did not solve the goal
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
\[ \{ \Gamma, \forall vs[x1',...], f[x1',...], t \} \]
\[ \{ \text{strip } f[t1,...], \Gamma, \\
\forall vs[x1,...], f[x1,...], t[1] \} \]

The following all handle paired abstractions in a similar manner.

Tactic
\[ \{ \Gamma, \forall x1 ... f[x1,...], t \} \]
\[ \{ \text{strip } f[t1,...], \Gamma, \\
\forall x1 x2 ... f[x1,...], t \} \]

Tactic
\[ \{ \Gamma 1...n-1, \forall x' f[x'], \Gamma \}, t1 \]
\[ \{ \text{strip } f[t2], \Gamma 1...n-1, \\
\forall x' f[x'], t1 \} \]

Tactic
\[ \{ \Gamma 1...n-1, \forall x1 ... f[x1,...], \Gamma \}, t \]
\[ \{ \text{strip } f[t1,...], \Gamma 1...n-1, \\
\forall x1 ... f[x1,...], t \} \]

Definitions
\[ \text{fun spec_asm_tac asm instance } = \]
\[ \text{GET_ASM_T asm (strip_asm_tac o } \forall \text{ elim instance); } \]
\[ \text{fun list_spec_asm_tac asm instances } = \]
\[ \text{GET_ASM_T asm (strip_asm_tac o list } \forall \text{ elim instances);} \]
\[ \text{fun spec_nth_asm_tac n instance } = \]
\[ \text{GET_NTH_ASM_T n (strip_asm_tac o } \forall \text{ elim instance); } \]
\[ \text{fun list_spec_nth_asm_tac n instances } = \]
\[ \text{GET_NTH_ASM_T n (strip_asm_tac o list } \forall \text{ elim instances);} \]

Errors As the constituents of the implementing functions.
8.3. General Tactics and Tacticals

Tactic

\[
\frac{
\{ \Gamma, \forall vs[x1',\ldots], f [x1',\ldots] \} t
\}
\text{thm}_tac (asm_rule \ f [t1,t2,\ldots] t2)
}{
\{ \Gamma, \forall vs[x1,\ldots], f [x1,\ldots] \} t1
}\]

\[SPEC_{ASM \ T}\]

\[\forall vs[x1,\ldots], f [x1,\ldots] t\]

\[\text{thm}_tac\]

The following all handle paired abstractions in a similar manner.

Tactic

\[
\frac{
\{ \Gamma, \forall x1 x2 \ldots, f [x1,x2,\ldots] \} t
\}
\text{thm}_tac (asm_rule \ f [t1,t2,\ldots] t2)
}{
\{ \Gamma, \forall x1 x2 \ldots, f [x1,x2,\ldots] \} t1
}\]

\[LIST_{SPEC_{ASM \ T}}\]

\[\forall x1 x2 \ldots, f [x1,x2,\ldots] t\]

\[\text{thm}_tac\]

Tactic

\[
\frac{
\{ \Gamma1, n-1, \forall x', f [x'] \}, \Gamma \} t1
\}
\text{thm}_tac (asm_rule \ f [t1,t2] t2)
}{
\{ \Gamma1, n-1, \forall x', f [x'] \} t1
}\]

\[SPEC_{NTH_{ASM \ T}}\]

\[n \]

\[\text{thm}_tac\]

Tactic

\[
\frac{
\{ \Gamma1, n-1, \forall x, f [x] \}, \Gamma \} t
\}
\text{thm}_tac (asm_rule \ f [t1,t2] t2)
}{
\{ \Gamma1, n-1, \forall x, f [x] \} t
}\]

\[LIST_{SPEC_{NTH_{ASM \ T}}\ T}\]

\[n \]

\[\text{thm}_tac\]

Definitions

\[
\begin{align*}
\text{fun} \ SPEC_{ASM \ T} \text{ asm instance thmtac} &= \text{GET_{ASM \ T} asm (thmtac o \forall\text{ elim instance)};} \\
\text{fun} \ LIST_{SPEC_{ASM \ T}} \text{ asm instances thmtac} &= \text{GET_{ASM \ T} asm (thmtac o list\forall\text{ elim instances)};} \\
\text{fun} \ SPEC_{NTH_{ASM \ T}} \text{ n instance thmtac} &= \text{GET_{NTH_{ASM \ T}} n (thmtac o \forall\text{ elim instance)};} \\
\text{fun} \ LIST_{SPEC_{NTH_{ASM \ T}}} \text{ n instances thmtac} &= \text{GET_{NTH_{ASM \ T}} n (thmtac o list\forall\text{ elim instances)};} \\
\end{align*}
\]

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
8.3. General Tactics and Tacticals

**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`:

- \[ \text{sat}(\neg a \land b) \rightarrow \text{sat}(\neg a) \text{ THEN } \text{sat}(\neg b) \]
- \[ \text{sat}(\exists x a) \rightarrow \text{sat}(a[x'/x] \vdash a[x'/x]) \]
- \[ \text{sat}(\neg a \lor b)(\{\Gamma\} t) \rightarrow \text{sat}(a \vdash a)(\{\Gamma\} t) ; \text{sat}(b \vdash b)(\{\Gamma\} t) \]

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`.

**Errors**

28003 There is no stripping technique for ?0 in the current proof context
SML

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, 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.

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∨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

SML

val STRIP_CONCL_T : (THM -> TACTIC) -> TACTIC;
val STRIP_T : (THM -> TACTIC) -> 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):

| simple∀tac, ∧_tac, ⇒_T ttac, t_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
8.3. General Tactics and Tacticals

**val STRIP_THM_THEN : THM_TACTICAL;**

**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):

- $\land\_THEN$
- $\lor\_THEN$
- SIMPLE $\exists\_THEN$

The conversion in the current proof context got by current_ad_st_conv (q.v.) is derived by applying eqn_cxt_conv to an equational context in the proof context extracted by get_st_eqn_cxt.

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

**val swap_asm_concl_tac : TERM -> TACTIC;**
**val swap_nth_asm_concl_tac : int -> TACTIC;**

**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

$contr\_tac$ THEN $asm\_ante\_tac$ asm

and swap_nth_asm_concl_tac nis equivalent to

Example

$contr\_tac$ THEN DROP_NTH_ASM $T$ $n$ $ante\_tac$

Tactic

\[
\frac{\{ \Gamma, \neg t1 \} t2}{\{ \text{strip } \neg t2, \Gamma \} \neg t1}
\]

swap_asm_concl_tac

\[
\neg t1
\]

Tactic

\[
\frac{\{ \neg t1, \neg tm, \ldots, \neg tn \} t}{\{ \text{strip } \neg t, \neg t1, \ldots, \neg tn \} \neg tm}
\]

swap_nth_asm_concl_tac

\[
m
\]

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
val SWAP_ASM_CONCL_T : TERM -> (THM -> TACTIC) -> TACTIC;
val SWAP_NTH_ASM_CONCL_T : int -> (THM -> TACTIC) -> TACTIC;

Description  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,
SWAP_ASM_CONCL_T asm ttac is equivalent to

Example

CONTR_T (fn x => asm_ante_tac asm THEN ttac x)

and SWAP_NTH_ASM_CONCL_T n ttac is equivalent to

Example

CONTR_T (fn x => (DROP_NTH_ASM_T n ante_tac) THEN ttac x)

Tactic

\[
\begin{array}{c}
\{ \Gamma, \neg t1 \} t2 \\
\hline
\text{ttac(asm_rule } \neg \neg t2) (\{\Gamma\} \neg t1)
\end{array}
\]

SWAP_ASM_CONCL_T

\[
\begin{array}{c}
\{ \Gamma, \neg t1 \} \\
\hline
\text{ttac}
\end{array}
\]

Tactic

\[
\begin{array}{c}
\{ \Gamma, \neg t1, \ldots, \neg tm, \ldots, \neg tn \} t \\
\hline
\text{ttac(asm_rule } \neg \neg t) (\{\Gamma, \neg t1, \ldots, \neg tn \} \neg tm)
\end{array}
\]

SWAP_NTH_ASM_CONCL_T

\[
\begin{array}{c}
m \\
\hline
\text{ttac}
\end{array}
\]

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
28027  Conclusion of goal does not have type \neg:BOOL}
28052  Term ?0 is not in the assumptions

val swap\_\_tac : TACTIC;

Description  Interchange the disjuncts of a disjunctive goal.

Tactic

\[
\begin{array}{c}
\{ \Gamma \} a \lor b \\
\hline
\{ \Gamma \} b \lor a
\end{array}
\]

swap\_\_tac

Uses  For use in conjunction with strip\_tac (q.v.) when the reduction of \{\Gamma\}a\lor b to \{\neg a, \Gamma\}b
is inappropriate.

See Also  \lor\_left\_tac, \lor\_right\_tac, swap\_\_tac, strip\_tac

Errors

28041  Goal is not of the form: \{ \Gamma \} a \lor b

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
8.3. 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 $\alpha$-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 $\alpha$-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
9504 the proof returned by the tactic proved $?0 which could not be converted into the desired goal.
9505 the goal contains $\alpha$-equivalent assumptions (?0 and ?1)
9506 the assumption ?0 is not of type $\langle BOOL \rangle$
9507 the conclusion ?0 is not of type $\langle BOOL \rangle$

**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:

$\land$, $\lor$, $\Rightarrow$, $\Leftrightarrow$, $\neg$, if ... then ... 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**

\[ \vdash t \Leftrightarrow \top \]

**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.

\[ \vdash t \quad \text{taut_rule} \quad \{ t \} \]

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.
8.3. General Tactics and Tacticals

SML

val THEN1 : (TACTIC * TACTIC) -> TACTIC;
val THEN : (TACTIC * TACTIC) -> TACTIC;

Description  THEN_1 is a tactical used as an infix operator. tac1 THEN_1 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_1 tac2. THEN1 is an alias for THEN_1.

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_T tlist is a tactic that applies tac, and then attempts to apply 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. If any member of tlist 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:

\((tcl1 \space THEN\_TTCL\space tcl2) \space thmtac \space thm = (tcl1 \circ tcl2) \space thmtac \space thm\)

Uses  For use in programming theorem tacticals.
THEN : (TACTIC * TACTIC) -> TACTIC;

Description  THEN is a tactical used as an infix operator. tac1 THEN tac2 is the tactic that applies tac1 and then applies tac2 to each resulting subgoal (perhaps none). THEN is an alias for THEN.

Errors  As the errors of tac1 and tac2.

TOP_ASM : (THM -> TACTIC) -> TACTIC;

Description  If the top entry in the assumption list is asm say, TOP_ASM thmtac acts as thmtac(asm \vdash asm).

Tactic  \[ \frac{\{ \text{asm}, \Gamma \} \ t}{\text{thmtac (asm \vdash \text{asm}) (\{ \text{asm}, \Gamma \} \ t)}} \]

Uses  To use an assumption as a theorem.

Errors  9302 the assumption list is empty

TRY_TTCL : THM_TACTICAL -> THM_TACTICAL;

Description  TRY_TTCL ttcl is a theorem tactical which applies ttcl if it can, and otherwise acts as ID_THEN.

Uses  For use in programming theorem tacticals.

TRY_T : TACTIC -> TACTIC;

Description  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 tac ORELSE id_tac. TRY is an alias for TRY_T.

t_tac : TACTIC;

Description  Prove a goal with conclusion ‘T’.

Tactic  \[ \frac{\{ \Gamma \} \ T}{t\_tac} \]

See Also  strip_tac, taut_tac.

Uses  Tactic programming.

Errors  28011 Goal does not have the form \{\Gamma\} T
8.3. 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: \( \text{⌜var = value\⌟} \) or \( \text{⌜value = var\⌟} \), where var is a variable and, if the subterm value does not contain var free, they substitute value for the free variable 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.

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 var with value throughout the goal, stripping any changed assumptions back into the goal (returning the rest by check_asm_tac), and then discard the original assumption. VAR_ELIM_ASM_T allows the users choice of function to be applied to the modified assumptions.

var_elim_nth_asm_tac works as var_elim_asm_tac, except it takes an integer indicating the "nth" assumption is to be used. VAR_ELIM_NTH_ASM_T allows the users choice of function to be applied to the modified assumptions.

See Also all_var_elim_asm_tac1 and its kin to apply this sort of functionality to all the assumptions simultaneously. 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 \( \text{⌜var = ...\⌟} \) or \( \text{⌜... = var\⌟} \) where the variable \( \text{⌜var\⌟} \) is not free in \( \text{⌜...\⌟} \)

SML

val \( ⇐_T2 \) : (THM -> TACTIC) -> (THM -> TACTIC) -> TACTIC;

Description Reduce a bi-implication by passing the operands to tactic generating functions.

Tactic

\[
\begin{array}{c}
\text{ttac1}\{ \text{t1, } \Gamma \} \vdash \text{t2; ttac2}\{ \text{t2, } \Gamma \} \vdash \text{t1} \\
\end{array}
\]

\( ⇐_T2 \) ttac1 ttac2

See Also \( ⇐_T, \text{STRIP\_CONCL\_T} \)

Errors

28061 Goal is not of the form: \( \{ \Gamma \} \text{t1} \leftrightarrow \text{t2} \)
\[\text{SML} \quad \text{val } \leftrightarrow\_\text{tac} : \text{TACTIC};\]

**Description**  
Reduce a bi-implication to two subgoals.

\[
\begin{array}{c}
\text{Tactic} \\
\{ \Gamma \} t \leftrightarrow T \\
\{ \Gamma \} t \\
\{ \Gamma \} t \leftrightarrow T \\
\end{array} \Rightarrow \leftrightarrow\_\text{tac}
\]

**See Also**  
\(\leftrightarrow\_T\)

**Errors**  
28012 Goal not of form: \{ \Gamma \} t \leftrightarrow T or \{ \Gamma \} t \leftrightarrow T

**Uses**  
Tactic programming.

---

\[\text{SML} \quad \text{val } \leftrightarrow\_\text{THEN2} : (\text{THM} \to \text{TACTIC}) \to (\text{THM} \to \text{TACTIC}) \to (\text{THM} \to \text{TACTIC});\]

**Description**  
A theorem tactical to apply given theorem tactics to the the result of eliminating \(\leftrightarrow\) from a theorem of the form \(\Gamma \vdash t_1 \leftrightarrow t_2\).

\[
\leftrightarrow\_\text{THEN2} \text{ ttac1 ttac2}(\Gamma \vdash t_1 \leftrightarrow t_2) = \text{ttac1}(\Gamma \vdash t_1 \Rightarrow t_2) \text{ THEN ttac2}(\Gamma \vdash t_2 \Rightarrow t_1)
\]

The function is partially evaluated with only the theorem tactic and theorem arguments.

**See Also**  
\(\leftrightarrow\_\text{THEN}, \text{STRIP\_THM\_THEN}\)

**Errors**  
28062 ?0 is not of the form: '\(\Gamma \vdash t_1 \leftrightarrow t_2\)'

---

\[\text{SML} \quad \text{val } \leftrightarrow\_\text{THEN} : (\text{THM} \to \text{TACTIC}) \to (\text{THM} \to \text{TACTIC});\]

**Description**  
A theorem tactical to apply a given theorem tactic to the result of eliminating \(\leftrightarrow\) from a theorem of the form \(\Gamma \vdash t_1 \leftrightarrow t_2\).

\[
\leftrightarrow\_\text{THEN} \text{ thmtac}(\Gamma \vdash t_1 \leftrightarrow t_2) = \text{thmtac}(\Gamma \vdash t_1 \Rightarrow t_2) \text{ THEN thmtac}(\Gamma \vdash t_2 \Rightarrow t_1)
\]

The function is partially evaluated with only the theorem tactic and theorem arguments.

**See Also**  
\(\leftrightarrow\_\text{THEN2}, \text{STRIP\_THM\_THEN}\)

**Errors**  
28062 ?0 is not of the form: '\(\Gamma \vdash t_1 \leftrightarrow t_2\)'

---

\[\text{SML} \quad \text{val } \leftrightarrow\_\text{t_tac} : \text{TACTIC};\]

**Description**  
Simplifies a goal of the form: \(...)\leftrightarrow T or T\leftrightarrow\(...\).

\[
\begin{array}{c}
\text{Tactic} \\
\{ \Gamma \} t \leftrightarrow T \\
\{ \Gamma \} t \\
\{ \Gamma \} t \leftrightarrow T \\
\end{array} \Rightarrow \leftrightarrow\_\text{t_tac}
\]

and

\[
\begin{array}{c}
\text{Tactic} \\
\{ \Gamma \} t \leftrightarrow T \\
\{ \Gamma \} t \\
\{ \Gamma \} t \leftrightarrow T \\
\end{array} \Rightarrow \leftrightarrow\_\text{t_tac}
\]

**Errors**  
28012 Goal not of form: \{ \Gamma \} t \leftrightarrow T or \{ \Gamma \} T \leftrightarrow t

**See Also**  
\(\text{strip\_tac}\)

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
8.3. General Tactics and Tacticals

SML
val _T : (THM -> TACTIC) -> TACTIC;

**Description** Reduce a bi-implication by passing each operand to a tactic generating function.

<table>
<thead>
<tr>
<th>Tactic</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ Γ } t1 ⇔ t2 [ \rightarrow ] ttac { t1, Γ } \vdash t2; ttac { t2, Γ } \vdash t1 [ \rightarrow ] ttac</td>
</tr>
</tbody>
</table>

**See Also** ⇔_T, STRIP_CONCL_T

**Errors**
[28061 Goal is not of the form: \{ Γ \} t1 ⇔ t2]

---

SML
val _tac : TACTIC;

**Description** Reduce the proof of a conjunction to the proof of its conjuncts.

<table>
<thead>
<tr>
<th>Tactic</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ Γ } t1 ^ t2 [ \rightarrow ] { Γ } t1; { Γ } t2 [ \rightarrow ] _tac</td>
</tr>
</tbody>
</table>

**See Also** strip_tac

**Errors**
[28031 Goal is not of the form: \{ Γ \} t1 ^ t2]

---

SML
val _THEN2 : (THM -> TACTIC) -> (THM -> TACTIC) -> (THM -> TACTIC);

**Description** A theorem tactical to apply given theorem tactics to the conjuncts of a theorem of the form \( Γ \vdash t1 ^ t2 \).

\[ _\text{THEN2} \; \text{thmtac1 \; thmtac2} \; (Γ \vdash t1 ^ t2) = \text{thmtac1} \; (Γ \vdash t1) \; \text{THEN} \; \text{thmtac2} \; (Γ \vdash t2) \]

**See Also** _THEN, STRIP_THM_THEN

**Errors**
[28032 ?0 is not of the form: \'Γ \vdash t1 ^ t2\']

---

SML
val _THEN : (THM -> TACTIC) -> (THM -> TACTIC);

**Description** A theorem tactical to apply a given theorem tactic to the conjuncts of a theorem of the form \( Γ \vdash t1 ^ t2 \).

\[ _\text{THEN} \; \text{thmtac} \; (Γ \vdash t1 ^ t2) = \text{thmtac} \; (Γ \vdash t1) \; \text{THEN} \; \text{thmtac} \; (Γ \vdash t2) \]

The function may be partially evaluated with only its theorem tactic and theorem arguments.

**See Also** _THEN2, STRIP_THM_THEN

**Errors**
[28032 ?0 is not of the form: \'Γ \vdash t1 ^ t2\']

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
SML
\[\text{val } \lor\text{ left_tac : TACTIC;}
\]

**Description**  Take the left disjunct of the current goal as the subgoal.

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

Tactic
\[
\frac{\{ \Gamma \} \ a \lor b}{\{ \Gamma' \} \ a}
\]
\[\lor\text{ left_tac}\]

**See Also**  \lor\text{ left_tac}, swap \lor\text{ tac}, strip\text{ tac}

**Errors**  \[28041 \text{ Goal is not of the form: } \{ \Gamma \} \ a \lor b\]

---

SML
\[\text{val } \lor\text{ right_tac : TACTIC;}
\]

**Description**  Take the right disjunct of the current subgoal as the new subgoal.

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the original goal was provable.

Tactic
\[
\frac{\{ \Gamma \} \ a \lor b}{\{ \Gamma' \} \ b}
\]
\[\lor\text{ right_tac}\]

**See Also**  \lor\text{ right_tac}, swap \lor\text{ tac}, strip\text{ tac}

**Errors**  \[28041 \text{ Goal is not of the form: } \{ \Gamma \} \ a \lor b\]

---

SML
\[\text{val } \lor\text{ THEN2 : (THM \rightarrow TACTIC) \rightarrow (THM \rightarrow TACTIC) \rightarrow (THM \rightarrow TACTIC);}\]

**Description**  A theorem tactical to perform a case split on a given disjunctive theorem applying tactic generating functions to the extra assumption in each branch.

\[\lor\text{ THEN2 ttac1 ttac2 (}\Delta \vdash t1 \lor t2)(\{\Gamma\} t) =
\]
\[ttac1 (t1 \vdash t1)(\{\Gamma\} t); ttac2 (t2 \vdash t2)(\{\Gamma\} t)\]

The function may be partially evaluated with only its theorem tactic and theorem arguments.

**See Also**  STRIP_THM\_THEN, \lor\_THEN

**Errors**  \[28042 \text{ ?0 is not of the form: } '\Gamma \vdash t1 \lor t2'\]

---

SML
\[\text{val } \lor\text{ THEN : (THM \rightarrow TACTIC) \rightarrow (THM \rightarrow TACTIC);}\]

**Description**  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.

\[\lor\text{ THEN ttac (}\Delta \vdash t1 \lor t2)(\{\Gamma\} t) =
\]
\[ttac (t1 \vdash t1)(\{\Gamma\} t); ttac (t2 \vdash t2)(\{\Gamma\} t)\]

The function may be partially evaluated with only its theorem tactic and theorem arguments.

**See Also**  STRIP_THM\_THEN, \lor\_THEN2

**Errors**  \[28042 \text{ ?0 is not of the form: } '\Gamma \vdash t1 \lor t2'\]
8.3. General Tactics and Tacticals

SML
\[ \text{val } \neg_{\text{elim_tac}} : \text{TERM } \rightarrow \text{TACTIC}; \]

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.

\[ \text{Tactic} \]
\[ \frac{\{ \Gamma \} \ t2}{\{ \Gamma \} \ t1; \ \{ \Gamma \} \ \neg t1} \]
\[ \neg_{\text{elim_tac}} \quad \neg \ t1 \]

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. \textit{asm\_ante\_tac} \textit{t1} \text{ THEN } \textit{strip\_tac} is a more memorable idiom for handling such a case in interactive use but is a little slower.

See Also  \textit{strip\_tac}

Errors  \[28022 \ ?0 \text{ is not boolean}\]

SML
\[ \text{val } \neg_{\text{in_conv}} : \text{CONV}; \]

Description  This is a conversion which moves a top level negation inside other predicate calculus connectives using whichever one of the following rules applies:

\[ \neg \neg t = t \]
\[ \neg (t1 \land t2) = \neg t1 \lor \neg t2 \]
\[ \neg (t1 \lor t2) = \neg t1 \land \neg t2 \]
\[ \neg (t1 \Rightarrow t2) = t1 \land \neg t2 \]
\[ \neg (t1 \Leftrightarrow t2) = (t1 \land \neg t2) \lor (t2 \land \neg t1) \]
\[ \neg (\text{if } a \text{ then } t1 \text{ else } t2) = (\text{if } a \text{ then } \neg t1 \text{ else } \neg t2) \]
\[ \neg \forall vs\cdot t = \exists vs\cdot \neg t \]
\[ \neg \exists vs\cdot t = \forall vs\cdot \neg t \]
\[ \neg \exists 1\ vs\cdot t = \forall vs\cdot \neg (t \land \forall vs'\cdot t[vs'] \Rightarrow vs' = vs) \]
\[ \neg T = F \]
\[ \neg F = T \]

Uses  Tactic and conversion programming.

See Also  \textit{simple\_\neg\_in\_conv}, \textit{\neg\_in\_tac}

Errors  \[28131 \text{ No applicable rules for the term } ?0\]

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
\begin{verbatim}
val \texttt{\neg\_in\_tac} : TACTIC;

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 & \to t \\
\neg (t_1 \land t_2) & \to \neg t_1 \lor \neg t_2 \\
\neg (t_1 \lor t_2) & \to \neg t_1 \land \neg t_2 \\
\neg (t_1 \Rightarrow t_2) & \to t_1 \land \neg t_2 \\
\neg (t_1 \iff t_2) & \to (t_1 \land \neg t_2) \lor (t_2 \land \neg t_1) \\
\neg \forall v_1 \cdot t & \to \exists v_1 \cdot \neg t \\
\neg \exists v_1 \cdot t & \to \forall v_1 \cdot \neg t \\

\neg \exists v_1 \cdot \forall v_2 \cdot t & \to \forall v_2 \cdot (t \land \forall v_1' \cdot t[v_2'] \Rightarrow v_2' = v_1) \\
\neg T & \to F \\
\neg F & \to T
\end{align*}

Uses

See Also \texttt{simple\_\neg\_in\_tac}, \texttt{\neg\_in\_conv}

Errors
\texttt{28025} No applicable rule for this goal
\end{verbatim}

\begin{verbatim}
val \texttt{\neg\_IN\_THEN} : THM\_TACTICAL;

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 & \to t \\
\neg (t_1 \land t_2) & \to \neg t_1 \lor \neg t_2 \\
\neg (t_1 \lor t_2) & \to \neg t_1 \land \neg t_2 \\
\neg (t_1 \Rightarrow t_2) & \to t_1 \land \neg t_2 \\
\neg (t_1 \iff t_2) & \to (t_1 \land \neg t_2) \lor (t_2 \land \neg t_1) \\
\neg \forall v_1 \cdot t & \to \exists v_1 \cdot \neg t \\
\neg \exists v_1 \cdot t & \to \forall v_1 \cdot \neg t \\
\neg \exists v_1 \cdot \forall v_2 \cdot t & \to \forall v_2 \cdot (t \land \forall v_1' \cdot t[v_2'] \Rightarrow v_2' = v_1) \\
\neg T & \to F \\
\neg F & \to T
\end{align*}

This function partially evaluates given only the theorem and theorem-tactical.

See Also \texttt{SIMPLE\_\neg\_IN\_THEN}

Errors
\texttt{29010} No applicable rule for ?0
\end{verbatim}
8.3. 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:

- \(\neg\_rewrite\_canon\) \((\Gamma \vdash \neg(t1 \lor t2)) = (\Gamma \vdash \neg t1 \land \neg t2)\)
- \(\neg\_rewrite\_canon\) \((\Gamma \vdash \neg\exists vs \bullet t) = (\Gamma \vdash \forall vs \bullet \neg t)\)
- \(\neg\_rewrite\_canon\) \((\Gamma \vdash \neg t) = (\Gamma \vdash t)\)
- \(\forall\_rewrite\_canon\) \((\Gamma \vdash \forall vs \bullet t) = (\Gamma \vdash t \leftrightarrow F)\)

**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.

- \(\neg\_T2\) \(\{ \Gamma \} \neg t2 \)\( ttac1\) \( ttac2\)\( \Gamma \vdash t1 \neg\)

**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 \(\neg t\)
SML
\begin{verbatim}
val \_T : TERM \rightarrow (THM \rightarrow TACTIC) \rightarrow TACTIC;
\end{verbatim}

**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.

\[
\begin{array}{c}
\text{Tactic} \\
\{ \Gamma \} \neg t2 \\
\text{ttac} (t2 \vdash t2) \{ \Gamma \} \neg t1 \\
\text{ttac} (t2 \vdash t2) \{ \Gamma \} \neg t1 \\
\end{array}
\]

\[
\begin{array}{c}
\text{Uses} \quad \text{To prove a negated term by showing that assuming the term gives rise to a contradiction.} \\
\text{See Also} \quad \text{strip_tac, contr_tac, \_T} \\
\text{Errors} \\
28022 \quad \text{?0 is not boolean} \\
28023 \quad \text{Goal is not of the form } \neg t \} \\
\end{array}
\]
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. \(\neg t\_thm\) 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.

\[
\begin{align*}
\neg\neg\_thm & \vdash \forall a \neg \neg a \iff a \\
\neg\lor\_thm & \vdash \forall a b \neg (a \lor b) \iff (\neg a \land \neg b) \\
\neg\land\_thm & \vdash (\neg (a \land b)) \iff (\neg a \lor \neg b) \\
\neg\Rightarrow\_thm & \vdash \forall a b \neg (a \Rightarrow b) \iff (a \land \neg b) \\
\neg\Leftarrow\_thm & \vdash \forall a b \neg (a \Leftarrow b) \iff a \land \neg b \lor b \land \neg a \\
\neg\if\_thm & \vdash \forall a b \neg (if a then T else T) \iff (if a then \neg T else \neg T) \\
\neg\f\_thm & \vdash \neg F \iff T \\
\Rightarrow\_thm & \vdash \forall a b \neg (a \Rightarrow b) \iff (\neg a \lor b) \\
\Leftarrow\_thm & \vdash \forall a b \neg (a \Leftarrow b) \iff (a \Rightarrow b) \land (b \Rightarrow a) \\
\if\_thm & \vdash \forall a b \neg (if a then b else c) \iff (a \land b) \lor (\neg a \land c)
\end{align*}
\]

See Also \(\neg t\_thm\).
SML
\[
\text{val } \Rightarrow \text{ THEN } : (\text{THM } \rightarrow \text{TACTIC}) \rightarrow (\text{THM } \rightarrow \text{TACTIC});
\]

Description A theorem tactical to apply a given theorem tactic to the result of eliminating \( \Rightarrow \) from a theorem of the form \( \Gamma \vdash t1 \Rightarrow t2 \).

\[
\Rightarrow _\text{THEN} \text{ thmtac} (\Gamma \vdash t1 \Rightarrow t2) = \text{ thmtac} (\Gamma \vdash \neg t1 \lor t2)
\]
The function is partially evaluated with only the theorem tactic and theorem arguments.

Errors
28054 \( ?0 \) is not of the form: \( \Gamma \vdash t1 \Rightarrow t2 \)

SML
\[
\text{val } \Rightarrow _\text{thm_tac} : \text{THM } \rightarrow \text{TACTIC};
\]

Description A tactic which uses a theorem whose conclusion is an implication, \( t1 \Rightarrow t2 \), to reduce a goal with conclusion \( t2 \) to \( t1 \).

\[
\text{Tactic} \quad \begin{array}{c}
\{ \Gamma \} t2 \\
\{ \Gamma \} t1 \\
\end{array} \quad \Rightarrow _\text{thm_tac} \quad \begin{array}{c}
\Gamma 1 \vdash t1 \Rightarrow t2 \\
\end{array}
\]
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 \( \text{bc}_\text{thm_tac} \) and \( \text{bc}_\text{tac} \) is not required.

See Also \( \text{bc}_\text{thm_tac}, \text{bc}_\text{tac} \).

Errors
29013 Conclusion of the goal is not \( ?0 \)

SML
\[
\text{val } \Rightarrow _\text{T} : (\text{THM } \rightarrow \text{TACTIC}) \rightarrow \text{TACTIC};
\]

Description Reduce an implicative goal by passing the antecedent to a tactic generating function.

\[
\text{Tactic} \quad \begin{array}{c}
\{ \Gamma \} t1 \Rightarrow t2 \\
\text{ttac}\{ t1, \Gamma \} t2 \\
\end{array} \quad \Rightarrow _\text{T} \quad \text{ttac}
\]

Errors
28051 Goal is not of form: \( \{ \Gamma \} t1 \Rightarrow t2 \)

SML
\[
\text{val } \forall _\text{tac} : \text{TACTIC};
\]

Description Reduce a universally quantified goal.

\[
\text{Tactic} \quad \begin{array}{c}
\{ \Gamma \} \forall vs[x1,\ldots] \cdot t[x1,\ldots] \\
\{ \Gamma \} t[x1',\ldots] \\
\end{array} \quad \forall _\text{tac}
\]
where \( x1' \) is a variant name of \( x1 \), etc, different from any variable in \( \Gamma \) or \( t \).

See Also \( \text{simple}_\forall _\text{tac} \)

Errors
29001 Goal is not of the form: \( \{ \Gamma \} \forall vs \cdot t[vs] \)
8.3. General Tactics and Tacticals

\begin{verbatim}
SML
val \exists_tac : TERM \rightarrow 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.

Tactic   \[
\begin{array}{rcl}
\{ \Gamma \} \exists \ vs[x1,...] \bullet t2[x1,...] & \rightarrow & \exists_tac \ t' \\
\{ \Gamma \} \ t2[x1',...] \\
\end{array}
\]

where \( vs[t_1,...] \) is \( t \), type instantiated to have the same type as \( vs[x1,...] \), and broken up using
\( Fst \) and \( Snd \) as necessary.

See Also   \texttt{simple\_\exists_tac}

\end{verbatim}

Errors

\begin{verbatim}
29002  Goal is not of the form: \{ \Gamma \} \exists vs \bullet t2[vs]
29008  Cannot match witness ?0 to varstruct ?1
\end{verbatim}

\begin{verbatim}
SML
val \exists\_THEN : (THM \rightarrow TACTIC) \rightarrow (THM \rightarrow 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 \( \Gamma \vdash \exists vs \bullet t \).

\exists\_THEN thmtac (\( \Gamma \vdash \exists vs[x1,...] \bullet t \)) = thmtac (\( \Gamma \vdash t[x1'/x1,...] \))

where \( x1' \) is a variant of \( x1' \), etc, which does not appear in \( \Gamma \) or in the assumption or
conclusion of the goal.

See Also   \texttt{SIMPLE\_\exists\_THEN}

\end{verbatim}

Errors

\begin{verbatim}
29003  ?0 is not of the form: \' \Gamma \vdash \exists vs \bullet t'
\end{verbatim}

\begin{verbatim}
SML
val \exists_1_tac : TERM \rightarrow TACTIC;

Description   Provide a witness for a goal with conclusion of the form \( \exists_1 x \bullet t \).

N.B. this tactic strengthens the goal, i.e. it may result in unprovable subgoals even when the
original goal was provable.

Tactic   \[
\begin{array}{rcl}
\{ \Gamma \} \exists_1 vs[x1,...] \bullet P[x1,...] & \rightarrow & \exists_1_tac1 \ t' \\
\{ \Gamma \} \ P[x1',...] \\
\{ \Gamma \} \ \forall vs[x1',...] \bullet \\
\{ \Gamma \} \ \forall vs[x1',...] \\
\end{array}
\]

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 \( vs[x1,...] \), and \( vs[t1',...] \) equals \( t' \) (perhaps using \( Fst \) and \( Snd \)).

Errors

\begin{verbatim}
29004  Goal is not of the form: \{ \Gamma \} \exists_1 vs \bullet t
29008  Cannot match witness ?0 to varstruct ?1
\end{verbatim}

\end{verbatim}
\[ \exists_1 \text{THEN} : (\text{TACTIC} \rightarrow \text{TACTIC}) \rightarrow (\text{TACTIC} \rightarrow \text{TACTIC}); \]

**Description**  A theorem tactical which applies a given theorem tactic to the result of eliminating the outermost quantifier from a theorem of the form \( \Gamma \vdash \exists_1 \forall vs \cdot t \).

\[ \exists_1 \text{THEN} \quad \text{thmtac} (\Gamma \vdash \exists_1 vs \cdot t) = \]
\[ \text{thmtac} (\Gamma \vdash t[x_1'/x_1, ...] \land \forall vs[x_1'',...] \cdot P[x_1'',...] \Rightarrow vs[x_1'',...]) = vs[x_1'',...] \]

where \( x_1'' \) and \( x_1''' \) are distinct variants of \( x_1 \), 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: \( \Gamma \vdash \exists_1 vs \cdot t' \)

\[ \text{val} \quad \epsilon_\ast \text{tac} : \text{TERM} \rightarrow \text{TACTIC}; \]
\[ \text{val} \quad \epsilon_\ast T : \text{TERM} \rightarrow (\text{THM} \rightarrow \text{TACTIC}) \rightarrow \text{TACTIC}; \]

**Description**  Given a choice term, \( \epsilon x \cdot t \) say, \( \epsilon_\ast \text{tac} \) sets \( \exists x \cdot t \) as a lemma, and derives the new assumption \( t[\epsilon x \cdot t/x] \) from it.

\( \epsilon_\ast T \) is the same as \( \epsilon_\ast \text{tac} \) except that it passes the new assumption to a tactic generating function.

\[ \begin{array}{c}
\{ \Gamma \} \quad t1 \\
\{ \Gamma \} \quad \exists x \cdot t; \text{strip } t[\epsilon x \cdot t/x], \Gamma \} \quad t1 \\
\epsilon_\ast \text{tac} \\
\end{array} \]

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\( \epsilon_\ast \text{tac} \), all\( \epsilon_\ast T \) (which are easier to use in most cases).

**Errors** 29050 ?0 is not of the form \( \epsilon x \cdot p x \)
### 8.4 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 ⇔ 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:

- `Fst(a,b) = x`  →  `a = x`
- `Snd(a,b) = y`  →  `b = y`
- `x = Fst(a,b)`  →  `x = a`
- `y = Snd(a,b)`  →  `y = b`
- `(a,b) = (c,d)`  →  `a = c ∧ b = d`
- `(a,b) = z`  →  `a = Fst z ∧ b = Snd z`
- `z = (a,b)`  →  `Fst z = a ∧ Snd z = b`
- `z = w`  →  `Fst z = Fst w ∧ Snd z = Snd w`

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 ⇔ 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.
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:

\[
\begin{align*}
\text{Fst}(a,b) = x & \rightarrow a = x \\
\text{Snd}(a,b) = y & \rightarrow b = y \\
x = \text{Fst}(a,b) & \rightarrow x = a \\
y = \text{Snd}(a,b) & \rightarrow y = b \\
(a,b) = (c,d) & \rightarrow a = c \land b = d \\
(a,b) = z & \rightarrow a = \text{Fst} z \land b = \text{Snd} z \\
z = (a,b) & \rightarrow \text{Fst} z = a \land \text{Snd} z = b \\
z = w & \rightarrow \text{Fst} z = \text{Fst} w \land \text{Snd} z = \text{Snd} w
\end{align*}
\]

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 \( \vdash a_i = b_i \) 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:

\[
\begin{align*}
\vdash a1 = b1 & \quad \vdash z1 = a1 & \quad \vdash w1 = a1 & \quad \vdash d1 = a1 \\
\vdash y1 = a1 & \quad \vdash b1 = a1 & \quad \vdash c1 = a1 \\
\vdash x2 = a2 & \quad \vdash z2 = a2 & \quad \vdash w2 = a2 & \quad \vdash d2 = a2 \\
\vdash y2 = a2 & \quad \vdash b2 = a2 & \quad \vdash c2 = a2
\end{align*}
\]

plus the non equational theorems the second element of the result tuple.
8.5 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 \( \vdash t_1 = t_2 \) where \( t_1 \) is a term formed from atoms of type \( \mathbb{N} \) and \( t_2 \) is in what we may call additive normal form, i.e. it has the form: \( t_1 + t_2 + \ldots \), where the \( t_i \) have the form \( s_1 * s_2 * \ldots \) where the \( s_i \) are atoms. Here, by atom we mean a term which is not of the form \( t_1 + t_2 + \ldots \) or \( s_1 * s_2 * \ldots \).

The summands \( t_i \) and, within them, the factors \( s_j \) are given in increasing order with respect to the ordering on terms given by the function \texttt{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.

\texttt{ANF\_C conv} is a conversion which acts like \texttt{anf\_conv} but which applies \texttt{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 \( \texttt{⌜:\mathbb{N}⌝} \) or is already in additive normal form

SML

val ASYM_C : CONV -> CONV
val GEN_ASYM_C : TERM ORDER -> CONV -> CONV

Description These conversionals 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 \( t_1 = t_2 \) in which \( t_2 \) strictly precedes \( t_1 \) in the ordering.

\texttt{ASYM\_C c} is a conversion which behaves like \( c \) on terms \( t_1 \) for which \( c \ t_1 \) is a theorem with conclusion \( t_1 = t_2 \) where \( t_2 \) (strictly) precedes \( t_1 \) in the standard ordering on terms given by \texttt{term\_order} q.v. and fails on other terms.

\texttt{GEN\_ASYM\_C} is like \texttt{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 \texttt{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 $\vdash t1 \leftrightarrow t2$ where $t2$ 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.

| $a \land T$  | $a$  |
| $T \land a$ | $a$  |
| $F \land a$ | $F$  |
| $a \land F$ | $F$  |
| $a \land a$ | $a$  |
| $a \land \neg a$ | $F$ |
| $a \lor T$ | $T$  |
| $T \lor a$ | $T$  |
| $F \lor a$ | $a$  |
| $a \lor F$ | $a$  |
| $a \lor a$ | $a$  |
| $a \lor \neg a$ | $T$ |
| $\neg T$ | $F$  |
| $\neg F$ | $T$  |

Note, however, that more global simplifications are not done, e.g. there is no attempt to eliminate a 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.
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.

\[
\begin{align*}
    a \land T & \rightarrow a \\
    T \land a & \rightarrow a \\
    F \land a & \rightarrow F \\
    a \land F & \rightarrow F \\
    a \land a & \rightarrow a \\
    a \land \neg a & \rightarrow F \\
    a \lor T & \rightarrow T \\
    T \lor a & \rightarrow T \\
    F \lor a & \rightarrow a \\
    a \lor F & \rightarrow a \\
    a \lor a & \rightarrow a \\
    a \lor \neg a & \rightarrow T \\
    \neg T & \rightarrow F \\
    \neg F & \rightarrow T
\end{align*}
\]

Note, however, that more global simplifications are not done, e.g. there is no attempt to eliminate a 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 "BOOL" or is already in disjunctive normal form
```
### gen_term_order

**Definition**

\[
\text{gen\_term\_order} : (\text{TERM} \rightarrow (\text{TERM} \times \text{INTEGER})) \rightarrow \text{TERM} \rightarrow \text{TERM} \rightarrow \text{int};
\]

**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 \(X_1, X_2, \ldots\), we have that `special tm = (tm, 0)` if \(tm \not\in X_i\) for any \(i\) and that `special tm = (x_i, f_i(tm))` if \(tm \in X_i\), 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.

### make_term_order

**Definition**

\[
\text{make\_term\_order} : \text{(TERM ORDER} \rightarrow \text{TERM ORDER)} \text{ list} \rightarrow \text{TERM ORDER};
\]

**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 were 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 which makes the immediate successor of any term of type `BOOL` its immediate successor:

\[
\text{fun } f \ t = (\text{dest} \neg t, 1) \text{ handle Fail } \Rightarrow (t, 0); \]

\[
\text{val } \neg\text{order} = \text{make\_term\_order} [\text{fn } r => \text{induced\_order}(f, \text{pair\_order } r \text{ int\_order})];
\]
SML

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 \( op_+ \), distributes over the other, say \( op_\ast \). 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 \( t_1 \ op_\ast \ t_2 \) or \( t_1 \ op_\ast \ t_2 \). The theorems computed by the conversion have the form \( t = t_1 \ op_\ast \ t_2 \ op_\ast \ t_3 \), 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 \( s_1 \ op_\ast \ s_2 \ op_\ast \ ... \), 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 \( t_1 \ op_\ast \ ... \) or \( t_1 \ op_\ast \ ... \) as it is created. In more detail the parameters are, in order, as follows:

1. A term ordering, such as \( \text{term\_order} \), q.v.
2. A theorem of the form \( \vdash \forall x \ y \ z \bullet (x \ op_\ast \ y) \ op_\ast \ z = x \ op_\ast \ y \ op_\ast \ z \).
3. A theorem of the form \( \vdash \forall x \ y \ z \bullet (x \ op_\ast \ y) \ op_\ast \ z = x \ op_\ast \ y \ op_\ast \ z \).
4. A theorem of the form \( \vdash \forall x \ y \ z \bullet (x \ op_\ast \ y) \ op_\ast \ z = x \ op_\ast \ y \ op_\ast \ z \).
5. A theorem of the form \( \vdash \forall x \ y \ z \bullet (x \ op_\ast \ y) \ op_\ast \ z = x \ op_\ast \ y \ op_\ast \ z \).
6. A theorem of the form \( \vdash \forall x \ y \ z \bullet (x \ op_\ast \ y) \ op_\ast \ z = x \ op_\ast \ y \ op_\ast \ z \).
7. A conversion to be applied to any subterm of the form \( t_1 \ op_\ast \ ... \) 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 \( t_1 \ op_\ast \ ... \) 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 \( \vdash t_1 \ op_1 \ t_2 \ op_2 \ t_3 = (t_1 \ op_1 \ t_2) \ op_2 \ (t_1 \ op_1 \ t_3) \)
81024 ?0 and ?1 do not have the forms \( \vdash t_1 \ op_1 \ t_2 = t_2 \ op_1 \ t_1 \)
and \( \vdash t_1 \ op_1 \ t_2 \ op_2 \ t_3 = (t_1 \ op_1 \ t_2) \ op_2 \ (t_1 \ op_1 \ t_3) \)
81025 ?0 is already sorted

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
val sort_conv : TERM ORDER ->
               THM -> THM -> CONV -> CONV -> CONV;

Description  This conversion normalises a term constructed from atoms using an associative and
commutative binary operator, \textit{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 \( t_1 \text{ op } t_2 \). The theorems computed by the conversion have the form \( t = t_1 \text{ op } t_2 \text{ op } ... \),
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 \textit{op} is; n.b. \textit{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 \( t = t_1 \text{ op } ... \) as it is created. In more detail the
parameters are, in order, as follows:

1. A term ordering, such as \textit{term_order}, q.v.
2. A theorem of the form \( \forall x \; y \; t \ x \ y = t \ y \ x \).
3. A theorem of the form \( \forall x \; y \; z \; (x \text{ op } y) \text{ op } z = x \text{ op } y \text{ op } z \).
4. A conversion to be applied to each subterm of the form: \( t \text{ op } ... \) 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
\begin{itemize}
  \item \texttt{81021} \texttt{?0 does not have the form } \texttt{\vdash t1 op t2 = t2 op t1}
  \item \texttt{81022} \texttt{?0 does not have the form } \texttt{\vdash (t1 op t2) op t3 = t1 op (t2 op t3)}
  \item \texttt{81025} \texttt{?0 is already sorted}
  \item \texttt{81029} \texttt{Internal error: unexpected error in term normalisation package}
\end{itemize}

val term_order : TERM -> TERM -> int;

Description  \textit{term_order} gives an ordering relation on HOL terms. The ordering relation follows
the same conventions as those used by the sorting function \textit{sort}, namely, \textit{term_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, 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, \textit{type_order}
or \textit{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 \( i \cdot x \) where \( i \) is a numeric literal are ordered so that the terms \( x, \ 0 \cdot x, \ 1 \cdot x, \ 2 \cdot x, \ ... \)
are consecutive.

See Also  \textit{gen_term_orderI} 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).
8.6 First Order Resolution

**SML**

```
signature Resolution = sig

Description This is the signature of a structure providing Resolution facilities to ICL HOL.
```

**SML**

```
(* 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 intended for use by others. May be withdrawn.
```
8.6. First Order Resolution

SML

```ml
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`).

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUAL USR029
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 |
8.6. 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, ?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.

\[
\begin{align*}
\Gamma \vdash P \ [\text{subterm}] \\
\Delta \vdash N \ [\text{subterm}]
\end{align*}
\]

simplify \( (\Gamma, \Delta \vdash P[F] \lor N[T]) \)  basic_resolve_rule
subterm

Where simplify carries out the simplifications in the predicate calculus where an argument is the constant \( \top \) or \( \bot \), plus a few others.

Errors

3031  ?0 is not of type \( \top:BOOL \top 
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.SUBS -> 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.
8.6. First Order Resolution

**SML.val** basic_res_rule : int -> THM list -> THM list ->

   THM list;

**Description**  
basic_res_rule 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 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 \( \lnot F \) 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

**SML.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 (segasms, 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 (segasms, 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
8.6. First Order Resolution

SML
\[
val \text{basic\_res\_tac}3 : \text{int} \rightarrow \text{THM list} \rightarrow \text{TACTIC};
\]

**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
\[
val \text{basic\_res\_tac}4 : \text{int} \rightarrow \text{int list} \rightarrow \text{int list} \rightarrow \text{THM list} \rightarrow \text{THM list} \rightarrow \text{TACTIC};
\]

**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 with 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
val basic_res_tac : int -> THM list -> TACTIC;

**Description**  
`basic_res_tac` takes a limit and a list of theorems and performs basic resolution. It first strips the negated goal into the assumption list. This uses the `strip_tac` function, 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 theorems 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 where necessary and possible.

The resulting list of theorems will have all the theorems 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, `\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
8.6. First Order Resolution 333

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) -> (* next item to process *)
('a list * '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.SUBS -> (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.
8.7 Quantifier Elimination Toolkit

SML

val dest_quant : TERM -> TERM * TERM * TERM;
val is_quant : TERM -> bool;
val mk_quant : TERM * TERM * TERM -> TERM;

Description These functions are the destructor, discriminator and constructor functions for quantified terms, i.e., boolean terms formed by applying a constant to a (possibly paired) \( \lambda \)-abstraction.

Errors

119001?0 is not a quantified term
119003?0 is not a valid varstruct
119004?0 is not of type "BOOL"

SML

val FAIL_C : CONV -> CONV;

Description Conversional that always fails: \( ID_C \) conv is the same as \( fail \_conv \). This is for use with other conversionals which take a conversional as an input.

SML

val find_in_\&conv : (TERM -> bool) -> CONV;

Description Given a function specifying a property of terms and an iterated conjunction, \( a \land b \land ... \), this conversion rearranges it in the form: \( c \land a \land b \land ... \) where \( c \) is the first conjunct satisfying the property. It fails if there is nothing to do or if no conjunct satisfies the property.

Errors

119008?0 cannot be rearranged or is already in the required form

SML

val FIRST\_THEN\_C : (CONV * (CONV -> CONV)) list -> CONV;

Description \( FIRST\_THEN\_C \) takes a list of conversion-conversional pairs and gives which acts as follows: the conversions in the list are tried one after the other, and the process only fails if none of the conversions is applicable; when a conversion succeeds, \( FIRST\_THEN\_C \) is applied recursively to the remainder of the list as if by \( X\_C FIRST\_THEN\_C \), where \( X\_C \) is the conversional associated with the successful conversion. The conversionals are intended to allow the process to be diverted into subterms of the result of a transformation. For example, one might transform a unique existential quantification into an existential whose matrix contains a universal and then divert attention to the universal.

Errors

119010 None of the conversions was applicable to ?0

SML

val ID_C : CONV -> CONV;

Description The identity conversional: \( ID\_C \) conv is the same as \( conv \). This is for use with other conversionals such as \( FIRST\_THEN\_C \), q.v., which take a conversional as an input.

SML

val INNERMOST_QUANT\_C : CONV -> CONV;

Description \( INNERMOST\_QUANT\_C \) is a conversional that maps a conversion over the innermost quantified subterms of the first-order structure of a term. It fails if the first-order structure of the term is quantifier-free.

Errors

119011 No quantified subterms were found
val nnf_conv : CONV;

Description  This conversion puts a propositional formula into negation normal form, i.e., a normal form in which $\neg$ is pushed through the other propositional connectives using the following rules:

\[\neg(p_1 \land p_2) \iff \neg p_1 \lor \neg p_2 \quad \neg(p_1 \lor p_2) \iff \neg p_1 \land \neg p_2\]
\[\neg(p_1 \Rightarrow p_2) \iff p_1 \land \neg p_2 \quad \neg(p_1 \iff p_2) \iff p_1 \land \neg p_2 \lor \neg p_1 \land p_2\]
\[\neg(\neg p) \iff p\]

Propositional simplification using prop_simp_conv is attempted before and after the above rules.

Errors
119007?0 is not of type $\langle$BOOL$\rangle$ or is already in negation normal form

val prenex_clauses : THM;

Description  Higher-order rewriting with this theorem carries out conversion to prenex normal form.

val PROP_ATOM_C : CONV $\rightarrow$ CONV;
val PROP_LIT_C : CONV $\rightarrow$ CONV;

Description  PROP_ATOM_C is a conversional that maps its argument over the outermost propositional atoms in a term. PROP_LIT_C maps its argument over the outermost propositional literals in a term. Here a propositional atom is any term other than one formed using one of the propositional connectives $\land$, $\lor$, $\Rightarrow$, $\iff$ or $\neg$. A propositional literal is either a propositional atom or the negation of a propositional atom.

val prop_simp_conv : CONV;

Description  This conversion simplifies propositional formula using the following rules:

\[\neg(\neg p) \iff p \quad \neg F \iff T \quad \neg T \iff F\]
\[F \lor p \iff p \quad p \lor F \iff p \quad T \lor p \iff T \quad p \lor T \iff T\]
\[p \land F \iff F \quad F \land p \iff F \quad T \land p \iff p \quad p \land T \iff p\]
\[p \Rightarrow T \iff T \quad T \Rightarrow p \iff p \quad F \Rightarrow p \iff T \quad p \Rightarrow F \iff \neg p\]
\[(p \iff T) \iff p \quad (T \iff p) \iff p \quad (F \iff p) \iff \neg p \quad (p \iff F) \iff \neg p\]
\[(p \lor p) \iff p \quad p \land p \iff p \quad (p \lor p) \iff T \quad (p \land p) \iff T\]
\[(p \lor \neg p) \iff T \quad p \land \neg p \iff F \quad (p \Rightarrow \neg p) \iff \neg p \quad (p \leftrightarrow \neg p) \iff F\]
\[\neg(p \lor \neg p) \iff T \quad \neg p \land \neg p \iff F \quad (\neg p \Rightarrow p) \iff p \quad (\neg p \leftrightarrow p) \iff F\]

The conversion fails if no simplification is possible.

Errors
119012 No simplifications apply to ?0

val QUANTS_C : CONV $\rightarrow$ CONV;

Description  QUANTS_C is a conversional that maps its argument conversion over a term formed by repeatedly applying quantifiers to a non-quantified term. The conversion is applied to each quantified term obtained by stripping off a leading part of the prefix, the innermost being processed first.
8.7. Quantifier Elimination Toolkit

SML

val QUANT_C : CONV -> CONV;

Description QUANT_C is a conversional that applies its argument to the matrix of a quantified term.

SML

val QUANT_MAP_C : CONV -> CONV;

Description QUANT_MAP_C is a conversional that maps a conversion over all the quantifiers in a term, innermost quantifiers first. If the conversion succeeds it is retried, so it should fail if it cannot make any progress.

SML

val simple_one_point_conv : CONV;

Description This conversion proves all theorems of the forms ⊢ (∃x•x = t ∧ p) ↔ p[t/x] and ⊢ (∃x•x = t) ↔ T where p[t/x] denotes the result of substituting the term t for the variable x in p. In the second form, t may be the same as x.

Errors 119009?0 is not of the form ⊢ ∃x•x = t ∧ p or ∃x•x = t

SML

val simple_∀¬∃¬conv : CONV;

Description This conversion proves theorems of the form ⊢ (∀x•t) ↔ (¬(∃x•¬t))

Errors 119005?0 is not of the form ⊢ ∀x•t where x is a variable

SML

val simple_∃const_elim_conv : CONV;

Description This conversion proves all theorems of the form: ⊢ (∃x•p) ↔ p where the variable x does not appear free in p.

Errors 119006?0 is not of the form ⊢ ∃x•p where x is a variable that does not occur free in p

SML

val split_∧conv : (TERM -> bool) -> CONV;

Description Given a function specifying a property of terms and an iterated conjunction, a ∧ b ∧ ..., this conversion rearranges the conjunction up into the form: (d ∧ c ∧ ...) ∧ (e ∧ f ∧ ...), where c,d,... satisfy the property and e,f,... do not. It fails there is nothing to do, i.e., if all the conjuncts satisfy the property, if none of the conjuncts satisfy the property or if the first conjunct satisfies the property and the others do not.
Chapter 8. PROOF IN HOL

8.8 Proof Contexts

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:

<table>
<thead>
<tr>
<th>pc(s)</th>
<th>proof context(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rewriting</td>
</tr>
<tr>
<td>cs</td>
<td>constant specification</td>
</tr>
<tr>
<td>∃</td>
<td>existential theorem prover</td>
</tr>
<tr>
<td>pr</td>
<td>prove_tac and related tools</td>
</tr>
<tr>
<td>sg</td>
<td>goal stripping</td>
</tr>
<tr>
<td>st</td>
<td>theorem stripping</td>
</tr>
<tr>
<td>cd</td>
<td>clausal definition</td>
</tr>
<tr>
<td>vs</td>
<td>variable structure</td>
</tr>
<tr>
<td>ad</td>
<td>application data</td>
</tr>
<tr>
<td>net</td>
<td>discrimination net</td>
</tr>
<tr>
<td>eqn_cxt</td>
<td>equational context</td>
</tr>
<tr>
<td>nd</td>
<td>dictionary of discrimination nets (and sources)</td>
</tr>
<tr>
<td>canon(s)</td>
<td>theorem canonicalisation</td>
</tr>
<tr>
<td>mmp</td>
<td>matching mp rule</td>
</tr>
<tr>
<td>eqn</td>
<td>equation matcher</td>
</tr>
</tbody>
</table>

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.

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.).
8.8. Proof Contexts

**SML**

```sml
val asm_prove_tac : THM list -> 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 when available (i.e. the result is partially evaluated with only the list of theorems).

Tactic

```
\{ \Gamma \} t
```

```sml
asm_prove_tac thms
```

**See Also** `PC_T1` to defer accessing the proof context until application to the goal; `prove_tac` for the form that does not 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.

**SML**

```sml
val asm_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 using `conv_tac`.

Tactic

```
\{ \Gamma \} t
```

```sml
conv_tac (current_ad_cs_∃_conv ()) thms
```

**See Also** `prove_∃_tac` that does not 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.

**SML**

```sml
val commit_pc : string -> unit;
```

**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 `new_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

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

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

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

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
8.8. Proof Contexts

SML

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 Initial 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

SML

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

SML

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
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.

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.
8.8. Proof Contexts

```sml
val extend_pc_rule1 : string -> ('a -> 'b -> THM) -> 'a -> 'b -> THM;
val extend_pcs_rule1 : string list -> ('a -> 'b -> THM) -> 'a -> 'b -> 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.

```sml
val extend_pc_rule : string -> ('a -> THM) -> ('a -> THM);
val extend_pcs_rule : string list -> ('a -> THM) -> ('a -> 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.
EXTEND_PC_T1, EXTEND_PCS_T1

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.

EXTEND_PC_T, EXTEND_PCS_T

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 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.
8.8. Proof Contexts

SML

```sml
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.

SML

```sml
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`

SML

```sml
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`.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUAL USR029
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

```
| (["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`.

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
SML

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

SML

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
val PC_C1 : string -> ('a -> CONV) -> 'a -> CONV;
val MERGE_PCS_C1 : string list -> ('a -> CONV) -> 'a -> CONV;

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.

val PC_C : string -> CONV -> CONV;
val MERGE_PCS_C : string list -> CONV -> CONV;

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.

val pc_rule1 : string -> ('a -> 'b -> THM) -> 'a -> 'b -> THM;
val merge_pcs_rule1 : string list -> ('a -> 'b -> THM) -> 'a -> 'b -> THM;

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.
### 8.8. Proof Contexts

SML

```sml
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.

---

SML

```sml
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.

---

SML

```sml
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.
SML

```ml
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

```ml
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
8.8. Proof Contexts

```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.

```
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`.

```
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`

```
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

```
val pp/set_eval_ad_rw_net : (EQN_CTX -> 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
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

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

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.
val pp\textquoteleft set\_eval\_ad\_nd\_net : 
\((\text{string } \rightarrow (\text{TERM } \leftrightarrow (\text{TERM } \rightarrow \text{THM}))) \text{ list } \rightarrow \text{unit}\); 
val current\_ad\_nd\_net : \text{string } \rightarrow (\text{TERM } \rightarrow \text{THM}) \text{ NET};

\textbf{Description}  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 \textit{pp\textquoteleft 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 \textit{make\_net} on each list of sources.

\textit{current\_ad\_nd\_net} \text{ key} returns the net indexed by the key \text{key} in the current proof context. If no entry exists it returns the empty net \textit{empty\_net}. Note that the returned net can be viewed as something of type \textit{EQN\_CXT}, and made into a conversion by \textit{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}  \textit{set\_nd\_entry} for the associated user data.

val pp\textquoteleft set\_eval\_ad\_cs\_∃\_convs : (\text{CONV} \text{ list } \rightarrow \text{CONV}) 
\rightarrow \text{unit}; 
val current\_ad\_cs\_∃\_conv : \text{unit } \rightarrow \text{CONV};

\textbf{Description}  These functions provide the interface to the existence prover for constant specifications (see \textit{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}  \textit{set\_cs\_∃\_rule} for the associated user data.

\textbf{Errors} \begin{enumerate}
\item 51015  No automated existence prover in the current proof context succeeds
\item 51021  The current proof context was created in theory ?0 at a point now either not in scope, deleted or modified
\end{enumerate}
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
prove_conv

\[ \text{current_ad_pr_conv} (\text{thms} \mapsto t) \mapsto \text{thms} \mapsto t \]

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 ‘\( \vdash \text{term} \equiv T \)’ (with no assumptions) where term is \( \alpha \)-convertible to term’,
then apply \( \equiv \text{t_elim} \), and otherwise fail.

Rule
prove_rule

\[ \vdash \text{tm} \mapsto \text{thms} \mapsto \text{tm} \]

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: ‘\( \vdash \text{input} \equiv T \)’

and as the proof context setting.
8.8. Proof Contexts

**SML**

```sml
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.

**Conversion**

```sml
current_ad_cs_∃_conv () ⊢ ™
prove_∃_conv
⌜ ™ ⌜
```

**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**

```sml
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 `⊢ ™ ⇔ T` (with no assumptions) where `term` is α-convertible to `term'`, then apply `⇔ t_elim`, and otherwise fail.

**Rule**

```sml
prove_∃_rule
⌜ ™ ⌜
⊢ ™
```

**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.
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 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

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_extend_pc : string -> unit;
val set_extend_pc : string -> unit;

Description  push_extend_pc stacks the previous “current” proof context, 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

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
### 8.8. Proof Contexts

SML

```sml
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

SML

```sml
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
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

\[
conv4 \text{ AND } OR_C conv3 \text{ AND } OR_C conv2 \text{ AND } OR_C conv1
\]

“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

val set_mmp_rule : (THM -> THM -> THM) -> string -> unit;
val get_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
8.8. Proof Contexts

SML

```
val set_nd_entry : string -> (TERM * (TERM -> THM))list -> string -> unit;
val get_nd_entry : string -> string ->
    ((TERM * (TERM -> 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.

`set_nd_entry dict_key entry pc_name` overwrites (or creates, if necessary) the proof context’s name’s subfield dictionary entry whose key is `dict_key` in the proof context `pc_name` with the value `entry`.

`get_nd_entry dict_key pc_name` returns the dictionary entries whose keys are `dict_key` from each of the subfields in the proof context `pc_name`, paired with the source proof context name, or an empty list if the entry is not present in the dictionaries of any of the subfields of that proof context.

**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_pr_conv : (THM list -> CONV) -> string -> unit;
val get_pr_conv : string -> (THM list -> CONV);
val get_pr_conv1 : string -> (THM list -> 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
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
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_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
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
### 8.8. Proof Contexts

#### SML

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>val set_rw_eqn_rule : (THM -&gt; TERM * CONV) -&gt; string -&gt; unit;</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>val get_rw_eqn_rule : string -&gt; (THM -&gt; TERM * CONV) OPT;</code></td>
<td></td>
</tr>
<tr>
<td><code>val set_RW_eqm_rule : (THM -&gt; TERM * CONV) -&gt; string -&gt; unit;</code></td>
<td></td>
</tr>
<tr>
<td><code>val get_RW_eqm_rule : string -&gt; (THM -&gt; TERM * CONV) OPT;</code></td>
<td></td>
</tr>
</tbody>
</table>

**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>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>val set_rw_eqn_ctxt : EQN_CXT -&gt; string -&gt; unit;</code></td>
<td>These functions provide the interface to the equational context for rewriting (see e.g. <code>rewrite_tac</code>) 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 <code>thm_eqn_ctxt</code> and then adds them into the subfield whose key is the proof context’s name.</td>
</tr>
<tr>
<td><code>val get_rw_eqn_ctxt : string -&gt; (EQN_CXT * string) list;</code></td>
<td></td>
</tr>
<tr>
<td><code>val add_rw_thms : THM list -&gt; string -&gt; unit;</code></td>
<td></td>
</tr>
</tbody>
</table>

**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>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>val set_sc_eqn_ctxt : EQN_CXT -&gt; string -&gt; unit;</code></td>
<td>These functions provide the interface to the equational context for stripping goal conclusions (see e.g. <code>strip_tac</code>) 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 <code>thm_eqn_ctxt</code> and then adds them into the subfield whose key is the proof context’s name.</td>
</tr>
<tr>
<td><code>val get_sc_eqn_ctxt : string -&gt; (EQN_CXT * string) list;</code></td>
<td></td>
</tr>
<tr>
<td><code>val add_sc_thms : THM list -&gt; string -&gt; unit;</code></td>
<td></td>
</tr>
</tbody>
</table>

**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

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
val set_st_eqn_cxt : EQN_CXT -> string -> unit;
val get_st_eqn_cxt : string -> (EQN_CXT * string)list;
val add_st_thms : THM list -> string -> unit;

**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_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∃_cd_thms : THM list -> string -> unit;
val get∃_cd_thms : string -> THM list;
val add∃_cd_thms : THM list -> string -> unit;

**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
8.8. Proof Contexts

SML

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 \texttt{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 extract 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:

\[
\begin{align*}
\text{("", ",",} \\
\left[\text{"Fst"}, \text{"Snd"}\right], \text{\vdash } x, y, p \cdot \text{Fst} (x, y) = x \land \text{Snd} (x, y) = y \land (\text{Fst} p, \text{Snd} p) = p)
\end{align*}
\]

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

SML

val simple_ho_thm_eqn_ctx : 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 \texttt{(EQN_CXT q.v.)}

\[
\begin{align*}
\text{thm_eqn_ctx} \ ' \ \Gamma \vdash \forall \ x1 \ldots \bullet \ LHS = RHS' \to (LHS', \text{simple_eq_match_conv1} \ ' \ \Gamma \vdash \forall \ x1 \ldots \bullet \ LHS = RHS')
\end{align*}
\]

Here the pattern term \( LHS' \) is derived from \( LHS \) by replacing linear patterns (see \texttt{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 \texttt{simple_ho_match}.

See Also \texttt{cthm_eqn_ctx} which canonicalises the theorem before transformation.

Errors

7095 ?0 is not of the form \( \Gamma \vdash \forall \ x1 \ldots x_n \bullet \ u = v' \) where \( \forall xi \) 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 (EQN_CTX q.v.)

\[ \Gamma \vdash \forall x_1 \ldots \bullet \text{LHS} = \text{RHS} \rightarrow (\text{LHS}, \text{simple_eq-match_conv1} \ ' \Gamma \vdash \forall x_1 \ldots \bullet \text{LHS} = \text{RHS}) \]

The universal quantifiers must be over simple variables (not patterns).

See Also  ctthm_eqn_cxt which canonicalises the theorem before transformation.

Errors

7095  ?0 is not of the form 'Γ \vdash \forall x_1 \ldots xn \bullet u = v' where 'xi' are variables

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>'simple_abstractions'</td>
<td>predicates</td>
</tr>
<tr>
<td>'paired_abstractions'</td>
<td>predicates1</td>
</tr>
<tr>
<td>'propositions'</td>
<td>basic_hol</td>
</tr>
<tr>
<td>'fun_ext'</td>
<td>basic_hol1</td>
</tr>
<tr>
<td>'pair'</td>
<td>sets_ext</td>
</tr>
<tr>
<td>'pair1'</td>
<td>hol</td>
</tr>
<tr>
<td>'N'</td>
<td>hol1</td>
</tr>
<tr>
<td>'N_lit'</td>
<td></td>
</tr>
<tr>
<td>'list'</td>
<td></td>
</tr>
<tr>
<td>'char'</td>
<td></td>
</tr>
<tr>
<td>'sum'</td>
<td></td>
</tr>
<tr>
<td>'one'</td>
<td></td>
</tr>
<tr>
<td>'combin'</td>
<td></td>
</tr>
<tr>
<td>'sets_alg'</td>
<td></td>
</tr>
<tr>
<td>'sets_ext'</td>
<td></td>
</tr>
<tr>
<td>'basic_prove_∃_conv'</td>
<td></td>
</tr>
</tbody>
</table>

(* 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.
8.8. Proof Contexts

SML

(* Proof Context: 'simple_abstractions *)

Description A component proof context for handling only simple abstractions in stripping and canonicalisation.

Contents Rewriting:
Stripping theorems:

|simple_¬_in_conv

Stripping conclusions:

|simple_¬_in_conv

Rewriting canonicalisation:

|simple_∀_rewrite_canon, simple_¬_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.

SML

(* Proof Context: 'paired_abstractions *)

Description A component proof context for handling simple and paired abstractions in stripping and canonicalisation.

Contents Rewriting:

|β_conv

Stripping theorems:

|¬_in_conv, ∃_1_conv,

|∀_uncurry_conv, ∃_uncurry_conv

Stripping conclusions:

|¬_in_conv, ∀_uncurry_conv

Rewriting canonicalisation:

|∀_rewrite_canon, ¬_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.
Proof Context: 'propositions *

Description  A component proof context for reasoning about propositions.

Contents  Rewriting:

\[ \text{eq_rewrite_thm, } \iff \text{rewrite_thm, } \neg \text{rewrite_thm, } \land \text{rewrite_thm, } \lor \text{rewrite_thm, } \Rightarrow \text{rewrite_thm, } \]
\[ \text{if_rewrite_thm, } \forall \text{rewrite_thm, } \exists \text{rewrite_thm, } \beta \text{rewrite_thm, } \text{simple}_\beta \text{conv} \]

Stripping theorems:

\[ \Rightarrow \text{thm, } \iff \text{thm, } \text{simple}_\exists \text{conv, } \]
\[ \forall x \bullet ((x = x) \iff T)' \]
\[ \forall x \bullet (\neg(x = x) \iff F)' \]
\[ \forall a t1 t2 \bullet (\text{if } a \text{ then } t1 \text{ else } t2) \iff (a \Rightarrow t1) \land (\neg a \Rightarrow t2) \]

Note that these are intended to be used with (simple_ \neg \text{in_conv} from "paired_abstractions" or "simple_abstractions", which covers the cases of an outermost \neg for each operator.

Stripping conclusions:

\[ \iff \text{thm, } \]
\[ \forall x \bullet ((x = x) \iff T)' \]
\[ \forall x \bullet (\neg(x = x) \iff F)' \]
\[ \forall a t1 t2 \bullet (\text{if } a \text{ then } t1 \text{ else } t2) \iff (a \Rightarrow t1) \land (\neg a \Rightarrow t2) \]
\[ \forall a b (a \lor \neg b) \iff (b \Rightarrow a) '\]
\[ \forall a b (a \land b) \iff (a \Rightarrow b)' \]
\[ \forall a b (a \land b) \iff (a \Rightarrow b) ' \]

Note that the above are intended to be used in combination with (simple_ \neg \text{in_conv} from "paired_abstractions" or "simple_abstractions", which covers the cases of an outermost \neg for each operator.

Rewriting canonicalisation:

\[ \land \text{rewrite_canon, } f \_ \text{rewrite_canon} \]

Automatic proof procedures are respectively taut_tac, taut_conv and basic_prove_3_conv.

Usage Notes  Usually used in conjunction with "paired_abstractions" or "simple_abstractions", requires theory basic_hol.
8.8. Proof Contexts

SML

(*) Proof Context: '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.

-------------------------------------------------------------

SML

(*) Proof Context: 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.

-------------------------------------------------------------

SML

(*) Proof Context: 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.
SML

(* Proof Context: 'pair *)

Description A “mild” component proof context for theory pair.

Contents Rewriting (selected from pair_clauses):

\[\forall x y a b p \text{ fu} \text{ fc} \]
- \(\text{Fst} (x, y) = x\)
- \(\text{Snd} (x, y) = y\)
- \((a, b) = (x, y) \iff a = x \land b = y\)
- \((\text{Fst} p, \text{Snd} p) = p\)
- \(\text{Curry fc} x y = fc (x, y)\)
- \(\text{Uncurry fu} (x, y) = fu x y\)
- \(\text{Uncurry fu} p = fu (\text{Fst} p) (\text{Snd} p)\)

Stripping theorems:

\[\forall a b x y \cdot ((a, b) = (x, y) \iff a = x \land b = y)\]

Stripping conclusions:

\[\forall a b x y \cdot ((a, b) = (x, y) \iff a = x \land b = y)\]

Existential variable structures:

\[\forall x y p \cdot\]
- \(\text{Fst} (x, y) = x \land\)
- \(\text{Snd} (x, y) = y \land\)
- \((\text{Fst} p, \text{Snd} p) = p\)

Automatic proof procedures are respectively basic_prove_tac, basic_prove_conv and no existence prover.

Usage Notes Requires theory basic_hol.
8.8. Proof Contexts

SML

(* Proof Context: 'pair1 *)

Description  An “aggressive” component proof context for theory pair.

Contents  Rewriting:

\[ \forall a b p \quad (a, b) = p \iff a = \text{Fst } p \land b = \text{Snd } p \land (p = (a, b) \iff \text{Fst } p = a \land \text{Snd } p = b) \]

Stripping theorems (selected from pair_clauses):

\[ \forall a b p \quad (a, b) = p \iff a = \text{Fst } p \land b = \text{Snd } p \land (p = (a, b) \iff \text{Fst } p = a \land \text{Snd } p = b) \]

Stripping conclusions:

\[ \forall a b p \quad (a, b) = p \iff a = \text{Fst } p \land b = \text{Snd } p \land (p = (a, b) \iff \text{Fst } p = a \land \text{Snd } p = b) \]

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:

\[ \geq \text{def}, \text{greater_def}, \text{plus_clauses}, \text{times_clauses}, \]
\[ \leq \text{clauses}, \text{less_clauses}, \text{minus_clauses} \]

Stripping theorems:

\[ \geq \text{def}, \text{greater_def}, \text{plus_clauses}, \text{times_clauses}, \]
\[ \leq \text{clauses}, \text{less_clauses}, \text{minus_clauses}, \]
\[ \text{and all boolean equations also pushed through } \neg\neg \]

Stripping conclusions:

\[ \geq \text{def}, \text{greater_def}, \text{plus_clauses}, \text{times_clauses}, \]
\[ \leq \text{clauses}, \text{less_clauses}, \text{minus_clauses}, \]
\[ \text{and all boolean equations also pushed through } \neg\neg \]

Existential clausal definition theorems:

\[ \text{prim_rec_thm} \]

Automatic proof procedures are respectively basic_prove_tac, basic_prove_conv and no existence prover.

Usage Notes  Requires theory basic_hol.
(* Proof Context: \texttt{\textbackslash N\_lit \textbackslash} *)

**Description** A component proof context for theory \texttt{N}, that will, e.g., evaluate any arithmetic expression involving only numeric literals and certain arithmetic operators, namely \(+, \times, -, \div, \mod, \leq, <, >, \geq, \text{and} =\).

**Contents** Rewriting:

\begin{align*}
plus\_conv, & \times\_conv, minus\_conv, \div\_conv, \\
mod\_conv, & \leq\_conv, less\_conv, greater\_conv, \\
\geq\_conv, & \text{N}\_eq\_conv
\end{align*}

Stripping theorems:

\begin{align*}
\leq\_conv, & less\_conv, greater\_conv, \\
\geq\_conv, & \text{N}\_eq\_conv
\end{align*}

Stripping conclusions:

\begin{align*}
\leq\_conv, & less\_conv, greater\_conv, \\
\geq\_conv, & \text{N}\_eq\_conv
\end{align*}

Existential clausal definition theorems:

Automatic proof procedures are respectively \texttt{basic\_prove\_tac}, \texttt{basic\_prove\_conv} and no existence prover.

**Usage Notes** Requires theory \texttt{basic\_hol}, expected to be used with proof context \texttt{"N"}. It is separated from it as spotting the application of the conversions is time consuming, and may be known to be irrelevant.
8.8. Proof Contexts

SML

|(* Proof Context: 'list *)

Description A component proof context for the theory list.

Contents Rewriting:

|list_clauses

Stripping theorems:

\[\begin{align*}
\forall x_1 x_2 \text{list1 list2} & \\
\quad \bullet \neg \text{Cons } x_1 \text{list1} = [] & \\
\quad \wedge \neg [] = \text{Cons } x_1 \text{list1} & \\
\quad \wedge (\text{Cons } x_1 \text{list1} = \text{Cons } x_2 \text{list2} \iff x_1 = x_2 \wedge \text{list1} = \text{list2})
\end{align*}\]

Stripping conclusions:

\[\begin{align*}
\forall x_1 x_2 \text{list1 list2} & \\
\quad \bullet \neg \text{Cons } x_1 \text{list1} = [] & \\
\quad \wedge \neg [] = \text{Cons } x_1 \text{list1} & \\
\quad \wedge (\text{Cons } x_1 \text{list1} = \text{Cons } x_2 \text{list2} \iff x_1 = x_2 \wedge \text{list1} = \text{list2})
\end{align*}\]

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.

SML

|(* 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.
Chapter 8. PROOF IN HOL

<table>
<thead>
<tr>
<th>SML</th>
<th>(* Proof Context: basic_hol *)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>A “mild” complete proof context for the ancestors of theory basic_hol.</td>
</tr>
<tr>
<td><strong>Contents</strong></td>
<td>Proof contexts “predicates”, “pair”, “N”, “list”, and “char”. 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).</td>
</tr>
<tr>
<td><strong>Usage Notes</strong></td>
<td>Requires theory basic_hol.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>(* Proof Context: basic_hol1 *)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>An “aggressive” complete proof context for the ancestors of theory basic_hol.</td>
</tr>
<tr>
<td><strong>Contents</strong></td>
<td>Proof contexts “predicates1”, “pair”, “pair1”, “N”, “N_lit”, “list”, and “char”. 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).</td>
</tr>
<tr>
<td><strong>Usage Notes</strong></td>
<td>Requires theory basic_hol.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>(* Proof Context: 'mmp1 *)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>A component proof context with the matching modus ponens rule set to ⇒_match_mp_rule1. All other fields are empty.</td>
</tr>
<tr>
<td><strong>Usage Notes</strong></td>
<td>This makes forward chaining work as in releases prior to 2.9.1 (so that bound variables that are not constrained by the pattern matching are specialised to themselves).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>(* Proof Context: 'mmp2 *)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>A component proof context with the matching modus ponens rule set to ⇒_match_mp_rule2. All other fields are empty.</td>
</tr>
<tr>
<td><strong>Usage Notes</strong></td>
<td>Use this to ensure the default behaviour in forward chaining (so that bound variables that are not constrained by the pattern matching are specialised with new names as necessary to avoid variable capture).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SML</th>
<th>(* Proof Context: 'sho_rw *)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>A component proof context with the equation matching rule set to simple_higher_order_thm_eqn_cxt. All other fields are empty.</td>
</tr>
<tr>
<td><strong>Usage Notes</strong></td>
<td>With this proof context, rewriting treats the rewriting theorems as higher order rewrite rules. For example, rewriting with the theorem prenex_clauses (q.v.) will convert a term into prenex normal form.</td>
</tr>
</tbody>
</table>
8.8. Proof Contexts

SML

(* Proof Context: 'sum *)

Description A “mild” component proof context for theory sum.

Contents Rewriting:

\[ \forall x_1 \ x_2 \ y_1 \ y_2 \ z \]
\[ \ (\text{InL } x_1 = \text{InL } x_2 \iff x_1 = x_2) \]
\[ \land (\text{InR } y_1 = \text{InR } y_2 \iff y_1 = y_2) \]
\[ \land \neg \text{InL } x_1 = \text{InR } y_1 \]
\[ \land \neg \text{InR } y_1 = \text{InL } x_1 \]
\[ \land \text{OutL } (\text{InL } x_1) = x_1 \]
\[ \land \text{OutR } (\text{InR } y_1) = y_1 \]
\[ \land \text{IsL}(\text{InL } x_1) \land \text{IsR}(\text{InR } y_1) \]
\[ \land \neg \text{IsL}(\text{InR } y_1) \land \neg \text{IsR}(\text{InL } x_1) \]

Stripping theorems:

\[ \forall x_1 \ x_2 \ y_1 \ y_2 \ z \]
\[ \ (\text{InL } x_1 = \text{InL } x_2 \iff x_1 = x_2) \]
\[ \land (\text{InR } y_1 = \text{InR } y_2 \iff y_1 = y_2) \]
\[ \land \neg \text{InL } x_1 = \text{InR } y_1 \]
\[ \land \neg \text{InR } y_1 = \text{InL } x_1 \]
\[ \land \text{IsL}(\text{InL } x_1) \land \text{IsR}(\text{InR } y_1) \]
\[ \land \neg \text{IsL}(\text{InR } y_1) \land \neg \text{IsR}(\text{InL } x_1) \]

Stripping conclusions:

\[ \forall x_1 \ x_2 \ y_1 \ y_2 \ z \]
\[ \ (\text{InL } x_1 = \text{InL } x_2 \iff x_1 = x_2) \]
\[ \land (\text{InR } y_1 = \text{InR } y_2 \iff y_1 = y_2) \]
\[ \land \neg \text{InL } x_1 = \text{InR } y_1 \]
\[ \land \neg \text{InR } y_1 = \text{InL } x_1 \]
\[ \land \text{IsL}(\text{InL } x_1) \land \text{IsR}(\text{InR } y_1) \]
\[ \land \neg \text{IsL}(\text{InR } y_1) \land \neg \text{IsR}(\text{InL } x_1) \]

Existential clausal definition theorems:

\[ \forall f \ g \exists h \ ((\forall x \bullet h (\text{InL } x) = f x) \land (\forall x \bullet h (\text{InR } x) = g x)) \]

Automatic proof procedures are respectively basicprove_tac, basicprove_conv and no existence prover.

Usage Notes Requires theory sum.
Chapter 8. PROOF IN HOL

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:

\[ \vdash \forall x\ y: ONE \cdot (x = y) \Leftrightarrow T' \]
\[ \vdash \forall x\ y: 'a \rightarrow ONE \cdot (x = y) \Leftrightarrow T' \]
and through \neg

Stripping conclusions:

\[ \vdash \forall x\ y: ONE \cdot (x = y) \Leftrightarrow T' \]
\[ \vdash \forall x\ y: 'a \rightarrow ONE \cdot (x = y) \Leftrightarrow T' \]
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.
SML

(* Proof Context: 'sets_alg *)

Description A “mild” component proof context for theory set.

Contents Rewriting:

\[ \begin{align*}
\in_{\text{comp}} & \colon \text{comp} \quad \text{conv} \\
\in_{\text{enum}} & \colon \text{enum.set.conv} \quad \text{complement.clauses} \\
\cup_{\text{clauses}} & \colon \text{clauses} \quad \cap_{\text{clauses}} \quad \setminus_{\text{clauses}} \\
\subseteq_{\text{clauses}} & \colon \text{clauses} \quad \subset_{\text{clauses}} \quad \bigcup_{\text{clauses}} \quad \bigcap_{\text{clauses}} \quad \mathbb{P}_{\text{clauses}}
\end{align*} \]

\[ t \vdash \forall x \ y \quad (\neg x \in \emptyset \quad \wedge x \in \text{Universe} \\
\quad \wedge (x \in \{y\} \iff x = y) \]  

Stripping theorems:

\[ \begin{align*}
\in_{\text{comp}} & \colon \text{comp} \quad \text{conv} \quad \in_{\text{enum.set.conv}} \\
\subseteq_{\text{clauses}} & \colon \text{clauses} \quad \subseteq_{\text{clauses}} \quad \subset_{\text{clauses}} \quad \bigcup_{\text{clauses}} \quad \bigcap_{\text{clauses}} \quad \mathbb{P}_{\text{clauses}}
\end{align*} \]

plus these all pushed in through \( \neg \)

Stripping conclusions:

\[ \begin{align*}
\in_{\text{comp}} & \colon \text{comp} \quad \text{conv} \quad \in_{\text{enum.set.conv}} \\
\subseteq_{\text{clauses}} & \colon \text{clauses} \quad \subseteq_{\text{clauses}} \quad \subset_{\text{clauses}} \quad \bigcup_{\text{clauses}} \quad \bigcap_{\text{clauses}} \quad \mathbb{P}_{\text{clauses}}
\end{align*} \]

plus these all pushed in through \( \neg \)

Automatic proof procedures are respectively basic.prove_tac, basic.prove_conv and the existence prover preprocessor:

\[ \text{TOP.MAP.C (all.\_uncurry.conv AND.OR.C sets.simple.\_conv)} \]

The preprocessor causes set membership (\( \in \)) to be treated as function application in some cases.

Usage Notes Should not be used with proof context "sets_ext", requires theory sets.
Description  A component proof context for theory set, “aggressively” using the extensionality of sets.

Contents  Rewriting:
\[\in\_\text{comp}\_\text{conv}, \in\_\text{enum}\_\text{set}\_\text{conv}, \in\_\text{in}\_\text{clauses}, \text{sets}\_\text{ext}\_\text{clauses}\]

Stripping theorems:
\[\in\_\text{comp}\_\text{conv}, \in\_\text{enum}\_\text{set}\_\text{conv}, \in\_\text{in}\_\text{clauses}, \text{sets}\_\text{ext}\_\text{clauses}\]
\text{plus these all pushed in through } \neg

Stripping conclusions:
\[\in\_\text{comp}\_\text{conv}, \in\_\text{enum}\_\text{set}\_\text{conv}, \in\_\text{in}\_\text{clauses}, \text{sets}\_\text{ext}\_\text{clauses}\]
\text{plus these all pushed in through } \neg

Automatic proof procedures are respectively \text{basic}\_\text{prove}\_\text{tac}, \text{basic}\_\text{prove}\_\text{conv} and the existence prover preprocessor:
\[\text{TOP}\_\text{MAP}\_\text{C} (\text{all}\_\exists\_\text{uncurry}\_\text{conv} \mathbin{\text{AND}} \text{OR}\_\text{C} \text{sets}\_\text{simple}\_\exists\_\text{conv})\]

The preprocessor causes set membership (\(\in\)) to be treated as function application in some cases.

Usage Notes  Should not be used with proof context “sets\_alg”, requires theory \text{sets}.

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 \text{sets}. The proof context “sets\_ext1” offers a much more useful treatment of sets of pairs.
8.8. Proof Contexts

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:

\[
\begin{align*}
\in_{\text{comp.conv}}, \in_{\text{enum.set.conv}}, \in_{\text{in.clauses}}, \\
\seteq_{\text{conv}}, \subseteq_{\text{conv}}, \subset_{\text{conv}}
\end{align*}
\]

Stripping theorems:

\[
\begin{align*}
\in_{\text{comp.conv}}, \in_{\text{enum.set.conv}}, \in_{\text{in.clauses}}, \\
\seteq_{\text{conv}}, \subseteq_{\text{conv}}, \subset_{\text{conv}} \\
\text{plus these all pushed in through } \neg
\end{align*}
\]

Stripping conclusions:

\[
\begin{align*}
\in_{\text{comp.conv}}, \in_{\text{enum.set.conv}}, \in_{\text{in.clauses}}, \\
\seteq_{\text{conv}}, \subseteq_{\text{conv}}, \subset_{\text{conv}} \\
\text{plus these all pushed in through } \neg
\end{align*}
\]

Automatic proof procedures are respectively *basic_prove_tac*, *basic_prove_conv* and the existence prover preprocessor:

\[
\text{TOP.MAP.C (all} \exists_{\text{uncurry.conv AND.OR.C sets.simple} \exists_{\text{conv}})
\]

The preprocessor causes set membership (\(\in\)) 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_{\text{conv}}* (merged in from the proof context of the same name).

**Usage Notes**  Requires theory *hol*.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUAL USR029
### Chapter 8. PROOF IN HOL

#### 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>Automatic proof procedures are respectively <code>basic_prove_tac</code>, <code>basic_prove_conv</code> and <code>basic_prove-\_∃\_conv</code> (merged in from the proof context of the same name).</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>Automatic proof procedures are respectively <code>basic_prove_tac</code>, <code>basic_prove_conv</code> and <code>basic_prove-\_∃\_conv</code> (merged in from the proof context of the same name).</td>
</tr>
<tr>
<td><strong>Usage Notes</strong></td>
</tr>
</tbody>
</table>
Proof Contexts

val basic_prove_conv : THM list -> CONV;

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 \( \vdash \text{term} \Leftrightarrow T \) 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:

\[
\begin{align*}
\text{fun basic_prove_conv thms tm } &= \nabla_t \text{intro} (\nabla_t \text{proof}(([],tm),
\text{TRY} T (\text{rewrite_tac thms}) \text{ THEN }
\text{REPEAT} strip_tac \text{ THEN_TRY}
(basic_res_tac2 3 [\vdash \forall x \bullet x = x]
ORELSE_T basic_res_tac3 3 [\vdash \forall x \bullet x = x])))
\end{align*}
\]

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

```sml
fun basic_prove_tac thms = 
TRY_T all_var_elim_asm_tac THEN
DROP_ASMS_T (MAP_EVERY (strip_asm_tac o rewrite_rule thms) o rev) THEN
(TRY_T (rewrite_tac thms)) THEN
TRY_T all_var_elim_asm_tac THEN
REPEAT strip_tac THEN_TRY
(basic_res_tac2 3 |\ x • x = x) 
ORELSE_T basic_res_tac3 3 |\ x • x = x])
```

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
9.1 Theory Listings

This section contains the listings of each theory.
9.1.1 The Theory basic_hol

9.1.1.1 Parents

\textit{char}

9.1.1.2 Children

\textit{sets combin one}

9.1.1.3 Notes

This theory is a cache theory; its contents have not been listed.
9.1. The Theory Listings

9.1.2 The Theory bin_rel

9.1.2.1 Parents

\textit{hol}

9.1.2.2 Children

\textit{fun_rel}

9.1.2.3 Constants

\begin{itemize}
  \item $\rightarrow$ \quad \texttt{a -> 'b -> 'a x 'b}
  \item $\times$ \quad \texttt{a SET -> 'b SET -> ('a x 'b) SET}
  \item $\leftrightarrow$ \quad \texttt{a SET -> 'b SET -> ('a x 'b) SET SET}
  \item \texttt{Dom} \quad \texttt{('a x 'b) SET -> 'a SET}
  \item \texttt{Ran} \quad \texttt{('a x 'b) SET -> 'b SET}
  \item \texttt{Id} \quad \texttt{'a SET -> ('a x 'a) SET}
  \item \texttt{Graph} \quad \texttt{('a -> 'b) -> ('a x 'b) SET}
  \item $\wedge_\times$ \quad \texttt{('a x 'b) SET -> ('b x 'c) SET -> ('a x 'c) SET}
  \item $\circ_\times$ \quad \texttt{('b x 'c) SET -> ('a x 'b) SET -> ('a x 'c) SET}
  \item $\times_\wedge$ \quad \texttt{('a SET -> ('a x 'b) SET -> ('a x 'b) SET}
  \item $\circ_\wedge$ \quad \texttt{('a x 'b) SET -> 'b SET -> ('a x 'b) SET}
  \item $\wedge_\circ$ \quad \texttt{('a SET -> ('a x 'b) SET -> ('a x 'b) SET}
  \item $\circ_\circ$ \quad \texttt{('a x 'b) SET -> 'b SET -> ('a x 'b) SET}
  \item \texttt{InvRel} \quad \texttt{('a x 'b) SET -> ('b x 'a) SET}
  \item \texttt{Image} \quad \texttt{('a x 'b) SET -> 'a SET -> 'b SET}
  \item \texttt{Reflexive} \quad \texttt{('a x 'a) SET SET}
  \item \texttt{Symmetric} \quad \texttt{('a x 'a) SET SET}
  \item \texttt{Transitive} \quad \texttt{('a x 'a) SET SET}
  \item \texttt{Injective} \quad \texttt{('a x 'b) SET SET}
  \item \texttt{Surjective} \quad \texttt{'b SET -> ('a x 'b) SET SET}
  \item \texttt{Total} \quad \texttt{'a SET -> ('a x 'b) SET SET}
  \item \texttt{Functional} \quad \texttt{('a x 'b) SET SET}
  \item $\oplus_\times$ \quad \texttt{('a x 'b) SET -> ('a x 'b) SET -> ('a x 'b) SET}
  \item $\circlearrowright_\times$ \quad \texttt{('a x 'a) SET -> ('a x 'a) SET}
  \item $\circlearrowleft_\times$ \quad \texttt{('a x 'a) SET -> ('a x 'a) SET}
  \item \texttt{RelCombine} \quad \texttt{('a x 'b) SET -> ('a x 'c) SET -> ('a x 'b x 'c) SET}
\end{itemize}

9.1.2.4 Aliases

\begin{itemize}
  \item $\circlearrowleft_\times \circlearrowright_\times$ : \texttt{('b x 'a) SET -> ('a x 'c) SET -> ('b x 'c) SET}
  \item $\circlearrowleft_\times \circlearrowright_\times$ : \texttt{('a x 'c) SET -> ('b x 'a) SET -> ('b x 'c) SET}
  \item $\circlearrowleft_\times \circlearrowright_\times$ : \texttt{('b x 'a) SET -> ('a x 'b) SET}
\end{itemize}

9.1.2.5 Type Abbreviations

\begin{itemize}
  \item \texttt{a P} \quad \texttt{'a SET}
  \item \texttt{a -> b} \quad \texttt{('a x 'b) SET}
\end{itemize}
9.1.2.6 Fixity

Right Infix 240:
\[ \begin{align*}
R_{o\cdot R} & \quad R_{\circ \cdot R^\rightarrow} & \rightarrow & \leftrightarrow & \oplus & \circ \circ \\
\end{align*} \]

Right Infix 280:
Image

Right Infix 300:
\[ \rightarrow, +, \sim \]

9.1.2.7 Definitions

\[ \begin{align*}
\mapsto & \quad \vdash s \mapsto = s, \\
\times & \quad \vdash \forall x \, y \bullet (x \times y) = \{(v, w) | v \in x \land w \in y\} \\
\leftrightarrow & \quad \vdash \forall x \, y \bullet x \leftrightarrow y = \mathbb{P} (x \times y) \\
\text{Dom} & \quad \vdash \forall r \bullet \text{Dom } r = \{x | \exists y \bullet (x, y) \in r\} \\
\text{Ran} & \quad \vdash \forall r \bullet \text{Ran } r = \{y | \exists x \bullet (x, y) \in r\} \\
\text{Id} & \quad \vdash \forall s \bullet \text{Id } s = \{(x, y) | x = y \land x \in s\} \\
\text{Graph} & \quad \vdash \forall f \bullet \text{Graph } f = \{(x, y) | y = f x\} \\
\% & \quad \vdash \forall g \bullet f \circ g = g \circ f \\
R_{\circ \cdot R} & \quad \vdash \forall r \, s \bullet r \circ s = \{(x, z) | \exists y \bullet (x, y) \in r \land (y, z) \in s\} \\
R_{o\cdot R} & \quad \vdash \forall r \, s \bullet r \circ o \circ s = s \circ r \\
\circ & \quad \vdash \forall a \bullet a \circ r = \{(x, y) | x \in a \land (x, y) \in r\} \\
\circ \circ & \quad \vdash \forall a \bullet r \circ \circ a = \{(x, y) | y \in a \land (x, y) \in r\} \\
\circ \circ & \quad \vdash \forall a \bullet a \circ \circ r = \{(x, y) | \neg x \in a \land (x, y) \in r\} \\
\circ \circ & \quad \vdash \forall a \bullet r \circ \circ a = \{(x, y) | \neg y \in a \land (x, y) \in r\} \\
\text{InvRel} & \quad \vdash \forall r \bullet r \sim = \{(x, y) | (y, x) \in r\} \\
\text{Image} & \quad \vdash \forall s \bullet r \text{ Image } s = \{y | \exists x \bullet x \in s \land (x, y) \in r\} \\
\text{Reflexive} & \quad \vdash \text{Reflexive} = \{r | \forall x \bullet (x, x) \in r\} \\
\text{Symmetric} & \quad \vdash \text{Symmetric} = \{r | \forall x \, y \bullet (x, y) \in r \Rightarrow (x, y) \in r\} \\
\text{Transitive} & \quad \vdash \text{Transitive} \\
& \quad = \{r | \forall x \, y \, z \bullet (x, y) \in r \land (y, z) \in r \Rightarrow (x, z) \in r\} \\
\text{Injective} & \quad \vdash \text{Injective} \\
& \quad = \{r | \forall x \, y \, z \bullet (x, z) \in r \land (y, z) \in r \Rightarrow x = y\} \\
\text{Surjective} & \quad \vdash \forall s \bullet \text{Surjective } s = \{r | s = \text{Ran } r\} \\
\text{Total} & \quad \vdash \forall s \bullet \text{Total } s = \{r | s = \text{Dom } r\} \\
\text{Functional} & \quad \vdash \text{Functional} \\
& \quad = \{r | \forall x \, y \, z \bullet (x, z) \in r \land (y, z) \in r \Rightarrow w = z\} \\
\oplus & \quad \vdash \forall r \, s \bullet r \oplus s = (\text{Dom } s \preceq r) \cup s \\
\circ & \quad \vdash \forall r \bullet r \circ = \bigcap \{q | r \subseteq q \land q \in \text{Transitive}\} \\
\ast & \quad \vdash \forall r \\
& \bullet r ^* \\
& \quad = \bigcap \{q | r \subseteq q \land q \in \text{Reflexive} \land q \in \text{Transitive}\} \\
\text{RelCombine} & \quad \vdash \forall f \, g \\
& \bullet \text{RelCombine } f \, g \\
& \quad = \{(x, y, z) | (x, y) \in f \land (x, z) \in g\} \\
\end{align*} \]
9.1.2.8 Theorems

rel_in_clauses
\[ \vdash \forall a \ b \ x \ y \ z \ r \ (x \rightarrow y \in r \iff (x, y) \in r) \]
\[ \land ((x, y) \in (a \times b) \iff x \in a \land y \in b) \]
\[ \land (r \in a \iff b \iff r \subseteq (a \times b)) \]
\[ \land (x \in \text{Dom} \ r \iff (\exists y \bullet (x, y) \in r)) \]
\[ \land (y \in \text{Ran} \ r \iff (\exists x \bullet (x, y) \in r)) \]
\[ \land ((x, y) \in \text{Id} \ a \iff x = x1 \land x \in a) \]
\[ \land ((x, y) \in \text{Graph} \ f \iff y = f \ x) \]
\[ \land ((x, z) \in r \iff s \iff (\exists y \bullet (x, y) \in r \land (y, z) \in s)) \]
\[ \land ((x, z) \in s \circ r \iff (x, z) \in r \circ s) \]
\[ \land ((x, y) \in a \iff r \iff x \in a \land (x, y) \in r) \]
\[ \land ((x, y) \in r \iff b \iff y \in b \land (x, y) \in r) \]
\[ \land ((x, y) \in a \iff r \iff \neg x \in a \land (x, y) \in r) \]
\[ \land ((x, y) \in r \iff b \iff \neg y \in b \land (x, y) \in r) \]
\[ \land ((y, x) \in r \iff \neg \iff (x, y) \in r) \]
\[ \land (y \in r \iff \text{Image} \ a \iff (\exists x \bullet x \in a \land (x, y) \in r)) \]
\[ \land (q \in \text{Reflexive} \iff (\forall x \bullet (x, x) \in q)) \]
\[ \land (q \in \text{Symmetric} \iff (\forall x1 \ x2 \bullet (x1, x2) \in q \Rightarrow (x2, x1) \in q)) \]
\[ \land (q \in \text{Transitive} \iff (\forall x1 \ x2 \ x3 \bullet (x1, x2) \in q \land (x2, x3) \in q \Rightarrow (x1, x3) \in q)) \]
\[ \land (r \in \text{Injective} \iff (\forall x1 \ x2 \ y \bullet (x1, y) \in r \land (x2, y) \in r \Rightarrow x1 = x2)) \]
\[ \land (r \in \text{Surjective} \ b \iff (\forall y \bullet y \in b \iff (\exists x \bullet (x, y) \in r))) \]
\[ \land (r \in \text{Total} \ a \iff (\forall x \bullet x \in a \iff (\exists y \bullet (x, y) \in r))) \]
\[ \land (r \in \text{Functional} \iff (\forall x y \ y2 \bullet (x, y1) \in r \land (x, y2) \in r \Rightarrow y1 = y2)) \]
\[ \land ((x, y) \in r1 \oplus r2 \iff (x, y) \in (\text{Dom} \ r2 \leq r1) \cup r2) \]
\[ \land ((x, y, z) \in \text{RelCombine} \ r \ t \iff (x, y) \in r \land (x, z) \in t) \]

bin_rel_ext_clauses
\[ \vdash \forall r1 \ r2 \bullet (r1 \subseteq r2) \]
\[ \iff (\forall x \bullet (x, y) \in r1 \Rightarrow (x, y) \in r2) \]
\[ \land (\exists x \bullet y \neg (x, y) \in r1 \land (x, y) \in r2)) \]
\[ \land (r1 \subseteq r2 \iff (\forall x y \bullet (x, y) \in r1 \Rightarrow (x, y) \in r2)) \]
\[ \land (r1 = r2 \iff (\forall x y \bullet (x, y) \in r1 \iff (x, y) \in r2)) \]

inv_thm
\[ \vdash \forall f \ a \ b \bullet (f ^\sim \in \text{Functional} \iff f \in \text{Injective}) \]
\[ \land (f ^\sim \in \text{Injective} \iff f \in \text{Functional}) \]
\[ \land (f ^\sim \in \text{Surjective} \ a \iff f \in \text{Total} \ a) \]
\[ \land (f ^\sim \in \text{Total} \ b \iff f \in \text{Surjective} \ b) \]
bin_rel_∅_universe_thm

\[ \vdash \forall f \ g \ r0 \ r1 \ a \ b \]

\[ (\text{Dom } r0 = \{\} \iff r0 = \{\}) \]
\[ \land (\text{Dom } r0 = \{\} \iff r0 = \{\}) \]
\[ \land (\text{Dom } \{\} = \{\}) \]
\[ \land (\text{Ran } \{\} = \{\}) \]
\[ \land (\text{Dom Universe} = \text{Universe}) \]
\[ \land (\text{Ran Universe} = \text{Universe}) \]
\[ \land (\text{Dom } r0 = \{\} \iff r0 = \{\}) \]
\[ \land (\{\} \circ r0 = \{\} \iff r0 = \{\}) \]
\[ \land (\text{Dom } \{\} = \{\}) \]
\[ \land (\text{Id } \{\} = \{\}) \]
\[ \land (\text{Dom } r0 = \{\} \iff r0 = \{\}) \]
\[ \land (\{\} \circ r0 = \{\} \iff r0 = \{\}) \]
\[ \land (\text{Id } \{\} = \{\}) \]
\[ \land (\text{Dom } r0 = \{\} \iff r0 = \{\}) \]
\[ \land (\{\} \circ r0 = \{\} \iff r0 = \{\}) \]
\[ \land (\text{RelCombine } r0 \{\} = \{\}) \]
\[ \land (\text{RelCombine } \{\} \circ r0 = \{\}) \]
\[ \land (\text{Dom Image } a = \{\}) \]
\[ \land (\text{Image } a = \{\}) \]
\[ \land (f \oplus \{\} = f) \]
\[ \land (\{\} \oplus f = f) \]
\[ \land (f \oplus g = \{\} \iff f = \{\} \land g = \{\}) \]
\[ \land (f = \{\} \land g = \{\} \Rightarrow f \oplus g = \{\}) \]
\[ \land (f \oplus g = \{\} \iff f = \{\} \land g = \{\}) \]
\[ \land f \oplus \text{Universe} = \text{Universe} \]
\[ \land (\text{Universe} \less r0 = \{\}) \]
\[ \land (a \less \{\} = \{\}) \]
\[ \land (\{\} \less r0 = r0) \]
\[ \land (\text{Universe} \less r0 = r0) \]
\[ \land (a \less \{\} = \{\}) \]
\[ \land (\{\} \less r0 = \{\}) \]
\[ \land (r0 \less \text{Universe} = \{\}) \]
\[ \land (\{\} \less b = \{\}) \]
\[ \land (r0 \less \{\} = r0) \]
\[ \land (r0 \less \text{Universe} = r0) \]
\[ \land (\{\} \less b = \{\}) \]
\[ \land (r0 \less \{\} = \{\}) \]

bin_rel_insert_thm

\[ \vdash \forall a \ b \ r \ x \ y \]

\[ \land (\text{Dom } (\text{Insert } (x, y) \ r) = \text{Insert } x (\text{Dom } r) \]
\[ \land (\text{Dom } (\text{Insert } xy \ r) = \text{Insert } (\text{Fst } xy) (\text{Dom } r) \]
\[ \land (\text{Ran } (\text{Insert } (x, y) \ r) = \text{Insert } y (\text{Ran } r) \]
\[ \land (\text{Ran } (\text{Insert } xy \ r) = \text{Insert } (\text{Snd } xy) (\text{Ran } r) \]
\[ \land (\text{Id } (\text{Insert } x \ a) = \text{Insert } x x (\text{Id } a) \]
\[ \land (\text{Insert } (x, y) \ r \text{ Image } a \]
\[ = (r \text{ Image } a \cup \{(\text{if } x \in a \text{ then } y \text{ else } \}) \]
\[ \land (\text{Insert } (x, y) \ r \bowtie b \]
\[ = (r \bowtie b) \cup \{(\text{if } \neg y \in b \text{ then } ((x, y) \text{ else } \}) \]
\[ \land (\text{Insert } (x, y) \ r \bowtie b \]

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
\[ (r \upharpoonright b) \cup \{ (x, y) \mid y \in b \} \]
\[ \land a \trianglerighteq \operatorname{Insert}(x, y) r \]
\[ = (a \trianglerighteq r) \cup \{ (x, y) \mid \neg x \in a \} \]
\[ \land a \trianglerighteq \operatorname{Insert}(x, y) r \]
\[ = (a \triangleright r) \cup \{ (x, y) \mid x \in a \} \]
\[ \land \operatorname{Insert}(x, y) r \sim = \operatorname{Insert}(y, x) (r \sim) \]
9.1.3 The Theory char

9.1.3.1 Parents

   list

9.1.3.2 Children

   basic_hol

9.1.3.3 Constants

IsCharRep \( \mathbb{N} \to \mathbb{BOOL} \)
RepChar \( \text{CHAR} \to \mathbb{N} \)
AbsChar \( \mathbb{N} \to \text{CHAR} \)

9.1.3.4 Types

CHAR

9.1.3.5 Type Abbreviations

STRING \( \text{CHAR LIST} \)

9.1.3.6 Definitions

IsCharRep
is_char_rep_def
\[ \vdash \forall x \cdot \text{IsCharRep} \; x \iff x < 256 \]

CHAR
char_def
\[ \vdash \exists f \cdot \text{TypeDefn} \; \text{IsCharRep} \; f \]

AbsChar
RepChar
abs_char_rep_char_def
\[ \vdash (\forall a \cdot \text{AbsChar} \; (\text{RepChar} \; a) = a) \]
\[ \land (\forall r \cdot \text{IsCharRep} \; r \iff \text{RepChar} \; (\text{AbsChar} \; r) = r) \]
9.1.4  The Theory combin

9.1.4.1  Parents

\[ basic\_hol \]

9.1.4.2  Children

\[ sum \]

9.1.4.3  Constants

- \$o\$ \( (\text{'}b \to \text{'}c) \to (\text{'}a \to \text{'}b) \to \text{'}a \to \text{'}c \)
- \text{CombS}\quad (\text{'}a \to \text{'}b \to \text{'}c) \to (\text{'}a \to \text{'}b) \to \text{'}a \to \text{'}c
- \text{CombK}\quad \text{'}a \to \text{'}b \to \text{'}a
- \text{CombI}\quad \text{'}a \to \text{'}a

9.1.4.4  Fixity

Right Infix 300:

\[ o \]

9.1.4.5  Definitions

\[ o \]
\[ o\_def \quad \vdash \forall f \ g \ x \bullet (f \ o \ g) \ x = f \ (g \ x) \]
\[ \text{CombS}\quad \text{comb\_s\_def} \quad \vdash \forall f \ g \ x \bullet \text{CombS} \ f \ g \ x = f \ x \ (g \ x) \]
\[ \text{CombK}\quad \text{comb\_k\_def} \quad \vdash \forall x \ y \bullet \text{CombK} \ x \ y = x \]
\[ \text{CombI}\quad \text{comb\_i\_def} \quad \vdash \forall x \bullet \text{CombI} \ x = x \]

9.1.4.6  Theorems

\[ s\_k\_thm \quad \vdash \forall x \bullet \text{CombS} \ \text{CombK} \ x = \text{CombI} \]
\[ o\_assoc\_thm \quad \vdash \forall f \ g \ h \bullet f \ o \ g \ o \ h = (f \ o \ g) \ o \ h \]
\[ o\_i\_thm \quad \vdash \forall f \bullet \text{CombI} \ o \ f = f \ \land \ f \ o \ \text{CombI} = f \]
9.1.5 The Theory dyadic

9.1.5.1 Parents

$Z$

9.1.5.2 Children

$R$

9.1.5.3 Constants

$\vdash N \rightarrow N \rightarrow N$

$\vdash dy\_times N \times Z \rightarrow N \times Z \rightarrow N \times Z$

$\vdash dy\_one N \times Z$

$\vdash dy\_exp N \times Z \rightarrow N \rightarrow N \times Z$

$\vdash dy\_less N \times Z \rightarrow N \times Z \rightarrow BOOL$

9.1.5.4 Type Abbreviations

DYADIC $N \times Z$

9.1.5.5 Fixity

Right Infix 210:

\[ dy\_less \]

Right Infix 310:

\[ dy\_exp \quad dy\_times \]

Right Infix 320:

\[ ~ \]

9.1.5.6 Definitions

\[ \vdash (\forall i \cdot i \sim 0 = 1) \land (\forall i m \cdot i \sim (m + 1) = i \star i \sim m) \]

\[ \vdash \forall (m, i) (n, j) \]

\[ \bullet (m, i) \ dy\_times (n, j) = (2 \star m \star n + m + n, i + j) \]

\[ \vdash dy\_one = (0, NZ 0) \]

\[ \vdash (\forall x \bullet x \ dy\_exp 0 = dy\_one) \]

\[ \land (\forall m x \bullet x \ dy\_exp (m + 1) = x \ dy\_times x \ dy\_exp m) \]

\[ \vdash \forall (m, i) (n, j) \]

\[ \bullet (m, i) \ dy\_less (n, j) \]

\[ \leftrightarrow (\exists a b \]

\[ \bullet NZ a + i = NZ b + j \]

\[ \land 2 \sim a \star (2 \star m + 1) < 2 \sim b \star (2 \star n + 1)) \]
9.1. Theory Listings

9.1.5.7 Theorems

dy_less_irrefl_thm
\[ \vdash \forall x \neg \text{dy_less } x \]
dy_less_antisym_thm
\[ \vdash \forall x y (x \text{dy_less } y \wedge y \text{dy_less } x) \]
dy_less_trans_thm
\[ \vdash \forall x y z (x \text{dy_less } y \wedge y \text{dy_less } z \Rightarrow x \text{dy_less } z) \]
dy_less_trich_thm
\[ \vdash \forall x y z \neg (x \text{dy_less } y \wedge y \text{dy_less } x) \]
dy_times_comm_thm
\[ \vdash \forall x y (x \text{dy_times } y = y \text{dy_times } x) \]
dy_times_assoc_thm
\[ \vdash \forall x y z (x \text{dy_less } y \Rightarrow x \text{dy_less } z) \]
dy_times_order_thm
\[ \vdash \forall u x y \quad \bullet (x \text{dy_times } y) \text{dy_times } z = x \text{dy_times } y \text{dy_times } z \]
dy_times_unit_thm
\[ \vdash \forall x \bullet x \text{dy_times } \text{dy_one} = x \]
dy_times_unit_clauses
\[ \vdash \forall x \text{dy_times } \text{dy_one} = x \land \text{dy_one } \text{dy_times } x = x \]
dy_exp_clauses
\[ \vdash \forall x m n \quad \bullet x \text{dy_exp } (m + n) = (x \text{dy_exp } m) \text{dy_times } x \text{dy_exp } n \]
\[ \land x \text{dy_exp } 0 = \text{dy_one} \]
\[ \land x \text{dy_exp } 1 = x \]
\[ \land x \text{dy_exp } 2 = x \text{dy_times } x \]
\[ \land x \text{dy_exp } 3 = x \text{dy_times } x \text{dy_times } x \]
dy_times_mono_thm
\[ \vdash \forall x y z \quad \bullet y \text{dy_less } z \Rightarrow x \text{dy_times } y \text{dy_less } x \text{dy_times } z \]
dy_times_mono_thm1
\[ \vdash \forall x y z \quad \bullet y \text{dy_less } z \Rightarrow y \text{dy_times } x \text{dy_less } z \text{dy_times } x \]
dy_times_mono_thm2
\[ \vdash \forall x y v w \quad \bullet x \text{dy_less } y \land v \text{dy_less } w \]
\[ \Rightarrow x \text{dy_times } v \text{dy_less } y \text{dy_times } w \]
dy_times_mono⇔_thm
\[ \vdash \forall x y z \quad \bullet x \text{dy_times } y \text{dy_less } x \text{dy_times } z \Leftrightarrow y \text{dy_less } z \]
dy_arch_thm
\[ \vdash \forall x y (\text{dy_one } \text{dy_less } x \Rightarrow (\exists t \bullet y \text{dy_less } x \text{dy_exp } t)) \]
dy_balance_thm1
\[ \vdash \forall x \exists y \bullet \text{dy_one } \text{dy_less } x \text{dy_times } y \]
dy_balance_thm2
\[ \vdash \forall x \exists y \bullet x \text{dy_times } y \text{dy_less } \text{dy_one} \]
dy_right_dense_thm

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
\( \forall x \ y \)

- \( x \ dy\_less \ y \)

\( \Rightarrow (\exists z \)

- \( x \ dy\_less \ x \ dy\_times \ z \)

- \( x \ dy\_times \ z \ dy\_less \ y \)

\text{dy\_less\_dense\_thm}

\( \forall x \ y \bullet x \ dy\_less \ y \Rightarrow (\exists z \bullet x \ dy\_less \ z \ \& \ z \ dy\_less \ y) \)

\text{dy\_left\_dense\_thm}

\( \forall x \ y \)

- \( x \ dy\_less \ y \)

\( \Rightarrow (\exists z \)

- \( x \ dy\_less \ y \ dy\_times \ z \)

- \( y \ dy\_times \ z \ dy\_less \ y \)
9.1.6 The Theory fin_set

9.1.6.1 Parents

\textit{seq}

9.1.6.2 Constants

\begin{align*}
\mathbb{N} & \quad \mathbb{N} \text{ SET} \\
\text{Finite} & \quad 'a \text{ SET SET} \\
\mathbb{F} & \quad 'a \text{ SET } \to 'a \text{ SET SET} \\
\mathbb{F}_1 & \quad 'a \text{ SET } \to 'a \text{ SET SET} \\
\text{Min} & \quad \mathbb{N} \text{ SET } \to \mathbb{N} \\
\text{Max} & \quad \mathbb{N} \text{ SET } \to \mathbb{N} \\
\text{Size} & \quad 'a \text{ SET } \to \mathbb{N} \\
\text{Iter} & \quad \mathbb{N} \rightarrow (\mathbb{N} \times 'a) \text{ SET } \to (\mathbb{N} \times 'a) \text{ SET} \\
\$ & \quad 'a \text{ SET } \to 'b \text{ SET } \to (\mathbb{N} \times 'b) \text{ SET SET} \\
\$ & \quad 'a \text{ SET } \to 'b \text{ SET } \to (\mathbb{N} \times 'b) \text{ SET SET}
\end{align*}

9.1.6.3 Aliases

\# \quad \text{Size : 'a SET } \rightarrow \mathbb{N}

9.1.6.4 Fixity

Right Infix 240:

\begin{align*}
\$ & \quad \rightarrow \\
\$ & \quad \rightarrow \\
\$ & \quad \rightarrow
\end{align*}

9.1.6.5 Definitions

\begin{align*}
\mathbb{N} & \vdash \mathbb{N} = \text{Universe} \\
\text{Finite} & \vdash \text{Finite} = \bigcap \{u|\{\} \in u \land (\forall a \exists x \bullet a \in u \Rightarrow \{x\} \cup a \in u)\} \\
\mathbb{F} & \vdash \forall x \bullet \mathbb{F} x = \mathbb{P} x \cap \text{Finite} \\
\mathbb{F}_1 & \vdash \forall x \bullet \mathbb{F}_1 x = \mathbb{F} x \setminus \{\}\} \\
\text{Min} & \vdash \text{ConstSpec} \\
& \quad (\lambda \text{Min'} \\
& \quad \bullet \forall m a \\
& \quad \quad \bullet m \in a \land (\forall i \bullet i \in a \Rightarrow m \leq i) \Rightarrow \text{Min'} a = m) \\
\text{Min} & \vdash \text{ConstSpec} \\
& \quad (\lambda \text{Max'} \\
& \quad \bullet \forall m a \\
& \quad \quad \bullet m \in a \land (\forall i \bullet i \in a \Rightarrow i \leq m) \Rightarrow \text{Max'} a = m) \\
\text{Max} & \vdash \text{ConstSpec} \\
\text{Size} & \vdash \forall a \\
& \quad \# a = \text{Min} \{i \exists \text{list} \bullet \# \text{list} = i \land \text{Elems list} = a\} \\
\text{Iter} & \vdash \forall r n \\
& \quad \bullet \text{Iter } 0 \ n = \text{Id Universe} \\
& \quad \land \text{Iter } (n + 1) \ r = r \land \text{Iter } n \ r \\
\$ & \vdash \forall a b \bullet a \rightarrow b = (a \rightarrow b) \land \text{Finite} \\
\$ & \vdash \forall a b \bullet a \rightarrow b = (a \rightarrow b) \land (a \rightarrow b)
\end{align*}
9.1.6.6 Theorems

\textbf{Min\_consistent}

\[\vdash \text{Consistent} \quad \left( \lambda \text{Min}' \right)\]
\[\bullet \quad \forall m \ a \quad m \in a \land (\forall i \cdot i \in a \Rightarrow m \leq i) \Rightarrow \text{Min'} a = m\]

\textbf{Max\_consistent}

\[\vdash \text{Consistent} \quad \left( \lambda \text{Max}' \right)\]
\[\bullet \quad \forall m \ a \quad m \in a \land (\forall i \cdot i \in a \Rightarrow i \leq m) \Rightarrow \text{Max'} a = m\]

\textbf{finite\textunderscore induction\textunderscore thm}

\[\vdash \forall p \quad p \{\} \quad \land (\forall a x \cdot a \in \text{Finite} \land p a \land \neg x \in a \Rightarrow p (\{x\} \cup a)) \Rightarrow (\forall a \cdot a \in \text{Finite} \Rightarrow p a)\]

\textbf{empty\textunderscore finite\textunderscore thm}

\[\vdash \{\} \in \text{Finite}\]

\textbf{singleton\cup\textunderscore finite\textunderscore thm}

\[\vdash \forall a b \cdot a \in \text{Finite} \land b \subseteq a \Rightarrow b \in \text{Finite}\]

\textbf{\subseteq\textunderscore finite\textunderscore thm}

\[\vdash \forall a b \cdot a \in \text{Finite} \land b \subseteq a \Rightarrow b \in \text{Finite}\]

\textbf{finite\cup\textunderscore thm}

\[\vdash \forall a b \cdot a \in \text{Finite} \land b \in \text{Finite} \Rightarrow a \in \text{Finite} \land b \in \text{Finite}\]

\textbf{finite\∩\textunderscore thm}

\[\vdash \forall a b \cdot a \in \text{Finite} \lor b \in \text{Finite} \Rightarrow a \cap b \in \text{Finite}\]

\textbf{finite\_distinct\_elems\_thm}

\[\vdash \forall a \quad a \in \text{Finite} \Rightarrow (\exists \text{list} \cdot \text{list} \in \text{Distinct} \land \text{Elems list} = a)\]

\textbf{length\_\textunderscore \textunderscore \leq\textunderscore thm}

\[\vdash \forall \text{list} \ a \cdot \# (\text{list} \mathbin{\omega} a) \leq \# \text{list}\]

\textbf{length\_\textunderscore \textunderscore less\textunderscore thm}

\[\vdash \forall \text{list} \ a \quad \neg \text{Elems list} \mathbin{\omega} a = \{\} \Rightarrow \# (\text{list} \mathbin{\omega} a) < \# \text{list}\]

\textbf{elems\_\textunderscore \textunderscore \leq\textunderscore thm}

\[\vdash \forall \text{list} \ a \quad \text{Elems} (\text{list} \mathbin{\omega} a) = \text{Elems list} \cap a\]

\textbf{distinct\_length\_\textunderscore \leq\textunderscore thm}

\[\vdash \forall \text{list1 list2} \quad \text{list1} \in \text{Distinct} \land \text{Elems list1} = \text{Elems list2} \Rightarrow \# \text{list1} \leq \# \text{list2}\]

\textbf{distinct\_size\_length\_thm}

\[\vdash \forall \text{list} \ a \quad a \in \text{Distinct} \land \text{Elems list} = a \Rightarrow \# a = \# \text{list}\]

\textbf{size\_empty\_thm}

\[\vdash \# \{\} = 0\]

\textbf{size\_singleton\cup\textunderscore thm}

\[\vdash \forall x a \cdot a \in \text{Finite} \land \neg x \in a \Rightarrow \# (\{x\} \cup a) = \# a + 1\]

\textbf{size\_singleton\_thm}

\[\vdash \forall x \cdot \# \{x\} = 1\]

\textbf{size\_\cup\thm}

\[\vdash \forall a b \quad a \in \text{Finite} \land b \in \text{Finite} \Rightarrow \# (a \cup b) + \# (a \cap b) = \# a + \# b\]

\textbf{size\_0\_thm}

\[\vdash \forall a \cdot a \in \text{Finite} \Rightarrow (\# a = 0 \iff a = \{\})\]
size_1_thm
\[ \forall a \bullet a \in \text{Finite} \Rightarrow (\# a = 1 \iff (\exists x \bullet a = \{x\})) \]

∪_finite_thm
\[ \forall u \bullet u \in \text{Finite} \land u \subseteq \text{Finite} \Rightarrow \bigcup u \in \text{Finite} \]

pigeon_hole_thm
\[ \forall u \bullet u \in \text{Finite} \land u \subseteq \text{Finite} \land \# u < \# (\bigcup u) \Rightarrow (\exists a \bullet a \in u \land \# a > 1) \]

⊆_size_≤_thm
\[ \forall a b \bullet a \in \text{Finite} \land b \subseteq a \Rightarrow \# b \leq \# a \]

⊆_size_less_thm
\[ \forall a b \bullet a \in \text{Finite} \land b \subseteq a \Rightarrow \# b < \# a \]

min_in_thm
\[ \forall n a \bullet n \in a \Rightarrow \text{Min} a \in a \]

min_≤_thm
\[ \forall n a \bullet n \in a \Rightarrow \text{Min} a \leq n \]

max_in_thm
\[ \forall m n a \bullet (\forall i \bullet i \in a \Rightarrow i \leq m) \land n \in a \Rightarrow \text{Max} a \in a \]

≤_max_thm
\[ \forall m n a \bullet (\forall i \bullet i \in a \Rightarrow i \leq m) \land n \in a \Rightarrow n \leq \text{Max} a \]

finite_⊆_well_founded_thm
\[ \forall p a \bullet a \in \text{Finite} \land p a \Rightarrow (\exists b \bullet b \subseteq a \land p b \land (\forall c \bullet c \subseteq b \land p c \Rightarrow c = b)) \]
9.1.7 The Theory fun\_rel

9.1.7.1 Parents

bin\_rel

9.1.7.2 Children

fun\_rel\_thms  seq

9.1.7.3 Constants

\$
\xrightarrow{}
'a \text{ SET} \rightarrow 'b \text{ SET} \rightarrow ('a \times 'b) \text{ SET SET}
$

\$
\xrightarrow{}
'a \text{ SET} \rightarrow 'b \text{ SET} \rightarrow ('a \times 'b) \text{ SET SET}
$

\$
\xrightarrow{}
'a \text{ SET} \rightarrow 'b \text{ SET} \rightarrow ('a \times 'b) \text{ SET SET}
$

\$
\xrightarrow{}
'a \text{ SET} \rightarrow 'b \text{ SET} \rightarrow ('a \times 'b) \text{ SET SET}
$

\$
\xrightarrow{}
'a \text{ SET} \rightarrow 'b \text{ SET} \rightarrow ('a \times 'b) \text{ SET SET}
$

\$
\xrightarrow{}
'a \text{ SET} \rightarrow 'b \text{ SET} \rightarrow ('a \times 'b) \text{ SET SET}
$

\$
\xrightarrow{}
'a \text{ SET} \rightarrow 'b \text{ SET} \rightarrow ('a \times 'b) \text{ SET SET}
$

\$
\xrightarrow{}
'a \text{ SET} \rightarrow 'b \text{ SET} \rightarrow ('a \times 'b) \text{ SET SET}
$

\$
\xrightarrow{}
(a \times 'b) \text{ SET} \rightarrow 'a \rightarrow 'b
$

9.1.7.4 Aliases

\$
\xrightarrow{}\xrightarrow{}
(a \times 'b) \text{ SET} \rightarrow 'a \rightarrow 'b
$

9.1.7.5 Fixity

Left Infix 600:

At

Right Infix 240:

\xrightarrow{}\xrightarrow{}\xrightarrow{}\xrightarrow{}\xrightarrow{}\xrightarrow{}

9.1.7.6 Definitions

\$
\xrightarrow{}
\forall \ a \ b \bullet \ a \rightarrow b = (a \leftrightarrow b) \cap \text{ Functional}
$

\$
\rightarrow
\forall \ a \ b \bullet (a \rightarrow b) = (a \leftrightarrow b) \cap \text{ Functional} \cap \text{ Total} \ a
$

\$
\xrightarrow{}
\forall \ a \ b \bullet \ a \leftrightarrow b = (a \rightarrow b) \cap \text{ Functional} \cap \text{ Injective}
$

\$
\xrightarrow{}
\forall \ a \ b

\bullet a \rightarrow b = (a \leftrightarrow b) \cap \text{ Functional} \cap \text{ Injective} \cap \text{ Total} \ a
$

\$
\rightarrow
\forall \ a \ b \bullet \ a \leftrightarrow b = (a \rightarrow b) \cap \text{ Functional} \cap \text{ Surjective} \ b
$

\$
\rightarrow
\forall \ a \ b

\bullet a \rightarrow b

= (a \leftrightarrow b) \cap \text{ Functional} \cap \text{ Surjective} \ b \cap \text{ Total} \ a
$

\$
\xrightarrow{}
\forall \ a \ b

\bullet a \leftrightarrow b

= (a \leftrightarrow b)

\cap \text{ Functional}

\cap \text{ Injective}

\cap \text{ Surjective} \ b

\cap \text{ Total} \ a
$

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUAL USR029
At \[
\vdash \text{ConstSpec} \\
\quad \lambda \text{At}' \\
\quad \bullet \forall f \ x \ y \\
\quad \bullet (x, y) \in f \land (\forall z \bullet (x, z) \in f \Rightarrow z = y) \\
\quad \Rightarrow \text{At}' f \ x = y
\]

9.1.7.7 Theorems

At\_consistent
\[
\vdash \text{Consistent} \\
\quad \lambda \text{At}' \\
\quad \bullet \forall f \ x \ y \\
\quad \bullet (x, y) \in f \land (\forall z \bullet (x, z) \in f \Rightarrow z = y) \\
\quad \Rightarrow \text{At}' f \ x = y
\]

Graph\_at\_thm \[
\vdash \forall f \ x \bullet \text{Graph } f \circ x = f \ x
\]

Inv\_rel\_in\_arrow\_thm \[
\vdash \forall f \ a \ b \\
\quad \bullet (f \sim \in b \leftrightarrow a \leftrightarrow f \in a \leftrightarrow b) \\
\quad \land (f \sim \in b \leftrightarrow a \leftrightarrow f \in (a \leftrightarrow b) \cap \text{Injective}) \\
\quad \land (f \sim \in (b \rightarrow a) \\
\quad \leftrightarrow f \in (a \leftrightarrow b) \cap \text{Injective} \cap \text{Surjective} b) \\
\quad \land (f \sim \in b \rightarrow a \\
\quad \leftrightarrow f \in (a \leftrightarrow b) \cap \text{Injective} \cap \text{Functional}) \\
\quad \land (f \sim \in b \rightarrow a \\
\quad \leftrightarrow f \in (a \leftrightarrow b) \cap \text{Injective} \cap \text{Total} a) \\
\quad \land (f \sim \in b \rightarrow a \\
\quad \leftrightarrow f \\
\quad \in (a \leftrightarrow b) \\
\quad \cap \text{Injective} \\
\quad \cap \text{Total} a \\
\quad \cap \text{Surjective} b) \\
\quad \land (f \sim \in b \rightarrow a \\
\quad \leftrightarrow f \\
\quad \in (a \leftrightarrow b) \\
\quad \cap \text{Injective} \\
\quad \cap \text{Functional} \\
\quad \cap \text{Surjective} b \\
\quad \cap \text{Total} a)
\]

At\_at\_eq\_thm \[
\vdash \forall f \ X \ Y \ x \ y \\
\quad \bullet f \in X \rightarrow Y \land x \in \text{Dom } f \land y \in \text{Dom } f \\
\quad \Rightarrow (f \circ x = f \circ y \\
\quad \leftrightarrow (\exists z \bullet (x, z) \in f \land (y, z) \in f))
\]
9.1.8 The Theory fun_rel_thms

9.1.8.1 Parents

\textit{set_thms} \quad \textit{fun_rel}

9.1.8.2 Constants

\textbf{ReflexiveIn} \quad \text{'a SET} → \text{('a × 'a) SET SET}
\textbf{Antisymmetric} \quad \text{('a × 'a) SET SET}
\textbf{Chains} \quad \text{('a × 'a) SET} → \text{'a SET SET}
\textbf{UpperBounds} \quad \text{('a × 'a) SET} → \text{'a SET} → \text{'a SET}
\textbf{MaximalElements} \quad \text{('a × 'a) SET} → \text{'a SET} → \text{'a SET}

9.1.8.3 Definitions

\textbf{ReflexiveIn} \quad \vdash \forall a \bullet \text{ReflexiveIn } a = \{r|\forall x \bullet x \in a \Rightarrow (x, x) \in r\}
\textbf{Antisymmetric} \quad \vdash \text{Antisymmetric} = \{r|\forall x \bullet (x, y) \in r \land (y, x) \in r \Rightarrow x = y\}
\textbf{Chains} \quad \vdash \forall r \bullet \text{Chains } r = \{a|\forall x \bullet x \in a \land x \in a \Rightarrow (x, y) \in r \lor (y, x) \in r\}
\textbf{UpperBounds} \quad \vdash \forall r \bullet \text{UpperBounds } r a = \{x|\forall y \bullet y \in a \Rightarrow (y, x) \in r\}
\textbf{MaximalElements} \quad \vdash \forall r \bullet \text{MaximalElements } r a = \{x|\forall y \bullet y \in a \land (x, y) \in r \Rightarrow y = x\}

9.1.8.4 Theorems

\textbf{zorn_thm} \quad \vdash \forall r \bullet \text{zorn_thm}\begin{align*}
&\bullet r \in \text{ReflexiveIn } a \\
&\land r \in \text{Transitive} \\
&\land r \in \text{Antisymmetric} \\
&\land (\forall c \\
&\bullet c \subseteq a \land c \in \text{Chains } r \\
&\Rightarrow (\exists x \bullet x \in a \land x \in \text{UpperBounds } r c)) \\
&\Rightarrow (\exists x \bullet x \in \text{MaximalElements } r a)
\end{align*}
\textbf{comparability_thm} \quad \vdash \forall a \bullet (f: f \in a \Rightarrow b) \lor (g: g \in b \Rightarrow a)
\textbf{schroeder_bernstein_thm1} \quad \vdash \forall a \bullet c \in b \land c \subseteq a \land c \in \text{Chains } r \\
\Rightarrow (\exists h \bullet h \in a \Rightarrow b)
\textbf{schroeder_bernstein_thm} \quad \vdash \forall a \bullet b \land f \in a \Rightarrow b \land g \in b \Rightarrow a \\
\Rightarrow (\exists h \bullet h \in a \Rightarrow b)
9.1.9 The Theory hol

9.1.9.1 Parents

\[ \text{sets sum one} \]

9.1.9.2 Children

\[ \text{bin\_rel} \]
9.1.10 The Theory init

9.1.10.1 Parents

log

9.1.10.2 Children

misc

9.1.10.3 Axioms

bool_cases_axiom
\[ \vdash \forall \mathit{b} \cdot (\mathit{b} \Leftrightarrow \mathit{T}) \lor (\mathit{b} \Leftrightarrow \mathit{F}) \]

⇒_antisym_axiom
\[ \vdash \forall \mathit{b1} \ \mathit{b2} \cdot (\mathit{b1} \Rightarrow \mathit{b2}) \Rightarrow (\mathit{b2} \Rightarrow \mathit{b1}) \Rightarrow (\mathit{b1} \Leftrightarrow \mathit{b2}) \]

η_axiom
\[ \vdash \forall \mathit{f} \cdot (\lambda \mathit{x} \cdot \mathit{f} \ \mathit{x}) = \mathit{f} \]

ε_axiom
\[ \vdash \forall \mathit{P} \ \mathit{x} \cdot \mathit{P} \ \mathit{x} \Rightarrow \mathit{P} (\$\mathit{\epsilon} \ \mathit{P}) \]

infinity_axiom
\[ \vdash \exists \mathit{f} \cdot \text{OneOne} \ \mathit{f} \land \neg \text{Onto} \ \mathit{f} \]
9.1. Theory Listings

9.1.11 The Theory list

9.1.11.1 Parents

\[ \mathbb{N} \]

9.1.11.2 Children

\( \text{char} \)

9.1.11.3 Constants

\begin{align*}
\text{IsListRep} & : (\mathbb{N} \rightarrow '\text{a}) \times \mathbb{N} \rightarrow \text{BOOL} \\
\text{Cons} & : '\text{a} \rightarrow '\text{a} \text{ LIST} \rightarrow '\text{a} \text{ LIST} \\
\text{[]} & : '\text{a} \text{ LIST} \\
\text{Length} & : '\text{a} \text{ LIST} \rightarrow \mathbb{N} \\
\text{Hd} & : '\text{a} \text{ LIST} \rightarrow '\text{a} \\
\text{Tl} & : '\text{a} \text{ LIST} \rightarrow '\text{a} \text{ LIST} \\
\text{Append} & : '\text{a} \text{ LIST} \rightarrow '\text{a} \text{ LIST} \rightarrow '\text{a} \text{ LIST} \\
\text{Rev} & : '\text{a} \text{ LIST} \rightarrow '\text{a} \text{ LIST} \\
\text{Split} & : ((\text{a} \times \text{b}) \text{ LIST} \rightarrow '\text{a} \text{ LIST} \times \text{b} \text{ LIST} \\
\text{Combine} & : '\text{a} \text{ LIST} \rightarrow \text{b} \text{ LIST} \rightarrow ((\text{a} \times \text{b}) \text{ LIST} \\
\text{Map} & : ((\text{a} \rightarrow \text{b}) \rightarrow '\text{a} \text{ LIST} \rightarrow \text{b} \text{ LIST} \\
\text{Fold} & : ((\text{a} \rightarrow \text{b} \rightarrow \text{b}) \rightarrow '\text{a} \text{ LIST} \rightarrow \text{b} \rightarrow \text{b}
\end{align*}

9.1.11.4 Aliases

\( \text{@} \)

\text{Append} : '\text{a} \text{ LIST} \rightarrow '\text{a} \text{ LIST} \rightarrow '\text{a} \text{ LIST}

9.1.11.5 Types

\( '\text{t} \text{ LIST} \)

9.1.11.6 Fixity

\text{Right Infix 300:}

\( \text{@} \)
9.1.11.7 Definitions

\textbf{IsListRep}

\textbf{is\_list\_rep\_def}

\[ \exists \text{nil cons} \]

\begin{itemize}
  \item \text{IsListRep nil}
  \[ \land (\forall x. \text{IsListRep} x \Rightarrow (\forall h. \neg \text{cons} h x = \text{nil})) \]
  \[ \land (\forall x y. \text{IsListRep} x \land \text{IsListRep} y \Rightarrow (\forall a b. \text{cons} a x = \text{cons} b y \Leftrightarrow a = b \land x = y)) \]
  \[ \land (\forall p. \neg \text{nil} \land (\forall m. p m \Rightarrow (\forall h. p (\text{cons} h m))) \Rightarrow (\forall n. \text{IsListRep} n \Rightarrow p n)) \]
\end{itemize}

\textbf{LIST}

\textbf{list\_def}

\[ \exists f. \text{TypeDefn IsListRep f} \]

\textbf{Nil}

\textbf{Cons}

\textbf{nil\_cons\_def}

\[ (\forall x. \neg \text{Cons} x \text{ list} = []) \]

\[ \land (\forall x1 x2. \text{list}1 \text{ list}2 \Rightarrow x1 = x2 \land \text{list}1 = \text{list}2) \]

\[ \land (\forall p. \neg \text{nil} \land (\forall \text{list}. p \text{ list} \Rightarrow (\forall x. p (\text{cons} x \text{ list}))) \Rightarrow (\forall \text{list}. p \text{ list})) \]

\textbf{Length}

\textbf{length\_def}

\[ \text{Length} [] = 0 \]

\[ \land (\forall h. \text{list} \Rightarrow \text{Length} (\text{Cons} h \text{ list}) = \text{Length} \text{ list} + 1) \]

\textbf{Hd}

\textbf{hd\_def}

\[ \forall h. \text{list} \Rightarrow \text{Hd} (\text{Cons} h \text{ list}) = h \]

\textbf{Tl}

\textbf{tl\_def}

\[ \forall h. \text{list} \Rightarrow \text{Tl} (\text{Cons} h \text{ list}) = \text{list} \]

\textbf{Append}

\textbf{append\_def}

\[ (\forall \text{list}. [] @ \text{list} = \text{list}) \]

\[ \land (\forall h. \text{list}' \Rightarrow \text{Cons} h \text{ list} @ \text{list}' = \text{Cons} h (\text{list} @ \text{list}')) \]

\textbf{Rev}

\textbf{rev\_def}

\[ \text{Rev} [] = [] \]

\[ \land (\forall h. \text{list} \Rightarrow \text{Rev} (\text{Cons} h \text{ list}) = \text{Rev} \text{ list} @ [h]) \]

\textbf{Split}

\textbf{split\_def}

\[ \text{Split} [] = ([], []) \]

\[ \land (\forall \text{list} h1 h2. \text{Split} (\text{Cons} (h1, h2) \text{ list}) = (\text{Cons} h1 (\text{Fst} (\text{Split} \text{ list})), \text{Cons} h2 (\text{Snd} (\text{Split} \text{ list})))) \]

\textbf{Combine}

\textbf{combine\_def}

\[ \text{Combine} [] [] = [] \]

\[ \land (\forall h1 h2. \text{list}1 \text{ list}2 \Rightarrow \text{Combine} (\text{Cons} h1 \text{ list}1) (\text{Cons} h2 \text{ list}2) \]
= Cons (h1, h2) (Combine list1 list2))

Map
def \map \def \begin{array}{l}
\vdash (\forall g \bullet \map g \emptyset = \emptyset) \\
\land (\forall h \ g \ list \bullet \map g (\Cons h \ list) = \Cons (g \ h) \ (\map g \ list))
\end{array}

Fold
def \fold \def \begin{array}{l}
\vdash (\forall g \ x \bullet \fold g \emptyset x = x) \\
\land (\forall h \ g \ x \ list \bullet \fold g (\Cons h \ list) x = g \ h \ (\fold g \ list \ x))
\end{array}

9.1.11.8 Theorems

list_induction_thm
\begin{array}{l}
\vdash \forall p \bullet p \emptyset \land (\forall \ list \bullet p \ list \Rightarrow (\forall x \bullet p \ (\Cons x \ list))) \\
\Rightarrow (\forall \ list \bullet p \ list)
\end{array}

list_prim_rec_thm
\begin{array}{l}
\vdash \forall n c \bullet \exists f \bullet f \emptyset = n \\
\land (\forall \ list \ a \bullet f (\Cons a \ list) = c \ a \ (f \ list \ a))
\end{array}

list_clauses
\begin{array}{l}
\vdash \forall x1 \ x2 \ list1 \ list2 \bullet \neg \Cons x1 \ list1 = \emptyset \\
\land \neg \emptyset = \Cons x1 \ list1 \\
\land (\Cons x1 \ list1 = \Cons x2 \ list2 \\
\iff x1 = x2 \land list1 = list2) \\
\land \Hd \ (\Cons x1 \ list1) = x1 \\
\land \Tl \ (\Cons x1 \ list1) = list1
\end{array}

list_cases_thm
\begin{array}{l}
\vdash \forall list1 \bullet list1 = \emptyset \lor (\exists x \ list2 \bullet list1 = \Cons x \ list2)
\end{array}
9.1.12 The Theory log

9.1.12.1 Parents

\( \text{min} \)

9.1.12.2 Children

\( \text{init} \)

9.1.12.3 Constants

\[ \begin{align*}
T & \quad \text{BOOL} \\
\forall & \quad \left( a \to \text{BOOL} \right) \to \text{BOOL} \\
\exists & \quad \left( a \to \text{BOOL} \right) \to \text{BOOL} \\
\text{F} & \quad \text{BOOL} \\
\neg & \quad \text{BOOL} \to \text{BOOL} \\
\land & \quad \text{BOOL} \to \text{BOOL} \to \text{BOOL} \\
\lor & \quad \text{BOOL} \to \text{BOOL} \to \text{BOOL} \\
\text{OneOne} & \quad \left( a \to 'b \right) \to \text{BOOL} \\
\text{Onto} & \quad \left( a \to 'b \right) \to \text{BOOL} \\
\text{TypeDefn} & \quad \left( 'b \to \text{BOOL} \right) \to \left( a \to 'b \right) \to \text{BOOL}
\end{align*} \]

9.1.12.4 Fixity

Binder: \( \forall \quad \exists \)

Right Infix 30:

\( \lor \)

Right Infix 40:

\( \land \)

Prefix 50:

\( \neg \)

9.1.12.5 Terminators

\( \lor \quad \land \quad \neg \quad \exists \quad \forall \)

9.1.12.6 Definitions

\[ \begin{align*}
T & \quad \vdash T \leftrightarrow (\lambda x \bullet x) = (\lambda x \bullet x) \\
\forall & \quad \vdash \forall \psi = (\lambda P \bullet P = (\lambda x \bullet T)) \\
\exists & \quad \vdash \exists \psi = (\lambda P \bullet P (\psi \bullet P)) \\
\text{F} & \quad \vdash F \leftrightarrow (\forall b \bullet b) \\
\neg & \quad \vdash \neg = (\lambda b \bullet b \Rightarrow F) \\
\land & \quad \vdash \land = (\lambda b1 \ b2 \bullet \forall b \bullet (b1 \Rightarrow b2 \Rightarrow b) \Rightarrow b)
\end{align*} \]
∀
∀_def ⊢ $∀ = (\lambda \ b1 \ b2 \cdot \forall \ b\cdot (b1 \Rightarrow b) \Rightarrow (b2 \Rightarrow b) \Rightarrow b)$

OneOne
One_one_def ⊢ OneOne = (\lambda f\cdot \forall x1 \ x2\cdot f \ x1 = f \ x2 \Rightarrow x1 = x2)

Onto
onto_def ⊢ Onto = (\lambda f\cdot \forall y\cdot \exists x\cdot y = f \ x)

TypeDefn
type_defn_def ⊢ Typedefn
= (\lambda P \ rep
  \bullet OneOne \ rep \land (\forall x\cdot P \ x \Leftrightarrow (\exists y\cdot x = \text{rep} \ y)))
9.1.13 The Theory min

9.1.13.1 Children

\[ \log \]

9.1.13.2 Constants

\[
\begin{align*}
\& \Rightarrow & \quad BOOL \to BOOL \to BOOL \\
\& = & \quad 'a \to 'a \to BOOL \\
\& \epsilon & \quad ('a \to BOOL) \to 'a 
\end{align*}
\]

9.1.13.3 Types

\[
'1 \to '2 \\
BOOL \\
IND
\]

9.1.13.4 Fixity

\[
\begin{align*}
\textbf{Binder:} & \quad \epsilon \quad \lambda \\
\textbf{Right Infix 20:} \quad \Rightarrow \\
\textbf{Right Infix 100:} \quad \to \\
\textbf{Right Infix 200:} \quad =
\end{align*}
\]

9.1.13.5 Terminators

\[
\epsilon \quad \lambda \quad = \quad \Rightarrow \quad \to
\]
9.1.14  The Theory misc

9.1.14.1  Parents

\textit{init}

9.1.14.2  Children

\textit{pair}

9.1.14.3  Constants

- Arbitrary: \(\text{'}a\)
- $\exists_1$ (\(\text{'}a \rightarrow \text{BOOL} \rightarrow \text{BOOL}\))
- Let: (\(\text{'}a \rightarrow \text{'}b\) \(\rightarrow \text{'}a \rightarrow \text{'}b\))
- Cond: (\(\text{BOOL} \rightarrow \text{'}a \rightarrow \text{'}a \rightarrow \text{'}a\))
- pp'\text{TS}: (\(\text{BOOL} \rightarrow \text{BOOL}\))

9.1.14.4  Aliases

\(\Leftrightarrow\) (\(\$= : \text{BOOL} \rightarrow \text{BOOL} \rightarrow \text{BOOL}\))

9.1.14.5  Fixity

- Binder: \(\exists_1\)
- Right Infix 10: \(\Leftrightarrow\)

9.1.14.6  Terminators

\(\Leftrightarrow\) (\(\exists_i\))

9.1.14.7  Definitions

- $\exists_1$
- $\exists_{1\_def}$ (\(\vdash \$\exists_1 = (\lambda \ P \bullet \ \exists \ t \bullet \ P \ t \ \wedge \ (\forall \ x \bullet \ P \ x \ \Rightarrow \ x = t)\))
- Let (\(\vdash \text{Let} = (\lambda \ f \ x \bullet \ f \ x)\))
- Cond (\(\vdash \text{Cond} = (\lambda \ b \ x1 \ x2 \bullet \ \epsilon \ x \bullet ((b \ Leftrightarrow \ T) \ \Rightarrow \ x = x1) \ \wedge \ ((b \ Leftrightarrow \ F) \ \Rightarrow \ x = x2))\))
- pp'\text{TS} (\(\vdash \forall \ x \bullet \ pp'\text{TS} \ x \ Leftrightarrow x\))

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
9.1.14.8 Theorems

\[ T \]

\[ \forall t1 \ t2 \cdot t1 \lor t2 \iff (\forall b \cdot (t1 \Rightarrow b) \Rightarrow (t2 \Rightarrow b) \Rightarrow b) \]

\[ \neg \exists t \; \neg t \iff t \Rightarrow F \]

\[ \neg \exists t \; \neg t \iff t \Rightarrow F \]

\[ \forall P \ x \cdot P \ x \Rightarrow \exists P \]

\[ \forall a \ t1 \ t2 \cdot (if \ a \ then \ t1 \ else t2) \Rightarrow (if \ a \ then \ \neg \neg b \ else \ \neg \neg c) \]

\[ \forall a \ b \ c \cdot (if \ a \ then \ b \ else \ c) \Rightarrow (if \ a \ then \ \neg \ b \ else \ \neg \ c) \]

\[ \forall x \cdot x = x \iff T \]

\[ \forall a \ b \cdot \neg \neg t \iff t \lor \neg \neg t \iff t \lor \neg \neg t \iff t \]

\[ \forall t \cdot (T \land t \iff t) \lor (t \land T \iff t) \lor (F \land t \iff \neg t) \lor (t \land F \iff \neg t) \]

\[ \forall t \cdot (T \lor t \iff t) \lor (t \lor T \iff t) \lor (F \lor t \iff \neg t) \lor (t \lor F \iff \neg t) \lor (t \lor t \iff t) \]

\[ \forall a \ b \ c \cdot (if \ a \ then \ b \ else \ c) \Rightarrow (if \ a \ then \ \neg \ b \ else \ \neg \ c) \]

\[ \exists \intro \thm \iff T \]

\[ \forall \exists \thm \iff T \lor \exists \thm \iff T \land \exists \thm \iff T \land \exists \thm \iff T \lor \exists \thm \iff T \lor \exists \thm \iff T \]

\[ \forall a \ b \ c \cdot (if \ a \ then \ b \ else \ c) \Rightarrow (if \ a \ then \ \neg \ b \ else \ \neg \ c) \]

\[ \forall \exists \thm \iff T \]

\[ \forall \exists \thm \iff T \lor \exists \thm \iff T \land \exists \thm \iff T \land \exists \thm \iff T \lor \exists \thm \iff T \lor \exists \thm \iff T \]

\[ \forall a \ b \ c \cdot (if \ a \ then \ b \ else \ c) \Rightarrow (if \ a \ then \ \neg \ b \ else \ \neg \ c) \]
\( \vdash \forall t \\
\quad \bullet (T \Rightarrow t \iff t) \\
\quad \land (F \Rightarrow t \iff T) \\
\quad \land (t \Rightarrow T \iff T) \\
\quad \land (t \Rightarrow t \iff T) \\
\quad \land (t \Rightarrow F \iff \neg t) \)

*if_rewrite_thm*

\( \vdash \forall t_1 \ t_2 \\
\quad \bullet (if \ T \ then \ t_1 \ else \ t_2) = t_1 \\
\quad \land (if \ F \ then \ t_1 \ else \ t_2) = t_2 \)

*∀_rewrite_thm*

\( \vdash \forall t \bullet (\forall \ x \bullet t) \iff t \)

*∃_rewrite_thm*

\( \vdash \forall t \bullet (\exists \ x \bullet t) \iff t \)

*β_rewrite_thm*

\( \vdash \forall t_1 \ t_2 \bullet (\lambda \ x \bullet t_1) t_2 = t_1 \)

*one_one_thm*

\( \vdash \forall f \bullet OneOne f \iff (\forall \ x \ y \bullet f \ x = f \ y \iff x = y) \)

*ext_thm*

\( \vdash \forall f \ g \bullet f \ = \ g \iff (\forall \ x \bullet f \ x = g \ x) \)

*type_lemmas_thm*

\( \vdash \forall \ pred \\
\quad \bullet (\exists f \bullet TypeDefn \ pred \ f) \\
\quad \Rightarrow (\exists abs \ rep \\
\quad \quad \bullet (\forall \ a \bullet abs \ (rep \ a) = a) \\
\quad \quad \land (\forall \ r \bullet pred \ r \iff rep \ (abs \ r) = r)) \)

*fun_rel_thm*

\( \vdash \forall r \\
\quad \bullet (\exists f \bullet \forall \ x \ y \bullet f \ x = y \iff r \ x \ y) \\
\quad \iff (\forall \ x \bullet \exists y \bullet r \ x \ y \land (\forall \ z \bullet r \ x \ z \Rightarrow z = y)) \)
9.1.15 The Theory one

9.1.15.1 Parents

basic_hol

9.1.15.2 Children

hol

9.1.15.3 Constants

IsOneRep \( \text{BOOL} \rightarrow \text{BOOL} \)
One \( \text{ONE} \)

9.1.15.4 Types

ONE

9.1.15.5 Definitions

IsOneRep
is_one_rep_def
\( \vdash \exists \text{one} \cdot \forall x \cdot \text{IsOneRep } x \leftrightarrow x \leftrightarrow \text{one} \)
One
one_def
\( \vdash \exists f \cdot \text{TypeDefn } \text{IsOneRep } f \)

9.1.15.6 Theorems

one_fns_thm
\( \vdash \forall f \cdot f = (\lambda x \cdot \text{One}) \)
9.1.16 The Theory orders

9.1.16.1 Parents

sets

9.1.16.2 Children

\( \mathbb{R} \)

9.1.16.3 Constants

Irrefl \( \forall X \quad \text{Irrefl} \quad \forall x \in X \Rightarrow \neg x \ll x \)

Antisym \( \forall X \quad \text{Antisym} \quad (\forall x y \quad x \ll y \Rightarrow \neg y \ll x) \)

Trans \( \forall X \quad \text{Trans} \quad (\forall x y z \quad x \ll y \land y \ll z \Rightarrow x \ll z) \)

9.1.16.4 Fixity

Right Infix 210:

DenseIn HasSupremum \( \ll \) \( \ll \ll \)

9.1.16.5 Definitions

Irrefl \( \forall X \quad \text{Irrefl} \quad \forall x \in X \Rightarrow \neg x \ll x \)

Antisym \( \forall X \quad \text{Antisym} \quad (\forall x y \quad x \ll y \Rightarrow \neg y \ll x) \)

Trans \( \forall X \quad \text{Trans} \quad (\forall x y z \quad x \ll y \land y \ll z \Rightarrow x \ll z) \)
\begin{align*}
\text{StrictPartialOrder} & \vdash \forall X \ b \\
& \bullet \text{StrictPartialOrder} (X, b) \\
& \quad \Leftrightarrow \text{Irefl} (X, b) \\
& \quad \land \text{Antisym} (X, b) \\
& \quad \land \text{Trans} (X, b) \\
\text{Trich} & \vdash \forall X \ b \\
& \bullet \text{Trich} (X, b) \\
& \quad \Leftrightarrow (\forall x y \cdot x \in X \land y \in X \land \neg x = y \Rightarrow x b y \lor y b x) \\
\text{StrictLinearOrder} & \vdash \forall X \ b \\
& \bullet \text{StrictLinearOrder} (X, b) \\
& \quad \Leftrightarrow \text{StrictPartialOrder} (X, b) \land \text{Trich} (X, b) \\
\text{DenseIn} & \vdash \forall A X \ b \\
& \bullet \text{DenseIn} (X, b) \\
& \quad \Leftrightarrow (\forall x y \cdot x \in X \land y \in X \land x b y \\
& \quad \quad \Rightarrow (\exists a \cdot a \in A \land x b a \land a b y)) \\
\text{Dense} & \vdash \forall X \ b \\
& \bullet \text{Dense} (X, b) \\
& \quad \Rightarrow \text{X DenseIn} (X, b) \\
\text{UpperBound} & \vdash \forall A \ b \ x \ X \\
& \bullet \text{UpperBound} (A, b, x) \\
& \quad \Leftrightarrow (\forall y \cdot y \in X \land \text{UpperBound} (A, b, y) \Rightarrow \neg y b x) \\
\text{HasSupremum} & \vdash \forall A \ b \\
& \bullet \text{HasSupremum} (x, X, b) \\
& \quad \Leftrightarrow \text{UpperBound} (A, b, x) \\
& \quad \land (\forall y \cdot y \in X \land \text{UpperBound} (A, b, y) \Rightarrow \neg y b x) \\
\text{UnboundedAbove} & \vdash \forall A \ b \\
& \bullet \text{UnboundedAbove} (A, b) \\
& \quad \Leftrightarrow (\forall b \cdot b \in A \Rightarrow (\exists c \cdot c \in A \land b c b)) \\
\text{UnboundedBelow} & \vdash \forall A \ b \\
& \bullet \text{UnboundedBelow} (A, b) \\
& \quad \Leftrightarrow (\forall b \cdot b \in A \Rightarrow (\exists c \cdot c \in A \land c b b)) \\
\text{Complete} & \vdash \forall X \ b \\
& \bullet \text{Complete} (X, b) \\
& \quad \Leftrightarrow (\forall A \ x \\
& \quad \quad \bullet \neg A = \{\} \\
& \quad \quad \land A \subseteq X \\
& \quad \quad \land \text{UnboundedAbove} (A, b) \\
& \quad \quad \land x \in X \\
& \quad \quad \land \text{UpperBound} (A, b, x) \\
& \quad \quad \Rightarrow (\exists y \cdot y \in X \land A \text{HasSupremum} (y, X, b))) \\
\text{Cuts} & \vdash \forall X \ b \ A \\
& \bullet A \in \text{Cuts} (X, b) \\
& \quad \Leftrightarrow A \subseteq X \\
& \quad \land \neg A = \{\} \\
& \quad \land \text{UnboundedAbove} (A, b) \\
& \quad \land (\exists x \cdot x \in X \land \text{UpperBound} (A, b, x)) \\
& \quad \land (\forall a b \cdot a \in A \land b \in X \land b b a \Rightarrow b \in A) 
\end{align*}
9.1. Theory Listings

\[ \text{DownSet} \quad \vdash \forall X. \forall x. \text{DownSet} \left( X, y \right) = \{ y | y \in X \land y \ll x \} \]

\[ \text{DownSets} \quad \vdash \forall X. \forall s. \text{DownSets} \left( X, s \right) \]

\[ \bullet \ s \in \text{DownSets} \left( X, s \right) \]

\[ \Leftrightarrow ( \exists x. x \in A \land s = \text{DownSet} \left( X, s \right) ) \]

9.1.16.6 Theorems

\[ \text{irrefl_thm} \quad \vdash \forall V. \text{Irrefl} \left( V, \subset \right) \]

\[ \text{antisym_thm} \quad \vdash \forall V. \text{Antisym} \left( V, \subset \right) \]

\[ \text{trans_thm} \quad \vdash \forall V. \text{Trans} \left( V, \subset \right) \]

\[ \text{cuts_trich_thm} \quad \vdash \forall X. \forall \cdot \text{Trich} \left( X, \ll \right) \]

\[ \Rightarrow \text{Trich} \left( \text{Cuts} \left( X, \ll \right), \subset \right) \]

\[ \text{cuts_strict_partial_order_thm} \quad \vdash \forall X. \forall \cdot \text{StrictPartialOrder} \left( \text{Cuts} \left( X, \ll \right), \subset \right) \]

\[ \text{cuts_strict_linear_order_thm} \quad \vdash \forall X. \forall \cdot \text{StrictLinearOrder} \left( X, \ll \right) \]

\[ \Rightarrow \text{StrictLinearOrder} \left( \text{Cuts} \left( X, \ll \right), \subset \right) \]

\[ \text{cuts_complete_thm} \quad \vdash \forall X. \forall \cdot \text{Complete} \left( \text{Cuts} \left( X, \ll \right), \subset \right) \]

\[ \text{down_sets_cuts_thm} \quad \vdash \forall X. \forall \cdot \text{DownSets} \left( X, \ll, A \right) \subseteq \text{Cuts} \left( X, \ll \right) \]

\[ \text{down_sets_dense_thm} \quad \vdash \forall X. \forall \cdot \text{DownSets} \left( X, \ll, A \right) \subseteq \text{DenseIn} \left( \text{Cuts} \left( X, \ll \right), \subset \right) \]

\[ \text{dense_superset_thm} \quad \vdash \forall X. \forall \cdot \text{B} \subseteq \text{A} \land \text{A DenseIn} \left( X, \ll \right) \]

\[ \Rightarrow \text{B DenseIn} \left( X, \ll \right) \]

\[ \text{dense_universe_thm} \quad \vdash \forall X. \forall \cdot \text{A DenseIn} \left( X, \ll \right) \Rightarrow \text{Dense} \left( X, \ll \right) \]

\[ \text{downset_cut_thm} \quad \vdash \forall X. \forall \cdot \text{A DenseIn} \left( X, \ll \right) \Rightarrow \text{Dense} \left( X, \ll \right) \]

\[ \text{trans} \left( X, \ll \right) \]

\[ \Rightarrow \text{UnboundedBelow} \left( X, \ll \right) \]
\[ \forall X \text{ DenseIn} (X, \langle \cdot, \cdot \rangle) \Rightarrow \text{DownSet} (X, \langle \cdot, \cdot \rangle, a) \in \text{Cuts} (X, \langle \cdot, \cdot \rangle) \]

down_sets_less_thm
\[ \vdash \forall X \langle \cdot, \cdot \rangle \ a \ b \]
\[ \bullet \ a \in X \land b \in X \land \text{StrictLinearOrder} (X, \langle \cdot, \cdot \rangle) \]
\[ \Rightarrow (\text{DownSet} (X, \langle \cdot, \cdot \rangle, a) \subset \text{DownSet} (X, \langle \cdot, \cdot \rangle, b) \]
\[ \simeq a \langle \cdot, \cdot \rangle b \]

cuts_unbounded_above_thm
\[ \vdash \forall X \langle \cdot, \cdot \rangle \]
\[ \bullet \text{Irrefl} (X, \langle \cdot, \cdot \rangle) \]
\[ \land \text{Trans} (X, \langle \cdot, \cdot \rangle) \]
\[ \land \text{UnboundedAbove} (X, \langle \cdot, \cdot \rangle) \]
\[ \land X \text{ DenseIn} (X, \langle \cdot, \cdot \rangle) \]
\[ \Rightarrow \text{UnboundedAbove} (\text{Cuts} (X, \langle \cdot, \cdot \rangle), \subset) \]

cuts_unbounded_below_thm
\[ \vdash \forall X \langle \cdot, \cdot \rangle \]
\[ \bullet \text{Irrefl} (X, \langle \cdot, \cdot \rangle) \]
\[ \land \text{Trans} (X, \langle \cdot, \cdot \rangle) \]
\[ \land \text{UnboundedBelow} (X, \langle \cdot, \cdot \rangle) \]
\[ \land X \text{ DenseIn} (X, \langle \cdot, \cdot \rangle) \]
\[ \Rightarrow \text{UnboundedBelow} (\text{Cuts} (X, \langle \cdot, \cdot \rangle), \subset) \]

dense_complete_subset_thm
\[ \vdash \forall X \langle \cdot, \cdot \rangle \ A \]
\[ \bullet \text{StrictLinearOrder} (X, \langle \cdot, \cdot \rangle) \]
\[ \land A \subseteq X \]
\[ \land \text{UnboundedAbove} (X, \langle \cdot, \cdot \rangle) \]
\[ \land \text{UnboundedBelow} (X, \langle \cdot, \cdot \rangle) \]
\[ \land A \text{ DenseIn} (X, \langle \cdot, \cdot \rangle) \]
\[ \land \text{Complete} (A, \langle \cdot, \cdot \rangle) \]
\[ \Rightarrow A = X \]

induced_order_irrefl_thm
\[ \vdash \forall X \langle \cdot, \cdot \rangle \ f \]
\[ \bullet (\forall a \bullet f \ a \in X) \land \text{Irrefl} (X, \langle \cdot, \cdot \rangle) \]
\[ \Rightarrow \text{Irrefl} (\text{Universe}, (\lambda a \ b \bullet f \ a \langle \cdot, \cdot \rangle b)) \]

induced_order_antisym_thm
\[ \vdash \forall X \langle \cdot, \cdot \rangle \ f \]
\[ \bullet (\forall a \bullet f \ a \in X) \land \text{Irrefl} (X, \langle \cdot, \cdot \rangle) \land \text{Antisym} (X, \langle \cdot, \cdot \rangle) \]
\[ \Rightarrow \text{Antisym} (\text{Universe}, (\lambda a \ b \bullet f \ a \langle \cdot, \cdot \rangle b)) \]

induced_order_trans_thm
\[ \vdash \forall X \langle \cdot, \cdot \rangle \ f \]
\[ \bullet (\forall a \bullet f \ a \in X) \land \text{Trans} (X, \langle \cdot, \cdot \rangle) \]
\[ \Rightarrow \text{Trans} (\text{Universe}, (\lambda a \ b \bullet f \ a \langle \cdot, \cdot \rangle b)) \]

induced_order_trich_thm
\[ \vdash \forall X \langle \cdot, \cdot \rangle \ f \]
\[ \bullet (\forall a \bullet f \ a \in X) \land \text{OneOne} f \land \text{Trich} (X, \langle \cdot, \cdot \rangle) \]
\[ \Rightarrow \text{Trich} (\text{Universe}, (\lambda a \ b \bullet f \ a \langle \cdot, \cdot \rangle b)) \]

induced_strict_partial_order_thm
\[ \vdash \forall X \langle \cdot, \cdot \rangle \ f \]
\[ \bullet (\forall a \bullet f \ a \in X) \land \text{StrictPartialOrder} (X, \langle \cdot, \cdot \rangle) \]
\[ \Rightarrow \text{StrictPartialOrder} (\text{Universe}, (\lambda a \ b \bullet f \ a \langle \cdot, \cdot \rangle b)) \]

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUAL USR029
induced_strict_linear_order_thm
\[ \forall X \forall a \in X \rightarrow \big( \forall f \in X \rightarrow f(x) \in X \land OneOne(f) \land \text{StrictLinearOrder}(X, f) \implies \text{StrictLinearOrder}(\text{Universe}, \lambda a b \cdot f(a) < f(b)) \big) \]

induced_order_complete_thm
\[ \forall X \forall a \in X \rightarrow \big( \forall f \in X \rightarrow f(a) \in X \land OneOne(f) \land \forall x \in X \rightarrow \exists a \in A \land x = f(a) \land \text{Complete}(X, f) \implies \text{Complete}(\text{Universe}, \lambda a b \cdot f(a) < f(b)) \big) \]

induced_order_dense_thm
\[ \forall X \forall a \in X \rightarrow \big( \forall f \in X \rightarrow f(a) \in X \land OneOne(f) \land \forall x \in X \rightarrow \exists a \in A \land x = f(a) \land \{x \mid \exists a \in A \land x = f(a) \} \text{DenseIn}(X, f) \implies A \text{DenseIn}(\text{Universe}, \lambda a b \cdot f(a) < f(b)) \big) \]

induced_order_unbounded_above_thm
\[ \forall X \forall a \in X \rightarrow \big( \forall f \in X \rightarrow f(a) \in X \land OneOne(f) \land \forall x \in X \rightarrow \exists a \in A \land x = f(a) \land \text{UnboundedAbove}(X, f) \implies \text{UnboundedAbove}(\text{Universe}, \lambda a b \cdot f(a) < f(b)) \big) \]

induced_order_unbounded_below_thm
\[ \forall X \forall a \in X \rightarrow \big( \forall f \in X \rightarrow f(a) \in X \land OneOne(f) \land \forall x \in X \rightarrow \exists a \in A \land x = f(a) \land \text{UnboundedBelow}(X, f) \implies \text{UnboundedBelow}(\text{Universe}, \lambda a b \cdot f(a) < f(b)) \big) \]
9.1.17 The Theory pair

9.1.17.1 Parents

misc

9.1.17.2 Children

N

9.1.17.3 Constants

\begin{align*}
\text{IsPairRep} & : (\text{'}a \rightarrow \text{'}b \rightarrow \text{BOOL}) \rightarrow \text{BOOL} \\
\text{Snd} & : \text{'}a \times \text{'}b \rightarrow \text{'}b \\
\text{Fst} & : \text{'}a \times \text{'}b \rightarrow \text{'}a \\
\$ & : \text{'}a \rightarrow \text{'}b \rightarrow \text{'}a \times \text{'}b \\
\text{Uncurry} & : (\text{'}a \rightarrow \text{'}b \rightarrow \text{'}c) \rightarrow \text{'}a \times \text{'}b \rightarrow \text{'}c \\
\text{Curry} & : (\text{'}a \times \text{'}b \rightarrow \text{'}c) \rightarrow \text{'}a \rightarrow \text{'}b \rightarrow \text{'}c
\end{align*}

9.1.17.4 Types

\text{'}1 \times \text{'}2

9.1.17.5 Fixity

\text{Right Infix 150:}

,  \times

9.1.17.6 Terminators

\times

9.1.17.7 Definitions

\begin{align*}
\text{IsPairRep}
\hspace{1cm}
\text{is_pair_rep_def} & \quad \vdash \exists \text{comma fst snd} \\
\hspace{1cm} & \quad \bullet (\forall x y)
\hspace{1cm} & \quad \bullet \text{IsPairRep (comma x y)}
\hspace{1cm} & \quad \quad \land \text{fst (comma x y) = x}
\hspace{1cm} & \quad \quad \land \text{snd (comma x y) = y}
\hspace{1cm} & \quad \quad \land (\forall x y a b)
\hspace{1cm} & \quad \quad \bullet \text{comma a b = comma x y } \Leftrightarrow a = x \land b = y)
\hspace{1cm} & \quad \quad \land (\forall p \bullet \text{IsPairRep p } \Rightarrow \text{comma (fst p) (snd p) = p})
\hspace{1cm} & \quad \times
\hspace{1cm} & \quad \times\_\text{def} & \quad \vdash \exists f \bullet \text{TypeDefn IsPairRep f}
\hspace{1cm} & \quad ,
\hspace{1cm} & \quad \text{Fst}
\hspace{1cm} & \quad \text{Snd}
\hspace{1cm} & \quad \text{pair_ops_def} & \quad \vdash (\forall x y \bullet \text{Fst (x, y) = x } \land \text{Snd (x, y) = y})
\end{align*}
\[ \forall x y a b \cdot (a, b) = (x, y) \iff a = x \land b = y \]
\[ \forall p \cdot (\text{Fst } p, \text{Snd } p) = p \]

**Uncurry**

def \[ \vdash \forall f x y \cdot \text{Uncurry } f (x, y) = f x y \]

**Curry**

def \[ \vdash \forall f x y \cdot \text{Curry } f x y = f(x, y) \]

### 9.1.17.8 Theorems

**pair-clauses** \[ \vdash \forall x y a b p f u f c \]
\[ \cdot \text{Fst } (x, y) = x \]
\[ \land \text{Snd } (x, y) = y \]
\[ \land ((a, b) = (x, y) \iff a = x \land b = y) \]
\[ \land (\text{Fst } p, \text{Snd } p) = p \]
\[ \land \text{Curry } f c x y = fc(x, y) \]
\[ \land \text{Uncurry } f u (x, y) = fu x y \]
\[ \land \text{Uncurry } f u p = fu(\text{Fst } p)(\text{Snd } p) \]
\[ \land ((a, b) = p \iff a = \text{Fst } p \land b = \text{Snd } p) \]
\[ \land (p = (a, b) \iff \text{Fst } p = a \land \text{Snd } p = b) \]
9.1.18 The Theory seq

9.1.18.1 Parents

\[ \text{fun} \text{rel} \]

9.1.18.2 Children

\[ \text{fin} \text{set} \]

9.1.18.3 Constants

<table>
<thead>
<tr>
<th>Elem</th>
<th>('a \text{LIST} \rightarrow \text{}'a \text{SET} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distinct</td>
<td>('a \text{LIST} \text{SET} )</td>
</tr>
<tr>
<td>Lists</td>
<td>('a \text{SET} \rightarrow \text{}'a \text{LIST SET} )</td>
</tr>
<tr>
<td>Lists₁</td>
<td>('a \text{SET} \rightarrow \text{}'a \text{LIST SET} )</td>
</tr>
<tr>
<td>InjLists</td>
<td>('a \text{SET} \rightarrow \text{}'a \text{LIST SET} )</td>
</tr>
<tr>
<td>Nth</td>
<td>('a \text{LIST} \rightarrow \text{}'a \text{N} \rightarrow \text{}'a \text{SET} )</td>
</tr>
<tr>
<td>$.$</td>
<td>\text{N} \rightarrow \text{N} \rightarrow \text{N SET}</td>
</tr>
<tr>
<td>ListRel</td>
<td>('a \text{LIST} \rightarrow (\text{N} \times \text{}'a) \text{SET} )</td>
</tr>
<tr>
<td>RelList</td>
<td>((\text{N} \times \text{}'a) \text{SET} \rightarrow \text{}'a \text{LIST} )</td>
</tr>
<tr>
<td>Last</td>
<td>('a \text{LIST} \rightarrow \text{}'a \text{LIST} )</td>
</tr>
<tr>
<td>Front</td>
<td>('a \text{LIST} \rightarrow \text{}'a \text{LIST} )</td>
</tr>
<tr>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td>Enumerate</td>
<td>\text{N SET} \rightarrow (\text{N} \times \text{N}) \text{SET}</td>
</tr>
<tr>
<td>Squash</td>
<td>((\text{N} \times \text{}'a) \text{SET} \rightarrow (\text{N} \times \text{}'a) \text{SET} )</td>
</tr>
<tr>
<td>Extract</td>
<td>\text{N SET} \rightarrow \text{}'a \text{LIST} \rightarrow \text{}'a \text{LIST}</td>
</tr>
<tr>
<td>$$$\text{Prefix}</td>
<td>\text{}'a \text{LIST} \rightarrow \text{}'a \text{LIST} \rightarrow \text{}'a \text{LIST}</td>
</tr>
<tr>
<td>$$$\text{Suffix}</td>
<td>\text{}'a \text{LIST} \rightarrow \text{}'a \text{LIST} \rightarrow \text{}'a \text{LIST}</td>
</tr>
<tr>
<td>$$$\text{In}</td>
<td>\text{}'a \text{LIST} \rightarrow \text{}'a \text{LIST} \rightarrow \text{}'a \text{LIST}</td>
</tr>
<tr>
<td>Flat</td>
<td>\text{}'a \text{LIST LIST} \rightarrow \text{}'a \text{LIST}</td>
</tr>
</tbody>
</table>

9.1.18.4 Aliases

| # | Length : \('a \text{LIST} \rightarrow \text{N} \) |
| ~ | Append : \('a \text{LIST} \rightarrow \text{\}'a \text{LIST} \rightarrow \text{\}'a \text{LIST} |
| Head | \text{Hd} : \text{\}'a \text{LIST} \rightarrow \text{\}'a |
| Tail | \text{Tl} : \text{\}'a \text{LIST} \rightarrow \text{\}'a \text{LIST} |

9.1.18.5 Fixity

Left Infix 300:

\[ \text{\textbar} \]

Right Infix 290:

\[ \text{\textbar} \]

Right Infix 300:

\[ \text{In} \text{ Prefix Suffix} \text{\textbar} \]
9.1.18.6 Definitions

Elems \[\vdash \text{Elems } [] = \{\} \]
\[\land (\forall x \in \text{Elems} \land \text{Cons } x \in \text{Elems} \implies \{x\} \cup \text{Elems } l)\]

Distinct
\[\vdash [] \in \text{Distinct} \]
\[\land (\forall x \in \text{Distinct} \land \text{Cons } x \in \text{Distinct}) \]
\[\equiv \neg x \in \text{Elems } l \land l \in \text{Distinct}\]

Lists
\[\vdash \forall a \bullet \text{Lists } a = \{l \mid \text{Elems } l \subseteq a\}\]

Lists\(_1\)
\[\vdash \forall a \bullet \text{Lists}_1 a = \text{Lists } a \setminus \{[]\}\]

InjLists
\[\vdash \forall a \bullet \text{InjLists } a = \text{Lists } a \cap \text{Distinct}\]

Nth
\[\vdash \forall l \times n \]
\[\bullet \text{Nth } (\text{Cons } x \in l) n \]
\[= (\text{if } n = 1 \text{ then } x \text{ else } \text{Nth } l (n - 1))\]

.. \[\vdash \forall m \times m \bullet m \ldots n = \{i \mid m \leq i \land i \leq n\}\]

ListRel
\[\vdash \forall l \bullet \text{ListRel } l = 1 \ldots \# l < \text{Graph } (\text{Nth } l)\]

RelList
\[\vdash \text{ConstSpec} \]
\[\equiv \forall l \bullet \text{RelList}' (\text{ListRel } l) = l\]

Last
\[\vdash \forall x \in l \]
\[\bullet \text{Last } (\text{Cons } x \in l) = (\text{if } l = [] \text{ then } x \text{ else } \text{Last } l)\]

Front
\[\vdash \forall x \in l \]
\[\bullet \text{Front } (\text{Cons } x \in l) \]
\[= (\text{if } l = [] \text{ then } [] \text{ else } \text{Cons } x (\text{Front } l))\]

\[\vdash \forall a \times x \in l \]
\[\bullet \text{Enumerate } a \]
\[= \{(i, j) \mid \exists l \land j \in a \}
\[\land \text{Cons } x \in l \land a \leq \text{Distinct} \land \# l = i \land \text{Elems } l = a \cap \emptyset \ldots j\}\]

Enumerate
\[\vdash \forall a \]

\[\bullet \text{Squash } r = \text{Enumerate } (\text{Dom } r) \propto r\]

Extract
\[\vdash \forall a \bullet \text{Extract } a \in \text{RelList} (\text{Squash } (a \triangle LeftRel l))\]

Prefix
\[\vdash \forall s \in s \bullet \text{Prefix } t \equiv (\exists v \bullet s \triangle v = t)\]

Suffix
\[\vdash \forall s \in s \bullet \text{Suffix } t \equiv (\exists v \bullet u \triangle s = t)\]

In
\[\vdash \forall s \in s \bullet \text{In } t \equiv (\exists u v \bullet u \triangle s \triangle v = t)\]

Flat
\[\vdash \forall e \bullet \text{Flat } [] = [] \land \text{Flat } (\text{Cons } e \in l) = e \triangle \text{Flat } l\]
9.1.19  The Theory sets

9.1.19.1 Parents

\textit{basic\_hol}

9.1.19.2 Children

\text{orders} \text{ set\_thms} \text{ \ Z} \text{ hol}

9.1.19.3 Constants

\text{IsSetRep} \quad (\forall a \rightarrow \text{BOOL}) \rightarrow \text{BOOL}

\text{\$\in} \quad \forall a \rightarrow \forall a \text{ SET} \rightarrow \text{BOOL}

\text{\$SetComp} \quad (\forall a \rightarrow \text{BOOL}) \rightarrow \forall a \text{ SET}

\text{Insert} \quad \forall a \rightarrow \forall a \text{ SET} \rightarrow \forall a \text{ SET}

\text{Universe} \quad \forall a \text{ SET}

\{\} \quad \forall a \text{ SET}

\sim \quad \forall a \text{ SET} \rightarrow \forall a \text{ SET}

\text{\$\cup} \quad \forall a \text{ SET} \rightarrow \forall a \text{ SET} \rightarrow \forall a \text{ SET}

\text{\$\cap} \quad \forall a \text{ SET} \rightarrow \forall a \text{ SET} \rightarrow \forall a \text{ SET}

\text{\$\setminus} \quad \forall a \text{ SET} \rightarrow \forall a \text{ SET} \rightarrow \forall a \text{ SET}

\text{\$\subseteq} \quad \forall a \text{ SET} \rightarrow \forall a \text{ SET} \rightarrow \text{BOOL}

\text{\$\subset} \quad \forall a \text{ SET} \rightarrow \forall a \text{ SET} \rightarrow \text{BOOL}

\text{\cup} \quad \forall a \text{ SET} \rightarrow \forall a \text{ SET}

\text{\cap} \quad \forall a \text{ SET} \rightarrow \forall a \text{ SET}

\text{\P} \quad \forall a \text{ SET} \rightarrow \forall a \text{ SET} \rightarrow \text{SET}

9.1.19.4 Types

\forall \text{ SET}

9.1.19.5 Fixity

\textit{Binder:} \quad \text{SetComp}

\textit{Left Infix 265:}

\text{\setminus}

\textit{Right Infix 230:}

\text{\subseteq} \quad \text{\in} \quad \text{\subset}

\textit{Right Infix 250:}

\text{\ominus}

\textit{Right Infix 260:}

\text{\cup}

\textit{Right Infix 270:}

\text{\cap
9.1.19.6 Definitions

IsSetRep
is_set_rep_def
\(\vdash IsSetRep = (\lambda x \cdot T)\)

SET
set_def
\(\vdash \exists f \cdot TypeDefn \ IsSetRep \ f\)

SetComp
∈
set_comp_def
\(\vdash \forall x p a b \cdot \ (x \in \{v|p v\} \iff p x)\)
\& \ (a = b \iff (\forall x \cdot x \in a \iff x \in b))\)

Empty
Universe
Insert
insert_def
\(\vdash \forall x y a \cdot \neg x \in \{}\)
\& \ (x \in Universe
\& \ (x \in Insert y a \iff x = y \vee x \in a))\)

∼
complement_def
\(\vdash \forall x a \cdot x \in \sim a \iff \sim x \in a\)

∪
∪_def
\(\vdash \forall x a b \cdot x \in a \cup b \iff x \in a \vee x \in b\)

∩
∩_def
\(\vdash \forall x a b \cdot x \in a \cap b \iff x \in a \wedge x \in b\)

\−
set_dif_def
\(\vdash \forall x a b \cdot x \in a \setminus b \iff x \in a \wedge \sim x \in b\)

⊕
⊕_def
\(\vdash \forall x a b \cdot x \in a \oplus b \iff \sim (x \in a \iff x \in b)\)

⊆
⊆_def
\(\vdash \forall a b \cdot \ a \subseteq b \iff (\forall x \cdot x \in a \Rightarrow x \in b)\)

⊂
⊂_def
\(\vdash \forall a b \cdot \ a \subset b \iff a \subseteq b \wedge (\exists x \cdot \sim x \in a \wedge x \in b)\)

∪
∪_def
\(\vdash \forall x a \cdot x \in \bigcup a \iff (\exists s \cdot x \in s \wedge s \in a)\)

∩
∩_def
\(\vdash \forall x a \cdot x \in \bigcap a \iff (\forall s \cdot s \in a \Rightarrow x \in s)\)

P
P_def
\(\vdash \forall x a \cdot x \in P a \iff x \subseteq a\)

9.1.19.7 Theorems

sets_clauses
\(\vdash \forall x y p q \cdot \ (x \in \{\} \iff F)\)
\& \ (x \in Universe \iff T)\)
\& \ (x \in \{v|q\} \iff q)\)
\& \ (x \in \{v|p v\} \iff p x)\)
\& \ (x \in \{v|v = y\} \iff x = y)\)
\& \ (x \in \{y\} \iff x = y)\)

complement_clauses
\( \forall x \ a \bullet x \in a \iff \neg x \in a \)
\[ \wedge \sim \text{Universe} = \{\} \]
\[ \wedge \sim \{\} = \text{Universe} \]

**\( \cup \_ \text{clauses} \)**

\( \vdash \forall a \)

- \( a \cup \{\} = a \)
- \( \{\} \cup a = a \)
- \( a \cup \text{Universe} = \text{Universe} \)
- \( \text{Universe} \cup a = \text{Universe} \)
- \( a \cup a = a \)

**\( \cap \_ \text{clauses} \)**

\( \vdash \forall a \)

- \( a \cap \{\} = \{\} \)
- \( \{\} \cap a = \{\} \)
- \( a \cap \text{Universe} = a \)
- \( \text{Universe} \cap a = a \)
- \( a \cap a = a \)

**\( \setset \_ \text{clauses} \)**

\( \vdash \forall a \)

- \( a \setset \{\} = a \)
- \( \{\} \setset a = \{\} \)
- \( a \setset \text{Universe} = \{\} \)
- \( \text{Universe} \setset a = \sim a \)
- \( a \setset a = \{\} \)

**\( \ominus \_ \text{clauses} \)**

\( \vdash \forall a \)

- \( a \ominus \{\} = a \)
- \( \{\} \ominus a = \{\} \)
- \( a \ominus \text{Universe} = \{\} \)
- \( \text{Universe} \ominus a = \sim a \)
- \( a \ominus a = \{\} \)

**\( \subseteq \_ \text{clauses} \)**

\( \vdash \forall a \bullet a \subseteq \{\} \subseteq a \land a \subseteq \text{Universe} \)

**\( \subset \_ \text{clauses} \)**

\( \vdash \forall a \bullet \neg a \subset a \land \neg a \subset \{\} \land \{\} \subset \text{Universe} \)

**\( \cup \_ \text{clauses} \)**

\( \vdash \bigcup \{\} = \{\} \land \bigcup \text{Universe} = \text{Universe} \)

**\( \cap \_ \text{clauses} \)**

\( \vdash \bigcap \{\} = \text{Universe} \land \bigcap \text{Universe} = \{\} \)

**\( \mathcal{P} \_ \text{clauses} \)**

\( \vdash \forall a \bullet \mathcal{P} \{\} = \{\{}\} \land \mathcal{P} \text{Universe} = \text{Universe} \land \{\} \in \mathcal{P} a \land \{\} \in \mathcal{P} a \)

**\( \emptyset \_ \text{clauses} \)**

\( \vdash \forall x \ a \bullet \{x\mid F\} = \{\} \land \neg x \in \{\} \land \{\} \cup a = a \land a \cup \{\} = a \land \{\} \cap a = \{\} \land a \cap \{\} = \{\} \land a \setset \{\} = a \land \{\} \setset a = \{\} \land a \setset \{\} = a \land \{\} \setset a = a \land \{\} \subseteq a \land (a \subseteq \{\} \iff a = \{\}) \land (\{\} \subset a \iff a = \{\}) \)
\begin{itemize}
\item \( x \in \text{Universe} \iff T \)
\item \( x \in \{ \} \ iff F \)
\item \( \forall x \, a \, b \)
\item \( (x \in a \cup b \iff x \in a \land x \in b) \)
\item \( (x \in a \setminus b \iff x \in a \land \neg x \in b) \)
\item \( (x \in a \bigtriangleup b \iff \neg (x \in a \iff x \in b)) \)
\end{itemize}

\( \{ \} \in \mathcal{P} \{ \} = \{\{\}\} \)

\( \in_{\text{in\_clauses}} \vdash (\forall x \, y \, a \cdot (x \in \text{Universe} \iff T) \land (x \in \{ \} \iff F) \land (x \in \text{Insert} \, y \, a \iff x = y \lor x \in a)) \land (\forall x \, a \, b \cdot (x \in a \cup b \iff x \in a \lor x \in b) \land (x \in a \setminus b \iff x \in a \land \neg x \in b) \land (x \in a \bigtriangleup b \iff \neg (x \in a \iff x \in b))) \)

\( \in_{\text{sets\_ext\_clauses}} \vdash (\forall a \, b \cdot (a \subset b \iff (\forall x \cdot x \in a \Rightarrow x \in b) \land (\exists x \cdot \neg x \in a \land x \in b)) \land (a \subseteq b \iff (\forall x \cdot x \in a \Rightarrow x \in b)) \land (a = b \iff (\forall x \cdot x \in a \iff x \in b))) \)
9.1.20 The Theory set_thms

9.1.20.1 Parents

sets

9.1.20.2 Children

\textit{fun_rel_thms}

9.1.20.3 Constants

\texttt{Nest} 'a SET SET \to BOOL
\texttt{SubsetClosed} 'a SET SET \to BOOL
\texttt{NestClosed} 'a SET SET \to BOOL
\texttt{Maximal}_\subseteq 'a SET SET \to 'a SET \to BOOL
\texttt{Choose} 'a SET \to 'a

9.1.20.4 Definitions

\texttt{Nest} \vdash \forall u \bullet \text{Nest } u \iff (\forall a b \bullet a \in u \land b \in u \Rightarrow a \subseteq b \lor b \subseteq a)
\texttt{SubsetClosed} \vdash \forall u \bullet \text{SubsetClosed } u \iff (\forall a b \bullet a \in u \land b \subseteq a \Rightarrow b \in u)
\texttt{NestClosed} \vdash \forall u \bullet \text{NestClosed } u \iff (\forall v \bullet v \subseteq u \land \text{Nest } v \Rightarrow \bigcup v \in u)
\texttt{Maximal}_\subseteq \vdash \forall u a
\bullet \text{Maximal}_\subseteq u a
\iff a \in u \land (\forall b \bullet b \in u \land a \subseteq b \Rightarrow b = a)
\texttt{Choose} \vdash \text{ConstSpec}
(\lambda \text{Choose'} \bullet \forall a \bullet \neg a = \{\} \iff \text{Choose'} a \in a)
\texttt{Choose}

9.1.20.5 Theorems

\texttt{Choose\_consistent}
\vdash \text{Consistent} (\lambda \text{Choose'} \bullet \forall a \bullet \neg a = \{\} \iff \text{Choose'} a \in a)
\texttt{zorn\_thm1}
\vdash \forall u
\bullet \neg u = \{\} \land \text{SubsetClosed } u \land \text{NestClosed } u
\Rightarrow (\exists a \bullet \text{Maximal}_\subseteq u a)
9.1. Theory Listings

9.1.21 The Theory sum

9.1.21.1 Parents

\textit{combin}

9.1.21.2 Children

\textit{hol}

9.1.21.3 Constants

\begin{itemize}
  \item \texttt{IsSumRep} \quad \texttt{'a \times 'b \times BOOL \rightarrow BOOL}
  \item \texttt{IsR} \quad \texttt{'a + 'b \rightarrow BOOL}
  \item \texttt{IsL} \quad \texttt{'a + 'b \rightarrow BOOL}
  \item \texttt{OutR} \quad \texttt{'a + 'b \rightarrow 'b}
  \item \texttt{OutL} \quad \texttt{'a + 'b \rightarrow 'a}
  \item \texttt{InR} \quad \texttt{'b \rightarrow 'a + 'b}
  \item \texttt{InL} \quad \texttt{'a \rightarrow 'a + 'b}
\end{itemize}

9.1.21.4 Types

\texttt{'1 + '2}

9.1.21.5 Fixity

\textit{Right Infix 300:}

\begin{itemize}
  \item +
\end{itemize}

9.1.21.6 Definitions

\texttt{IsSumRep}

\texttt{is\_sum\_rep\_def}

\begin{itemize}
  \item \exists \texttt{inl inr outl outr isl isr}
    \begin{itemize}
      \item \forall x_1 x_2 y_1 y_2 z
        \begin{itemize}
          \item \texttt{IsSumRep (inl x1)}
          \item \texttt{IsSumRep (inr y1)}
          \item \texttt{(IsSumRep z)}
            \begin{itemize}
              \item \texttt{inl (outl z) = z \lor inr (outr z) = z}
              \item \texttt{(inl x1 = inl x2 \leftrightarrow x1 = x2)}
              \item \texttt{(inr y1 = inr y2 \leftrightarrow y1 = y2)}
              \item \texttt{\neg inl x1 = inr y1}
              \item \texttt{\neg inr y1 = inl x1}
              \item \texttt{outr (inl x1) = x1}
              \item \texttt{outr (inr y1) = y1}
              \item \texttt{(IsSumRep z \Rightarrow (isl z \leftrightarrow inl (outl z) = z))}
              \item \texttt{(IsSumRep z \Rightarrow (isr z \leftrightarrow inr (outr z) = z))}
            \end{itemize}
          \end{itemize}
        \end{itemize}
    \end{itemize}

\texttt{sum\_def}

\texttt{\exists f \bullet TypeDefn IsSumRep f}

\texttt{InL}

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
\textbf{9.1.21.7 Theorems}

\textit{sum\_cases\_thm} \quad \vdash \forall x \bullet (\exists y \bullet x = \text{InL} y) \lor (\exists z \bullet x = \text{InR} z)

\textit{sum\_fns\_thm} \quad \vdash \forall f \bullet g \exists h \bullet h \circ \text{InL} = f \land h \circ \text{InR} = g
9.1.22 The Theory $\mathbb{R}$

9.1.22.1 Parents

$\text{orders dyadic}$

9.1.22.2 Constants

| Is$_\mathbb{R}$.Rep & $(\mathbb{N} \times \mathbb{Z})$ SET $\rightarrow$ BOOL |
|---------------------|--------------------------------------------------|
| $\prec_\mathbb{R}$ & $\mathbb{R} \rightarrow \mathbb{R} \rightarrow$ BOOL |
| $\preceq_\mathbb{R}$ & $\mathbb{R} \rightarrow \mathbb{R} \rightarrow$ BOOL |
| $\succ_\mathbb{R}$ & $\mathbb{R} \rightarrow \mathbb{R} \rightarrow$ BOOL |
| $\succeq_\mathbb{R}$ & $\mathbb{R} \rightarrow \mathbb{R} \rightarrow$ BOOL |
| Sup & $\mathbb{R}$ SET $\rightarrow$ $\mathbb{R}$ |
| 1$_\mathbb{R}$ & $\mathbb{R}$ |
| 0$_\mathbb{R}$ & $\mathbb{R}$ |
| $\oplus_\mathbb{R}$ & $\mathbb{R} \rightarrow \mathbb{R} \rightarrow$ $\mathbb{R}$ |
| $\sim_\mathbb{R}$ & $\mathbb{R} \rightarrow$ $\mathbb{R}$ |
| $\neg_\mathbb{R}$ & $\mathbb{R} \rightarrow$ $\mathbb{R}$ |
| Abs$_\mathbb{R}$ & $\mathbb{R}$ $\rightarrow$ $\mathbb{R}$ |
| $\cdot_\mathbb{R}$ & $\mathbb{R} \rightarrow$ $\mathbb{R} \rightarrow$ $\mathbb{R}$ |
| $\div_\mathbb{R}$ & $\mathbb{R} \rightarrow$ $\mathbb{R} \rightarrow$ $\mathbb{R}$ |
| $\div_N$ & $\mathbb{N}$ $\rightarrow$ $\mathbb{N}$ $\rightarrow$ $\mathbb{R}$ |
| $\sim^{-1}$ & $\mathbb{R}$ $\rightarrow$ $\mathbb{R}$ |
| $\sim^{-N}$ & $\mathbb{R}$ $\rightarrow$ $\mathbb{N}$ $\rightarrow$ $\mathbb{R}$ |
| Z$\mathbb{R}$ & $\mathbb{Z}$ $\rightarrow$ $\mathbb{R}$ |
| $\sim^Z$ & $\mathbb{R}$ $\rightarrow$ $\mathbb{Z}$ $\rightarrow$ $\mathbb{R}$ |
| Float & $\mathbb{N}$ $\rightarrow$ $\mathbb{Z}$ $\rightarrow$ $\mathbb{Z}$ $\rightarrow$ $\mathbb{R}$ |
| Max$_\mathbb{R}$ & $\mathbb{R}$ LIST $\rightarrow$ $\mathbb{R}$ |
| Min$_\mathbb{R}$ & $\mathbb{R}$ LIST $\rightarrow$ $\mathbb{R}$ |

9.1.22.3 Aliases

| $<$ & $\prec_\mathbb{R}$ : $\mathbb{R}$ $\rightarrow$ $\mathbb{R}$ $\rightarrow$ BOOL |
| $\leq$ & $\preceq_\mathbb{R}$ : $\mathbb{R}$ $\rightarrow$ $\mathbb{R}$ $\rightarrow$ BOOL |
| $>$ & $\succ_\mathbb{R}$ : $\mathbb{R}$ $\rightarrow$ $\mathbb{R}$ $\rightarrow$ BOOL |
| $\geq$ & $\succeq_\mathbb{R}$ : $\mathbb{R}$ $\rightarrow$ $\mathbb{R}$ $\rightarrow$ BOOL |
| $+$ & $\oplus_\mathbb{R}$ : $\mathbb{R}$ $\rightarrow$ $\mathbb{R}$ $\rightarrow$ $\mathbb{R}$ |
| $\sim$ & $\sim_\mathbb{R}$ : $\mathbb{R}$ $\rightarrow$ $\mathbb{R}$ |
| $-$ & $\neg_\mathbb{R}$ : $\mathbb{R}$ $\rightarrow$ $\mathbb{R}$ $\rightarrow$ $\mathbb{R}$ |
| Abs & $\text{Abs}_\mathbb{R}$ : $\mathbb{R}$ $\rightarrow$ $\mathbb{R}$ |
| $\cdot$ & $\cdot_\mathbb{R}$ : $\mathbb{R}$ $\rightarrow$ $\mathbb{R}$ $\rightarrow$ $\mathbb{R}$ |
| $/$ & $\div_\mathbb{R}$ : $\mathbb{R}$ $\rightarrow$ $\mathbb{R}$ $\rightarrow$ $\mathbb{R}$ |
| $\div$ & $\div_N$ : $\mathbb{N}$ $\rightarrow$ $\mathbb{N}$ $\rightarrow$ $\mathbb{R}$ |
| $\sim$ & $\sim^{-1}$ : $\mathbb{R}$ $\rightarrow$ $\mathbb{R}$ |
| $\sim^{-N}$ & $\sim^{-N}$ : $\mathbb{R}$ $\rightarrow$ $\mathbb{N}$ $\rightarrow$ $\mathbb{R}$ |
| $\sim^Z$ & $\sim^Z$ : $\mathbb{R}$ $\rightarrow$ $\mathbb{Z}$ $\rightarrow$ $\mathbb{R}$ |

9.1.22.4 Types

| $\mathbb{R}$ |
9.1.22.5 Fixity

Left Infix 305:

\[ R \]

Left Infix 315:

\[ \div \ N \div R \]

Right Infix 210:

\[ \lt R \gt R \leq R \geq R \]

Right Infix 300:

\[ + R \]

Right Infix 310:

\[ * R \]

Right Infix 320:

\[ \hat{N} \hat{Z} \]

Postfix 320:

\[ -1 \]

9.1.22.6 Definitions

\[ \text{Is}_R \text{Rep} \vdash \forall a \cdot \text{Is}_R \text{Rep} a \iff a \in \text{Cuts (Universe, $dy\_less)} \]

\[ R \text{def} \vdash \exists f \cdot \text{TypeDefn Is}_R \text{Rep} f \]

\[ \lt R \vdash \text{ConstSpec (\lambda \lt' R \cdot } \text{StrictLinearOrder (Universe, } \lt' R) \]

\[ \land \text{UnboundedBelow (Universe, } \lt' R) \]

\[ \land \text{UnboundedAbove (Universe, } \lt' R) \]

\[ \land \text{Complete (Universe, } \lt' R) \]

\[ \land (\exists \iota) \]

\[ \bullet (\forall a b \cdot \lt R (\iota a) (\iota b) \iff a \diamond dy\_less b) \]

\[ \land \{ x | \exists a \cdot \iota a = x \} \]

\[ \text{DenseIn (Universe, } \lt' R) \}) \]

\[ \text{\$<} \]

\[ \leq R \vdash \forall x y \cdot x \leq y \iff x < y \lor x = y \]

\[ > R \vdash \forall x y \cdot x > y \iff y < x \]

\[ \geq R \vdash \forall x y \cdot x \geq y \iff y \leq x \]

\[ \text{Sup} \vdash \text{ConstSpec (\lambda } \text{Sup'} \cdot \]

\[ \bullet \forall A \]

\[ \bullet \neg A = \{ \} \land (\exists b \cdot \forall x \cdot x \in A \Rightarrow x \leq b) \]

\[ \Rightarrow (\forall x \cdot x \in A \Rightarrow x \leq \text{Sup'} A) \]

\[ \land (\forall b) \]

\[ \bullet (\forall x \cdot x \in A \Rightarrow x \leq b) \Rightarrow \text{Sup'} A \leq b) \}

\[ \text{Sup} \]

\[ + R \]

\[ 0_R \]

\[ 1_R \vdash \text{ConstSpec (\lambda (+' R, } 0'_R, '1'_R) \]

\[ \bullet (\forall x y z) \]

\[ \bullet (+'_ R (+'_ R x y) z) = (+'_ R x) (+'_ R y z)) \]

\[ \land (\forall x y \cdot +'_ R x y = '1'_ R y x) \]

\[ \land (\forall x \cdot +'_ R x 0'_ R = x) \]

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
9.1. Theory Listings 429

\[ \sim_R \vdash \text{ConstSpec} (\lambda \sim_R \bullet \forall x \bullet x + \sim_R x = 0_R) \sim \]

\[ \rightarrow_R \vdash \forall x y \bullet x - y = x + \sim y \]

\[ \neg_R \vdash 0. = 0_R \land (\forall m \bullet \neg_R (m + 1) = \neg_R m + 1_R) \]

\[ \text{Abs}_R \vdash \forall x \bullet \text{Abs} x = (if \ 0. \leq x \ then \ x \ else \sim x) \]

\[ \text{ConstSpec} (\lambda \bullet \forall x y \bullet \sim_R x y = 0_R) \]

\[ \text{ConstSpec} (\lambda \bullet \forall x y \bullet \sim_R x y = 0_R) \]

\[ \forall m \bullet m / n = \text{NR}_m / \text{NR}_n \]

\[ \forall x \bullet x - l = 1. / x \]

\[ \forall x \bullet x - 0 = 1.) \land (\forall x m \bullet x - (m + 1) = x * x - m) \]

\[ \text{ConstSpec} (\lambda \bullet \text{NZ} \sim_R m) \]

\[ \forall x \bullet (\text{NZ} m) = x - m \]

\[ \forall x \bullet (\text{NZ} (m + 1)) = (x - (m + 1)) - l \]

\[ \forall m \bullet (e + \sim) \]

\[ \text{ConstSpec} (\lambda \bullet \text{Max} x = x) \]

\[ \forall x y L \]

\[ \text{Max} (\text{Cons} x (\text{Cons} y L)) \]
\( \text{Min}_R \) \[\vdash \text{ConstSpec} \] 
\[ \begin{align*} & (\forall \ x \bullet \text{Min}_R [x] = x) \\
& \land (\forall \ x \ y \ L \\
& \bullet \text{Min}_R' (\text{Cons} \ x \ (\text{Cons} \ y \ L)) \\
& \quad = (\text{if } x > \text{Min}_R' (\text{Cons} \ y \ L) \\
& \quad \quad \text{then } \text{Min}_R' (\text{Cons} \ y \ L) \\
& \quad \quad \quad \text{else } x)) \end{align*} \]

9.1.22.7 Theorems

dy_less_order_lemmas_thm
\[ \vdash \text{StrictLinearOrder} \ (\text{Universe}, \ dy_{\less}) \]
\[ \land \ \text{UnboundedBelow} \ (\text{Universe}, \ dy_{\less}) \]
\[ \land \ \text{UnboundedAbove} \ (\text{Universe}, \ dy_{\less}) \]
\[ \land \ \text{Universe DenseIn} \ (\text{Universe}, \ dy_{\less}) \]
is_R_rep_consistent_thm
\[ \vdash \exists \ a \bullet \text{Is}_R \text{Rep} \ a \]
<_R_consistent
\[ \vdash \text{Consistent} \]
\[ (\lambda <'_R \]
\[ \bullet \text{StrictLinearOrder} \ (\text{Universe}, <'_R) \]
\[ \land \ \text{UnboundedBelow} \ (\text{Universe}, <'_R) \]
\[ \land \ \text{UnboundedAbove} \ (\text{Universe}, <'_R) \]
\[ \land \ \text{Complete} \ (\text{Universe}, <'_R) \]
\[ \land \ (\exists \ \iota \]
\[ \bullet (\forall \ a \ b \bullet <'_R (\iota \ a) \ (\iota \ b) \iff a \ dy_{\less} b) \]
\[ \land \ \{x|\exists \ a \bullet \iota \ a = x\} \]
\[ \text{DenseIn} \ (\text{Universe}, <'_R)) \]

R_unbounded_below_thm
\[ \vdash \forall \ x \bullet \exists \ y \bullet y < x \]

R_unbounded_above_thm
\[ \vdash \forall \ x \bullet \exists \ y \bullet x < y \]

R_less_irrefl_thm
\[ \vdash \forall \ x \bullet \neg x < x \]

R_less_antisy_thm
\[ \vdash \forall \ x \ y \bullet \neg (x < y \land y < x) \]

R_less_trans_thm
\[ \vdash \forall \ x \ y \ z \bullet x < y \land y < z \Rightarrow x < z \]

R_less_cases_thm
\[ \vdash \forall \ x \ y \bullet x < y \lor x = y \lor y < x \]

R_<_cases_thm
\[ \vdash \forall \ x \ y \bullet x \leq y \lor y \leq x \]

R_<_less_cases_thm
\[ \vdash \forall \ x \ y \bullet x \leq y \lor y < x \]

R_eq_<_thm
\[ \vdash \forall \ x \ y \bullet x = y \iff x \leq y \land y \leq x \]

R_<_antisym_thm
\[ \vdash \forall \ x \ y \bullet x < y \land y < x \Rightarrow x = y \]

R_less_<_trans_thm
\[ \vdash \forall \ x \ y \ z \bullet x < y \land y \leq z \Rightarrow x < z \]

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
\[ \vdash \forall x \; x \leq y \land y < z \Rightarrow x < z \]

\[ \vdash \forall x \; x \leq x \]

\[ \vdash \forall x \; x \leq y \land y \leq z \Rightarrow x \leq z \]

\[ \vdash \forall x \; x \leq y \iff \neg y < x \]

\[ \vdash \forall x \; x \leq y \iff y < x \]

\[ \vdash \forall x \; x < y \Rightarrow \neg x = y \]

\[ \vdash \forall x \; x < y \Rightarrow (\exists z \; x < z \land z < y) \]

\[ \vdash \forall A \\
\quad \bullet \neg A = \{\} \land (\exists b \bullet \forall x \; x \in A \Rightarrow x \leq b) \\
\quad \Rightarrow (\forall x \bullet x \in A \Rightarrow x \leq Sup' A) \\
\quad \land (\forall b \bullet (\forall x \bullet x \in A \Rightarrow x \leq b) \Rightarrow s \leq b)) \]

\[ \vdash Consistent \]

\[ (\lambda Sup' \\
\quad \bullet \forall A \\
\quad \quad \bullet \neg A = \{\} \land (\exists b \bullet \forall x \; x \in A \Rightarrow x \leq b) \\
\quad \quad \Rightarrow (\forall x \bullet x \in A \Rightarrow x \leq Sup' A) \\
\quad \quad \land (\forall b \bullet (\forall x \bullet x \in A \Rightarrow x \leq b) \Rightarrow Sup' A \leq b)) \]

\[ \vdash \forall A \; a \\
\quad \bullet \neg A = \{\} \land (\forall x \bullet x \in A \Rightarrow x \leq a) \\
\quad \Rightarrow (\forall x \bullet x \in A \Rightarrow x \leq Sup A) \\
\quad \land (\forall b \bullet (\forall x \bullet x \in A \Rightarrow x \leq b) \Rightarrow Sup A \leq b) \]

\[ \vdash \forall A \\
\quad \bullet \neg A = \{\} \land (\exists a \bullet \forall x \; x \in A \Rightarrow x \leq a) \\
\quad \Rightarrow (\forall x \bullet x < Sup A \iff (\exists y \bullet y \in A \land x < y)) \]

\[ \vdash \forall A \\
\quad \bullet \neg A = \{\} \land (\exists a \bullet \forall x \; x \in A \Rightarrow x \leq a) \\
\quad \land (\exists y \bullet y \in A \land x < y) \\
\quad \Rightarrow x < Sup A \]

\[ \vdash \forall A \\
\quad \bullet \neg A = \{\} \land (\exists a \bullet \forall x \; x \in A \Rightarrow x \leq a) \\
\quad \Rightarrow (\forall x \\
\quad \quad \bullet x \leq Sup A \\
\quad \quad \iff (\forall y \bullet (\forall z \bullet z \in A \Rightarrow z \leq y) \Rightarrow x \leq y)) \]

\[ \vdash \forall A \\
\quad \bullet \neg A = \{\} \land (\exists a \bullet \forall x \; x \in A \Rightarrow x \leq a) \\
\quad \land (\forall y \bullet (\forall z \bullet z \in A \Rightarrow z \leq y) \Rightarrow x \leq y) \]
\[ x \leq \text{Sup } A \]

\[ \vdash \forall A \ x \in A \land (\exists a \land \forall x \ x \in A \Rightarrow x \leq a) \Rightarrow x \leq \text{Sup } A \]

\[ \vdash \forall A \ B \]

\[ \bullet \neg A = \{\} \land (\exists a \land \forall x \ x \in A \Rightarrow x \leq a) \land \neg B = \{\} \land (\exists b \land \forall y \ y \in B \Rightarrow y \leq b) \land A \subseteq B \Rightarrow \text{Sup } A \leq \text{Sup } B \]

\[ \vdash \forall A \ a \ x \]

\[ \bullet \neg A = \{\} \land (\exists a \land \forall x \ x \in A \Rightarrow x \leq a) \land (\forall y \ y \in A \Rightarrow y \leq x) \Rightarrow \text{Sup } A \leq x \]

\[ \vdash \forall A \ a \ x \ z \]

\[ \bullet \neg A = \{\} \land (\exists a \land \forall x \ x \in A \Rightarrow x \leq a) \land (\forall y \ y \in A \Rightarrow y \leq x) \land x < z \Rightarrow \text{Sup } A < z \]

\[ \vdash \forall A \ a \ s \]

\[ \bullet \neg A = \{\} \land (\forall x \ x \in A \Rightarrow x \leq s) \land (\forall x \ (\forall y \ y \in A \Rightarrow y \leq x) \Rightarrow s \leq x) \Rightarrow \text{Sup } A = s \]

\[ \vdash \forall A \ a \ s \]

\[ \bullet \neg A = \{\} \land (\forall x \ x \in A \Rightarrow x \leq s) \land (\forall x \ (\forall y \ y \in A \Rightarrow y \leq x) \Rightarrow s \leq x) \land s = \text{Sup } A \]

\[ \vdash \forall A \ a \]

\[ \bullet \neg A = \{\} \land (\forall x \ x \in A \Rightarrow x \leq a) \land \neg \text{Sup } A \in A \Rightarrow (\forall x \ x < \text{Sup } A \Rightarrow (\exists y \ y < x \land y < \text{Sup } A \land y \in A)) \]

\[ +R_{\text{consistent}} \]

\[ 0R_{\text{consistent}} \]

\[ 1R_{\text{consistent}} \]

\[ \vdash \text{Consistent} \]

\[ (\lambda (+_R, 0'_R, 1'_R)) \]

\[ (\forall x \ y \ z \ (+_R (+_R x y) z = +'_R x (+'_R y z))) \land (\forall x \ y \ (+'_R x y) = +'_R x y) \land (\forall x \ (+'_R x 0'_R) = x) \land (\forall x \ y \ (+'_R x y = 0'_R) \land (\forall x \ y \ z \ y < z \Rightarrow +'_R x y < +'_R x z) \]

\[ \frac{\vdash}{\vdash} \]

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUAL USR029
\( \wedge 0'_R < 1'_R \)

\[ \sim_R \text{consistent} \]
\[ \vdash \text{Consistent} (\lambda \sim_R \bullet \forall x \bullet x + \sim_R x = 0_R) \]

\[ R_\text{plus_assoc_thm} \]
\[ \vdash \forall x y z \bullet (x + y) + z = x + y + z \]

\[ R_\text{plus_comm_thm} \]
\[ \vdash \forall x y \bullet x + y = y + x \]

\[ R_\text{plus_unit_thm} \]
\[ \vdash \forall x \bullet x + 0_R = x \]

\[ R_\text{plus_mono_thm} \]
\[ \vdash \forall x y z \bullet y < z \Rightarrow x + y < x + z \]

\[ R_\text{plus_assoc_thm1} \]
\[ \vdash \forall x y z \bullet x + y + z = (x + y) + z \]

\[ R_\text{plus_mono_thm1} \]
\[ \vdash \forall x y z \bullet y < z \Rightarrow y + x < z + x \]

\[ R_\text{plus_mono_thm2} \]
\[ \vdash \forall x y z t \bullet x < y \land s < t \Rightarrow x + s < y + t \]

\[ R_\text{plus_0_thm} \]
\[ \vdash \forall x y \bullet x + 0_R = x \land 0_R + x = x \]

\[ R_0_1_thm \]
\[ \vdash 0_R = 0_R \land 1_R = 1_R \]

\[ R_\text{plus_order_thm} \]
\[ \vdash \forall x y z \bullet y + x = x + y \]
\[ \wedge (x + y) + z = x + y + z \]
\[ \wedge y + x + z = x + y + z \]

\[ R_\text{plus_minus_thm} \]
\[ \vdash \forall x y z \bullet x + \sim_R x = 0_R \land \sim_R x + x = 0_R \]

\[ R_\text{eq_thm} \]
\[ \vdash \forall x y \bullet x = y \iff x + \sim_R y = 0_R \]

\[ NR_\text{plus_homomorphism_thm} \]
\[ \vdash \forall m n \bullet NR (m + n) = NR m + NR n \]

\[ R_\text{minus_clauses} \]
\[ \vdash \forall x \bullet \sim (\sim x) = x \]
\[ \wedge x + \sim x = 0_R \]
\[ \wedge \sim x + x = 0_R \]
\[ \wedge \sim (x + y) = \sim x + \sim y \]
\[ \wedge \sim 0_R = 0_R \]

\[ R_\text{minus_eq_thm} \]
\[ \vdash \forall x y \bullet \sim x = \sim y \iff x = y \]

\[ NR_0_less_thm \]
\[ \vdash \forall m \bullet 0_R < NR (m + 1) \]

\[ NR_\text{one_one_thm} \]
\[ \vdash \forall m n \bullet NR m = NR n \iff m = n \]

\[ R_\text{plus_clauses} \]
\[ \vdash \forall x y z \bullet (x + z = y + z \iff x = y) \]
\[ \wedge (z + x = y + z \iff x = y) \]
\[ \wedge (x + z = z + y \iff x = y) \]
\[ \wedge (z + x = z + y \iff x = y) \]
\[ \wedge (x + z = z \iff x = 0_R) \]
\[ \wedge (z + x = z \iff x = 0_R) \]
\[ \wedge (z = z + y \iff y = 0_R) \]
\( (z = y + z \Leftrightarrow y = 0) \)
\( x + 0. = x \)
\( 0. + x = x \)
\( \neg 1. = 0. \)
\( \neg 0. = 1. \)

**R_less_less_0_thm**

\(|\forall x \ y \bullet x < y \Leftrightarrow x + \sim y < 0.\)

**R_less_clauses**

\(|\forall x \ y \ z \bullet (x + z < y + z \Leftrightarrow x < y)\)
\(\land (z + x < y + z \Leftrightarrow x < y)\)
\(\land (x + z < z + y \Leftrightarrow x < y)\)
\(\land (z + x < z + y \Leftrightarrow x < y)\)
\(\land (x + z < z \Leftrightarrow x < 0.)\)
\(\land (z + x < z \Leftrightarrow x < 0.)\)
\(\land (x < z + x \Leftrightarrow 0. < z)\)
\(\land (x < x + z \Leftrightarrow 0. < z)\)
\(\neg x < x\)
\(\land 0. < 1.\)
\(\neg 1. < 0.\)

**R_less_0_less_thm**

\(|\forall x \ y \bullet x < y \Leftrightarrow 0. < y + \sim x\)

**R_\leq_clauses**

\(|\forall x \ y \ z \bullet (x + z \leq y + z \Leftrightarrow x \leq y)\)
\(\land (z + x \leq y + z \Leftrightarrow x \leq y)\)
\(\land (x + z \leq z + y \Leftrightarrow x \leq y)\)
\(\land (z + x \leq z + y \Leftrightarrow x \leq y)\)
\(\land (x + z \leq z \Leftrightarrow x \leq 0.)\)
\(\land (z + x \leq z \Leftrightarrow x \leq 0.)\)
\(\land (x \leq z + x \Leftrightarrow 0. \leq z)\)
\(\land (x \leq x + z \Leftrightarrow 0. \leq z)\)
\(\land x \leq x\)
\(\land 0. \leq 1.\)
\(\land \neg 1. \leq 0.\)

**R_\leq_\leq_0_thm**

\(|\forall x \ y \bullet x \leq y \Leftrightarrow x + \sim y \leq 0.\)

**R_\leq_0_\leq_thm**

\(|\forall x \ y \bullet x \leq y \Leftrightarrow 0. \leq y + \sim x\)

**NR_less_thm**

\(|\forall m \ n \bullet NR \ m < NR \ n \Leftrightarrow m < n\)

**R_less_strong_dense_thm**

\(|\forall x \ y \bullet x < y \Rightarrow (\exists d \bullet 0. < d \land x + d < y)\)

**NR_\leq_thm**

\(|\forall m \ n \bullet NR \ m \leq NR \ n \Leftrightarrow m \leq n\)

**R_sup_plus_thm**

\(|\forall A \ a \ x \bullet \neg A = \{\} \land (\forall x \bullet x \in A \Rightarrow x \leq a)\)
\(\Rightarrow Sup \ A + x = Sup \ {t | \exists a \bullet a \in A \land t < a + x}\)

**R_sup_plus_sup_thm**

\(|\forall A \ a \ B \ b \bullet \neg A = \{\} \land (\forall x \bullet x \in A \Rightarrow x \leq a)\)
\(\land \neg B = \{\} \land (\forall y \bullet y \in B \Rightarrow y \leq b)\)
\(\Rightarrow Sup \ A + Sup \ B\)

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUAL USR029
\[
\begin{align*}
\mathbb{R}\_\text{delta\_induction\_thm} &\quad \vdash \forall \, x \, p \\
&\quad \bullet (\exists \, d) \\
&\quad \bullet 0. < d \\
&\quad \land (\exists \, e) \\
&\quad \bullet d < e \land (\forall \, t \cdot x < t \land t < x + e \Rightarrow p \, t) \\
&\quad \land (\forall \, s \cdot x < s \land p \, s \Rightarrow p \, (s + d)) \\
&\Rightarrow (\forall \, y \cdot x < y \Rightarrow p \, y)
\end{align*}
\]

\[
\begin{align*}
\mathbb{R}\_\text{ord\_pres\_strict\_thm} &\quad \vdash \forall \, f \\
&\quad \bullet (\forall \, x \, y \cdot x < y \Rightarrow f \, x < f \, y) \\
&\Rightarrow (\forall \, x \, y \cdot f \, x < f \, y \Rightarrow x < y)
\end{align*}
\]

\[
\begin{align*}
\mathbb{R}\_\text{add\_hom\_0\_thm} &\quad \vdash \forall \, f \\
&\quad \bullet (\forall \, x \, y \cdot f \, (x + y) = f \, x + f \, y) \Rightarrow (\forall \, x \cdot f \, 0. = 0.)
\end{align*}
\]

\[
\begin{align*}
\mathbb{R}\_\text{add\_hom\_minus\_thm} &\quad \vdash \forall \, f \\
&\quad \bullet (\forall \, x \, y \cdot f \, (x + y) = f \, x + f \, y) \\
&\Rightarrow (\forall \, x \cdot f \, (\sim x) = \sim (f \, x))
\end{align*}
\]

\[
\begin{align*}
\mathbb{R}\_\text{add\_hom\_extension\_thm} &\quad \vdash \forall \, f \\
&\quad \bullet (\forall \, x \, y \cdot 0. \leq x \land 0. \leq y \Rightarrow f \, (x + y) = f \, x + f \, y) \\
&\quad \land (\forall \, x \cdot 0. \leq x \Rightarrow f \, (\sim x) = \sim (f \, x)) \\
&\Rightarrow (\forall \, x \, y \cdot f \, (x + y) = f \, x + f \, y)
\end{align*}
\]

\[
\begin{align*}
\mathbb{R}\_\text{monoid\_delta\_dense\_thm} &\quad \vdash \forall \, G \, d \\
&\quad \bullet 0. \in G \\
&\quad \land (\forall \, g \, h \cdot g \in G \land h \in G \Rightarrow g + h \in G) \\
&\quad \land d \in G \\
&\quad \land 0. < d \\
&\Rightarrow (\forall \, x \\
&\quad \bullet 0. < x \Rightarrow (\exists \, g \cdot g \in G \land g \leq x \land x < g + d))
\end{align*}
\]

\[
\begin{align*}
\mathbb{R}\_\text{monoid\_dense\_thm} &\quad \vdash \forall \, G \\
&\quad \bullet 0. \in G \\
&\quad \land (\forall \, g \, h \cdot g \in G \land h \in G \Rightarrow g + h \in G) \\
&\quad \land (\forall \, x \cdot 0. < x \Rightarrow (\exists \, g \cdot g \in G \land 0. < g \land g < x)) \\
&\Rightarrow (\forall \, x \, y \\
&\quad \bullet 0. < x \land x < y \Rightarrow (\exists \, g \cdot g \in G \land x < g \land g < y))
\end{align*}
\]

\[
\begin{align*}
\mathbb{R}\_\text{subgroup\_dense\_thm} &\quad \vdash \forall \, G \\
&\quad \bullet 0. \in G \\
&\quad \land (\forall \, g \, h \cdot g \in G \land h \in G \Rightarrow g + h \in G) \\
&\quad \land (\forall \, g \cdot g \in G \Rightarrow g \in G) \\
&\quad \land (\forall \, x \cdot 0. < x \Rightarrow (\exists \, g \cdot g \in G \land 0. < g \land g < x)) \\
&\Rightarrow (\forall \, x \, y \\
&\quad \bullet x < y \Rightarrow (\exists \, g \cdot g \in G \land x < g \land g < y))
\end{align*}
\]

\[
\begin{align*}
\mathbb{R}\_\text{semigroup\_dense\_thm} &\quad \vdash \forall \, G \\
&\quad \bullet (\forall \, g \, h \cdot g \in G \land h \in G \Rightarrow g + h \in G) \\
&\quad \land (\forall \, x \cdot 0. < x \Rightarrow (\exists \, g \cdot g \in G \land 0. < g \land g < x))
\end{align*}
\]
\[
\Rightarrow (\forall \, x \, y \quad 0. < x \land x < y \Rightarrow (\exists \, g \bullet g \in G \land x < g \land g < y))
\]

**R_add_hom_image_group_thm**

\[\vdash \forall \, f \, I
\quad \bullet (\forall \, x \, y \bullet f \,(x + y) = f \, x + f \, y)\]
\[\land \, I = \{b|\exists \, a \bullet b = f \, a}\]
\[\Rightarrow 0. \in I\]
\[\land (\forall \, g \, h \bullet g \in I \land h \in I \Rightarrow g + h \in I)\]
\[\land (\forall \, g \bullet g \in I \Rightarrow \sim g \in I)\]

**R_add_hom_kernel_group_thm**

\[\vdash \forall \, f \, K
\quad \bullet (\forall \, x \, y \bullet f \,(x + y) = f \, x + f \, y)\]
\[\land (\forall \, x \bullet 0. < x \Rightarrow 0. < f \, x)\]
\[\Rightarrow (\forall \, x \, y \bullet x < y \Rightarrow f \, x < f \, y)\]

**R_opah_strict_thm**

\[\vdash \forall \, f
\quad \bullet (\forall \, x \, y \bullet f \,(x + y) = f \, x + f \, y)\]
\[\land (\forall \, x \bullet 0. < x \Rightarrow 0. < f \, x)\]
\[\Rightarrow (\forall \, x \, y \bullet f \, x < f \, y \Rightarrow x < y)\]

**R_opah_one_one_thm**

\[\vdash \forall \, f
\quad \bullet (\forall \, x \, y \bullet f \,(x + y) = f \, x + f \, y)\]
\[\land (\forall \, x \bullet 0. < x \Rightarrow 0. < f \, x)\]
\[\Rightarrow \text{OneOne } f\]

**R_opah_dense_image_thm**

\[\vdash \forall \, f \, e
\quad \bullet (\forall \, x \, y \bullet f \,(x + y) = f \, x + f \, y)\]
\[\land (\forall \, x \bullet 0. < x \Rightarrow 0. < f \, x)\]
\[\land 0. < e\]
\[\Rightarrow (\exists \, d \bullet 0. < d \land f \, d < e)\]

**R_opah_onto_thm**

\[\vdash \forall \, f
\quad \bullet (\forall \, x \, y \bullet f \,(x + y) = f \, x + f \, y)\]
\[\land (\forall \, x \bullet 0. < x \Rightarrow 0. < f \, x)\]
\[\Rightarrow \text{Onto } f\]

**R_opah_inverse_add_hom_thm**

\[\vdash \forall \, f \, g
\quad \bullet (\forall \, x \, y \bullet f \,(x + y) = f \, x + f \, y)\]
\[\land (\forall \, x \bullet 0. < x \Rightarrow 0. < f \, x)\]
\[\land (\forall \, x \bullet g \,(f \, x) = x)\]
\[\land (\forall \, x \bullet f \,(g \, x) = x)\]
\[\Rightarrow (\forall \, x \, y \bullet g \,(x + y) = g \, x + g \, y)\]
\[\land (\forall \, x \bullet 0. < x \Rightarrow 0. < g \, x)\]

**R_opah_inverse_thm**

\[\vdash \forall \, f
\quad \bullet (\forall \, x \, y \bullet f \,(x + y) = f \, x + f \, y)\]
\[\forall x \bullet 0 < x \Rightarrow 0 < f x\]
\[\Rightarrow (\exists g\]
\[\bullet (\forall x \bullet g (f x) = x)\]
\[\land (\forall x \bullet f (g x) = x)\]
\[\land (\forall x y \bullet g (x + y) = g x + g y)\]
\[\land (\forall x \bullet 0 < x \Rightarrow 0 < g x)\]

\[\mathbb{R}_{\text{copah.id.thm}}\]
\[\vdash \exists \iota\]
\[\bullet (\forall x \bullet \iota x = x)\]
\[\land (\forall x y \bullet \iota (x + y) = \iota x + \iota y)\]
\[\land (\forall x \bullet 0 < x \Rightarrow 0 < \iota x)\]
\[\land (\forall f\]
\[\bullet (\forall x y \bullet f (x + y) = f x + f y)\]
\[\Rightarrow (\forall x \bullet \iota (f x) = f (\iota x))\]

\[\mathbb{R}_{\text{copah.double.thm}}\]
\[\vdash \exists \alpha\]
\[\bullet (\forall x \bullet \alpha x = x + x)\]
\[\land (\forall x y \bullet \alpha (x + y) = \alpha x + \alpha y)\]
\[\land (\forall x \bullet 0 < x \Rightarrow 0 < \alpha x)\]
\[\land (\forall x \bullet 0 < x \Rightarrow \alpha x < x)\]
\[\land (\forall f\]
\[\bullet (\forall x y \bullet f (x + y) = f x + f y)\]
\[\Rightarrow (\forall x \bullet \alpha (f x) = f (\alpha x))\]

\[\mathbb{R}_{\text{copah.halve.thm}}\]
\[\vdash \exists \beta\]
\[\bullet (\forall x \bullet \beta x + \beta x = x)\]
\[\land (\forall x y \bullet \beta (x + y) = \beta x + \beta y)\]
\[\land (\forall x \bullet 0 < x \Rightarrow 0 < \beta x)\]
\[\land (\forall x \bullet 0 < x \Rightarrow \beta x < x)\]
\[\land (\forall f\]
\[\bullet (\forall x y \bullet f (x + y) = f x + f y)\]
\[\Rightarrow (\forall x \bullet \beta (f x) = f (\beta x))\]

\[\mathbb{R}_{\text{copah.comp.thm}}\]
\[\vdash \forall \alpha \beta\]
\[\bullet (\forall x y \bullet \alpha (x + y) = \alpha x + \alpha y)\]
\[\land (\forall x \bullet 0 < x \Rightarrow 0 < \alpha x)\]
\[\land (\forall f\]
\[\bullet (\forall x y \bullet f (x + y) = f x + f y)\]
\[\Rightarrow (\forall x \bullet \alpha (f x) = f (\alpha x))\]
\[\land (\forall x y \bullet \beta (x + y) = \beta x + \beta y)\]
\[\land (\forall x \bullet 0 < x \Rightarrow 0 < \beta x)\]
\[\land (\forall f\]
\[\bullet (\forall x y \bullet f (x + y) = f x + f y)\]
\[\Rightarrow (\forall x \bullet \beta (f x) = f (\beta x))\]
\[\Rightarrow (\exists \gamma\]
\[\bullet (\forall x \bullet \gamma x = \alpha (\beta x))\]
\[\land (\forall x \bullet 0 < x \Rightarrow 0 < \gamma x)\]
\[\land (\forall x y \bullet \gamma (x + y) = \gamma x + \gamma y)\]
\[\land (\forall f\]
\[\bullet (\forall x y \bullet f (x + y) = f x + f y)\]
\[\Rightarrow (\forall x \bullet \gamma (f x) = f (\gamma x))\)
\(\text{R\_copah\_sum\_thm}\)

\[\vdash \forall \alpha \beta \to (\forall x y \bullet \alpha (x + y) = \alpha x + \alpha y) \land (\forall x \bullet 0. < x \Rightarrow 0. < \alpha x) \land (\forall f \bullet (\forall x y \bullet f (x + y) = f x + f y) \Rightarrow (\forall x \bullet \alpha (f x) = f (\alpha x))) \land (\forall x y \bullet \beta (x + y) = \beta x + \beta y) \land (\forall x \bullet 0. < x \Rightarrow 0. < \beta x) \land (\forall f \bullet (\forall x y \bullet f (x + y) = f x + f y) \Rightarrow (\forall x \bullet \beta (f x) = f (\beta x))) \Rightarrow (\exists \gamma \bullet (\forall x \bullet \gamma x = \alpha x + \beta x) \land (\forall x \bullet 0. < x \Rightarrow 0. < \gamma x) \land (\forall x y \bullet \gamma (x + y) = \gamma x + \gamma y) \land (\forall f \bullet (\forall x y \bullet f (x + y) = f x + f y) \Rightarrow (\forall x \bullet \gamma (f x) = f (\gamma x))))\]

\(\text{R\_halve\_closed\_dense\_thm}\)

\[\vdash \forall A e \to (0. < e \land e \in A \land (\forall y \bullet y \in A \Rightarrow (\exists z \bullet z \in A \land z + z = y)) \Rightarrow (\forall d \bullet 0. < d \Rightarrow (\exists a \bullet a \in A \land 0. < a \land a < d))\]

\(\text{R\_copah\_dense\_thm}\)

\[\vdash \forall d x y \to (0. < d \land 0. < x \land x < y) \Rightarrow (\exists \gamma \bullet (\forall x y \bullet \gamma (x + y) = \gamma x + \gamma y) \land (\forall x \bullet 0. < x \Rightarrow 0. < \gamma x) \land (\forall f \bullet (\forall x y \bullet f (x + y) = f x + f y) \Rightarrow (\forall x \bullet \gamma (f x) = f (\gamma x)))) \land x < \gamma d \land \gamma d < y)\]

\(\text{R\_opah\_extension\_thm1}\)

\[\vdash \forall f \bullet (\forall x y \bullet 0. < x \land 0. < y \Rightarrow f (x + y) = f x + f y) \land (\forall x \bullet 0. < x \Rightarrow 0. < f x) \Rightarrow (\exists \phi \bullet \phi 0. = 0. \land (\forall x \bullet 0. < x \Rightarrow \phi x = f x) \land (\forall x y \bullet 0. \leq x \land 0. \leq y \Rightarrow \phi (x + y) = \phi x + \phi y))\]

\(\text{R\_opah\_extension\_thm2}\)

\[\vdash \forall f \bullet f 0. = 0. \land (\forall x y \bullet 0. \leq x \land 0. \leq y \Rightarrow f (x + y) = f x + f y) \Rightarrow (\exists \psi)\]
• \((\forall x \bullet 0. \leq x \Rightarrow \psi x = f x)\)
  \& \((\forall x y \bullet \psi (x + y) = \psi x + \psi y))\)

**R_opah_extension_thm**

\[\vdash \forall f\]

\[\bullet (\forall x y \bullet 0. < x \land 0. < y \Rightarrow f (x + y) = f x + f y)\]
\& \((\forall x \bullet 0. < x \Rightarrow 0. < f x)\)
\[\Rightarrow (\exists \phi \bullet (\forall x y \bullet \phi (x + y) = \phi x + \phi y)\]
\& \((\forall x \bullet 0. < x \Rightarrow 0. < \phi x))\)

**R_opah_order_thm**

\[\vdash \forall f g d\]

\[\bullet (\forall x y \bullet f (x + y) = f x + f y)\]
\& \((\forall x \bullet 0. < x \Rightarrow 0. < f x)\)
\& \((\forall x y \bullet g (x + y) = g x + g y)\)
\& \((\forall x \bullet 0. < x \Rightarrow 0. < g x)\)
\& \(0. < d\)
\& \(f d = g d\)
\[\Rightarrow (\forall x \bullet 0. < x \Rightarrow f x < g x)\)

**R_opah_eq_thm**

\[\vdash \forall f g d\]

\[\bullet (\forall x y \bullet f (x + y) = f x + f y)\]
\& \((\forall x \bullet 0. < x \Rightarrow 0. < f x)\)
\& \((\forall x y \bullet g (x + y) = g x + g y)\)
\& \((\forall x \bullet 0. < x \Rightarrow 0. < g x)\)
\& \(0. < d\)
\& \(f d = g d\)
\[\Rightarrow f = g\)

**R_opah_complete_thm**

\[\vdash \forall d e\]

\[\bullet 0. < d \land 0. < e\]
\[\Rightarrow (\exists f \bullet (\forall x \bullet f (x + y) = f x + f y)\]
\& \((\forall x \bullet 0. < x \Rightarrow 0. < f x)\)
\& \(f d = e\)

***R-consistent**

\[\vdash Consistent\]

\((\lambda \ast'_{R})\)

\[\bullet (\forall x y \bullet (\forall y \bullet \ast'_{R} x y z) z = \ast'_{R} x (\ast'_{R} y z))\]
\& \((\forall x y \bullet \ast'_{R} x 1. = x)\)
\& \((\forall x y \bullet (\forall y \bullet \ast'_{R} x y z) z = \ast'_{R} x (\ast'_{R} y z))\)
\& \((\forall x y \bullet \ast'_{R} x 1. = x)\)
\& \((\forall x y \bullet 0. < x \land 0. < y \Rightarrow 0. < \ast'_{R} x y)\)

**R_times_assoc_thm**

\[\vdash \forall x y z \bullet (x \ast y) \ast z = x \ast (y \ast z)\)

**R_times_comm_thm**

\[\vdash \forall x y \bullet x \ast y = y \ast x\)

**R_times_unit_thm**

\[\vdash \forall x \bullet x \ast 1. = x\)
\( \mathbb{R} \_0 \_less \_0 \_less \_times \_thm \)

\[ \vdash \forall x \ y \bullet \ 0. < x \land 0. < y \Rightarrow 0. < x \ast y \]

\( \mathbb{R} \_times \_assoc \_thm1 \)

\[ \vdash \forall x \ y \ z \bullet x \ast y \ast z = (x \ast y) \ast z \]

\( \mathbb{R} \_times \_plus \_distrib \_thm \)

\[ \vdash \forall x \ y \ z \\
\bullet x \ast (y + z) = x \ast y + x \ast z \\
\land (x + y) \ast z = x \ast z + y \ast z \]

\( \mathbb{R} \_times \_order \_thm \)

\[ \vdash \forall x \ y \ z \\
\bullet y \ast x = x \ast y \\
\land (x \ast y) \ast z = x \ast y \ast z \\
\land y \ast x \ast z = x \ast y \ast z \]

\( \mathbb{R} \_times \_0 \_thm \)

\[ \vdash \forall x \bullet x \ast 0. = 0. \land 0. \ast x = 0. \]

\( \mathbb{R} \_times \_1 \_thm \)

\[ \vdash \forall x \bullet x \ast 1. = x \land 1. \ast x = x \]

\( \mathbb{NR} \_times \_homomorphism \_thm \)

\[ \vdash \forall m \ n \bullet \mathbb{NR} \ (m \ast n) = \mathbb{NR} \ m \ast \mathbb{NR} \ n \]

\( \mathbb{R} \_times \_minus \_thm \)

\[ \vdash \forall x \ y \\
\bullet \sim x \ast y = \sim (x \ast y) \\
\land x \ast \sim y = \sim (x \ast y) \\
\land \sim x \ast \sim y = x \ast y \\
/\_R \_consistent \]

\[ \vdash Consistent \\
(\lambda \ /_R \\
\bullet (\forall y \ z \bullet \sim z = 0. \Rightarrow /_R (y \ast z) = y) \\
\land (\forall x \ y \ z \\
\bullet \sim z = 0. \Rightarrow /_R (x \ast y) = x \ast /_R y z)) \]

\( \mathbb{R} \_over \_times \_recip \_thm \)

\[ \vdash \forall z \bullet \sim z = 0. \Rightarrow (\forall x \bullet x / z = x \ast z \^{-1}) \]

\( \mathbb{R} \_times \_recip \_thm \)

\[ \vdash \forall z \bullet \sim z = 0. \Rightarrow z \ast z \^{-1} = 1. \]

\( \mathbb{R} \_eq \_recip \_thm \)

\[ \vdash \forall z \bullet \sim z = 0. \Rightarrow (\forall y \bullet y = z \iff y \ast z \^{-1} = 1.) \]

\( \mathbb{R} \_times \_cancel \_thm \)

\[ \vdash \forall x \ y \ z \bullet \sim z = 0. \Rightarrow (x \ast z = y \ast z \iff x = y) \]

\( \mathbb{R} \_times \_eq \_0 \_thm \)

\[ \vdash \forall x \ y \bullet x \ast y = 0. \iff x = 0. \lor y = 0. \]

\( \mathbb{R} \_times \_clauses \)

\[ \vdash \forall x \\
\bullet 0. \ast x = 0. \land x \ast 0. = 0. \land x \ast 1. = x \land 1. \ast x = x \]

\( \mathbb{R} \_times \_mono \_\Rightarrow \_thm \)

\[ \vdash \forall x \bullet 0. < x \Rightarrow (\forall y \bullet y < z \iff x \ast y < x \ast z) \]

\( \mathbb{R} \_times \_mono \_thm \)

\[ \vdash \forall x \ y \ z \bullet 0. < x \land y < z \Rightarrow x \ast y < x \ast z \]

\( \mathbb{R} \_0 \_\leq \_0 \_\leq \_times \_thm \)

\[ \vdash \forall x \ y \bullet 0. \leq x \land 0. \leq y \Rightarrow 0. \leq x \ast y \]

\( \mathbb{R} \_\sim \_recip \_0 \_thm \)

\[ \vdash \forall z \bullet \sim z = 0. \Rightarrow \sim z \^{-1} = 0. \]

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUAL USR029
\( \mathbb{R} \_\text{recip clauses} \)
\[
\vdash (1. -1 = 1. \\
\quad \land (\forall w \\
\quad \quad \bullet \neg w = 0. \\
\quad \quad \Rightarrow w^{-1} -1 = w \\
\quad \quad \land w \ast w^{-1} = 1. \\
\quad \quad \land w^{-1} \ast w = 1.)) \\
\land (\forall w z \\
\quad \bullet \neg w = 0. \land \neg z = 0. \\
\quad \Rightarrow (w \ast z)^{-1} = w^{-1} \ast z^{-1})
\]

\( \mathbb{R} \_\text{cross mult_eq_thm} \)
\[
\vdash \forall w z \\
\quad \bullet \neg w = 0. \land \neg z = 0. \\
\quad \Rightarrow (\forall x y \bullet x / w = y / z \iff x \ast z = w \ast y)
\]

\( \mathbb{R} \_\text{less _eq_0_thm} \)
\[
\vdash \forall z \bullet 0. < z \Rightarrow \neg z = 0.
\]

\( \mathbb{R} \_\text{0_less_0_recip_thm} \)
\[
\vdash \forall z \bullet 0. < z \Rightarrow 0. < z^{-1}
\]

\( \mathbb{R} \_\text{cross mult less thm} \)
\[
\vdash \forall w z \\
\quad \bullet 0. < w \land 0. < z \\
\quad \Rightarrow (\forall x y \bullet x / w < y / z \iff x \ast z < w \ast y)
\]

\( \mathbb{R} \_\text{over cancel_eq_thm} \)
\[
\vdash \forall w z \\
\quad \bullet \neg w = 0. \land \neg z = 0. \\
\quad \Rightarrow (\forall x y \bullet x / w = y / z \iff x \ast z < w \ast y)
\]

\( \mathbb{R} \_\text{over plus_over_thm} \)
\[
\vdash \forall x y u v \\
\quad \bullet \neg u = 0. \land \neg v = 0. \\
\quad \Rightarrow (\forall x \bullet (x \ast z) / (w \ast z) = x / w)
\]

\( \mathbb{R} \_\text{0_over_thm} \)
\[
\vdash \forall z \bullet \neg z = 0. \Rightarrow 0. / z = 0.
\]

\( \mathbb{R} \_\text{over 1_thm} \)
\[
\vdash \forall x \bullet x / 1. = x
\]

\( \text{NR\_plus_homomorphism_thm1} \)
\[
\vdash \forall m n \bullet \text{NR} m + \text{NR} n = \text{NR} (m + n)
\]

\( \text{NR\_times_homomorphism_thm1} \)
\[
\vdash \forall m n \bullet \text{NR} m \ast \text{NR} n = \text{NR} (m \ast n)
\]

\( \text{R\_frac_cross_mult_eq_thm} \)
\[
\vdash \forall m n i j \\
\quad \bullet i / (m + 1) = j / (n + 1) \\
\quad \iff i \ast (n + 1) = j \ast (m + 1)
\]

\( \text{R\_frac_cancel_eq_thm} \)
\[
\vdash \forall i m n \\
\quad \bullet (i \ast (n + 1)) / ((m + 1) \ast (n + 1)) = i / (m + 1)
\]

\( \text{R\_frac_0_thm} \)
\[
\vdash \forall m \bullet 0 / (m + 1) = 0.
\]

\( \text{R\_frac_N_thm} \)
\[
\vdash \forall i \bullet i / 1 = \text{NR} i
\]

\( \text{R\_frac_plus_frac_thm} \)
\[
\vdash \forall i j k m n \\
\quad \bullet i / (m + 1) + j / (n + 1) \\
\quad = (i \ast (n + 1) + j \ast (m + 1)) / ((m + 1) \ast (n + 1))
\]

\( \text{R\_frac_minus_frac_thm} \)
\(\forall i \ j \ k \ m \ n\)
\[\bullet j \cdot (m + 1) \leq i \cdot (n + 1)\]
\[\Rightarrow i \cdot (m + 1) + \sim \frac{(j \cdot (m + 1))}{(n + 1)}\]
\[= (i \cdot (n + 1) - j \cdot (m + 1)) / ((m + 1) \cdot (n + 1))\]

\(\mathbb{R}_{\_\text{frac\_minus\_frac\_thm1}}\)
\(\vdash \forall i \ j \ m \ n\)
\[\bullet \frac{i \cdot (n + 1)}{(m + 1)} \leq \frac{j \cdot (m + 1)}{(n + 1)}\]
\[\Rightarrow i \cdot (m + 1) + \sim \frac{(j \cdot (m + 1))}{(n + 1)}\]
\[= \sim \frac{(j \cdot (m + 1) - i \cdot (n + 1))}{((m + 1) \cdot (n + 1))}\]

\(\mathbb{R}_{\_\over\_times\_\over\_thm}\)
\(\vdash \forall x \ y \ u \ v\)
\[\bullet \sim u = 0 \wedge \sim v = 0\]
\[\Rightarrow \frac{x}{u \cdot y} / \frac{v}{u \cdot v} = \frac{x \cdot y}{u \cdot v}\]

\(\mathbb{R}_{\_\text{frac\_times\_frac}\_thm}\)
\(\vdash \forall i \ j \ m \ n\)
\[\bullet \frac{i}{(m + 1)} \leq \frac{j}{(n + 1)}\]
\[\Rightarrow \frac{i}{(m + 1) \cdot (n + 1)} = \frac{i \cdot j}{((m + 1) \cdot (n + 1))}\]

\(\mathbb{R}_{\_\over\_recip\_thm}\)
\(\vdash \forall u \ v \sim u = 0 \wedge \sim v = 0\)
\[\Rightarrow \frac{u}{v} = \frac{(u \cdot v)}{(u \cdot v)}\]

\(\mathbb{R}_{\_\text{frac\_recip\_thm}\_thm}\)
\(\vdash \forall m \ n\)
\[\bullet \frac{(m + 1)}{(n + 1)} \leq \frac{(m + 1)}{(n + 1)}\]
\[\Rightarrow \sim \frac{(m + 1)}{(n + 1)} = \frac{(m + 1)}{(n + 1)}\]

\(\mathbb{R}_{\_\text{over\_eq\_0\_thm}\_thm}\)
\(\vdash \forall u \ v \sim u = 0 \wedge \sim v = 0\)
\[\Rightarrow \sim u / v = 0\]

\(\mathbb{R}_{\_\over\_over\_over\_thm}\)
\(\vdash \forall x \ y \ u \ v\)
\[\bullet \sim u = 0 \wedge \sim v = 0 \wedge \sim y = 0\]
\[\Rightarrow \frac{x}{u} / \frac{y}{v} = \frac{x \cdot y}{u \cdot v}\]

\(\mathbb{R}_{\_\text{frac\_less\_frac\_thm}\_thm}\)
\(\vdash \forall i \ j \ m \ n\)
\[\bullet \frac{i}{(m + 1)} < \frac{j}{(n + 1)}\]
\[\Leftrightarrow \frac{i}{(n + 1)} < \frac{j}{(m + 1)}\]

\(\mathbb{R}_{\_\text{minus\_frac\_less\_frac\_thm}\_thm}\)
\(\vdash \forall i \ j \ m \ n\)
\[\sim (i \cdot (m + 1)) < \frac{j}{(n + 1)} \Leftrightarrow 0 < i + j\]

\(\mathbb{R}_{\_\text{frac\_less\_minus\_frac\_thm}\_thm}\)
\(\vdash \forall i \ j \ m \ n\)
\[\sim i / (m + 1) < j / (n + 1)\]
\[\Leftrightarrow i \cdot (n + 1) < j \cdot (m + 1)\]

\(\mathbb{R}_{\_0\_\leq\_\text{frac}\_thm}\)
\(\vdash \forall i \ m \)
\[\bullet 0 \leq i / (m + 1)\]

\(\mathbb{R}_{\_\text{abs\_frac\_thm}\_thm}\)
\(\vdash \forall i \ m\)
\[\bullet \text{Abs} \cdot (i / (m + 1)) = i / (m + 1)\]

\(\mathbb{R}_{\_\text{abs\_minus\_thm}\_thm}\)
\(\vdash \forall x \bullet \text{Abs} \cdot (\sim x) = \text{Abs} x\)

\(\text{Max}_{R}_{\_\text{consistent}}\)
\(\vdash \text{Consistent}\)
\[\left(\lambda \text{Max}_{R}\right)\]
\[\bullet (\forall x \bullet \text{Max}_{R} \cdot [x] = x)\]
\[\wedge (\forall x \ y \ L)\]
• \( \text{Max}'_R(\text{Cons } x \text{ (Cons } y \ L)) \)
  
  \[
  \begin{align*}
  &= (\text{if } x < \text{Max}'_R(\text{Cons } y \ L) \\
  &\quad \text{then } \text{Max}'_R(\text{Cons } y \ L) \\
  &\quad \text{else } x))
  \end{align*}
  \]

\textbf{Min}_{R\text{-consistent}}

\( \vdash \text{Consistent} \)

\[
(\lambda \text{Min}'_R
\begin{align*}
  &\bullet (\forall x \bullet \text{Min}'_R[x] = x) \\
  &\quad \land (\forall x \ y \ L \\
  &\bullet \text{Min}'_R(\text{Cons } x \text{ (Cons } y \ L)) \\
  &= (\text{if } x > \text{Min}'_R(\text{Cons } y \ L) \\
  &\quad \text{then } \text{Min}'_R(\text{Cons } y \ L) \\
  &\quad \text{else } x))
  \end{align*}
  \]

\textbf{R\text{-max-cons.thm}}

\( \vdash \forall x \ L \)

\[
\begin{align*}
  &\bullet \text{Max}_R(\text{Cons } x \ L) \\
  &= (\text{if } L = [] \\
  &\quad \text{then } x \\
  &\quad \text{else if } x < \text{Max}_R L \\
  &\quad \text{then } \text{Max}_R L \\
  &\quad \text{else } x)
  \end{align*}
  \]

\textbf{R\text{-max-conv.thm}}

\( \vdash \forall x \ y \ L \)

\[
\begin{align*}
  &\bullet \text{Max}_R(\text{Cons } x \text{ (Cons } y \ L)) \\
  &= (\text{if } x < y \\
  &\quad \text{then } \text{Max}_R(\text{Cons } y \ L) \\
  &\quad \text{else } \text{Max}_R(\text{Cons } x \ L))
  \end{align*}
  \]

\textbf{R\text{-min-cons.thm}}

\( \vdash \forall x \ L \)

\[
\begin{align*}
  &\bullet \text{Min}_R(\text{Cons } x \ L) \\
  &= (\text{if } L = [] \\
  &\quad \text{then } x \\
  &\quad \text{else if } \text{Min}_R L < x \\
  &\quad \text{then } \text{Min}_R L \\
  &\quad \text{else } x)
  \end{align*}
  \]

\textbf{R\text{-min-conv.thm}}

\( \vdash \forall x \ y \ L \)

\[
\begin{align*}
  &\bullet \text{Min}_R(\text{Cons } x \text{ (Cons } y \ L)) \\
  &= (\text{if } x < y \\
  &\quad \text{then } \text{Min}_R(\text{Cons } x \ L) \\
  &\quad \text{else } \text{Min}_R(\text{Cons } y \ L))
  \end{align*}
  \]

\textbf{\textsuperscript{\textvote}Z\text{-consistent}}

\( \vdash \text{Consistent} \)

\[
(\lambda \textsuperscript{\textvote}Z
\begin{align*}
  &\bullet \forall x \ m \\
  &\bullet \textsuperscript{\textvote}Z x (\text{NZ } m) = x \textsuperscript{\textvote}Z m \\
  &\quad \land \textsuperscript{\textvote}Z x (\sim (\text{NZ } (m + 1))) \\
  &= (x \textsuperscript{\textvote}(m + 1)) \textsuperscript{\textvote}-1)
  \end{align*}
  \]

\textbf{ZR\text{-consistent}}

\( \vdash \text{Consistent} \)

\[
(\lambda \text"ZR"'
\]

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
• $Z(R^n)(NZ 0) = 0$.
  \[\land Z(R^n)(NZ 1) = 1.\]
  \[\land (\forall i j\quad Z(R^n)(i + j) = Z(R^n)(i) + Z(R^n)(j))\]

$Z(R)_{\text{plus homomorphism thm}}$

\[\vdash (\forall i j\cdot Z(R)(i + j) = Z(R)(i) + Z(R)(j))\]

$Z(R)_{\text{minus thm}}$

\[\vdash (\forall i\cdot Z(R)(\sim i) = \sim (Z(R)(i))\]

$Z(R)_{\text{NZ thm}}$

\[\vdash (\forall m\cdot Z(R)(NZ m) = NR(m) \land Z(R)(\sim (NZ m)) = \sim (NR(m))\]

$Z(R)_{\text{times homomorphism thm}}$

\[\vdash (\forall i j\cdot Z(R)(i * j) = Z(R)(i) * Z(R)(j))\]
9.1.23 The Theory $\mathbb{N}$

9.1.23.1 Parents

\[ \text{pair} \]

9.1.23.2 Children

\[ \text{list} \]

9.1.23.3 Constants

\begin{align*}
\text{Is}_\mathbb{N}.\text{Rep} & : \text{IND} \to \text{BOOL} \\
\text{Suc} & : \mathbb{N} \to \mathbb{N} \\
\text{Zero} & : \mathbb{N} \\
\$+ & : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\
\$\leq & : \mathbb{N} \to \mathbb{N} \to \text{BOOL} \\
\$\geq & : \mathbb{N} \to \mathbb{N} \to \text{BOOL} \\
\$\lt & : \mathbb{N} \to \mathbb{N} \to \text{BOOL} \\
\$\gt & : \mathbb{N} \to \mathbb{N} \to \text{BOOL} \\
\$\ast & : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\
\$\text{Mod} & : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\
\$\text{Div} & : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\
\$\text{–} & : \mathbb{N} \to \mathbb{N} \to \mathbb{N}
\end{align*}

9.1.23.4 Types

\[ \mathbb{N} \]

9.1.23.5 Fixity

\begin{align*}
\text{Left Infix 305:} & \\
\text{Left Infix 315:} & \text{Div} \quad \text{Mod} \\
\text{Right Infix 210:} & \lt \quad \gt \quad \leq \quad \geq \\
\text{Right Infix 300:} & + \\
\text{Right Infix 310:} & \ast
\end{align*}
9.1.23.6 Definitions

\textbf{Is\_N\_Rep}
\[ \vdash \exists \text{zero suc} \]
\[ \quad \bullet (\text{Is\_N\_Rep zero} \]
\[ \quad \quad \land (\forall n \bullet \text{Is\_N\_Rep n} \Rightarrow \text{Is\_N\_Rep (suc n)}) \]
\[ \quad \land (\forall n \bullet \text{Is\_N\_Rep n} \Rightarrow \neg \text{suc n} = \text{zero}) \]
\[ \quad \land \text{OneOne suc} \]
\[ \quad \land (\forall p \]
\[ \quad \quad \bullet p \text{ zero} \land (\forall m \bullet p m \Rightarrow p (\text{suc m})) \]
\[ \quad \quad \Rightarrow (\forall n \bullet \text{Is\_N\_Rep n} \Rightarrow p n) \]
\[ \text{N} \]
\[ \vdash \exists f \bullet \text{TypeDefn Is\_N\_Rep f} \]
\[ \text{Zero} \]
\[ \text{Suc} \]
\[ \vdash (\forall n \bullet \neg \text{Suc n} = \text{Zero}) \]
\[ \land \text{OneOne Suc} \]
\[ \land (\forall p \]
\[ \quad \bullet p \text{ Zero} \land (\forall m \bullet p m \Rightarrow p (\text{Suc m})) \Rightarrow (\forall n \bullet p n) \]
\[ \text{+} \]
\[ \vdash \forall m n \]
\[ \quad \bullet 0 + n = n \]
\[ \quad \land (m + 1) + n = (m + n) + 1 \]
\[ \land \text{Suc m} = m + 1 \]
\[ \leq \]
\[ \vdash \forall m n \bullet m \leq n \iff (\exists i \bullet m + i = n) \]
\[ \geq \]
\[ \vdash \forall m n \bullet m \geq n \iff n \leq m \]
\[ \lt \]
\[ \vdash \forall m n \bullet m < n \iff m + 1 \leq n \]
\[ \gt \]
\[ \vdash \forall m n \bullet m > n \iff n < m \]
\[ \ast \]
\[ \vdash \forall m n \bullet 0 \ast n = 0 \land (m + 1) \ast n = m \ast n + n \]
\[ \text{Mod} \]
\[ \text{mod\_def} \]
\[ \vdash \forall m n \]
\[ \quad \bullet 0 < n \]
\[ \quad \Rightarrow 0 \text{ Mod n} = 0 \]
\[ \quad \land (m + 1) \text{ Mod n} \]
\[ \quad = (\text{if } m \text{ Mod n} + 1 < n \]
\[ \quad \text{then } m \text{ Mod n} + 1 \]
\[ \quad \text{else } 0) \]
\[ \text{Div} \]
\[ \text{div\_def} \]
\[ \vdash \forall m n \]
\[ \quad \bullet 0 < n \]
\[ \quad \Rightarrow 0 \text{ Div n} = 0 \]
\[ \quad \land (m + 1) \text{ Div n} \]
\[ \quad = (\text{if } m \text{ Mod n} + 1 < n \]
\[ \quad \text{then } m \text{ Div n} \]
\[ \quad \text{else } m \text{ Div n} + 1) \]
\[ \text{−} \]
\[ \text{minus\_def} \]
\[ \vdash \forall m n \bullet (m + n) - n = m \]
Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029

9.1. Theory Listings

9.1.23.7 Theorems

induction_thm
\[ \forall p \bullet p \ 0 \land (\forall m \bullet p \ m \Rightarrow p \ (m + 1)) \Rightarrow (\forall n \bullet p \ n) \]

\neg_{\text{plus1_thm}}
\[ \forall n \bullet \neg n + 1 = 0 \]

\text{one_one_plus1_thm}
\[ \forall x1 \ x2 \bullet x1 + 1 = x2 + 1 \Rightarrow x1 = x2 \]

\text{prim_rec_thm}
\[ \forall z \ s \bullet \exists t \ f \bullet f \ 0 = z \land (\forall n \bullet f \ (n + 1) = s \ (f \ n)) \]

\text{plus_assoc_thm}
\[ \forall i \ m \ n \bullet (i + m) + n = i + m + n \]

\text{plus_assoc_thm1}
\[ \forall i \ m \ n \bullet i + m + n = (i + m) + n \]

\text{plus_comm_thm}
\[ \forall m \ n \bullet m + n = n + m \]

\text{plus_order_thm}
\[ \forall i \ m \ n \]
\[ \bullet m + i = i + m \]
\[ \land (i + m) + n = i + m + n \]
\[ \land m + i + n = i + m + n \]

\text{plus_clauses}
\[ \forall m \ n \ i \]
\[ \bullet (m + i = n + i \Leftrightarrow m = n) \]
\[ \land (i + m = n + i \Leftrightarrow m = n) \]
\[ \land (m + i = i + n \Leftrightarrow m = n) \]
\[ \land (i + m = i + n \Leftrightarrow m = n) \]
\[ \land (m + i = i \Leftrightarrow m = 0) \]
\[ \land (i + m = i \Leftrightarrow m = 0) \]
\[ \land (i = i + n \Leftrightarrow n = 0) \]
\[ \land (i = n + i \Leftrightarrow n = 0) \]
\[ \land (m + i = 0 \Leftrightarrow m = 0 \land i = 0) \]
\[ \land (0 = m + i \Leftrightarrow m = 0 \land i = 0) \]
\[ \land (m + 0 = m \land 0 + m = m) \]
\[ \land \neg 1 = 0 \]
\[ \land \neg 0 = 1 \]

\text{\leq_{trans_thm}}
\[ \forall m \ n \ i \bullet m \leq i \land i \leq n \Rightarrow m \leq n \]

\text{less_trans_thm}
\[ \forall m \ n \ i \bullet m < i \land i < n \Rightarrow m < n \]

\text{\leq_{clauses}}
\[ \forall m \ n \ i \]
\[ \bullet (m + i \leq n + i \Leftrightarrow m \leq n) \]
\[ \land (i + m \leq n + i \Leftrightarrow m \leq n) \]
\[ \land (m + i \leq i + n \Leftrightarrow m \leq n) \]
\[ \land (i + m \leq i + n \Leftrightarrow m \leq n) \]
\[ \land (m + i \leq i \Leftrightarrow m = 0) \]
\[ \land (i + m \leq i \Leftrightarrow m = 0) \]
\[ \land (m + i \leq 0 \Leftrightarrow m = 0 \land i = 0) \]
\[ \land (m \leq 0 \Leftrightarrow m = 0) \]
\[ \land m \leq m + i \]
\[ \land m \leq i + m \]
\[ \land m \leq m \]
\[ \land 0 \leq m \]
\[ \land \neg 1 \leq 0 \]

\text{less_clauses}
\[ \forall m \ n \ i \]
\[ \bullet (m + i < n + i \Leftrightarrow m < n) \]

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
\(\wedge (i + m < n + i \Leftrightarrow m < n)\)
\(\wedge (m + i < i + n \Leftrightarrow m < n)\)
\(\wedge (i + m < i + n \Leftrightarrow m < n)\)
\(\wedge (m < m + i \Leftrightarrow 0 < i)\)
\(\wedge (m < i + m \Leftrightarrow 0 < i)\)
\(\wedge \neg m + i < m\)
\(\wedge \neg m + i < i\)
\(\wedge \neg m < 0\)
\(\wedge \neg m < m\)
\(\wedge 0 < m + 1\)
\(\wedge 0 < 1 + m\)
\(\wedge 0 < 1\)

\(\text{N_cases_thm} \vdash \forall m \bullet m = 0 \lor (\exists i \bullet m = i + 1)\)

\(\text{\le \_cases_thm} \vdash \forall m \bullet m \le n \lor n \le m\)

\(\text{\le \_plus1_thm} \vdash \forall m \bullet m \le n + 1 \Leftrightarrow m = n + 1 \lor m \le n\)

\(\text{plus1 \_\le \_thm} \vdash \forall m \bullet m + 1 \le n \Leftrightarrow m \le n \land \neg m = n\)

\(\neg \text{\_plus1 \_\le \_thm} \vdash \forall m \bullet \neg m + 1 \le n \Leftrightarrow n \le m\)

\(\text{less \_cases_thm} \vdash \forall m \bullet m < n \lor m = n \lor n < m\)

\(\neg \text{\_less \_plus1_thm} \vdash \forall m \bullet \neg m < n + 1 \Leftrightarrow n < m\)

\(\text{less \_plus1_thm} \vdash \forall m \bullet m < n + 1 \Leftrightarrow m = n \lor m < n\)

\(\text{plus1 \_less_thm} \vdash \forall m \bullet m + 1 < n \Leftrightarrow m < n \land \neg m + 1 = n\)

\(\text{\le \_antisym_thm} \vdash \forall m \bullet m \le n \land n \le m \Leftrightarrow m = n\)

\(\text{less \_irrefl_thm} \vdash \forall m \bullet \neg (m < n \land n < m)\)

\(\text{cov \_induction_thm} \vdash \forall p \bullet (\forall n \bullet (\forall m \bullet m < n \Rightarrow p m) \Rightarrow p n) \Rightarrow (\forall n \bullet p n)\)

\(\text{less \_well \_order_thm} \vdash \forall p \bullet (\exists i \bullet p i) \Leftrightarrow (\exists m \bullet p m \land (\forall i \bullet p i \Rightarrow \neg i < m))\)

\(\neg \text{\_less_thm} \vdash \forall m \bullet \neg m < n \Leftrightarrow n \le m\)

\(\neg \text{\_\le \_thm} \vdash \forall m \bullet \neg m \le n \Leftrightarrow n < m\)

\(\text{\le \_well \_order_thm} \vdash \forall p \bullet (\exists i \bullet p i) \Leftrightarrow (\exists m \bullet p m \land (\forall i \bullet p i \Rightarrow m \le i))\)

\(\text{\le \_least \_upper \_bound_thm} \vdash \forall p \bullet (\exists i \bullet p i) \land (\exists n \bullet \forall j \bullet p j \Rightarrow j \le n) \Leftrightarrow (\exists m \bullet p m \land (\forall j \bullet p j \Rightarrow j \le m))\)

\(\text{minimum \_\neg \_thm} \vdash \forall p \bullet p 0 \land \neg p b \Rightarrow (\exists m \bullet (\forall n \bullet n \le m \Rightarrow p n) \land \neg p (m + 1))\)

\(\text{times \_comm \_thm} \vdash \forall m \bullet m \cdot n = n \cdot m\)

\(\text{times \_assoc \_thm} \vdash \forall i m \bullet (i \cdot m) \cdot n = i \cdot m \cdot n\)

\(\text{times \_plus \_distrib \_thm}\)
\[ \forall i \ m \ n \\
\bullet (i + m) * n = i * n + m * n \\
\land i * (m + n) = i * m + i * n \]

**times_clauses**

\[ \forall m \bullet m * 0 = 0 \land 0 * m = 0 \land m * 1 = m \land 1 * m = m \]

**mod_less_thm**

\[ \forall m \ n \bullet 0 < n \Rightarrow m \ Mod n < n \]

**div_mod_thm**

\[ \forall m \ n \bullet 0 < n \Rightarrow m = m \ Div n * n + m \ Mod n \]

**div_mod_unique_thm**

\[ \forall m \ n \ d \ r \\
\bullet r < n \Rightarrow m = d * n + r \Rightarrow d = m \ Div n \land r = m \ Mod n \]

**minus_clauses**

\[ \forall m \ n \\
\bullet m - m = 0 \\
\land m - 0 = m \\
\land (m + n) - n = m \\
\land (m + n) - m = n \]
9.1.24 The Theory $\mathbb{Z}$

9.1.24.1 Parents

$sets$

9.1.24.2 Children

$dyadic$

9.1.24.3 Constants

$\text{Is}_\mathbb{Z}_\text{Rep}$ $(\mathbb{N} \times \mathbb{N}) \text{SET} \rightarrow \text{BOOL}$

$\text{NZ}$ $\mathbb{N} \rightarrow \mathbb{Z}$

$\sim_\mathbb{Z}$ $\mathbb{Z} \rightarrow \mathbb{Z}$

$\oplus_\mathbb{Z}$ $\mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{Z}$

$\ominus_\mathbb{Z}$ $\mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{Z}$

$\otimes_\mathbb{Z}$ $\mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{Z}$

$\preceq_\mathbb{Z}$ $\mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \text{BOOL}$

$\prec_\mathbb{Z}$ $\mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \text{BOOL}$

$\succeq_\mathbb{Z}$ $\mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \text{BOOL}$

$\succ_\mathbb{Z}$ $\mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \text{BOOL}$

$\text{Abs}_\mathbb{Z}$ $\mathbb{Z} \rightarrow \mathbb{Z}$

$\text{Mod}_\mathbb{Z}$ $\mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{Z}$

$\text{Div}_\mathbb{Z}$ $\mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{Z}$

9.1.24.4 Aliases

$+$ $\oplus_\mathbb{Z} : \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{Z}$

$-$ $\ominus_\mathbb{Z} : \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{Z}$

$\sim$ $\sim_\mathbb{Z} : \mathbb{Z} \rightarrow \mathbb{Z}$

$*$ $\otimes_\mathbb{Z} : \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{Z}$

$\leq$ $\preceq_\mathbb{Z} : \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \text{BOOL}$

$<$ $\prec_\mathbb{Z} : \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \text{BOOL}$

$\geq$ $\succeq_\mathbb{Z} : \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \text{BOOL}$

$>$ $\succ_\mathbb{Z} : \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \text{BOOL}$

$\text{Abs}$ $\text{Abs}_\mathbb{Z} : \mathbb{Z} \rightarrow \mathbb{Z}$

$\text{Div}$ $\text{Div}_\mathbb{Z} : \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{Z}$

$\text{Mod}$ $\text{Mod}_\mathbb{Z} : \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{Z}$

9.1.24.5 Types

$\mathbb{Z}$
9.1.24.6 Fixity

Left Infix 305:
−Z

Left Infix 315:
\text{Div}_Z \text{ Mod}_Z

Right Infix 210:
<Z >Z \leq Z \geq Z

Right Infix 300:
+Z

Right Infix 310:
*Z

Prefix 350:
\text{Abs}_Z \sim_Z

9.1.24.7 Definitions

\text{Is}_Z\text{_Rep} \vdash \forall a \bullet \text{Is}_Z\text{_Rep} a \leftrightarrow (\exists m n \bullet a = \{(x, y) | m + y = n + x\})

\text{Z}\text{_def} \vdash \exists f \bullet \text{TypeDefn Is}_Z\text{_Rep} f

+Z

\sim_Z \vdash \text{ConstSpec}
\(\lambda (+'_Z, \sim'_Z, \$''\text{NZ}''')\)
\(\forall i j k
\bullet +'_Z (+'_Z i j) k = +'_Z i (+'_Z j k)
\wedge +'_Z i j = +'_Z j i
\wedge +'_Z i (\sim'_Z i) = \$''\text{NZ}''' 0
\wedge +'_Z i (\$''\text{NZ}''' 0) = i
\wedge (\forall m n
\bullet +'_Z (\$''\text{NZ}''' n) (\$''\text{NZ}''' m)
= \$''\text{NZ}''' (m + n))
\wedge \text{OneOne} \$''\text{NZ}'''
\wedge (\forall i
\bullet \exists m \bullet \exists i = \$''\text{NZ}''' m \vee i = \sim'_Z (\$''\text{NZ}''' m)))
\(\$+, \sim, \text{NZ}\)

−Z \vdash \forall i j \bullet i - j = i + \sim j

*Z \vdash \text{ConstSpec}
\(\lambda *'_Z\)
\(\forall i j k
\bullet *'_Z i (j + k) = *'_Z i j + *'_Z i k
\wedge *'_Z i (\text{NZ} 1) = i
\)

\leq_Z \vdash \forall i j \bullet i \leq j \leftrightarrow (\exists m \bullet i + \text{NZ} m = j)

<Z \vdash \forall i j \bullet i < j \leftrightarrow i + \text{NZ} 1 \leq j

\geq_Z \vdash \forall i j \bullet i \geq j \leftrightarrow j \leq i

>Z \vdash \forall i j \bullet i > j \leftrightarrow j < i

\text{Abs}_Z \vdash \forall i \bullet \text{Abs} i = (\text{if} \ \text{NZ} 0 \leq i \ \text{then} \ i \ \text{else} \ \sim i)

\text{Div}_Z \text{ Mod}_Z \vdash \text{ConstSpec}
\(\lambda (\text{Div}'_Z, \text{ Mod}'_Z)\)
\(\forall i j
\)
• \( j = \text{NZ} \ 0 \)
\[ \Rightarrow i = \text{Div}_Z'\ i\ j + \text{Mod}_Z'\ i\ j \]
\[ \land \NZ\ 0 \leq \text{Mod}_Z'\ i\ j \]
\[ \land \text{Mod}_Z'\ i\ j < \text{Abs}\ j \]
\[ (\text{Div}, \text{Mod}) \]

9.1.24.8 Theorems

\( \text{id}_{\text{Z}\_\text{rep}\_\text{consistent}}\)
\[ \vdash \exists a \bullet \text{Is}_{\text{Z}\_\text{Rep}}\ a \]

\( +_{\text{Z}}\_\text{consistent} \)

\( \sim_{\text{Z}}\_\text{consistent} \)

\( \text{NZ}\_\text{consistent} \)
\[ \vdash \text{Consistent} \]
\[ (\lambda (+, \sim, \text{"NZ"})\]

\( \forall i\ j\ k\)
\[ +_Z'\ (+_Z\ i\ j)\ k = +_Z'\ i\ (+_Z\ j\ k) \]
\[ \land +_Z'\ i\ j = +_Z'\ j\ i \]
\[ \land +_Z'\ i\ (\sim_{_Z} i) = \text{"NZ"} 0 \]
\[ \land +_Z'\ i\ (\text{"NZ"} 0) = i \]
\[ \land (\forall m\ n)\]
\[ +_Z\ (\text{"NZ"} m)\ (\text{"NZ"} n) \]
\[ = \text{"NZ"} (m + n) \]
\[ \land \text{OneOne} \text{"NZ"} \]
\[ \land (\forall i)\]
\[ \exists m \bullet i = \text{"NZ"} m \lor i = \sim_{_Z} (\text{"NZ"} m) \]

\( \text{Z}\_\text{plus}\_\text{comm}\_\text{thm} \)
\[ \vdash \forall i\ j \bullet i + j = j + i \]

\( \text{Z}\_\text{plus}\_\text{assoc}\_\text{thm} \)
\[ \vdash \forall i\ j\ k \bullet (i + j) + k = i + j + k \]

\( \text{Z}\_\text{plus}\_\text{assoc}\_\text{thm}1 \)
\[ \vdash \forall i\ j\ k \bullet i + j + k = (i + j) + k \]

\( \text{Z}\_\text{plus}\_\text{order}\_\text{thm} \)
\[ \vdash \forall i\ j\ k \]
\[ \bullet j + i = i + j \]
\[ \land (i + j) + k = i + j + k \]
\[ \land j + i + k = i + j + k \]

\( \text{Z}\_\text{cases}\_\text{thm} \)
\[ \vdash \forall i \bullet \exists m \bullet i = \text{NZ} m \lor i = \sim (\text{NZ} m) \]

\( \text{Z}\_\text{plus0}\_\text{thm} \)
\[ \vdash \forall i \bullet i + \text{NZ} 0 = i \land \text{NZ} 0 + i = i \]

\( \text{Z}\_\text{plus}\_\text{minus}\_\text{thm} \)
\[ \vdash \forall i \bullet i + \sim i = \text{NZ} 0 \land \sim i + i = \text{NZ} 0 \]

\( \text{Z}\_\text{eq}\_\text{thm} \)
\[ \vdash \forall i\ j \bullet i = j \leftrightarrow i + \sim j = \text{NZ} 0 \]

\( \text{NZ}\_\text{plus}\_\text{homorphism}\_\text{thm} \)
\[ \vdash \forall m\ n \bullet \text{NZ} (m + n) = \text{NZ} m + \text{NZ} n \]

\( \text{Z}\_\text{minus}\_\text{cases} \)
\[ \vdash \forall i\ j \]
\[ \bullet \sim (\sim i) = i \]
\[ \land i + \sim i = \text{NZ} 0 \]
\[ \land \sim i + i = \text{NZ} 0 \]
\[ \land \sim (i + j) = \sim i + \sim j \]
\[ \land \sim (\text{NZ} 0) = \text{NZ} 0 \]
Z_cases_thm1 ⊢ ∀ i • ∃ m • i = NZ m ∨ i = ~ (NZ (m + 1))

Z_induction_thm

⊢ p (NZ 1)

∧ (∀ i • p i ⇒ p (~ i))
∧ (∀ i j • p i ∧ p j ⇒ p (i + j))
⇒ (∀ i • p i)

NZ_one_one_thm

⊢ ∀ m n • NZ m = NZ n ↔ m = n

Z_plus_clauses

⊢ ∀ i j k

• (i + k = j + k ⇔ i = j)
∧ (k + i = j + k ⇔ i = j)
∧ (i + k = k + j ⇔ i = j)
∧ (k + i = k + j ⇔ i = j)
∧ (i + k = k ⇔ i = NZ 0)
∧ (k + i = k ⇔ i = NZ 0)
∧ (k = j + k ⇔ j = NZ 0)
∧ (k = j + k ⇔ j = NZ 0)
∧ i + NZ 0 = i
∧ NZ 0 + i = i
∧ ~ NZ 1 = NZ 0
∧ ~ NZ 0 = NZ 1

Z_le_0_thm

⊢ ∀ i j • i ≤ j ⇔ i + ~ j ≤ NZ 0

Z_minus_le_thm

⊢ ∀ i j • ~ i ≤ ~ j ⇔ j ≤ i

Z_le_minus_thm

⊢ ∀ i j • i ≤ j ⇔ ~ j ≤ ~ i

Z_le_clauses

⊢ ∀ i j k

• (i + k ≤ j + k ⇔ i ≤ j)
∧ (k + i ≤ j + k ⇔ i ≤ j)
∧ (i + k ≤ k + j ⇔ i ≤ j)
∧ (k + i ≤ k + j ⇔ i ≤ j)
∧ (i + k ≤ k ⇔ i ≤ NZ 0)
∧ (k + i ≤ k ⇔ i ≤ NZ 0)
∧ (k ≤ k + j ⇔ NZ 0 ≤ j)
∧ (k ≤ j + k ⇔ NZ 0 ≤ j)
∧ i ≤ i
∧ ~ NZ 1 = NZ 0
∧ NZ 0 ≤ NZ 1

NZ_le_thm

⊢ ∀ m n • NZ m ≤ NZ n ↔ m ≤ n

*Z_consistent

⊢ Consistent
(λ *'Z

• ∀ i j k

• *'Z i (j + k) = *'Z i j + *'Z i k
∧ *'Z i (NZ 1) = i)

NZ_times_homomorphism_thm

⊢ ∀ m n • NZ (m * n) = NZ m * NZ n

Z_times_minus_thm

⊢ ∀ i j
\* \sim i \ast j = \sim (i \ast j) \\
\wedge i \ast \sim j = \sim (i \ast j) \\
\wedge \sim i \ast \sim j = i \ast j

**Z_times_comm_thm**

\[ \forall i j \bullet i \ast j = j \ast i \]

**Z_times_assoc_thm**

\[ \forall i j k \bullet (i \ast j) \ast k = i \ast j \ast k \]

**DivZ_consistent**

**ModZ_consistent**

\[ \vdash \text{Consistent} \]
\[ (\lambda (\text{Div}_Z', \text{Mod}_Z') \]
\[ \bullet \forall i j \]
\[ \bullet \sim j = \text{NZ} 0 \]
\[ \Rightarrow i = \text{Div}_Z' i j \ast j + \text{Mod}_Z' i j \]
\[ \wedge \text{NZ} 0 \leq \text{Mod}_Z' i j \]
\[ \wedge \text{Mod}_Z' i j < \text{Abs} j \]

**NZ_plus_homomorphism_thm1**

\[ \vdash \forall i j \bullet \text{NZ} 0 \leq i \Rightarrow \text{NZ} 0 \leq i + \text{NZ} 1 \]

**Z_N_induction_thm**

\[ \forall p \]
\[ \bullet p (\text{NZ} 0) \wedge (\forall i \bullet \text{NZ} 0 \leq i \wedge p i \Rightarrow p (i + \text{NZ} 1)) \]
\[ \Rightarrow (\forall m \bullet \text{NZ} 0 \leq m \Rightarrow p m) \]

**Z_N_plus_thm**

\[ \forall i j \bullet \text{NZ} 0 \leq i \wedge \text{NZ} 0 \leq j \Rightarrow \text{NZ} 0 \leq i + j \]

**Z_N_plus1_thm**

\[ \forall i \bullet \text{NZ} 0 \leq i \Rightarrow \text{NZ} 0 \leq i + \text{NZ} 1 \]

**Z_minus_thm**

\[ \forall i j \]
\[ \bullet \sim (\sim i) = i \]
\[ \wedge i + + i = \text{NZ} 0 \]
\[ \wedge \sim i + i = \text{NZ} 0 \]
\[ \wedge \sim (i + j) = \sim i + \sim j \]
\[ \wedge \sim (\text{NZ} 0) = \text{NZ} 0 \]

**Z_N_cases_thm**

\[ \forall i \]
\[ \bullet \text{NZ} 0 \leq i \]
\[ \Rightarrow i = \text{NZ} 0 \wedge (\exists j \bullet \text{NZ} 0 \leq j \wedge i = j + \text{NZ} 1) \]

**Z_N_minus_thm**

\[ \forall i j \bullet \text{NZ} 0 \leq i \Rightarrow i = \text{NZ} 0 \wedge \sim \text{NZ} 0 \leq \sim i \]

**Z_N_plus1_thm**

\[ \forall i \bullet \sim \text{NZ} 0 \leq i \Rightarrow \text{NZ} 0 \leq \sim i \]

**Z_plus_eq_thm**

\[ \forall i j k \bullet i + j = k \iff i = k + \sim j \]

**Z_N_plus1_thm**

\[ \forall i j \bullet \text{NZ} 0 \leq i \Rightarrow \sim i + \text{NZ} 1 = \text{NZ} 0 \]

**Z_times_assoc_thm1**

\[ \forall i j k \bullet i \ast j \ast k = (i \ast j) \ast k \]

**Z_times_order_thm**

\[ \forall i j k \]
\[ \bullet j \ast i = i \ast j \]
\[ \wedge (i \ast j) \ast k = i \ast j \ast k \]
\[ \wedge j \ast i \ast k = i \ast j \ast k \]

**NZ_times_homomorphism_thm1**

\[ \vdash \forall m n \bullet \text{NZ} m \ast \text{NZ} n = \text{NZ} (m \ast n) \]
\begin{align*}
\text{Z\_times1\_thm} & \vdash \forall i \bullet i \ast \text{NZ} \ 1 = i \land \text{NZ} \ 1 \ast i = i \\
\text{Z\_times\_plus\_distrib\_thm} & \vdash \forall i \ j \ k \\
& \bullet i \ast (j + k) = i \ast j + i \ast k \\
& \land (i + j) \ast k = i \ast k + j \ast k \\
\text{Z\_times0\_thm} & \vdash \forall i \bullet \text{NZ} \ 0 \ast i = \text{NZ} \ 0 \land i \ast \text{NZ} \ 0 = \text{NZ} \ 0 \\
\text{Z\_eq\_thm1} & \vdash \forall i \ j \bullet i = j \iff i + j = \text{NZ} \ 0 \\
\text{Z\_times\_eq\_0\_thm} & \vdash \forall i \ j \bullet i \ast j = \text{NZ} \ 0 \iff i = \text{NZ} \ 0 \lor j = \text{NZ} \ 0 \\
\text{Z\_times\_clauses} & \vdash \forall i \ j \\
& \bullet \text{NZ} \ 0 \ast i = \text{NZ} \ 0 \\
& \land i \ast \text{NZ} \ 0 = \text{NZ} \ 0 \\
& \land i \ast \text{NZ} \ 1 = i \\
& \land \text{NZ} \ 1 \ast i = i \\
\text{Z\_N\_times\_thm} & \vdash \forall i \ j \bullet \text{NZ} \ 0 \leq i \land \text{NZ} \ 0 \leq j \Rightarrow \text{NZ} \ 0 \leq i \ast j \\
\text{Z\_\leq\_trans\_thm} & \vdash \forall i \ j \ k \bullet i \leq j \land j \leq k \Rightarrow i \leq k \\
\text{Z\_\leq\_cases\_thm} & \vdash \forall i \ j \bullet i \leq j \lor j \leq i \\
\text{Z\_\leq\_refl\_thm} & \vdash \forall i \bullet i \leq i \\
\text{Z\_\leq\_0\_thm1} & \vdash \forall i \ j \bullet i \leq j \Leftrightarrow \text{NZ} \ 0 \leq j + \sim i \\
\text{Z\_\leq\_antisym\_thm} & \vdash \forall i \ j \bullet i \leq j \land j \leq i \Rightarrow i = j \\
\text{Z\_less\_trans\_thm} & \vdash \forall i \ j \ k \bullet i < j \land j < k \Rightarrow i < k \\
\text{Z\_less\_irrefl\_thm} & \vdash \forall i \ j \bullet \sim (i < j \land j < i) \\
\text{Z\_less\_cases\_thm} & \vdash \forall i \ j \bullet i < j \lor i = j \lor j < i \\
\text{NZ\_less\_thm} & \vdash \forall m \ n \bullet \text{NZ} \ m < \text{NZ} \ n \Leftrightarrow m < n \\
\text{Z\_less\_less\_0\_thm} & \vdash \forall i \ j \bullet i < j \Leftrightarrow i + \sim j < \text{NZ} \ 0 \\
\text{Z\_less\_less\_0\_thm1} & \vdash \forall i \ j \bullet i < j \Leftrightarrow \text{NZ} \ 0 < j + \sim i \\
\text{Z\_minus\_less\_thm} & \vdash \forall i \ j \bullet \sim i < \sim j \Leftrightarrow j < i \\
\text{Z\_\sim\_less\_thm} & \vdash \forall i \ j \bullet \sim i < j \Leftrightarrow j < i \\
\text{Z\_\sim\_\leq\_thm} & \vdash \forall i \ j \bullet \sim i \leq j \Leftrightarrow j < i \\
\text{Z\_\leq\_less\_eq\_thm} & \vdash \forall i \ j \bullet i \leq j \Leftrightarrow i < j \land i = j \\
\text{Z\_less\_\leq\_trans\_thm} & \vdash \forall i \ j \ k \bullet i < j \land j \leq k \Rightarrow i < k \\
\text{Z\_\leq\_less\_trans\_thm} & \vdash \forall i \ j \ k \bullet i \leq j \land j < k \Rightarrow i < k \\
\text{Z\_minus\_N\_\leq\_thm} & \vdash \forall i \ m \bullet i + \sim (\text{NZ} \ m) \leq i \\
\text{Z\_\leq\_plus\_N\_thm} & \vdash \forall i \ m \bullet i \leq i + \text{NZ} \ m \\
\text{Z\_\leq\_N\_thm} & \vdash \forall i \bullet \text{NZ} \ 0 \leq i \Leftrightarrow (\exists m \bullet i = \text{NZ} \ m) \\
\end{align*}
\textbf{Z\_less\_clauses}
\[\begin{align*}
&\vdash \forall i \ j \ k \\
&\quad \bullet (i + k < j + k \iff i < j) \\
&\quad \land (k + i < j + k \iff i < j) \\
&\quad \land (i + k < k + j \iff i < j) \\
&\quad \land (k + i < k \iff i < \NZ 0) \\
&\quad \land (i < k + i \iff \NZ 0 < k) \\
&\quad \land (i < i + k \iff \NZ 0 < k) \\
&\quad \land \neg i < i \\
&\quad \land \NZ 0 < \NZ 1 \\
&\quad \land \neg \NZ 1 < \NZ 0
\end{align*}\]

\textbf{Z\_N\_abs\_thm}
\[\begin{align*}
&\vdash \forall m \bullet \Abs (\NZ m) = \NZ m \land \Abs (\sim (\NZ m)) = \NZ m
\end{align*}\]

\textbf{Z\_abs\_thm}
\[\begin{align*}
&\vdash \forall i \bullet \NZ 0 \leq i \Rightarrow \Abs i = i \land \Abs (\sim i) = i
\end{align*}\]

\textbf{Z\_abs\_N\_thm}
\[\begin{align*}
&\vdash \forall i \bullet \NZ 0 \leq \Abs i
\end{align*}\]

\textbf{Z\_abs\_eq\_0\_thm}
\[\begin{align*}
&\vdash \forall i \bullet \Abs i = \NZ 0 \iff i = \NZ 0
\end{align*}\]

\textbf{Z\_abs\_minus\_thm}
\[\begin{align*}
&\vdash \forall i \bullet \Abs (\sim i) = \Abs i
\end{align*}\]

\textbf{Z\_N\_abs\_minus\_thm}
\[\begin{align*}
&\vdash \forall i \ j \\
&\quad \bullet \NZ 0 \leq i \land \NZ 0 \leq j \land i \Rightarrow \Abs (i + \sim j) \leq i
\end{align*}\]

\textbf{Z\_abs\_times\_thm}
\[\begin{align*}
&\vdash \forall i \ j \bullet \Abs (i \ast j) = \Abs i \ast \Abs j
\end{align*}\]

\textbf{Z\_abs\_plus\_thm}
\[\begin{align*}
&\vdash \forall i \ j \bullet \Abs (i + j) \leq \Abs i + \Abs j
\end{align*}\]

\textbf{Z\_div\_mod\_unique\_lemma1}
\[\begin{align*}
&\vdash \forall i \ j \bullet \NZ 0 \leq i \land \NZ 0 \leq j \land i \Rightarrow \Abs (i + \sim j) \leq i
\end{align*}\]

\textbf{Z\_div\_mod\_unique\_lemma2}
\[\begin{align*}
&\vdash \forall j \ d \ r \\
&\quad \bullet \sim j = \NZ 0 \\
&\quad \Rightarrow d \ast j + r = \NZ 0 \land \NZ 0 \leq r \land r < \Abs j \\
&\quad \Rightarrow d = \NZ 0 \land r = \NZ 0
\end{align*}\]

\textbf{Z\_div\_mod\_unique\_lemma3}
\[\begin{align*}
&\vdash \forall i \ j \ d \ r \ D \ R \\
&\quad \bullet \sim j = \NZ 0 \\
&\quad \Rightarrow D \ast j + R = d \ast j + r \\
&\quad \land \NZ 0 \leq r \\
&\quad \land r \leq R \\
&\quad \land R < \Abs j \\
&\quad \Rightarrow D = d \land R = r
\end{align*}\]

\textbf{Z\_div\_mod\_unique\_thm}
\[\begin{align*}
&\vdash \forall i \ j \ d \ r \\
&\quad \bullet \sim j = \NZ 0 \\
&\quad \Rightarrow (i = d \ast j + r \land \NZ 0 \leq r \land r < \Abs j) \\
&\quad \Rightarrow d = i \Div j \land r = i \Mod j
\end{align*}\]

\textbf{Z\_\le\_induction\_thm}
\[\begin{align*}
&\vdash \forall j \ p \\
&\quad \bullet p \ j \land (\forall i \bullet j \leq i \land p \ i \Rightarrow p \ (i + \NZ 1)) \\
&\quad \Rightarrow (\forall i \bullet j \leq i \Rightarrow p \ i)
\end{align*}\]
\[ Z_{\text{cov\_induction\_thm}} \]
\[ \vdash \forall j \ p \]
\[ \bullet (\forall i \bullet j \leq i \wedge (\forall k \bullet j \leq k \wedge k < i \Rightarrow p k) \Rightarrow p i) \]
\[ \Rightarrow (\forall i \bullet j \leq i \Rightarrow p i) \]

\[ Z_{\text{fun\_}\exists\_\text{thm}} \]
\[ \vdash \forall f \ g \ z \]
\[ \bullet (\forall x \bullet g (f \ x) = x) \wedge (\forall y \bullet f (g \ y) = y) \]
\[ \Rightarrow (\exists_1 h \]
\[ \bullet h (\text{NZ} 0) = z \]
\[ \wedge (\forall i \bullet h (i + \text{NZ} 1) = f (h i)) \]
\[ \wedge (\forall i \bullet h (i - \text{NZ} 1) = g (h i))) \]
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 Basic Definitions And Axioms

```
SML
signature InitTheory = sig
  val bool_cases_axiom : THM;
  val η_axiom : THM;
  val infinity_axiom : THM;
  val ⇒_antisym_axiom : THM;
  val ϵ_axiom : THM;
end;
```

**Description** The signature `InitTheory` contains the definitions for the theory `init` (q.v.). This contains the five primitive axioms of HOL.

```
SML
signature LogTheory = sig
  val f_def : THM;
  val one_one_def : THM;
  val onto_def : THM;
  val t_def : THM;
  val type_defn_def : THM;
  val ∃_def : THM;
  val ¬_def : THM;
  val ∧_def : THM;
  val ∀_def : THM;
  val ∨_def : THM;
end;
```

**Description** The signature `LogTheory` contains the definitions for the theory `log` (q.v.). This defines the basic predicate calculus connectives and other derived notions which are needed to state the axioms for HOL.

```
SML
signature MinTheory = sig end;
```

**Description** The signature `MinTheory` contains the definitions for the theory `min`. This introduces the primitive types and constants of HOL.

```
SML
signature MiscTheory = sig
  val cond_def : THM;
  val let_def : THM;
  val ∃_1_def : THM;
end;
```

**Description** The signature `MiscTheory` contains the initial definitions for the theory `misc`. This contains miscellaneous definitions and theorems required in constructing the proof development system. Further items may be saved in theory “misc”, and bound to ML variables in other signatures and structures.
9.2.2 Pairs

SML
signature PairTheory = sig

Description This is the signature in which we declare theory “pair”, and add a few theorems to theory “misc”.

SML
val pair_clauses : THM;
end;

Description This single theorem collects all the externally useful results of the theory of pairs into one portmanteau theorem. It is saved in theory “pair” with its name as its key.

SML
val pair_ops_def : THM;
val curry_def : THM;
val uncurry_def : THM;
val is_pair_rep_def : THM;

Description These definitions are saved under their names as keys in theory “misc”, and are the definitions for Fst, Snd, :, and IsPairRep.

SML
val type_lemmas_thm : THM;
val one_one_thm : THM;
val ext_thm : THM;
val fun_rel_thm : THM;

Description These four theorems are saved under their names as keys in theory “misc”. Their design is given in theory “misc”, and not here, though they are implemented within the structure PairTheory.

9.2.3 Elementary Arithmetic

SML
signature N = sig

Description This is the signature in which the theory “N” is declared. This theory contains facts about the natural numbers (i.e.: 0, 1, 2, . . .) and their usual arithmetic operators.

SML
val cov_induction_tac : TERM -> TACTIC;

Description This tactic implements course-of-values induction over the natural numbers. To prove \( t[x] \) it suffices to prove \( t[x] \) on the assumption \( \forall m \cdot m < x \Rightarrow t[m] \). (Course-of-values induction is also sometimes called as complete induction.) The term argument must be a variable of type \( \tau : \mathbb{N} \) and must appear free in the conclusion of the goal but not in its assumptions.

Tactic
\[
\begin{array}{c}
\{ F \} t[x] \\
\text{strip}(\forall m \cdot m < x \Rightarrow t[m]), F \} t[x] \\
\end{array}
\]
cov_induction_tac \( \tau \)

m will be renamed to a variant, if necessary.

Errors
38001 ?0 is not a variable of type \( \tau : \mathbb{N} \)
38002 ?0 does not appear free in the conclusion of the goal
38003 ?0 appears free in the assumptions of the goal

See Also induction_tac
SML
val COV_INDUCTION_T : TERM -> (THM -> TACTIC) -> TACTIC;

Description  This implements course-of-values induction over the natural numbers as a tactical. The term argument must be a variable of type $\Gamma\cdot N$ and must appear free in the conclusion of the goal but not in its assumptions. The inductive hypothesis is passed to the tactic generating function given by the second argument.

Tactic
\[
\{ \Gamma \} \ t[x] \\
\text{ttac} \\
(\forall m \bullet m < x \Rightarrow t \ m) \\
(\{ \Gamma \} \ t[x])
\]

COV_INDUCTION_T \ $\Gamma\cdot x$ \ ttac

Uses  Tactic programming.

See Also  cov_induction_tac, INDUCTION_T

Errors
38001  ?0 is not a variable of type $\Gamma\cdot N$
38002  ?0 does not appear free in the conclusion of the goal
38003  ?0 appears free in the assumptions of the goal

SML
val induction_tac : TERM -> TACTIC;

Description  This tactic implements induction over the natural numbers: to prove $t[x]$ it suffices to prove $t[0]$ and to prove $t[x+1]$ on the assumption that $t[x]$. The term argument must be a variable of type $\Gamma\cdot N$ and must appear free in the conclusion of the goal but not in its assumptions.

Tactic
\[
\{ \Gamma \} \ t[x] \\
\{ \Gamma \} \ t[0] ; \ \text{strip}\{t[x], \Gamma \} \ t[x+1]
\]

induction_tac \ $\Gamma\cdot x$

See Also  cov_induction_tac, INDUCTION_T

Errors
38001  ?0 is not a variable of type $\Gamma\cdot N$
38002  ?0 does not appear free in the conclusion of the goal
38003  ?0 appears free in the assumptions of the goal
9.2. Theory Related ML Values

SML

val INDUCTION_T : TERM -> (THM -> TACTIC) -> TACTIC;

Description This implements induction over the natural numbers as a tactical. The term argument must be a variable of type $\forall x: \mathbb{N}$ and must appear free in the conclusion of the goal but not in its assumptions. The inductive hypothesis is passed to the tactic generating function given by the second argument.

Tactic

$$\frac{\Gamma \vdash t[x]}{\{ \Gamma \} t[0] : ttac(t[x] \vdash t[x]) \quad ((\Gamma \vdash t[x+1])}$$

INDUCTION_T $\forall x \ddagger ttac$

Uses Most commonly used with \texttt{asm_tac} to avoid the stripping up of the inductive hypothesis which occurs with \texttt{induction_tac}.

See Also \texttt{induction_tac}, \texttt{COV\_INDUCTION\_T}

Errors

38001 ?0 is not a variable of type $\forall x: \mathbb{N}$
38002 ?0 does not appear free in the conclusion of the goal
38003 ?0 appears free in the assumptions of the goal

SML

val is_N_rep_def : THM;
val N_def : THM;
val div_def : THM;
val _def : THM;
val less_def : THM;
val greater_def : THM;
val minus_def : THM;
val mod_def : THM;
val plus_def : THM;
val times_def : THM;
val zeroSuc_def : THM;

Description These Standard ML variables are bound to the definitions in the theory “$\mathbb{N}$”.

SML

val plus1_conv : CONV;

Description This conversion proves theorems of the form $m = n+1$ where $m$ and $n$ are numeric literals.

Tactic

$$\vdash m = n + 1$$

plus1_conv ($mk_N m$)
Chapter 9. THEORIES

These Standard ML variables are bound to the theorems in the theory “N”.

9.2.4 Arithmetic Computation

This is the signature of the structure CompConv which contains conversions for performing computations on numeric, string and character literals and derived syntax functions for the arithmetic operations.
9.2. Theory Related ML Values

Description This conversion proves theorems of the form:

\[ \vdash b = c \iff t \]

Where \( \lceil b \rceil \) and \( \lceil c \rceil \) are character literals and where \( \lceil t \rceil \) is either \( \lceil T \rceil \) or \( \lceil F \rceil \).

Errors

\[ 57200 \ ?0 \text{ is not of the form: } \lceil \text{SML.mk_char } b \rceil = \lceil \text{SML.mk_char } c \rceil \]

See Also Proof context comb

Description These are destructor functions for the arithmetic operations.

Errors

\[ 57501 \ ?0 \text{ is not of the form: } \lceil x \leq y \rceil \]
\[ 57502 \ ?0 \text{ is not of the form: } \lceil x \geq y \rceil \]
\[ 57503 \ ?0 \text{ is not of the form: } \lceil x \text{ Div } y \rceil \]
\[ 57504 \ ?0 \text{ is not of the form: } \lceil x > y \rceil \]
\[ 57505 \ ?0 \text{ is not of the form: } \lceil x < y \rceil \]
\[ 57506 \ ?0 \text{ is not of the form: } \lceil x - y \rceil \]
\[ 57507 \ ?0 \text{ is not of the form: } \lceil x \text{ Mod } y \rceil \]
\[ 57508 \ ?0 \text{ is not of the form: } \lceil x + y \rceil \]
\[ 57509 \ ?0 \text{ is not of the form: } \lceil x \ast y \rceil \]

Description These are recogniser functions for the arithmetic operations.
### SML

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>val mk_≤</td>
<td>TERM*TERM -&gt; TERM</td>
</tr>
<tr>
<td>val mk_≥</td>
<td>TERM*TERM -&gt; TERM</td>
</tr>
<tr>
<td>val mk_div</td>
<td>TERM*TERM -&gt; TERM</td>
</tr>
<tr>
<td>val mk_greater</td>
<td>TERM*TERM -&gt; TERM</td>
</tr>
<tr>
<td>val mk_less</td>
<td>TERM*TERM -&gt; TERM</td>
</tr>
<tr>
<td>val mk_minus</td>
<td>TERM*TERM -&gt; TERM</td>
</tr>
<tr>
<td>val mk_plus</td>
<td>TERM*TERM -&gt; TERM</td>
</tr>
<tr>
<td>val mk_times</td>
<td>TERM*TERM -&gt; TERM</td>
</tr>
</tbody>
</table>

#### Description
These are constructor functions for the arithmetic operations.

#### Errors

| 57510 | 0 is not of type `⌜N⌝` |
| 57511 | 1 is not of type `⌜N⌝` |

### SML

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>val string_eq_conv : CONV</td>
</tr>
</tbody>
</table>

#### Description

These conversions prove theorems of the form:

\[ \vdash b = c \leftrightarrow t \]

Where `⌜b⌝` and `⌜c⌝` are string literals and where `⌜t⌝` is either `⌜T⌝` or `⌜F⌝`.

#### Errors

| 57300 | 0 is not of the form `⌜ML.mk_string b⌝ = ⌜ML.mk_string c⌝` |
9.2. Theory Related ML Values

<table>
<thead>
<tr>
<th>SML</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>val _≤_conv : CONV (* ≤ *)</code></td>
<td></td>
</tr>
<tr>
<td><code>val _≥_conv : CONV (* ≥ *)</code></td>
<td></td>
</tr>
<tr>
<td><code>val _div_conv : CONV (* Div *)</code></td>
<td></td>
</tr>
<tr>
<td><code>val _greater_conv : CONV (* &gt; *)</code></td>
<td></td>
</tr>
<tr>
<td><code>val _less_conv : CONV (* &lt; *)</code></td>
<td></td>
</tr>
<tr>
<td><code>val _minus_conv : CONV (* − *)</code></td>
<td></td>
</tr>
<tr>
<td><code>val _mod_conv : CONV (* Mod *)</code></td>
<td></td>
</tr>
<tr>
<td><code>val _plus_conv : CONV (* + *)</code></td>
<td></td>
</tr>
<tr>
<td><code>val _times_conv : CONV (* * *)</code></td>
<td></td>
</tr>
</tbody>
</table>

**Description** These conversions prove theorems of the form:

\[ \vdash i \ op \ j = k \]

Where \( \gamma i \gamma \), \( \gamma j \gamma \) and \( \gamma k \gamma \) are numeric literals and where \( \text{op} \) is one of the standard arithmetic operators as indicated in the comments above.

(Note that `plus_conv` is the same as the built-in rule `plus_conv` described in DTD009).

**Errors**

| 57001 ?0 is not of the form: \( \gamma ml.mk.N \ m \gamma \leq \gamma ml.mk.N \ n \gamma \) |
| 57002 ?0 is not of the form: \( \gamma ml.mk.N \ m \gamma \geq \gamma ml.mk.N \ n \gamma \) |
| 57003 ?0 is not of the form: \( \gamma ml.mk.N \ m \gamma \text{Div} \ ml.mk.N \ n \gamma \) |
| 57004 ?0 is not of the form: \( \gamma ml.mk.N \ m \gamma > \gamma ml.mk.N \ n \gamma \) |
| 57005 ?0 is not of the form: \( \gamma ml.mk.N \ m \gamma \text{<} ml.mk.N \ n \gamma \) |
| 57006 ?0 is not of the form: \( \gamma ml.mk.N \ m \gamma \text{−} ml.mk.N \ n \gamma \) |
| 57007 ?0 is not of the form: \( \gamma ml.mk.N \ m \gamma \text{Mod} \ ml.mk.N \ n \gamma \) |
| 6085 ?0 is not of the form: \( \gamma ml.mk.N \ m \gamma + ml.mk.N \ n \gamma \) |
| 57009 ?0 is not of the form: \( \gamma ml.mk.N \ m \gamma \ast ml.mk.N \ n \gamma \) |
| 57010 Cannot compute ?0 (\( \gamma m \text{−} n \gamma \) is undefined when \( \gamma m \text{<} n \gamma \)) |
| 57011 Cannot compute ?0 (\( \gamma m \text{Div} n \gamma \) is undefined when \( \gamma n = 0 \gamma \)) |
| 57012 Cannot compute ?0 (\( \gamma m \text{Mod} n \gamma \) is undefined when \( \gamma n = 0 \gamma \)) |

**See Also** Proof context `comb`

<table>
<thead>
<tr>
<th>SML</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>val _N_eq_ conv : CONV</code></td>
<td></td>
</tr>
</tbody>
</table>

**Description** This conversion proves theorems of the form:

\[ \vdash i = j \iff t \]

Where \( \gamma i \gamma \) and \( \gamma j \gamma \) are numeric literals and where \( \gamma t \gamma \) is either \( \gamma T \gamma \) or \( \gamma F \gamma \).

**Errors**

| 57100 ?0 is not of the form: \( \gamma ml.mk.N \ m \gamma = ml.mk.N \ n \gamma \) |

**See Also** Proof context `comb`

9.2.5 Linear Arithmetic

<table>
<thead>
<tr>
<th>SML</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>signature LinearArithmetic = sig</code></td>
<td></td>
</tr>
</tbody>
</table>

**Description** This is the signature of a structure containing proof procedures for linear arithmetic and related functions.
The automatic proof tactic works by (i) restripping all the assumptions of the goal, (ii) adding the argument theorems to the stock of assumptions using strip_asm_tac, (iii) applying contr_tac, and (iv) searching for a linear combination of the assumptions which will reduce, by multiplying out and cancelling like terms, to a contradiction of the form \(a = b\) or \(a \leq b\) with \(a\) and \(b\) numeric literals. The automatic proof conversion just tries to prove its argument, \(t\) say, using the automatic proof tactic and returns \(t = T\) if it succeeds.

Other components of the proof context are as for predicates.

For example, \(PC_T1"lin_arith"prove_tac[]\) will prove any of the following goals:

\[
\begin{align*}
&([], \forall a b c \cdot a \leq b \land (a + b < c + a) \Rightarrow a < c) \\
&([], \forall a b c \cdot a \geq b \land a < c \Rightarrow a \geq c) \\
&([], \forall a b c \cdot a + 2*b < 2*a \Rightarrow b + b < a) \\
&([], \forall x y \cdot (2*x + y = 4 \land 4*x + 2*y = 7))
\end{align*}
\]

In cases where the automatic proof tactic cannot prove the goal, it is sometimes possible to use scale_asm_tac, q.v., or one of its variants to transform the goal into a form it can prove.

See Also  'lin_arith, scale_asm_tac
9.2. Theory Related ML Values

SML

/* Proof Context: 'lin_arith */

**Description**  This is a component proof context whose main purpose is to supply a decision procedure for problems in linear arithmetic.

**Contents**  The rewriting, theorem stripping and conclusion stripping components are as for the proof context `lin_arith` but without any of the material from the proof context `predicates`. The automatic proof components are as for `lin_arith`. Other components are blank.

A typical use of the proof context would be to solve problems containing a mixture of (linear) arithmetic and set theory. For example, `MERGE_PCS_T1["sets_ext", "lin_arith"]prove_tac[]` will prove any of the following goals:

\[
\begin{align*}
&[[], \forall m \cdot \{ i \mid m \leq i \land i < m+3 \} = \{ m; m+1; m+2 \}] \\
&[[], \{(i, j) \mid 30i = 105j \} = \{(i, j) \mid 2i = 7j \}] \\
&[[], \{ i \mid 5i = 6i \} = \{ 0 \}]
\end{align*}
\]

The automatic proof procedures may give rise to the following error message (in which the message from the function which first detected the problem is given in brackets):

**Errors**  

\[
82200  \textit{Could not prove theorem with conclusion ?1 (?0)}
\]
structure LinearArithmeticTools : sig
  type POLY = (INTEGER * int) list *;
  datatype CONSTRAINT_TYPE = Eq | LessEq | Less;
  type CONSTRAINT = POLY * CONSTRAINT_TYPE * INTEGER * POLY *;
  val normalise_poly : POLY -> POLY;
  val normalise_constraint : CONSTRAINT -> CONSTRAINT;
  val format_poly : POLY -> string;
  val format_constraint : CONSTRAINT -> string;
  val mult_and_add_poly : INTEGER -> POLY -> INTEGER -> POLY -> POLY;
  val mult_and_add_constraint : INTEGER -> CONSTRAINT -> INTEGER -> CONSTRAINT -> CONSTRAINT;
  val lin_arith_contr : CONSTRAINT list -> POLY;
end;

Description  This is a structure contained in the structure LinearArithmetic giving types and functions used to represent systems of linear constraints with integer coefficients over the rational numbers. Most importantly, lin_arith_contr provides a decision procedure for the unsatisfiability of such systems. In more detail:

Degree one polynomials in many variables $x_1, x_2, \ldots$ (henceforth, just polynomials) are represented, using the type POLY, as lists of pairs of integers, a pair $(a, i)$ indicating that the $i$-th variable has coefficient $a$. The function normalise_poly puts any polynomial in a normal form, in which the variables in such a polynomial occur in strictly increasing order and pairs with zero coefficients are omitted. Provided two polynomials, $p$ and $q$, are in normal form, mult_and_add_poly $i$ $p$ $j$ $q$ computes the normalised representation of $i*p + j*q$. N.B. for efficiency reasons, these functions do not check that their arguments are normalised and may produce results which are not only not normalised but also incorrect in such cases.

Constraints are labelled with an indicator of their origin. This is a polynomial, and typically one would label the $j$-th constraint with $x_j$. The provenance of a generated constraint is then represented using a polynomial, where e.g. $2x_1 + 3x_2$ means add twice the polynomial labelled with $x_1$ to thrice that labelled $x_2$. A constraint $a_1*x_1 + a_2*x_2 + \ldots a_k * x_k$ $R$ $c$ with provenance $p$ is then represented as a 4-tuple $[(a_1, x_1), (a_2, x_2), \ldots], R, c, p$. The polynomials in a constraint may be put into the normal form described above using normalise_constraint and constraints may be multiplied by constants and added using mult_and_add_constraint.

Given a list, cs, of constraints (not necessarily normalised), lin_arith_contr cs determines whether cs is unsatisfiable (i.e. contradictory): if so, it returns a normalised polynomial indicating, in terms of of the origin labels in the inputs, a linear combination of the inputs which reduces, by cancellation, to an unsatisfiable constraint with no variables; if not, it fails with error number 82110.

Errors

82110  System is satisfiable
9.2. Theory Related ML Values

\[
\text{val lin_arith_rule : TERM list -> THM;}
\]
\[
\text{val lin_arith_rule1 : TERM list -> THM;}
\]

**Description**  Given a system, \( \Gamma = [r_1, r_2, \ldots] \), of numeric constraints, \( r_i \), of the form \( s_i : \mathbb{N} \) = \( t_i \) or \( s_i \leq t_i \) these rules attempt to prove a theorem of the form \( \Gamma \vdash F \) (terms in \( \Gamma \) which are not of either of these forms are ignored and do not appear in the assumptions of the result theorem). The usual interface to these rules is via the decision procedures in the proof context of either of these forms are ignored and do not appear in the assumptions of the result theorem). The usual interface to these rules is via the decision procedures in the proof context of either of these forms are ignored and do not appear in the assumptions of the result theorem).

1. The system of constraints is replaced by a new system in which the \( s_i \) and \( t_i \) are formed only from numeric literals, variables, + and *. This is done by choosing variables, \( x_k \), such that each \( r_i \) is a substitution instance of a constraint of the form \( u_{i1} + u_{i2} + \ldots = v_{i1} + v_{i2} + \ldots \) or \( u_{i1} + u_{i2} + \ldots \leq v_{i1} + v_{i2} + \ldots \), where the summands \( u_{ij} \) and \( v_{ij} \) have one of the forms: \( a_{ij} \times x_k \), \( a_{ij} \) or \( x_k \) where \( a_{ij} \) is a numeric literal; this is done using as few variables as possible subject to the constraint that a \( u_{ij} \) or a \( v_{ij} \) will only have the form \( x_k \) if it corresponds to a summand in \( r_i \) which is not of the form \( a \times t \) where \( a \) is a numeric literal.

2. In the case of \( \text{lin_arith_rule1} \) only, the new system is now augmented with an additional inequality \( x_k \geq 0 \) for each variable \( x_k \). (These additional inequalities do not appear in the assumptions of the result theorem.)

3. An attempt to derive a contradiction from the new system, \( \Delta = [r'_1, r'_2, \ldots] \) say, is then made using the search procedure \( \text{lin_arith_contr} \). If this attempt fails then an exception is raised (see below), otherwise, it produces integer values, \( i_1, i_2, \ldots \) say, such that the linear combination \( i_1 \times r'_1 + i_2 \times r'_2 + \ldots \) of the constraints \( r'_i \) reduces by multiplying through and cancelling like terms to a term of the form \( a = b \) (resp. \( a \leq b \)), where \( a \) and \( b \) are numeric literals such that \( \neg a = b \) (resp. \( \neg a \leq b \)). (Multiplying out by a negative value, \( -i \) say, amounts to interchanging the left- and right-hand side of the constraint, replacing \( \leq \) by \( \geq \) if appropriate, then multiplying by \( i \).)

4. The theorem \( \Delta \vdash F \) is then derived using \( \text{anf_conv} \) to effect the multiplication and cancelling, and then \( \Gamma \vdash F \) is derived from it using the substitution constructed in step 1.

If no contradiction can be derived then the rules both raise an exception which, if not handled, causes a diagnostic print of the input terms and of the system of linear equations which was derived from them. This is done by calling \( \text{fail} \), q.v., with its third argument set to a list containing a function which produces the diagnostic print-out by side-effect. The function \( \text{get_message} \), or \( \text{get_message_text} \) should not therefore be evaluated on the exception value unless the diagnostics are to be printed immediately.

**Errors**

82110  System is satisfiable
82111  A system with no constraints is satisfiable
82112  No constraints of the form \( \vdash t_1 \leq t_2 \) or \( \vdash (t1: \mathbb{N}) = t_2 \) could be derived from the assumptions
val lin_arith_tac : TACTIC;
val lin_arith_tac1 : TACTIC;

**Description**  These tactics are primarily intended for use in conjunction with the linear arithmetic proof context `lin_arith` and `lin_arith`, q.v. The normal interface to the tactics is via the decision procedures in these proof contexts, as in for example: `PC_T1"lin_arith"prove_tac[]`.

The tactics do however, have possible applications in specialised tactic programming, in which circumstances their behaviour may be understood from their definition, in terms of `lin_arith_rule` or `lin_arith_rule1`, q.v., essentially as:

```sml
val lin_arith_tac = GET_ASMS_T (f_thm_tac o lin_arith_rule o map concl);
val lin_arith_tac1 = GET_ASMS_T (f_thm_tac o lin_arith_rule1 o map concl);
```

**Uses**  The most likely application is in specialised tactic programming in situations where it is known that the assumptions are already in the normal form produced by the proof context `lin_arith` and it is important for performance not to restrip them.

```
val make≤conv : CONV

**Description**  This conversion transforms arithmetic relations formed with `<`, `≥` or `>`, or the negation of one formed with `=`, `≤`, `<`, `≥` or `>` into relations involving only `≤` as follows:

| `t1 < t2` | `t1 + 1 ≤ t2` |
| `t1 ≥ t2` | `t2 ≤ t1`   |
| `t1 > t2` | `t2 + 1 ≤ t1` |
| `¬ t1 = t2` | `t1 + 1 ≤ t2 ∨ t2 + 1 ≤ t1` |
| `¬ t1 ≤ t2` | `t2 + 1 ≤ t1` |
| `¬ t1 < t2` | `t2 ≤ t1`   |
| `¬ t1 ≥ t2` | `t1 + 1 ≤ t2` |
| `¬ t1 > t2` | `t1 ≤ t2`   |

**Uses**  The conversion is intended for use in tactic and conversion programming. The normal interactive interface is via rewriting or stripping in the proof context `lin_arith`, which performs other useful simplifications. For example, the call

```sml
ProofPower Input
```

```sml
PC_C1"lin_arith"rewrite_conv[]("¬x + 2*y + x + 3 = y + 2 + 2*x + y");
```

produces the following output:

```sml
val it = "¬x + 2*y + x + 3 = y + 2 + 2*x + y ↔ T : THM"
```

**Errors**  `82012` ?0 is not of the form `¬(t1:N) R1 t2 or ¬(t1:N) R2 t2` where R2 is one of `<`, `≥` or `>` and R2 is one of `=`, `≤`, `<`, `≥` or `>`. 

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUAL USR029
val scale_asm_tac : TERM -> TERM -> TACTIC;
val scale_nth_asm_tac : TERM -> int -> TACTIC;
val list_scale_nth_asm_tac : (TERM * int) list -> TACTIC;

Description  scale_asm_tac t asm removes asm from the assumption list and strips in a new assumption obtained by “scaling asm by the factor t” as explained in the description of scale_rule.

scale_nth_asm_tac is just like scale_asm_tac but allows the assumption to be identified by its number, and list_scale_nth_asm_tac is a list analogue of scale_nth_asm_tac, allowing several assumptions to be scaled at once (or indeed allowing one or more assumptions to be scaled by different factors).

For example, scale_asm_tac⇧(1+x)⇧y+1 ≤ z + 1⇧ removes the assumption ⇧y+1 ≤ z + 1⇧ from the assumption list and strips into the assumptions the new assumption: ⇧(1 + x)⇧(y + 1) ≤ (1 + x)⇧(z + 1). The scaling tactics are primarily intended for use with the decision procedure in the proof context lin_arith or other proof contexts containing lin_arith, to adjust a problem which it cannot solve into one it can. For example, consider the goal:

([], ⇧a*(d+1) = c*(b+1) ∧ c*(f+1) = e*(d+1) ⇒ a*(f+1) = e*(b+1)⇧)

To prove this, we first use contr_tac(say in the proof context hol) to move everything into the assumptions, giving:

(* 3 *) ⇧a * (d + 1) = c * (b + 1)⇧
(* 2 *) ⇧c * (f + 1) = e * (d + 1)⇧
(* 1 *) ⇧¬ a * (f + 1) = e * (b + 1)⇧

(If this were done in the proof context lin_arith, we would get an unwanted case split at this point.) We now see that, if we multiply assumption 1 by (d + 1), assumption 2 by (b + 1) and assumption 3 by (f + 1), then a number of like terms will appear which the linear arithmetic decision procedure should be able to take advantage of, and indeed the goal may be solved by the following tactic:

list_scale_nth_asm_tac[(⇧d+1⇧, 1), (⇧b+1⇧, 2), (⇧f+1⇧, 3)]

THEN PC_T1*lin_arith*asm_prove_tac[]

N.B. this tactic strengthens the goal in general, i.e. it may lead to unprovable subgoals even when the original goal was provable (since the scale factor may not be provably non-zero).

Errors  as for scale_rule and DROP_ASM_T and its variants
val scale_rule : TERM -> THM -> THM

Description  In the simplest cases, given a term, $t$, of type $\Gamma : \mathbb{N}$ and a theorem with conclusion of the form $(t1 : \mathbb{N}) = t2$ or $t1 \leq t2$, `scale_rule` returns a theorem which expresses the result of multiplying both sides of the conclusion of the theorem by $t$. More generally, `scale_rule t` processes theorems whose conclusions are atomic arithmetic propositions or negations of same as follows:

- $t1 = t2$ → $t \cdot t1 = t \cdot t2$
- $t1 \leq t2$ → $t \cdot t1 \leq t \cdot t2$
- $t1 < t2$ → $t \cdot t1 + t \leq t \cdot t2$
- $t1 \geq t2$ → $t \cdot t2 \leq t \cdot t1$
- $t1 > t2$ → $t \cdot t2 + t \leq t \cdot t1$
- $\neg t1 = t2$ → $t \cdot t1 + t \leq t \cdot t2 \lor t \cdot t2 + t \leq t \cdot t1$
- $\neg t1 \leq t2$ → $t \cdot t2 + t \leq t \cdot t1$
- $\neg t1 < t2$ → $t \cdot t2 \leq t \cdot t1$
- $\neg t1 > t2$ → $t \cdot t2 \leq t \cdot t1$
- $\neg t1 \geq t2$ → $t \cdot t2 \leq t \cdot t1$

Thus in the first two cases `scale_rule t` just scales the equation or inequality in the theorem by the factor $t$ and in the other cases the theorem is first converted to one or more inequalities with $\leq$ and then scaled by $t$.

Errors
82010  ?0 does not have type $\Gamma : \mathbb{N}$
82011  ?0 is not of the form \( \Gamma \vdash (t1 : \mathbb{N}) R t2 \) or \( \neg (t1 : \mathbb{N}) R t2 \) where $R$ is one of \( =, \leq, <, \geq \) or \( > \)

See Also  `scale_asm_tac`
9.2. Theory Related ML Values

SML

\[
\begin{align*}
\text{val } & \text{N\_eq\_cancel\_conv} : \text{CONV} \\
\text{val } & \text{\le\_cancel\_conv} : \text{CONV}
\end{align*}
\]

**Description**  
\(\text{N\_eq\_cancel\_conv}\) (resp. \(\le\_cancel\_conv\)) normalises arithmetic equations (resp. inequalities formed with \(\le\)) by putting the left- and right-hand sides in additive normal form, in the sense of \(\text{anf\_conv}\), q.v., then cancelling like terms.

For example, the calls:

**ProofPower Input**

\[
\begin{align*}
\text{N\_eq\_cancel\_conv} \{(x + 2\ast y + 5\ast z + 3 = 1 + 6\ast y + z)\}; \\
\le\_cancel\_conv \{(x + 2\ast y + x + 3 \le y + 2 + 2\ast x + y)\};
\end{align*}
\]

produce the following output

**ProofPower Output**

\[
\begin{align*}
\text{val it = } & \vdash x + 2 \ast y + 5 \ast z + 3 = 1 + 6 \ast y + z \\ & \Leftrightarrow 2 + x + 4 \ast z = 4 \ast y : \text{THM} \\
\text{val it = } & \vdash x + 2 \ast y + x + 3 \le y + 2 + 2 \ast x + y \\ & \Leftrightarrow 1 \le 0 : \text{THM}
\end{align*}
\]

Note that, as in the last example, if the result of the normalisation has numeric literals on both left- and right-hand sides, then one of the literals will be 0. However, the truth value is not evaluated. \(\text{N\_eq\_conv}\) or \(\le\_conv\) may be used to perform the evaluation, where required.

**Uses**  
The conversions are intended for use in tactic and conversion programming. The normal interactive interface is via rewriting or stripping in the proof context \(\text{lin\_arith}\), which performs other useful simplifications. For example, the call

**ProofPower Input**

\[
\begin{align*}
\text{PC\_C1"lin\_arith"rewrite\_conv}[] \{(x + 2\ast y + x + 3 \le y + 2 + 2\ast x + y)\};
\end{align*}
\]

produces the following output:

**ProofPower Output**

\[
\begin{align*}
\text{val it = } & \vdash x + 2 \ast y + x + 3 \le y + 2 + 2 \ast x + y \\ & \Leftrightarrow F : \text{THM}
\end{align*}
\]

**Errors**

\[
\begin{align*}
82120 & \text{?0 is not of the form (t1:N) = t2 or is already in normal form} \\
82121 & \text{?0 is not of the form (t1:N) \le t2 or is already in normal form}
\end{align*}
\]

**See Also**  
\(\text{lin\_arith}, \text{N\_eq\_conv}, \le\_conv, \text{make\_}\le\_conv\)

9.2.6 Integers

SML

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

**Description**  
This provides the basic definitions proof support for the HOL theory of integers. It creates the theory \(\text{Z}\).
SML

(* Proof Context: "Z *)

**Description**  A component proof context for handling the basic arithmetic operations for \( \mathbb{Z} \).

Expressions and predicates treated by this proof context are constructs formed from:

\(+, *, -, abs, div, mod, \leq, <, \geq, >, =, \mathbb{N}\)

**Contents**

Rewriting:

\[\text{\( \mathbb{Z} \_plus \_conv, \mathbb{Z} \_times \_conv, \mathbb{Z} \_subtract \_minus \_conv \)}
\[\text{\( \mathbb{Z} \_abs \_conv, \mathbb{Z} \_div \_conv, \mathbb{Z} \_mod \_conv \)}
\[\text{\( \mathbb{Z} \_eq \_conv, \mathbb{Z} \_\leq \_conv, \mathbb{Z} \_less \_conv \)}
\[\text{\( \mathbb{Z} \_\geq \_\leq \_conv, \mathbb{Z} \_greater \_less \_conv, \)}
\[\text{\( \mathbb{Z} \_plus \_clauses, \mathbb{Z} \_minus \_clauses, \mathbb{Z} \_\leq \_clauses \)}
\[\text{\( \mathbb{Z} \_less \_clauses, \mathbb{Z} \_\neg \_\leq \_thm, \mathbb{Z} \_\neg \_less \_thm \)}

Stripping theorems:

\[\text{\( \mathbb{Z} \_eq \_conv, \mathbb{Z} \_\leq \_conv, \mathbb{Z} \_less \_conv \)}
\[\text{\( \mathbb{Z} \_\geq \_\leq \_conv, \mathbb{Z} \_greater \_less \_conv, \)}
\[\text{\( \mathbb{Z} \_plus \_clauses, \mathbb{Z} \_minus \_clauses, \mathbb{Z} \_\leq \_clauses \)}
\[\text{\( \mathbb{Z} \_less \_clauses, \)}
\[\text{\( \text{and all the above pushed through } \neg \)}
\[\text{\( \mathbb{Z} \_\neg \_\leq \_thm, \mathbb{Z} \_\neg \_less \_thm, \mathbb{Z} \_\leq \_conv, \mathbb{Z} \_less \_conv \)}

Stripping conclusions: as for stripping theorems.

Rewriting canonicalisation: blank.

Automatic proof procedures: *basic_prove_tac, basic_prove_conv*. Automatic existence prover: blank.
### Theory Related ML Values

#### SML

```sml
val dest_Z_≤ : TERM -> TERM * TERM;
val dest_Z_≥ : 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_pluss : TERM -> TERM * TERM;
val dest_Z_subtract : TERM -> TERM * TERM;
val dest_Z_times : TERM -> TERM * TERM;
```

#### Description

These are derived destructor functions for the basic arithmetic operations on the integers. 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 \( \mathbb{Z} \) with alphabetic names assigned to give valid ML name as needed (`greater`: `<`, `less`: `>`, `minus`: `∼`, `plus`: `+`, `subtract`: `−`, `times`: `∗`).

As usual, there are also corresponding discriminator (`is_...`) and constructor functions (`dest_...`). For programming convenience, `dest_Z_signed_int` returns 0 and `mk_Z_signed_int` returns `true` when applied to `∼0`, but `mk_Z_signed_int` cannot be used to construct such a term.

#### Errors

93101 `?0` is not of the form `⌜i ≤ Z j⌝`
93102 `?0` is not of the form `⌜i ≥ Z j⌝`
93103 `?0` is not of the form `⌜Abs Z i⌝`
93104 `?0` is not of the form `⌜i Div Z j⌝`
93105 `?0` is not of the form `⌜i > Z j⌝`
93106 `?0` is not of the form `⌜i < Z j⌝`
93107 `?0` is not of the form `⌜∼Z i⌝`
93108 `?0` is not of the form `⌜i Mod Z j⌝`
93109 `?0` is not of the form `⌜i + Z j⌝`
93110 `?0` is not an optionally signed integer literal (theory \( \mathbb{Z} \))
93111 `?0` is not of the form `⌜i − Z j⌝`
93112 `?0` is not of the form `⌜i ∗ Z j⌝`

---

#### SML

```sml
val Is_Z_Rep_def : THM;
val Z_minus_def : THM;
val Z_subtract_def : THM;
val Z_≤_def : THM;
val Z_≥_def : THM;
val Z_abs_def : THM;
val Z_div_def : THM;
```

#### Description

These are the ML bindings of the definitions of the theory \( \mathbb{Z} \).

---

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
### SML

```sml
val is_Z_≤ : TERM -> bool;
val is_Z_≥ : 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.

### SML

```sml
val mk_Z_≤ : TERM * TERM -> TERM;
val mk_Z_≥ : 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**

```
93201  ?0 does not have type Z
```
These are the ML value bindings for the theorems saved in the theory \( \mathbb{Z} \).
The term argument must be a variable that appears free in the conclusion of the goal. The variable must also appear free once, and only once, in the assumptions, in an assumption of the form \( j \leq x \).

**Tactic**

\[
\Gamma, j \leq x \vdash t[x] \\
\text{strip } \{ j \leq x, \forall k \cdot j \leq k \land k < x \Rightarrow t[k/x], \Gamma \} \vdash t[x] \\
\text{Z}_{\leq \text{induction_tac}} \vdash x \neg
\]  

**Errors**

93401 ?0 is not a variable of type \( \vdash \mathbb{Z} \)
93402 A term of the form \( \vdash \mathbb{Z} j \leq i \) where \( i \) is the induction variable could not be found in the assumptions
93403 ?0 appears free in more than one assumption of the goal
93404 ?0 does not appear free in the conclusions of the goal

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.

**Tactic**

\[
\{ \Gamma \} \vdash t[i/x] ; \\
\text{strip } t[i/x], \Gamma \vdash t[\sim i/x] ; \\
\text{strip } t[i/x] \land t[j/x], \Gamma \vdash t[i+j/x] \\
\text{Z}_{\text{N_induction_tac}} \vdash x \neg
\]  

**Errors**

As for \( \text{gen_induction_tac} \).
9.2. Theory Related ML Values

SML

val \( \mathbb{Z}_{\leq} \) induction_tac : TERM \rightarrow TACTIC

Description This tactic implements induction over subsets of the integers that are bounded below: to prove that \( t[x] \) holds whenever \( x \geq j \), it suffices to prove \( t[j/x] \) and to prove \( t[x + 1/x] \) on the assumption that \( t[x] \) and \( x \geq j \).

The term argument must be a variable that appears free in the conclusion of the goal. The variable must also appear free once, and only once, in the assumptions, in an assumption of the form \( j \geq x \).

Tactic \[ \{ \Gamma; j \leq x \} \ t \] \[ \{ \Gamma \} \ t[j/x] ; \text{strip}\{t, j \leq x, \Gamma\} \ t[x+1/x] \] \( \mathbb{Z}_{\leq} \) induction_tac \( \Gamma \)

See Also \( \mathbb{Z} \) cases_thm, intro \( \forall \) tac, \( \mathbb{Z} \) induction_tac, \( \mathbb{Z}_{\leq} \) induction_tac, \( \mathbb{Z} \) \( \mathbb{N} \) induction_tac

Errors

93401 \( ?0 \) is not a variable of type \( \text{\( \mathbb{Z} \)} \)
93402 A term of the form \( \text{\( \mathbb{Z} \) j} \leq i \) where \( i \) is the induction variable could not be found in the assumptions
93403 \( ?0 \) appears free in more than one assumption of the goal
93404 \( ?0 \) does not appear free in the conclusions of the goal

SML

val \( \mathbb{Z}_{\mathbb{N}} \) induction_tac : TACTIC

Description This tactic implements induction over the natural numbers (as a subset of the HOL integers): to prove \( \text{NZ 0} \leq x \Rightarrow t \), 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 \( \text{NZ 0} \leq x \Rightarrow t \).

Tactic \[ \{ \Gamma \} \ \text{NZ 0} \leq x \Rightarrow t \] \[ \{ \Gamma \} \ t[0/x] ; \text{strip}\{t, \Gamma\} \ t[x+1/x] \] \( \mathbb{Z}_{\mathbb{N}} \) induction_tac

See Also \( \mathbb{Z} \) cases_thm, intro \( \forall \) tac, \( \mathbb{Z} \) induction_tac, \( \mathbb{Z}_{\leq} \) induction_tac, \( \mathbb{Z} \) \( \mathbb{N} \) 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 $\mathbb{Z}$ and other proof contexts which contain it.

The first block above gives conversions to evaluate expressions of the form $i \text{ op } j$ where $i$ and $j$ are numeric literals and $\text{ op }$ is one of $+$ or $\ast$. The second block gives conversions to transform terms of the form $i - j$, $i > j$, $i > j$ and $i \in \mathbb{N}$ into $i + \sim j$, $j < i$, $j \leq i$ and $0 \leq i$ respectively. The third block give conversions which evaluate expressions of the form $i \text{ op } j$ or $\text{ abs } i$, where $\text{ op }$ is one of $+$, $\ast$, $\div$, $\mod$, $\leq$, $<$, 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

\begin{align*}
93301 & \text{ ?0 is not of the form } ?1 \text{ where } \langle i \rangle \text{ and } \langle j \rangle \text{ are numeric literals (theory } \mathbb{Z}) \\
93302 & \text{ ?0 is not of the form } ?1 \text{ (theory } \mathbb{Z}) \\
93303 & \text{ ?0 is not of the form } ?1 \text{ where } \langle i \rangle \text{ and } \langle j \rangle \text{ are optionally signed numeric literals (theory } \mathbb{Z})
\end{align*}

9.2.7 Integer Proof Support

This is the signature of a structure containing arithmetic and an automatic linear arithmetic prover for the HOL integers (as defined in the theory $\mathbb{Z}$).
9.2. Theory Related ML Values

SML

(* Proof Context: Z_lin_arith *)

Description  This is a proof context whose main purpose is to supply a decision procedure for problems in linear arithmetic over the HOL integers.

Contents  The rewriting, theorem stripping and conclusion stripping components are as for the proof context predicates, q.v, each augmented with conversions effecting the following transformations, where $a$ and $b$ stand for numeric literals.

\[
\begin{align*}
  a = b & \rightarrow \ Z_{eq\_conv} a = b \wedge \\
  a \leq b & \rightarrow \ Z_{\leq\_conv} a \leq b \\
  t1 = t2 & \rightarrow \ Z_{eq\_cancel\_conv} t1 = t2 \wedge \\
  t1 \leq t2 & \rightarrow \ Z_{\leq\_cancel\_conv} t1 \leq t2 \\
  \neg t1 = t2 & \rightarrow t1 < t2 \lor t2 < t1 \\
  \neg t1 \leq t2 & \rightarrow t2 < t1 \\
  t1 < t2 & \rightarrow t1 + t1 \leq t2 \\
  t1 \geq t2 & \rightarrow t2 \leq t1 \\
  t1 > t2 & \rightarrow t2 < t1 \\
  \neg t1 < t2 & \rightarrow t2 \leq t1 \\
  \neg t1 \geq t2 & \rightarrow \neg t2 \leq t1 \\
  \neg t1 > t2 & \rightarrow t1 \leq t2 \\
  t1 = t2 & \rightarrow t1 = t2 \\
  t1 \leq t2 & \rightarrow t1 \leq t2
\end{align*}
\]

(\text{where all variables are of type } Z).\]

The automatic proof tactic works by (i) restripping all the assumptions of the goal, (ii) adding the argument theorems to the stock of assumptions using strip_asm_tac, (iii) applying contr_tac, and (iv) searching for a linear combination of the assumptions which will reduce, by multiplying out and cancelling like terms, to a contradiction of the form $a = b$ or $a \leq b$ with $a$ and $b$ numeric literals. The automatic proof conversion just tries to prove its argument, $t$ say, using the automatic proof tactic and returns $t = T$ if it succeeds.

Other components of the proof context are as for predicates.

For example, $PC_{.T1}"Z_lin_arith"prove_tac[]$ will prove any of the following goals:

\[
\begin{align*}
  ([]), \forall a \ b \ c : Z \cdot a \leq b \land (a + b < c + a) \Rightarrow a < c \wedge \\
  ([]), \forall a \ b \ c : Z \cdot a \geq b \land \neg b < c \Rightarrow a \geq c \wedge \\
  ([]), \forall a \ b \ c : NZ \cdot 2 \cdot b < NZ \cdot 2 \cdot a \Rightarrow b + b < a \wedge \\
  ([]), \forall x \ y : Z \cdot \neg (NZ \cdot 2 \cdot x + y = NZ \cdot 4 \land NZ \cdot 4 \cdot x + NZ \cdot 2 \cdot y = NZ \cdot 7) \wedge
\end{align*}
\]

See Also  $'Z_lin_arith$

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
Chapter 9. THEORIES

SML

(* Proof Context: 'Z_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 HOL integers.

Contents  The rewriting, theorem stripping and conclusion stripping components are as for the proof context 'Z_lin_arith but without any of the material from the proof context predicates. The automatic proof components are as for lin_arith. Other components are blank.

A typical use of the proof context would be to solve problems containing a mixture of (linear) arithmetic and set theory.

Errors  82200  Could not prove theorem with conclusion ?1 (?0)

SML

val Z_anf_conv : CONV;
val Z_ANF_C : CONV -> CONV;

Description  Z_anf_conv is a conversion which proves theorems of the form ⊢ t1 = t2 where t1 is a term formed from atoms of type Z 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 ti and, within them, the factors sj are given in increasing order with respect to the ordering on terms given by the function 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.

Z_ANF_C conv is a conversion which acts like Z_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 105032 if there are no changes to be made to the term.

Errors  105032 0 is not of type "Z" or is already in additive normal form
9.2. Theory Related ML Values

SML

```sml
val Z_eq_cancel_conv : CONV
val Z_le_cancel_conv : CONV
```

**Description**  
\(Z_{\text{eq-cancel-conv}}\) (resp. \(Z_{\text{le-cancel-conv}}\)) puts arithmetic equations (resp. inequalities formed with \(\leq\)) in a normal form in which the right-hand side is a signed literal and the left-hand side is in additive normal form, in the sense of \(Z_{\text{anf-conv}}\), q.v.

For example, the calls:

**ProofPower Input**

\[
\begin{align*}
Z_{\text{eq-cancel-conv}} & : x + NZ \ 2 * y + NZ \ 3 = NZ \ 1 + NZ \ 6 * y; \\
Z_{\text{le-cancel-conv}} & : x + NZ \ 2 * y + x + NZ \ 3 \leq y + NZ \ 2 + NZ \ 2 * x + y;
\end{align*}
\]

produce the following output

**ProofPower Output**

\[
\begin{align*}
\text{val it = } & \vdash x + NZ \ 2 * y + NZ \ 3 = NZ \ 1 + NZ \ 6 * y \\
& \iff x + \sim (NZ \ 4) * y = \sim (NZ \ 2) : \text{THM} \\
\text{val it = } & \vdash x + NZ \ 2 * y + x + NZ \ 3 \leq y + NZ \ 2 + NZ \ 2 * x + y \\
& \iff NZ \ 1 \leq NZ \ 0 : \text{THM}
\end{align*}
\]

Note that if the left-hand side reduces to 0 the truth value is not evaluated. However, \(Z_{\text{eq-conv}}\) or \(Z_{\text{le-conv}}\) may be used to perform the evaluation, where required.

**Uses**  
The conversions are intended for use in tactic and conversion programming. The normal interactive interface is via rewriting or stripping in the proof context \(Z_{\text{lin-arith}}\), which performs other useful simplifications.

**Errors**

105120 ? 0 is not of the form \((t_1 : \mathbb{Z}) = t_2\) or is already in normal form
105121 ? 0 is not of the form \((t_1 : \mathbb{Z}) \leq t_2\) or is already in normal form

**See Also**  
lin_arith, \(Z_{\text{lin-arith}}\), \(Z_{\text{eq-conv}}\), \(Z_{\text{le-conv}}\)

SML

```sml
val Z_lin_arith_rule : TERM list -> THM;
```

**Description**  
Given a system, \(\Gamma = [r_1, r_2, \ldots]\), of numeric constraints, \(r_i\), of the form \((s_i : \mathbb{Z}) = NZ \ 0\) or \(s_i \leq NZ \ 0\) these rules attempt to prove a theorem of the form \(\Gamma \vdash F\). Terms in \(\Gamma\) which are not of either of these forms are ignored and do not appear in the assumptions of the result theorem.

The usual interface to these rules is via the decision procedures in the proof context \(Z_{\text{lin-arith}}\), q.v., e.g. as invoked by \(PC \_ \text{T1}"Z_{\text{lin-arith}}"\_\text{prove_tac}[]\)

The algorithm for the decision procedure is very similar to the one used in \(lin\_arith\_rule\), q.v. The only significant difference is that there is no opportunity with the integers to add in assumptions that all atoms are non-negative.

**Errors**

82110 System is satisfiable
82111 A system with no constraints is satisfiable
105112 No constraints of the form \(\vdash (t_1 : \mathbb{Z}) \leq t_2\) or \(\vdash (t_1 : \mathbb{Z}) = t_2\) could be derived from the assumptions
\texttt{val Z\_lin\_arith\_tac : TACTIC;}

\textbf{Description} This tactic is primarily intended for use in conjunction with the integer linear arithmetic proof contexts \texttt{Z\_lin\_arith} and \texttt{'Z\_lin\_arith}, q.v. The normal interface to the tactics is via the decision procedures in these proof contexts, as in for example: \texttt{PC\_TI"Z\_lin\_arith"prove\_tac].

The tactics do however, have possible applications in specialised tactic programming, in which circumstances their behaviour may be understood from their definition, in terms of \texttt{Z\_lin\_arith\_rule}, q.v., essentially as:

\texttt{val Z\_lin\_arith\_tac = GET\_ASMS\_T (f\_thm\_tac o Z\_lin\_arith\_rule o map concl);}

\textbf{Uses} The most likely application is in specialised tactic programming in situations where it is known that the assumptions are already in the normal form produced by the proof context \texttt{lin\_arith} and it is important for performance not to restrip them.

\texttt{val Z\_term\_order : TERM -> TERM -> int;}

\textbf{Description} \texttt{Z\_term\_order} gives an ordering relation on HOL terms analogous to that given by \texttt{term\_order}, q.v., but which takes special arrangements for certain terms of type \texttt{Z}. In particular it ensures that a ‘monomial’ (formed using integer multiplication) without a sign immediately precedes the same monomial with a sign, which in turn precedes the same monomial with a signed literal multiplier, e.g.:

\[ a * b < \sim a * b < 0 * a * b < \sim 1 * a * b < 1 * a * b < .... \]

### 9.2.8 Lists

\texttt{signature List = sig}

\textbf{Description} This is the signature in which the theory “list” is declared.

\texttt{val is\_list\_rep\_def : THM
val nil\_cons\_def : THM
val length\_def : THM
val hd\_def : THM
val tl\_def : THM
val append\_def : THM
val map\_def : THM
val fold\_def : THM
val split\_def : THM
val combine\_def : THM
val rev\_def : THM}

\textbf{Description} These Standard ML variables are bound to the definitions in the theory “list”.

\texttt{val list\_clauses : THM
val list\_cases\_thm : THM
val list\_induction\_thm : THM
val list\_prim\_rec\_thm : THM}

\textbf{Description} These Standard ML variables are bound to the theorems in the theory “list”.

\copyright Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUAL USR029
9.2. Theory Related ML Values

SML

\[
\text{val list\_induction\_tac : TERM \to TACTIC}
\]

**Description**  This tactic implements induction over lists: to prove \(t[x]\) it suffices to prove \(t[\text{Nil}]\) and to prove \(t[\text{Cons } h \ x]\) on the assumption that \(t[x]\). The term argument must appear free in the conclusion of the goal but not in its assumptions.

Tactic

\[
\begin{align*}
\{ \Gamma \} & \quad t[x] \\
\{ \Gamma \} & \quad t[\text{Nil}] ; \text{strip}(t[x], \Gamma) \quad t[\text{Cons } h \ x]
\end{align*}
\]

**See Also**  \textsc{LIST\_INDUCTION\_T}

**Errors**

39001  ?0 is not a term variable
39002  ?0 does not appear free in the conclusion of the goal
39003  ?0 appears free in the assumptions of the goal

\[
\text{val LIST\_INDUCTION\_T : TERM \to (THM \to TACTIC) \to TACTIC;}
\]

**Description**  This implements induction over lists as a tactical. The term argument must appear free in the conclusion of the goal but not in its assumptions. The inductive hypothesis is passed to the tactic generating function given by the second argument.

Tactic

\[
\begin{align*}
\{ \Gamma \} & \quad t[x] \\
\{ \Gamma \} & \quad t[\text{Nil}] ; \text{ttac}(t[x] \vdash t[x]) \quad \text{LIST\_INDUCTION\_T } \vdash x \}\quad \text{ttac}
\end{align*}
\]

**Uses**  Most commonly used with \texttt{asm\_tac} to avoid the stripping up of the inductive hypothesis which occurs with \texttt{list\_induction\_tac}.

**See Also**  \texttt{list\_induction\_tac}

**Errors**

39001  ?0 is not a term variable
39002  ?0 does not appear free in the conclusion of the goal
39003  ?0 appears free in the assumptions of the goal

9.2.9 Characters and Strings

SML

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

**Description**  This is the signature for the theory of characters and strings.

SML

\[
\begin{align*}
\text{val is\_char\_rep\_def : THM} \\
\text{val char\_def : THM} \\
\text{val abs\_char\_rep\_char\_def : THM}
\end{align*}
\]

**Description**  These Standard ML variables are bound to the definitions in the theory “char”.

© Lemma 1 Ltd.  2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
9.2.10 Set Theory

```sml
signature SetsTheory = sig
val set_def : THM;
val sets_clauses : THM;
val ∅_clauses : THM;
val ∈_in_clauses : THM;
val sets_ext_clauses : THM;
val complement_clauses : THM;
val ∪_clauses : THM;
val ∩_clauses : THM;
val set_dif_clauses : THM;
val ∅_clauses : THM;
val ⊆_clauses : THM;
val ∈_clauses : THM;
val ∪_clauses : THM;
val ∩_clauses : THM;
val ＃_clauses : THM;
val is_set_rep_def : THM;
val set_comp_def : THM;
val insert_def : THM;
val complement_def : THM;
val ∪_def : THM;
val ∩_def : THM;
val set_dif_def : THM;
val ∅_def : THM;
val ⊆_def : THM;
val ∈_def : THM;
val ∪_def : THM;
val ∩_def : THM;
val ＃_def : THM;
val P_def : THM;
```

**Description**  This is the signature in which we declare theory “sets”. `sets_clauses` is bound the theorem saved with the same name as key, `insert_def` is bound to the definition of Insert, saved with key Insert, and the other `_def` follow the same pattern.
Description These conversions use the extensional property of sets to transform terms of the form $\{a = b\}$, $\{a \subseteq b\}$ or $\{a \subset b\}$ (where $a$ and $b$ are sets) into assertions about membership of $a$ and $b$. The conversions introduce variable structures in the membership assertions if the elements of $a$ and $b$ are pairs (i.e. have types of the form: $\sigma \times \tau$).

If the elements are not pairs, the conversions behave as follows:

**Conversion**

\[
\vdash (a = b) \iff (\forall x \cdot x \in a \iff x \in b)
\]

\[
\vdash (a \subseteq b) \iff (\forall x \cdot x \in a \Rightarrow x \in b)
\]

\[
\vdash (a \subset b) \iff (\forall x \cdot x \in a \Rightarrow x \in b) \land (\exists x \cdot \neg x \in a \land x \in b)
\]

where $x$ represents a variable whose name is “$x$” decorated to avoid clashing with the free variables of $a$ or $b$.

If the elements of $a$ and $b$ are pairs, then a variable structure (whose free variables have names chosen from the sequence “$x1$”, “$x2$”, . . . so as not to clash with any of the variables in $a$ or $b$) is used instead of $x$ on the left hand side of each $\in$, and the quantification is over the free variables of the variable structure. For example:

**Example**

\[
\vdash \{(1, 2), 3\} \subseteq x2 \\
\Rightarrow (\forall x1 x3 x4 \cdot ((x1, x3), x4) \in \{(1, 2), 3\}) \\
\Rightarrow ((x1, x3), x4) \in x2
\]

Here, $x2$ has not been used in the variable structure since it occurs in the input term.

(The distinction between the variable naming rules followed in the simple case and the paired case has some mnemonic value and makes the conversions compatible with rewriting with the theorem `sets_ext clauses` as done, for example, in the proof context `sets_ext`.)

**Errors**

44021 ?0 is not of the form: $\{a = b\}$ where $\{a\}$ and $\{b\}$ are sets
44022 ?0 is not of the form: $\{a \subseteq b\}$
44023 ?0 is not of the form: $\{a \subset b\}$
**SML**

```sml
val sets_simple_∃conv : CONV;
```

**Description**  This conversion changes an existentially quantified term in a (perhaps nullary) function whose range is a set to an existentially quantified term in a function whose whose range is a function from the type of the set element to \(\exists: BOOL\). It is only applicable if all instances of the bound function in the body of the term are applied to sufficient arguments and then tested for set membership by \(\in\).

```
\[
\begin{align*}
\Gamma & \vdash \exists f : \ldots \to ty \ \text{SET} \bullet \\
& \quad \quad P[ t1 \in f, \ t2 \in f \ldots, \ldots ] \\
\Leftrightarrow & \\
& \exists f' : \ldots \to ty \ \text{SET} \bullet \\
& \quad \quad P[ f' \ldots t1, f' \ldots t2, \ldots ]^1
\end{align*}
\]
```

where the \(t_i\) are arbitrary terms.

**Example**

```sml
sets_simple_∃conv \(\exists S \bullet \forall x \ y \bullet x \in S \ y \Leftrightarrow \neg(x = y)^1\) 
\(\vdash (\exists S' \bullet \forall x \ y \bullet S' y x \Leftrightarrow \neg(x = y)) \Leftrightarrow (\exists S' \bullet \forall x \ y \bullet S' y x \Leftrightarrow \neg(x = y))\)
```

**Uses**  This function is used to implement a automated existence proof preprocessor for the “sets” proof context.

**Errors**

- 44010 0 is not of the form: \(\exists f : \ldots \to ty \ \text{SET} \bullet \ldots\)^1
- 44011 0 is not of the form: \(\exists f \bullet \ldots \ f \ldots\)^1
- 44012 0 has instances of \(?1 not embedded in subterms of the form: \(t \in ?1 \ldots\)^1
- 44013 Unable to prove \(?0

**SML**

```sml
val simple_∈_comp_conv : CONV;
```

**Description**  A conversion for set membership. It cannot handle variable structures bound by the set comprehension.

```
\[
\begin{align*}
\Gamma & \vdash x \in \{ v \mid p[v]\} \Leftrightarrow p[x] \\
\end{align*}
\]
```

**See Also**  ∈_comp_conv

**Errors**

- 44001 0 is not of the form: \(x \in \{ v \mid p[v]\}\)^1
9.2. Theory Related ML Values

**SML**

\[ \text{val } \in_{\text{comp\_conv}} : \text{CONV}; \]

**Description** A conversion for set membership. It can handle variable structures bound by the set comprehension.

\[ \vdash t \in \{ vs[x,y,...]|p[x,y,...]\} \quad \in_{\text{comp\_conv}} \quad \neg t \in \{ v[x,y,...]|p[x,y,...]\} \]

where \( x1, y1, \text{etc.} \) are the appropriate components of \( t \), extracted via \( \text{Fst} \) and \( \text{Snd} \).

**See Also** simple \( \in_{\text{comp\_conv}} \)

**Errors**

\[ 27002 \quad ?0 \text{ is not of form: } x \in \{ v | p[v] \} \neg \]

**SML**

\[ \text{val } \in_{\text{enum\_set\_conv}} : \text{CONV}; \]

**Description** Give that something within an enumerated set is a member of that set.

\[ \vdash ti \in \{ t1, ..., tn \} \iff T \quad \in_{\text{enum\_set\_conv}} \quad \neg ti \in \{ t1, ..., tn \} \neg \]

where \( ti \) is \( \alpha \)-convertible to one of \( t1, ..., tn \).

**Errors**

\[ 27005 \quad ?0 \text{ not a member of the enumerated set } ?1 \]
\[ 27006 \quad ?0 \text{ is not of the form: } \neg x \in \{ t1, ..., tn \} \neg \]

Message 3012 occurs when the term list cannot be made into an enumerated set because of differing types.

**SML**

\[ \text{val } \in_{\text{enum\_set\_rule}} : \text{TERM } \rightarrow \text{TERM list } \rightarrow \text{THM}; \]

**Description** Give that something within an enumerated set is a member of that set.

\[ \vdash ti \in \{ t1, ..., tn \} \quad \in_{\text{enum\_set\_rule}} \quad \neg ti \neg \]

where \( ti \) is \( \alpha \)-convertible to one of \( t1, ..., tn \).

**Errors**

\[ 3012 \quad ?0 \text{ and } ?1 \text{ do not have the same types} \]
\[ 27001 \quad ?0 \text{ not a member of list of terms} \]

Message 3012 occurs when the term list cannot be made into an enumerated set because of differing types.

**9.2.11 One, Combin, and Sum**

**SML**

\[ \text{signature BasicHolTheory } = \text{sig} \]
\[ \text{end}; \]

**Description** This is the signature in which we declare theory “basic_hol”. The theory itself is empty.
490 Chapter 9. THEORIES

SML
signature CombinTheory = sig
val comb_s_def : THM;
val comb_k_def : THM;
val comb_i_def : THM;
val o_def : THM;
val o_assoc_thm : THM;
val o_i_thm : THM;
end;

Description In the signature of CombinTheory in which we declare theory “combin” we bind the definitions of CombS, CombK, CombI and o to comb_s_def, comb_k_def, comb_i_def and o_def respectively. The theorem saved with key “o_assoc_thm” is bound to the ML value of the same name, as is “o_i_thm”.

SML
signature HolTheory = sig
end;

Description This is the signature in which we declare theory “hol”. The theory itself is empty.

SML
signature OneTheory = sig
val is_one_rep_def : THM;
val one_def : THM;
val one_def1 : THM;
val one_fns_thm : THM;
end;

Description This is the signature in which we declare theory “one”. one_def is the definition of One, saved with the key “One”. one_def1 is the type definition of ONE, saved with the key “ONE”. one_fns_thm is the theorem saved with key “one_fns_thm”.

SML
signature SumTheory = sig
val is_sum_rep_def : THM;
val sum_def : THM;
val sum_clauses : THM;
val sum_cases_thm : THM;
val sum_fns_thm : THM;
end;

Description sum_clauses is bound to the definition of InL. sum_cases_thm and sum_fns_thm are bound to the theorems with those names as their key.
9.2. Theory Related ML Values

### Description
This tactic implements case splitting over sum types: to prove \( t[x] \) it suffices to prove \( t[\text{InLy}] \) and to prove \( t[\text{InRz}] \) (for newly introduced variable \( y \) and \( z \)). The term argument must be a variable of type \( \Gamma : \text{ty1 + ty2} \) and must appear free in the conclusion of the goal but not in its assumptions.

#### Tactic
\[
\frac{\Gamma \vdash P \[ x \] \\
\Gamma \vdash P \[ \text{InL} \ x' \], \\
\Gamma \vdash P \[ \text{InR} \ x'' \]}
{\text{sum_cases_tac}} \quad \Gamma : \text{ty1 + ty2}
\]
where \( x' \) and \( x'' \) are variables whose names are variants of \( x \), and types are \( \text{ty1} \) and \( \text{ty2} \) respectively.

#### Errors
- \( 45002 \) ?0 is not a variable of type \( \Gamma : \text{ty1 + ty2} \)
- \( 38002 \) ?0 does not appear free in the conclusion of the goal
- \( 38003 \) ?0 appears free in the assumptions of the goal

---

### Description
A conversion to break a term that is universally quantified by a sum type variable into two terms universally quantified by variables of the constituent types.

#### Conversion
\[ \vdash (\forall x : \text{ty1 + ty2} \bullet P[x]) \iff \forall \text{sum_conv} \quad \Gamma : \text{ty1 + ty2}
\]
where \( x' \) and \( x'' \) are variables whose names are variants of \( x \), and types are \( \text{ty1} \) and \( \text{ty2} \) respectively.

#### Errors
- \( 45001 \) ?0 is not of the form \( \forall x : \text{ty1 + ty2} \bullet P[x] \)
9.2.12 Binary Relations

SML

signature BinRelTheory = sig
    val ←_def : THM;
    val ×_def : THM;
    val ←_def : THM;
    val dom_def : THM;
    val ran_def : THM;
    val id_def : THM;
    val ¬_def : THM;
    val r_¬_r_def : THM;
    val r_o_r_def : THM;
    val _def : THM;
    val dom_def : THM;
    val ran_def : THM;
    val id_def : THM;
    val ¬_def : THM;
    val graph_def : THM;
    val inv_rel_def : THM;
    val image_def : THM;
    val ⊕_def : THM;
    val reflexive_def : THM;
    val symmetric_def : THM;
    val transitive_def : THM;
    val injective_def : THM;
    val surjective_def : THM;
    val total_def : THM;
    val functional_def : THM;
    val tc_def : THM;
    val rtc_def : THM;
    val rel_combine_def : THM;
    val rel ∈_in_clauses : THM;
    val inv_rel_thm : THM;
    val bin_rel_∅_universe_thm : THM;
    val bin_rel_insert_thm : THM;

Description  This is the signature in which we declare theory “bin_rel”.
9.2. Theory Related ML Values

(* Proof context key "bin_rel" *)

**Description**  A component proof context for theory `bin_rel`, “aggressively” converting problems involving the vocabulary of the theory `bin_rel` into that of the theory `sets`.

**Contents**  Rewriting: `rel ∈ in_clauses`, `inv_rel_thm` and the definitions of the constants defined in the theory `bin_rel`. Stripping theorems: `rel ∈ in_clauses`, `inv_rel_thm` (also pushed in through `¬`).

Stripping conclusions: `rel ∈ in_clauses`, `inv_rel_thm` (also pushed in through `¬`).

Automatic proof procedures are `basic_prove_tac` and `basic_prove_conv`.

**Usage Notes**  Should not be used in conjunction with component proof context "`sets_ext`", or with a complete proof context which includes it. Requires theories `bin_rel` and `sets`. Intended for use in conjunction with a proof context such as `sets_alg` or `sets_ext` which is capable of solving the resulting set-theoretic conjectures.

### 9.2.13 Functional Relations

(* SML signature FunRelTheory = sig

  val →_def : THM;
  val →→_def : THM;
  val →→_def : THM;
  val →→_def : THM;
  val →→_def : THM;
  val →→_def : THM;
  val graph_at_thm : THM;
  val inv_rel_∈_arrow_thm : THM;
  val at_at_eq_thm : THM;

  Description  This is the signature in which we declare theory “fun_rel”.

(* proof context key "fun_rel_ext" *)

**Description**  This proof context extends "`bin_rel_ext`" by:

**Name**  Becomes “fun_rel_ext”.

**Stripping Goals and Theorems**

Adding in `?` applied at the top level or under a single negation.

**Rewriting Context**

Adding in `?`.
Chapter 9. THEORIES

### 9.2.14 Finite Sets

**SML**

```sml
signature FinSetTheory = sig

val N_def : THM;
val F_def : THM;
val min_def : THM;
val size_def : THM;
val ⇔_def : THM;
val finite_induction_thm : THM;
val singleton_union_finite_thm : THM;
val ∪_finite_thm : THM;
val finite_distinct elems_thm : THM;
val length_|≤_thm : THM;
val distinct_length_|≤_thm : THM;
val size_empty_thm : THM;
val size_singleton_thm : THM;
val size_0_thm : THM;
val ∪_finite_thm : THM;
val ≤_size thm : THM;
val min_∈ thm : THM;
val max_∈ thm : THM;
val finite⊆ well-founded_thm : THM

val finite_def : THM;
val F1_def : THM;
val max_def : THM;
val iter_def : THM;
val ⇔_def : THM;
val empty_finite_thm : THM;
val ⊆_finite_thm : THM;
val ∩_finite_thm : THM;
val length_|≤_thm : THM;
val elems_|≤_thm : THM;
val distinct_size_length_thm : THM;
val size_singleton_∪_thm : THM;
val size_∪_thm : THM;
val size_0_thm : THM;
val pigeon_hole_thm : THM;
val size less_thm : THM;
val ≤_max_thm : THM;

val size empty_thm : THM;
val size singleton thm : THM;
val size∪ thm : THM;
val finite⊆ well-founded_thm : THM

val size 0 thm : THM;
val size 1 thm : THM;
val pigeon hole thm : THM;
val size less_thm : THM;
val min_≤ thm : THM;
val ≤ max thm : THM;
```

**Description** These are the ML bindings for the definitions and theorems in the declare theory "fin_set".

---

*A proof context key "fun_rel_alg"*)

**Description** `fun_rel_alg` extends `relation_alg` by:

**Name**

Becomes “fun_rel_alg”.

**Stripping Goals and Theorems**

Adding in ? applied at the top level or under a single negation.

**Rewriting Context**

Adding in ?.
9.2. Theory Related ML Values 495

**SML**

```sml
val finite_induction_tac : TERM -> TACTIC;
```

**Description** An induction tactic for finite sets. To prove \( t \) on the assumption that \( s \in \text{Finite} \), it suffices to prove \( t[\{\}\]/s \) and to prove \( t[(\{x\} \cup s)/s] \) on the assumption that \( t \) holds, that \( s \in \text{Finite} \), and that \( \neg x \in s \). The term argument must be a variable, \( s \), with type an instance of \( \tau : 'a \text{ SET} \) and must appear free in the conclusion of the goal. It must also appear once, and only once, in an assumption of the form \( s \in \text{Finite} \).

**Tactic**

\[
\begin{align*}
\{ \Gamma, s \in \text{Finite} \} & \quad t[s] \\
\{ \Gamma \} & \quad t[\{\}\]/s \\
\text{strip} & \quad \{ t, s \in \text{Finite}, \neg x \in s, \Gamma \} \quad t[(\{x\} \cup s)/s]
\end{align*}
\]

**Errors**

73001 ?0 is not a variable
73002 A term of the form \( \forall v \in \text{ Finite } \) where \( v \) is the induction variable could not be found in the assumptions
73003 ?0 does not appear free in the conclusion of the goal
73004 ?0 appears free in more than one assumption of the goal

---

9.2.15 Sequences

**SML**

```sml
signature SeqTheory = sig
  val elems_def : THM;
  val distinct_def : THM;
  val lists_def : THM;
  val lists1_def : THM;
  val inj_lists_def : THM;
  val nth_def : THM;
  val dot_dot_def : THM;
  val list_rel_def : THM;
  val rel_list_def : THM;
  val _def : THM;
  val head_def : THM;
  val last_def : THM;
  val front_def : THM;
  val tail_def : THM;
  val _def : THM;
  val enumerate_def : THM;
  val squash_def : THM;
  val extract_def : THM;
  val prefix_def : THM;
  val suffix_def : THM;
  val in_def : THM;
  val flat_def : THM;
```

**Description** This is the signature in which we declare theory “funrel”. 
SML

(* proof context key "seq_ext" *)

Description  seq_ext extends finset_ext by:
Name
  Becomes “finset_ext”.

Stripping Goals and Theorems
  Adding in ? applied at the top level or under a single negation.

Rewriting Context
  Adding in ?.

SML

(* proof context key "funrel_alg" *)

Description  finset_alg extends funrel_alg by:
Name
  Becomes “finset_alg”.

Stripping Goals and Theorems
  Adding in ? applied at the top level or under a single negation.

Rewriting Context
  Adding in ?.
9.2.16 Real Numbers

SML

(* Proof Context: 'R *)

Description A component proof context for handling the basic arithmetic operations for real numbers in HOL.

Expressions and predicates treated by this proof context are constructs formed from:

\[ +, *, -, Abs_R, /_R, \leq, \lt, \geq, \gt, = \]

Contents

Rewriting:

\[ R\_plus\_conv, R\_times\_conv, R\_subtract\_minus\_conv \]
\[ R\_abs\_conv, R\_over\_conv, R\_recip\_conv, R\_N\_exp\_conv, R\_Z\_exp\_conv, \]
\[ R\_eq\_conv, R\_\leq\_conv, R\_less\_conv \]
\[ R\_\geq\_conv, R\_greater\_conv, \]
\[ R\_plus\_clauses, R\_minus\_clauses, R\_\leq\_clauses \]
\[ R\_less\_clauses, R\_frac\_norm\_conv, float\_conv \]

Here float_conv is only applied to floating point literals with a zero exponent.

Stripping theorems:

\[ R\_eq\_conv, R\_\leq\_conv, R\_less\_conv \]
\[ R\_\geq\_conv, R\_greater\_conv, \]
\[ R\_plus\_clauses, R\_minus\_clauses, R\_\leq\_clauses \]
\[ R\_less\_clauses, \]
\[ and \ all \ the \ above \ pushed \ through \ \neg \]

Stripping conclusions: as for stripping theorems.

Rewriting canonicalisation: blank.

Automatic proof procedures: basic_prove_tac, basic_prove_conv.

Automatic existence prover: blank.
**SML**

```sml
(* Proof Context: R_lin_arith *)

**Description**  This is a proof context whose main purpose is to supply a decision procedure for problems in linear arithmetic over the HOL integers.

**Contents**  The rewriting, theorem stripping and conclusion stripping components are as for the proof context *predicates*, q.v, each augmented with conversions effecting the following transformations, where \( a \) and \( b \) stand for numeric literals.

\[
\begin{align*}
\forall a \equiv b & \rightarrow R_eq_conv a = b \\
\forall a \leq b & \rightarrow R_eq_conv a \leq b \\
\forall t1 = t2 & \rightarrow R_eq_cancel_conv t1 = t2 \\
\forall t1 \leq t2 & \rightarrow R_eq_cancel_conv t1 \leq t2 \\
\neg t1 = t2 & \rightarrow t1 < t2 \lor t2 < t1 \\
\neg t1 \leq t2 & \rightarrow t2 < t1 \\
\forall t1 \geq t2 & \rightarrow t2 \leq t1 \\
\forall t1 > t2 & \rightarrow t2 < t \\
\neg t1 < t2 & \rightarrow t2 \leq t1 \\
\neg t1 \geq t2 & \rightarrow \neg t2 \leq t1 \\
\neg t1 > t2 & \rightarrow \neg t1 \leq t2 \\
\forall t1 = t2 & \rightarrow t1 = t2 \\
\forall t1 \leq t2 & \rightarrow t1 \leq t2
\end{align*}
\]

(where all variables are of type \( R \)).

The automatic proof tactic works by (i) restripping all the assumptions of the goal, (ii) adding the argument theorems to the stock of assumptions using strip_asm_tac, (iii) applying contr_tac, and (iv) searching for a linear combination of the assumptions which will reduce, by multiplying out and cancelling like terms, to a contradiction of the form \( a = b \) or \( a \leq b \) with \( a \) and \( b \) numeric literals. The automatic proof conversion just tries to prove its argument, \( t \) say, using the automatic proof tactic and returns \( t = T \) if it succeeds.

Other components of the proof context are as for *predicates*.

For example, `PC_T1"R_lin_arith"prove_tac[]` will prove any of the following goals:

\[
\begin{align*}
([], \forall a \land c : R a \leq b \land (a + b \leq c + a) \Rightarrow a < c) \\
([], \forall a \land c : R a \geq b \land \neg b \leq c \Rightarrow (1/2) \cdot a \geq (1/2) \cdot c) \\
([], \forall a \land b : R a + 2 \cdot b < 2 \cdot a \Rightarrow b + b < a) \\
([], \forall x \land y : R \neg (2 \cdot x + y = 1/3 \land 4 \cdot x + 2 \cdot y = 2/5))
\end{align*}
\]

**See Also**  `R_lin_arith`

---

**SML**

```sml
(* Proof Context: 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 HOL integers.

**Contents**  The rewriting, theorem stripping and conclusion stripping components are as for the proof context `R_lin_arith` but without any of the material from the proof context *predicates*. The automatic proof components are as for `lin_arith`. Other components are blank.

A typical use of the proof context would be to solve problems containing a mixture of linear real number arithmetic and set theory.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
9.2. Theory Related ML Values

SML

val dest_R_less : TERM -> TERM * TERM; (* <R *)
val dest_R_le : TERM -> TERM * TERM (* ≤R *);
val dest_R_greater : TERM -> TERM * TERM (* >R *);
val dest_R_ge : TERM -> TERM * TERM (* ≥R *);
val dest_R_plus : TERM -> TERM * TERM (* +R *);
val dest_R_subtract : TERM -> TERM * TERM (* −R *);
val dest_R_times : TERM -> TERM * TERM (* ∗R *);
val dest_R_over : TERM -> TERM * TERM (* /R *);
val dest_R_frac : TERM -> TERM * TERM (* /N *);
val dest_R_abs : TERM -> TERM (* Abs_R *);
val dest_R_recip : TERM -> TERM (* −1 *);
val dest_R_N_exp : TERM -> TERM * TERM;(* ^N *)
val dest_R_Z_exp : TERM -> TERM * TERM;(* ^Z *)
val dest_NR : TERM -> TERM;(* NR *)
val dest_R_max : TERM -> TERM (* Max_R *);
val dest_R_min : TERM -> TERM (* Min_R *);

Description  These are the derived destructor functions for the basic arithmetic operators on the real numbers.

Errors

116101? 0 is not of the form⌜x <R y⌟
116102? 0 is not of the form⌜x ≤R y⌟
116103? 0 is not of the form⌜x >R y⌟
116104? 0 is not of the form⌜x ≤R y⌟
116105? 0 is not of the form⌜x +R y⌟
116106? 0 is not of the form⌜x −R y⌟
116107? 0 is not of the form⌜x *R y⌟
116108? 0 is not of the form⌜x /R y⌟
116109? 0 is not of the form⌜x /N y⌟
116110? 0 is not of the form⌜−R y⌟
116111? 0 is not of the form⌜Abs_R x⌟
116112? 0 is not of the form⌜x −1⌟
116113? 0 is not of the form⌜x ∗N m⌟
116114? 0 is not of the form⌜NR m⌟
116115? 0 is not of the form⌜Max_R L⌟
116116? 0 is not of the form⌜Z i⌟
116117? 0 is not of the form⌜Max_R L⌟
116118? 0 is not of the form⌜Min_R L⌟
**Description**  These conversions compute theorems of the form and

\[
\begin{align*}
\vdash \text{Max}_R x &= y \\
\vdash \text{Min}_R x &= y 
\end{align*}
\]

where \(x\) is either a singleton list display or a list display or list-valued expression formed with \(\text{Cons}\) whose first two elements are rational literals. E.g.,

\[
\vdash \text{Max}_R [\frac{4}{3}; 1.0; \frac{2}{3}] = \text{Max}_R [\frac{4}{3}; \frac{2}{3}]
\]

**Errors**

116331 Could not simplify ?0

---

| val float_conv : CONV; |
| --- | --- |
| **Description** | These are the derived discriminator functions for the basic arithmetic operators on the real numbers. The ML comments above give the name of the corresponding operators. These names all have aliases without the subscript. |

### SML

```sml
val is_R_less : TERM -> bool = (* <_R *);
val is_R_le : TERM -> bool = (* _le_R *);
val is_R_greater : TERM -> bool = (* _ge_R *);
val is_R_plus : TERM -> bool = (* _plus_R *);
val is_R_subtract : TERM -> bool = (* _minus_R *);
val is_R_times : TERM -> bool = (* _times_R *);
val is_R_over : TERM -> bool = (* _over_R *);
val is_R_frac : TERM -> bool = (* _frac_N *);
val is_R_min : TERM -> bool = (* _min_R *);
val is_R_max : TERM -> bool = (* _max_R *);
```
9.2. Theory Related ML Values

Description

These are the derived constructor functions for the basic arithmetic operators on the real numbers. The ML comments above give the name of the corresponding operators. These names all have aliases without the subscript.

Both operands of `mk_{R,frac}` and the second operand of `mk_{R,N,exp}` must be of type `N`. All the other operands of type `R`. All other operands must have type `R`.

Errors

116201 ?0 does not have type \([R]\)
116202 ?0 does not have type \([N]\)
116203 ?0 does not have type \([R\, \text{LIST}]\)

Description

These conversions compute theorems of the form:

\[\vdash \text{Abs } x = z\]
\[\vdash x - 1 = z\]

where `x` and `z` are rational literals (see `R,plus_conv`).

Errors

116307 cannot take the reciprocal of ?0 because its denominator or numerator is 0
116308 ?0 is not of the form \([R\, x^{-1}\)] where \([R\, x]\) is a rational literal
116316 ?0 is not of the form \([R\, \text{Abs } (i/m)\)]
  where \([R\, m]\) is a natural number literal
\textbf{Description} \texttt{R.anf.conv} is a conversion which proves theorems of the form $\vdash t_1 = t_2$ where $t_1$ is a term formed from atoms of type $\mathbb{R}$ and $t_2$ is in what we may call additive normal form, i.e. it has the form: $t_1 + t_2 + \ldots$, where the $t_i$ have the form $s_1 \ast s_2 \ast \ldots$ where the $s_i$ are atoms. Here, by atom we mean a term which is not of the form $t_1 + t_2 + \ldots$ or $s_1 \ast s_2 \ast \ldots$.

The summands $t_i$ and, within them, the factors $s_j$ are given in increasing order with respect to the ordering on terms given by the function $\texttt{R.term.order}$, q.v. Arithmetic computation is carried out on atoms to ensure that at most one of the summands is a rational literal and that, within each summand, at most one factor is a rational literal. Any literal appears at the beginning of its factor or summand and addition of 0 or multiplication by 1 is simplified out. Floating point literals are converted into rational literals.

$\texttt{R.ANF.C.conv}$ is a conversion which acts like $\texttt{R.anf.conv}$ but which applies $\texttt{conv}$ 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 116318 if there are no changes to be made to the term.

\textbf{Errors} 116318?0 is not of type $\langle:\mathbb{R}\rangle$ or is already in additive normal form.

\textbf{Description} An order-preserving additive homomorphism from $\mathbb{R}$ to $\mathbb{R}$ is said to be “central” if it commutes with any homomorphism from $\mathbb{R}$ to $\mathbb{R}$. These are the ML bindings for theorems giving various properties of central order-preserving additive from $\mathbb{R}$ to $\mathbb{R}$.

\textbf{Description} This is the theorem that supports $\texttt{R.delta_induction_tac}$ (q.v.).
Description  

This is an induction-like tactic that can be used to prove that a property holds in a half-open interval of the real numbers: to prove that $t[y]$ holds whenever $x < y$, it suffices to exhibit a positive number $d$ such that (i) $t[u/y]$ holds for each $u$ with $x < u < x + e$ where $d < e$ and (ii) $t[s + d/y]$ holds if $t[s/y]$ holds, for any $s$ with $x < s$.

The term argument $y$ must be a variable that appears free in the conclusion of the goal. The variable must also appear free once, and only once, in the assumptions, in an assumption of the form $x < y$.

Tactic

\[
\{ \Gamma; x < y \} \quad t[y/y] \\
\{ \Gamma \} \quad \exists d \cdot \theta, < d \\
\wedge (\exists e \cdot d < e \wedge \\
(\forall u \cdot x < u \Rightarrow u < x + e \Rightarrow \\
t[u/y])) \\
\wedge (\forall s \cdot x < s \Rightarrow p s \Rightarrow \\
t[s + d/y]))
\]

Errors

116001?0 is not a variable of type $\R$.
116002 A term of the form $\exists x < y$ where $y$ is the induction variable could not be found in the assumptions.
116003?0 does not appear free in the conclusions of the goal.
116004?0 appears free in more than one assumption of the goal.
val \texttt{R\_eq\_cancel\_conv} : \texttt{CONV}  \\
val \texttt{R\_\leq\_cancel\_conv} : \texttt{CONV}  \\
val \texttt{R\_less\_cancel\_conv} : \texttt{CONV}

**Description** \texttt{R\_eq\_cancel\_conv} (resp. \texttt{R\_\leq\_cancel\_conv}, resp. \texttt{R\_less\_cancel\_conv}) puts arithmetic equations (resp. inequalities formed with $\leq$) in a normal form in which the right-hand side is a signed literal and the left-hand side is in additive normal form, in the sense of \texttt{R\_anf\_conv}, q.v.

For example, the calls:

- \texttt{ProofPower Input}  \\
  \texttt{R\_eq\_cancel\_conv}$\{x+2.*y+3.=1.+6.*y^2\};  \\
  \texttt{R\_\leq\_cancel\_conv}$\{x+2.*y+x+.3\leq y+2.+.2*x+y^2\};  \\
  \texttt{R\_less\_cancel\_conv}$\{x+2.*y+x+.3<y+2.+.2*x+y^3\};

produce the following output

- \texttt{ProofPower Output}  \\
  \texttt{val it} = $\vdash x+2.*y+3.=1.+6.*y$
  \iff $x+\sim (4.)*y=\sim (2.)$ : \texttt{THM}  \\
  \texttt{val it} = $\vdash x+2.*y+x+.3\leq y+2.+.2*x+y$
  \iff $1.\leq 0.$ : \texttt{THM}  \\
  \texttt{val it} = $\vdash x+2.*y+x+.3<y+2.+.2*x+y$
  \iff $1.< 0.$ : \texttt{THM}

Note that if the left-hand side reduces to 0 the truth value is not evaluated. However, \texttt{R\_eq\_conv}, \texttt{R\_\leq\_conv} or \texttt{R\_less\_conv} may be used to perform the evaluation, where required.

**Uses** The conversions are intended for use in tactic and conversion programming. The normal interactive interface is via rewriting or stripping in the proof context \texttt{R\_lin\_arith}, which performs other useful simplifications.

**Errors**  \\
116520?0 is not of the form \texttt{(t1:R) = t2} or is already in normal form  \\
116521?0 is not of the form \texttt{(t1:R) \leq t2} or is already in normal form  \\
116522?0 is not of the form \texttt{(t1:R) < t2} or is already in normal form

**See Also** \texttt{R\_lin\_arith (proof context)}

---

val \texttt{R\_eval\_conv} : \texttt{CONV};  
val \texttt{R\_EVAL\_C} : \texttt{CONV \rightarrow CONV};

**Description** \texttt{R\_eval\_conv} computes theorems of the form $\vdash t1 = t2$ where \( t1 \) is an expression made up from rational literals (see \texttt{R\_plus\_conv}) and/or floating point literals 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).

\texttt{R\_EVAL\_C conv} is similar to \texttt{R\_eval\_conv} but it also applies \texttt{conv} to any subterm that cannot be evaluated using the conversions for the arithmetic operations listed above. E.g., \texttt{R\_EVAL\_C R\_N\_exp\_conv} will evaluate expressions involving the usual arithmetic operations and also exponentiation of rational literals by natural number literals.

**Errors**  \\
116320?0 cannot be evaluated
9.2. Theory Related ML Values

SML

val \texttt{R\_frac\_cross\_mult\_eq\_thm} : THM;
val \texttt{R\_frac\_cancel\_thm} : THM;
val \texttt{R\_frac\_0\_thm} : THM;
val \texttt{R\_frac\_N\_thm} : THM;
val \texttt{R\_frac\_plus\_frac\_thm} : THM;
val \texttt{R\_frac\_minus\_frac\_thm} : THM;
val \texttt{R\_over\_times\_over\_thm} : THM;
val \texttt{R\_frac\_times\_frac\_thm} : THM;
val \texttt{R\_over\_recip\_thm} : THM;
val \texttt{R\_frac\_recip\_thm} : THM;
val \texttt{R\_frac\_minus\_recip\_thm} : THM;
val \texttt{R\_frac\_over\_eq\_0\_thm} : THM;
val \texttt{R\_frac\_over\_over\_over\_thm} : THM;
val \texttt{R\_frac\_less\_frac\_thm} : THM;
val \texttt{R\_minus\_frac\_less\_frac\_thm} : THM;
val \texttt{R\_frac\_less\_minus\_frac\_thm} : THM;
val \texttt{R\_0\_\le\_frac\_thm} : THM;
val \texttt{R\_abs\_frac\_thm} : THM;
val \texttt{R\_abs\_minus\_thm} : THM;

Description These are the ML bindings for some theorems about rational numbers (in \(\mathbb{R}\)) expressed as fractions with natural number numerators and denominators.

The very last theorem is a general fact about absolute values.

SML

val \texttt{R\_frac\_norm\_conv} : CONV;  val \texttt{R\_frac\_plus\_frac\_conv} : CONV;

Description \(\texttt{R\_frac\_norm\_conv}\) proves theorems giving a normal form for real literals expressed as numeric fractions. The theorems have the form:

\[
\begin{align*}
\vdash m / N n &= NR m' \\
\vdash m / N n &= m' / N n'
\end{align*}
\]

where \(m, n, m'\) and \(n'\) are natural number literals and, in the second case, \(m'\) and \(n'\) are coprime and \(n' > 1\).

\(\texttt{R\_frac\_plus\_frac\_conv}\) proves theorems of the forms:

\[
\begin{align*}
\vdash m / N n + m' / N n' &= t
\end{align*}
\]

where \(m, n, m'\) and \(n'\) are natural number literals and \(t\) is in the same kind of normal form as produced by \(\texttt{R\_frac\_norm\_conv}\).

Errors

116301: ?0 is not of the form \(\frac{m}{N} n\) or \(NR m\)
116302: cannot simplify ?0 because it denominator is 0.
116303: ?0 is already in normal form
116304: ?0 is not of the form \(\frac{i}{N} m + R (j / N n)\)
These conversions prove theorems of the following forms:

\[ \vdash x > y \iff y < x \]
\[ \vdash x \geq y \iff y \leq x \]
\[ \vdash x - y \iff y + \sim x \]

Errors:

116313?0 is not of the form \( \lceil x > y \rceil \) where \( \lceil x \rceil \) and \( \lceil y \rceil \) have type \( \lceil \mathbb{R} \rceil \)
116314?0 is not of the form \( \lceil x \geq y \rceil \) where \( \lceil x \rceil \) and \( \lceil y \rceil \) have type \( \lceil \mathbb{R} \rceil \)
116315?0 is not of the form \( \lceil x - y \rceil \) where \( \lceil x \rceil \) and \( \lceil y \rceil \) have type \( \lceil \mathbb{R} \rceil \)

Given a system, \( \Gamma = [r_1, r_2, ...] \), of numeric constraints, \( r_i \), of the form \( (s_i: \mathbb{R}) = c_i \) or \( s_i \leq c_i \), where the \( c_i \) are rational literals, the rule attempts to prove a theorem of the form \( \Gamma \vdash F \). Terms in \( \Gamma \) which are not of either of these forms are ignored and do not appear in the assumptions of the result theorem.

The tactics use the rule in much the same way as the integer linear arithmetics use \( \mathbb{Z} \text{ lin arith rule} \) (see \( \mathbb{Z} \text{ lin arith tac} \)). The usual interface to these functions is via the decision procedures in the proof context \( \mathbb{R} \text{ lin arith} \), q.v., e.g. as invoked by \( PC_T1"\mathbb{R} \text{ lin arith"prove_tac}[] \)

The algorithm for the decision procedure is very similar to the one used in \( \text{lin arith rule} \), q.v.

Errors:

116341 The linear arithmetic proof procedure cannot prove this conjecture

This conversion implements the Hodes-Fourier-Motzkin quantifier elimination procedure for linear arithmetic with real coefficients.

These are the ML bindings for theorems in the theory \( \mathbb{R} \) about maxima and minima of lists.

These conversions compute theorems of the form:
9.2. Theory Related ML Values

SML

val R_minus_conv : CONV;

Description This conversion simplifies expressions involving real negation. It returns theorems of the form:

\[ \vdash \neg(\neg x) = x \]
\[ \vdash \neg(0/1) = 0. \]
\[ \vdash \neg 0 \cdot = 0. \]

Errors

116319?0 is not of the form \( \neg(\neg x) \), \( \neg(0/1) \), or \( \neg 0 \).

SML

val R_monoid_delta_dense_thm : THM;
val R_monoid_dense_thm : THM;
val R_subgroup_dense_thm : THM;
val R_semigroup_dense_thm : THM;
val R_add_hom_image_group_thm : THM;
val R_add_hom_kernel_group_thm : THM;

Description These are the ML bindings for theorems giving various properties of substructures of the additive group \( \mathbb{R} \) and of (additive) homomorphisms from \( \mathbb{R} \) to \( \mathbb{R} \).

SML

val R_opah_thm : THM; val R_opah_strict_thm : THM;
val R_opah_one_one_thm : THM; val R_opah_dense_image_thm : THM;
val R_opah_inverse_thm : THM; val R_opah_inverse_add_hom_thm : THM;
val R_opah_extension_thm1 : THM; val R_opah_extension_thm : THM;
val R_opah_extension_thm2 : THM; val R_opah_extension1_thm : THM;
val R_opah_order_thm : THM; val R_opah_eq_thm : THM;
val R_opah_complete_thm : THM;

Description These are the ML bindings for theorems giving various properties of order-preserving additive homomorphisms from \( \mathbb{R} \) to \( \mathbb{R} \).

SML

val R_ord_pres_strict_thm : THM;
val R_add_hom_0_thm : THM;
val R_add_hom_minus_thm : THM;
val R_add_hom_extension_thm : THM;

Description These are ML bindings for theorems about order-preserving functions and additive endomorphisms from \( \mathbb{R} \) to \( \mathbb{R} \).

SML

val R_over_times_recip_thm : THM; val R_times_recip_thm : THM;
val R_eq_recip_thm : THM; val R_recip_clauses : THM;
val R_0_less_0_less_recip_thm : THM; val R_over_cancel_thm : THM;
val R_over_plus_over_thm : THM; val R_0_over_thm : THM;
val R_over_1_thm : THM; val R_\neg_recip_0_thm : THM;

Description These are the ML bindings for some basic theorems about reciprocals (multiplicative inverses) in the real numbers and their relationship with the other structure.
These are the ML bindings for basic theorems about the additive structure of the real numbers and its relationship with the ordering relations and the supremum function.

Description

Also, the following conversions compute theorems of the form $\vdash x \mathrel{op} y = z$

where $x$ and $y$ are rational literals and $\mathrel{op}$ is the operation indicated by the name of the conversion. A rational literal is taken to be an optionally signed expression made up from natural number literals using $\mathbb{N}$ or $\mathbb{R}/$. (Note that $\mathbb{N}$ applied to a natural number literal is pretty-printed and may be entered as the natural number followed by a decimal point. The term $z$ is a truth value or an optionally signed rational literal in normal form (see $\mathbb{R}$-frac_norm_conv as appropriate.

Errors

- 116305\?0 is not of the form $\lfloor x + R \rfloor y$ where $\lfloor x \rfloor$ and $\lfloor y \rfloor$ are rational literals
- 116306\?0 is not of the form $\lfloor x \times R \rfloor y$ where $\lfloor x \rfloor$ and $\lfloor y \rfloor$ are rational literals
- 116309\?0 is not of the form $\lfloor x < y \rfloor$ where $\lfloor x \rfloor$ and $\lfloor y \rfloor$ are rational literals
- 116310\?0 is not of the form $\lfloor x \leq y \rfloor$ where $\lfloor x \rfloor$ and $\lfloor y \rfloor$ are rational literals
- 116311\?0 is not of the form $\lfloor x = y \rfloor$ where $\lfloor x \rfloor$ and $\lfloor y \rfloor$ are rational literals
- 116312\?0 is not of the form $\lfloor x / y \rfloor$ where $\lfloor y \rfloor$ is a rational literal
- 116317\?0 is not of the form $\lfloor x \rightarrow R m \rfloor$ where $\lfloor x \rfloor$ is a rational number literal and $\lfloor m \rfloor$ is a natural number literal.
Description These are the ML bindings for the definitions in the theory $\mathbb{R}$.

<table>
<thead>
<tr>
<th>SML</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>val $\mathbb{R}$ _plus_def : THM;</td>
<td>val $\mathbb{R}$ _subtract_def : THM;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>val $\mathbb{R}$ _less_def : THM;</td>
<td>val $\mathbb{R}$ _less_def : THM;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>val $\mathbb{R}$ _greater_def : THM;</td>
<td>val $\mathbb{R}$ _greater_def : THM;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>val $\mathbb{R}$ _minus_def : THM;</td>
<td>val $\mathbb{R}$ _times_def : THM;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>val $\mathbb{R}$ _over_def : THM;</td>
<td>val $\mathbb{R}$ _frac_def : THM;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>val NR_def : THM;</td>
<td>val N_exp_def : THM;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>val $\mathbb{R}$ _exp_def : THM;</td>
<td>val ZR_def : THM;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>val $\mathbb{R}$ _abs_def : THM;</td>
<td>val float_def : THM;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>val $\mathbb{R}$ _max_def : THM;</td>
<td>val $\mathbb{R}$ _min_def : THM;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description These are the ML bindings for the theorems in the theory $\mathbb{R}$ to assist with reasoning about the values of the supremum function.

<table>
<thead>
<tr>
<th>SML</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>val $\mathbb{R}$ _sup_thm : THM;</td>
<td>val $\mathbb{R}$ _less_sup_thm : THM;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>val $\mathbb{R}$ _less_sup_bc_thm : THM;</td>
<td>val $\mathbb{R}$ _less_sup_thm : THM;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>val $\mathbb{R}$ _sup_bc_thm : THM;</td>
<td>val $\mathbb{R}$ _less_sup_bc_thm : THM;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>val $\mathbb{R}$ _lesssup_bc_thm : THM;</td>
<td>val $\mathbb{R}$ _sup_le_bc_thm : THM;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>val $\mathbb{R}$ _supless_bc_thm : THM;</td>
<td>val $\mathbb{R}$ _sup_eq_bc_thm : THM;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>val $\mathbb{R}$ _supless_eq_bc_thm : THM;</td>
<td>val $\mathbb{R}$ _lesssup_eq_bc_thm : THM;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description These are the ML bindings for the term order combinators used as arguments to make_term_order to implement $\mathbb{R}$ _term_order and $\mathbb{R}$ _term_order1, q.v.

| SML | | |
| --- | --- | |
| val $\mathbb{R}$ _term_order_comb : TERM ORDER $\rightarrow$ TERM ORDER; |
| val $\mathbb{R}$ _term_order_comb1 : | |
| {graded : bool, inverse : bool} $\rightarrow$ | |
| TERM ORDER $\rightarrow$ TERM ORDER; | |

Description These are the term order combinators used as arguments to make_term_order to implement $\mathbb{R}$ _term_order and $\mathbb{R}$ _term_order1, q.v.
Description \( \mathbb{R}_{\text{term order}} \) gives an ordering relation on HOL terms analogous to that given by \( \text{term order} \), q.v., but which takes special arrangements for certain terms of type \( \mathbb{R} \). Terms of the form \( t_1 \ast \ldots \ast t_l \) where each \( t_i \) are called monomials and are ordered lexicographically using an ordering on the \( t_i \) which takes terms formed by exponentiation by natural number literals in the following order: \( x^0 < x < x^1 < x^2 < \ldots \).

Rational literals, i.e., terms formed using \( \mathbb{N}_{\mathbb{R}} \) or natural number division with natural number literal operands are ordered numerically.

Each monomial is grouped next to the the same monomial with coefficients and signs or reciprocals (where a coefficient is a rational literal appearing in a monomial as the term \( t_1 \)). Thus:

\[
\begin{align*}
  a * b &< \sim a * b < 2, \quad a * b < \sim (2/1)^{-1} * a * b < \ldots \\
  \sim a * b &< 2, \quad a * b < \sim (2/1)^{-1} * a * b < \ldots \\
\end{align*}
\]

Note that \( \mathbb{R}_{\text{term order}} \) takes the lexicographic order in the sense appropriate to a sparse representation of monomials. This means that \( xy \) precedes \( xz \) (whereas in a dense representation, \( xz = x^1 y^0 z^1 \) precedes \( xy = x^1 y^1 z^0 \)). This is intuitive, but does not give an admissible order in the sense of Gröebner basis theory, since, for example, \( xy \) is collated before \( xz \), but \( x^2 y = (xy)y \) is collated after \( xyz = (xz)y \).

\( \mathbb{R}_{\text{term order1}} \) provides the usual admissible lexicographic orderings used in Gröebner basis theory (following section 5.1 of T. Becker and V. Weispfenning, \textit{Gröebner Bases}, Springer, 1993). For example, \( \mathbb{R}_{\text{term order1}} \{ \text{graded} = \text{true}, \text{inverse} = \text{true} \} \) gives the graded inverse lexicographic order.

See Also \( \mathbb{R}_{\text{term order comb}}, \text{graded } \mathbb{R}_{\text{term order comb}} \)
9.2. Theory Related ML Values

```sml
val R_unbounded_below_thm : THM;
val R_unbounded_above_thm : THM;
val R_less_irrefl_thm : THM;  val R_less_antisym_thm : THM;
val R_less_trans_thm : THM;  val R_less_cases_thm : THM;
val R_eq_less_thm : THM;    val R_less_less_cases_thm : THM;
val R_less_cases_thm : THM;  val R_less_trans_thm : THM;
val R_less_refl_thm : THM;   val R_less_trans_thm : THM;
val R_eq_less_thm : THM;    val R_less_antisym_thm : THM;
val R_less_trans_thm : THM;  val R_less_trans_thm : THM;
val R_less_eq_thm : THM;    val R_less_trans_thm : THM;
val R_less_dense_thm : THM;  val R_complete_thm : THM;
```

**Description** These are the ML bindings for theorems in the theory \( \mathbb{R} \) concerned with basic properties of the ordering relations. Note that \( > \) and \( \geq \) are defined in terms of \( < \) and \( \leq \) and the latter are the preferred form in all theorems in this theory.

```sml
val Z_exp_conv : CONV;
```

**Description** This conversions computes theorems of the form and

\[
\vdash x \sim_Z NZ \ m = x \sim_N \ m \\
\vdash x \sim_Z (NZ \ m) = (x \sim_N \ m)^{-1}
\]

**Errors**

116321??0 is not of the form \( \vdash x \sim_Z m \)

```sml
val ZR_plus_homomorphism_thm : THM;
val ZR_minus_thm : THM;
val ZR_NZ_thm : THM;
val ZR_times_homomorphism_thm : THM;
```

**Description** These are the ML bindings for theorems in the theory \( \mathbb{R} \) to assist with reasoning about the embedding of the integers in the reals.
REFERENCES


INDEX

'b_basic_prove_3_conv ........................................ 364 + ........................................ 445
'b_bin_rel ..................................................... 493 + ........................................ 446
'b_char ......................................................... 371 + ........................................ 450
'b_combin ....................................................... 374 ........................................ 416
'fun_ext ......................................................... 367 − \in R ........................................ 427
'fun_rel_ext .................................................... 493 − \in R ........................................ 428
'lin_arith ....................................................... 467 − \in R ........................................ 429
'list .......................................................... 371 − \in Z ........................................ 450
'mpp1 ........................................................... 372 − \in Z ........................................ 451
'mpp2 ........................................................... 372 − ............................................ 427
'one .............................................................. 374 − ............................................ 445
'pair ............................................................. 369 − ............................................ 446
'paired_abstractions ........................................... 365 − ............................................ 450
'pair ............................................................. 368 ............................................ 418
'propositions ..................................................... 366 ............................................ 419
'prop_eq_pair .................................................... 313 /N ............................................ 427
'prop_eq ........................................................ 313 /N ............................................ 428
'sets_alg ........................................................ 375 /N ............................................ 429
'sets_ext1 ....................................................... 377 /R−consistent ..................................... 440
'sets_ext ......................................................... 376 /R ............................................ 427
'sho rw .......................................................... 372 /R ............................................ 428
'simple_abstractions .......................................... 365 /R ............................................ 429
'sum ............................................................. 373 /R ............................................ 427
'\N_lit ........................................................... 370 /R ............................................ 428
'\N ............................................................... 369 0\in R−consistent ..................................... 432
'R ................................................................. 498 0\in R ............................................ 427
'R−lin_arith ...................................................... 497 0\in R ............................................ 428
'Z ................................................................. 482 1\in R−consistent ..................................... 432
'Z−lin_arith ...................................................... 482 1\in R ............................................ 427
'Z ................................................................. 474 1\in R ............................................ 428
** ............................................................... 28 <<< ............................................ 411
*\in R−consistent ............................................... 439 <<< ............................................ 411
*\in R ........................................................... 427 <<< ............................................ 411
*\in R−consistent ............................................... 428 <<< \in R−consistent ..................................... 430
*\in R ........................................................... 429 <<< \in R ............................................ 427
*\in Z−consistent ............................................... 453 <<< \in R ............................................ 428
*\in Z ........................................................... 450 <<< \in Z ............................................ 450
*\in Z ........................................................... 451 <<< \in Z ............................................ 451
* ................................................................. 427 <<< ............................................ 427
* ................................................................. 445 <<< ............................................ 445
* ................................................................. 446 <<< ............................................ 446
* ................................................................. 450 <<< ............................................ 450
+\in R−consistent ................................................ 432 =: ............................................ 83
+\in R .............................................................. 427 = # ............................................ 134
+\in R .............................................................. 428 = $ ............................................ 82
+\in Z−consistent ................................................ 452 = − ............................................ 134
+\in Z .............................................................. 450 = ............................................ 406
+\in Z .............................................................. 451 >\in R ............................................ 427
+ ................................................................. 425 >\in R ............................................ 428
+ ................................................................. 427 >\in Z ............................................ 450

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>asm_inst_type_rule</td>
<td>179</td>
</tr>
<tr>
<td>asm_intro</td>
<td>179</td>
</tr>
<tr>
<td>ASM_PROP_EQ_T</td>
<td>314</td>
</tr>
<tr>
<td>asm_prove_tac</td>
<td>339</td>
</tr>
<tr>
<td>asm_prove_I_tac</td>
<td>339</td>
</tr>
<tr>
<td>asm_rewrite_rule</td>
<td>204</td>
</tr>
<tr>
<td>asm_rewrite_tac</td>
<td>283</td>
</tr>
<tr>
<td>asm_rewrite_thm_tac</td>
<td>283</td>
</tr>
<tr>
<td>asm_rule</td>
<td>180</td>
</tr>
<tr>
<td>ASSOC</td>
<td>71</td>
</tr>
<tr>
<td>ASYM_C</td>
<td>317</td>
</tr>
<tr>
<td>at_at_eq_thm</td>
<td>493</td>
</tr>
<tr>
<td>at_at_eq_thm</td>
<td>493</td>
</tr>
<tr>
<td>At consistent</td>
<td>397</td>
</tr>
<tr>
<td>at_def</td>
<td>396</td>
</tr>
<tr>
<td>At</td>
<td>397</td>
</tr>
<tr>
<td>Back-chain-tac</td>
<td>243</td>
</tr>
<tr>
<td>back_chain_tac</td>
<td>255</td>
</tr>
<tr>
<td>back_chain_thm_tac</td>
<td>256</td>
</tr>
<tr>
<td>bad_proof</td>
<td>256</td>
</tr>
<tr>
<td>Basic           error</td>
<td>16</td>
</tr>
<tr>
<td>BasicHolTheory</td>
<td>489</td>
</tr>
<tr>
<td>BasicIO</td>
<td>43</td>
</tr>
<tr>
<td>basic_hol</td>
<td>372</td>
</tr>
<tr>
<td>basic_prove_conv</td>
<td>379</td>
</tr>
<tr>
<td>basic_prove_tac</td>
<td>380</td>
</tr>
<tr>
<td>basic_prove_I_conv</td>
<td>160</td>
</tr>
<tr>
<td>basic_prove_I_tac</td>
<td>161</td>
</tr>
<tr>
<td>basic_prove_tac</td>
<td>161</td>
</tr>
<tr>
<td>BASIC_RESOLUTION_T1</td>
<td>327</td>
</tr>
<tr>
<td>BASIC_RESOLUTION_T</td>
<td>326</td>
</tr>
<tr>
<td>basic_resolve_rule</td>
<td>327</td>
</tr>
<tr>
<td>basic_res_extract</td>
<td>327</td>
</tr>
<tr>
<td>basic_res_next_to_process</td>
<td>328</td>
</tr>
<tr>
<td>basic_res_post</td>
<td>328</td>
</tr>
<tr>
<td>basic_res_pre</td>
<td>328</td>
</tr>
<tr>
<td>basic_res Resolver</td>
<td>328</td>
</tr>
<tr>
<td>basic_res_rule</td>
<td>329</td>
</tr>
<tr>
<td>basic_res_subsumption</td>
<td>329</td>
</tr>
<tr>
<td>basic_res_tac1</td>
<td>330</td>
</tr>
<tr>
<td>basic_res_tac2</td>
<td>330</td>
</tr>
<tr>
<td>basic_res_tac3</td>
<td>331</td>
</tr>
<tr>
<td>basic_res_tac4</td>
<td>331</td>
</tr>
<tr>
<td>basic_res_tac</td>
<td>332</td>
</tr>
<tr>
<td>BASIC_RES_TYPE</td>
<td>325</td>
</tr>
<tr>
<td>bc_tac</td>
<td>255</td>
</tr>
<tr>
<td>bc_thm_tac</td>
<td>256</td>
</tr>
<tr>
<td>before_kernel_state_change</td>
<td>135</td>
</tr>
<tr>
<td>Binder</td>
<td>180</td>
</tr>
<tr>
<td>BinRelTheory</td>
<td>71</td>
</tr>
<tr>
<td>bin bool ap</td>
<td>492</td>
</tr>
<tr>
<td>bin rel cut clauses</td>
<td>83</td>
</tr>
<tr>
<td>bin rel cut clauses</td>
<td>385</td>
</tr>
<tr>
<td>bin rel cut clauses</td>
<td>492</td>
</tr>
<tr>
<td>bin rel insert thm</td>
<td>386</td>
</tr>
<tr>
<td>bin_rel_insert_thm</td>
<td>492</td>
</tr>
<tr>
<td>bin_rel_O_universe_thm</td>
<td>386</td>
</tr>
<tr>
<td>boolean_cases_axiom</td>
<td>400</td>
</tr>
<tr>
<td>boolean_cases_axiom</td>
<td>458</td>
</tr>
<tr>
<td>Cases</td>
<td>257</td>
</tr>
<tr>
<td>CASES_T2</td>
<td>257</td>
</tr>
<tr>
<td>CASES_T</td>
<td>257</td>
</tr>
<tr>
<td>Chains</td>
<td>398</td>
</tr>
<tr>
<td>CHANGED_C</td>
<td>180</td>
</tr>
<tr>
<td>CHANGED_T</td>
<td>257</td>
</tr>
<tr>
<td>Character Utilities</td>
<td>31</td>
</tr>
<tr>
<td>char_conv</td>
<td>180</td>
</tr>
<tr>
<td>char_def</td>
<td>388</td>
</tr>
<tr>
<td>char_eq_conv</td>
<td>463</td>
</tr>
<tr>
<td>choose_consistent</td>
<td>424</td>
</tr>
<tr>
<td>Choose</td>
<td>424</td>
</tr>
<tr>
<td>clear_compactification_cache</td>
<td>132</td>
</tr>
<tr>
<td>close_in</td>
<td>43</td>
</tr>
<tr>
<td>close_out</td>
<td>43</td>
</tr>
<tr>
<td>conf_con</td>
<td>318</td>
</tr>
<tr>
<td>Combinators</td>
<td>30</td>
</tr>
<tr>
<td>Combinators</td>
<td>30</td>
</tr>
<tr>
<td>Combinators</td>
<td>30</td>
</tr>
<tr>
<td>Combinators</td>
<td>30</td>
</tr>
<tr>
<td>Combine</td>
<td>401</td>
</tr>
<tr>
<td>Combine</td>
<td>401</td>
</tr>
<tr>
<td>CombinTheory</td>
<td>490</td>
</tr>
<tr>
<td>CombI</td>
<td>389</td>
</tr>
<tr>
<td>CombK</td>
<td>389</td>
</tr>
<tr>
<td>CombS</td>
<td>389</td>
</tr>
<tr>
<td>comb_i_def</td>
<td>389</td>
</tr>
<tr>
<td>comb_k_def</td>
<td>389</td>
</tr>
<tr>
<td>comb_s_def</td>
<td>389</td>
</tr>
<tr>
<td>compact_term</td>
<td>132</td>
</tr>
<tr>
<td>compactification_mask</td>
<td>339</td>
</tr>
<tr>
<td>compact_term</td>
<td>339</td>
</tr>
<tr>
<td>compact_thm</td>
<td>339</td>
</tr>
<tr>
<td>compact_type</td>
<td>339</td>
</tr>
<tr>
<td>comparability_thm</td>
<td>398</td>
</tr>
<tr>
<td>INDEX</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>CompConv</td>
<td>462</td>
</tr>
<tr>
<td>complement_clauses</td>
<td>421</td>
</tr>
<tr>
<td>complement_clauses</td>
<td>486</td>
</tr>
<tr>
<td>complement_def</td>
<td>421</td>
</tr>
<tr>
<td>complement_def</td>
<td>486</td>
</tr>
<tr>
<td>Complete</td>
<td>411</td>
</tr>
<tr>
<td>Complete</td>
<td>412</td>
</tr>
<tr>
<td>concl_in_asms_tac</td>
<td>258</td>
</tr>
<tr>
<td>concl</td>
<td>135</td>
</tr>
<tr>
<td>COND_C</td>
<td>181</td>
</tr>
<tr>
<td>cond_def</td>
<td>407</td>
</tr>
<tr>
<td>cond_def</td>
<td>458</td>
</tr>
<tr>
<td>cond_thm</td>
<td>181</td>
</tr>
<tr>
<td>cond_thm</td>
<td>408</td>
</tr>
<tr>
<td>COND_T</td>
<td>258</td>
</tr>
<tr>
<td>Cond</td>
<td>407</td>
</tr>
<tr>
<td>consistent_def</td>
<td>155</td>
</tr>
<tr>
<td>ConstantSpecification</td>
<td>155</td>
</tr>
<tr>
<td>CONSTRAINT_TYPE</td>
<td>468</td>
</tr>
<tr>
<td>CONSTRAINT</td>
<td>468</td>
</tr>
<tr>
<td>const_spec_def</td>
<td>155</td>
</tr>
<tr>
<td>const_spec_thm</td>
<td>155</td>
</tr>
<tr>
<td>const_spec</td>
<td>156</td>
</tr>
<tr>
<td>Const</td>
<td>81</td>
</tr>
<tr>
<td>Cons</td>
<td>401</td>
</tr>
<tr>
<td>Cons</td>
<td>402</td>
</tr>
<tr>
<td>contains</td>
<td>20</td>
</tr>
<tr>
<td>contr_rule</td>
<td>181</td>
</tr>
<tr>
<td>contr_tac</td>
<td>259</td>
</tr>
<tr>
<td>CONTR_T</td>
<td>259</td>
</tr>
<tr>
<td>conv_ascii</td>
<td>12</td>
</tr>
<tr>
<td>conv_extended</td>
<td>12</td>
</tr>
<tr>
<td>conv_rule</td>
<td>181</td>
</tr>
<tr>
<td>conv_tac</td>
<td>259</td>
</tr>
<tr>
<td>COV_THEN</td>
<td>260</td>
</tr>
<tr>
<td>CONV</td>
<td>121</td>
</tr>
<tr>
<td>counts</td>
<td>55</td>
</tr>
<tr>
<td>cov_induction_tac</td>
<td>459</td>
</tr>
<tr>
<td>cov_induction_thm</td>
<td>448</td>
</tr>
<tr>
<td>cov_induction_thm</td>
<td>462</td>
</tr>
<tr>
<td>COV_INDUCTION_T</td>
<td>460</td>
</tr>
<tr>
<td>cthm_eqn_cxt</td>
<td>181</td>
</tr>
<tr>
<td>Ctype</td>
<td>81</td>
</tr>
<tr>
<td>cup</td>
<td>20</td>
</tr>
<tr>
<td>current_ad_cs_∃_conv</td>
<td>353</td>
</tr>
<tr>
<td>current_ad_mmp_rule</td>
<td>340</td>
</tr>
<tr>
<td>current_ad_nd_net</td>
<td>353</td>
</tr>
<tr>
<td>current_ad_pr_cncv</td>
<td>340</td>
</tr>
<tr>
<td>current_ad_pr_tac</td>
<td>340</td>
</tr>
<tr>
<td>current_ad_rw_canon</td>
<td>352</td>
</tr>
<tr>
<td>current_ad_rw_eqn_rule</td>
<td>340</td>
</tr>
<tr>
<td>current_ad_rw_net</td>
<td>351</td>
</tr>
<tr>
<td>current_ad_sc_cnv</td>
<td>352</td>
</tr>
<tr>
<td>current_ad_st_conv</td>
<td>352</td>
</tr>
<tr>
<td>current_ad_∃_cd_thms</td>
<td>353</td>
</tr>
<tr>
<td>current_ad_∃_es_thms</td>
<td>354</td>
</tr>
<tr>
<td>curry_def</td>
<td>417</td>
</tr>
<tr>
<td>curry_def</td>
<td>459</td>
</tr>
<tr>
<td>curry</td>
<td>28</td>
</tr>
<tr>
<td>Curry</td>
<td>416</td>
</tr>
<tr>
<td>Curry</td>
<td>417</td>
</tr>
<tr>
<td>cuts_complete_thm</td>
<td>413</td>
</tr>
<tr>
<td>cuts_strict_linear_order_thm</td>
<td>413</td>
</tr>
<tr>
<td>cuts_strict_partial_order_thm</td>
<td>413</td>
</tr>
<tr>
<td>cuts_trich_thm</td>
<td>413</td>
</tr>
<tr>
<td>cuts_unbounded_above_thm</td>
<td>414</td>
</tr>
<tr>
<td>cuts_unbounded_below_thm</td>
<td>414</td>
</tr>
<tr>
<td>DAapp</td>
<td>82</td>
</tr>
<tr>
<td>DChar</td>
<td>82</td>
</tr>
<tr>
<td>DConst</td>
<td>82</td>
</tr>
<tr>
<td>declare_alias</td>
<td>124</td>
</tr>
<tr>
<td>declare_binder</td>
<td>124</td>
</tr>
<tr>
<td>declare_const_language</td>
<td>124</td>
</tr>
<tr>
<td>declare_infix</td>
<td>125</td>
</tr>
<tr>
<td>declare_left_infix</td>
<td>125</td>
</tr>
<tr>
<td>declare_nonfix</td>
<td>125</td>
</tr>
<tr>
<td>declare_postfix</td>
<td>125</td>
</tr>
<tr>
<td>declare_prefix</td>
<td>126</td>
</tr>
<tr>
<td>declare_right_infix</td>
<td>125</td>
</tr>
<tr>
<td>delete_axiom</td>
<td>136</td>
</tr>
<tr>
<td>delete_conjecture</td>
<td>152</td>
</tr>
<tr>
<td>delete_const</td>
<td>136</td>
</tr>
<tr>
<td>delete_thm</td>
<td>133</td>
</tr>
<tr>
<td>DeleteAxiom</td>
<td>133</td>
</tr>
<tr>
<td>DeleteConst</td>
<td>133</td>
</tr>
<tr>
<td>DeleteTheory</td>
<td>133</td>
</tr>
<tr>
<td>DeleteThm</td>
<td>133</td>
</tr>
<tr>
<td>delete_all_conjectures</td>
<td>152</td>
</tr>
<tr>
<td>delete_ascending</td>
<td>136</td>
</tr>
<tr>
<td>delete_converse</td>
<td>152</td>
</tr>
<tr>
<td>delete_theory</td>
<td>137</td>
</tr>
<tr>
<td>delete_thm</td>
<td>137</td>
</tr>
<tr>
<td>delete_to_level</td>
<td>137</td>
</tr>
<tr>
<td>delete_type</td>
<td>138</td>
</tr>
<tr>
<td>DEmptyList</td>
<td>82</td>
</tr>
<tr>
<td>DenseIn</td>
<td>411</td>
</tr>
<tr>
<td>DenseIn</td>
<td>412</td>
</tr>
<tr>
<td>dense_complete_subset_thm</td>
<td>414</td>
</tr>
<tr>
<td>dense_superset_thm</td>
<td>413</td>
</tr>
<tr>
<td>dense_universe_thm</td>
<td>413</td>
</tr>
<tr>
<td>Dense</td>
<td>411</td>
</tr>
<tr>
<td>Dense</td>
<td>412</td>
</tr>
<tr>
<td>Dense</td>
<td>412</td>
</tr>
<tr>
<td>DEnumSet</td>
<td>82</td>
</tr>
<tr>
<td>DEq</td>
<td>82</td>
</tr>
<tr>
<td>DerivedRules1</td>
<td>173</td>
</tr>
<tr>
<td>DerivedRules2</td>
<td>173</td>
</tr>
<tr>
<td>dest_app</td>
<td>83</td>
</tr>
<tr>
<td>dest_binder</td>
<td>83</td>
</tr>
<tr>
<td>dest_bin_op</td>
<td>84</td>
</tr>
<tr>
<td>dest_char</td>
<td>84</td>
</tr>
<tr>
<td>dest_const</td>
<td>84</td>
</tr>
<tr>
<td>INDEX</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>EXTEND_PCS_T</td>
<td>344</td>
</tr>
<tr>
<td>EXTEND_PC_CI</td>
<td>342</td>
</tr>
<tr>
<td>EXTEND_PC_C</td>
<td>342</td>
</tr>
<tr>
<td>ext_extend</td>
<td>343</td>
</tr>
<tr>
<td>ext_pc_rule</td>
<td>343</td>
</tr>
<tr>
<td>EXTEND_PC_TI</td>
<td>344</td>
</tr>
<tr>
<td>EXTEND_PC_T</td>
<td>344</td>
</tr>
<tr>
<td>extract_def</td>
<td>495</td>
</tr>
<tr>
<td>Extract</td>
<td>418</td>
</tr>
<tr>
<td>ext_rule</td>
<td>186</td>
</tr>
<tr>
<td>ext_thm</td>
<td>409</td>
</tr>
<tr>
<td>ext_thm</td>
<td>459</td>
</tr>
<tr>
<td>e_delete</td>
<td>33</td>
</tr>
<tr>
<td>e_dict_of_oe_dict</td>
<td>35</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_enter</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_canon</td>
<td>186</td>
</tr>
<tr>
<td>fail_conv</td>
<td>186</td>
</tr>
<tr>
<td>FAIL_C</td>
<td>335</td>
</tr>
<tr>
<td>fail_tac</td>
<td>263</td>
</tr>
<tr>
<td>FAIL_THEN</td>
<td>263</td>
</tr>
<tr>
<td>fail_with_canon</td>
<td>186</td>
</tr>
<tr>
<td>fail_with_conv</td>
<td>186</td>
</tr>
<tr>
<td>fail_with_tac</td>
<td>263</td>
</tr>
<tr>
<td>FAIL_WITH_THEN</td>
<td>263</td>
</tr>
<tr>
<td>Fail</td>
<td>16</td>
</tr>
<tr>
<td>fail</td>
<td>17</td>
</tr>
<tr>
<td>fc_canon1</td>
<td>188</td>
</tr>
<tr>
<td>fc_canon</td>
<td>188</td>
</tr>
<tr>
<td>FC_CAN</td>
<td>187</td>
</tr>
<tr>
<td>fc_rule</td>
<td>189</td>
</tr>
<tr>
<td>FC_TI</td>
<td>267</td>
</tr>
<tr>
<td>fc_tac</td>
<td>265</td>
</tr>
<tr>
<td>FC🌡</td>
<td>266</td>
</tr>
<tr>
<td>fc_c↓_canon</td>
<td>190</td>
</tr>
<tr>
<td>FC_c≈_CAN</td>
<td>189</td>
</tr>
<tr>
<td>filter</td>
<td>21</td>
</tr>
<tr>
<td>find_char</td>
<td>63</td>
</tr>
<tr>
<td>find_in_L_conv</td>
<td>335</td>
</tr>
<tr>
<td>find_name</td>
<td>63</td>
</tr>
<tr>
<td>find_thm</td>
<td>153</td>
</tr>
<tr>
<td>find</td>
<td>21</td>
</tr>
<tr>
<td>finite_def</td>
<td>494</td>
</tr>
<tr>
<td>finite_distinct_elems_thm</td>
<td>394</td>
</tr>
<tr>
<td>finite_distinct_elems_thm</td>
<td>494</td>
</tr>
<tr>
<td>finite_induction_tac</td>
<td>495</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FunctionUtilities</td>
<td>28</td>
</tr>
<tr>
<td>FunReftTheory</td>
<td>493</td>
</tr>
<tr>
<td>funrel_alg</td>
<td>496</td>
</tr>
<tr>
<td>fun_and</td>
<td>28</td>
</tr>
<tr>
<td>fun_false</td>
<td>28</td>
</tr>
<tr>
<td>fun_not</td>
<td>28</td>
</tr>
<tr>
<td>fun_or</td>
<td>28</td>
</tr>
<tr>
<td>fun_pow</td>
<td>28</td>
</tr>
<tr>
<td>fun_rel alg</td>
<td>494</td>
</tr>
<tr>
<td>fun_rel_thm</td>
<td>409</td>
</tr>
<tr>
<td>fun_rel_thm</td>
<td>459</td>
</tr>
<tr>
<td>fun_true</td>
<td>28</td>
</tr>
<tr>
<td>f_def</td>
<td>404</td>
</tr>
<tr>
<td>f_rewrite_canon</td>
<td>223</td>
</tr>
<tr>
<td>f_thm_tac</td>
<td>267</td>
</tr>
<tr>
<td>f_thm</td>
<td>190</td>
</tr>
<tr>
<td>f_thm</td>
<td>408</td>
</tr>
<tr>
<td>F</td>
<td>404</td>
</tr>
<tr>
<td>gc_messages</td>
<td>50</td>
</tr>
<tr>
<td>general_quotation</td>
<td>63</td>
</tr>
<tr>
<td>GEN_ASM_C</td>
<td>317</td>
</tr>
<tr>
<td>gen_find_thm_in_theories</td>
<td>154</td>
</tr>
<tr>
<td>gen_find_thm</td>
<td>154</td>
</tr>
<tr>
<td>GEN_INDUCTION_T1</td>
<td>269</td>
</tr>
<tr>
<td>gen_induction_tac1</td>
<td>269</td>
</tr>
<tr>
<td>gen_induction_tac</td>
<td>268</td>
</tr>
<tr>
<td>GEN_INDUCTION_T</td>
<td>268</td>
</tr>
<tr>
<td>gen_term_order</td>
<td>320</td>
</tr>
<tr>
<td>gen_theory_lister</td>
<td>78</td>
</tr>
<tr>
<td>gen_theory_lister_l</td>
<td>78</td>
</tr>
<tr>
<td>gen_vars</td>
<td>92</td>
</tr>
<tr>
<td>get_aliases</td>
<td>126</td>
</tr>
<tr>
<td>get_aliases_info</td>
<td>127</td>
</tr>
<tr>
<td>get_alias</td>
<td>127</td>
</tr>
<tr>
<td>get_ancestors</td>
<td>140</td>
</tr>
<tr>
<td>GET_ASM_T</td>
<td>269</td>
</tr>
<tr>
<td>GET_ASM_T</td>
<td>269</td>
</tr>
<tr>
<td>get_asm</td>
<td>243</td>
</tr>
<tr>
<td>get_axioms</td>
<td>140</td>
</tr>
<tr>
<td>get_axiom_dict</td>
<td>140</td>
</tr>
<tr>
<td>get_axiom</td>
<td>140</td>
</tr>
<tr>
<td>get_binders</td>
<td>127</td>
</tr>
<tr>
<td>get_brace_braces</td>
<td>64</td>
</tr>
<tr>
<td>get_cache_theories</td>
<td>171</td>
</tr>
<tr>
<td>get_cache_theory</td>
<td>171</td>
</tr>
<tr>
<td>get_children</td>
<td>141</td>
</tr>
<tr>
<td>get_compactification_cache</td>
<td>132</td>
</tr>
<tr>
<td>get_conjectures</td>
<td>152</td>
</tr>
<tr>
<td>get_conjecture</td>
<td>152</td>
</tr>
<tr>
<td>get_consts</td>
<td>141</td>
</tr>
<tr>
<td>get_const_info</td>
<td>127</td>
</tr>
<tr>
<td>get_const_keys</td>
<td>141</td>
</tr>
<tr>
<td>get_const_language</td>
<td>127</td>
</tr>
<tr>
<td>get_const_theory</td>
<td>141</td>
</tr>
<tr>
<td>get_consts_type</td>
<td>141</td>
</tr>
<tr>
<td>get_controls</td>
<td>48</td>
</tr>
<tr>
<td>get_cs_3_cones</td>
<td>358</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
<table>
<thead>
<tr>
<th>Index Entry</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_theory_info</td>
<td>143</td>
</tr>
<tr>
<td>get_theory_names</td>
<td>142</td>
</tr>
<tr>
<td>get_theory_status</td>
<td>142</td>
</tr>
<tr>
<td>get_theory</td>
<td>143</td>
</tr>
<tr>
<td>get_thms</td>
<td>143</td>
</tr>
<tr>
<td>get_thm_dict</td>
<td>143</td>
</tr>
<tr>
<td>get_thm</td>
<td>143</td>
</tr>
<tr>
<td>get_types</td>
<td>143</td>
</tr>
<tr>
<td>get_type_abbr</td>
<td>129</td>
</tr>
<tr>
<td>get_type_abbr</td>
<td>128</td>
</tr>
<tr>
<td>get_type_arity</td>
<td>143</td>
</tr>
<tr>
<td>get_type_info</td>
<td>129</td>
</tr>
<tr>
<td>get_type_keys</td>
<td>143</td>
</tr>
<tr>
<td>get_type_theory</td>
<td>144</td>
</tr>
<tr>
<td>get_undeclared_aliases</td>
<td>129</td>
</tr>
<tr>
<td>get_undeclared_terminators</td>
<td>129</td>
</tr>
<tr>
<td>get_undeclared_type_abbrs</td>
<td>129</td>
</tr>
<tr>
<td>get_unproved_conjectures</td>
<td>151</td>
</tr>
<tr>
<td>get_user_data</td>
<td>144</td>
</tr>
<tr>
<td>get_valid_extended_chars_flag</td>
<td>65</td>
</tr>
<tr>
<td>get_valid_cache_theories</td>
<td>171</td>
</tr>
<tr>
<td>get_variant_suffix</td>
<td>92</td>
</tr>
<tr>
<td>get__cd_thms</td>
<td>362</td>
</tr>
<tr>
<td>get__es_thms</td>
<td>363</td>
</tr>
<tr>
<td>GOAL_STATE</td>
<td>242</td>
</tr>
<tr>
<td>GOAL</td>
<td>251</td>
</tr>
<tr>
<td>graph</td>
<td>22</td>
</tr>
<tr>
<td>grap</td>
<td>397</td>
</tr>
<tr>
<td>graph_at_thm</td>
<td>493</td>
</tr>
<tr>
<td>graph_def</td>
<td>492</td>
</tr>
<tr>
<td>Graph</td>
<td>383</td>
</tr>
<tr>
<td>Graph</td>
<td>384</td>
</tr>
<tr>
<td>greater_conv</td>
<td>435</td>
</tr>
<tr>
<td>greater_def</td>
<td>446</td>
</tr>
<tr>
<td>HasSupremum</td>
<td>411</td>
</tr>
<tr>
<td>HasSupremum</td>
<td>412</td>
</tr>
<tr>
<td>Id</td>
<td>383</td>
</tr>
<tr>
<td>Id</td>
<td>384</td>
</tr>
<tr>
<td>hd_def</td>
<td>402</td>
</tr>
<tr>
<td>hd_def</td>
<td>402</td>
</tr>
<tr>
<td>hd</td>
<td>484</td>
</tr>
<tr>
<td>Hd</td>
<td>22</td>
</tr>
<tr>
<td>Hd</td>
<td>401</td>
</tr>
<tr>
<td>Hd</td>
<td>402</td>
</tr>
<tr>
<td>head_def</td>
<td>495</td>
</tr>
<tr>
<td>Head</td>
<td>418</td>
</tr>
<tr>
<td>hol1</td>
<td>378</td>
</tr>
<tr>
<td>hol2</td>
<td>378</td>
</tr>
<tr>
<td>HOLReaderWriter</td>
<td>57</td>
</tr>
<tr>
<td>HOLSystem</td>
<td>50</td>
</tr>
<tr>
<td>HolTheory</td>
<td>490</td>
</tr>
<tr>
<td>HOL_const_recogniser</td>
<td>158</td>
</tr>
<tr>
<td>HOL_lab_prod_reader</td>
<td>65</td>
</tr>
<tr>
<td>HOL_lab_prod_recogniser</td>
<td>160</td>
</tr>
<tr>
<td>hol_list</td>
<td>7</td>
</tr>
<tr>
<td>HOL_reader</td>
<td>66</td>
</tr>
<tr>
<td>HOL_TOKEN</td>
<td>72</td>
</tr>
<tr>
<td>hol</td>
<td>7</td>
</tr>
<tr>
<td>hol</td>
<td>377</td>
</tr>
<tr>
<td>HTAnd</td>
<td>72</td>
</tr>
</tbody>
</table>
INDEX

new_const .......................... 145  one_def .................................. 490
new_error_message .................. 18  one_fns_thm .................................. 410
new_flag ........................... 48  one_fns_thm .................................. 490
new_init_fun ......................... 51  one_one_def .................................. 405
new_init_control .................... 48  one_one_def .................................. 458
new_parent ........................... 145  one_one_plus1_thm ......................... 447
new_pc ................................ 347  one_one_plus1_thm ......................... 462
new_save_fun ......................... 51  one_one_thm .................................. 409
new_spec ................................ 146  one_one_thm .................................. 459
new_string_control .................. 48  one ............................................. 42
new_theory ........................... 146  ONE ........................................ 410
new_type_defn ........................ 147  One ........................................ 410
new_type ............................. 147  onto_def ................................ 405
nil_cons_def ......................... 402  onto_def ................................ 458
nil_cons_def ......................... 484  Onto .................................... 404
Nil .................................... 19  Onto .................................... 405
Nil .................................... 402  on_kernel_inference ................... 131
nnf_conv ............................. 336  on_kernel_state_change ................. 148
Nonfix ................................ 71  OpenTheory ................................ 133
Normalisation ......................... 317  open_append .............................. 43
normalise_constraint ................. 468  open_in .................................. 43
normalise_poly ........................ 468  open_out .................................. 43
no_order ............................. 36  open_theory .............................. 148
no_submatch_tt ...................... 154  opt_order .................................. 37
no_substring_tt ...................... 154  opt_order .................................. 37
no_subterm_tt ....................... 154  OPT ......................................... 19
nth_def ................................ 495  ORDER ........................................ 36
nth .................................... 25  ORELSE ..................................... 197
Nth ..................................... 418  ORELSE_C .................................. 197
Nth ..................................... 419  ORELSE_T .................................. 277
num_lit_of_string ................... 74  ORELSE ..................................... 277
oe_delete ............................. 35  ORELSE ..................................... 425
OE_DICT .............................. 35  OutL ....................................... 426
oe_enter .............................. 35  OutL ....................................... 78
oe_extend ............................. 35  output_theory ......................... 79
oe_flatten ............................ 35  output_theory ......................... 43
oe_key_delete ....................... 35  output ...................................... 43
oe_key_enter ......................... 35  output ...................................... 426
oe_key_extend ....................... 35  output ...................................... 277
oe_key_flatten ...................... 35  output ...................................... 425
oe_key_lookup ....................... 35  output ...................................... 197
oe_lookup ............................ 35  overwrite .................................. 25
oe_merge ............................. 35  o_assoc_thm ............................... 389
once_asm_rewrite_rule .............. 204  o_assoc_thm ............................... 490
once_asm_rewrite_tac ................ 283  o_def ...................................... 389
once_asm_rewrite_thm_tac .......... 283  o_def ...................................... 490
ONCE_MAP_C .......................... 196  o_1_thm .................................. 389
ONCE_MAP_WARN_C .................... 197  o .......................................... 405
once_rewrite_conv ................... 203  o .......................................... 389
once_rewrite_rule ................... 204  PairTheory .............................. 459
once_rewrite_tac ..................... 283  pair_clauses ............................ 417
once_rewrite_thm_tac ............... 283  pair_clauses ............................ 459
OneOne ................................ 404  pair_eq_conv ............................. 314
OneOne ................................ 405  pair_ops_def ............................. 416
OneTheory ............................ 490  pair_ops_def ............................. 459
one_def ................................ 410  pair_order ................................ 37
one_def ................................ 490  pair_rw_canon ......................... 278
one_def ................................ 410  pass_on .................................. 18

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
<table>
<thead>
<tr>
<th>Function/Command</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPArray</td>
<td>44</td>
</tr>
<tr>
<td>PPVector</td>
<td>44</td>
</tr>
<tr>
<td>pending_push_extend_pcs</td>
<td>350</td>
</tr>
<tr>
<td>pending_push_extend_pc</td>
<td>350</td>
</tr>
<tr>
<td>pending_push_merge_pcs</td>
<td>350</td>
</tr>
<tr>
<td>pending_push_pc</td>
<td>350</td>
</tr>
<tr>
<td>pending_reset_control_state</td>
<td>49</td>
</tr>
<tr>
<td>pending_reset_error_messages</td>
<td>18</td>
</tr>
<tr>
<td>pending_reset_kernel_interface</td>
<td>148</td>
</tr>
<tr>
<td>pending_reset_pc_database</td>
<td>351</td>
</tr>
<tr>
<td>pending_reset_pc_evaluators</td>
<td>351</td>
</tr>
<tr>
<td>pending_reset_pc_stack</td>
<td>244</td>
</tr>
<tr>
<td>pigeon_hole_thm</td>
<td>395</td>
</tr>
<tr>
<td>pigeon_hole_thm_prefix</td>
<td>494</td>
</tr>
<tr>
<td>plus1_conv</td>
<td>461</td>
</tr>
<tr>
<td>plus1_less_thm</td>
<td>448</td>
</tr>
<tr>
<td>plus1_less_thm1</td>
<td>447</td>
</tr>
<tr>
<td>plus1_leq_thm</td>
<td>448</td>
</tr>
<tr>
<td>plus1_leq_thm1</td>
<td>447</td>
</tr>
<tr>
<td>plus_assoc_thm1</td>
<td>462</td>
</tr>
<tr>
<td>plus_assoc_thm11</td>
<td>462</td>
</tr>
<tr>
<td>plus_assoc_thm</td>
<td>447</td>
</tr>
<tr>
<td>plus_assoc_thm1</td>
<td>462</td>
</tr>
<tr>
<td>plus_clauses</td>
<td>447</td>
</tr>
<tr>
<td>plus_clauses1</td>
<td>462</td>
</tr>
<tr>
<td>plus_comm_thm</td>
<td>447</td>
</tr>
<tr>
<td>plus_comm_thm1</td>
<td>462</td>
</tr>
<tr>
<td>plus_cone</td>
<td>198</td>
</tr>
<tr>
<td>plus_cone1</td>
<td>465</td>
</tr>
<tr>
<td>plus_def</td>
<td>446</td>
</tr>
<tr>
<td>plus_def1</td>
<td>461</td>
</tr>
<tr>
<td>plus_order_thm</td>
<td>447</td>
</tr>
<tr>
<td>plus_order_thm1</td>
<td>462</td>
</tr>
<tr>
<td>poly_cone</td>
<td>321</td>
</tr>
<tr>
<td>POLY</td>
<td>468</td>
</tr>
<tr>
<td>POP_ASM_T</td>
<td>278</td>
</tr>
<tr>
<td>pop_pc</td>
<td>351</td>
</tr>
<tr>
<td>pop_thm</td>
<td>245</td>
</tr>
<tr>
<td>Postfix</td>
<td>71</td>
</tr>
<tr>
<td>pp_change_error_message</td>
<td>19</td>
</tr>
<tr>
<td>pp_database_info</td>
<td>50</td>
</tr>
<tr>
<td>pp_error_init</td>
<td>19</td>
</tr>
<tr>
<td>pp_reset_database_info</td>
<td>51</td>
</tr>
<tr>
<td>pp_set_banner</td>
<td>51</td>
</tr>
<tr>
<td>pp_set_database_info</td>
<td>51</td>
</tr>
<tr>
<td>pp_set Eval_ad_con</td>
<td>353</td>
</tr>
<tr>
<td>pp_set_eval_ad_con1</td>
<td>353</td>
</tr>
<tr>
<td>pp_set_eval_ad_net</td>
<td>353</td>
</tr>
<tr>
<td>pp_set_eval_ad_net1</td>
<td>352</td>
</tr>
<tr>
<td>pp_set_eval_ad_rwt_canon</td>
<td>352</td>
</tr>
<tr>
<td>pp_set_eval_ad_rwt</td>
<td>351</td>
</tr>
<tr>
<td>pp_set_eval_ad_so_con</td>
<td>352</td>
</tr>
<tr>
<td>pp_set_eval_ad_st_con</td>
<td>352</td>
</tr>
<tr>
<td>pp_set_eval_ad_od_thms</td>
<td>353</td>
</tr>
<tr>
<td>pp_set_eval_ad_vs_thms</td>
<td>354</td>
</tr>
<tr>
<td>pp' theory_hierarchy</td>
<td>52</td>
</tr>
<tr>
<td>pp' ts_def</td>
<td>407</td>
</tr>
<tr>
<td>pp' TS</td>
<td>241</td>
</tr>
<tr>
<td>pp' TS</td>
<td>407</td>
</tr>
<tr>
<td>pp' TypesAndTerms</td>
<td>81</td>
</tr>
<tr>
<td>pp' format_depth</td>
<td>75</td>
</tr>
<tr>
<td>pp' init</td>
<td>76</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>75</td>
</tr>
<tr>
<td>pp_top_level_depth</td>
<td>75</td>
</tr>
<tr>
<td>pp' predicates1</td>
<td>367</td>
</tr>
<tr>
<td>predicates</td>
<td>367</td>
</tr>
<tr>
<td>prefix_def</td>
<td>495</td>
</tr>
<tr>
<td>PrettyPrinter</td>
<td>75</td>
</tr>
<tr>
<td>PRETTY_NAME</td>
<td>60</td>
</tr>
<tr>
<td>prim_rec_thm</td>
<td>447</td>
</tr>
<tr>
<td>prim_rec_thm1</td>
<td>462</td>
</tr>
<tr>
<td>prim_res_rule</td>
<td>333</td>
</tr>
<tr>
<td>prim_rewrite_conv</td>
<td>199</td>
</tr>
<tr>
<td>prim_rewrite_rule</td>
<td>200</td>
</tr>
<tr>
<td>prim_rewrite_tac</td>
<td>279</td>
</tr>
<tr>
<td>prim_suc_conv</td>
<td>200</td>
</tr>
<tr>
<td>print_banner</td>
<td>52</td>
</tr>
<tr>
<td>print_current_goal</td>
<td>245</td>
</tr>
<tr>
<td>print_goal</td>
<td>245</td>
</tr>
<tr>
<td>print_good_state</td>
<td>245</td>
</tr>
<tr>
<td>print_stats</td>
<td>55</td>
</tr>
<tr>
<td>print_status</td>
<td>52</td>
</tr>
<tr>
<td>print_theory</td>
<td>79</td>
</tr>
<tr>
<td>ProductTypes</td>
<td>167</td>
</tr>
<tr>
<td>Profiling</td>
<td>55</td>
</tr>
<tr>
<td>profiling</td>
<td>55</td>
</tr>
<tr>
<td>prof</td>
<td>55</td>
</tr>
<tr>
<td>ProofContexts1</td>
<td>364</td>
</tr>
<tr>
<td>ProofContext</td>
<td>338</td>
</tr>
<tr>
<td>PROOF</td>
<td>251</td>
</tr>
<tr>
<td>PropositionalEquality</td>
<td>313</td>
</tr>
<tr>
<td>PROP_ATOM_C</td>
<td>336</td>
</tr>
<tr>
<td>prop_eq_pair</td>
<td>313</td>
</tr>
<tr>
<td>prop_eq_prove_tac</td>
<td>315</td>
</tr>
<tr>
<td>prop_eq_rule</td>
<td>316</td>
</tr>
<tr>
<td>prop_EQ_T</td>
<td>314</td>
</tr>
<tr>
<td>prop_eq</td>
<td>313</td>
</tr>
<tr>
<td>prop_LIT_C</td>
<td>336</td>
</tr>
<tr>
<td>prop.simps_con</td>
<td>336</td>
</tr>
<tr>
<td>INDEX</td>
<td>534</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>STRIP_THM_THEN</td>
<td>295</td>
</tr>
<tr>
<td>STRIP_T</td>
<td>294</td>
</tr>
<tr>
<td>strip_e</td>
<td>114</td>
</tr>
<tr>
<td>strip_3</td>
<td>114</td>
</tr>
<tr>
<td>strip_v</td>
<td>114</td>
</tr>
<tr>
<td>strip_λ</td>
<td>114</td>
</tr>
<tr>
<td>strip_rule</td>
<td>214</td>
</tr>
<tr>
<td>strip_ /_rule</td>
<td>214</td>
</tr>
<tr>
<td>strip_⇒_rule</td>
<td>214</td>
</tr>
<tr>
<td>strip_⇒_type</td>
<td>113</td>
</tr>
<tr>
<td>strip⇒</td>
<td>114</td>
</tr>
<tr>
<td>SubgoalPackage</td>
<td>241</td>
</tr>
<tr>
<td>subgoal_package_quiet</td>
<td>241</td>
</tr>
<tr>
<td>subgoal_package_size</td>
<td>248</td>
</tr>
<tr>
<td>subgoal_package_to_context</td>
<td>242</td>
</tr>
<tr>
<td>SubsetClosed</td>
<td>424</td>
</tr>
<tr>
<td>subset</td>
<td>26</td>
</tr>
<tr>
<td>subst_conv</td>
<td>215</td>
</tr>
<tr>
<td>subst_rule</td>
<td>216</td>
</tr>
<tr>
<td>subst</td>
<td>115</td>
</tr>
<tr>
<td>SUB_C1</td>
<td>217</td>
</tr>
<tr>
<td>SUB_C</td>
<td>217</td>
</tr>
<tr>
<td>sub_opt</td>
<td>39</td>
</tr>
<tr>
<td>sub</td>
<td>39</td>
</tr>
<tr>
<td>suc_conv</td>
<td>218</td>
</tr>
<tr>
<td>Suc</td>
<td>445</td>
</tr>
<tr>
<td>Suc</td>
<td>446</td>
</tr>
<tr>
<td>suffix_def</td>
<td>495</td>
</tr>
<tr>
<td>Suffix</td>
<td>418</td>
</tr>
<tr>
<td>Suffix</td>
<td>419</td>
</tr>
<tr>
<td>SumTheory</td>
<td>490</td>
</tr>
<tr>
<td>sum_cases_tac</td>
<td>491</td>
</tr>
<tr>
<td>sum_cases_thm</td>
<td>426</td>
</tr>
<tr>
<td>sum_cases_thm</td>
<td>490</td>
</tr>
<tr>
<td>sum_clauses</td>
<td>426</td>
</tr>
<tr>
<td>sum_clauses</td>
<td>490</td>
</tr>
<tr>
<td>sum_def</td>
<td>425</td>
</tr>
<tr>
<td>sum_def</td>
<td>490</td>
</tr>
<tr>
<td>sum_fns_thm</td>
<td>426</td>
</tr>
<tr>
<td>sum_fns_thm</td>
<td>490</td>
</tr>
<tr>
<td>Sup_consistent</td>
<td>431</td>
</tr>
<tr>
<td>Sup</td>
<td>427</td>
</tr>
<tr>
<td>Sup</td>
<td>428</td>
</tr>
<tr>
<td>surjective_def</td>
<td>492</td>
</tr>
<tr>
<td>Surjective</td>
<td>383</td>
</tr>
<tr>
<td>Surjective</td>
<td>384</td>
</tr>
<tr>
<td>swap_asm_concl_tac</td>
<td>295</td>
</tr>
<tr>
<td>SWAP_ASM_CONCL_T</td>
<td>296</td>
</tr>
<tr>
<td>swap_nth_asm_concl_tac</td>
<td>295</td>
</tr>
<tr>
<td>SWAP_NTH_ASM_CONCL_T</td>
<td>296</td>
</tr>
<tr>
<td>swap_v_tac</td>
<td>296</td>
</tr>
<tr>
<td>swap</td>
<td>29</td>
</tr>
<tr>
<td>switch</td>
<td>29</td>
</tr>
<tr>
<td>SymbolTable</td>
<td>124</td>
</tr>
<tr>
<td>SYMBOL</td>
<td>59</td>
</tr>
</tbody>
</table>

SymCharacter                                                | 59  |
SymDoublePercent                                            | 59  |
SymEndOfFileInput                                           | 59  |
SymKnown                                                   | 59  |
symmetric_def                                              | 492 |
Symmetric                                                   | 383 |
Symmetric                                                   | 384 |
SymUnknownChar                                             | 59  |
SymUnknownKw                                               | 59  |
SystemControl                                              | 48  |
system_banner                                              | 51  |
system                                                    | 43  |
S_DICT                                                     | 19  |
s_delete                                                   | 32  |
s_enter                                                   | 32  |
s_extend                                                  | 32  |
tactic_subgoal_warning                                     | 242 |
TACTIC                                                     | 251 |
tail_def                                                  | 495 |
Tail                                                       | 418 |
TALL                                                       | 153 |
TAny                                                       | 153 |
taut_conw                                                  | 297 |
taut_rule                                                  | 298 |
taut_tac                                                   | 298 |
term_any                                                   | 115 |
term_consts                                                | 115 |
term_diff                                                  | 115 |
term_fail                                                  | 115 |
term_fold                                                  | 115 |
term_grab                                                  | 115 |
term_less                                                  | 116 |
term_map                                                   | 116 |
term_match                                                 | 116 |
term_mem                                                  | 116 |
term_order                                                 | 322 |
term_tycons                                                | 116 |
term_types                                                 | 116 |
term_tyvars                                                | 116 |
term_unify                                                 | 334 |
term_union                                                 | 116 |
term_vars                                                  | 116 |
Term                                                       | 72  |
TERM                                                       | 82  |
TEST                                                       | 153 |
tex2dvi                                                    | 13  |
Text                                                        | 72  |
TFun                                                       | 153 |

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
UnboundedAbove ........................................ 411  Zero .................................................. 446
UnboundedAbove ........................................ 412  zip ......................................................... 27
UnboundedBelow ......................................... 411  zorn_thm ................................................. 424
UnboundedBelow ......................................... 412  zorn_thm ................................................. 398
uncurry_def ............................................ 417  # ....................................................... 393
uncurry_def ............................................ 459  # ....................................................... 418
uncurry .................................................. 29  α_conv ................................................. 236
Uncurry ................................................... 416  β_conv ................................................... 237
Uncurry ................................................... 417  β_rewrite_thm ........................................... 184
undeclare_alias ......................................... 130  β_rewrite_thm ........................................... 409
undeclare_terminator .................................. 130  β_rule ................................................... 237
undeclare_type_abbrev ................................ 130  ∩ clauses ................................................. 422
undisch_rule ........................................... 219  ∩ clauses ................................................. 486
undo_buffer_length .................................... 242  ∩ def ..................................................... 421
undo ....................................................... 250  ∩ def ..................................................... 486
union ..................................................... 27  ∧ ........................................................ 420
 Universe ............................................... 420  ∧ ........................................................ 421
unlabelled_product_spec ................................ 172  ∪ clauses ................................................. 486
UnlockTheory ........................................... 133  ∪ def ..................................................... 421
unlock_theory ........................................... 150  ∪ def ..................................................... 486
update ..................................................... 40  ∪ finite_thm ............................................ 395
update ..................................................... 41  ∪ finite_thm ............................................ 494
UpperBounds ............................................ 398  ∪ ......................................................... 420
UpperBound .............................................. 411  ∪ ......................................................... 421
UpperBound .............................................. 412  ∪ ......................................................... 486
user_banner .............................................. 51  ∩ clauses ................................................. 422
USER_DATA ............................................... 123  ∩ def ..................................................... 486
USER_DATUM ............................................ 121  ∩ def ..................................................... 486
use_extended_chars .................................... 58  ∩ finite_thm ............................................ 394
use_file .................................................. 68  ∩ finite_thm ............................................ 494
use_file_non_stop_mode ................................. 58  ∩ ......................................................... 420
use_file .................................................. 68  ∩ ......................................................... 421
use_string ................................................ 68  ∩ ......................................................... 421
use_terminal ............................................ 68  ∪ clauses ................................................. 422
UtilitySharedTypes ...................................... 19  ∪ clauses ................................................. 486
valid_thm ................................................ 150  ∪ def ..................................................... 421
Value ...................................................... 19  ∪ def ..................................................... 486
variant .................................................... 118  ∪ finite_thm ............................................ 394
varstruct_variant ....................................... 220  ∪ finite_thm ............................................ 494
Vartype ................................................... 81  ∪ ......................................................... 420
var_elim_asm_tac ....................................... 301 \ ..................................................... 420
VAR_ELIM_ASM_T ....................................... 301 \ ..................................................... 421
var_elim_nth_asm_tac ................................... 301 < ∩ def .................................................. 492
VAR_ELIM_NTH_ASM_T .................................. 301 < ..................................................... 383
var_subst ................................................ 118 < ..................................................... 384
Var ........................................................ 81 ∈ clauses ............................................... 422
var_intro ................................................ 220 ∈ clauses ............................................... 486
Warning ................................................... 54 ∈ axiom ............................................... 400
warn ....................................................... 54 ∈ axiom ............................................... 458
which ....................................................... 27 ∈ elim_rule ............................................. 238
xpp ......................................................... 11 ∈ intro_rule ............................................. 238
zet_list ................................................... 11 ∈_tac ................................................... 312
zed ......................................................... 7 ∈ T ......................................................... 312
zero_suc_def ............................................ 446 ∈ ......................................................... 406
zero_suc_def ............................................ 461 ∈ axiom ............................................... 400
zero ........................................................ 42 ∈ axiom ............................................... 458
Zero ....................................................... 445 ∈ conv ................................................... 238

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
<p>| \texttt{R_complete_thm} | 511 | \texttt{R_greater_conv} | 506 |
| \texttt{R_copah_comp_thm} | 437 | \texttt{R_greater_def} | 509 |
| \texttt{R_copah_comp_thm} | 502 | \texttt{R_halve_closed_dense_thm} | 438 |
| \texttt{R_copah_dense_thm} | 438 | \texttt{R_halve_closed_dense_thm} | 502 |
| \texttt{R_copah_dense_thm} | 502 | \texttt{R_less_0_less_thm} | 434 |
| \texttt{R_copah_double_thm} | 437 | \texttt{R_less_0_less_thm} | 508 |
| \texttt{R_copah_double_thm} | 502 | \texttt{R_less_antisym_thm} | 430 |
| \texttt{R_copah_halve_thm} | 437 | \texttt{R_less_antisym_thm} | 511 |
| \texttt{R_copah_halve_thm} | 502 | \texttt{R_less_cancel_conv} | 504 |
| \texttt{R_copah_id_thm} | 437 | \texttt{R_less_cases_thm} | 430 |
| \texttt{R_copah_id_thm} | 502 | \texttt{R_less_cases_thm} | 511 |
| \texttt{R_copah_sum_thm} | 438 | \texttt{R_less clauses} | 434 |
| \texttt{R_copah_sum_thm} | 502 | \texttt{R_less clauses} | 508 |
| \texttt{R_cross_mul_eq_thm} | 441 | \texttt{R_less_conv} | 508 |
| \texttt{R_cross_mul_eq_thm} | 510 | \texttt{R_less_def} | 509 |
| \texttt{R_cross_mul_less_thm} | 441 | \texttt{R_less_dense_thm} | 431 |
| \texttt{R_cross_mul_less_thm} | 510 | \texttt{R_less_dense_thm} | 511 |
| \texttt{R_def} | 428 | \texttt{R_less_irrefl_thm} | 430 |
| \texttt{R_delta_induction_tac} | 503 | \texttt{R_less_irrefl_thm} | 511 |
| \texttt{R_delta_induction_thm} | 435 | \texttt{R_less_0_thm} | 434 |
| \texttt{R_delta_induction_thm} | 502 | \texttt{R_less_0_thm} | 508 |
| \texttt{R_eq_cancel_conv} | 504 | \texttt{R_less_strong_dense_thm} | 434 |
| \texttt{R_eq Conv} | 508 | \texttt{R_less_strong_dense_thm} | 508 |
| \texttt{R_eq_recip_thm} | 440 | \texttt{R_less_sup_bc_thm} | 431 |
| \texttt{R_eq_recip_thm} | 507 | \texttt{R_less_sup_bc_thm} | 509 |
| \texttt{R_eq_sup_bc_thm} | 432 | \texttt{R_less_sup_thm} | 431 |
| \texttt{R_eq_sup_bc_thm} | 509 | \texttt{R_less_sup_thm} | 509 |
| \texttt{R_eq_thm} | 433 | \texttt{R_less_sup} | 432 |
| \texttt{R_eq_thm} | 508 | \texttt{R_less_trans_thm} | 511 |
| \texttt{R_eq le thm} | 430 | \texttt{R_less_trans_thm} | 430 |
| \texttt{R_eval_conv} | 504 | \texttt{R_less le_thm} | 511 |
| \texttt{R_EVAL C} | 504 | \texttt{R_less_thm} | 511 |
| \texttt{R_frac 0 thm} | 441 | \texttt{R_less le_thm} | 511 |
| \texttt{R_frac 0 thm} | 505 | \texttt{R_less_eq 0 thm} | 441 |
| \texttt{R_frac cancel_eq_thm} | 441 | \texttt{R_less_eq 0 thm} | 508 |
| \texttt{R_frac cancel_eq_thm} | 505 | \texttt{R_less_eq thm} | 431 |
| \texttt{R_frac cross_mul_eq_thm} | 441 | \texttt{R_less_eq thm} | 511 |
| \texttt{R_frac cross_mul_eq_thm} | 505 | \texttt{R_lin arith prove_thm} | 506 |
| \texttt{R_frac def} | 509 | \texttt{R_lin arith rule} | 506 |
| \texttt{R_frac less frac thm} | 442 | \texttt{R_lin arith_tac} | 506 |
| \texttt{R_frac less frac thm} | 442 | \texttt{R_lin arith} | 498 |
| \texttt{R_frac less frac thm} | 505 | \texttt{R_lin geq conv} | 506 |
| \texttt{R_frac less frac thm} | 505 | \texttt{R_max cons thm} | 443 |
| \texttt{R_frac less frac thm} | 505 | \texttt{R_max cons thm} | 506 |
| \texttt{R_frac less frac thm} | 505 | \texttt{R_max conv thm} | 443 |
| \texttt{R_frac less frac thm} | 505 | \texttt{R_max conv thm} | 506 |
| \texttt{R_frac less frac thm} | 505 | \texttt{R_max def} | 509 |
| \texttt{R_frac norm_conv} | 505 | \texttt{R_minus clauses} | 433 |
| \texttt{R_frac plus frac conv} | 505 | \texttt{R_minus clauses} | 508 |
| \texttt{R_frac plus frac conv} | 505 | \texttt{R_minus conv} | 507 |
| \texttt{R_frac plus frac conv} | 441 | \texttt{R_minus def} | 509 |
| \texttt{R_frac plus frac conv} | 505 | \texttt{R-minus eq thm} | 433 |
| \texttt{R_frac recip_thm} | 442 | \texttt{R-minus eq thm} | 508 |
| \texttt{R_frac recip_thm} | 505 | \texttt{R-minus frac less frac thm} | 442 |
| \texttt{R_frac times frac thm} | 442 | \texttt{R-minus frac less frac thm} | 505 |
| \texttt{R_frac times frac thm} | 505 | \texttt{R-minus recip thm} | 442 |
| \texttt{R_frac N thm} | 441 | \texttt{R-minus recip thm} | 442 |
| \texttt{R_frac N thm} | 505 | \texttt{R-minus recip thm} | 505 |</p>
<table>
<thead>
<tr>
<th>INDEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>543</td>
</tr>
</tbody>
</table>

| lemmas  | clauses | 462 | _∃_ thm | 228 |
|         | conv    | 465 | _∃_ thm | 408 |
|         | def     | 446 | _∀_ conv | 227 |
|         | lemmas  | 461 | _∀_ thm | 227 |
|         | least_upper_bound_thm | 448 | _∀_ thm | 408 |
|         | least_upper_bound_thm | 462 | _∀_ thm | 309 |
|         | max_thm | 395 | _≤_ thm | 408 |
|         | max_thm | 494 | _≤_ thm | 309 |
|         | plus1_thm | 448 | _≤_ thm | 408 |
|         | plus1_thm | 462 | _≤_ thm | 448 |
|         | trans_thm | 447 | _≤_ thm | 462 |
|         | trans_thm | 462 | _<_ conv | 226 |
|         | well_order_thm | 448 | _<_ elim | 227 |
|         | well_order_thm | 462 | _<_ intro | 227 |
|         | R       | 427 | _<_ thm | 309 |
|         | R       | 428 | _<_ thm | 408 |
|         | Z       | 450 | _<_ thm | 309 |
|         | Z       | 451 | _<_ thm | 408 |
|         |        | 427 | _<=> thm | 309 |
|         |        | 445 | _<=> thm | 408 |
|         |        | 446 | _<= thm | 404 |
|         |        | 450 | _<=> thm | 224 |
|         | def     | 404 | _<=> thm | 405 |
|         | def     | 458 | _<=> thm | 458 |
|         | elim_tac | 305 | _<=> thm | 224 |
|         | elim     | 225 | _<=> thm | 224 |
|         | eg_sym_rule | 225 | _<=> thm | 304 |
|         | f_thm   | 309 | _<=> thm | 184 |
|         | f_thm   | 408 | _<=> thm | 408 |
|         | if_thm  | 309 | _<=> thm | 408 |
|         | intro    | 225 | _<=> thm | 225 |
|         | in_conv  | 305 | _<=> thm | 408 |
|         | tac      | 306 | _<=> thm | 225 |
|         | IN.THEN | 306 | _<=> thm | 408 |
|         | less_plus1_thm | 448 | _<=> thm | 404 |
|         | less_plus1_thm | 462 | _<=> thm | 405 |
|         | less_thm | 448 | _<=> thm | 492 |
|         | less_thm | 462 | _<=> thm | 492 |
|         | plus1_thm | 447 | _<=> thm | 492 |
|         | plus1_thm | 462 | _<=> thm | 492 |
|         | plus1_≤_thm | 448 | _<=> thm | 383 |
|         | plus1_≤_thm | 462 | _<=> thm | 383 |
|         | rewrite_canon | 307 | _<=> thm | 384 |
|         | rewrite_thm | 184 | _<=> thm | 492 |
|         | rewrite_thm | 408 | _<=> thm | 492 |
|         | simple_∃_conv | 226 | _<=> thm | 383 |
|         | simple_∀_conv | 226 | _<=> thm | 383 |
|         | T2      | 307 | _<=> thm | 384 |
|         | tac     | 308 | _<=> thm | 418 |
|         | thm1    | 226 | _<=> thm | 495 |
|         | thm1    | 408 | _<=> thm | 493 |
|         | thm     | 226 | _<=> thm | 396 |
|         | thm     | 408 | _<=> thm | 427 |
|         | t.thm   | 226 | _<=> thm | 428 |
|         | t.thm   | 408 | _<=> thm | 429 |
|         | T       | 308 | _<=> thm | 443 |
|         | Z       | 227 | _<=> thm | 427 |

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
KEYWORD INDEX

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic_prove_3_conv</td>
<td>364</td>
</tr>
<tr>
<td>bin_rel</td>
<td>493</td>
</tr>
<tr>
<td>char</td>
<td>371</td>
</tr>
<tr>
<td>combin</td>
<td>374</td>
</tr>
<tr>
<td>fun_ext</td>
<td>367</td>
</tr>
<tr>
<td>fun_rel_ext</td>
<td>493</td>
</tr>
<tr>
<td>lin_arith</td>
<td>467</td>
</tr>
<tr>
<td>list</td>
<td>371</td>
</tr>
<tr>
<td>mmp1</td>
<td>372</td>
</tr>
<tr>
<td>mmp2</td>
<td>372</td>
</tr>
<tr>
<td>one</td>
<td>374</td>
</tr>
<tr>
<td>pair</td>
<td>308</td>
</tr>
<tr>
<td>pair1</td>
<td>369</td>
</tr>
<tr>
<td>paired_abstractions</td>
<td>365</td>
</tr>
<tr>
<td>prop_eq</td>
<td>313</td>
</tr>
<tr>
<td>prop_eq_pair</td>
<td>313</td>
</tr>
<tr>
<td>propositions</td>
<td>366</td>
</tr>
<tr>
<td>sets_alg</td>
<td>375</td>
</tr>
<tr>
<td>sets_ext</td>
<td>376</td>
</tr>
<tr>
<td>sets_ext1</td>
<td>377</td>
</tr>
<tr>
<td>sho_rw</td>
<td>372</td>
</tr>
<tr>
<td>simple_abstractions</td>
<td>365</td>
</tr>
<tr>
<td>sum</td>
<td>373</td>
</tr>
<tr>
<td>R</td>
<td>497</td>
</tr>
<tr>
<td>R_lin_arith</td>
<td>498</td>
</tr>
<tr>
<td>N</td>
<td>369</td>
</tr>
<tr>
<td>N_lit</td>
<td>370</td>
</tr>
<tr>
<td>Z</td>
<td>474</td>
</tr>
<tr>
<td>Z_lin_arith</td>
<td>482</td>
</tr>
<tr>
<td>*</td>
<td>427</td>
</tr>
<tr>
<td>*</td>
<td>445</td>
</tr>
<tr>
<td>*</td>
<td>446</td>
</tr>
<tr>
<td>*</td>
<td>450</td>
</tr>
<tr>
<td>**</td>
<td>28</td>
</tr>
<tr>
<td>R</td>
<td>427</td>
</tr>
<tr>
<td>R</td>
<td>428</td>
</tr>
<tr>
<td>R</td>
<td>429</td>
</tr>
<tr>
<td>R</td>
<td>439</td>
</tr>
<tr>
<td>R</td>
<td>450</td>
</tr>
<tr>
<td>R</td>
<td>451</td>
</tr>
<tr>
<td>R</td>
<td>453</td>
</tr>
<tr>
<td>+</td>
<td>425</td>
</tr>
<tr>
<td>+</td>
<td>427</td>
</tr>
<tr>
<td>+</td>
<td>445</td>
</tr>
<tr>
<td>+</td>
<td>446</td>
</tr>
<tr>
<td>+</td>
<td>450</td>
</tr>
<tr>
<td>R_add_hom</td>
<td>427</td>
</tr>
<tr>
<td>R_add_hom</td>
<td>428</td>
</tr>
<tr>
<td>R_add_hom</td>
<td>432</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
### KEYWORD INDEX

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ALL_FORWARDCHAIN_T</code></td>
<td>266</td>
</tr>
<tr>
<td><code>ALL_FORWARDCHAIN_T1</code></td>
<td>267</td>
</tr>
<tr>
<td><code>all_forward_chain_tac</code></td>
<td>265</td>
</tr>
<tr>
<td><code>ALL_SIMPLE ∀ C</code></td>
<td>173</td>
</tr>
<tr>
<td><code>all_simple ∀ elim</code></td>
<td>174</td>
</tr>
<tr>
<td><code>ALL_SIMPLE ∃ C</code></td>
<td>174</td>
</tr>
<tr>
<td><code>all_simple β_conv</code></td>
<td>174</td>
</tr>
<tr>
<td><code>all_simple β_rule</code></td>
<td>174</td>
</tr>
<tr>
<td><code>all_submatch_tt</code></td>
<td>154</td>
</tr>
<tr>
<td><code>all_substring_tt</code></td>
<td>154</td>
</tr>
<tr>
<td><code>all_subterm_tt</code></td>
<td>154</td>
</tr>
<tr>
<td><code>T All</code></td>
<td>153</td>
</tr>
<tr>
<td><code>ALL_VAR_ELIMASM T</code></td>
<td>252</td>
</tr>
<tr>
<td><code>ALL_VAR_ELIMASM T1</code></td>
<td>252</td>
</tr>
<tr>
<td><code>all_var_elim_asm tac I</code></td>
<td>252</td>
</tr>
<tr>
<td><code>ALL ∧ C</code></td>
<td>175</td>
</tr>
<tr>
<td><code>ALL ∃ C</code></td>
<td>175</td>
</tr>
<tr>
<td><code>all ⇒ intro</code></td>
<td>175</td>
</tr>
<tr>
<td><code>all ∀ arb_elim</code></td>
<td>175</td>
</tr>
<tr>
<td><code>all ∀ elim</code></td>
<td>175</td>
</tr>
<tr>
<td><code>all ∀ intro</code></td>
<td>176</td>
</tr>
<tr>
<td><code>all ∀ uncurry_conv</code></td>
<td>176</td>
</tr>
<tr>
<td><code>ALL ∃ uncurry_conv</code></td>
<td>176</td>
</tr>
<tr>
<td><code>all β_conv</code></td>
<td>177</td>
</tr>
<tr>
<td><code>all β_rule</code></td>
<td>177</td>
</tr>
<tr>
<td><code>all β tac</code></td>
<td>253</td>
</tr>
<tr>
<td><code>ALL ∧ T</code></td>
<td>253</td>
</tr>
<tr>
<td><code>all t tac</code></td>
<td>253</td>
</tr>
<tr>
<td><code>is_theory ancestor</code></td>
<td>144</td>
</tr>
<tr>
<td><code>TS Ancestor</code></td>
<td>121</td>
</tr>
<tr>
<td><code>get ancestors</code></td>
<td>140</td>
</tr>
<tr>
<td><code>mult and add constraint</code></td>
<td>468</td>
</tr>
<tr>
<td><code>mult and add poly</code></td>
<td>468</td>
</tr>
<tr>
<td><code>save and exit</code></td>
<td>53</td>
</tr>
<tr>
<td><code>fun and</code></td>
<td>28</td>
</tr>
<tr>
<td><code>HT And</code></td>
<td>72</td>
</tr>
<tr>
<td><code>skip and look at next</code></td>
<td>60</td>
</tr>
<tr>
<td><code>AND OR C</code></td>
<td>177</td>
</tr>
<tr>
<td><code>save and quit</code></td>
<td>53</td>
</tr>
<tr>
<td><code>pp'Types AndTerms</code></td>
<td>81</td>
</tr>
<tr>
<td><code>Types AndTerms</code></td>
<td>81</td>
</tr>
<tr>
<td><code>ANF C</code></td>
<td>317</td>
</tr>
<tr>
<td><code>R ANF C</code></td>
<td>502</td>
</tr>
<tr>
<td><code>Z ANF C</code></td>
<td>482</td>
</tr>
<tr>
<td><code>anf_conv</code></td>
<td>317</td>
</tr>
<tr>
<td><code>R anf_conv</code></td>
<td>502</td>
</tr>
<tr>
<td><code>Z anf_conv</code></td>
<td>482</td>
</tr>
<tr>
<td><code>ante tac</code></td>
<td>253</td>
</tr>
<tr>
<td><code>all_asm ante tac</code></td>
<td>251</td>
</tr>
<tr>
<td><code>asm ante tac</code></td>
<td>254</td>
</tr>
<tr>
<td><code>list_asm ante tac</code></td>
<td>274</td>
</tr>
<tr>
<td><code>Antisym</code></td>
<td>411</td>
</tr>
<tr>
<td><code>⇒ antisymp</code></td>
<td>400</td>
</tr>
<tr>
<td><code>⇒ antisymp</code></td>
<td>458</td>
</tr>
<tr>
<td><code>dy less</code></td>
<td>431</td>
</tr>
<tr>
<td><code>induced order</code></td>
<td>414</td>
</tr>
<tr>
<td><code>⊂ antisymp</code></td>
<td>430</td>
</tr>
<tr>
<td><code>R less</code></td>
<td>430</td>
</tr>
<tr>
<td><code>R less</code></td>
<td>511</td>
</tr>
<tr>
<td><code>R ≤ antisymp</code></td>
<td>430</td>
</tr>
<tr>
<td><code>R ≤ antisymp</code></td>
<td>511</td>
</tr>
<tr>
<td><code>≤ antisymp</code></td>
<td>448</td>
</tr>
<tr>
<td><code>≤ antisymp</code></td>
<td>462</td>
</tr>
<tr>
<td><code>Z ≤ antisymp</code></td>
<td>455</td>
</tr>
<tr>
<td><code>Z ≤ antisymp</code></td>
<td>477</td>
</tr>
<tr>
<td><code>Antisymmetric</code></td>
<td>398</td>
</tr>
<tr>
<td><code>any</code></td>
<td>64</td>
</tr>
<tr>
<td><code>any</code></td>
<td>64</td>
</tr>
<tr>
<td><code>any substitch tt</code></td>
<td>154</td>
</tr>
<tr>
<td><code>any substitch tt</code></td>
<td>154</td>
</tr>
<tr>
<td><code>any substitch tt</code></td>
<td>154</td>
</tr>
<tr>
<td><code>T Any</code></td>
<td>153</td>
</tr>
<tr>
<td><code>term any</code></td>
<td>115</td>
</tr>
<tr>
<td><code>type any</code></td>
<td>117</td>
</tr>
<tr>
<td><code>app any</code></td>
<td>20</td>
</tr>
<tr>
<td><code>App any</code></td>
<td>81</td>
</tr>
<tr>
<td><code>app arg rule</code></td>
<td>178</td>
</tr>
<tr>
<td><code>APP C</code></td>
<td>178</td>
</tr>
<tr>
<td><code>if app conv</code></td>
<td>190</td>
</tr>
<tr>
<td><code>D App</code></td>
<td>82</td>
</tr>
<tr>
<td><code>dest app</code></td>
<td>83</td>
</tr>
<tr>
<td><code>app fun rule</code></td>
<td>178</td>
</tr>
<tr>
<td><code>fun if conv</code></td>
<td>178</td>
</tr>
<tr>
<td><code>is app</code></td>
<td>92</td>
</tr>
<tr>
<td><code>list mk app mk</code></td>
<td>97</td>
</tr>
<tr>
<td><code>app mk</code></td>
<td>101</td>
</tr>
<tr>
<td><code>KIM K</code></td>
<td>131</td>
</tr>
<tr>
<td><code>AppRule mk</code></td>
<td>196</td>
</tr>
<tr>
<td><code>strip</code></td>
<td>111</td>
</tr>
<tr>
<td><code>time</code></td>
<td>56</td>
</tr>
<tr>
<td><code>open append</code></td>
<td>43</td>
</tr>
<tr>
<td><code>apply tactic</code></td>
<td>243</td>
</tr>
<tr>
<td><code>HT AqTm</code></td>
<td>72</td>
</tr>
<tr>
<td><code>HT AqTy</code></td>
<td>72</td>
</tr>
<tr>
<td><code>all ∃ arb elim</code></td>
<td>175</td>
</tr>
<tr>
<td><code>∀ arb elim</code></td>
<td>230</td>
</tr>
<tr>
<td><code>Arbitrary</code></td>
<td>407</td>
</tr>
<tr>
<td><code>dy arch thm</code></td>
<td>391</td>
</tr>
<tr>
<td><code>area_of</code></td>
<td>16</td>
</tr>
<tr>
<td><code>app arg rule</code></td>
<td>178</td>
</tr>
<tr>
<td><code>lin arith</code></td>
<td>467</td>
</tr>
<tr>
<td><code>R lin arith</code></td>
<td>498</td>
</tr>
<tr>
<td><code>Z lin arith</code></td>
<td>482</td>
</tr>
<tr>
<td><code>lin arith contr</code></td>
<td>468</td>
</tr>
<tr>
<td><code>lin arith</code></td>
<td>466</td>
</tr>
</tbody>
</table>
KEYWORD INDEX

\[ \begin{array}{llll}
\text{R. lin} & \text{arith_prove_tac} & 506 & \text{get. asm} \\
\text{lin} & \text{arith_rule} & 469 & \text{asm_inst_term_rule} \\
\text{R. lin} & \text{arith_rule} & 506 & \text{asm_inst_type_rule} \\
\text{Z. lin} & \text{arith_rule} & 483 & \text{asm_intro} \\
\text{lin} & \text{arith_tac} & 469 & \text{ASM_PROP_EQ_T} \\
\text{lin} & \text{arith_tac} & 470 & \text{asm_prove_tac} \\
\text{lin} & \text{arith_tac} & 484 & \text{asm_rewrite_rule} \\
\text{lin} & \text{arith_tac} & 470 & \text{once. asm_rewrite_rule} \\
\text{R. lin} & \text{arith} & 498 & \text{pure. asm_rewrite_rule} \\
\text{Z. lin} & \text{arith} & 481 & \text{pure_once. asm_rewrite_rule} \\
\text{Linear} & \text{Arithmetic} & 465 & \text{asm_rewrite_tac} \\
\text{Linear} & \text{ArithmeticTools} & 468 & \text{asm_rewrite_tac} \\
\text{get_type} & \text{arity} & 143 & \text{asm_rewrite_tac} \\
\text{array} & 39 & \text{pure_once. asm_rewrite_tac} \\
\text{array} & 41 & \text{asm_rewrite_thm_tac} \\
\text{Dynamic} & \text{Array} & 41 & \text{asm_rewrite_thm_tac} \\
\text{DYNAMIC} & \text{ARRAY} & 44 & \text{asm_rewrite_thm_tac} \\
\text{PP} & \text{Array} & 39 & \text{asm_rule} \\
\text{Sparse} & \text{Array} & 39 & \text{AsmRule} \\
\text{Sparse} & \text{ARRAY} & 41 & \text{asm_rule} \\
\text{inv_rel} & \text{arrow_thm} & 397 & \text{prove. asm_rule} \\
\text{inv_rel} & \text{arrow_thm} & 493 & \text{asm_rule} \\
\text{save} & \text{as} & 53 & \text{asm_rule} \\
\text{conv} & \text{ascii} & 12 & \text{ALL_VAR_ELIM. ASM_T} \\
\text{ascii_order} & 36 & \text{DROP. ASM_T} \\
\text{asm_anter_tac} & 254 & \text{GET. ASM_T} \\
\text{all} & \text{asm_anter_tac} & 251 & \text{GET_NTH. ASM_T} \\
\text{list} & \text{asm_anter_tac} & 274 & \text{LIST_DROP. ASM_T} \\
\text{LIST_SWAP} & \text{ASM_CONCL} & 276 & \text{LIST_DROP_NTH. ASM_T} \\
\text{LIST_SWAP_NTH} & \text{ASM_CONCL} & 276 & \text{LIST_GET. ASM_T} \\
\text{SWAP} & \text{ASM_CONCL} & 296 & \text{LIST_SPEC. ASM_T} \\
\text{SWAP_NTH} & \text{ASM_CONCL} & 296 & \text{LIST_SPEC. ASM_T} \\
\text{list_swap} & \text{asm_concl_tac} & 276 & \text{LIST_SPEC_NTH. ASM_T} \\
\text{list_swap_nth} & \text{asm_concl_tac} & 276 & \text{POP. ASM_T} \\
\text{swap} & \text{asm_concl_tac} & 295 & \text{SPEC. ASM_T} \\
\text{swap_nth} & \text{asm_concl_tac} & 295 & \text{SPEC_NTH. ASM_T} \\
\text{strip} & \text{asm_conv} & 293 & \text{TOP. ASM_T} \\
\text{asm_elim} & 179 & \text{VAR_ELIM. ASM_T} \\
\text{ASM_FC_T} & 266 & \text{VAR_ELIM_NTH. ASM_T} \\
\text{ALL} & \text{ASM_FC_T} & 266 & \text{ALL_VAR_ELIM. ASM_T1} \\
\text{ASM_FC_T1} & 267 & \text{asm_tac} \\
\text{ALL} & \text{ASM_FC_T1} & 267 & \text{asm_tac} \\
\text{asm_fc_tac} & 265 & \text{check. asm_tac} \\
\text{asm_fc_tac} & 265 & \text{eq_sym. asm_tac} \\
\text{eq_sym_nth} & \text{asm_tac} & 262 & \text{eq_sym_nth. asm_tac} \\
\text{ALL} & \text{ASM_FORWARD} & 266 & \text{list_scale_nth asm_tac} \\
\text{CHAIN_T} & 266 & \text{list_scale_nth asm_tac} \\
\text{ALL} & \text{ASM_FORWARD} & 266 & \text{list_spec asm_tac} \\
\text{CHAIN_T} & 266 & \text{list_spec nth asm_tac} \\
\text{ALL} & \text{ASM_FORWARD} & 267 & \text{scale nth asm_tac} \\
\text{CHAIN_T1} & 267 & \text{scale nth asm_tac} \\
\text{ALL} & \text{ASM_FORWARD} & 267 & \text{spec nth asm_tac} \\
\text{CHAIN_T1} & 267 & \text{spec nth asm_tac} \\
\text{asm_forward_chain} & \text{tac} & 265 & \text{step strip. asm_tac} \\
\text{asm_forward_chain} & \text{tac} & 265 & \text{strip. asm_tac} \\
\text{all} & \text{asm_forward_chain} & 265 & \text{var elim nth. asm_tac} \\
\text{tac} & 265 & \text{var elim nth. asm_tac} \\
\end{array} \]
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_less_sup</td>
<td>431</td>
</tr>
<tr>
<td>R_less_sup</td>
<td>509</td>
</tr>
<tr>
<td>R_sup_eq</td>
<td>432</td>
</tr>
<tr>
<td>R_sup_eq</td>
<td>509</td>
</tr>
<tr>
<td>R_sup_less</td>
<td>432</td>
</tr>
<tr>
<td>R_sup_less</td>
<td>509</td>
</tr>
<tr>
<td>R_c sup</td>
<td>432</td>
</tr>
<tr>
<td>R_c sup</td>
<td>509</td>
</tr>
<tr>
<td>R_c ≤ sup</td>
<td>432</td>
</tr>
<tr>
<td>R_c ≤ sup</td>
<td>509</td>
</tr>
<tr>
<td>R ≤ sup</td>
<td>431</td>
</tr>
<tr>
<td>R ≤ sup</td>
<td>509</td>
</tr>
<tr>
<td>cuts_unbounded</td>
<td>135</td>
</tr>
<tr>
<td>induced_order</td>
<td>414</td>
</tr>
<tr>
<td>_unbounded</td>
<td>415</td>
</tr>
<tr>
<td>Unbounded</td>
<td>430</td>
</tr>
<tr>
<td>Unbounded</td>
<td>511</td>
</tr>
<tr>
<td>Schroeder</td>
<td>398</td>
</tr>
<tr>
<td>Schroeder</td>
<td>398</td>
</tr>
<tr>
<td>bin bool</td>
<td>83</td>
</tr>
<tr>
<td>dest</td>
<td>84</td>
</tr>
<tr>
<td>is</td>
<td>93</td>
</tr>
<tr>
<td>list_mk</td>
<td>98</td>
</tr>
<tr>
<td>mk</td>
<td>102</td>
</tr>
<tr>
<td>strip</td>
<td>112</td>
</tr>
<tr>
<td>bin rel</td>
<td>493</td>
</tr>
<tr>
<td>bin rel/ext clauses</td>
<td>385</td>
</tr>
<tr>
<td>bin rel/ext clauses</td>
<td>492</td>
</tr>
<tr>
<td>bin rel/insert_thm</td>
<td>386</td>
</tr>
<tr>
<td>BinRelTheory</td>
<td>492</td>
</tr>
<tr>
<td>bin rel/Θ_universe</td>
<td>386</td>
</tr>
<tr>
<td>Binder</td>
<td>71</td>
</tr>
<tr>
<td>BINDER_C</td>
<td>180</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>205</td>
</tr>
<tr>
<td>declare</td>
<td>124</td>
</tr>
<tr>
<td>dest</td>
<td>83</td>
</tr>
<tr>
<td>dest_simple</td>
<td>87</td>
</tr>
<tr>
<td>HT</td>
<td>72</td>
</tr>
<tr>
<td>is</td>
<td>93</td>
</tr>
<tr>
<td>is_simple</td>
<td>95</td>
</tr>
<tr>
<td>list_mk</td>
<td>98</td>
</tr>
<tr>
<td>mk</td>
<td>101</td>
</tr>
<tr>
<td>mk_simple</td>
<td>105</td>
</tr>
<tr>
<td>strip</td>
<td>112</td>
</tr>
<tr>
<td>strip_simple</td>
<td>112</td>
</tr>
<tr>
<td>get</td>
<td>127</td>
</tr>
<tr>
<td>HT</td>
<td>72</td>
</tr>
<tr>
<td>BOOL</td>
<td>83</td>
</tr>
<tr>
<td>bool_cases_axiom</td>
<td>400</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
KEYWORD INDEX

-compactification cache 132
-get_compactification cache 132
-set_compactification cache 132
-get_cache_theories 171
-set_cache_theories 171
-force_get_cache_theory 171
-get_cache_theory 171
-cached_labelled 167
-product_rule 167
-cache_unlabelled 168
-EVERY CAN 185
-FC CAN 187
-FC\Rightarrow CAN 189
-FIRST CAN 186
-FORWARD_CHAIN CAN 187
-CHAIN\Rightarrow CAN 189
-THEN_LIST CAN 218
-R_eq conv 504
-R_less conv 504
-R_\le conv 504
-N_eq conv 473
-Z_eq conv 483
-Z_\le conv 483
-R_frac conv 441
-R over conv 441
-\forall conv 224
-R_frac conv 505
-R over conv 507
-R_times conv 510
-current_ad rw canon 352
-f_rewrite canon 223
-fail canon 186
-fail_with canon 186
-THEN_TRY C 219
-TOP_MAP C 219
-TRY C 219
-R_ANF C 502
-R_EVAL C 504
-\lambda C 239
-Z_ANF C 482
-EXTEND_PC C 342
-EXTEND_PCS C 342
-MERGE_PCS C 348
-REPEAT C 202
-SUB C 217
-clear \_compactification 132
-get \_compactification 132
-set \_compactification 132
-get \_cache_theories 171
-set \_cache_theories 171
-force_get \_cache_theory 171
-get \_cache_theory 171
-cached_labelled \_product_rule 167
-cache_unlabelled \_product_rule 168
-EVERY CAN 185
-FC CAN 187
-FC\Rightarrow CAN 189
-FIRST CAN 186
-FORWARD_CHAIN CAN 187
-CHAIN\Rightarrow CAN 189
-THEN_LIST CAN 218
-R_eq conv 504
-R_less conv 504
-R_\le conv 504
-N_eq conv 473
-Z_eq conv 483
-Z_\le conv 483
-R_frac conv 441
-R over conv 441
-\forall conv 224
-R_frac conv 505
-R over conv 507
-R_times conv 510
-current_ad_rw CANON 173
-f_rewrite canon 352
-fail canon 186
-fail_with canon 186

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>forward, chain_canon1</td>
<td>188</td>
<td>check_asm_tac 258</td>
</tr>
<tr>
<td>forward, chain_rule</td>
<td>189</td>
<td>CHECKPOINT 134</td>
</tr>
<tr>
<td>ALL_ASM</td>
<td>chain_rule</td>
<td>134</td>
</tr>
<tr>
<td>FORWARD</td>
<td>CHAIN_T</td>
<td>266</td>
</tr>
<tr>
<td>ALL_FORWARD</td>
<td>CHAIN_T</td>
<td>266</td>
</tr>
<tr>
<td>ASM_FORWARD</td>
<td>CHAIN_T</td>
<td>266</td>
</tr>
<tr>
<td>FORWARD</td>
<td>CHAIN_T</td>
<td>266</td>
</tr>
<tr>
<td>ALL_ASM</td>
<td>CHAIN_T1</td>
<td>267</td>
</tr>
<tr>
<td>FORWARD</td>
<td>CHAIN_T1</td>
<td>267</td>
</tr>
<tr>
<td>all_asm_forward</td>
<td>chain_tac</td>
<td>265</td>
</tr>
<tr>
<td>all_forward</td>
<td>chain_tac</td>
<td>265</td>
</tr>
<tr>
<td>back</td>
<td>chain_tac</td>
<td>255</td>
</tr>
<tr>
<td>forward_back</td>
<td>chain_tac</td>
<td>255</td>
</tr>
<tr>
<td>FORWARD</td>
<td>CHAIN_T1⇔CAN</td>
<td>189</td>
</tr>
<tr>
<td>forward</td>
<td>chain⇔canon</td>
<td>190</td>
</tr>
<tr>
<td>all</td>
<td>asm</td>
<td>398</td>
</tr>
<tr>
<td>before_kernel_state</td>
<td>change</td>
<td>135</td>
</tr>
<tr>
<td>pp</td>
<td>change_error_message</td>
<td>19</td>
</tr>
<tr>
<td>KERNEL_STATE</td>
<td>change</td>
<td>133</td>
</tr>
<tr>
<td>on_kernel_state</td>
<td>change</td>
<td>148</td>
</tr>
<tr>
<td></td>
<td>CHANGED_C</td>
<td>180</td>
</tr>
<tr>
<td></td>
<td>CHANGED_T</td>
<td>257</td>
</tr>
<tr>
<td></td>
<td>Char</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>CHAR</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>Char</td>
<td>388</td>
</tr>
<tr>
<td></td>
<td>char</td>
<td>485</td>
</tr>
<tr>
<td></td>
<td>Abs</td>
<td>388</td>
</tr>
<tr>
<td></td>
<td>char_conv</td>
<td>180</td>
</tr>
<tr>
<td></td>
<td>KI</td>
<td>131</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td>char_def</td>
<td>388</td>
</tr>
<tr>
<td></td>
<td>char_def</td>
<td>485</td>
</tr>
<tr>
<td></td>
<td>abs_char_rep_char_def</td>
<td>388</td>
</tr>
<tr>
<td></td>
<td>abs_char_rep_char_def</td>
<td>485</td>
</tr>
<tr>
<td></td>
<td>dest_char</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>char_eq_conv</td>
<td>463</td>
</tr>
<tr>
<td></td>
<td>find_char</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>HT</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>is</td>
<td>93</td>
</tr>
<tr>
<td></td>
<td>is_special</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>mk</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>Rep</td>
<td>388</td>
</tr>
<tr>
<td></td>
<td>abs_char_rep_char_def</td>
<td>388</td>
</tr>
<tr>
<td></td>
<td>abs_char_rep_char_def</td>
<td>485</td>
</tr>
<tr>
<td></td>
<td>is</td>
<td>388</td>
</tr>
<tr>
<td></td>
<td>is_char_def</td>
<td>485</td>
</tr>
<tr>
<td></td>
<td>SymUnknown</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>Sym</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>CharacterUtilities</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>get_use_extended_chars_flag</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>use_extended_chars</td>
<td>58</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
<table>
<thead>
<tr>
<th>CONV</th>
<th>121 pure_rewrite</th>
<th>conv</th>
<th>203</th>
</tr>
</thead>
<tbody>
<tr>
<td>all_simple_β</td>
<td>364 refl</td>
<td>conv</td>
<td>201</td>
</tr>
<tr>
<td>all_β</td>
<td>174 rewrite</td>
<td>conv</td>
<td>203</td>
</tr>
<tr>
<td>all_β</td>
<td>176 set_pr</td>
<td>conv</td>
<td>359</td>
</tr>
<tr>
<td>all_β</td>
<td>177 sets_eq</td>
<td>conv</td>
<td>487</td>
</tr>
<tr>
<td>all_β</td>
<td>317 sets_simple_β</td>
<td>conv</td>
<td>488</td>
</tr>
<tr>
<td>app_if</td>
<td>178 simple_eq_match</td>
<td>conv</td>
<td>205</td>
</tr>
<tr>
<td>conv_ascii</td>
<td>12 simple_ho_eq_match</td>
<td>conv</td>
<td>206</td>
</tr>
<tr>
<td>basic_prove</td>
<td>379 simple_one_point</td>
<td>conv</td>
<td>337</td>
</tr>
<tr>
<td>basic_prove_β</td>
<td>160 simple_ ∈_comp</td>
<td>conv</td>
<td>488</td>
</tr>
<tr>
<td>char</td>
<td>180 simple_ ≠_in</td>
<td>conv</td>
<td>285</td>
</tr>
<tr>
<td>char_eq</td>
<td>463 simple_ ∨<em>¬</em>¬</td>
<td>conv</td>
<td>337</td>
</tr>
<tr>
<td>cnf</td>
<td>318 simple_ ∨_¬</td>
<td>conv</td>
<td>209</td>
</tr>
<tr>
<td>Comp</td>
<td>462 simple_¬<em>const</em>¬_¬</td>
<td>conv</td>
<td>337</td>
</tr>
<tr>
<td>current_ad_cs_β</td>
<td>353 simple_¬_equation</td>
<td>conv</td>
<td>163</td>
</tr>
<tr>
<td>current_ad_pr</td>
<td>340 simple_¬<em>∧</em></td>
<td>conv</td>
<td>164</td>
</tr>
<tr>
<td>current_ad_se</td>
<td>352 simple_¬<em>∨</em></td>
<td>conv</td>
<td>165</td>
</tr>
<tr>
<td>current_ad_sl</td>
<td>352 simple_¬<em>∨</em></td>
<td>conv</td>
<td>210</td>
</tr>
<tr>
<td>div</td>
<td>465 simple_¬<em>¬</em>¬<em>¬</em>¬_¬</td>
<td>conv</td>
<td>210</td>
</tr>
<tr>
<td>dnf</td>
<td>319 simple_¬<em>¬</em>¬<em>¬</em>¬_¬</td>
<td>conv</td>
<td>288</td>
</tr>
<tr>
<td>eq_match</td>
<td>183 simple_α_</td>
<td>conv</td>
<td>211</td>
</tr>
<tr>
<td>eq_sym</td>
<td>184 simple_β_</td>
<td>conv</td>
<td>212</td>
</tr>
<tr>
<td>eqn_ext</td>
<td>341 simple_β_η_</td>
<td>conv</td>
<td>212</td>
</tr>
<tr>
<td>convextended</td>
<td>12 simple_β_η_ norm</td>
<td>conv</td>
<td>212</td>
</tr>
<tr>
<td>fail</td>
<td>186 sort</td>
<td>conv</td>
<td>322</td>
</tr>
<tr>
<td>fail_with</td>
<td>186 split_∧_</td>
<td>conv</td>
<td>337</td>
</tr>
<tr>
<td>find_in_∧</td>
<td>335 string</td>
<td>conv</td>
<td>214</td>
</tr>
<tr>
<td>float</td>
<td>500 string_eq</td>
<td>conv</td>
<td>464</td>
</tr>
<tr>
<td>get_pr</td>
<td>359 strip_asm</td>
<td>conv</td>
<td>293</td>
</tr>
<tr>
<td>greater</td>
<td>465 strip_concl</td>
<td>conv</td>
<td>293</td>
</tr>
<tr>
<td>id</td>
<td>190 subst</td>
<td>conv</td>
<td>215</td>
</tr>
<tr>
<td>if_app</td>
<td>190 subst</td>
<td>conv</td>
<td>218</td>
</tr>
<tr>
<td>KIChar</td>
<td>131 taut</td>
<td>conv</td>
<td>259</td>
</tr>
<tr>
<td>KIPlus</td>
<td>131 CONV_THEN</td>
<td></td>
<td>297</td>
</tr>
<tr>
<td>KIRef</td>
<td>131</td>
<td></td>
<td>260</td>
</tr>
<tr>
<td>KIString</td>
<td>131</td>
<td></td>
<td>260</td>
</tr>
<tr>
<td>KISuc</td>
<td>131</td>
<td></td>
<td>443</td>
</tr>
<tr>
<td>less</td>
<td>465</td>
<td></td>
<td>506</td>
</tr>
<tr>
<td>let</td>
<td>193</td>
<td></td>
<td>443</td>
</tr>
<tr>
<td>make_≤</td>
<td>470</td>
<td></td>
<td>506</td>
</tr>
<tr>
<td>minus</td>
<td>465</td>
<td></td>
<td>465</td>
</tr>
<tr>
<td>mod</td>
<td>465</td>
<td></td>
<td>487</td>
</tr>
<tr>
<td>modf</td>
<td>336</td>
<td></td>
<td>489</td>
</tr>
<tr>
<td>once_rewrite</td>
<td>203</td>
<td></td>
<td>489</td>
</tr>
<tr>
<td>pair_eq</td>
<td>314</td>
<td></td>
<td>487</td>
</tr>
<tr>
<td>plus</td>
<td>198</td>
<td></td>
<td>501</td>
</tr>
<tr>
<td>plus</td>
<td>465</td>
<td></td>
<td>502</td>
</tr>
<tr>
<td>plus1</td>
<td>461</td>
<td></td>
<td>508</td>
</tr>
<tr>
<td>poly</td>
<td>321</td>
<td></td>
<td>504</td>
</tr>
<tr>
<td>pp_set_eval_ad_sl</td>
<td>352</td>
<td></td>
<td>504</td>
</tr>
<tr>
<td>pp_set_eval_ad_sl</td>
<td>352</td>
<td></td>
<td>504</td>
</tr>
<tr>
<td>prim_rewrite</td>
<td>199</td>
<td></td>
<td>504</td>
</tr>
<tr>
<td>prim_suc</td>
<td>200</td>
<td></td>
<td>504</td>
</tr>
<tr>
<td>prop_simp</td>
<td>336</td>
<td></td>
<td>504</td>
</tr>
<tr>
<td>prove</td>
<td>354</td>
<td></td>
<td>504</td>
</tr>
<tr>
<td>prove_β</td>
<td>355</td>
<td></td>
<td>504</td>
</tr>
<tr>
<td>pure_once_rewrite</td>
<td>203</td>
<td></td>
<td>506</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_language</td>
<td>130</td>
</tr>
<tr>
<td>current_pc</td>
<td>345</td>
</tr>
<tr>
<td>current_terminators</td>
<td>127</td>
</tr>
<tr>
<td>current_theory_name</td>
<td>141</td>
</tr>
<tr>
<td>current_theory_status</td>
<td>141</td>
</tr>
<tr>
<td>curry</td>
<td>28</td>
</tr>
<tr>
<td>Curry</td>
<td>416</td>
</tr>
<tr>
<td>Curry</td>
<td>417</td>
</tr>
<tr>
<td>curry_def</td>
<td>417</td>
</tr>
<tr>
<td>curry_def</td>
<td>459</td>
</tr>
<tr>
<td>Cuts</td>
<td>411</td>
</tr>
<tr>
<td>Cuts</td>
<td>412</td>
</tr>
<tr>
<td>cuts_complete_thm</td>
<td>413</td>
</tr>
<tr>
<td>cuts_strict_linear</td>
<td>413</td>
</tr>
<tr>
<td>_order_thm</td>
<td>413</td>
</tr>
<tr>
<td>cuts_strict_partial</td>
<td>413</td>
</tr>
<tr>
<td>down_sets</td>
<td>413</td>
</tr>
<tr>
<td>_order_thm</td>
<td>413</td>
</tr>
<tr>
<td>cuts_thm</td>
<td>413</td>
</tr>
<tr>
<td>cuts_trich_thm</td>
<td>413</td>
</tr>
<tr>
<td>cuts_unbounded_above</td>
<td>414</td>
</tr>
<tr>
<td>_thm</td>
<td>414</td>
</tr>
<tr>
<td>cuts_unbounded_below</td>
<td>414</td>
</tr>
<tr>
<td>eqn</td>
<td>341</td>
</tr>
<tr>
<td>cthm_eqn</td>
<td>181</td>
</tr>
<tr>
<td>EQN</td>
<td>338</td>
</tr>
<tr>
<td>get_rw_eqn</td>
<td>361</td>
</tr>
<tr>
<td>get_sc_eqn</td>
<td>361</td>
</tr>
<tr>
<td>get_st_eqn</td>
<td>362</td>
</tr>
<tr>
<td>set_rw_eqn</td>
<td>361</td>
</tr>
<tr>
<td>set_sc_eqn</td>
<td>361</td>
</tr>
<tr>
<td>set_st_eqn</td>
<td>362</td>
</tr>
<tr>
<td>simple_ho_thm_eqn</td>
<td>363</td>
</tr>
<tr>
<td>thm_eqn</td>
<td>364</td>
</tr>
<tr>
<td>DApp</td>
<td>82</td>
</tr>
<tr>
<td>DChar</td>
<td>82</td>
</tr>
<tr>
<td>DConst</td>
<td>82</td>
</tr>
<tr>
<td>DEmptyList</td>
<td>82</td>
</tr>
<tr>
<td>DEnumSet</td>
<td>82</td>
</tr>
<tr>
<td>DEq</td>
<td>82</td>
</tr>
<tr>
<td>DF</td>
<td>82</td>
</tr>
<tr>
<td>DLet</td>
<td>82</td>
</tr>
<tr>
<td>DList</td>
<td>82</td>
</tr>
<tr>
<td>DPair</td>
<td>82</td>
</tr>
<tr>
<td>DString</td>
<td>82</td>
</tr>
<tr>
<td>DSetComp</td>
<td>82</td>
</tr>
<tr>
<td>DVar</td>
<td>82</td>
</tr>
<tr>
<td>DATA</td>
<td>123</td>
</tr>
<tr>
<td>DATA</td>
<td>149</td>
</tr>
<tr>
<td>Datum</td>
<td>133</td>
</tr>
<tr>
<td>datum</td>
<td>149</td>
</tr>
<tr>
<td>USER</td>
<td>121</td>
</tr>
<tr>
<td>USER</td>
<td>325</td>
</tr>
<tr>
<td>RES</td>
<td>141</td>
</tr>
<tr>
<td>DEC_TYPE</td>
<td>31</td>
</tr>
<tr>
<td>decimal</td>
<td>31</td>
</tr>
<tr>
<td>declare alias</td>
<td>124</td>
</tr>
<tr>
<td>declare binder</td>
<td>124</td>
</tr>
<tr>
<td>declare const</td>
<td>124</td>
</tr>
<tr>
<td>_language</td>
<td>124</td>
</tr>
<tr>
<td>declare_left_infix</td>
<td>125</td>
</tr>
<tr>
<td>declare_nonfix</td>
<td>125</td>
</tr>
<tr>
<td>declare_postfix</td>
<td>125</td>
</tr>
<tr>
<td>declare_prefix</td>
<td>126</td>
</tr>
<tr>
<td>declare_right_infix</td>
<td>125</td>
</tr>
<tr>
<td>declare_terminator</td>
<td>126</td>
</tr>
<tr>
<td>declare_type_abbrev</td>
<td>126</td>
</tr>
<tr>
<td>abs_char_rep_char</td>
<td>388</td>
</tr>
<tr>
<td>append</td>
<td>402</td>
</tr>
<tr>
<td>append</td>
<td>484</td>
</tr>
<tr>
<td>at</td>
<td>493</td>
</tr>
<tr>
<td>char</td>
<td>388</td>
</tr>
<tr>
<td>char</td>
<td>485</td>
</tr>
<tr>
<td>comb_i</td>
<td>389</td>
</tr>
<tr>
<td>comb_i</td>
<td>490</td>
</tr>
<tr>
<td>comb_k</td>
<td>389</td>
</tr>
<tr>
<td>comb_k</td>
<td>490</td>
</tr>
<tr>
<td>comb_s</td>
<td>390</td>
</tr>
<tr>
<td>comb_s</td>
<td>490</td>
</tr>
<tr>
<td>combine</td>
<td>402</td>
</tr>
<tr>
<td>combine</td>
<td>484</td>
</tr>
<tr>
<td>complement</td>
<td>421</td>
</tr>
<tr>
<td>complement</td>
<td>486</td>
</tr>
<tr>
<td>cond</td>
<td>407</td>
</tr>
<tr>
<td>cond</td>
<td>458</td>
</tr>
<tr>
<td>consistent</td>
<td>155</td>
</tr>
<tr>
<td>const_spec</td>
<td>155</td>
</tr>
<tr>
<td>curry</td>
<td>417</td>
</tr>
<tr>
<td>curry</td>
<td>459</td>
</tr>
<tr>
<td>distinct</td>
<td>495</td>
</tr>
<tr>
<td>div</td>
<td>446</td>
</tr>
<tr>
<td>div</td>
<td>461</td>
</tr>
<tr>
<td>dom</td>
<td>492</td>
</tr>
<tr>
<td>dot_dot</td>
<td>495</td>
</tr>
<tr>
<td>elems</td>
<td>495</td>
</tr>
<tr>
<td>enumerate</td>
<td>495</td>
</tr>
<tr>
<td>extract</td>
<td>495</td>
</tr>
<tr>
<td>f</td>
<td>404</td>
</tr>
<tr>
<td>f</td>
<td>458</td>
</tr>
<tr>
<td>finite</td>
<td>494</td>
</tr>
<tr>
<td>float</td>
<td>495</td>
</tr>
<tr>
<td>fold</td>
<td>403</td>
</tr>
<tr>
<td>fold</td>
<td>484</td>
</tr>
<tr>
<td>front</td>
<td>495</td>
</tr>
<tr>
<td>functional</td>
<td>492</td>
</tr>
<tr>
<td>graph</td>
<td>492</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
| dest_ε   | 91 | dest_λ   | 91 |
| dest_N   | 91 | dest_NIR | 499 |
| dest_Z^abs | 475 | dest_Z^div | 475 |
| dest_Z^greater | 475 | dest_Z^less | 475 |
| dest_Z^minus | 475 | dest_Z^mod | 475 |
| dest_Z^plus | 475 | dest_Z^signed_int | 475 |
| dest_Z^subtract | 475 | dest_Z^times | 475 |
| dest_Z^≤ | 475 | dest_Z^≥ | 475 |
| DF       | 82 | diag_line | 62 |
| raw_line | 69 | diag_line | 62 |
| list_line | 66 | diag_line | 66 |
| list_raw_line | 69 | raw_line | 70 |
| kernel_interface | diagnostics | 144 |
| RW       | diagnostics | 58 |
| E_DICT   | 33 |
| e_dict_of_oed | dict | 35 |
| get_axiom | dict | 140 |
| get_defn | dict | 142 |
| get_thm | dict | 143 |
| initial_e | dict | 34 |
| initial_oed | dict | 35 |
| initial_s | dict | 32 |
| OE_DICT  | 35 |
| e_dict_of_oed_dict | 35 |
| S_DICT   | 19 |
| Efficient Dictionary | 33 |
| Simple Dictionary | 32 |
| set_dif_clauses | 422 |
| set_dif_clauses | 486 |
| set_dif_def | 421 |
| set_dif_def | 486 |
| diff | 21 |
| term | 115 |
| all | 20 |
| discard_tac | 260 |
| disch_rule | 182 |
| Distinct | 418 |
| Distinct | 419 |
| all | 20 |
| distinct_def | 495 |
| finite | 394 |
| finite | 394 |
| distinct_elems_thm | 394 |
| distinct_elems_thm | 494 |
| distinct_length ≤ thm | 394 |
| distinct_length ≤ thm | 494 |
| distinct_size_length ≤ thm | 394 |
| distinct_size_length ≤ thm | 494 |
| distinct_size_length ≤ thm | 394 |
KEYWORD INDEX

<table>
<thead>
<tr>
<th>KEYWORD</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>down_sets_dense_thm</td>
<td>413</td>
</tr>
<tr>
<td>down_sets_less_thm</td>
<td>414</td>
</tr>
<tr>
<td>downset_cut_thm</td>
<td>413</td>
</tr>
<tr>
<td>drop</td>
<td>21</td>
</tr>
<tr>
<td>DROP_ASM_T</td>
<td>260</td>
</tr>
<tr>
<td>DROP_ASM_T</td>
<td>274</td>
</tr>
<tr>
<td>DROP_FILTER</td>
<td>260</td>
</tr>
<tr>
<td>ASMS_T</td>
<td>261</td>
</tr>
<tr>
<td>drop_main_goal</td>
<td>243</td>
</tr>
<tr>
<td>DROP_NTH_ASM_T</td>
<td>261</td>
</tr>
<tr>
<td>LIST.</td>
<td>274</td>
</tr>
<tr>
<td>DT</td>
<td>82</td>
</tr>
<tr>
<td>DuplicateTheory</td>
<td>133</td>
</tr>
<tr>
<td>duplicate_theory</td>
<td>140</td>
</tr>
<tr>
<td>dy_arch_thm</td>
<td>391</td>
</tr>
<tr>
<td>dy_balance_thm1</td>
<td>391</td>
</tr>
<tr>
<td>dy_balance_thm2</td>
<td>391</td>
</tr>
<tr>
<td>dy_exp</td>
<td>390</td>
</tr>
<tr>
<td>dy_exp_clauses</td>
<td>391</td>
</tr>
<tr>
<td>dy_left_dense_thm</td>
<td>392</td>
</tr>
<tr>
<td>dy_less</td>
<td>390</td>
</tr>
<tr>
<td>dy_less_antisym_thm</td>
<td>391</td>
</tr>
<tr>
<td>dy_less_dense_thm</td>
<td>392</td>
</tr>
<tr>
<td>dy_less_irrefl_thm</td>
<td>391</td>
</tr>
<tr>
<td>dy_less_order_lemmas_thm</td>
<td>430</td>
</tr>
<tr>
<td>dy_less_trans_thm</td>
<td>391</td>
</tr>
<tr>
<td>dy_less_trich_thm</td>
<td>391</td>
</tr>
<tr>
<td>dy_one</td>
<td>390</td>
</tr>
<tr>
<td>dy_right_dense_thm</td>
<td>391</td>
</tr>
<tr>
<td>dy_times</td>
<td>390</td>
</tr>
<tr>
<td>dy_times_assoc_thm</td>
<td>391</td>
</tr>
<tr>
<td>dy_times_comm_thm</td>
<td>391</td>
</tr>
<tr>
<td>dy_times_mono_thm</td>
<td>391</td>
</tr>
<tr>
<td>dy_times_mono_thm1</td>
<td>391</td>
</tr>
<tr>
<td>dy_times_mono_thm2</td>
<td>391</td>
</tr>
<tr>
<td>dy_times_mono_thm2</td>
<td>391</td>
</tr>
<tr>
<td>dy_times_order_thm</td>
<td>391</td>
</tr>
<tr>
<td>dy_times_unit_clauses</td>
<td>391</td>
</tr>
<tr>
<td>dy_times_unit_thm</td>
<td>391</td>
</tr>
<tr>
<td>DYADIC</td>
<td>390</td>
</tr>
<tr>
<td>DynamicArray</td>
<td>41</td>
</tr>
<tr>
<td>DYNAMIC_ARRAY</td>
<td>41</td>
</tr>
<tr>
<td>DΩ</td>
<td>82</td>
</tr>
<tr>
<td>D⇒</td>
<td>82</td>
</tr>
<tr>
<td>D∧</td>
<td>82</td>
</tr>
<tr>
<td>D∨</td>
<td>82</td>
</tr>
<tr>
<td>D≡</td>
<td>82</td>
</tr>
<tr>
<td>D⇌</td>
<td>82</td>
</tr>
<tr>
<td>D∃</td>
<td>82</td>
</tr>
<tr>
<td>D≡₁</td>
<td>82</td>
</tr>
<tr>
<td>D∈</td>
<td>82</td>
</tr>
<tr>
<td>D∈₁</td>
<td>82</td>
</tr>
<tr>
<td>D⇒</td>
<td>82</td>
</tr>
<tr>
<td>D⇒</td>
<td>82</td>
</tr>
<tr>
<td>D⇒</td>
<td>82</td>
</tr>
<tr>
<td>D⇒</td>
<td>82</td>
</tr>
<tr>
<td>D⇒</td>
<td>82</td>
</tr>
<tr>
<td>D⇒</td>
<td>82</td>
</tr>
<tr>
<td>D⇒</td>
<td>82</td>
</tr>
<tr>
<td>D⇒</td>
<td>82</td>
</tr>
<tr>
<td>e_delete</td>
<td>33</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\Rightarrow)</td>
<td>Elim</td>
<td>228</td>
</tr>
<tr>
<td>(\forall)</td>
<td>Elim</td>
<td>231</td>
</tr>
<tr>
<td>(\forall_{arb})</td>
<td>Elim</td>
<td>230</td>
</tr>
<tr>
<td>(\exists)</td>
<td>Elim</td>
<td>233</td>
</tr>
<tr>
<td>(\exists_{j})</td>
<td>Elim</td>
<td>191</td>
</tr>
<tr>
<td>if, else_elim</td>
<td>HT</td>
<td>72</td>
</tr>
<tr>
<td>TooManyRead</td>
<td>Empties</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>Empty</td>
<td>421</td>
</tr>
<tr>
<td></td>
<td>empty_finite_thm</td>
<td>394</td>
</tr>
<tr>
<td></td>
<td>dest</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td>list</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>list_net</td>
<td>119</td>
</tr>
<tr>
<td></td>
<td>net</td>
<td>119</td>
</tr>
<tr>
<td></td>
<td>o_e</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>o_e_key</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>32</td>
</tr>
<tr>
<td>get_rd</td>
<td>entry</td>
<td>359</td>
</tr>
<tr>
<td>set_rd</td>
<td>entry</td>
<td>359</td>
</tr>
<tr>
<td>(\in)</td>
<td>enum_set_conv</td>
<td>489</td>
</tr>
<tr>
<td></td>
<td>EnumSet</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td>enum_set</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>enum_set_rule</td>
<td>103</td>
</tr>
<tr>
<td></td>
<td>Enumerate</td>
<td>418</td>
</tr>
<tr>
<td></td>
<td>enumerate_def</td>
<td>495</td>
</tr>
<tr>
<td>READER</td>
<td>Env</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>Eos</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>Eq</td>
<td>468</td>
</tr>
<tr>
<td>('prop)</td>
<td>eq</td>
<td>313</td>
</tr>
<tr>
<td>R_less</td>
<td>eq_0_thm</td>
<td>441</td>
</tr>
<tr>
<td>R_less</td>
<td>eq_0_thm</td>
<td>508</td>
</tr>
<tr>
<td>R_over</td>
<td>eq_0_thm</td>
<td>442</td>
</tr>
<tr>
<td>R_over</td>
<td>eq_0_thm</td>
<td>505</td>
</tr>
<tr>
<td>R_times</td>
<td>eq_0_thm</td>
<td>440</td>
</tr>
<tr>
<td>R_times</td>
<td>eq_0_thm</td>
<td>510</td>
</tr>
<tr>
<td>Z_abs</td>
<td>eq_0_thm</td>
<td>456</td>
</tr>
<tr>
<td>Z_abs</td>
<td>eq_0_thm</td>
<td>477</td>
</tr>
<tr>
<td>Z_times</td>
<td>eq_0_thm</td>
<td>455</td>
</tr>
<tr>
<td>Z_times</td>
<td>eq_0_thm</td>
<td>477</td>
</tr>
<tr>
<td>R_sup</td>
<td>eq_bc_thm</td>
<td>432</td>
</tr>
<tr>
<td>R_sup</td>
<td>eq_bc_thm</td>
<td>509</td>
</tr>
<tr>
<td>R</td>
<td>eq_cancel_conv</td>
<td>504</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
KEYWORD INDEX

extend 32
get_use  extended_chars_flag  65
use  extended_chars  58
conv  extended  12
ExtendedIO  43
merge_pc  341
fields  347
use  file_non_stop_mode  58
use  file  68
use  file1  68
load  files  51
filter  21
DROP_FILTER_ASM_T  261
GET_FILTER_ASM_T  267
FinSt theory  494
find  21
find_char  63
find in ∧ conv  335
find_name  63
find_thm  153
find_thm  154
gens  find_thm_in_theories  154
Finite  393
finite_def  494
finite_distinct_elems  394
finite_distinct_elems  494
finite_induction_tac  495
finite_induction_thm  394
finite_induction_thm  494
empty  finite_thm  394
empty  finite_thm  494
singleton ∪ finite_thm  394
singleton ∪ finite_thm  494
∪ finite_thm  394
∪ finite_thm  494
∪ finite_thm  494
∪ finite_thm  495
∪ finite_thm  494
∪ finite_thm  494
∪ finite_thm  495
finite ⊆ well_founded  395
finite ⊆ well_founded  494
FIRST  264
FIRST_C  187
FIRST_CAN  186
FIRST_THEN_C  335
FIRST TTCL  264
FIXITY  71
get_ fixity  127
LS  Fixity  77
get_ flag  48

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_FILTER_ASMS</td>
<td>270</td>
</tr>
<tr>
<td>_T</td>
<td>270</td>
</tr>
<tr>
<td>get_ficity</td>
<td>127</td>
</tr>
<tr>
<td>get_flag</td>
<td>48</td>
</tr>
<tr>
<td>get_flags</td>
<td>48</td>
</tr>
<tr>
<td>get_HOL_any</td>
<td>64</td>
</tr>
<tr>
<td>get_init_funs</td>
<td>50</td>
</tr>
<tr>
<td>get_int_control</td>
<td>48</td>
</tr>
<tr>
<td>get_int_controls</td>
<td>48</td>
</tr>
<tr>
<td>get_key</td>
<td>33</td>
</tr>
<tr>
<td>get_language</td>
<td>128</td>
</tr>
<tr>
<td>get_left_infixes</td>
<td>128</td>
</tr>
<tr>
<td>get_line_length</td>
<td>69</td>
</tr>
<tr>
<td>get_message</td>
<td>18</td>
</tr>
<tr>
<td>get_message_text</td>
<td>18</td>
</tr>
<tr>
<td>get_ML_any</td>
<td>64</td>
</tr>
<tr>
<td>get_ML_string</td>
<td>65</td>
</tr>
<tr>
<td>get_mmp_rule</td>
<td>358</td>
</tr>
<tr>
<td>get_nd_entry</td>
<td>359</td>
</tr>
<tr>
<td>get_nonfixes</td>
<td>128</td>
</tr>
<tr>
<td>GET_NTH_ASM_T</td>
<td>270</td>
</tr>
<tr>
<td>GET_NTH_ASM_T</td>
<td>275</td>
</tr>
<tr>
<td>drop_main</td>
<td>goal</td>
</tr>
<tr>
<td>print</td>
<td>goal</td>
</tr>
<tr>
<td>print_current</td>
<td>goal</td>
</tr>
<tr>
<td>push</td>
<td>goal</td>
</tr>
<tr>
<td>set</td>
<td>goal</td>
</tr>
<tr>
<td>set_labelled</td>
<td>goal</td>
</tr>
<tr>
<td>GOAL</td>
<td>251</td>
</tr>
<tr>
<td>GOAL_STATE</td>
<td>242</td>
</tr>
<tr>
<td>get_parents</td>
<td>142</td>
</tr>
<tr>
<td>get_pcs</td>
<td>345</td>
</tr>
<tr>
<td>get_percent_name</td>
<td>65</td>
</tr>
<tr>
<td>get_postfixes</td>
<td>128</td>
</tr>
<tr>
<td>get_pr_conv</td>
<td>359</td>
</tr>
<tr>
<td>get_pr_conv1</td>
<td>359</td>
</tr>
<tr>
<td>get_pr_tac</td>
<td>360</td>
</tr>
<tr>
<td>get_pr_tac1</td>
<td>360</td>
</tr>
<tr>
<td>get_prefixes</td>
<td>128</td>
</tr>
<tr>
<td>get_primed_string</td>
<td>65</td>
</tr>
<tr>
<td>get_proved_conjectures</td>
<td>151</td>
</tr>
<tr>
<td>get_right_infixes</td>
<td>128</td>
</tr>
<tr>
<td>get_round_braces</td>
<td>64</td>
</tr>
<tr>
<td>get_rw_canons</td>
<td>360</td>
</tr>
<tr>
<td>get_rw_eqm_rule</td>
<td>361</td>
</tr>
<tr>
<td>get_rw_eqm_czt</td>
<td>361</td>
</tr>
<tr>
<td>get_save_funs</td>
<td>50</td>
</tr>
<tr>
<td>get_sc_eqm_czt</td>
<td>361</td>
</tr>
<tr>
<td>get_shell_var</td>
<td>50</td>
</tr>
<tr>
<td>get_spec</td>
<td>157</td>
</tr>
<tr>
<td>get_st_eqm_czt</td>
<td>362</td>
</tr>
<tr>
<td>get_stack_pcs</td>
<td>346</td>
</tr>
<tr>
<td>get_stats</td>
<td>55</td>
</tr>
<tr>
<td>get_string_control</td>
<td>48</td>
</tr>
<tr>
<td>get_string_controls</td>
<td>48</td>
</tr>
<tr>
<td>get_terminators</td>
<td>128</td>
</tr>
<tr>
<td>get_thm</td>
<td>143</td>
</tr>
<tr>
<td>get_thm_dict</td>
<td>143</td>
</tr>
<tr>
<td>get_thms</td>
<td>143</td>
</tr>
<tr>
<td>get_type_abbrev</td>
<td>128</td>
</tr>
<tr>
<td>get_type_abbrevs</td>
<td>129</td>
</tr>
<tr>
<td>get_type_arity</td>
<td>143</td>
</tr>
<tr>
<td>get_type_abbrev</td>
<td>128</td>
</tr>
<tr>
<td>get_type_abbrevs</td>
<td>129</td>
</tr>
<tr>
<td>get_type_arity</td>
<td>143</td>
</tr>
<tr>
<td>get_type_info</td>
<td>129</td>
</tr>
<tr>
<td>get_type_keys</td>
<td>143</td>
</tr>
<tr>
<td>get_type_theory</td>
<td>144</td>
</tr>
<tr>
<td>get_types</td>
<td>143</td>
</tr>
<tr>
<td>get_undeclared_aliases</td>
<td>129</td>
</tr>
<tr>
<td>get_undeclared</td>
<td></td>
</tr>
<tr>
<td>get_undeclared_terminators</td>
<td>129</td>
</tr>
<tr>
<td>get_undeclared_abbrevs</td>
<td>129</td>
</tr>
<tr>
<td>get_unproved</td>
<td></td>
</tr>
<tr>
<td>get_use_extended</td>
<td></td>
</tr>
<tr>
<td>get_user_datum</td>
<td>144</td>
</tr>
<tr>
<td>get_valid_cache</td>
<td></td>
</tr>
<tr>
<td>get_variant_suffix</td>
<td>92</td>
</tr>
<tr>
<td>get∃_cd_thms</td>
<td>362</td>
</tr>
<tr>
<td>get∃_vs_thms</td>
<td>363</td>
</tr>
<tr>
<td>GOAL</td>
<td>251</td>
</tr>
<tr>
<td>Graph</td>
<td>383</td>
</tr>
<tr>
<td>Graph</td>
<td>384</td>
</tr>
<tr>
<td>graph_at_thm</td>
<td>397</td>
</tr>
<tr>
<td>graph_at_thm</td>
<td>493</td>
</tr>
<tr>
<td>graph_def</td>
<td>492</td>
</tr>
<tr>
<td>greater_conv</td>
<td>465</td>
</tr>
<tr>
<td>greater_def</td>
<td>506</td>
</tr>
<tr>
<td>greater_def</td>
<td>446</td>
</tr>
<tr>
<td>greater_def</td>
<td>461</td>
</tr>
<tr>
<td>greater_def</td>
<td>509</td>
</tr>
<tr>
<td>greater_def</td>
<td>475</td>
</tr>
<tr>
<td>greater_def</td>
<td>463</td>
</tr>
<tr>
<td>greater_def</td>
<td>499</td>
</tr>
<tr>
<td>greater_def</td>
<td>475</td>
</tr>
<tr>
<td>greater_def</td>
<td>463</td>
</tr>
<tr>
<td>greater_def</td>
<td>500</td>
</tr>
<tr>
<td>greater_def</td>
<td>476</td>
</tr>
</tbody>
</table>
induction_tac 478
induction_tac 478
induction_tac 479
induction_tac 479
induction_tac 269
induction_thm 447
induction_thm 462
induction_thm 448
cov induction_thm 478
cov induction_thm 478
≤ induction_thm 479
≤ induction_thm 479
N induction_thm 479
gen induction_thm 269
thm 447
thm 462
cov induction_thm 448
cov induction_thm 462
cov induction_thm 478
cov induction_thm 478
≤ induction_thm 479
≤ induction_thm 479
N induction_thm 479
N induction_thm 479
KERNEL info 131
KERNEL info 131
on kernel inference 131
infinity_axiom 400
infinity_axiom 458
Inf x 71
declare infix 125
declare infix 125
declare left infix 125
declare right infix 125
get left infixes 128
get left infixes 128
get alias info 127
get alias info 127
get const info 127
get const info 127
g et th eory info 143
g et type info 129
pp database info 50
pp database info 50
pp reset database info 51
pp reset database info 51
THM INFO TEST 153
THM INFO 153
ICL DATABASE INFO TYPE 50
init 50
init 50
init fun 50
init 76
pp error init 19
init stats 55
InitTheory 458
initial 338
initial e dict 34
initial e dict 34
initial oe dict 35
initial oe dict 35
initial rw canon 192
initial s dict 32
initialisation 50
InfLists 418
inf_lists_def 495
Injective 383
Injective 384
injective_def 492
QUANT C 335
input 43
INPUT 72
SymEndOf Input 59
insert 22
Insert 420
Insert 421
insert def 421
insert def 486
insert thm 492
inst 92
inst term rule 192
inst term rule 179
InstTermRule 131
inst type 92
inst type rule 193
inst type rule 179
InstTypeRule 131
instance 95
instance 43
int_control 48
int_control 48
int_control 49
int_control 49
int controls 48
int controls 49
int controls 49
int 475
integer of int 42
integer of int 476
integer of int 476
integer of integer 42
integer order 36
integer 31
UD Int 121
int3 67
INTEGER 42
Integer 42
integer 42
integer of int 42
integer of string 42
integer order 42
integer 42
interface diagnostics 144
Interface 131
interface 148
interval 22
intro 175
intro 176

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>asm</td>
<td>intro</td>
<td>179</td>
<td></td>
</tr>
<tr>
<td>if</td>
<td>intro</td>
<td>191</td>
<td></td>
</tr>
<tr>
<td>list_simple_∀</td>
<td>intro</td>
<td>194</td>
<td></td>
</tr>
<tr>
<td>list_simple_∃</td>
<td>intro</td>
<td>194</td>
<td></td>
</tr>
<tr>
<td>list_∧</td>
<td>intro</td>
<td>195</td>
<td></td>
</tr>
<tr>
<td>list_∨</td>
<td>intro</td>
<td>195</td>
<td></td>
</tr>
<tr>
<td>⋁_ intro_rule</td>
<td></td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>simple_∀</td>
<td>intro</td>
<td>209</td>
<td></td>
</tr>
<tr>
<td>simple_∃</td>
<td>intro</td>
<td>210</td>
<td></td>
</tr>
<tr>
<td>simple_∃_1</td>
<td>intro</td>
<td>211</td>
<td></td>
</tr>
<tr>
<td>∃_ intro_thm</td>
<td></td>
<td>233</td>
<td></td>
</tr>
<tr>
<td>∃_ intro_thm</td>
<td></td>
<td>408</td>
<td></td>
</tr>
<tr>
<td>∀_ intro</td>
<td></td>
<td>220</td>
<td></td>
</tr>
<tr>
<td>⇒_ intro</td>
<td></td>
<td>220</td>
<td></td>
</tr>
<tr>
<td>⇐_ intro</td>
<td></td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>∧_ intro</td>
<td></td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>ν_ intro</td>
<td></td>
<td>224</td>
<td></td>
</tr>
<tr>
<td>ν_ intro</td>
<td></td>
<td>225</td>
<td></td>
</tr>
<tr>
<td>ν_ intro</td>
<td></td>
<td>225</td>
<td></td>
</tr>
<tr>
<td>ν_ intro</td>
<td></td>
<td>228</td>
<td></td>
</tr>
<tr>
<td>∃_ intro</td>
<td></td>
<td>231</td>
<td></td>
</tr>
<tr>
<td>∃_ intro</td>
<td></td>
<td>234</td>
<td></td>
</tr>
<tr>
<td>∃_ intro</td>
<td></td>
<td>236</td>
<td></td>
</tr>
<tr>
<td>inv_list_order</td>
<td></td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>InvRel</td>
<td></td>
<td>383</td>
<td></td>
</tr>
<tr>
<td>InvRel</td>
<td></td>
<td>384</td>
<td></td>
</tr>
<tr>
<td>inv_rel_def</td>
<td></td>
<td>492</td>
<td></td>
</tr>
<tr>
<td>inv_rel_thm</td>
<td></td>
<td>385</td>
<td></td>
</tr>
<tr>
<td>inv_rel_thm</td>
<td></td>
<td>492</td>
<td></td>
</tr>
<tr>
<td>inv_rel_thm</td>
<td></td>
<td>397</td>
<td></td>
</tr>
<tr>
<td>inv_rel_thm</td>
<td></td>
<td>493</td>
<td></td>
</tr>
<tr>
<td>Invalid</td>
<td></td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>R_opah</td>
<td></td>
<td>436</td>
<td></td>
</tr>
<tr>
<td>R_opah</td>
<td></td>
<td>507</td>
<td></td>
</tr>
<tr>
<td>R_opah</td>
<td></td>
<td>436</td>
<td></td>
</tr>
<tr>
<td>R_opah</td>
<td></td>
<td>507</td>
<td></td>
</tr>
<tr>
<td>R_opah</td>
<td></td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>Basic</td>
<td></td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>Extended</td>
<td></td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>dγ_less</td>
<td></td>
<td>411</td>
<td></td>
</tr>
<tr>
<td>induced_order</td>
<td></td>
<td>391</td>
<td></td>
</tr>
<tr>
<td>less</td>
<td></td>
<td>414</td>
<td></td>
</tr>
<tr>
<td>less</td>
<td></td>
<td>448</td>
<td></td>
</tr>
<tr>
<td>less</td>
<td></td>
<td>462</td>
<td></td>
</tr>
<tr>
<td>&lt;_ intro</td>
<td></td>
<td>413</td>
<td></td>
</tr>
<tr>
<td>R_less</td>
<td></td>
<td>430</td>
<td></td>
</tr>
<tr>
<td>R_less</td>
<td></td>
<td>511</td>
<td></td>
</tr>
<tr>
<td>Z_less</td>
<td></td>
<td>455</td>
<td></td>
</tr>
<tr>
<td>Z_less</td>
<td></td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>is_all_decimal</td>
<td></td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>is_app</td>
<td></td>
<td>92</td>
<td></td>
</tr>
<tr>
<td>is_bin_op</td>
<td></td>
<td>93</td>
<td></td>
</tr>
<tr>
<td>is_binder</td>
<td></td>
<td>93</td>
<td></td>
</tr>
<tr>
<td>is_char</td>
<td></td>
<td>93</td>
<td></td>
</tr>
<tr>
<td>IsCharRep</td>
<td></td>
<td>388</td>
<td></td>
</tr>
<tr>
<td>is_char_rep_def</td>
<td></td>
<td>388</td>
<td></td>
</tr>
<tr>
<td>is_char_rep_def</td>
<td></td>
<td>485</td>
<td></td>
</tr>
<tr>
<td>is_constr</td>
<td></td>
<td>93</td>
<td></td>
</tr>
<tr>
<td>is_ctor</td>
<td></td>
<td>93</td>
<td></td>
</tr>
<tr>
<td>is_div</td>
<td></td>
<td>463</td>
<td></td>
</tr>
<tr>
<td>is_empty_list</td>
<td></td>
<td>93</td>
<td></td>
</tr>
<tr>
<td>is_enum_set</td>
<td></td>
<td>93</td>
<td></td>
</tr>
<tr>
<td>is_eq</td>
<td></td>
<td>93</td>
<td></td>
</tr>
<tr>
<td>is_f</td>
<td></td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>is_float</td>
<td></td>
<td>93</td>
<td></td>
</tr>
<tr>
<td>is_free</td>
<td></td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>is_free_in</td>
<td></td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>is_great</td>
<td></td>
<td>463</td>
<td></td>
</tr>
<tr>
<td>is_if</td>
<td></td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>IsL</td>
<td></td>
<td>425</td>
<td></td>
</tr>
<tr>
<td>IsL</td>
<td></td>
<td>426</td>
<td></td>
</tr>
<tr>
<td>is_less</td>
<td></td>
<td>463</td>
<td></td>
</tr>
<tr>
<td>is_list</td>
<td></td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>IsListRep</td>
<td></td>
<td>401</td>
<td></td>
</tr>
<tr>
<td>IsListRep</td>
<td></td>
<td>402</td>
<td></td>
</tr>
<tr>
<td>is_list_rep_def</td>
<td></td>
<td>402</td>
<td></td>
</tr>
<tr>
<td>is_list_rep_def</td>
<td></td>
<td>484</td>
<td></td>
</tr>
<tr>
<td>is_min</td>
<td></td>
<td>463</td>
<td></td>
</tr>
<tr>
<td>is_mod</td>
<td></td>
<td>463</td>
<td></td>
</tr>
<tr>
<td>is_mon</td>
<td></td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>is.Nil</td>
<td></td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>is_nil</td>
<td></td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>IsOneRep</td>
<td></td>
<td>410</td>
<td></td>
</tr>
<tr>
<td>is_one_rep_def</td>
<td></td>
<td>410</td>
<td></td>
</tr>
<tr>
<td>is_one_rep_def</td>
<td></td>
<td>490</td>
<td></td>
</tr>
<tr>
<td>is_pair</td>
<td></td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>IsPairRep</td>
<td></td>
<td>416</td>
<td></td>
</tr>
<tr>
<td>is_pair_rep_def</td>
<td></td>
<td>416</td>
<td></td>
</tr>
<tr>
<td>is_pair_rep_def</td>
<td></td>
<td>459</td>
<td></td>
</tr>
<tr>
<td>is_plus</td>
<td></td>
<td>463</td>
<td></td>
</tr>
<tr>
<td>is_proved_conjecture</td>
<td></td>
<td>151</td>
<td></td>
</tr>
<tr>
<td>is_quant</td>
<td></td>
<td>335</td>
<td></td>
</tr>
<tr>
<td>IsR</td>
<td></td>
<td>425</td>
<td></td>
</tr>
<tr>
<td>IsR</td>
<td></td>
<td>426</td>
<td></td>
</tr>
<tr>
<td>is_same_symbol</td>
<td></td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>is_set_comp</td>
<td></td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>IsSetRep</td>
<td></td>
<td>420</td>
<td></td>
</tr>
<tr>
<td>IsSetRep</td>
<td></td>
<td>421</td>
<td></td>
</tr>
<tr>
<td>is_set_rep_def</td>
<td></td>
<td>421</td>
<td></td>
</tr>
<tr>
<td>is_set_rep_def</td>
<td></td>
<td>486</td>
<td></td>
</tr>
<tr>
<td>is_simple_binder</td>
<td></td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>is_simple_∀</td>
<td></td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>is_simple_∃</td>
<td></td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>is_simple_∃_1</td>
<td></td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>is_simple_∃</td>
<td></td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>is_simple_λ</td>
<td></td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>is_special_char</td>
<td></td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>is_string</td>
<td></td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>IsSumRep</td>
<td></td>
<td>425</td>
<td></td>
</tr>
<tr>
<td>is_sum_rep_def</td>
<td></td>
<td>425</td>
<td></td>
</tr>
<tr>
<td>is_sum_rep_def</td>
<td></td>
<td>490</td>
<td></td>
</tr>
<tr>
<td>is_t</td>
<td></td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>is_term_in</td>
<td></td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>Term</td>
<td>Page</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_term_out</td>
<td>43</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_theory_ancestor</td>
<td>144</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_times</td>
<td>463</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_type_abbrev</td>
<td>129</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_type_instance</td>
<td>95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_var</td>
<td>96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_vartype</td>
<td>95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_white</td>
<td>66</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_∅</td>
<td>96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_⇔</td>
<td>96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_→</td>
<td>96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_ificate</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_ificate</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_R_abs</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_R_frac</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_R_greater</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_R_less</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_R_max</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_R_min</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_R_minus</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_R_over</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_R_plus</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_R_recip</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is_Rep</td>
<td>427</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is_Rep</td>
<td>428</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_R_rep_consistent</td>
<td>430</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_R_subtract</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_R_times</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_R_≤</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_R_≥</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_R_N_exp</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_R_Z_exp</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_∧</td>
<td>96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_∨</td>
<td>96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_¬</td>
<td>96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_⇒</td>
<td>96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_∀</td>
<td>96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_∃</td>
<td>97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_∃!</td>
<td>97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_×_type</td>
<td>97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_≤</td>
<td>463</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_≥</td>
<td>463</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_e</td>
<td>97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_λ</td>
<td>97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_N</td>
<td>97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is_N_Rep</td>
<td>445</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is_N_Rep</td>
<td>446</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is_N_rep_def</td>
<td>446</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is_N_rep_def</td>
<td>461</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_NIR</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_Z_abs</td>
<td>476</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_Z_div</td>
<td>476</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_Z_greater</td>
<td>476</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_Z_less</td>
<td>476</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_Z_minus</td>
<td>476</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_Z_mod</td>
<td>476</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_Z_plus</td>
<td>476</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is_Z_Rep</td>
<td>450</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is_Z_Rep</td>
<td>451</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_Z_rep_consistent</td>
<td>452</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is_Z_Rep_def</td>
<td>475</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_Z_signed_int</td>
<td>476</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_Z_subtract</td>
<td>476</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_Z_times</td>
<td>476</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_≤</td>
<td>476</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_≥</td>
<td>476</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iter</td>
<td>393</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iterate</td>
<td>29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>389</td>
<td></td>
<td></td>
</tr>
<tr>
<td>comb</td>
<td>389</td>
<td></td>
<td></td>
</tr>
<tr>
<td>comb</td>
<td>490</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>389</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kernel_group_thm</td>
<td>436</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kernel_group_thm</td>
<td>507</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KERNEL</td>
<td>131</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INERENCE</td>
<td>131</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KERNEL</td>
<td>131</td>
<td></td>
<td></td>
</tr>
<tr>
<td>interface</td>
<td>144</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIAGNOSTIC</td>
<td>148</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kernel_state_change</td>
<td>135</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kernel_state_change</td>
<td>148</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>33</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>141</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>143</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KIAsmRule</td>
<td>131</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KICharConv</td>
<td>131</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KIEqSymRule</td>
<td>131</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KIEqTransRule</td>
<td>131</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KIInstTermRule</td>
<td>131</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KIInstTypeRule</td>
<td>131</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KIListSimple\Elim</td>
<td>131</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KIMkAppRule</td>
<td>131</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KIPlusConv</td>
<td>131</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KIRetConv</td>
<td>131</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
<table>
<thead>
<tr>
<th>Function/Operator</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>R. minus_frac_less_frac_thm</code></td>
<td>505</td>
</tr>
<tr>
<td><code>R. frac. minus_frac_thm</code></td>
<td>441</td>
</tr>
<tr>
<td><code>R. frac. minus_frac_thm</code></td>
<td>441</td>
</tr>
<tr>
<td><code>R. frac. less. minus_frac_thm</code></td>
<td>505</td>
</tr>
<tr>
<td><code>R. frac. less. minus_frac_thm</code></td>
<td>505</td>
</tr>
<tr>
<td><code>R. frac. minus_frac_thm1</code></td>
<td>442</td>
</tr>
<tr>
<td><code>R. frac. minus_frac_thm1</code></td>
<td>442</td>
</tr>
<tr>
<td><code>is</code> minus</td>
<td>463</td>
</tr>
<tr>
<td><code>is_Z</code> minus</td>
<td>500</td>
</tr>
<tr>
<td><code>Z. minus_less_thm</code></td>
<td>476</td>
</tr>
<tr>
<td><code>Z. minus_less_thm</code></td>
<td>477</td>
</tr>
<tr>
<td><code>mk</code> minus</td>
<td>464</td>
</tr>
<tr>
<td><code>mk_R. minus</code></td>
<td>501</td>
</tr>
<tr>
<td><code>mk_Z. minus</code></td>
<td>476</td>
</tr>
<tr>
<td><code>R. minus_recip_thm</code></td>
<td>442</td>
</tr>
<tr>
<td><code>R. minus_recip_thm</code></td>
<td>505</td>
</tr>
<tr>
<td><code>R. abs. minus_thm</code></td>
<td>442</td>
</tr>
<tr>
<td><code>R. abs. minus_thm</code></td>
<td>505</td>
</tr>
<tr>
<td><code>R. add_hom. minus_thm</code></td>
<td>435</td>
</tr>
<tr>
<td><code>R. add_hom. minus_thm</code></td>
<td>507</td>
</tr>
<tr>
<td><code>R. plus. minus_thm</code></td>
<td>433</td>
</tr>
<tr>
<td><code>R. plus. minus_thm</code></td>
<td>508</td>
</tr>
<tr>
<td><code>R. times. minus_thm</code></td>
<td>440</td>
</tr>
<tr>
<td><code>R. times. minus_thm</code></td>
<td>510</td>
</tr>
<tr>
<td><code>Z. minus_thm</code></td>
<td>454</td>
</tr>
<tr>
<td><code>Z. minus_thm</code></td>
<td>477</td>
</tr>
<tr>
<td><code>Z. abs. minus_thm</code></td>
<td>456</td>
</tr>
<tr>
<td><code>Z. abs. minus_thm</code></td>
<td>477</td>
</tr>
<tr>
<td><code>Z. plus. minus_thm</code></td>
<td>452</td>
</tr>
<tr>
<td><code>Z. plus. minus_thm</code></td>
<td>477</td>
</tr>
<tr>
<td><code>Z. times. minus_thm</code></td>
<td>453</td>
</tr>
<tr>
<td><code>Z. times. minus_thm</code></td>
<td>477</td>
</tr>
<tr>
<td><code>Z. &lt;. minus_thm</code></td>
<td>453</td>
</tr>
<tr>
<td><code>Z. &lt;. minus_thm</code></td>
<td>477</td>
</tr>
<tr>
<td><code>Z. N. abs. minus_thm</code></td>
<td>456</td>
</tr>
<tr>
<td><code>Z. N. abs. minus_thm</code></td>
<td>477</td>
</tr>
<tr>
<td><code>Z. N. minus_thm</code></td>
<td>454</td>
</tr>
<tr>
<td><code>Z. N. &lt;. minus_thm</code></td>
<td>477</td>
</tr>
<tr>
<td><code>ZR. minus_thm</code></td>
<td>444</td>
</tr>
<tr>
<td><code>ZR. minus_thm</code></td>
<td>511</td>
</tr>
<tr>
<td><code>ZR. minus_&lt;. thm</code></td>
<td>453</td>
</tr>
<tr>
<td><code>ZR. minus_&lt;. thm</code></td>
<td>477</td>
</tr>
<tr>
<td><code>Z. minus_N. &lt;=. thm</code></td>
<td>455</td>
</tr>
<tr>
<td><code>Z. minus_N. &lt;=. thm</code></td>
<td>477</td>
</tr>
<tr>
<td><code>Min_R</code></td>
<td>427</td>
</tr>
<tr>
<td><code>Min_R</code></td>
<td>430</td>
</tr>
<tr>
<td><code>Min_R. consistent</code></td>
<td>443</td>
</tr>
<tr>
<td><code>MiscTheory</code></td>
<td>458</td>
</tr>
<tr>
<td><code>mk_app</code></td>
<td>101</td>
</tr>
<tr>
<td><code>mk_app</code></td>
<td>97</td>
</tr>
<tr>
<td><code>mk_app_rule</code></td>
<td>196</td>
</tr>
<tr>
<td><code>KI MkAppRule</code></td>
<td>131</td>
</tr>
<tr>
<td><code>mk_bin_op</code></td>
<td>102</td>
</tr>
<tr>
<td><code>mk_bin_op</code></td>
<td>98</td>
</tr>
<tr>
<td><code>mk_binder</code></td>
<td>101</td>
</tr>
<tr>
<td><code>mk_binder</code></td>
<td>98</td>
</tr>
</tbody>
</table>
KEYWORD INDEX

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>mk_R__ &amp; 501</td>
<td></td>
</tr>
<tr>
<td>mk_R_N_exp &amp; 501</td>
<td></td>
</tr>
<tr>
<td>mk_R_Z_exp &amp; 501</td>
<td></td>
</tr>
<tr>
<td>mk__ &amp; 108</td>
<td></td>
</tr>
<tr>
<td>list_ mk__ &amp; 99</td>
<td></td>
</tr>
<tr>
<td>list_ mk__ &amp; 109</td>
<td></td>
</tr>
<tr>
<td>list_ mk__ &amp; 109</td>
<td></td>
</tr>
<tr>
<td>list_ mk__ &amp; 109</td>
<td></td>
</tr>
<tr>
<td>list_ mk__ &amp; 100</td>
<td></td>
</tr>
<tr>
<td>list_ mk__ &amp; 100</td>
<td></td>
</tr>
<tr>
<td>list_ mk__ &amp; 100</td>
<td></td>
</tr>
<tr>
<td>list_ mk__ &amp; 100</td>
<td></td>
</tr>
<tr>
<td>list_ mk__ &amp; 100</td>
<td></td>
</tr>
<tr>
<td>list_ mk__ &amp; 100</td>
<td></td>
</tr>
<tr>
<td>list_ mk__ &amp; 100</td>
<td></td>
</tr>
<tr>
<td>list_ mk__ &amp; 100</td>
<td></td>
</tr>
<tr>
<td>list_ mk__ &amp; 100</td>
<td></td>
</tr>
<tr>
<td>list_ mk__ &amp; 100</td>
<td></td>
</tr>
<tr>
<td>list_ mk__ &amp; 100</td>
<td></td>
</tr>
<tr>
<td>list_ mk__ &amp; 100</td>
<td></td>
</tr>
<tr>
<td>list_ mk__ &amp; 100</td>
<td></td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{R}_{\text{frac}, \text{cross}}$</td>
<td>$\text{mult}_\text{eq}.\text{thm}$</td>
</tr>
<tr>
<td>$\mathbb{R}_{\text{cross}}$</td>
<td>$\text{mult}_\text{less}.\text{thm}$</td>
</tr>
<tr>
<td>$\mathbb{R}_{\text{cross}}$</td>
<td>$\text{mult}_\text{less}.\text{thm}$</td>
</tr>
<tr>
<td>dest</td>
<td>$\text{multi} \to$</td>
</tr>
<tr>
<td>mk</td>
<td>$\text{multi} \to$</td>
</tr>
<tr>
<td>REPEAT</td>
<td>$\text{N _T}$</td>
</tr>
<tr>
<td>$\text{get_current_theory}$</td>
<td>$\text{name}$</td>
</tr>
<tr>
<td>$\text{get_percent}$</td>
<td>$\text{name}$</td>
</tr>
<tr>
<td>$\text{HT}$</td>
<td>$\text{Name}$</td>
</tr>
<tr>
<td>$\text{PRETTY}$</td>
<td>$\text{Name}$</td>
</tr>
<tr>
<td>$\text{add}$</td>
<td>$\text{named_quotation}$</td>
</tr>
<tr>
<td>$\text{look_up}$</td>
<td>$\text{named_reader}$</td>
</tr>
<tr>
<td>$\text{get_theory}$</td>
<td>$\text{Names}$</td>
</tr>
<tr>
<td>$\text{Pretty}$</td>
<td>$\text{theory}$</td>
</tr>
<tr>
<td>current_ad</td>
<td>$\text{nd_entry}$</td>
</tr>
<tr>
<td>current_ad</td>
<td>$\text{nd_net}$</td>
</tr>
<tr>
<td>$\text{pp_set_eval_ad}$</td>
<td>$\text{Net}$</td>
</tr>
<tr>
<td>$\text{pp_set_eval_ad}$</td>
<td>$\text{Net}$</td>
</tr>
<tr>
<td>LSAD</td>
<td>$\text{NestedStructure}$</td>
</tr>
<tr>
<td>$\text{current_ad_nd}$</td>
<td>$\text{net}$</td>
</tr>
<tr>
<td>$\text{current_ad_rw}$</td>
<td>$\text{net}$</td>
</tr>
<tr>
<td>empty</td>
<td>$\text{net}$</td>
</tr>
<tr>
<td>list</td>
<td>$\text{net}$</td>
</tr>
<tr>
<td>list</td>
<td>$\text{net_enter}$</td>
</tr>
<tr>
<td>$\text{make}$</td>
<td>$\text{net}$</td>
</tr>
<tr>
<td>$\text{pp_set_eval_ad}$</td>
<td>$\text{net}$</td>
</tr>
<tr>
<td>$\text{pp_set_eval_ad}$</td>
<td>$\text{net}$</td>
</tr>
<tr>
<td>NetTools</td>
<td>$\text{Net}$</td>
</tr>
<tr>
<td>NewAxiom</td>
<td>$\text{Net}$</td>
</tr>
<tr>
<td>new_axiom</td>
<td>$\text{Net}$</td>
</tr>
<tr>
<td>new_conjecture</td>
<td>$\text{Net}$</td>
</tr>
<tr>
<td>NewConst</td>
<td>$\text{Net}$</td>
</tr>
<tr>
<td>new_const</td>
<td>$\text{Net}$</td>
</tr>
<tr>
<td>NewDefn</td>
<td>$\text{Net}$</td>
</tr>
<tr>
<td>new_defn</td>
<td>$\text{Net}$</td>
</tr>
<tr>
<td>new_error_message</td>
<td>$\text{Net}$</td>
</tr>
<tr>
<td>new_flag</td>
<td>$\text{Net}$</td>
</tr>
<tr>
<td>new_init_fun</td>
<td>$\text{Net}$</td>
</tr>
<tr>
<td>new_int_control</td>
<td>$\text{Net}$</td>
</tr>
<tr>
<td>NewParent</td>
<td>$\text{Net}$</td>
</tr>
<tr>
<td>new_parent</td>
<td>$\text{Net}$</td>
</tr>
<tr>
<td>new_pc</td>
<td>$\text{Net}$</td>
</tr>
<tr>
<td>new_save_fun</td>
<td>$\text{Net}$</td>
</tr>
<tr>
<td>NewSpec</td>
<td>$\text{Net}$</td>
</tr>
<tr>
<td>new_spec</td>
<td>$\text{Net}$</td>
</tr>
<tr>
<td>new_string_control</td>
<td>$\text{Net}$</td>
</tr>
<tr>
<td>add</td>
<td>new_symbols</td>
</tr>
</tbody>
</table>

NewTheory | 133 |
new_theory | 146 |
NewType | 133 |
new_type | 147 |
NewTypeDefn | 133 |
new_type_defn | 147 |
look_at | next | 60 |
look_at | next | 60 |
skip_and_look_at | next | 328 |
basic_res | Nil | 19 |
Nil | 402 |
nil_cons_def | 402 |
nil_cons_def | 484 |
Nil | 22 |
nil | 22 |
rmf_conv | 336 |
no_order | 36 |
no_submatch_tt | 154 |
no_substring_tt | 154 |
no_subterm_tt | 154 |
norm_conv | 505 |
Nonfix | 125 |
norm_conv | 212 |
norm_poly | 128 |
normalise | 468 |
normalise_poly | 468 |
not | 28 |
nth | 25 |
Nth | 418 |
Nth | 419 |
list_swap | NTH_ASM_CONCL | 276 |
list_swap | NTH_ASM_CONCL | 296 |
swap | NTH_ASM_CONCL | 276 |
swap | NTH_ASM_CONCL | 295 |
DROP | NTH_ASM_T | 261 |
GET | NTH_ASM_T | 270 |
LIST | NTH_ASM_T | 274 |
LIST | NTH_ASM_T | 275 |
LIST_SPEC | NTH_ASM_T | 291 |
SPEC | NTH_ASM_T | 291 |
VAR_ELIM | NTH_ASM_T | 301 |
eq_sym | nth_asm_tac | 262 |
list_scale | nth_asm_tac | 471 |
list_spec | nth_asm_tac | 290 |
scale | nth_asm_tac | 471 |
spec | nth_asm_tac | 290 |
var_elim | nth_asm_tac | 301 |
ath_def | 495 |
HT | NumLit | 72 |
num_lit_of_string | 74 |
\(\mathbb{R}_{\text{frac}}\) | 383 |
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>o</td>
<td>389</td>
</tr>
<tr>
<td>o_assoc_thm</td>
<td>389</td>
</tr>
<tr>
<td>o_assoc_thm</td>
<td>490</td>
</tr>
<tr>
<td>o_def</td>
<td>389</td>
</tr>
<tr>
<td>o_def</td>
<td>490</td>
</tr>
<tr>
<td>o_i_thm</td>
<td>389</td>
</tr>
<tr>
<td>o_i_thm</td>
<td>490</td>
</tr>
<tr>
<td>r__r_def</td>
<td>492</td>
</tr>
<tr>
<td>R__R</td>
<td>383</td>
</tr>
<tr>
<td>R__R</td>
<td>384</td>
</tr>
<tr>
<td>THEN</td>
<td>37</td>
</tr>
<tr>
<td>oe_delete</td>
<td>35</td>
</tr>
<tr>
<td>OE_DICT</td>
<td>35</td>
</tr>
<tr>
<td>e_dict_of_initial</td>
<td>35</td>
</tr>
<tr>
<td>oe_dict</td>
<td>35</td>
</tr>
<tr>
<td>oe_dict</td>
<td>35</td>
</tr>
<tr>
<td>oe_enter</td>
<td>35</td>
</tr>
<tr>
<td>oe_extend</td>
<td>35</td>
</tr>
<tr>
<td>oe_flatten</td>
<td>35</td>
</tr>
<tr>
<td>oe_key_delete</td>
<td>35</td>
</tr>
<tr>
<td>oe_key_enter</td>
<td>35</td>
</tr>
<tr>
<td>oe_key_extend</td>
<td>35</td>
</tr>
<tr>
<td>oe_key_flatten</td>
<td>35</td>
</tr>
<tr>
<td>oe_key_lookup</td>
<td>35</td>
</tr>
<tr>
<td>oe_lookup</td>
<td>35</td>
</tr>
<tr>
<td>oe_merge</td>
<td>35</td>
</tr>
<tr>
<td>list_of_oe_merge</td>
<td>35</td>
</tr>
<tr>
<td>area_of</td>
<td>16</td>
</tr>
<tr>
<td>string_of_e_key</td>
<td>33</td>
</tr>
<tr>
<td>string_of_float</td>
<td>42</td>
</tr>
<tr>
<td>SymEnd_OffInput</td>
<td>59</td>
</tr>
<tr>
<td>integer_of_int</td>
<td>42</td>
</tr>
<tr>
<td>string_of_int</td>
<td>31</td>
</tr>
<tr>
<td>string_of_int3</td>
<td>67</td>
</tr>
<tr>
<td>int_of_integer</td>
<td>42</td>
</tr>
<tr>
<td>string_of_integer</td>
<td>42</td>
</tr>
<tr>
<td>e_dict_of_oe_dict</td>
<td>35</td>
</tr>
<tr>
<td>end_of_stream</td>
<td>43</td>
</tr>
<tr>
<td>integer_of_string</td>
<td>42</td>
</tr>
<tr>
<td>nat_of_string</td>
<td>31</td>
</tr>
<tr>
<td>natural_of_string</td>
<td>42</td>
</tr>
<tr>
<td>num_lit_of_string</td>
<td>74</td>
</tr>
<tr>
<td>string_of_term</td>
<td>111</td>
</tr>
<tr>
<td>string_of_thm</td>
<td>149</td>
</tr>
<tr>
<td>type_of</td>
<td>117</td>
</tr>
<tr>
<td>string_of_type</td>
<td>111</td>
</tr>
<tr>
<td>on_kernel_inference_change</td>
<td>131</td>
</tr>
<tr>
<td>on_kernel_state_change</td>
<td>148</td>
</tr>
<tr>
<td>pass_on</td>
<td>18</td>
</tr>
<tr>
<td>pure_once_asm_rewrite_rule</td>
<td>204</td>
</tr>
<tr>
<td>pure_once_asm_rewrite_rule</td>
<td>204</td>
</tr>
<tr>
<td>pure_once_asm_rewrite_tac</td>
<td>283</td>
</tr>
<tr>
<td>pure_once_asm_rewrite_tac</td>
<td>283</td>
</tr>
<tr>
<td>pure_once_asm_rewrite_thm_tac</td>
<td>283</td>
</tr>
<tr>
<td>pure_once_asm_rewrite_thm_tac</td>
<td>283</td>
</tr>
<tr>
<td>ONCE_MAP_C</td>
<td>196</td>
</tr>
<tr>
<td>ONCE_MAP_WARN_C</td>
<td>197</td>
</tr>
<tr>
<td>once_rewrite_conv</td>
<td>203</td>
</tr>
<tr>
<td>once_rewrite_conv</td>
<td>203</td>
</tr>
<tr>
<td>once_rewrite_rule</td>
<td>204</td>
</tr>
<tr>
<td>once_rewrite_rule</td>
<td>204</td>
</tr>
<tr>
<td>once_rewrite_tac</td>
<td>283</td>
</tr>
<tr>
<td>once_rewrite_tac</td>
<td>283</td>
</tr>
<tr>
<td>ONCE_rewrite_thm_tac</td>
<td>283</td>
</tr>
<tr>
<td>ONCE_rewrite_thm_tac</td>
<td>283</td>
</tr>
<tr>
<td>one</td>
<td>42</td>
</tr>
<tr>
<td>ONE</td>
<td>410</td>
</tr>
<tr>
<td>one</td>
<td>374</td>
</tr>
<tr>
<td>one</td>
<td>410</td>
</tr>
<tr>
<td>one_def</td>
<td>410</td>
</tr>
<tr>
<td>one_def</td>
<td>490</td>
</tr>
<tr>
<td>one_def1</td>
<td>458</td>
</tr>
<tr>
<td>one_def1</td>
<td>490</td>
</tr>
</tbody>
</table>
dy\_one| 390|
<p>|one_fns_thm| 410|
|one_fns_thm| 490|
|One_One| 404|
|One_One| 405|
|One_One| 405|
|one_one_def| 405|
|one_one_def| 458|
|one_one_plus1_thm| 447|
|one_one_plus1_thm| 462|
|one_one_thm| 409|
|one_one_thm| 459|
|one_one_thm| 436|
|one_thm| 507|
|NR_one_thm| 433|
|NR_one_thm| 508|
|NZ_one_thm| 453|
|NZ_one_thm| 477|
|one_one_plus1_thm| 447|
|one_one_plus1_thm| 462|
|simple_one_point_conv| 337|
|is_one_rep_def| 410|
|is_one_rep_def| 490|
|Is_OneRep| 410|
|One_Theory| 490|
|on_thm| 409|
|on_thm| 459|
|R_opah_one_thm| 436|
|R_opah_one_thm| 507|
|R_opah_one_thm| 507|
|NR_one_thm| 433|
|NR_one_thm| 508|
|NZ_one_thm| 453|
|NZ_one_thm| 477|
|onto_def| 405|
|onto_def| 405|
|onto_def| 458|</p>
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>R. opah_onto_thm</td>
<td>436</td>
</tr>
<tr>
<td>R. opah_onto_thm</td>
<td>507</td>
</tr>
<tr>
<td>bin bool. op</td>
<td>83</td>
</tr>
<tr>
<td>dest_bin. op</td>
<td>84</td>
</tr>
<tr>
<td>dest_mon. op</td>
<td>86</td>
</tr>
<tr>
<td>HTIn Op</td>
<td>72</td>
</tr>
<tr>
<td>HTPost Op</td>
<td>72</td>
</tr>
<tr>
<td>is_bin. op</td>
<td>93</td>
</tr>
<tr>
<td>is_mon. op</td>
<td>94</td>
</tr>
<tr>
<td>list.mk_bin. op</td>
<td>98</td>
</tr>
<tr>
<td>mk_bin. op</td>
<td>102</td>
</tr>
<tr>
<td>mk_mon. op</td>
<td>105</td>
</tr>
<tr>
<td>strip_bin. op</td>
<td>112</td>
</tr>
<tr>
<td>R. opah_complete_thm</td>
<td>439</td>
</tr>
<tr>
<td>R. opah_complete_thm</td>
<td>507</td>
</tr>
<tr>
<td>R. opah_dense_image_thm</td>
<td>436</td>
</tr>
<tr>
<td>R. opah_dense_image_thm</td>
<td>507</td>
</tr>
<tr>
<td>R. opah_eq_thm</td>
<td>439</td>
</tr>
<tr>
<td>R. opah_eq_thm</td>
<td>507</td>
</tr>
<tr>
<td>R. opah_extension_thm</td>
<td>439</td>
</tr>
<tr>
<td>StrictLinear Order</td>
<td>411</td>
</tr>
<tr>
<td>R. opah_extension_thm</td>
<td>507</td>
</tr>
<tr>
<td>StrictPartial Order</td>
<td>412</td>
</tr>
<tr>
<td>R. opah_extension_thm1</td>
<td>438</td>
</tr>
<tr>
<td>StrictPartial Order</td>
<td>411</td>
</tr>
<tr>
<td>R. opah_extension_thm1</td>
<td>507</td>
</tr>
<tr>
<td>R. opah_extension_thm2</td>
<td>438</td>
</tr>
<tr>
<td>R. opah_extension_thm2</td>
<td>507</td>
</tr>
<tr>
<td>R. opah_inverse_add_hom_thm</td>
<td>436</td>
</tr>
<tr>
<td>R. opah_inverse_add_hom_thm</td>
<td>507</td>
</tr>
<tr>
<td>R. opah_inverse_thm</td>
<td>436</td>
</tr>
<tr>
<td>R. opah_inverse_thm</td>
<td>507</td>
</tr>
<tr>
<td>R. opah_one_one_thm</td>
<td>436</td>
</tr>
<tr>
<td>R. opah_one_one_thm</td>
<td>507</td>
</tr>
<tr>
<td>R. opah_onto_thm</td>
<td>436</td>
</tr>
<tr>
<td>R. opah_order_thm</td>
<td>439</td>
</tr>
<tr>
<td>R. opah_order_thm</td>
<td>507</td>
</tr>
<tr>
<td>R. opah_ordered_thm</td>
<td>436</td>
</tr>
<tr>
<td>R. opah_order_thm</td>
<td>507</td>
</tr>
<tr>
<td>R. opah_strict_thm</td>
<td>507</td>
</tr>
<tr>
<td>R. opah_strict_thm</td>
<td>436</td>
</tr>
<tr>
<td>R. opah_thm</td>
<td>436</td>
</tr>
<tr>
<td>open_append</td>
<td>43</td>
</tr>
<tr>
<td>open_append</td>
<td>43</td>
</tr>
<tr>
<td>OpenTheory</td>
<td>133</td>
</tr>
<tr>
<td>open_theory</td>
<td>148</td>
</tr>
<tr>
<td>pair. ops_def</td>
<td>416</td>
</tr>
<tr>
<td>pair. ops_def</td>
<td>459</td>
</tr>
<tr>
<td>OPT</td>
<td>19</td>
</tr>
<tr>
<td>opt. order</td>
<td>37</td>
</tr>
<tr>
<td>opt. order</td>
<td>37</td>
</tr>
<tr>
<td>sub. opt</td>
<td>39</td>
</tr>
<tr>
<td>sub. opt</td>
<td>41</td>
</tr>
<tr>
<td>AND, OR, C</td>
<td>177</td>
</tr>
<tr>
<td>fun. or</td>
<td>28</td>
</tr>
<tr>
<td>R. ord_pres_strict_thm</td>
<td>435</td>
</tr>
<tr>
<td>R. ord_pres_strict_thm</td>
<td>507</td>
</tr>
<tr>
<td>ORDER</td>
<td>36</td>
</tr>
<tr>
<td>induced. order. antisym_thm</td>
<td>414</td>
</tr>
<tr>
<td>ascii. order</td>
<td>36</td>
</tr>
<tr>
<td>induced. order. complete_thm</td>
<td>415</td>
</tr>
<tr>
<td>induced. order. dense_thm</td>
<td>415</td>
</tr>
<tr>
<td>gen. term. order</td>
<td>320</td>
</tr>
<tr>
<td>induced. order</td>
<td>36</td>
</tr>
<tr>
<td>integer. order</td>
<td>42</td>
</tr>
<tr>
<td>induced. order</td>
<td>37</td>
</tr>
<tr>
<td>induced. order. irrefl_thm</td>
<td>414</td>
</tr>
<tr>
<td>list. order</td>
<td>37</td>
</tr>
<tr>
<td>dy_less. order. lemmas_thm</td>
<td>430</td>
</tr>
<tr>
<td>no. order</td>
<td>36</td>
</tr>
<tr>
<td>opt. order</td>
<td>37</td>
</tr>
<tr>
<td>pair. order</td>
<td>37</td>
</tr>
<tr>
<td>rev. order</td>
<td>37</td>
</tr>
<tr>
<td>induced. strict_linear_order</td>
<td>415</td>
</tr>
<tr>
<td>induced. strict_partial_order</td>
<td>414</td>
</tr>
<tr>
<td>less. well. order</td>
<td>448</td>
</tr>
<tr>
<td>less. well. order</td>
<td>462</td>
</tr>
<tr>
<td>plus. order</td>
<td>447</td>
</tr>
<tr>
<td>plus. order</td>
<td>462</td>
</tr>
<tr>
<td>plus. order</td>
<td>462</td>
</tr>
<tr>
<td>less. well. order</td>
<td>448</td>
</tr>
<tr>
<td>less. well. order</td>
<td>462</td>
</tr>
<tr>
<td>Z. plus. order</td>
<td>452</td>
</tr>
<tr>
<td>Z. plus. order</td>
<td>477</td>
</tr>
<tr>
<td>Z. times. order</td>
<td>454</td>
</tr>
<tr>
<td>Z. times. order</td>
<td>477</td>
</tr>
<tr>
<td>R. times. order</td>
<td>501</td>
</tr>
<tr>
<td>R. times. order</td>
<td>510</td>
</tr>
<tr>
<td>R. times. order</td>
<td>510</td>
</tr>
<tr>
<td>≤. well. order</td>
<td>448</td>
</tr>
<tr>
<td>≤. well. order</td>
<td>462</td>
</tr>
<tr>
<td>Z. plus. order</td>
<td>452</td>
</tr>
<tr>
<td>Z. times. order</td>
<td>477</td>
</tr>
<tr>
<td>Z. times. order</td>
<td>477</td>
</tr>
<tr>
<td>induced. order. trans_thm</td>
<td>414</td>
</tr>
<tr>
<td>induced. order. trich_thm</td>
<td>414</td>
</tr>
<tr>
<td>type. order</td>
<td>323</td>
</tr>
<tr>
<td>induced. order. unbounded</td>
<td>415</td>
</tr>
<tr>
<td>_above. thm</td>
<td>415</td>
</tr>
<tr>
<td>induced. order. unbounded</td>
<td>415</td>
</tr>
<tr>
<td>_below. thm</td>
<td>415</td>
</tr>
<tr>
<td>R. term. order</td>
<td>510</td>
</tr>
<tr>
<td>Z. term. order</td>
<td>484</td>
</tr>
<tr>
<td>opt.</td>
<td>order1</td>
</tr>
<tr>
<td>R_term.</td>
<td>order1</td>
</tr>
<tr>
<td>ORELSE</td>
<td>277</td>
</tr>
<tr>
<td>ORELSE_C</td>
<td>197</td>
</tr>
<tr>
<td>ORELSE_CAN</td>
<td>197</td>
</tr>
<tr>
<td>ORELSE_T</td>
<td>277</td>
</tr>
<tr>
<td>ORELSE_TTCL</td>
<td>277</td>
</tr>
<tr>
<td>close.</td>
<td>out</td>
</tr>
<tr>
<td>flush.</td>
<td>out</td>
</tr>
<tr>
<td>is_term.</td>
<td>out</td>
</tr>
<tr>
<td>OutL</td>
<td>425</td>
</tr>
<tr>
<td>OutL</td>
<td>426</td>
</tr>
<tr>
<td>open.</td>
<td>out</td>
</tr>
<tr>
<td>OutR</td>
<td>425</td>
</tr>
<tr>
<td>OutR</td>
<td>426</td>
</tr>
<tr>
<td>std.</td>
<td>out</td>
</tr>
<tr>
<td>output</td>
<td></td>
</tr>
<tr>
<td>Simple</td>
<td>Output</td>
</tr>
<tr>
<td>Simple</td>
<td>output_theory</td>
</tr>
<tr>
<td>translate_for.</td>
<td>output</td>
</tr>
<tr>
<td>translate_for.</td>
<td>outstream</td>
</tr>
<tr>
<td>R.</td>
<td>over_1_thm</td>
</tr>
<tr>
<td>R.</td>
<td>over_1_thm</td>
</tr>
<tr>
<td>R.</td>
<td>over_cancel_eq_thm</td>
</tr>
<tr>
<td>R.</td>
<td>over_cancel_eq_thm</td>
</tr>
<tr>
<td>R.</td>
<td>over_conv</td>
</tr>
<tr>
<td>R.</td>
<td>over_def</td>
</tr>
<tr>
<td>dest.</td>
<td>over</td>
</tr>
<tr>
<td>R.</td>
<td>over_eq_0_thm</td>
</tr>
<tr>
<td>R.</td>
<td>over_eq_0_thm</td>
</tr>
<tr>
<td>is.</td>
<td>over</td>
</tr>
<tr>
<td>mk.</td>
<td>over</td>
</tr>
<tr>
<td>R.</td>
<td>over_over_1_thm</td>
</tr>
<tr>
<td>R.</td>
<td>over_over_1_thm</td>
</tr>
<tr>
<td>R.</td>
<td>over_over_1_thm</td>
</tr>
<tr>
<td>R.</td>
<td>over_plus_over_1_thm</td>
</tr>
<tr>
<td>R.</td>
<td>over_plus_over_1_thm</td>
</tr>
<tr>
<td>R.</td>
<td>over_recip_thm</td>
</tr>
<tr>
<td>R.</td>
<td>over_recip_thm</td>
</tr>
<tr>
<td>R.</td>
<td>0.</td>
</tr>
<tr>
<td>R.</td>
<td>0.</td>
</tr>
<tr>
<td>R.</td>
<td>over_over_1_thm</td>
</tr>
<tr>
<td>R.</td>
<td>over_over_1_thm</td>
</tr>
<tr>
<td>R.</td>
<td>over_plus_over_1_thm</td>
</tr>
<tr>
<td>R.</td>
<td>over_plus_over_1_thm</td>
</tr>
<tr>
<td>R.</td>
<td>over_times_1_thm</td>
</tr>
<tr>
<td>R.</td>
<td>over_times_1_thm</td>
</tr>
<tr>
<td>R.</td>
<td>over_times_over_1_thm</td>
</tr>
<tr>
<td>R.</td>
<td>over_times_over_1_thm</td>
</tr>
<tr>
<td>R.</td>
<td>over_times_over_1_thm</td>
</tr>
<tr>
<td>R.</td>
<td>over_times_over_1_thm</td>
</tr>
<tr>
<td>R.</td>
<td>overwrite</td>
</tr>
<tr>
<td>list</td>
<td>overwrite</td>
</tr>
<tr>
<td>pending_reset_subgoal</td>
<td>package</td>
</tr>
<tr>
<td>subgoal</td>
<td>package_quiet</td>
</tr>
<tr>
<td>subgoal</td>
<td>package_size</td>
</tr>
</tbody>
</table>
NZ.1 plus_homomorphism 477
    plus1.\leq_thm 462
    plus1.\leq_.thm 448
    \neg. plus1.\leq_thm 462
    \neg. plus1.\leq_.thm 448
is.1 plus 463
    simple_one. 500
is.R.1 plus 500
    point_conv 337
is.Z.1 plus 476
    POLY 468
    poly_conv 321
R.1 plus_minus_thm 433
    poly 468
R.1 plus_minus_thm 508
    poly 468
R.1 plus_minus_thm1 433
    poly 468
Z.1 plus_minus_thm 452
    POLY 468
Z.1 plus_minus_thm 477
    POLY 468
mk.1 plus 464
mk.R.1 plus 501
mk.Z.1 plus 476
R.1 plus_mono_thm 433
    save. 328
R.1 plus_mono_thm 508
    basic_res. 72
R.1 plus_mono_thm1 433
    HT 72
R.1 plus_minus_thm1 452
    PostOp 125
R.1 plus_minus_thm2 433
    postfixes 128
R.1 plus_minus_thm2 508
    get. 28
R.1 plus_order_thm 447
    fun. 28
R.1 plus_order_thm 462
R.1 plus_order_thm 477
R.1 plus_order_thm 477
Z.abs.1 plus_thm 456
Z.abs.1 plus_thm 477
Z.N.1 plus_thm 454
Z.N.1 plus_thm 477
R.1 plus_unit_thm 433
R.1 plus_unit_thm 508
Z.1 plus_N.thm 455
Z.1 plus_N.thm 477
Z.1 plus_N.thm 477
Z.1 plus0.thm 452
Z.1 plus0.thm 477
Z.1 plus1.conv 461
    plus1.less_thm 448
    plus1.less_thm 462
    \neg. plus1.thm 448
    \neg. plus1.thm 462
one.1 one.thm 477

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
<table>
<thead>
<tr>
<th>Key Word</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>refl_thm</td>
<td>477</td>
</tr>
<tr>
<td>Refl</td>
<td>383</td>
</tr>
<tr>
<td>Reflexive</td>
<td>384</td>
</tr>
<tr>
<td>reflexive_def</td>
<td>492</td>
</tr>
<tr>
<td>ReflexiveIn</td>
<td>398</td>
</tr>
<tr>
<td>'bin_rel</td>
<td>493</td>
</tr>
<tr>
<td>fun_rel_alg</td>
<td>494</td>
</tr>
<tr>
<td>RelCombine</td>
<td>383</td>
</tr>
<tr>
<td>RelCombine</td>
<td>384</td>
</tr>
<tr>
<td>rel_combine_def</td>
<td>492</td>
</tr>
<tr>
<td>inv_rel_def</td>
<td>492</td>
</tr>
<tr>
<td>list_rel_def</td>
<td>495</td>
</tr>
<tr>
<td>'fun_rel_ext</td>
<td>493</td>
</tr>
<tr>
<td>bin_rel_ext_clauses</td>
<td>385</td>
</tr>
<tr>
<td>bin_rel_ext_clauses</td>
<td>492</td>
</tr>
<tr>
<td>bin_rel_insert_thm</td>
<td>386</td>
</tr>
<tr>
<td>bin_rel_insert_thm</td>
<td>492</td>
</tr>
<tr>
<td>Inv Rel</td>
<td>383</td>
</tr>
<tr>
<td>Inv Rel</td>
<td>384</td>
</tr>
<tr>
<td>RelList</td>
<td>418</td>
</tr>
<tr>
<td>RelList</td>
<td>419</td>
</tr>
<tr>
<td>Bin RelTheory</td>
<td>492</td>
</tr>
<tr>
<td>Fun RelTheory</td>
<td>493</td>
</tr>
<tr>
<td>fun_rel_thm</td>
<td>409</td>
</tr>
<tr>
<td>fun_rel_thm</td>
<td>459</td>
</tr>
<tr>
<td>inv_rel_thm</td>
<td>385</td>
</tr>
<tr>
<td>inv_rel_thm</td>
<td>492</td>
</tr>
<tr>
<td>inv_rel_in_arrow_thm</td>
<td>397</td>
</tr>
<tr>
<td>inv_rel_in_arrow_thm</td>
<td>493</td>
</tr>
<tr>
<td>rel_in_clauses</td>
<td>385</td>
</tr>
<tr>
<td>rel_in_clauses</td>
<td>492</td>
</tr>
<tr>
<td>bin_rel_universe_thm</td>
<td>386</td>
</tr>
<tr>
<td>bin_rel_universe_thm</td>
<td>492</td>
</tr>
<tr>
<td>rename</td>
<td>111</td>
</tr>
<tr>
<td>rename_tac</td>
<td>281</td>
</tr>
<tr>
<td>reconsider_conv</td>
<td>232</td>
</tr>
<tr>
<td>reconsider_conv</td>
<td>234</td>
</tr>
<tr>
<td>RepChar</td>
<td>383</td>
</tr>
<tr>
<td>abs_char</td>
<td>388</td>
</tr>
<tr>
<td>abs_char</td>
<td>485</td>
</tr>
<tr>
<td>is_R</td>
<td>430</td>
</tr>
<tr>
<td>is_R</td>
<td>452</td>
</tr>
<tr>
<td>is_char</td>
<td>388</td>
</tr>
<tr>
<td>is_char</td>
<td>485</td>
</tr>
<tr>
<td>is_list</td>
<td>402</td>
</tr>
<tr>
<td>is_list</td>
<td>484</td>
</tr>
<tr>
<td>is_one</td>
<td>410</td>
</tr>
<tr>
<td>is_one</td>
<td>490</td>
</tr>
<tr>
<td>is_pair</td>
<td>416</td>
</tr>
<tr>
<td>is_pair</td>
<td>459</td>
</tr>
<tr>
<td>is_set</td>
<td>421</td>
</tr>
<tr>
<td>is_set</td>
<td>486</td>
</tr>
<tr>
<td>is_sum</td>
<td>425</td>
</tr>
<tr>
<td>is_sum</td>
<td>490</td>
</tr>
<tr>
<td>is_N</td>
<td>446</td>
</tr>
</tbody>
</table>
save 53  set_def 421
save_and_exit 53  set_def 486
save_and_quit 53  dest_enum.
save_as 53  set    84
save_consistency_thm 159  set_def_clauses 422
new_save 51  set_def_clauses 486
save fun 50  set_def 421
get save_pop_thm 247  set_def 486
SaveThm 133  Down Set 411
save_thm 148  Down Set 413
List SaveThm 133  pp' set_eval_ad_cs_3
T T Saved 153  _convs 353
set_conv 352  pp' set_eval_ad_net 353
set_convs 352  pp' set_eval_ad_rw_canon 352
set_eqn_convs 361  pp' set_eval_ad_rw_net 351
set_eqn_convs 361  pp' set_eval_ad_sc_convs 352
set_thms 361  pp' set_eval_ad_3_cd 352
scale_asm_tac 471  _thms 353
scale_nth_asm_tac 471  pp' set_eval_ad_3_vs
scale_rule 472
schoeder_bernstein 398
_s_thm 398
schoeder_bernstein 398
_s_thm1
scratch 39
scratch 41
Seconds 56
LISTER SECTION 77  is_enum.
LSAD Section 77  set labelled_goal 247
HH Semi 72  set_line_length 70
set_merge_pcs 357
R_ semigroup_dense_thm 435  mk_enum.
R_ semigroup_dense_thm 507
Separator 72
SEQ 121
seq_ext 496
SeqTheory 495
SET 420
SET 421
pp' set_banner 51  Is SetRep 420
Is SetRep 421
set_cache_theories 171
SetComp 420
SetComp 421
D SetComp 82
set_comp_def 421
set_comp_def 486
dest_set_comp 87
is_ set_comp 94
mk_set_comp 105
set_compactification 132
_cache 49
set_controls 49
C_enum set_cone 489
set_cs_3_convs 358
set_current_language 130
DEnum Set 82
pp set_database_info 51
Fin
set_rw_canons 360
set_rw_eqn_rule 361
set_rw_eqn_rule 361
set_rw_eqn_ext 361
set_sc_eqn_cxt 361
set_st_eqn_cxt 362
set_stats 55
set_string_control 49
set_string_contols 49
SetTheory 494
SetUserDatum 133
set_user_datum 149
set_variant_suffix 111
set_3_3鲧thms 362
set_3_3vs_thms 363
set alg 375

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>sets, clauses</td>
<td>421</td>
</tr>
<tr>
<td>sets, clauses</td>
<td>486</td>
</tr>
<tr>
<td>down, sets_cuts_thm</td>
<td>413</td>
</tr>
<tr>
<td>down, sets_dense_thm</td>
<td>413</td>
</tr>
<tr>
<td>Down, Sets</td>
<td>411</td>
</tr>
<tr>
<td>Down Sets</td>
<td>413</td>
</tr>
<tr>
<td>sets_eq_conv</td>
<td>487</td>
</tr>
<tr>
<td>sets_ext</td>
<td>376</td>
</tr>
<tr>
<td>sets_ext</td>
<td>377</td>
</tr>
<tr>
<td>sets_ext</td>
<td>376</td>
</tr>
<tr>
<td>sets_ext_clauses</td>
<td>423</td>
</tr>
<tr>
<td>sets_ext_clauses</td>
<td>486</td>
</tr>
<tr>
<td>sets_ext</td>
<td>377</td>
</tr>
<tr>
<td>sets_ext</td>
<td>377</td>
</tr>
<tr>
<td>down, sets_less_thm</td>
<td>414</td>
</tr>
<tr>
<td>sets_simple∃conv</td>
<td>488</td>
</tr>
<tr>
<td>SetsTheory</td>
<td>486</td>
</tr>
<tr>
<td>map, shape</td>
<td>277</td>
</tr>
<tr>
<td>Utility SharedTypes</td>
<td>19</td>
</tr>
<tr>
<td>get, shell_var</td>
<td>50</td>
</tr>
<tr>
<td>sho, rw</td>
<td>372</td>
</tr>
<tr>
<td>show, term</td>
<td>76</td>
</tr>
<tr>
<td>show, term</td>
<td>76</td>
</tr>
<tr>
<td>show, type</td>
<td>76</td>
</tr>
<tr>
<td>dest_Z, signed_int</td>
<td>475</td>
</tr>
<tr>
<td>is_Z, signed_int</td>
<td>476</td>
</tr>
<tr>
<td>mk_Z, signed_int</td>
<td>476</td>
</tr>
<tr>
<td>prop, simp_conv</td>
<td>336</td>
</tr>
<tr>
<td>Simple</td>
<td>58</td>
</tr>
<tr>
<td>Simple, simple_abstractions</td>
<td>365</td>
</tr>
<tr>
<td>SIMPLE_BINDER_C</td>
<td>205</td>
</tr>
<tr>
<td>dest, simple_binder</td>
<td>87</td>
</tr>
<tr>
<td>is, simple_binder</td>
<td>95</td>
</tr>
<tr>
<td>mk, simple_binder</td>
<td>105</td>
</tr>
<tr>
<td>strip</td>
<td>112</td>
</tr>
<tr>
<td>SimpleDictionary</td>
<td>32</td>
</tr>
<tr>
<td>simple_eq_match_conv</td>
<td>205</td>
</tr>
<tr>
<td>simple_eq_match</td>
<td>206</td>
</tr>
<tr>
<td>simple_eq_match</td>
<td>206</td>
</tr>
<tr>
<td>simple_eq_match</td>
<td>207</td>
</tr>
<tr>
<td>simple_eq_match</td>
<td>120</td>
</tr>
<tr>
<td>simple_eq_match_ext</td>
<td>363</td>
</tr>
<tr>
<td>SimpleNewDefn</td>
<td>133</td>
</tr>
<tr>
<td>simple_new_defn</td>
<td>149</td>
</tr>
<tr>
<td>simple_one_point</td>
<td>337</td>
</tr>
<tr>
<td>SimpleOutput</td>
<td>69</td>
</tr>
<tr>
<td>simple_tac_proof</td>
<td>284</td>
</tr>
<tr>
<td>simple_tacProof</td>
<td>285</td>
</tr>
<tr>
<td>DEST, SIMPLE_TERM</td>
<td>81</td>
</tr>
<tr>
<td>dest, simple_term</td>
<td>87</td>
</tr>
<tr>
<td>mk, simple_term</td>
<td>106</td>
</tr>
<tr>
<td>DEST, SIMPLE_TYPE</td>
<td>81</td>
</tr>
<tr>
<td>dest, simple_type</td>
<td>87</td>
</tr>
<tr>
<td>Symbol</td>
<td>Table</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td><strong>REPEAT</strong></td>
<td>281</td>
</tr>
<tr>
<td><strong>REPEAT_UNTIL</strong></td>
<td>282</td>
</tr>
<tr>
<td>⇒; _t_rewrite_canon</td>
<td>223</td>
</tr>
<tr>
<td><strong>ROTATE</strong></td>
<td>284</td>
</tr>
<tr>
<td><strong>SOLVED</strong></td>
<td>289</td>
</tr>
<tr>
<td><strong>SPEC_ASM</strong></td>
<td>291</td>
</tr>
<tr>
<td><strong>SPEC_NTH_ASM</strong></td>
<td>291</td>
</tr>
<tr>
<td><strong>STRIP</strong></td>
<td>294</td>
</tr>
<tr>
<td><strong>STRIP_CONCL</strong></td>
<td>294</td>
</tr>
<tr>
<td><strong>SWAP_ASM</strong></td>
<td>296</td>
</tr>
<tr>
<td><strong>SWAP_NTH_ASM</strong></td>
<td>296</td>
</tr>
<tr>
<td><strong>THEN</strong></td>
<td>300</td>
</tr>
<tr>
<td><strong>THEN_LIST</strong></td>
<td>298</td>
</tr>
<tr>
<td><strong>THEN_TRY</strong></td>
<td>299</td>
</tr>
<tr>
<td><strong>THEN_TRY_LIST</strong></td>
<td>299</td>
</tr>
<tr>
<td><strong>TOP_ASM</strong></td>
<td>300</td>
</tr>
<tr>
<td><strong>VAR_ELIM_ASM</strong></td>
<td>301</td>
</tr>
<tr>
<td><strong>VAR_ELIM_NTH_ASM</strong></td>
<td>301</td>
</tr>
<tr>
<td><strong>ALL_ASM_FC</strong></td>
<td>T1</td>
</tr>
<tr>
<td><strong>ALL_ASM_FORWARD</strong></td>
<td>T1</td>
</tr>
<tr>
<td><strong>ALL_FORWARD</strong></td>
<td>T1</td>
</tr>
<tr>
<td><strong>ALL_VAR_ELIM_ASM</strong></td>
<td>T1</td>
</tr>
<tr>
<td><strong>ASM_FC</strong></td>
<td>T1</td>
</tr>
<tr>
<td><strong>ASM_FORWARD</strong></td>
<td>T1</td>
</tr>
<tr>
<td><strong>BASIC</strong></td>
<td>T1</td>
</tr>
<tr>
<td><strong>RESOLUTION</strong></td>
<td>T1</td>
</tr>
<tr>
<td><strong>EXTEND_PC</strong></td>
<td>T1</td>
</tr>
<tr>
<td><strong>EXTEND_PCS</strong></td>
<td>T1</td>
</tr>
<tr>
<td><strong>FC</strong></td>
<td>T1</td>
</tr>
<tr>
<td><strong>FORWARD_CHAIN</strong></td>
<td>T1</td>
</tr>
<tr>
<td><strong>GEN_INDUCTION</strong></td>
<td>T1</td>
</tr>
<tr>
<td><strong>MERGE_PCS</strong></td>
<td>T1</td>
</tr>
<tr>
<td><strong>PC</strong></td>
<td>T1</td>
</tr>
<tr>
<td><strong>REPEAT_UNTIL</strong></td>
<td>T1</td>
</tr>
<tr>
<td><strong>THEN</strong></td>
<td>T1</td>
</tr>
<tr>
<td><strong>CASES</strong></td>
<td>T2</td>
</tr>
<tr>
<td><strong>IF</strong></td>
<td>T2</td>
</tr>
<tr>
<td>⇒; T2</td>
<td>301</td>
</tr>
<tr>
<td>¬; T2</td>
<td>307</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Category</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>list_spec_asm</td>
<td>tac</td>
<td>290</td>
</tr>
<tr>
<td>list_swap_nth_asm</td>
<td>tac</td>
<td>290</td>
</tr>
<tr>
<td>list_swap_asm_concl</td>
<td>tac</td>
<td>276</td>
</tr>
<tr>
<td>once_asm_rewrite</td>
<td>tac</td>
<td>283</td>
</tr>
<tr>
<td>once_asm_rewrite_concl</td>
<td>tac</td>
<td>276</td>
</tr>
<tr>
<td>once_rewrite</td>
<td>tac</td>
<td>283</td>
</tr>
<tr>
<td>once_rewrite_thm</td>
<td>tac</td>
<td>283</td>
</tr>
<tr>
<td>prim_rewrite</td>
<td>tac</td>
<td>279</td>
</tr>
<tr>
<td>tac_proof</td>
<td>tac</td>
<td>297</td>
</tr>
<tr>
<td>simple</td>
<td>tac</td>
<td>284</td>
</tr>
<tr>
<td>prop_eq_prove</td>
<td>tac</td>
<td>315</td>
</tr>
<tr>
<td>prove</td>
<td>tac</td>
<td>297</td>
</tr>
<tr>
<td>prove_3</td>
<td>tac</td>
<td>315</td>
</tr>
<tr>
<td>pure_asm_rewrite</td>
<td>tac</td>
<td>283</td>
</tr>
<tr>
<td>pure_asm_rewrite_concl</td>
<td>tac</td>
<td>283</td>
</tr>
<tr>
<td>pure_once_asm</td>
<td>tac</td>
<td>283</td>
</tr>
<tr>
<td>pure_once_asm_concl</td>
<td>tac</td>
<td>283</td>
</tr>
<tr>
<td>pure_once_rewrite</td>
<td>tac</td>
<td>283</td>
</tr>
<tr>
<td>pure_once_rewrite_concl</td>
<td>tac</td>
<td>283</td>
</tr>
<tr>
<td>pure_rewrite</td>
<td>tac</td>
<td>283</td>
</tr>
<tr>
<td>pure_rewrite_thm</td>
<td>tac</td>
<td>283</td>
</tr>
<tr>
<td>rename</td>
<td>tac</td>
<td>281</td>
</tr>
<tr>
<td>rewrite</td>
<td>tac</td>
<td>283</td>
</tr>
<tr>
<td>rewrite_thm</td>
<td>tac</td>
<td>283</td>
</tr>
<tr>
<td>scale_asm</td>
<td>tac</td>
<td>471</td>
</tr>
<tr>
<td>scale_nth_asm</td>
<td>tac</td>
<td>471</td>
</tr>
<tr>
<td>set_pr</td>
<td>tac</td>
<td>360</td>
</tr>
<tr>
<td>simple_taut</td>
<td>tac</td>
<td>285</td>
</tr>
<tr>
<td>simple(trim)</td>
<td>tac</td>
<td>286</td>
</tr>
<tr>
<td>simple_in</td>
<td>tac</td>
<td>287</td>
</tr>
<tr>
<td>simple_v</td>
<td>tac</td>
<td>287</td>
</tr>
<tr>
<td>simple_3</td>
<td>tac</td>
<td>289</td>
</tr>
<tr>
<td>spec_asm</td>
<td>tac</td>
<td>290</td>
</tr>
<tr>
<td>spec_nth_asm</td>
<td>tac</td>
<td>290</td>
</tr>
<tr>
<td>step Strip</td>
<td>tac</td>
<td>292</td>
</tr>
<tr>
<td>step_strip_asm</td>
<td>tac</td>
<td>292</td>
</tr>
<tr>
<td>strip</td>
<td>tac</td>
<td>293</td>
</tr>
<tr>
<td>strip_asm</td>
<td>tac</td>
<td>294</td>
</tr>
<tr>
<td>strip_concl</td>
<td>tac</td>
<td>294</td>
</tr>
<tr>
<td>sum_cases</td>
<td>tac</td>
<td>491</td>
</tr>
<tr>
<td>swap_asm_concl</td>
<td>tac</td>
<td>295</td>
</tr>
<tr>
<td>swap_nth_asm_concl</td>
<td>tac</td>
<td>295</td>
</tr>
<tr>
<td>swap_v</td>
<td>tac</td>
<td>296</td>
</tr>
<tr>
<td>t</td>
<td>tac</td>
<td>300</td>
</tr>
<tr>
<td>taut</td>
<td>tac</td>
<td>298</td>
</tr>
<tr>
<td>var_elim_asm</td>
<td>tac</td>
<td>301</td>
</tr>
<tr>
<td>var_elim_nth_asm</td>
<td>tac</td>
<td>301</td>
</tr>
<tr>
<td>⇒</td>
<td>tac</td>
<td>302</td>
</tr>
<tr>
<td>⇔</td>
<td>tac</td>
<td>302</td>
</tr>
<tr>
<td>R_delta_induction</td>
<td>tac</td>
<td>503</td>
</tr>
<tr>
<td>R_lin_arith</td>
<td>tac</td>
<td>506</td>
</tr>
<tr>
<td>_lin_arith_prove</td>
<td>tac</td>
<td>506</td>
</tr>
<tr>
<td>_lin_arith</td>
<td>tac</td>
<td>303</td>
</tr>
<tr>
<td>_left</td>
<td>tac</td>
<td>304</td>
</tr>
<tr>
<td>_right</td>
<td>tac</td>
<td>304</td>
</tr>
<tr>
<td>_elim</td>
<td>tac</td>
<td>308</td>
</tr>
<tr>
<td>_in</td>
<td>tac</td>
<td>305</td>
</tr>
<tr>
<td>⇒</td>
<td>tac</td>
<td>306</td>
</tr>
<tr>
<td>⇒_thm</td>
<td>tac</td>
<td>309</td>
</tr>
<tr>
<td>∀</td>
<td>tac</td>
<td>310</td>
</tr>
<tr>
<td>∃</td>
<td>tac</td>
<td>311</td>
</tr>
<tr>
<td>all_var_elim_asm</td>
<td>tac</td>
<td>252</td>
</tr>
<tr>
<td>basic_res</td>
<td>tac</td>
<td>330</td>
</tr>
<tr>
<td>gen_induction</td>
<td>tac</td>
<td>269</td>
</tr>
<tr>
<td>get_pr</td>
<td>tac</td>
<td>360</td>
</tr>
<tr>
<td>intro_v</td>
<td>tac</td>
<td>272</td>
</tr>
<tr>
<td>lin_induction</td>
<td>tac</td>
<td>470</td>
</tr>
<tr>
<td>basic_res</td>
<td>tac</td>
<td>330</td>
</tr>
<tr>
<td>basic_res</td>
<td>tac</td>
<td>331</td>
</tr>
<tr>
<td>basic_res</td>
<td>tac</td>
<td>331</td>
</tr>
<tr>
<td>apply</td>
<td>tac</td>
<td>243</td>
</tr>
<tr>
<td>tactic</td>
<td>tac</td>
<td>243</td>
</tr>
<tr>
<td>tactic_subgoal_warning</td>
<td>tac</td>
<td>242</td>
</tr>
<tr>
<td>Tactics1</td>
<td>tac</td>
<td>251</td>
</tr>
<tr>
<td>Tactics2</td>
<td>tac</td>
<td>251</td>
</tr>
<tr>
<td>Tactics3</td>
<td>tac</td>
<td>251</td>
</tr>
<tr>
<td>Tail</td>
<td>tac</td>
<td>418</td>
</tr>
<tr>
<td>tail_def</td>
<td>tac</td>
<td>495</td>
</tr>
<tr>
<td>tau_conv</td>
<td>tac</td>
<td>297</td>
</tr>
<tr>
<td>tau_rule</td>
<td>tac</td>
<td>298</td>
</tr>
<tr>
<td>tau_tac</td>
<td>tac</td>
<td>298</td>
</tr>
<tr>
<td>simple</td>
<td>tac</td>
<td>285</td>
</tr>
<tr>
<td>tc_def</td>
<td>tac</td>
<td>492</td>
</tr>
<tr>
<td>Term</td>
<td>tac</td>
<td>72</td>
</tr>
<tr>
<td>TERM</td>
<td>tac</td>
<td>82</td>
</tr>
<tr>
<td>term</td>
<td>tac</td>
<td>115</td>
</tr>
<tr>
<td>term_consts</td>
<td>tac</td>
<td>115</td>
</tr>
<tr>
<td>compact</td>
<td>tac</td>
<td>135</td>
</tr>
<tr>
<td>term</td>
<td>tac</td>
<td>82</td>
</tr>
<tr>
<td>dest</td>
<td>tac</td>
<td>88</td>
</tr>
<tr>
<td>dest_simple</td>
<td>tac</td>
<td>87</td>
</tr>
<tr>
<td>term</td>
<td>tac</td>
<td>88</td>
</tr>
<tr>
<td>term_diff</td>
<td>tac</td>
<td>115</td>
</tr>
<tr>
<td>term_fail</td>
<td>tac</td>
<td>115</td>
</tr>
<tr>
<td>term_fold</td>
<td>tac</td>
<td>115</td>
</tr>
<tr>
<td>format</td>
<td>tac</td>
<td>75</td>
</tr>
<tr>
<td>term</td>
<td>tac</td>
<td>115</td>
</tr>
<tr>
<td>term_grab</td>
<td>tac</td>
<td>115</td>
</tr>
<tr>
<td>is</td>
<td>tac</td>
<td>43</td>
</tr>
<tr>
<td>term_in</td>
<td>tac</td>
<td>116</td>
</tr>
<tr>
<td>term_less</td>
<td>tac</td>
<td>116</td>
</tr>
</tbody>
</table>
KEYWORD INDEX

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>term_map</td>
<td>116</td>
</tr>
<tr>
<td>term_match</td>
<td>116</td>
</tr>
<tr>
<td>term_mem</td>
<td>116</td>
</tr>
<tr>
<td>mk_term</td>
<td>107</td>
</tr>
<tr>
<td>mk_simple_term</td>
<td>106</td>
</tr>
<tr>
<td>term_order</td>
<td>322</td>
</tr>
<tr>
<td>R_term_order</td>
<td>509</td>
</tr>
<tr>
<td>R_term_order1</td>
<td>509</td>
</tr>
<tr>
<td>gen_term_order</td>
<td>320</td>
</tr>
<tr>
<td>make_term_order</td>
<td>320</td>
</tr>
<tr>
<td>R_term_order1</td>
<td>510</td>
</tr>
<tr>
<td>Z_term_order</td>
<td>484</td>
</tr>
<tr>
<td>R_term_order1</td>
<td>510</td>
</tr>
<tr>
<td>is_term</td>
<td>43</td>
</tr>
<tr>
<td>asm_inst_term</td>
<td>179</td>
</tr>
<tr>
<td>inst_term_rule</td>
<td>192</td>
</tr>
<tr>
<td>KIInst TermRule</td>
<td>131</td>
</tr>
<tr>
<td>show_term</td>
<td>76</td>
</tr>
<tr>
<td>string_of_term</td>
<td>111</td>
</tr>
<tr>
<td>term_tycons</td>
<td>116</td>
</tr>
<tr>
<td>term_types</td>
<td>116</td>
</tr>
<tr>
<td>term_tyvars</td>
<td>116</td>
</tr>
<tr>
<td>UD_Term</td>
<td>121</td>
</tr>
<tr>
<td>term_unify</td>
<td>334</td>
</tr>
<tr>
<td>term_union</td>
<td>116</td>
</tr>
<tr>
<td>list_term_union</td>
<td>101</td>
</tr>
<tr>
<td>term_vars</td>
<td>116</td>
</tr>
<tr>
<td>format_term1</td>
<td>75</td>
</tr>
<tr>
<td>ask_at_terminal</td>
<td>62</td>
</tr>
<tr>
<td>reset_use</td>
<td>67</td>
</tr>
<tr>
<td>use_terminal</td>
<td>68</td>
</tr>
<tr>
<td>declare_terminator</td>
<td>126</td>
</tr>
<tr>
<td>undeclare_terminator</td>
<td>130</td>
</tr>
<tr>
<td>get_terminators</td>
<td>128</td>
</tr>
<tr>
<td>get_current_terminators</td>
<td>127</td>
</tr>
<tr>
<td>get_undeclared_terminators</td>
<td>129</td>
</tr>
<tr>
<td>LS_Terminals</td>
<td>77</td>
</tr>
<tr>
<td>Terminators</td>
<td>77</td>
</tr>
<tr>
<td>LSAD_Terms</td>
<td>77</td>
</tr>
<tr>
<td>Terms</td>
<td>81</td>
</tr>
<tr>
<td>pp’TypesAnd</td>
<td>81</td>
</tr>
<tr>
<td>TypesAnd</td>
<td>81</td>
</tr>
<tr>
<td>TEST</td>
<td>153</td>
</tr>
<tr>
<td>THM_INFO_TEST</td>
<td>153</td>
</tr>
<tr>
<td>get_text</td>
<td>13</td>
</tr>
<tr>
<td>get_message_text</td>
<td>18</td>
</tr>
<tr>
<td>THEN_C</td>
<td>218</td>
</tr>
<tr>
<td>THEN_CAN</td>
<td>218</td>
</tr>
<tr>
<td>THEN_C</td>
<td>335</td>
</tr>
<tr>
<td>THEN_CAN</td>
<td>218</td>
</tr>
<tr>
<td>CONV_THEN</td>
<td>260</td>
</tr>
<tr>
<td>if_then_clim</td>
<td>191</td>
</tr>
<tr>
<td>FAIL_THEN</td>
<td>263</td>
</tr>
<tr>
<td>FAIL_WITH_THEN</td>
<td>263</td>
</tr>
<tr>
<td>HT_Then</td>
<td>72</td>
</tr>
<tr>
<td>ID_THEN</td>
<td>270</td>
</tr>
<tr>
<td>IF_THEN</td>
<td>271</td>
</tr>
<tr>
<td>THEN_LIST</td>
<td>298</td>
</tr>
<tr>
<td>Keyword</td>
<td>Type</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------</td>
</tr>
<tr>
<td><code>R_frac_cancel</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R_frac_cancel.eq</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R_frac_cross_mult_eq</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R_frac_cross_mult_eq</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R frac_less</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R frac_less</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R frac_less_minus</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R frac_less_minus</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R frac_minus</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R frac_minus_frac</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R frac_plus</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R frac_plus</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R frac_plus</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R frac_recip</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R frac_times</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R frac_times</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R frac_N</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R frac_N</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R halve_closed_dense</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R halve_closed_dense</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_0</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_0</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_antism</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_antism</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_cases</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_cases</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_dense</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_dense</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_irrefl</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_irrefl</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_0</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_0</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_strong_dense</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_strong_dense</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_sup</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_sup</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_sup_bc</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_sup_bc</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_sup_c</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_sup_c</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_trans</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_trans</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_eq</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_eq</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_eq_0</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_eq_0</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_trans</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R less_trans</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R max_cons</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R max_cons</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R max_conc</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R max_conc</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R min_cons</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R min_cons</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R min_conc</code></td>
<td>thm</td>
</tr>
<tr>
<td><code>R min_conc</code></td>
<td>thm</td>
</tr>
<tr>
<td>Keyword</td>
<td>Type</td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>R._plus_assoc</td>
<td>thm</td>
</tr>
<tr>
<td>R._plus_assoc</td>
<td>thm</td>
</tr>
<tr>
<td>R._plus_comm</td>
<td>thm</td>
</tr>
<tr>
<td>R._plus_comm</td>
<td>thm</td>
</tr>
<tr>
<td>R._plus_minus</td>
<td>thm</td>
</tr>
<tr>
<td>R._plus_minus</td>
<td>thm</td>
</tr>
<tr>
<td>R._plus_mono</td>
<td>thm</td>
</tr>
<tr>
<td>R._plus_mono</td>
<td>thm</td>
</tr>
<tr>
<td>R._plus_order</td>
<td>thm</td>
</tr>
<tr>
<td>R._plus_order</td>
<td>thm</td>
</tr>
<tr>
<td>R._plus_unit</td>
<td>thm</td>
</tr>
<tr>
<td>R._plus_unit</td>
<td>thm</td>
</tr>
<tr>
<td>R._sup_eq_bc</td>
<td>thm</td>
</tr>
<tr>
<td>R._sup_eq_bc</td>
<td>thm</td>
</tr>
<tr>
<td>R._sup_less_bc</td>
<td>thm</td>
</tr>
<tr>
<td>R._sup_less_bc</td>
<td>thm</td>
</tr>
<tr>
<td>R._sup_plus</td>
<td>thm</td>
</tr>
<tr>
<td>R._sup_plus</td>
<td>thm</td>
</tr>
<tr>
<td>R._sup_plus Sup</td>
<td>thm</td>
</tr>
<tr>
<td>R._sup_plus Sup</td>
<td>thm</td>
</tr>
<tr>
<td>R._times_0</td>
<td>thm</td>
</tr>
<tr>
<td>R._times_0</td>
<td>thm</td>
</tr>
<tr>
<td>R._times_1</td>
<td>thm</td>
</tr>
<tr>
<td>R._times_1</td>
<td>thm</td>
</tr>
<tr>
<td>R._times_assoc</td>
<td>thm</td>
</tr>
<tr>
<td>R._times_assoc</td>
<td>thm</td>
</tr>
<tr>
<td>R._times_cancel</td>
<td>thm</td>
</tr>
<tr>
<td>R._times_cancel</td>
<td>thm</td>
</tr>
<tr>
<td>R._times_comm</td>
<td>thm</td>
</tr>
<tr>
<td>R._times_comm</td>
<td>thm</td>
</tr>
<tr>
<td>R._times_eq_0</td>
<td>thm</td>
</tr>
<tr>
<td>R._times_eq_0</td>
<td>thm</td>
</tr>
<tr>
<td>R._times_minus</td>
<td>thm</td>
</tr>
<tr>
<td>R._times_minus</td>
<td>thm</td>
</tr>
<tr>
<td>R._times mono</td>
<td>thm</td>
</tr>
<tr>
<td>R._times mono</td>
<td>thm</td>
</tr>
<tr>
<td>R._times mono =&gt;</td>
<td>thm</td>
</tr>
<tr>
<td>R._times mono =&gt;</td>
<td>thm</td>
</tr>
<tr>
<td>R._times_order</td>
<td>thm</td>
</tr>
<tr>
<td>R._times_order</td>
<td>thm</td>
</tr>
<tr>
<td>R._times plus distrib</td>
<td>thm</td>
</tr>
<tr>
<td>R._times plus distrib</td>
<td>thm</td>
</tr>
<tr>
<td>R._times recip</td>
<td>thm</td>
</tr>
<tr>
<td>R._times recip</td>
<td>thm</td>
</tr>
<tr>
<td>R._times unit</td>
<td>thm</td>
</tr>
<tr>
<td>R._times unit</td>
<td>thm</td>
</tr>
<tr>
<td>R._unbounded above</td>
<td>thm</td>
</tr>
<tr>
<td>R._unbounded above</td>
<td>thm</td>
</tr>
<tr>
<td>R._unbounded below</td>
<td>thm</td>
</tr>
<tr>
<td>R._unbounded below</td>
<td>thm</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Section</th>
<th>Line</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>\not\iff, thm</td>
<td>408</td>
<td>508</td>
<td></td>
</tr>
<tr>
<td>\not\land, thm</td>
<td>309</td>
<td>440</td>
<td></td>
</tr>
<tr>
<td>\not\lor, thm</td>
<td>408</td>
<td>510</td>
<td></td>
</tr>
<tr>
<td>\not\land, thm</td>
<td>309</td>
<td>434</td>
<td></td>
</tr>
<tr>
<td>\not\lor, thm</td>
<td>397</td>
<td>508</td>
<td></td>
</tr>
<tr>
<td>\not\land, thm</td>
<td>408</td>
<td>440</td>
<td></td>
</tr>
<tr>
<td>\not\lor, thm</td>
<td>227</td>
<td>455</td>
<td></td>
</tr>
<tr>
<td>\not\land, thm</td>
<td>408</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\not\lor, thm</td>
<td>228</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\not\land, thm</td>
<td>408</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\not\lor, thm</td>
<td>448</td>
<td>452</td>
<td></td>
</tr>
<tr>
<td>\not\land, thm</td>
<td>462</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\Rightarrow rewrite, thm</td>
<td>184</td>
<td>453</td>
<td></td>
</tr>
<tr>
<td>\Rightarrow rewrite, thm</td>
<td>408</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\forall rewrite, thm</td>
<td>184</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\forall rewrite, thm</td>
<td>409</td>
<td>453</td>
<td></td>
</tr>
<tr>
<td>\exists intro, thm</td>
<td>233</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\exists intro, thm</td>
<td>408</td>
<td>456</td>
<td></td>
</tr>
<tr>
<td>\exists rewrite, thm</td>
<td>184</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\exists rewrite, thm</td>
<td>409</td>
<td>456</td>
<td></td>
</tr>
<tr>
<td>\exists rewrite, thm</td>
<td>236</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\leq antisym, thm</td>
<td>448</td>
<td>456</td>
<td></td>
</tr>
<tr>
<td>\leq antisym, thm</td>
<td>462</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\leq cases, thm</td>
<td>448</td>
<td>456</td>
<td></td>
</tr>
<tr>
<td>\leq cases, thm</td>
<td>462</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\leq least_upper_bound, thm</td>
<td>448</td>
<td>456</td>
<td></td>
</tr>
<tr>
<td>\leq_least_upper_bound, thm</td>
<td>462</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\leq max, thm</td>
<td>395</td>
<td>452</td>
<td></td>
</tr>
<tr>
<td>\leq max, thm</td>
<td>494</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\leq plus1, thm</td>
<td>448</td>
<td>457</td>
<td></td>
</tr>
<tr>
<td>\leq plus1, thm</td>
<td>462</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\leq trans, thm</td>
<td>447</td>
<td>457</td>
<td></td>
</tr>
<tr>
<td>\leq trans, thm</td>
<td>462</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\leq well_order, thm</td>
<td>448</td>
<td>452</td>
<td></td>
</tr>
<tr>
<td>\leq well_order, thm</td>
<td>462</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\cup finite, thm</td>
<td>394</td>
<td>457</td>
<td></td>
</tr>
<tr>
<td>\cup finite, thm</td>
<td>494</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\beta rewrite, thm</td>
<td>184</td>
<td>453</td>
<td></td>
</tr>
<tr>
<td>\beta rewrite, thm</td>
<td>409</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\cup finite, thm</td>
<td>395</td>
<td>453</td>
<td></td>
</tr>
<tr>
<td>\cup finite, thm</td>
<td>494</td>
<td>455</td>
<td></td>
</tr>
<tr>
<td>\N_cases, thm</td>
<td>448</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\N_cases, thm</td>
<td>462</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\N_less_thm</td>
<td>433</td>
<td>455</td>
<td></td>
</tr>
<tr>
<td>\N_less_thm</td>
<td>508</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\N_less_thm</td>
<td>434</td>
<td>455</td>
<td></td>
</tr>
<tr>
<td>\N_less_thm</td>
<td>508</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\N_less_thm</td>
<td>433</td>
<td>455</td>
<td></td>
</tr>
<tr>
<td>\N_less_thm</td>
<td>508</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\N_less_thm</td>
<td>433</td>
<td>455</td>
<td></td>
</tr>
<tr>
<td>\N_less_thm</td>
<td>508</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\N_less_thm</td>
<td>433</td>
<td>455</td>
<td></td>
</tr>
<tr>
<td>\N_less_thm</td>
<td>508</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\N_less_thm</td>
<td>433</td>
<td>455</td>
<td></td>
</tr>
<tr>
<td>\N_less_thm</td>
<td>508</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\N_less_thm</td>
<td>433</td>
<td>455</td>
<td></td>
</tr>
<tr>
<td>\N_less_thm</td>
<td>508</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>\N_less_thm</td>
<td>433</td>
<td>455</td>
<td></td>
</tr>
<tr>
<td>\N_less_thm</td>
<td>508</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>KEYWORD</td>
<td>INDEX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NR., times</td>
<td>510</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_homomorphism</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>thm1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NZ., plus</td>
<td>454</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_homomorphism</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>thm1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NZ., times</td>
<td>477</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_homomorphism</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>thm1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NS., times</td>
<td>454</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_homomorphism</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>thm1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z., cases</td>
<td>453</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z., cases</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>thm1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z., eq</td>
<td>455</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thm1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z., less, less_0</td>
<td>455</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thm1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z., plus, assoc.</td>
<td>452</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thm1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z., plus, assoc.</td>
<td>477</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thm1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z., times, assoc.</td>
<td>454</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thm1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z., times, assoc.</td>
<td>477</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thm1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z., ≤, ≤_0</td>
<td>455</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thm1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dy., balance</td>
<td>391</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thm2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dy., times, mono.</td>
<td>391</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thm2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R., opah, extension</td>
<td>507</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thm2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R., plus, mono.</td>
<td>433</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thm2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R., plus, mono.</td>
<td>508</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thm2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add., rw</td>
<td>361</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add., sc</td>
<td>361</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add., st</td>
<td>362</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add., ∃, cd</td>
<td>362</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>current., ∃, cd</td>
<td>353</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>current., ∃, vs</td>
<td>354</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>get.,</td>
<td>143</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>get., ∃, cd</td>
<td>362</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>get., ∃, vs</td>
<td>363</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LS, Thms</td>
<td>77</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSAD, Thms</td>
<td>77</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pp., set, eval., ∃, cd</td>
<td>353</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pp., set, eval., ∃, vs</td>
<td>354</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set., ∃, cd</td>
<td>362</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set., ∃, vs</td>
<td>363</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subgoal., package</td>
<td>424</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ti., context</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>time_app</td>
<td>56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMED</td>
<td>56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMER_UNITS</td>
<td>56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R., times, 0, thm</td>
<td>440</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R., times, 0, thm</td>
<td>510</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R., times, 1, thm</td>
<td>440</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R., times, 1, thm</td>
<td>510</td>
<td></td>
<td></td>
</tr>
<tr>
<td>times, assoc, thm</td>
<td>448</td>
<td></td>
<td></td>
</tr>
<tr>
<td>times, assoc, thm1</td>
<td>462</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dy., times, assoc, thm</td>
<td>391</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R., times, assoc, thm</td>
<td>439</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R., times, assoc, thm</td>
<td>510</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z., times, assoc, thm</td>
<td>454</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z., times, assoc, thm</td>
<td>477</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R., times, assoc, thm1</td>
<td>440</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R., times, assoc, thm1</td>
<td>510</td>
<td></td>
<td></td>
</tr>
<tr>
<td>times, cancel, thm</td>
<td>440</td>
<td></td>
<td></td>
</tr>
<tr>
<td>times, cancel, thm1</td>
<td>477</td>
<td></td>
<td></td>
</tr>
<tr>
<td>times, cases</td>
<td>449</td>
<td></td>
<td></td>
</tr>
<tr>
<td>times, clauses</td>
<td>462</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R., times, clauses</td>
<td>440</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R., times, clauses</td>
<td>510</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z., times, clauses</td>
<td>455</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z., times, clauses</td>
<td>477</td>
<td></td>
<td></td>
</tr>
<tr>
<td>times, comm, thm</td>
<td>448</td>
<td></td>
<td></td>
</tr>
<tr>
<td>times, conv</td>
<td>462</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dy., times, conv</td>
<td>391</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R., times, conv</td>
<td>508</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z., times, conv</td>
<td>480</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z., N., conv</td>
<td>480</td>
<td></td>
<td></td>
</tr>
<tr>
<td>times, def</td>
<td>446</td>
<td></td>
<td></td>
</tr>
<tr>
<td>times, def</td>
<td>461</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R., times, def</td>
<td>509</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z., times, def</td>
<td>475</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dest., times</td>
<td>463</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dest., R., times</td>
<td>499</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dest., Z., times</td>
<td>475</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ry., times</td>
<td>390</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R., times, eq, 0, thm</td>
<td>440</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R., times, eq, 0, thm</td>
<td>510</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z., times, eq, 0, thm</td>
<td>455</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z., times, eq, 0, thm</td>
<td>477</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R., frac., frac, thm</td>
<td>442</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R., frac., frac, thm</td>
<td>505</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NR., times, homomorphism</td>
<td>440</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_thm,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NR., times, homomorphism</td>
<td>510</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_thm,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NZ., times, homomorphism</td>
<td>453</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_thm,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NZ., times, homomorphism</td>
<td>477</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_thm,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZR., times, homomorphism</td>
<td>444</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_thm,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZR., times, homomorphism</td>
<td>511</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_thm,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NR., times, homomorphism</td>
<td>441</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_thm1,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NR., times, homomorphism</td>
<td>510</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_thm1,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NR., times, homomorphism</td>
<td>454</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_thm1,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NZ., times, homomorphism</td>
<td>477</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_thm1,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>is., times</td>
<td>463</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUAL USR029
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
<th>Line</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>transitive_def</td>
<td>492</td>
<td></td>
<td></td>
</tr>
<tr>
<td>translate_for_output</td>
<td>68</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trich</td>
<td>411</td>
<td></td>
<td>BASIC_RES_TYPE</td>
</tr>
<tr>
<td>Trich</td>
<td>412</td>
<td></td>
<td>TYPE</td>
</tr>
<tr>
<td>cuts</td>
<td>413</td>
<td></td>
<td>TYPE</td>
</tr>
<tr>
<td>dy_less</td>
<td>391</td>
<td></td>
<td>basic.variant TYPE</td>
</tr>
<tr>
<td>induced_order</td>
<td>414</td>
<td></td>
<td>TYPE</td>
</tr>
<tr>
<td>fun</td>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRY</td>
<td>300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRY_C</td>
<td>219</td>
<td></td>
<td>New</td>
</tr>
<tr>
<td>THEN_</td>
<td>219</td>
<td></td>
<td>new</td>
</tr>
<tr>
<td>TRY_LIST_T</td>
<td>299</td>
<td></td>
<td>Delete</td>
</tr>
<tr>
<td>THEN_</td>
<td>299</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRY_T</td>
<td>300</td>
<td></td>
<td>DEST_SIMPLE_TYPE</td>
</tr>
<tr>
<td>THEN_</td>
<td>299</td>
<td></td>
<td>dest_simple TYPE</td>
</tr>
<tr>
<td>TRY_TTCL</td>
<td>300</td>
<td></td>
<td>dest_simple TYPE</td>
</tr>
<tr>
<td>THEN_</td>
<td>299</td>
<td></td>
<td>dest_simple TYPE</td>
</tr>
<tr>
<td>TRY_TTCL</td>
<td>299</td>
<td></td>
<td>dest_simple TYPE</td>
</tr>
<tr>
<td>TSAncestor</td>
<td>121</td>
<td></td>
<td>format TYPE</td>
</tr>
<tr>
<td>pp</td>
<td>407</td>
<td></td>
<td>get_constr TYPE</td>
</tr>
<tr>
<td>TSDeleted</td>
<td>121</td>
<td></td>
<td>ICL_DATABASE TYPE</td>
</tr>
<tr>
<td>TSNormal</td>
<td>121</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pp</td>
<td>241</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pp</td>
<td>407</td>
<td></td>
<td></td>
</tr>
<tr>
<td>all_submatch</td>
<td>154</td>
<td></td>
<td></td>
</tr>
<tr>
<td>all_substring</td>
<td>154</td>
<td></td>
<td></td>
</tr>
<tr>
<td>all_subterm</td>
<td>154</td>
<td></td>
<td></td>
</tr>
<tr>
<td>any_submatch</td>
<td>154</td>
<td></td>
<td></td>
</tr>
<tr>
<td>any_substring</td>
<td>154</td>
<td></td>
<td></td>
</tr>
<tr>
<td>any_subterm</td>
<td>154</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TTAXiom</td>
<td>153</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TTDfn</td>
<td>153</td>
<td></td>
<td></td>
</tr>
<tr>
<td>no_submatch</td>
<td>154</td>
<td></td>
<td></td>
</tr>
<tr>
<td>no_substring</td>
<td>154</td>
<td></td>
<td></td>
</tr>
<tr>
<td>no_subterm</td>
<td>154</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TTSaved</td>
<td>153</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EVERY</td>
<td>262</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIRST</td>
<td>264</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORELSE</td>
<td>277</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REPEAT</td>
<td>282</td>
<td></td>
<td></td>
</tr>
<tr>
<td>THEN_TTCM</td>
<td>299</td>
<td></td>
<td>RES_DB_TYPE</td>
</tr>
<tr>
<td>THEN_TRY</td>
<td>299</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRY_TTCL</td>
<td>300</td>
<td></td>
<td>asm_inst TYPE</td>
</tr>
<tr>
<td>HTAq</td>
<td>72</td>
<td></td>
<td>KInst TYPE</td>
</tr>
<tr>
<td>term</td>
<td>116</td>
<td></td>
<td>show TYPE</td>
</tr>
<tr>
<td>tycos</td>
<td>118</td>
<td></td>
<td>string_of TYPE</td>
</tr>
<tr>
<td>type</td>
<td>72</td>
<td></td>
<td>strip-&gt; TYPE</td>
</tr>
<tr>
<td>TYPE</td>
<td>82</td>
<td></td>
<td></td>
</tr>
<tr>
<td>declare</td>
<td>126</td>
<td></td>
<td></td>
</tr>
<tr>
<td>expand</td>
<td>126</td>
<td></td>
<td></td>
</tr>
<tr>
<td>get</td>
<td>128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is</td>
<td>129</td>
<td></td>
<td></td>
</tr>
<tr>
<td>undeclare</td>
<td>130</td>
<td></td>
<td></td>
</tr>
<tr>
<td>get</td>
<td>129</td>
<td></td>
<td></td>
</tr>
<tr>
<td>get_undeclared</td>
<td>129</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LS</td>
<td>77</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSUndeclared</td>
<td>77</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type_abbrev</td>
<td>126</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type_abbrev</td>
<td>126</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type_abbrev</td>
<td>128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type_abbrev</td>
<td>129</td>
<td></td>
<td></td>
</tr>
<tr>
<td>format</td>
<td>130</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type_abbrev</td>
<td>129</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pp</td>
<td>81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TypesAndTerms</td>
<td>81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>get</td>
<td>81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TypesAndTerms</td>
<td>81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Types</td>
<td>143</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Types</td>
<td>77</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type_abbrev</td>
<td>117</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type_arity</td>
<td>143</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYPE</td>
<td>325</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>135</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>468</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>404</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>405</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>405</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>458</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type_def</td>
<td>308</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type_def</td>
<td>133</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type_def</td>
<td>147</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>133</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>138</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>87</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>91</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>117</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>76</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICL_DATABASE_INFO</td>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>129</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>92</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>143</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>409</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>459</td>
<td></td>
<td></td>
</tr>
<tr>
<td>list_mk</td>
<td>99</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>117</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type_map</td>
<td>117</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type_match</td>
<td>117</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type_match</td>
<td>117</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mk_simple</td>
<td>106</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>108</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>133</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>147</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type_of</td>
<td>117</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type_order</td>
<td>323</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>325</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type_rule</td>
<td>179</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type_rule</td>
<td>193</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TypeRule</td>
<td>131</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>76</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>113</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>144</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type_theory</td>
<td>153</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYPE</td>
<td>153</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type_tycos</td>
<td>118</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type_tyvars</td>
<td>118</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>121</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type_1</td>
<td>76</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TypesAndTerms</td>
<td>81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TypesAndTerms</td>
<td>81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Types</td>
<td>143</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Types</td>
<td>77</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEYWORD INDEX</td>
<td>613</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>-----</td>
<td></td>
<td></td>
</tr>
<tr>
<td>valid_thm</td>
<td>150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>force_value</td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Var</td>
<td>81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D Var</td>
<td>82</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dest_var</td>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAR_ELIM_ASM_T</td>
<td>301</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL_VAR_ELIM_ASM_T</td>
<td>252</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL_VAR_ELIM_ASM_T1</td>
<td>252</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL_var_elim_asm_tac</td>
<td>301</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL_var_elim_asm_tac1</td>
<td>252</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAR_ELIM_NTH_ASM_T</td>
<td>301</td>
<td></td>
<td></td>
</tr>
<tr>
<td>get_shell_var</td>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_free_var</td>
<td>94</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_var</td>
<td>96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mk_var</td>
<td>108</td>
<td></td>
<td></td>
</tr>
<tr>
<td>var_subst</td>
<td>118</td>
<td></td>
<td></td>
</tr>
<tr>
<td>variant</td>
<td>118</td>
<td></td>
<td></td>
</tr>
<tr>
<td>list_variant</td>
<td>101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>string_variant</td>
<td>111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>get_variant_suffix</td>
<td>92</td>
<td></td>
<td></td>
</tr>
<tr>
<td>set_variant_suffix</td>
<td>111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>varstruct_variant</td>
<td>220</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gen_vars</td>
<td>92</td>
<td></td>
<td></td>
</tr>
<tr>
<td>term_vars</td>
<td>116</td>
<td></td>
<td></td>
</tr>
<tr>
<td>λ_varstruct_conv</td>
<td>240</td>
<td></td>
<td></td>
</tr>
<tr>
<td>λ_varstruct_variant</td>
<td>220</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vartype</td>
<td>81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dest_vartype</td>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_vartype</td>
<td>95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mk_vartype</td>
<td>107</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PP_Vector</td>
<td>44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HT_Vert</td>
<td>72</td>
<td></td>
<td></td>
</tr>
<tr>
<td>current_ad_3 vs_thms</td>
<td>354</td>
<td></td>
<td></td>
</tr>
<tr>
<td>get_3 vs_thms</td>
<td>363</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pp_set_eval_ad_3 vs_thms</td>
<td>354</td>
<td></td>
<td></td>
</tr>
<tr>
<td>set_3 vs_thms</td>
<td>363</td>
<td></td>
<td></td>
</tr>
<tr>
<td>warn</td>
<td>54</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ONCE_MAP_WARN_C</td>
<td>197</td>
<td></td>
<td></td>
</tr>
<tr>
<td>local_warn</td>
<td>66</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>54</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ilformed_rewrite_warning</td>
<td>173</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tactic_subgoal_warning</td>
<td>242</td>
<td></td>
<td></td>
</tr>
<tr>
<td>finite_⊆ well_founded_thm</td>
<td>395</td>
<td></td>
<td></td>
</tr>
<tr>
<td>finite_⊆ well_founded_thm</td>
<td>494</td>
<td></td>
<td></td>
</tr>
<tr>
<td>less_⊆ well_order_thm</td>
<td>448</td>
<td></td>
<td></td>
</tr>
<tr>
<td>less_⊆ well_order_thm</td>
<td>462</td>
<td></td>
<td></td>
</tr>
<tr>
<td>≤ well_order_thm</td>
<td>462</td>
<td></td>
<td></td>
</tr>
<tr>
<td>≤ well_order_thm</td>
<td>462</td>
<td></td>
<td></td>
</tr>
<tr>
<td>which</td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_white</td>
<td>66</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sym_White</td>
<td>59</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fail_with_canon</td>
<td>186</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fail_with_conv</td>
<td>186</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fail_with_tac</td>
<td>263</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>≤</td>
<td>accompanied by (\leq)</td>
<td>421</td>
</tr>
<tr>
<td>∈</td>
<td>accompanied by (\in)</td>
<td>422</td>
</tr>
<tr>
<td>(\in) clauses</td>
<td>accompanied by (\in) clauses</td>
<td>486</td>
</tr>
<tr>
<td>(\in) clauses</td>
<td>accompanied by (\in) clauses</td>
<td>486</td>
</tr>
<tr>
<td>(\in) def</td>
<td>accompanied by (\in) def</td>
<td>486</td>
</tr>
<tr>
<td>(\in) def</td>
<td>accompanied by (\in) def</td>
<td>486</td>
</tr>
<tr>
<td>(\in) def</td>
<td>accompanied by (\in) def</td>
<td>486</td>
</tr>
<tr>
<td>(\in) def</td>
<td>accompanied by (\in) def</td>
<td>486</td>
</tr>
<tr>
<td>(\in) finite_thm</td>
<td>accompanied by (\in) finite_thm</td>
<td>494</td>
</tr>
<tr>
<td>(\in) finite_thm</td>
<td>accompanied by (\in) finite_thm</td>
<td>494</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>420</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>420</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>(\in)</td>
<td>accompanied by (\in)</td>
<td>421</td>
</tr>
<tr>
<td>def</td>
<td>492</td>
<td></td>
</tr>
<tr>
<td>→</td>
<td>396</td>
<td></td>
</tr>
<tr>
<td>→</td>
<td>406</td>
<td></td>
</tr>
<tr>
<td>→</td>
<td>493</td>
<td></td>
</tr>
<tr>
<td>dest</td>
<td>89</td>
<td></td>
</tr>
<tr>
<td>→</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>→</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>→</td>
<td>108</td>
<td></td>
</tr>
<tr>
<td>→</td>
<td>113</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>127</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>428</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>497</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>508</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>441</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>507</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>440</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>510</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>441</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>507</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>440</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>510</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>442</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>505</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>501</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>442</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>505</td>
<td></td>
</tr>
<tr>
<td>dest</td>
<td>499</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>442</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>505</td>
<td></td>
</tr>
<tr>
<td>is</td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>442</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>505</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>501</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>435</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>507</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>435</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>507</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>436</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>507</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>436</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>507</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>435</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>507</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>502</td>
<td></td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textfrac{\text{minus}}{\text{frac}} \text{thm}</td>
<td>505</td>
</tr>
<tr>
<td>\textfrac{\text{minus}}{\text{frac}} \text{thm1}</td>
<td>442</td>
</tr>
<tr>
<td>\textfrac{\text{minus}}{\text{frac}} \text{thm1}</td>
<td>505</td>
</tr>
<tr>
<td>\textfrac{\text{frac}}{\text{norm}} \text{conv}</td>
<td>501</td>
</tr>
<tr>
<td>\textfrac{\text{plus}}{\text{frac}} \text{conv}</td>
<td>505</td>
</tr>
<tr>
<td>\textfrac{\text{plus}}{\text{frac}} \text{thm}</td>
<td>441</td>
</tr>
<tr>
<td>\textfrac{\text{plus}}{\text{frac}} \text{thm}</td>
<td>505</td>
</tr>
<tr>
<td>\textfrac{\text{recip}}{\text{thm}}</td>
<td>442</td>
</tr>
<tr>
<td>\textfrac{\text{recip}}{\text{thm}}</td>
<td>505</td>
</tr>
<tr>
<td>\textfrac{\text{times}}{\text{frac}} \text{thm}</td>
<td>442</td>
</tr>
<tr>
<td>\textfrac{\text{times}}{\text{frac}} \text{thm}</td>
<td>505</td>
</tr>
<tr>
<td>\textfrac{\text{N}}{\text{thm}}</td>
<td>441</td>
</tr>
<tr>
<td>\textfrac{\text{N}}{\text{thm}}</td>
<td>505</td>
</tr>
<tr>
<td>\text{greater} \text{conv}</td>
<td>506</td>
</tr>
<tr>
<td>\text{greater} \text{def}</td>
<td>509</td>
</tr>
<tr>
<td>\text{greater} \text{def}</td>
<td>509</td>
</tr>
<tr>
<td>\text{greater}</td>
<td>499</td>
</tr>
<tr>
<td>\text{greater}</td>
<td>500</td>
</tr>
<tr>
<td>\text{greater}</td>
<td>501</td>
</tr>
<tr>
<td>\text{halve} \text{closed} \text{dense} \text{thm}</td>
<td>438</td>
</tr>
<tr>
<td>\text{halve} \text{closed} \text{dense} \text{thm}</td>
<td>502</td>
</tr>
<tr>
<td>\text{less} \text{0} \text{less} \text{thm}</td>
<td>434</td>
</tr>
<tr>
<td>\text{less} \text{0} \text{less} \text{thm}</td>
<td>508</td>
</tr>
<tr>
<td>\text{less} \text{antisym} \text{thm}</td>
<td>430</td>
</tr>
<tr>
<td>\text{less} \text{antisym} \text{thm}</td>
<td>511</td>
</tr>
<tr>
<td>\text{less} \text{cancel} \text{conv}</td>
<td>504</td>
</tr>
<tr>
<td>\text{less} \text{cases} \text{thm}</td>
<td>430</td>
</tr>
<tr>
<td>\text{less} \text{cases} \text{thm}</td>
<td>511</td>
</tr>
<tr>
<td>\text{less} \text{clauses}</td>
<td>434</td>
</tr>
<tr>
<td>\text{less} \text{clauses}</td>
<td>508</td>
</tr>
<tr>
<td>\text{less} \text{conv}</td>
<td>508</td>
</tr>
<tr>
<td>\text{less} \text{def}</td>
<td>509</td>
</tr>
<tr>
<td>\text{less} \text{dense} \text{thm}</td>
<td>431</td>
</tr>
<tr>
<td>\text{less} \text{dense} \text{thm}</td>
<td>511</td>
</tr>
<tr>
<td>\text{less} \text{eq} \text{thm}</td>
<td>499</td>
</tr>
<tr>
<td>\text{less} \text{irrefl} \text{thm}</td>
<td>430</td>
</tr>
<tr>
<td>\text{less} \text{irrefl} \text{thm}</td>
<td>511</td>
</tr>
<tr>
<td>\text{less}</td>
<td>500</td>
</tr>
<tr>
<td>\text{less} \text{0} \text{less} \text{thm}</td>
<td>434</td>
</tr>
<tr>
<td>\text{less} \text{0} \text{less} \text{thm}</td>
<td>508</td>
</tr>
<tr>
<td>\text{less}</td>
<td>501</td>
</tr>
<tr>
<td>\text{less} \text{strong} \text{dense} \text{thm}</td>
<td>434</td>
</tr>
<tr>
<td>\text{less} \text{strong} \text{dense} \text{thm}</td>
<td>508</td>
</tr>
<tr>
<td>\text{less} \text{sup} \text{be} \text{thm}</td>
<td>431</td>
</tr>
<tr>
<td>\text{less} \text{sup} \text{be} \text{thm}</td>
<td>509</td>
</tr>
<tr>
<td>\text{less} \text{sup} \text{thm}</td>
<td>431</td>
</tr>
<tr>
<td>\text{less} \text{sup} \text{thm}</td>
<td>509</td>
</tr>
<tr>
<td>\text{less} \text{sup} \text{\in} \text{thm}</td>
<td>432</td>
</tr>
<tr>
<td>\text{less} \text{sup} \text{\in} \text{thm}</td>
<td>509</td>
</tr>
<tr>
<td>\text{less} \text{trans} \text{thm}</td>
<td>430</td>
</tr>
<tr>
<td>\text{less} \text{trans} \text{thm}</td>
<td>511</td>
</tr>
<tr>
<td>\text{less} \text{\less} \text{eq} \text{0} \text{thm}</td>
<td>441</td>
</tr>
<tr>
<td>\text{less} \text{\less} \text{eq} \text{0} \text{thm}</td>
<td>508</td>
</tr>
<tr>
<td>\text{less} \text{\less} \text{eq} \text{thm}</td>
<td>431</td>
</tr>
<tr>
<td>\text{less} \text{\less} \text{eq} \text{thm}</td>
<td>511</td>
</tr>
<tr>
<td>\text{less} \text{\leq} \text{trans} \text{thm}</td>
<td>430</td>
</tr>
<tr>
<td>\text{less} \text{\leq} \text{trans} \text{thm}</td>
<td>511</td>
</tr>
<tr>
<td>\text{lin} \text{arith}</td>
<td>498</td>
</tr>
<tr>
<td>\text{lin} \text{arith}</td>
<td>498</td>
</tr>
<tr>
<td>\text{lin} \text{arith} \text{prove} \text{tac}</td>
<td>506</td>
</tr>
<tr>
<td>\text{lin} \text{arith} \text{rule}</td>
<td>506</td>
</tr>
<tr>
<td>\text{lin} \text{arith} \text{tac}</td>
<td>506</td>
</tr>
<tr>
<td>\text{lin} \text{qep} \text{conv}</td>
<td>506</td>
</tr>
<tr>
<td>\text{max} \text{cons} \text{thm}</td>
<td>443</td>
</tr>
<tr>
<td>\text{max} \text{cons} \text{thm}</td>
<td>506</td>
</tr>
<tr>
<td>\text{max} \text{conv}</td>
<td>506</td>
</tr>
<tr>
<td>\text{max} \text{conv} \text{thm}</td>
<td>443</td>
</tr>
<tr>
<td>\text{max} \text{conv} \text{thm}</td>
<td>506</td>
</tr>
<tr>
<td>\text{max} \text{def}</td>
<td>509</td>
</tr>
<tr>
<td>\text{max}</td>
<td>499</td>
</tr>
<tr>
<td>\text{max}</td>
<td>500</td>
</tr>
<tr>
<td>\text{max}</td>
<td>501</td>
</tr>
<tr>
<td>\text{min} \text{cons} \text{thm}</td>
<td>443</td>
</tr>
<tr>
<td>\text{min} \text{cons} \text{thm}</td>
<td>506</td>
</tr>
<tr>
<td>\text{min} \text{conv}</td>
<td>506</td>
</tr>
<tr>
<td>\text{min} \text{conv} \text{thm}</td>
<td>443</td>
</tr>
<tr>
<td>\text{min} \text{conv} \text{thm}</td>
<td>506</td>
</tr>
<tr>
<td>\text{min} \text{def}</td>
<td>509</td>
</tr>
<tr>
<td>\text{min}</td>
<td>499</td>
</tr>
<tr>
<td>\text{min}</td>
<td>500</td>
</tr>
<tr>
<td>\text{min}</td>
<td>501</td>
</tr>
<tr>
<td>\text{minus} \text{clauses}</td>
<td>433</td>
</tr>
<tr>
<td>\text{minus} \text{clauses}</td>
<td>508</td>
</tr>
<tr>
<td>\text{minus} \text{conv}</td>
<td>507</td>
</tr>
<tr>
<td>\text{minus} \text{def}</td>
<td>509</td>
</tr>
<tr>
<td>\text{minus}</td>
<td>499</td>
</tr>
<tr>
<td>\text{minus} \text{eq} \text{thm}</td>
<td>433</td>
</tr>
<tr>
<td>\text{minus} \text{eq} \text{thm}</td>
<td>508</td>
</tr>
<tr>
<td>\text{minus} \text{frac} \text{less} \text{frac} \text{thm}</td>
<td>442</td>
</tr>
<tr>
<td>\text{minus} \text{frac} \text{less} \text{frac} \text{thm}</td>
<td>505</td>
</tr>
<tr>
<td>\text{minus}</td>
<td>500</td>
</tr>
<tr>
<td>\text{minus}</td>
<td>501</td>
</tr>
<tr>
<td>\text{minus} \text{recip} \text{thm}</td>
<td>442</td>
</tr>
<tr>
<td>\text{minus} \text{recip} \text{thm}</td>
<td>505</td>
</tr>
<tr>
<td>\text{monoid} \text{delta} \text{dense} \text{thm}</td>
<td>435</td>
</tr>
<tr>
<td>\text{monoid} \text{delta} \text{dense} \text{thm}</td>
<td>507</td>
</tr>
<tr>
<td>\text{monoid} \text{dense} \text{thm}</td>
<td>435</td>
</tr>
<tr>
<td>\text{monoid} \text{dense} \text{thm}</td>
<td>507</td>
</tr>
<tr>
<td>\text{opah} \text{complete} \text{thm}</td>
<td>439</td>
</tr>
<tr>
<td>\text{opah} \text{complete} \text{thm}</td>
<td>507</td>
</tr>
<tr>
<td>\text{opah} \text{dense} \text{image} \text{thm}</td>
<td>436</td>
</tr>
<tr>
<td>KEYWORD INDEX</td>
<td>621</td>
</tr>
<tr>
<td>---------------</td>
<td>-----</td>
</tr>
<tr>
<td>ALL_SIMPEL</td>
<td>3_C</td>
</tr>
<tr>
<td>evaluate</td>
<td>3_cd_thm</td>
</tr>
<tr>
<td>add</td>
<td>3_cd_thms</td>
</tr>
<tr>
<td>current_ad</td>
<td>3_cd_thms</td>
</tr>
<tr>
<td>get</td>
<td>3_cd_thms</td>
</tr>
<tr>
<td>pp/set_eval_ad</td>
<td>3_cd_thms</td>
</tr>
<tr>
<td>set</td>
<td>3_cd_thms</td>
</tr>
</tbody>
</table>
| simple        | 3_const_elim_conv | 337 | simple. | 3
| basic_prove_  | 3_conv | 364 | \( \land \) | simple. | 3
| basic_prove   | 3_conv | 160 | \( \lor \) | simple. | 3
| current_ad_cs | 3_conv | 353 | \( \forall \) | simple. | 3
| prove         | 3_conv | 355 | \( \exists \) | simple. | 3
| sets_simple   | 3_conv | 488 | \( \exists \) | simple. | 3
| simple. \( \forall \) | 3_conv | 209 | simple. | 3
| \( \neg \)    | 3_conv | 227 | \( \exists \) | simple. | 3
| \( \neg \)    | 3_conv | 226 | \( \exists \) | simple. | 3
| get_cs        | 3_conws | 358 | \( \exists \) | simple. | 3
| pp/set Eval_ad.cs | 3_conws | 353 | \( \exists \) | simple. | 3
| set_cs        | 3_conws | 358 | \( \exists \) | simple. | 3
<p>| dest           | 3 | 91 | dest. | 3 | 90 |
| dest_simple   | 3 | 88 | dest. | 3 | 87 |
| simple        | 3 | 233 | simpel. | 3 | 236 |
| simple        | 3 | 209 | simpel. | 3 | 211 |
| ( \neg )    | 3 | 96 | ( \neg ) | 3 | 96 |
| simple        | 3 | 233 | simpel. | 3 | 109 |
| ( \neg )    | 3 | 210 | simpel. | 3 | 106 |
| ( \neg )    | 3 | 233 | simpel. | 3_( \neg ) | 311 |
| ( \neg )    | 3 | 408 | simpel. | 3_( \neg ) | 289 |
| ( \neg )    | 3 | 220 | simpel. | 3_( \neg ) | 312 |
| ( \neg )    | 3 | 97 | SIMPLE. | 3_( \neg ) | 289 |
| ( \neg )    | 3 | 95 | SIMPLE. | 3_( \neg ) | 236 |
| ( \neg )    | 3 | 100 | SIMPLE. | 3_( \neg ) | 408 |
| ( \neg )    | 3 | 99 | SIMPLE. | 3_( \neg ) | 383 |
| ( \neg )    | 3 | 110 | SIMPLE. | 3_( \neg ) | 384 |
| ( \neg )    | 3 | 107 | SIMPLE. | 3_( \neg ) | 416 |
| ( \neg )    | 3 | 114 | SIMPLE. | 3_( \neg ) | 416 |
| ( \neg )    | 3 | 113 | SIMPLE. | 3_( \neg ) | 492 |
| ( \neg )    | 3 | 311 | SIMPLE. | 3_( \neg ) | 91 |
| ( \neg )    | 3 | 339 | ( \neg ) | 3_( \neg ) | 97 |
| ( \neg )    | 3 | 161 | ( \neg ) | 3_( \neg ) | 110 |
| ( \neg )    | 3 | 275 | ( \neg ) | 3_( \neg ) | 383 |
| ( \neg )    | 3 | 280 | ( \neg ) | 3_( \neg ) | 384 |
| ( \neg )    | 3 | 287 | ( \neg ) | 3_( \neg ) | 427 |
| ( \neg )    | 3 | 311 | ( \neg ) | 3_( \neg ) | 445 |
| ( \neg )    | 3 | 228 | ( \neg ) | 3_( \neg ) | 446 |
| ( \neg )    | 3 | 228 | ( \neg ) | 3_( \neg ) | 450 |
| ( \neg )    | 3 | 408 | ( \neg ) | 3_( \neg ) | 434 |
| ( \neg )    | 3 | 457 | ( \neg ) | 3_( \neg ) | 508 |
| ( \neg )    | 3 | 477 | ( \neg ) | 3_( \neg ) | 453 |
| ( \neg )    | 3 | 234 | ( \neg ) | 3_( \neg ) | 477 |
| ( \neg )    | 3 | 234 | ( \neg ) | 3_( \neg ) | 455 |</p>
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z,\leq_0,\leq_0_thm1</td>
<td>477</td>
</tr>
<tr>
<td>R,\leq_0,\leq_\thm</td>
<td>434</td>
</tr>
<tr>
<td>R,\leq_0_\leq_\thm</td>
<td>508</td>
</tr>
<tr>
<td>R,\leq_0,\leq_\times_\thm</td>
<td>440</td>
</tr>
<tr>
<td>R,\leq_0,\leq_\times_\thm</td>
<td>510</td>
</tr>
<tr>
<td>R,\leq_\times_\thm</td>
<td>448</td>
</tr>
<tr>
<td>R,\leq_\times_\thm</td>
<td>462</td>
</tr>
<tr>
<td>R,\leq_\thm</td>
<td>430</td>
</tr>
<tr>
<td>R,\leq_\thm</td>
<td>511</td>
</tr>
<tr>
<td>Z,\leq_\thm</td>
<td>455</td>
</tr>
<tr>
<td>Z,\leq_\thm</td>
<td>477</td>
</tr>
<tr>
<td>R,\leq_\times_\thm</td>
<td>509</td>
</tr>
<tr>
<td>R,\leq_\times_\thm</td>
<td>473</td>
</tr>
<tr>
<td>R,\leq_\times_\thm</td>
<td>504</td>
</tr>
<tr>
<td>Z,\leq_\times_\thm</td>
<td>483</td>
</tr>
<tr>
<td>R,\leq_\times_\thm</td>
<td>448</td>
</tr>
<tr>
<td>R,\leq_\times_\thm</td>
<td>462</td>
</tr>
<tr>
<td>R,\leq_\times_\thm</td>
<td>430</td>
</tr>
<tr>
<td>Z,\leq_\times_\thm</td>
<td>511</td>
</tr>
<tr>
<td>Z,\leq_\times_\thm</td>
<td>455</td>
</tr>
<tr>
<td>Z,\leq_\times_\thm</td>
<td>477</td>
</tr>
<tr>
<td>R,\leq_\times_\thm</td>
<td>477</td>
</tr>
<tr>
<td>R,\leq_\times_\thm</td>
<td>447</td>
</tr>
<tr>
<td>R,\leq_\times_\thm</td>
<td>462</td>
</tr>
<tr>
<td>R,\leq_\times_\thm</td>
<td>434</td>
</tr>
<tr>
<td>R,\leq_\times_\thm</td>
<td>508</td>
</tr>
<tr>
<td>Z,\leq_\times_\thm</td>
<td>453</td>
</tr>
<tr>
<td>Z,\leq_\times_\thm</td>
<td>477</td>
</tr>
<tr>
<td>R,\leq_\times_\thm</td>
<td>465</td>
</tr>
<tr>
<td>Z,\leq_\times_\thm</td>
<td>470</td>
</tr>
<tr>
<td>Z,\leq_\times_\thm</td>
<td>480</td>
</tr>
<tr>
<td>Z,\leq_\times_\thm</td>
<td>480</td>
</tr>
<tr>
<td>R,\leq_\times_\thm</td>
<td>446</td>
</tr>
<tr>
<td>R,\leq_\times_\thm</td>
<td>461</td>
</tr>
<tr>
<td>Z,\leq_\times_\thm</td>
<td>509</td>
</tr>
<tr>
<td>dest,\leq_0,\leq_0,\leq_0,\leq_0</td>
<td>463</td>
</tr>
<tr>
<td>dest,\leq_0,\leq_0,\leq_0,\leq_0</td>
<td>499</td>
</tr>
<tr>
<td>dest,\leq_0,\leq_0,\leq_0,\leq_0</td>
<td>475</td>
</tr>
<tr>
<td>R,\leq_\frac_\thm</td>
<td>442</td>
</tr>
<tr>
<td>R,\leq_\frac_\thm</td>
<td>505</td>
</tr>
<tr>
<td>Z,\leq_\induction_tac</td>
<td>479</td>
</tr>
<tr>
<td>Z,\leq_\induction_\thm</td>
<td>456</td>
</tr>
<tr>
<td>Z,\leq_\induction_\thm</td>
<td>477</td>
</tr>
<tr>
<td>is,\leq_0,\leq_0,\leq_0,\leq_0</td>
<td>463</td>
</tr>
<tr>
<td>is,\leq_0,\leq_0,\leq_0,\leq_0</td>
<td>500</td>
</tr>
<tr>
<td>is,\leq_0,\leq_0,\leq_0,\leq_0</td>
<td>476</td>
</tr>
<tr>
<td>is,\leq_0,\leq_0,\leq_0,\leq_0</td>
<td>448</td>
</tr>
<tr>
<td>\leq_\leastupperbound_\thm</td>
<td>462</td>
</tr>
<tr>
<td>\leq_\leastupperbound_\thm</td>
<td>430</td>
</tr>
<tr>
<td>\leq_\leastupperbound_\thm</td>
<td>511</td>
</tr>
<tr>
<td>\leq_\leastupperbound_\thm</td>
<td>455</td>
</tr>
<tr>
<td>\leq_\leastupperbound_\thm</td>
<td>477</td>
</tr>
<tr>
<td>\leq_\leastupperbound_\thm</td>
<td>431</td>
</tr>
<tr>
<td>\leq_\leastupperbound_\thm</td>
<td>511</td>
</tr>
<tr>
<td>\leq_\leastupperbound_\thm</td>
<td>431</td>
</tr>
<tr>
<td>\leq_\leastupperbound_\thm</td>
<td>431</td>
</tr>
<tr>
<td>\leq_\leastupperbound_\thm</td>
<td>431</td>
</tr>
<tr>
<td>Keyword</td>
<td>Line</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td><code>R, ≤_trans_thm</code></td>
<td>511</td>
</tr>
<tr>
<td><code>R_less, ≤_trans_thm</code></td>
<td>430</td>
</tr>
<tr>
<td><code>R_less, ≤_trans_thm</code></td>
<td>511</td>
</tr>
<tr>
<td><code>z, ≤_trans_thm</code></td>
<td>455</td>
</tr>
<tr>
<td><code>Z, ≤_trans_thm</code></td>
<td>477</td>
</tr>
<tr>
<td><code>R, ≤_less_thm</code></td>
<td>431</td>
</tr>
<tr>
<td><code>R, ≤_less_thm</code></td>
<td>511</td>
</tr>
<tr>
<td><code>R, ≤_0_thm</code></td>
<td>434</td>
</tr>
<tr>
<td><code>R, ≤_0_thm</code></td>
<td>508</td>
</tr>
<tr>
<td><code>z, ≤_0_thm</code></td>
<td>453</td>
</tr>
<tr>
<td><code>Z, ≤_0_thm</code></td>
<td>477</td>
</tr>
<tr>
<td><code>R, ≤_0_thm1</code></td>
<td>477</td>
</tr>
<tr>
<td><code>R, ≤_R</code></td>
<td>427</td>
</tr>
<tr>
<td><code>R, ≤_R</code></td>
<td>428</td>
</tr>
<tr>
<td><code>R, Z, ≤_Z</code></td>
<td>450</td>
</tr>
<tr>
<td><code>R, Z, ≤_Z</code></td>
<td>451</td>
</tr>
<tr>
<td><code>Z, ≥_R</code></td>
<td>427</td>
</tr>
<tr>
<td><code>Z, ≥_R</code></td>
<td>428</td>
</tr>
<tr>
<td><code>Z, ≥_Z</code></td>
<td>450</td>
</tr>
<tr>
<td><code>Z, ≥_Z</code></td>
<td>451</td>
</tr>
<tr>
<td><code>≥_conv, R</code></td>
<td>465</td>
</tr>
<tr>
<td><code>≥_conv, R</code></td>
<td>466</td>
</tr>
<tr>
<td><code>≥_def, R</code></td>
<td>466</td>
</tr>
<tr>
<td><code>≥_def, R</code></td>
<td>467</td>
</tr>
<tr>
<td><code>≥_def, Z</code></td>
<td>475</td>
</tr>
<tr>
<td><code>≥_def, Z</code></td>
<td>476</td>
</tr>
<tr>
<td><code>dest, ≥_R</code></td>
<td>463</td>
</tr>
<tr>
<td><code>dest, ≥_R</code></td>
<td>469</td>
</tr>
<tr>
<td><code>dest, ≥_Z</code></td>
<td>475</td>
</tr>
<tr>
<td><code>dest, ≥_Z</code></td>
<td>476</td>
</tr>
<tr>
<td><code>is, ≥_R</code></td>
<td>463</td>
</tr>
<tr>
<td><code>is, ≥_R</code></td>
<td>464</td>
</tr>
<tr>
<td><code>is, ≥_Z</code></td>
<td>475</td>
</tr>
<tr>
<td><code>is, ≥_Z</code></td>
<td>476</td>
</tr>
<tr>
<td><code>mk, ≥_R</code></td>
<td>464</td>
</tr>
<tr>
<td><code>mk, ≥_R</code></td>
<td>465</td>
</tr>
<tr>
<td><code>mk, ≥_Z</code></td>
<td>476</td>
</tr>
<tr>
<td><code>mk, ≥_Z</code></td>
<td>477</td>
</tr>
<tr>
<td><code>≥_&lt;conv, Z</code></td>
<td>480</td>
</tr>
<tr>
<td><code>≥_R</code></td>
<td>427</td>
</tr>
<tr>
<td><code>≥_R</code></td>
<td>428</td>
</tr>
<tr>
<td><code>≥_Z</code></td>
<td>450</td>
</tr>
<tr>
<td><code>≥_Z</code></td>
<td>451</td>
</tr>
<tr>
<td><code>∪</code></td>
<td>420</td>
</tr>
<tr>
<td><code>∪</code></td>
<td>421</td>
</tr>
<tr>
<td><code>∪_clauses</code></td>
<td>422</td>
</tr>
<tr>
<td><code>∪_clauses</code></td>
<td>486</td>
</tr>
<tr>
<td><code>∪_def</code></td>
<td>421</td>
</tr>
<tr>
<td><code>∪_def</code></td>
<td>486</td>
</tr>
<tr>
<td><code>∪_finite_thm</code></td>
<td>394</td>
</tr>
<tr>
<td><code>∪_finite_thm</code></td>
<td>404</td>
</tr>
<tr>
<td><code>∪_finite_thm</code></td>
<td>394</td>
</tr>
<tr>
<td><code>∪_finite_thm</code></td>
<td>394</td>
</tr>
<tr>
<td><code>∪_thm</code></td>
<td>394</td>
</tr>
<tr>
<td><code>∪_thm</code></td>
<td>394</td>
</tr>
<tr>
<td><code>size Singleton</code></td>
<td>394</td>
</tr>
<tr>
<td><code>size Singleton</code></td>
<td>394</td>
</tr>
<tr>
<td><code>singleton</code></td>
<td>394</td>
</tr>
<tr>
<td><code>singleton</code></td>
<td>394</td>
</tr>
</tbody>
</table>
KEYWORD INDEX

<table>
<thead>
<tr>
<th>thm</th>
<th>420</th>
</tr>
</thead>
<tbody>
<tr>
<td>plus</td>
<td>421</td>
</tr>
<tr>
<td>plus1</td>
<td>486</td>
</tr>
<tr>
<td>minus</td>
<td>486</td>
</tr>
<tr>
<td>_finite_thm</td>
<td>395</td>
</tr>
<tr>
<td>_finite_thm</td>
<td>494</td>
</tr>
<tr>
<td>→</td>
<td>396</td>
</tr>
<tr>
<td>→_def</td>
<td>493</td>
</tr>
<tr>
<td>←</td>
<td>396</td>
</tr>
<tr>
<td>←_def</td>
<td>493</td>
</tr>
<tr>
<td>≤</td>
<td>383</td>
</tr>
<tr>
<td>≤_def</td>
<td>492</td>
</tr>
<tr>
<td>∈</td>
<td>393</td>
</tr>
<tr>
<td>∈_def</td>
<td>494</td>
</tr>
<tr>
<td>∈_t</td>
<td>494</td>
</tr>
<tr>
<td>∈_t_def</td>
<td>383</td>
</tr>
<tr>
<td>/\</td>
<td>384</td>
</tr>
<tr>
<td>/\</td>
<td>384</td>
</tr>
<tr>
<td>/\</td>
<td>383</td>
</tr>
<tr>
<td>/\</td>
<td>383</td>
</tr>
<tr>
<td>/\</td>
<td>384</td>
</tr>
<tr>
<td>/\</td>
<td>384</td>
</tr>
<tr>
<td>¬</td>
<td>418</td>
</tr>
<tr>
<td>¬_def</td>
<td>495</td>
</tr>
<tr>
<td>¬→</td>
<td>383</td>
</tr>
<tr>
<td>¬→</td>
<td>384</td>
</tr>
<tr>
<td>¬→_def</td>
<td>492</td>
</tr>
<tr>
<td>( R )</td>
<td>118</td>
</tr>
<tr>
<td>( N )</td>
<td>393</td>
</tr>
<tr>
<td>( N )</td>
<td>445</td>
</tr>
<tr>
<td>( N )</td>
<td>446</td>
</tr>
<tr>
<td>( N )</td>
<td>459</td>
</tr>
<tr>
<td>( N )</td>
<td>369</td>
</tr>
<tr>
<td>( Z )</td>
<td>456</td>
</tr>
<tr>
<td>( Z )</td>
<td>477</td>
</tr>
<tr>
<td>( Z )</td>
<td>456</td>
</tr>
<tr>
<td>( Z )</td>
<td>477</td>
</tr>
<tr>
<td>( N )</td>
<td>448</td>
</tr>
<tr>
<td>( N )</td>
<td>462</td>
</tr>
<tr>
<td>( Z )</td>
<td>454</td>
</tr>
<tr>
<td>( Z )</td>
<td>477</td>
</tr>
<tr>
<td>( N )</td>
<td>446</td>
</tr>
<tr>
<td>( N )</td>
<td>461</td>
</tr>
<tr>
<td>( N )</td>
<td>494</td>
</tr>
<tr>
<td>dest_</td>
<td>91</td>
</tr>
<tr>
<td>( N )</td>
<td>473</td>
</tr>
<tr>
<td>( N )</td>
<td>465</td>
</tr>
<tr>
<td>( R )</td>
<td>508</td>
</tr>
<tr>
<td>( R )</td>
<td>509</td>
</tr>
<tr>
<td>dest_R</td>
<td>499</td>
</tr>
<tr>
<td>is_R</td>
<td>500</td>
</tr>
<tr>
<td>mk_R</td>
<td>501</td>
</tr>
<tr>
<td>( Z )</td>
<td>479</td>
</tr>
<tr>
<td>( Z )</td>
<td>454</td>
</tr>
<tr>
<td>( Z )</td>
<td>477</td>
</tr>
</tbody>
</table>

is_ | 97 |
\( N \) | 370 |
\( mk_ \) | 110 |
\( Z \) | 480 |
\( Z \) | 454 |
\( Z \) | 477 |
\( Z \) | 454 |
\( Z \) | 477 |
\( is_ \) | 446 |
\( is_ \) | 461 |
\( Is_ \) | 445 |
\( Is_ \) | 446 |
\( R \) | 441 |
\( N \) | 505 |
\( Z \) | 456 |
\( Z \) | 477 |
\( Z \) | 455 |
\( Z \) | 477 |
\( Z \) | 454 |
\( Z \) | 477 |
\( Z \) | 455 |
\( Z \) | 477 |
\( Z \) | 477 |

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z_eq_thm</td>
<td>452</td>
</tr>
<tr>
<td>Z_eq_thm</td>
<td>477</td>
</tr>
<tr>
<td>Z_eq_thm</td>
<td>455</td>
</tr>
<tr>
<td>Z_eq_thm</td>
<td>477</td>
</tr>
<tr>
<td>dest</td>
<td>Z_mod</td>
</tr>
<tr>
<td>is</td>
<td>Z_mod</td>
</tr>
<tr>
<td>mk</td>
<td>Z_mod</td>
</tr>
<tr>
<td>dest</td>
<td>Z_plus_assoc_thm</td>
</tr>
<tr>
<td>is</td>
<td>Z_plus_assoc_thm</td>
</tr>
<tr>
<td>mk</td>
<td>Z_plus_assoc_thm</td>
</tr>
<tr>
<td>dest</td>
<td>Z_plus_assoc_thm</td>
</tr>
<tr>
<td>is</td>
<td>Z_plus_assoc_thm1</td>
</tr>
<tr>
<td>mk</td>
<td>Z_plus_assoc_thm1</td>
</tr>
<tr>
<td>dest</td>
<td>Z_plus_assoc_thm1</td>
</tr>
<tr>
<td>is</td>
<td>Z_plus_clauses</td>
</tr>
<tr>
<td>mk</td>
<td>Z_plus_clauses</td>
</tr>
<tr>
<td>dest</td>
<td>Z_plus_clauses</td>
</tr>
<tr>
<td>is</td>
<td>Z_plus_conv</td>
</tr>
<tr>
<td>mk</td>
<td>Z_plus_def</td>
</tr>
<tr>
<td>dest</td>
<td>Z_plus_def</td>
</tr>
<tr>
<td>is</td>
<td>Z_plus_eq_thm</td>
</tr>
<tr>
<td>mk</td>
<td>Z_plus_eq_thm</td>
</tr>
<tr>
<td>dest</td>
<td>Z_plus_eq_thm</td>
</tr>
<tr>
<td>is</td>
<td>Z_plus_minus_thm</td>
</tr>
<tr>
<td>mk</td>
<td>Z_plus_minus_thm</td>
</tr>
<tr>
<td>dest</td>
<td>Z_plus_minus_thm</td>
</tr>
<tr>
<td>is</td>
<td>Z_plus_minus_thm</td>
</tr>
<tr>
<td>mk</td>
<td>Z_plus0_thm</td>
</tr>
<tr>
<td>dest</td>
<td>Z_plus0_thm</td>
</tr>
<tr>
<td>is</td>
<td>Z_rep_consistent_thm</td>
</tr>
<tr>
<td>Is</td>
<td>Z_Rep_def</td>
</tr>
<tr>
<td>Is</td>
<td>Z_Rep</td>
</tr>
<tr>
<td>Is</td>
<td>Z_Rep</td>
</tr>
<tr>
<td>dest</td>
<td>Z_signed_int</td>
</tr>
<tr>
<td>is</td>
<td>Z_signed_int</td>
</tr>
<tr>
<td>mk</td>
<td>Z_signed_int</td>
</tr>
<tr>
<td>dest</td>
<td>Z_subtract</td>
</tr>
<tr>
<td>is</td>
<td>Z_subtract</td>
</tr>
<tr>
<td>mk</td>
<td>Z_subtract</td>
</tr>
<tr>
<td>dest</td>
<td>Z_subtract</td>
</tr>
<tr>
<td>is</td>
<td>Z_subtract</td>
</tr>
<tr>
<td>mk</td>
<td>Z_subtract</td>
</tr>
<tr>
<td>dest</td>
<td>Z_term_order</td>
</tr>
<tr>
<td>is</td>
<td>Z_times_assoc_thm</td>
</tr>
<tr>
<td>mk</td>
<td>Z_times_assoc_thm</td>
</tr>
<tr>
<td>dest</td>
<td>Z_times_assoc_thm</td>
</tr>
<tr>
<td>is</td>
<td>Z_times_assoc_thm1</td>
</tr>
<tr>
<td>mk</td>
<td>Z_times_assoc_thm1</td>
</tr>
<tr>
<td>dest</td>
<td>Z_times_assoc_thm1</td>
</tr>
<tr>
<td>is</td>
<td>Z_times_clauses</td>
</tr>
<tr>
<td>mk</td>
<td>Z_times_clauses</td>
</tr>
<tr>
<td>dest</td>
<td>Z_times_clauses</td>
</tr>
<tr>
<td>is</td>
<td>Z_times_conv</td>
</tr>
<tr>
<td>mk</td>
<td>Z_times_def</td>
</tr>
<tr>
<td>dest</td>
<td>Z_times</td>
</tr>
<tr>
<td>is</td>
<td>Z_times</td>
</tr>
<tr>
<td>mk</td>
<td>Z_times</td>
</tr>
<tr>
<td>dest</td>
<td>Z_times</td>
</tr>
<tr>
<td>is</td>
<td>Z_times</td>
</tr>
<tr>
<td>mk</td>
<td>Z_times</td>
</tr>
<tr>
<td>dest</td>
<td>Z_times</td>
</tr>
<tr>
<td>is</td>
<td>Z_times</td>
</tr>
<tr>
<td>mk</td>
<td>Z_times</td>
</tr>
<tr>
<td>dest</td>
<td>Z_times</td>
</tr>
<tr>
<td>is</td>
<td>Z_times</td>
</tr>
<tr>
<td>mk</td>
<td>Z_times</td>
</tr>
<tr>
<td>dest</td>
<td>Z_times</td>
</tr>
<tr>
<td>is</td>
<td>Z_times</td>
</tr>
<tr>
<td>mk</td>
<td>Z_times</td>
</tr>
<tr>
<td>dest</td>
<td>Z_times</td>
</tr>
<tr>
<td>is</td>
<td>Z_times</td>
</tr>
<tr>
<td>mk</td>
<td>Z_times</td>
</tr>
<tr>
<td>dest</td>
<td>Z_times</td>
</tr>
<tr>
<td>is</td>
<td>Z_times</td>
</tr>
<tr>
<td>mk</td>
<td>Z_times</td>
</tr>
<tr>
<td>dest</td>
<td>Z_times</td>
</tr>
<tr>
<td>is</td>
<td>Z_times</td>
</tr>
<tr>
<td>mk</td>
<td>Z_times</td>
</tr>
<tr>
<td>dest</td>
<td>Z_times</td>
</tr>
<tr>
<td>is</td>
<td>Z_times</td>
</tr>
<tr>
<td>mk</td>
<td>Z_times</td>
</tr>
<tr>
<td>dest</td>
<td>Z_times</td>
</tr>
<tr>
<td>is</td>
<td>Z_times</td>
</tr>
<tr>
<td>mk</td>
<td>Z_times</td>
</tr>
<tr>
<td>dest</td>
<td>Z_times</td>
</tr>
<tr>
<td>is</td>
<td>Z_times</td>
</tr>
<tr>
<td>mk</td>
<td>Z_times</td>
</tr>
<tr>
<td>dest</td>
<td>Z_times</td>
</tr>
<tr>
<td>is</td>
<td>Z_times</td>
</tr>
<tr>
<td>mk</td>
<td>Z_times</td>
</tr>
<tr>
<td>dest</td>
<td>Z_times</td>
</tr>
<tr>
<td>is</td>
<td>Z_times</td>
</tr>
<tr>
<td>mk</td>
<td>Z_times</td>
</tr>
<tr>
<td>Keyword</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>Z _times_plus_distrib _thm</td>
<td>455</td>
</tr>
<tr>
<td>Z _times_plus_distrib _thm</td>
<td>477</td>
</tr>
<tr>
<td>Z _times0 _thm</td>
<td>455</td>
</tr>
<tr>
<td>Z _times0 _thm</td>
<td>477</td>
</tr>
<tr>
<td>Z _times1 _thm</td>
<td>455</td>
</tr>
<tr>
<td>Z _times1 _thm</td>
<td>477</td>
</tr>
<tr>
<td>Z ∈ N _thm</td>
<td>455</td>
</tr>
<tr>
<td>Z ∈ N _thm</td>
<td>477</td>
</tr>
<tr>
<td>Z _¬ less _thm</td>
<td>455</td>
</tr>
<tr>
<td>Z _¬ less _thm</td>
<td>477</td>
</tr>
<tr>
<td>Z _¬ ≤ _thm</td>
<td>455</td>
</tr>
<tr>
<td>Z _¬ ≤ _thm</td>
<td>477</td>
</tr>
<tr>
<td>Z _¬ N _thm</td>
<td>454</td>
</tr>
<tr>
<td>Z _¬ N _thm</td>
<td>477</td>
</tr>
<tr>
<td>Z _¬ N _thm</td>
<td>477</td>
</tr>
<tr>
<td>Z _¬ ≤ _ Antisym _thm</td>
<td>457</td>
</tr>
<tr>
<td>Z _¬ ≤ _ Antisym _thm</td>
<td>477</td>
</tr>
<tr>
<td>Z _¬ ≤ _ Cancel _conv</td>
<td>483</td>
</tr>
<tr>
<td>Z _¬ ≤ _ Cases _thm</td>
<td>455</td>
</tr>
<tr>
<td>Z _¬ ≤ _ Cases _thm</td>
<td>477</td>
</tr>
<tr>
<td>Z _¬ ≤ _ Clauses</td>
<td>453</td>
</tr>
<tr>
<td>Z _¬ ≤ _ Clauses</td>
<td>477</td>
</tr>
<tr>
<td>Z _¬ ≤ _ Conv</td>
<td>480</td>
</tr>
<tr>
<td>Z _¬ ≤ _ Def</td>
<td>475</td>
</tr>
<tr>
<td>dest _</td>
<td>475</td>
</tr>
<tr>
<td>is _</td>
<td>475</td>
</tr>
<tr>
<td>mk _</td>
<td>475</td>
</tr>
<tr>
<td>dest _</td>
<td>475</td>
</tr>
<tr>
<td>is _</td>
<td>476</td>
</tr>
<tr>
<td>mk _</td>
<td>476</td>
</tr>
</tbody>
</table>

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - HOL REFERENCE MANUALUSR029