The Isabelle System Manual

Markus Wenzel and *Stefan Berghofer*
TU München

November 22, 2007

Contents

Chapter 1

The Isabelle system environment

This manual describes Isabelle together with related tools and user interfaces as seen from an outside (system oriented) view. See also the *Isabelle/Isar Reference Manual* [3] and the *Isabelle Reference Manual* [2] for the actual Isabelle commands and related functions.

The Isabelle system environment emerges from a few general concepts.

- The *Isabelle settings mechanism* provides environment variables to all Isabelle programs (including tools and user interfaces).
- The *Isabelle tools wrapper* (`isatool`) provides a generic startup platform for Isabelle related utilities. Thus tools automatically benefit from the settings mechanism.
- The raw *Isabelle process* (`isabelle` or `isabelle-process`) runs logic sessions either interactively or in batch mode. In particular, this view abstracts over the invocation of the actual ML system to be used.
- The *Isabelle interface wrapper* (`Isabelle` or `isabelle-interface`) provides some abstraction over the actual user interface to be used. The de-facto standard interface for Isabelle is Proof General [1].

The beginning user would probably just run the default user interface (by invoking the capital `Isabelle`). This assumes that the system has already been installed, of course. In case you have to do this yourself, see the `INSTALL` file in the top-level directory of the distribution of how to proceed; binary packages for various system components are available as well.

1.1 Isabelle settings

The Isabelle system heavily depends on the *settings mechanism*. Essentially, this is a statically scoped collection of environment variables, such as

`ISABELLE_HOME`, `ML_SYSTEM`, `ML_HOME`. These variables are *not* intended to be set directly from the shell, though. Isabelle employs a somewhat more sophisticated scheme of *settings files* — one for site-wide defaults, another for additional user-specific modifications. With all configuration variables in at most two places, this scheme is more maintainable and user-friendly than global shell environment variables.

In particular, we avoid the typical situation where prospective users of a software package are told to put several things into their shell startup scripts, before being able to actually run the program. Isabelle requires none such administrative chores of its end-users — the executables can be invoked straight away.[1]

Building the environment

Whenever any of the Isabelle executables is run, their settings environment is put together as follows.

1. The special variable `ISABELLE_HOME` is determined automatically from the location of the binary that has been run.

   You should not try to set `ISABELLE_HOME` manually. Also note that the Isabelle executables either have to be run from their original location in the distribution directory, or via the executable objects created by the `install` utility (see §3.7). Just doing a plain copy of the `bin` files will not work!

2. The file `$ISABELLE_HOME/etc/settings` ist run as a shell script with the auto-export option for variables enabled.

   This file holds a rather long list of shell variable assigments, thus providing the site-wide default settings. The Isabelle distribution already contains a global settings file with sensible defaults for most variables. When installing the system, only a few of these may have to be adapted (probably `ML_SYSTEM` etc.).

3. The file `$ISABELLE_HOME_USER/etc/settings` (if it exists) is run in the same way as the site default settings. Note that the variable `ISABELLE_HOME_USER` has already been set before — usually to `~/isabelle`.

   Thus individual users may override the site-wide defaults. See also file `etc/user-settings.sample` in the distribution. Typically, a user

[1] Occasionally, users would still want to put the Isabelle `bin` directory into their shell's search path, but this is not required.

settings file would contain only a few lines, just the assigments that are really changed. One should definitely *not* start with a full copy the basic `$ISABELLE_HOME/etc/settings`. This could cause very annoying maintainance problems later, when the Isabelle installation is updated or changed otherwise.

Note that settings files are actually full GNU bash scripts. So one may use complex shell commands, such as `if` or `case` statements to set variables depending on the system architecture or other environment variables. Such advanced features should be added only with great care, though. In particular, external environment references should be kept at a minimum.

A few variables are somewhat special:

- `ISABELLE` and `ISATOOL` are set automatically to the absolute path names of the `isabelle-process` and `isatool` executables, respectively.
- `ISABELLE_OUTPUT` will have the identifiers of the Isabelle distribution (cf. `ISABELLE_IDENTIFIER`) and the ML system (cf. `ML_IDENTIFIER`) appended automatically to its value.

The Isabelle settings scheme is conceptually simple, but not completely trivial. For debugging purposes, the resulting environment may be inspected with the `getenv` utility, see §3.6.

Common variables

This is a reference of common Isabelle settings variables. Note that the list is somewhat open-ended. Third-party utilities or interfaces may add their own selection. Variables that are special in some sense are marked with *.

`ISABELLE_HOME*` is the location of the top-level Isabelle distribution directory. This is automatically determined from the Isabelle executable that has been invoked. Do not attempt to set `ISABELLE_HOME` yourself from the shell.

`ISABELLE_HOME_USER` is the user-specific counterpart of `ISABELLE_HOME`. The default value is `~/isabelle`, under rare circumstances this may be changed in the global setting file. Typically, the `ISABELLE_HOME_USER` directory mimics `ISABELLE_HOME` to some extend. In particular, site-wide defaults may be overridden by a private `etc/settings`.

`ISABELLE*`, `ISATOOL*` are automatically set to the full path names of the `isabelle-process` and `isatool` executables, respectively. Thus other tools and scripts need not assume that the Isabelle `bin` directory is on the current search path of the shell.

`ISABELLE_IDENTIFIER*` refers to the name of this Isabelle distribution, e.g. "`Isabelle2007`".

`ML_SYSTEM`, `ML_HOME`, `ML_OPTIONS`, `ML_PLATFORM`, `ML_IDENTIFIER*` specify the underlying ML system to be used for Isabelle. There is only a fixed set of admissable `ML_SYSTEM` names (see the `etc/settings` file of the distribution).

The actual compiler binary will be run from the directory `ML_HOME`, with `ML_OPTIONS` as first arguments on the command line. The optional `ML_PLATFORM` may specify the binary format of ML heap images, which is useful for cross-platform installations. The value of `ML_IDENTIFIER` is automatically obtained by composing the values of `ML_SYSTEM`, `ML_PLATFORM` and the Isabelle version values.

`ISABELLE_PATH` is a list of directories (separated by colons) where Isabelle logic images may reside. When looking up heaps files, the value of `ML_IDENTIFIER` is appended to each component internally.

`ISABELLE_OUTPUT*` is a directory where output heap files should be stored by default. The ML system and Isabelle version identifier is appended here, too.

`ISABELLE_BROWSER_INFO` is the directory where theory browser information (HTML text, graph data, and printable documents) is stored (see also §2.1). The default value is `$ISABELLE_HOME_USER/browser_info`.

`ISABELLE_LOGIC` specifies the default logic to load if none is given explicitely by the user. The default value is `HOL`.

`ISABELLE_USEDIR_OPTIONS` is implicitly prefixed to the command line of any `isatool usedir` invocation (see also §2.4). This typically contains compilation options for object-logics — `usedir` is the basic utility for managing logic sessions (cf. the `IsaMakefiles` in the distribution).

`ISABELLE_FILE_IDENT` specifies a shell command for producing a source file identification, based on the actual content instead of the full physical path and date stamp (which is the default). A typical identification would produce a "digest" of the text, using a cryptographic hash function like SHA-1, for example.

`ISABELLE_LATEX`, `ISABELLE_PDFLATEX`, `ISABELLE_BIBTEX`, `ISABELLE_DVIPS` refer to LaTeX related tools for Isabelle document preparation (see also §2.6).

`ISABELLE_TOOLS` is a colon separated list of directories that are scanned by `isatool` for external utility programs (see also §1.2).

`ISABELLE_DOCS` is a colon separated list of directories with documentation files.

`ISABELLE_DOC_FORMAT` specifies the preferred document format, typically `dvi` or `pdf`.

`DVI_VIEWER` specifies the command to be used for displaying `dvi` files.

`PDF_VIEWER` specifies the command to be used for displaying `pdf` files.

`PRINT_COMMAND` specifies the standard printer spool command, which is expected to accept `ps` files.

`ISABELLE_TMP_PREFIX*` is the prefix from which any running `isabelle` process derives an individual directory for temporary files. The default is somewhere in `/tmp`.

`ISABELLE_INTERFACE` is an identifier that specifies the actual user interface that the capital `Isabelle` or `isabelle-interface` should invoke. See §1.4 for more details.

1.2 The Isabelle tools wrapper

All Isabelle related utilities are called via a common wrapper — `isatool`:

```
Usage: isatool TOOL [ARGS ...]

  Start Isabelle utility program TOOL with ARGS. Pass "-?" to TOOL
  for more specific help.

  Available tools are:

    browser - Isabelle graph browser
    ...
```

In principle, Isabelle tools are ordinary executable scripts that are run within the Isabelle settings environment, see §1.1. The set of available tools is collected by `isatool` from the directories listed in the `ISABELLE_TOOLS` setting. Do not try to call the scripts directly from the shell. Neither should you add the tool directories to your shell's search path!

Examples

Show the list of available documentation of the current Isabelle installation like this:

```
isatool doc
```

View a certain document as follows:

```
isatool doc isar-ref
```

Create an Isabelle session derived from HOL (see also §2.3 and §3.9):

```
isatool mkdir HOL Test && isatool make
```

Note that `isatool mkdir` is usually only invoked once; existing sessions (including document output etc.) are then updated by `isatool make` alone.

1.3 The raw Isabelle process

The `isabelle` (or `isabelle-process`) executable runs bare-bones Isabelle logic sessions — either interactively or in batch mode. It provides an abstraction over the underlying ML system, and over the actual heap file locations. Its usage is:

```
Usage: isabelle [OPTIONS] [INPUT] [OUTPUT]

  Options are:
    -C           tell ML system to copy output image
    -I           startup Isar interaction mode
    -P           startup Proof General interaction mode
    -S           secure mode -- disallow critical operations
    -X           startup PGIP interaction mode
    -c           tell ML system to compress output image
    -e MLTEXT    pass MLTEXT to the ML session
    -f           pass 'Session.finish();' to the ML session
    -m MODE      add print mode for output
    -q           non-interactive session
    -r           open heap file read-only
    -u           pass 'use"ROOT.ML";' to the ML session
    -w           reset write permissions on OUTPUT

  INPUT (default "$ISABELLE_LOGIC") and OUTPUT specify in/out heaps.
  These are either names to be searched in the Isabelle path, or
  actual file names (containing at least one /).
  If INPUT is "RAW_ML_SYSTEM", just start the bare bones ML system.
```

Input files without path specifications are looked up in the `ISABELLE_PATH` setting, which may consist of multiple components separated by colons —

these are tried in the given order with the value of `ML_IDENTIFIER` appended internally. In a similar way, base names are relative to the directory specified by `ISABELLE_OUTPUT`. In any case, actual file locations may also be given by including at least one slash (`/`) in the name (hint: use `./` to refer to the current directory).

Options

If the input heap file does not have write permission bits set, or the `-r` option is given explicitely, then the session started will be read-only. That is, the ML world cannot be committed back into the image file. Otherwise, a writable session enables commits into either the input file, or into another output heap file (if that is given as the second argument on the command line).

The read-write state of sessions is determined at startup only, it cannot be changed intermediately. Also note that heap images may require considerable amounts of disk space (approximately 40–80 MB). Users are responsible for themselves to dispose their heap files when they are no longer needed.

The `-w` option makes the output heap file read-only after terminating. Thus subsequent invocations cause the logic image to be read-only automatically.

The `-c` option tells the underlying ML system to compress the output heap (fully transparently). On Poly/ML for example, the image is garbage collected and all stored values are maximally shared, resulting in up to 50% less disk space consumption.

The `-C` option tells the ML system to produce a completely self-contained output image, probably including a copy of the ML runtime system itself.

Using the `-e` option, arbitrary ML code may be passed to the Isabelle session from the command line. Multiple `-e`'s are evaluated in the given order. Strange things may happen when errorneous ML code is provided. Also make sure that the ML commands are terminated properly by semicolon.

The `-u` option is a shortcut for `-e` passing "`use"ROOT.ML";`" to the ML session. The `-f` option passes "`Session.finish();`", which is intended mainly for administrative purposes.

The `-m` option adds identifiers of print modes to be made active for this session. Typically, this is used by some user interface, e.g. to enable output of proper mathematical symbols.

Isabelle normally enters an interactive top-level loop (after processing the `-e` texts). The `-q` option inhibits interaction, thus providing a pure batch mode facility.

The `-I` option makes Isabelle enter Isar interaction mode on startup, instead of the primitive ML top-level. The `-P` option configures the top-level loop for interaction with the Proof General user interface, and the `-X` option enables XML-based PGIP communication.

The `-S` option makes the Isabelle process more secure by disabling some critical operations, notably runtime compilation and evaluation of ML source code.

Examples

Run an interactive session of the default object-logic (as specified by the `ISABELLE_LOGIC` setting) like this:

```
isabelle
```

Usually `ISABELLE_LOGIC` refers to one of the standard logic images, which are read-only by default. A writable session — based on `FOL`, but output to `Foo` (in the directory specified by the `ISABELLE_OUTPUT` setting) — may be invoked as follows:

```
isabelle FOL Foo
```

Ending this session normally (e.g. by typing control-D) dumps the whole ML system state into `Foo`. Be prepared for several tens of megabytes.

The `Foo` session may be continued later (still in writable state) by:

```
isabelle Foo
```

A read-only `Foo` session may be started by:

```
isabelle -r Foo
```

Note that manual session management like this does *not* provide proper setup for theory presentation. This would require the `usedir` utility, see §2.4.

The next example demonstrates batch execution of Isabelle. We print a certain theorem of `FOL`:

```
isabelle -e "prth allE;" -q -r FOL
```

Note that the output text will be interspersed with additional junk messages by the ML runtime environment.

1.4 The Isabelle interface wrapper

Isabelle is a generic theorem prover, even w.r.t. its user interface. The `Isabelle` (or `isabelle-interface`) executable provides a uniform way for end-users to invoke a certain interface; which one to start is determined by the `ISABELLE_INTERFACE` setting variable, which should give a full path specification to the actual executable. Also note that the `install` utility provides some options to install desktop environment icons (see §3.7).

Presently, the most prominent Isabelle interface is Proof General [1]. There are separate versions for the raw ML top-level and the proper Isabelle/Isar interpreter loop. The Proof General distribution includes separate interface wrapper scripts (in `ProofGeneral/isa` and `ProofGeneral/isar`) for either of these. The canonical settings for Isabelle/Isar are as follows:

```
ISABELLE_INTERFACE=$ISABELLE_HOME/contrib/ProofGeneral/isar/interface
PROOFGENERAL_OPTIONS=""
```

Thus `Isabelle` would automatically invoke Emacs with proper setup of the Proof General Lisp packages. There are some options available, such as `-l` for passing the logic image to be used by default, or `-m` to tune the standard print mode. The `-I` option allows to switch between the Isar and ML view, independently of the interface script being used.

Note that the world may be also seen the other way round: Emacs may be started first (with proper setup of Proof General mode), and `isabelle` run from within. This requires further Emacs Lisp configuration, see the Proof General documentation [1] for more information.

Chapter 2

Presenting theories

Isabelle provides several ways to present the outcome of formal developments, including WWW-based browsable libraries or actual printable documents. Presentation is centered around the concept of *logic sessions*. The global session structure is that of a tree, with Isabelle Pure at its root, further object-logics derived (e.g. HOLCF from HOL, and HOL from Pure), and application sessions in leaf positions (usually without a separate image).

The `mkdir` (see §2.3) and `make` (see §3.9) tools of Isabelle provide the primary means for managing Isabelle sessions, including proper setup for presentation. Here the `usedir` (see §2.4) tool takes care to let the `isabelle` process run any additional stages required for document preparation, notably the tools `document` (see §2.5) and `latex` (see §2.6). The complete tool chain for managing batch-mode Isabelle sessions is illustrated in figure 2.1.

`isatool mkdir`	invoked once by the user to create the initial source setup (common `IsaMakefile` plus a single session directory);
`isatool make`	invoked repeatedly by the user to keep session output up-to-date (HTML, documents etc.);
`isatool usedir`	part of the standard `IsaMakefile` entry of a session;
`isabelle`	run through `isatool usedir`;
`isatool document`	run by the Isabelle process if document preparation is enabled;
`isatool latex`	universal LaTeX tool wrapper invoked multiple times by `isatool document`; also useful for manual experiments;

Figure 2.1: The tool chain of Isabelle session presentation

2.1 Generating theory browser information

As a side-effect of running a logic sessions, Isabelle is able to generate theory browsing information, including HTML documents that show a theory's definition, the theorems proved in its ML file and the relationship with its ancestors and descendants. Besides the HTML file that is generated for every theory, Isabelle stores links to all theories in an index file. These indexes are linked with other indexes to represent the overall tree structure of logic sessions.

Isabelle also generates graph files that represent the theory hierarchy of a logic. There is a graph browser Java applet embedded in the generated HTML pages, and also a stand-alone application that allows browsing theory graphs without having to start a WWW client first. The latter version also includes features such as generating Postscript files, which are not available in the applet version. See §2.2 for further information.

The easiest way to let Isabelle generate theory browsing information for existing sessions is to append "`-i true`" to the `ISABELLE_USEDIR_OPTIONS` before invoking `isatool make` (or `./build` in the distribution). For example, add something like this to your Isabelle settings file

```
ISABELLE_USEDIR_OPTIONS="-i true"
```

and then change into the `src/FOL` directory of the Isabelle distribution and run `isatool make`, or even `isatool make all`. The presentation output will appear in `$ISABELLE_BROWSER_INFO/FOL`, which usually refers to `~/isabelle/browser_info/FOL`. Note that option `-v true` will make the internal runs of `usedir` more explicit about such details.

Many standard Isabelle sessions (such as `HOL/ex`) also provide actual printable documents. These are prepared automatically as well if enabled like this, using the `-d` option

```
ISABELLE_USEDIR_OPTIONS="-i true -d dvi"
```

Enabling options `-i` and `-d` simultaneausly as shown above causes an appropriate "document" link to be included in the HTML index. Documents (or raw document sources) may be generated independently of browser information as well, see §2.5 for further details.

The theory browsing information is stored in a sub-directory directory determined by the `ISABELLE_BROWSER_INFO` setting plus a prefix corresponding to the session identifier (according to the tree structure of sub-sessions by default). A complete WWW view of all standard object-logics and examples of the Isabelle distribution is available at the Cambridge or Munich Isabelle sites:

http://www.cl.cam.ac.uk/Research/HVG/Isabelle/library/
http://isabelle.in.tum.de/library/

In order to present your own theories on the web, simply copy the corresponding subdirectory from `ISABELLE_BROWSER_INFO` to your WWW server, having generated browser info like this:

```
isatool usedir -i true HOL Foo
```

This assumes that directory `Foo` contains some `ROOT.ML` file to load all your theories, and HOL is your parent logic image (`isatool mkdir` assists in setting up Isabelle session directories, see §2.3). Theory browser information for HOL should have been generated already beforehand. Alternatively, one may specify an external link to an existing body of HTML data by giving `usedir` a `-P` option like this:

```
isatool usedir -i true -P http://isabelle.in.tum.de/library/ HOL Foo
```

For production use, the `usedir` tool is usually invoked in an appropriate `IsaMakefile`, via the Isabelle `make` utility. There is a separate `mkdir` tool to provide easy setup of all this, with only minimal manual editing required.

```
isatool mkdir HOL Foo && isatool make
```

See §2.3 for more information on preparing Isabelle session directories, including the setup for documents.

2.2 Browsing theory graphs

The Isabelle graph browser is a general tool for visualizing dependency graphs. Certain nodes of the graph (i.e. theories) can be grouped together in "directories", whose contents may be hidden, thus enabling the user to collapse irrelevant portions of information. The browser is written in Java, it can be used both as a stand-alone application and as an applet. Note that the option `-g` of `isatool usedir` (see §2.4) creates graph presentations in batch mode for inclusion in session documents.

Invoking the graph browser

The stand-alone version of the graph browser is wrapped up as an Isabelle tool called `browser`:

```
Usage: browser [OPTIONS] [GRAPHFILE]

  Options are:
    -c           cleanup -- remove GRAPHFILE after use
    -o FILE      output to FILE (ps, eps, pdf)
```

When no filename is specified, the browser automatically changes to the directory `ISABELLE_BROWSER_INFO`.

The `-c` option causes the input file to be removed after use.

The `-o` option indicates batch-mode operation, with the output written to the indicated file; note that `pdf` produces an `eps` copy as well.

The applet version of the browser is part of the standard WWW theory presentation, see the link "theory dependencies" within each session index.

Using the graph browser

The browser's main window, which is shown in figure 2.2, consists of two sub-windows: In the left sub-window, the directory tree is displayed. The graph itself is displayed in the right sub-window.

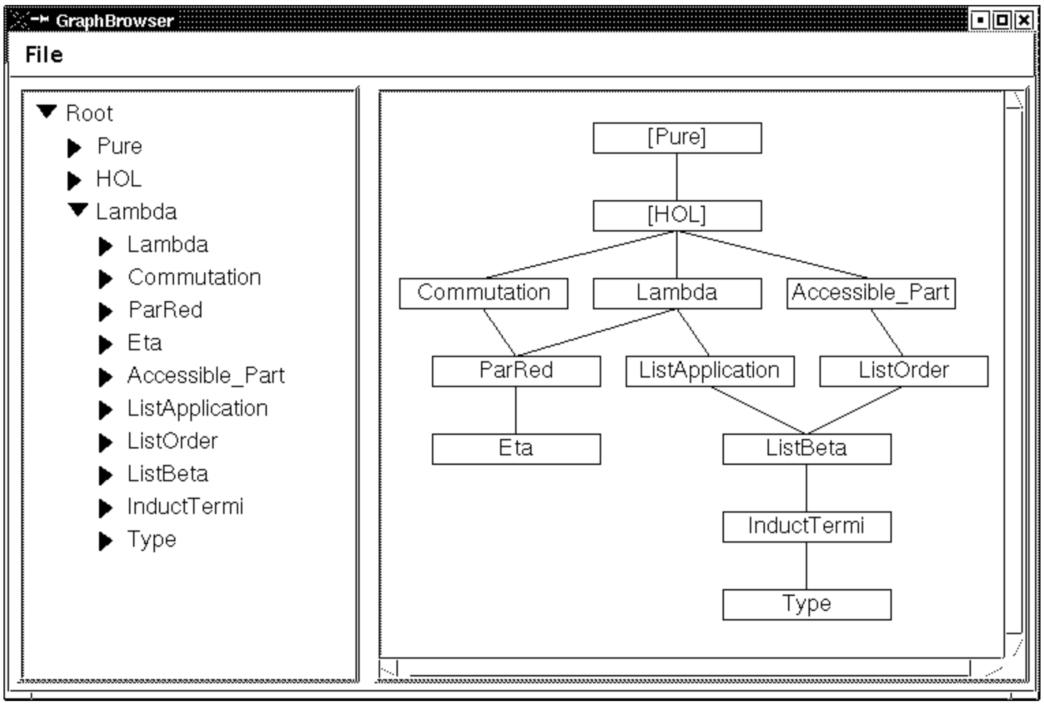

Figure 2.2: Browser main window

The directory tree window

We describe the usage of the directory browser and the meaning of the different items in the browser window.

- A red arrow before a directory name indicates that the directory is currently "folded", i.e. the nodes in this directory are collapsed to one single node. In the right sub-window, the names of nodes corresponding to folded directories are enclosed in square brackets and displayed in red color.
- A green downward arrow before a directory name indicates that the directory is currently "unfolded". It can be folded by clicking on the directory name. Clicking on the name for a second time unfolds the directory again. Alternatively, a directory can also be unfolded by clicking on the corresponding node in the right sub-window.
- Blue arrows stand before ordinary node names. When clicking on such a name (i.e. that of a theory), the graph display window focuses to the corresponding node. Double clicking invokes a text viewer window in which the contents of the theory file are displayed.

The graph display window

When pointing on an ordinary node, an upward and a downward arrow is shown. Initially, both of these arrows are green. Clicking on the upward or downward arrow collapses all predecessor or successor nodes, respectively. The arrow's color then changes to red, indicating that the predecessor or successor nodes are currently collapsed. The node corresponding to the collapsed nodes has the name "`[....]`". To uncollapse the nodes again, simply click on the red arrow or on the node with the name "`[....]`". Similar to the directory browser, the contents of theory files can be displayed by double clicking on the corresponding node.

The "File" menu

Please note that due to Java security restrictions this menu is not available in the applet version. The meaning of the menu items is as follows:

Open ... Open a new graph file.

Export to PostScript Outputs the current graph in Postscript format, appropriately scaled to fit on one single sheet of A4 paper. The resulting file can be printed directly.

Export to EPS Outputs the current graph in Encapsulated Postscript format. The resulting file can be included in other documents.

Quit Quit the graph browser.

*Syntax of graph definition files

A graph definition file has the following syntax:

$$\begin{array}{rcl} graph & = & \{\ vertex\ \texttt{;}\ \}^{+} \\ vertex & = & vertexname\ vertexID\ dirname\ [\texttt{+}]\ path\ [\texttt{<}|\texttt{>}]\ \{\ vertexID\ \}^{*} \end{array}$$

The meaning of the items in a vertex description is as follows:

vertexname The name of the vertex.

vertexID The vertex identifier. Note that there may be two vertices with equal names, whereas identifiers must be unique.

dirname The name of the "directory" the vertex should be placed in. A "`+`" sign after *dirname* indicates that the nodes in the directory are initially visible. Directories are initially invisible by default.

path The path of the corresponding theory file. This is specified relatively to the path of the graph definition file.

List of successor/predecessor nodes A "`<`" sign before the list means that successor nodes are listed, a "`>`" sign means that predecessor nodes are listed. If neither "`<`" nor "`>`" is found, the browser assumes that successor nodes are listed.

2.3 Creating Isabelle session directories — `isatool mkdir`

The `mkdir` utility prepares Isabelle session source directories, including a sensible default setup of `IsaMakefile`, `ROOT.ML`, and a `document` directory with a minimal `root.tex` that is sufficient to print all theories of the session (in the order of appearance); see §2.5 for further information on Isabelle document preparation. The usage of `isatool mkdir` is:

```
Usage: mkdir [OPTIONS] [LOGIC] NAME

  Options are:
    -I FILE      alternative IsaMakefile output
    -P           include parent logic target
    -b           setup build mode (session outputs heap image)
    -q           quiet mode

  Prepare session directory, including IsaMakefile and document source,
  with parent LOGIC (default ISABELLE_LOGIC=$ISABELLE_LOGIC)
```

The `mkdir` tool is conservative in the sense that any existing `IsaMakefile` etc. is left unchanged. Thus it is safe to invoke it multiple times, although later runs may not have the desired effect.

Note that `mkdir` is unable to change `IsaMakefile` incrementally — manual changes are required for multiple sub-sessions. On order to get an initial working session, the only editing needed is to add appropriate `use_thy` calls to the generated `ROOT.ML` file.

Options

The `-I` option specifies an alternative to `IsaMakefile` for dependencies. Note that "`-`" refers to *stdout*, i.e. "`-I-`" provides an easy way to peek at `mkdir`'s idea of `make` setup required for some particular of Isabelle session.

The `-P` option includes a target for the parent `LOGIC` session in the generated `IsaMakefile`. The corresponding sources are assumed to be located within the Isabelle distribution.

The `-b` option sets up the current directory as the base for a new session that provides an actual logic image, as opposed to one that only runs several theories based on an existing image. Note that in the latter case, everything except `IsaMakefile` would be placed into a separate directory `NAME`, rather than the current one. See §2.4 for further information on *build mode* vs. *example mode* of the `usedir` utility.

The `-q` enables quiet mode, suppressing further notes on how to proceed.

Examples

The standard setup of a single "example session" based on the default logic, with proper document generation is generated like this:

```
isatool mkdir Foo && isatool make
```

The theory sources should be put into the `Foo` directory, and its `ROOT.ML` should be edited to load all required theories. Invoking `isatool make` again would run the whole session, generating browser information and the document automatically. The `IsaMakefile` is typically tuned manually later, e.g. adding source dependencies, or changing the options passed to `usedir`.

Large projects may demand further sessions, potentially with separate logic images being created. This usually requires manual editing of the generated `IsaMakefile`, which is meant to cover all of the sub-session directories at the same time (this is the deeper reasong why `IsaMakefile` is not made part of the initial session directory created by `isatool mkdir`). See `src/HOL/IsaMakefile` of the Isabelle distribution for a full-blown example.

2.4 Running Isabelle sessions — `isatool usedir`

The `usedir` utility builds object-logic images, or runs example sessions based on existing logics. Its usage is:

```
Usage: usedir [OPTIONS] LOGIC NAME

  Options are:
    -C BOOL      copy existing document directory to -D PATH (default true)
    -D PATH      dump generated document sources into PATH
    -M MAX       multithreading: maximum number of worker threads (default 1)
    -P PATH      set path for remote theory browsing information
    -T LEVEL     multithreading: trace level (default 0)
    -V VERSION   declare alternative document VERSION
    -b           build mode (output heap image, using current dir)
    -c BOOL      tell ML system to compress output image (default true)
    -d FORMAT    build document as FORMAT (default false)
    -f NAME      use ML file NAME (default ROOT.ML)
    -g BOOL      generate session graph image for document (default false)
    -i BOOL      generate theory browser information (default false)
    -m MODE      add print mode for output
    -p LEVEL     set level of detail for proof objects
    -r           reset session path
    -s NAME      override session NAME
    -v BOOL      be verbose (default false)

  Build object-logic or run examples. Also creates browsing
  information (HTML etc.) according to settings.

  ISABELLE_USEDIR_OPTIONS=
  HOL_USEDIR_OPTIONS=
```

Note that the value of the `ISABELLE_USEDIR_OPTIONS` setting is implicitly prefixed to *any* `usedir` call. Since the `IsaMakefile`s of all object-logics distributed with Isabelle just invoke `usedir` for the real work, one may control compilation options globally via above variable. In particular, generation of HTML browsing information and document preparation is controlled here.

The `HOL_USEDIR_OPTIONS` setting is specific to the main Isabelle/HOL image; its value is appended to `ISABELLE_USEDIR_OPTIONS` for that particular session only.

Options

Basically, there are two different modes of operation: *build mode* (enabled through the `-b` option) and *example mode* (default).

Calling `usedir` with `-b` runs `isabelle` with input image `LOGIC` and output to `NAME`, as provided on the command line. This will be a batch session, running `ROOT.ML` from the current directory and then quitting. It is assumed that `ROOT.ML` contains all ML commands required to build the logic.

In example mode, `usedir` runs a read-only session of `LOGIC` and automatically runs `ROOT.ML` from within directory `NAME`. It assumes that this file contains appropriate ML commands to run the desired examples.

The `-i` option controls theory browser data generation. It may be explicitly turned on or off — as usual, the last occurrence of `-i` on the command line wins.

The `-P` option specifies a path (or actual URL) to be prefixed to any *non-local* reference of existing theories. Thus user sessions may easily link to existing Isabelle libraries already present on the WWW.

The `-m` options specifies additional print modes to be activated temporarily while the session is processed.

The `-d` option controls document preparation. Valid arguments are `false` (do not prepare any document; this is default), or any of `dvi`, `dvi.gz`, `ps`, `ps.gz`, `pdf`. The logic session has to provide a properly setup `document` directory. See §2.5 and §2.6 for more details.

The `-V` option declares alternative document versions, consisting of name/tags pairs (cf. options `-n` and `-t` of the `document` tool, §2.5). The standard document is equivalent to "`document=theory,proof,ML`", which means that all theory begin/end commands, proof body texts, and ML code will be presented faithfully. An alternative version "`outline=/proof/ML`" would fold proof and ML parts, replacing the original text by a short placeholder. The form "*name*`=-`" means to remove document *name* from the list of versions to be processed. Any number of `-V` options may be given; later declarations have precedence over earlier ones.

The `-g` option produces images of the theory dependency graph (cf. §2.2) for inclusion in the generated document, both as `session_graph.eps` and `session_graph.pdf` at the same time. To include this in the final LaTeX document one could say `\includegraphics{session_graph}` in `document/root.tex` (omitting the file-name extension enables LaTeX to select to correct version, either for the DVI or PDF output path).

The `-D` option causes the generated document sources to be dumped at location `PATH`; this path is relative to the session's main directory. If the `-C` option is true, this will include a copy of an existing `document` directory as provided by the user. For example, `isatool usedir -D generated HOL Foo` produces a complete set of document sources at `Foo/generated`. Subsequent invocation of `isatool document Foo/generated` (see also §2.5) will process the final result independently of an Isabelle job. This decoupled mode of operation facilitates debugging of serious LaTeX errors, for example.

The `-p` option determines the level of detail for internal proof objects, see also the *Isabelle Reference Manual* [2].

The `-v` option causes additional information to be printed while running the session, notably the location of prepared documents.

The `-M` option specifies the maximum number of parallel threads used for processing independent theory files (multithreading only works on suitable ML platforms). When tuning the performance of large Isabelle sessions, the number of actual CPU cores of the underlying hardware is a good starting point for option `-M`. The `-T` option determines the level of detail in tracing output concerning the internal locking and scheduling in multithreaded operation. This may be helpful in isolating performance bottle-necks, e.g. due to excessive wait states when locking critical code sections.

Any `usedir` session is named by some *session identifier*. These accumulate, documenting the way sessions depend on others. For example, consider `Pure/FOL/ex`, which refers to the examples of FOL, which in turn is built upon Pure.

The current session's identifier is by default just the base name of the `LOGIC` argument (in build mode), or of the `NAME` argument (in example mode). This may be overridden explicitly via the `-s` option.

Examples

Refer to the `IsaMakefile`s of the Isabelle distribution's object-logics as a model for your own developments. For example, see `src/FOL/IsaMakefile`. The Isabelle `mkdir` tool (see §2.3) creates `IsaMakefile`s with proper invocation of `usedir` as well.

2.5 Preparing Isabelle session documents — `isatool document`

The `document` utility prepares logic session documents, processing the sources both as provided by the user and generated by Isabelle. Its usage is:

```
Usage: document [OPTIONS] [DIR]

  Options are:
    -c           cleanup -- be aggressive in removing old stuff
    -n NAME      specify document name (default 'document')
    -o FORMAT    specify output format: dvi (default), dvi.gz, ps,
                 ps.gz, pdf
    -t TAGS      specify tagged region markup

  Prepare the theory session document in DIR (default 'document')
  producing the specified output format.
```

This tool is usually run automatically as part of the corresponding Isabelle batch process, provided document preparation has been enabled (cf. the `-d` option of the `usedir` utility, §2.4). It may be manually invoked on the generated browser information document output as well, e.g. in case of errors encountered in the batch run.

The `-c` option tells the `document` tool to dispose the document sources after successful operation. This is the right thing to do for sources generated by an Isabelle process, but take care of your files in manual document preparation!

The `-n` and `-o` option specify the final output file name and format, the default is "`document.dvi`". Note that the result will appear in the parent of the target `DIR`.

The `-t` option tells LaTeX how to interpret tagged Isabelle command regions. Tags are specified as a comma separated list of modifier/name pairs: "`+`*foo*" (or just "*foo*") means to keep, "`-`*foo*" to drop, and "`/`*foo*" to fold text tagged as *foo*. The builtin default is equivalent to the tag specification "`/theory,/proof,/ML,+visible,-invisible`"; see also the LaTeX macros `\isakeeptag`, `\isadroptag`, and `\isafoldtag` in `isabelle.sty`.

Document preparation requires a properly setup "`document`" directory within the logic session sources. This directory is supposed to contain all the files needed to produce the final document — apart from the actual theories which are generated by Isabelle.

For most practical purposes, the `document` tool is smart enough to create any of the specified output formats, taking `root.tex` supplied by the user as a starting point. This even includes multiple runs of LaTeX to accommodate references and bibliographies (the latter assumes `root.bib` within the same directory).

In more complex situations, a separate `IsaMakefile` for the document sources may be given instead. This should provide targets for any admissible document format; these have to produce corresponding output files named after `root` as well, e.g. `root.dvi` for target format `dvi`.

When running the session, Isabelle copies the original `document` directory into its proper place within `ISABELLE_BROWSER_INFO` according to the session path. Then, for any processed theory A some LaTeX source is generated and put there as A`.tex`. Furthermore, a list of all generated theory files is put into `session.tex`. Typically, the root LaTeX file provided by the user would include `session.tex` to get a document containing all the theories.

The LaTeX versions of the theories require some macros defined in `isabelle.sty` as distributed with Isabelle. Doing `\usepackage{isabelle}` in `root.tex` should be fine; the underlying Isabelle `latex` utility already includes an appropriate TeX inputs path.

If the text contains any references to Isabelle symbols (such as `\<forall>`) then `isabellesym.sty` should be included as well. This package contains a standard set of LaTeX macro definitions `\isasym`*foo* corresponding to `\<`*foo*`>` (see Appendix A for a complete list of predefined Isabelle symbols). Users may invent further symbols as well, just by providing LaTeX macros in a similar fashion as in `isabellesym.sty` of the distribution.

For proper setup of PDF documents (with hyperlinks, bookmarks, and thumbnail images), we recommend to include `pdfsetup.sty` as well. It is safe to do so even without using PDF LaTeX.

As a final step of document preparation within Isabelle, `isatool document -c` is run on the resulting `document` directory. Thus the actual output document is built and installed in its proper place (as linked by the session's `index.html` if option `-i` of `usedir` has been enabled, cf. §2.1). The generated sources are deleted after successful run of LaTeX and friends. Note that a separate copy of the sources may be retained by passing an option `-D` to the `usedir` utility when running the session (see also §2.4).

2.6 Running LaTeX within the Isabelle environment — `isatool latex`

The `latex` utility provides the basic interface for Isabelle document preparation. Its usage is:

```
Usage: latex [OPTIONS] [FILE]

  Options are:
    -o FORMAT    specify output format: dvi (default), dvi.gz, ps,
                 ps.gz, pdf, bbl, png, sty, syms

  Run LaTeX (and related tools) on FILE (default root.tex),
  producing the specified output format.
```

Appropriate LaTeX-related programs are run on the input file, according to the given output format: `latex`, `pdflatex`, `dvips`, `bibtex` (for `bbl`), and `thumbpdf` (for `png`). The actual commands are determined from the settings environment (`ISABELLE_LATEX` etc., see §1.1).

The `sty` output format causes the Isabelle style files to be updated from the distribution. This is useful in special situations where the document sources are to be processed another time by separate tools (cf. option `-D` of the `usedir` utility, see §2.4).

The `syms` output is for internal use; it generates lists of symbols that are available without loading additional LaTeX packages.

Examples

Invoking `isatool latex` by hand may be occasionally useful when debugging failed attempts of the automatic document preparation stage of batch-mode Isabelle. The abortive process leaves the sources at a certain place within `ISABELLE_BROWSER_INFO`, see the runtime error message for details. This enables users to inspect LaTeX runs in further detail, e.g. like this:

```
  cd ~/isabelle/browser_info/HOL/Test/document
  isatool latex -o pdf
```

Chapter 3

Miscellaneous tools

Subsequently we describe various Isabelle related utilities, given in alphabetical order.

3.1 Converting legacy ML scripts — `isatool convert`

The `convert` utility assists in converting legacy ML proof scripts into the new-style format of Isabelle/Isar:

```
Usage: convert [FILES|DIRS...]

  Recursively find .ML files, converting legacy tactic scripts to
  Isabelle/Isar tactic emulation.
  Note: conversion is only approximated, based on some heuristics.

  Renames old versions of FILES by appending "~0~".
  Creates new versions of FILES by appending ".thy".
```

The resulting theory text uses the tactic emulation facilities of Isabelle/Isar (see also [2], especially the "Conversion guide" in the appendix). Usually there is some manual tuning required to get an automatically converted script work again — the success rate is around 99% for common ML scripts.

3.2 Displaying documents — `isatool display`

The `display` utility displays documents in DVI format:

```
Usage: display [OPTIONS] FILE

  Options are:
    -c           cleanup -- remove FILE after use

  Display document FILE (in DVI format).
```

The `-c` option causes the input file to be removed after use. The program for viewing `dvi` files is determined by the `DVI_VIEWER` setting.

3.3 Viewing documentation — `isatool doc`

The `doc` utility displays online documentation:

```
Usage: doc [DOC]

  View Isabelle documentation DOC, or show list of available documents.
```

If called without arguments, it lists all available documents. Each line starts with an identifier, followed by a short description. Any of these identifiers may be specified as the first argument in order to have the corresponding document displayed.

The `ISABELLE_DOCS` setting specifies the list of directories (separated by colons) to be scanned for documentations. The program for viewing `dvi` files is determined by the `DVI_VIEWER` setting.

3.4 Tuning proof scripts — `isatool expandshort`

The `expandshort` utility tunes ML proof scripts to enhance readability:

```
Usage: expandshort [FILES|DIRS...]

  Recursively find .ML files, expand shorthand goal commands.  Also
  contracts uses of resolve_tac, dresolve_tac, eresolve_tac,
  forward_tac, rewrite_goals_tac on 1-element lists; furthermore
  expands tabs, which are forbidden in SML string constants.

  Renames old versions of files by appending "~~".
```

In the files or directories supplied as arguments, all occurrences of the shorthand commands `br`, `be` etc. in proof scripts are replaced with the corresponding full commands. The old versions of the files are renamed to have the suffix "`~~`".

3.5 Getting logic images — `isatool findlogics`

The `findlogics` utility traverses all directories specified in `ISABELLE_PATH`, looking for Isabelle logic images. Its usage is:

```
Usage: findlogics

  Collect heap file names from ISABELLE_PATH.
```

The base names of all files found on the path are printed — sorted and with duplicates removed. Also note that lookup in `ISABELLE_PATH` includes the current values of `ML_SYSTEM` and `ML_PLATFORM`. Thus switching to another ML compiler may change the set of logic images available.

3.6 Inspecting the settings environment — `isatool getenv`

The Isabelle settings environment — as provided by the site-default and user-specific settings files — can be inspected with the `getenv` utility:

```
Usage: getenv [OPTIONS] [VARNAMES ...]

  Options are:
    -a          display complete environment
    -b          print values only (doesn't work for -a)

  Get value of VARNAMES from the Isabelle settings.
```

With the `-a` option, one may inspect the full process environment that Isabelle related programs are run in. This usually contains much more variables than are actually Isabelle settings. Normally, output is a list of lines of the form *name*=*value*. The `-b` option causes only the values to be printed.

Examples

Get the ML system name and the location where the compiler binaries are supposed to reside as follows:

```
isatool getenv ML_SYSTEM ML_HOME
  ML_SYSTEM=polyml
  ML_HOME=/usr/share/polyml/x86-linux
```

The next one peeks at the output directory for `isabelle` logic images:

```
isatool getenv -b ISABELLE_OUTPUT
  /home/me/isabelle/heaps/polyml_x86-linux
```

Here we have used the `-b` option to suppress the `ISABELLE_OUTPUT=` prefix. The value above is what became of the following assignment in the default settings file:

```
ISABELLE_OUTPUT="$ISABELLE_HOME_USER/heaps"
```

Note how the `ML_IDENTIFIER` value got appended automatically to each path component. This is a special feature of `ISABELLE_OUTPUT`.

3.7 Installing standalone Isabelle executables — `isatool install`

By default, the Isabelle binaries (`isabelle`, `isatool` etc.) are just run from their location within the distribution directory, probably indirectly by the shell through its `PATH`. Other schemes of installation are supported by the `install` utility:

```
Usage: install [OPTIONS]

  Options are:
    -d DISTDIR   use DISTDIR as Isabelle distribution
                 (default ISABELLE_HOME)
    -p DIR       install standalone binaries in DIR

  Install Isabelle executables with absolute references to the current
  distribution directory.
```

The `-d` option overrides the current Isabelle distribution directory as determined by `ISABELLE_HOME`.

The `-p` option installs executable wrapper scripts for `isabelle`, `isatool`, `Isabelle`, containing proper absolute references to the Isabelle distribution directory. A typical `DIR` specification would be some directory expected to be in the shell's `PATH`, such as `/usr/local/bin`. It is important to note that a plain manual copy of the original Isabelle executables just would not work!

3.8 Creating instances of the Isabelle logo — `isatool logo`

The `logo` utility creates any instance of the generic Isabelle logo as an Encapsuled Postscript file (EPS):

```
Usage: logo [OPTIONS] NAME

  Create instance NAME of the Isabelle logo (as EPS).

  Options are:
    -o OUTFILE   set output file (default determined from NAME)
    -q           quiet mode
```

You are encouraged to use this to create a derived logo for your Isabelle project. For example, `isatool logo Bali` creates `isabelle_bali.eps`.

3.9 Isabelle's version of make — `isatool make`

The Isabelle `make` utility is a very simple wrapper for ordinary Unix `make`:

```
Usage: make [ARGS ...]

  Compile the logic in current directory using IsaMakefile.
  ARGS are directly passed to the system make program.
```

Note that the Isabelle settings environment is also active. Thus one may refer to its values within the `IsaMakefile`, e.g. `$(ISABELLE_OUTPUT)`. Furthermore, programs started from the make file also inherit this environment. Typically, `IsaMakefile`s defer the real work to the `usedir` utility, see §2.4.

The basic `IsaMakefile` convention is that the default target builds the actual logic, including its parents if appropriate. The `images` target is intended to build all local logic images, while the `test` target shall build all related examples. The `all` target shall do `images` and `test`.

Examples

Refer to the `IsaMakefile`s of the Isabelle distribution's object-logics as a model for your own developments. For example, see `src/FOL/IsaMakefile`.

3.10 Make all logics — `isatool makeall`

The `makeall` utility applies Isabelle make to all logic directories of the distribution:

```
Usage: makeall [ARGS ...]

  Apply isatool make to all logics (passing ARGS).
```

The arguments `ARGS` are just passed verbatim to each `make` invocation.

3.11 Printing documents — `isatool print`

The `print` utility prints documents:

```
Usage: print [OPTIONS] FILE

  Options are:
    -c           cleanup -- remove FILE after use

  Print document FILE.
```

The `-c` option causes the input file to be removed after use. The printer spool command is determined by the `PRINT_COMMAND` setting.

3.12 Remove awkward symbol names from theory sources — `isatool unsymbolize`

The `unsymbolize` utility tunes Isabelle theory sources to improve readability for plain ASCII output (e.g. in email communication). Most notably, `unsymbolize` replaces awkward arrow symbols such as `\<Longrightarrow>` by `==>`.

```
Usage: unsymbolize [FILES|DIRS...]

  Recursively find .thy/.ML files, removing unreadable symbol names.
  Note: this is an ad-hoc script; there is no systematic way to replace
  symbols independently of the inner syntax of a theory!

  Renames old versions of FILES by appending "~~".
```

3.13 Output the version identifier of the Isabelle distribution — `isatool version`

The `version` utility outputs the full version string of the Isabelle distribution being used, e.g. “`Isabelle2007: November 2007`”. There are no options nor arguments.

Appendix A

Standard Isabelle symbols

Isabelle supports an infinite number of non-ASCII symbols, which are represented in source text as `\<`*name*`>` (where *name* may be any identifier). It is left to front-end tools how to present these symbols to the user. The collection of predefined standard symbols given below is available by default for Isabelle document output, due to appropriate definitions of `\isasym`*name* for each `\<`*name*`>` in the `isabellesym.sty` file. Most of these symbols are displayed properly in Proof General if used with the X-Symbol package.

Moreover, any single symbol (or ASCII character) may be prefixed by `\<^sup>` for superscript and `\<^sub>` for subscript, such as `A\<^sup>\<star>` for $A^\star$; the alternative versions `\<^isub>` and `\<^isup>` are considered as quasi letters and may be used within identifiers. Sub- and superscripts that span a region of text are marked up with `\<^bsub>`...`\<^esub>` and `\<^bsup>`...`\<^esup>`, respectively. Furthermore, all ASCII characters and most other symbols may be printed in bold by prefixing `\<^bold>`, such as `\<^bold>\<alpha>` for $\boldsymbol{\alpha}$. Note that `\<^bold>` may *not* be combined with sub- or superscripts for single symbols.

Further details of Isabelle document preparation are covered in chapter 2.

`\<zero>`	$\mathbf{0}$	`\<one>`	$\mathbf{1}$
`\<two>`	$\mathbf{2}$	`\<three>`	$\mathbf{3}$
`\<four>`	$\mathbf{4}$	`\<five>`	$\mathbf{5}$
`\<six>`	$\mathbf{6}$	`\<seven>`	$\mathbf{7}$
`\<eight>`	$\mathbf{8}$	`\<nine>`	$\mathbf{9}$
`\<A>`	$\mathcal{A}$	`\<B>`	$\mathcal{B}$
`\<C>`	$\mathcal{C}$	`\<D>`	$\mathcal{D}$
`\<E>`	$\mathcal{E}$	`\<F>`	$\mathcal{F}$
`\<G>`	$\mathcal{G}$	`\<H>`	$\mathcal{H}$
`\<I>`	$\mathcal{I}$	`\<J>`	$\mathcal{J}$
`\<K>`	$\mathcal{K}$	`\<L>`	$\mathcal{L}$
`\<M>`	$\mathcal{M}$	`\<N>`	$\mathcal{N}$
`\<O>`	$\mathcal{O}$	`\<P>`	$\mathcal{P}$
`\<Q>`	$\mathcal{Q}$	`\<R>`	$\mathcal{R}$

\<S>	$\mathcal{S}$	\<T>	$\mathcal{T}$
\<U>	$\mathcal{U}$	\<V>	$\mathcal{V}$
\<W>	$\mathcal{W}$	\<X>	$\mathcal{X}$
\<Y>	$\mathcal{Y}$	\<Z>	$\mathcal{Z}$
\<a>	a	\<b>	b
\<c>	c	\<d>	d
\<e>	e	\<f>	f
\<g>	g	\<h>	h
\<i>	i	\<j>	j
\<k>	k	\<l>	l
\<m>	m	\<n>	n
\<o>	o	\<p>	p
\<q>	q	\<r>	r
\<s>	s	\<t>	t
\<u>	u	\<v>	v
\<w>	w	\<x>	x
\<y>	y	\<z>	z
\<AA>	$\mathfrak{A}$	\<BB>	$\mathfrak{B}$
\<CC>	$\mathfrak{C}$	\<DD>	$\mathfrak{D}$
\<EE>	$\mathfrak{E}$	\<FF>	$\mathfrak{F}$
\<GG>	$\mathfrak{G}$	\<HH>	$\mathfrak{H}$
\<II>	$\mathfrak{I}$	\<JJ>	$\mathfrak{J}$
\<KK>	$\mathfrak{K}$	\<LL>	$\mathfrak{L}$
\<MM>	$\mathfrak{M}$	\<NN>	$\mathfrak{N}$
\<OO>	$\mathfrak{O}$	\<PP>	$\mathfrak{P}$
\<QQ>	$\mathfrak{Q}$	\<RR>	$\mathfrak{R}$
\<SS>	$\mathfrak{S}$	\<TT>	$\mathfrak{T}$
\<UU>	$\mathfrak{U}$	\<VV>	$\mathfrak{V}$
\<WW>	$\mathfrak{W}$	\<XX>	$\mathfrak{X}$
\<YY>	$\mathfrak{Y}$	\<ZZ>	$\mathfrak{Z}$
\<aa>	$\mathfrak{a}$	\<bb>	$\mathfrak{b}$
\<cc>	$\mathfrak{c}$	\<dd>	$\mathfrak{d}$
\<ee>	$\mathfrak{e}$	\<ff>	$\mathfrak{f}$
\<gg>	$\mathfrak{g}$	\<hh>	$\mathfrak{h}$
\<ii>	$\mathfrak{i}$	\<jj>	$\mathfrak{j}$
\<kk>	$\mathfrak{k}$	\<ll>	$\mathfrak{l}$
\<mm>	$\mathfrak{m}$	\<nn>	$\mathfrak{n}$
\<oo>	$\mathfrak{o}$	\<pp>	$\mathfrak{p}$
\<qq>	$\mathfrak{q}$	\<rr>	$\mathfrak{r}$

\<ss>	$\mathfrak{s}$	\<tt>	$\mathfrak{t}$
\<uu>	$\mathfrak{u}$	\<vv>	$\mathfrak{v}$
\<ww>	$\mathfrak{w}$	\<xx>	$\mathfrak{x}$
\<yy>	$\mathfrak{y}$	\<zz>	$\mathfrak{z}$
\<alpha>	α	\<beta>	β
\<gamma>	γ	\<delta>	δ
\<epsilon>	ε	\<zeta>	ζ
\<eta>	η	\<theta>	ϑ
\<iota>	ι	\<kappa>	κ
\<lambda>	λ	\<mu>	μ
\<nu>	ν	\<xi>	ξ
\<pi>	π	\<rho>	ϱ
\<sigma>	σ	\<tau>	τ
\<upsilon>	υ	\<phi>	φ
\<chi>	χ	\<psi>	ψ
\<omega>	ω	\<Gamma>	Γ
\<Delta>	Δ	\<Theta>	Θ
\<Lambda>	Λ	\<Xi>	Ξ
\<Pi>	Π	\<Sigma>	Σ
\<Upsilon>	Υ	\<Phi>	Φ
\<Psi>	Ψ	\<Omega>	Ω
\<bool>	$\mathbb{B}$	\<complex>	$\mathbb{C}$
\<nat>	$\mathbb{N}$	\<rat>	$\mathbb{Q}$
\<real>	$\mathbb{R}$	\<int>	$\mathbb{Z}$
\<leftarrow>	$\leftarrow$	\<longleftarrow>	$\longleftarrow$
\<rightarrow>	$\rightarrow$	\<longrightarrow>	$\longrightarrow$
\<Leftarrow>	$\Leftarrow$	\<Longleftarrow>	$\Longleftarrow$
\<Rightarrow>	$\Rightarrow$	\<Longrightarrow>	$\Longrightarrow$
\<leftrightarrow>	$\leftrightarrow$	\<longleftrightarrow>	$\longleftrightarrow$
\<Leftrightarrow>	$\Leftrightarrow$	\<Longleftrightarrow>	$\Longleftrightarrow$
\<mapsto>	$\mapsto$	\<longmapsto>	$\longmapsto$
\<midarrow>	$-$	\<Midarrow>	$=$
\<hookleftarrow>	$\hookleftarrow$	\<hookrightarrow>	$\hookrightarrow$
\<leftharpoondown>	$\leftharpoondown$	\<rightharpoondown>	$\rightharpoondown$
\<leftharpoonup>	$\leftharpoonup$	\<rightharpoonup>	$\rightharpoonup$
\<rightleftharpoons>	$\rightleftharpoons$	\<leadsto>	$\leadsto$
\<downharpoonleft>	$\downharpoonleft$	\<downharpoonright>	$\downharpoonright$
\<upharpoonleft>	$\upharpoonleft$	\<upharpoonright>	$\upharpoonright$
\<restriction>	$\restriction$	\<Colon>	$::$

\<up>	↑	\<Up>	⇑
\<down>	↓	\<Down>	⇓
\<updown>	↕	\<Updown>	⇕
\<langle>	⟨	\<rangle>	⟩
\<lceil>	⌈	\<rceil>	⌉
\<lfloor>	⌊	\<rfloor>	⌋
\<lparr>	⦇	\<rparr>	⦈
\<lbrakk>	⟦	\<rbrakk>	⟧
\<lbrace>	⦃	\<rbrace>	⦄
\<guillemotleft>	«	\<guillemotright>	»
\<bottom>	⊥	\<top>	⊤
\<and>	∧	\<And>	⋀
\<or>	∨	\<Or>	⋁
\<forall>	∀	\<exists>	∃
\<nexists>	∄	\<not>	¬
\<box>	□	\<diamond>	◇
\<turnstile>	⊢	\<Turnstile>	⊨
\<tturnstile>	⊩	\<TTurnstile>	⊫
\<stileturn>	⊣	\<surd>	√
\<le>	≤	\<ge>	≥
\<lless>	≪	\<ggreater>	≫
\<lesssim>	≲	\<greatersim>	≳
\<lessapprox>	⪅	\<greaterapprox>	⪆
\<in>	∈	\<notin>	∉
\<subset>	⊂	\<supset>	⊃
\<subseteq>	⊆	\<supseteq>	⊇
\<sqsubset>	⊏	\<sqsupset>	⊐
\<sqsubseteq>	⊑	\<sqsupseteq>	⊒
\<inter>	∩	\<Inter>	⋂
\<union>	∪	\<Union>	⋃
\<squnion>	⊔	\<Squnion>	⨆
\<sqinter>	⊓	\<Sqinter>	⨅
\<setminus>	∖	\<propto>	∝
\<uplus>	⊎	\<Uplus>	⨄
\<noteq>	≠	\<sim>	∼
\<doteq>	≐	\<simeq>	≃
\<approx>	≈	\<asymp>	≍
\<cong>	≅	\<smile>	⌣
\<equiv>	≡	\<frown>	⌢

`\<Join>`	⋈	`\<bowtie>`	⋈
`\<prec>`	≺	`\<succ>`	≻
`\<preceq>`	⪯	`\<succeq>`	⪰
`\<parallel>`	‖	`\<bar>`	|
`\<plusminus>`	±	`\<minusplus>`	∓
`\<times>`	×	`\<div>`	÷
`\<cdot>`	·	`\<star>`	⋆
`\<bullet>`	•	`\<circ>`	∘
`\<dagger>`	†	`\<ddagger>`	‡
`\<lhd>`	⊲	`\<rhd>`	⊳
`\<unlhd>`	⊴	`\<unrhd>`	⊵
`\<triangleleft>`	◃	`\<triangleright>`	▹
`\<triangle>`	△	`\<triangleq>`	≜
`\<oplus>`	⊕	`\<Oplus>`	⨁
`\<otimes>`	⊗	`\<Otimes>`	⨂
`\<odot>`	⊙	`\<Odot>`	⨀
`\<ominus>`	⊖	`\<oslash>`	⊘
`\<dots>`	…	`\<cdots>`	⋯
`\<Sum>`	∑	`\<Prod>`	∏
`\<Coprod>`	∐	`\<infinity>`	∞
`\<integral>`	∫	`\<ointegral>`	∮
`\<clubsuit>`	♣	`\<diamondsuit>`	♢
`\<heartsuit>`	♡	`\<spadesuit>`	♠
`\<aleph>`	ℵ	`\<emptyset>`	∅
`\<nabla>`	∇	`\<partial>`	∂
`\<Re>`	ℜ	`\<Im>`	ℑ
`\<flat>`	♭	`\<natural>`	♮
`\<sharp>`	♯	`\<angle>`	∠
`\<copyright>`	©	`\<registered>`	®
`\<hyphen>`	-	`\<inverse>`	$^{-1}$
`\<onesuperior>`	^{1}	`\<onequarter>`	$\frac{1}{4}$
`\<twosuperior>`	^{2}	`\<onehalf>`	$\frac{1}{2}$
`\<threesuperior>`	^{3}	`\<threequarters>`	$\frac{3}{4}$
`\<ordfeminine>`	$^{\underline{a}}$	`\<ordmasculine>`	$^{\underline{o}}$
`\<section>`	§	`\<paragraph>`	¶
`\<exclamdown>`	¡	`\<questiondown>`	¿
`\<euro>`	€	`\<pounds>`	£
`\<yen>`	¥	`\<cent>`	¢
`\<currency>`	¤	`\<degree>`	°

`\<amalg>`	$\amalg$	`\<mho>`	$\mho$
`\<lozenge>`	$\lozenge$	`\<wp>`	$\wp$
`\<wrong>`	$\wr$	`\<struct>`	$\diamond$
`\<acute>`	´	`\<index>`	ı
`\<dieresis>`	¨	`\<cedilla>`	¸
`\<hungarumlaut>`	˝	`\<spacespace>`	
`\<module>`	$\langle$**module**$\rangle$	`\<some>`	ϵ

Bibliography

[1] David Aspinall. Proof General. http://proofgeneral.inf.ed.ac.uk/.

[2] Lawrence C. Paulson. *The Isabelle Reference Manual.* http://isabelle.in.tum.de/doc/ref.pdf.

[3] Markus Wenzel. *The Isabelle/Isar Reference Manual.* http://isabelle.in.tum.de/doc/isar-ref.pdf.

Index