A Tree-Based, Graphical Interface for Large Proof Development

Tom Schubert*
John Biggs
Department of Computer Science
Portland State University

Abstract

This paper describes a new X Windows based front end for the HOL theorem proving system, entitled "`xhol`". The centerpiece of `xhol` is a graphical display that depicts the entire active proof tree. This display presents not only a road map of what has and hasn't been proved, but also provides the user with clues about what techniques or tactics may be useful in proving the remaining unsolved subgoals. Support is provided for both the standard and *Prooftree* subgoal packages running on top of HOL88 and we anticipate the release of a version for HOL90 shortly. The tool also integrates enhancements to the goals package that allows the user to control the massive volume of output generated by large proof efforts.

The source code is freely available and requires only the X Toolkit that is bundled with the X Window System. We believe the design general enough to support incorporation of extensions by other developers.

1 Introduction

Workstations with moderate to high resolution bitmap displays are now widely available and there are a variety of user interface systems based on windowing paradigms. Increasingly, users have come to expect and rely on state-of-the-art data presentation to accompany application environments and HOL users should be no exception. In this paper, we present an X windows[1] based front end for HOL that we have entitled "`xhol`."

The `xhol` interface provides significant display enhancements along with the benefit of a mouse/menu interface that simplifies and speeds up communication with the HOL system. Interaction with HOL is accomplished through an emacs-style input window, a utility toolbar, and a collection of windows that display information about the state of a proof. Featured is a graphical display of the entire proof tree. This display presents not only a road map of what has and hasn't been proved, but also provides the user with clues about what techniques or tactics may be useful in proving the remaining unsolved subgoals.

*The author may be reached at: `schubert@cs.pdx.edu` and the source code can be obtained via anonymous ftp at `ftp.cs.pdx.edu` in the directory `/pub/faculty/schubert/xhol`.

[1] X Window System is a trademark of the Massachusetts Institute of Technology.

The interface project had a number of competing objectives. We wished to:

- Provide experienced users with a set of visual tools to speed up interactions with HOL.
- Assist new users in the development of proofs.
- Provide an interface similar to and compatible with the Emacs interface[Win90].

  Users may need to work on both bitmapped and non bitmapped displays. Certainly in academic settings, many users frequently wish to work over relatively low speed dialup lines that cannot adequately support graphic interfaces.

  It is also hoped that the interface is flexible enough to permit other developers to easily add other interface enhancements.
- Ensure an easy port of the interface to HOL90.

  With an academic class as the initial target audience and all system documentation written for the HOL88 dialect, it was deemed appropriate to build the interface for HOL88 and then later, support HOL90.

A central motivation for this project was to support the needs of large proof development. HOL currently provides little support for such efforts, as experienced during the UCD Viper proof project[ALL+93]. Subgoal packages such as *Prooftree* have developed useful proof administration facilities, but have not addressed managing the immense volume of output produced during large interactive proofs where assumption lists can easily grow to fill several hundred lines. Despite little changing from one step to the next during an interactive proof, HOL spends an enormous amount of time displaying the new state. Even small proofs, such as a simple 16 function MUX circuit proof, will produce far more information than is useful or can be quickly interpreted.

To assist large proof efforts, the tool integrates enhancements to the goals package that allow the user to control the massive volume of output generated by a large proof. This enhanced goal package was very helpful during the Viper proof and has been useful in other similar efforts. The interface also supports the multiple proof tree functions available with the *Prooftree* subgoal package[Kal91b].

1.1 Approach

We have incorporated a number of ideas from other packages in `xhol`, while minimizing the amount of ML code needed to support `xhol`. Delegating interface features to `xhol` rather than to HOL has two significant advantages: a) HOL is smaller and b) the interface can run on a separate machine. Large proofs demand a significant amount of memory and the cost of swapping can be very expensive, so moving the interface to another machine can be quite benefical.

We chose to write `xhol` using X windows and the *Xt Intrinsics* toolkit[MAS91] rather than as an interface on top of another system such as Centar[Cen91] or the Cornell Synthesizer Generator[RT89]. These other systems tend to require a large amount of resoures and are less likely to be as portable as one built on *Xt Intrinsics*, which is included in the X Window System.

The X Window System: The X Window System was developed at the Massachusetts Institute of Technology (MIT) to provide a hierarchy of resizable windows that support high-performance, device-independent graphics[KW89]. Unlike most window systems, X does not impose stylistic constraints on the construction of a user interface. In fact, X is really more a network protocol than a conventional window system. This protocol basis and portability has lead to X becoming an industry de facto standard. The code that implements X is freely distributed and has been ported to a wide range of hardware and software platforms.

The *Xt Intrinsics* toolkit provides a collection of tools that simplify the design of an application's user interface in the X windows environment by constructing *widgets*. A widget is a combination of a window or subwindow and associated input output semantics. Intrinsics attempts to be "policy–free," delegating policy, consistency, and style decisions to the application environment and widget set. While there is no standard widget set, MIT does provide sample implementation of a widget set built upon the Xt Intrinsics, called the Athena widget sets[Pet91]. This widget set provide a sufficient basis for `xhol` development.

1.2 Related Work

During the past several years, a number of user interfaces and proof support packages have been developed for the HOL theorem proving system. In [Kal91a], three distinct interface areas were listed as amenable to improvement by an interface: help information, term surgery, and proof management. Several projects have investigated these issues. A partial list includes:

- The `xholhelp` and the Theorem Retrieval System utilities. These tools are extremely useful for obtaining ML function documentation and finding already proven HOL theorems. More recently, the xmosaic HOL information server developed at BYU provides access to on-line manuals, technical reports, and keyword search access to *info-hol* mailing group messages[Win93].
- The Windows Library and `xholterm`. These packages have investigated techniques to perform subterm manipulation.
- The interface described in [Thé93] and [TBK92]. This interface supports a number of useful proof construction ideas and is built on the Centar interface system[Cen91].
- Tree based theorem proving interfaces. This style of interaction is quite natural and reflects the breaking down of goals into a collection of subgoals. An emacs tree interface[FAY91] and two tree based alternatives to the system default stack based goals package are available[Fle90, Kal91b].

2 XHOL Features

The `xhol` program allows interaction with HOL through both keyboard and mouse-selection input. The display provides five paned areas: a toolbar, an edit window, an output window, a goal window, and a tree window. An example display appears in Appendix B[2].

The toolbar runs the length of the top of the display, with the four windows positioned in a tiled (as opposed to overlapping) format below. Options are available that allow a user to customize the ordering of the windows. Each window provides distinct scrollbars on an as needed basis. While the ordering of windows is fixed during execution, the relative size of each window may be modified. Figure 1 presents a (multiple row) image of the toolbar. The tool bar provides an application message box and a collection of icons whose functionality will be described below.

One objective of the display layout was to minimize the amount of display space used for application overhead.

Since icons are smaller than their textual meaning, the use of icons helps minimize the required size of the tool bar. However, since the meaning of icons is not always self evident, when the mouse pointer is moved into an icon, the message box displays a textual description of the related function. When the mouse pointer is not on an icon, the message box displays the name of the currently loaded file. If the *Prooftree* subgoal package is used, the name of the active proof will be displayed in the message box.

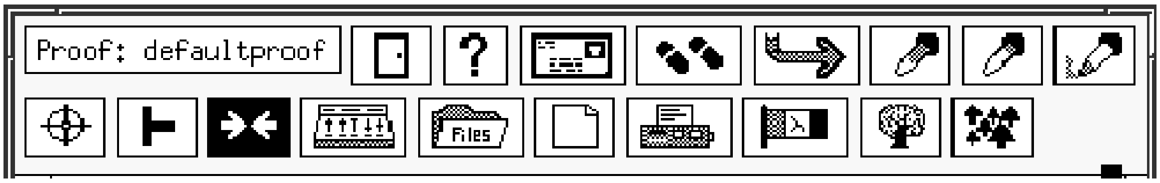

Figure 1: XHOL toolbar icons

A description of toolbar functions is deferred until after a discussion of the various windows. Briefly, the following functions are provided:

exit application	provide help information
send region to HOL	request HOL backup
request HOL rotate	extract tactics
extract a compressed tactic	insert extraction
display current goal	present proof log
compress tree node	application display controls (menu)
file operations (menu)	clear window (menu)
print window (menu)	send HOL a set_flag() request (menu)
Prooftree active proof functions	*Prooftree* forest functions.

2.1 The Input Window

The input window extends the functionality provided an Athena text widget. The input window allows the users to edit text with a set of emacs key functions for cursor movement, text selection, text deletion, searching, and file insertion. Note that the widget's behavior is similar to, but not identical to emacs. An extensive set of text selection functions are

[2] Due to the large size of captured display images, only a limited number of images will be presented in this paper.

also provided by the mouse (select insertion point, select word, select line, select paragraph, select region, select all). A summary of the key bindings is available in Appendix A.

Notable key functions for the HOL user include:

Send text to HOL: To send a specific text region, the user may select (highlight) any range of text, leave the cursor at the end of the region, and click on the send icon or press the send key. By default, send is bound to the escape key.

If there is no selected region, the current paragraph (all text between the previous blank line or the beginning of the text and the next blank line) is highlighted and sent to HOL. When a text region is sent to HOL, it remains highlighted until another region is sent or the user selects a different text region.

Send line to HOL: The user can elect for `xhol` to send each input line when the carriage return / enter key is pressed. This option may be selected from the application display controls menu, or by setting the appropriate X environment variable. If this option is chosen, users can insert new lines into the input file via the *Ctrl-m* key combination.

Template functions: Single key template functions to insert frequently used HOL functions such as `e(_ );;` and `e( REWRITE_TAC[_ ] );;` are provided. After insertion, the input cursor is positioned at the underscore. A mechanism to support a menu of user defined templates is currently being added. The HOL90 release will provide functions for inserting HOL term (`--" "--`) and HOL type (`==" "==`) parsing delimiters.

Move between paragraphs: Key functions allow a user to step between blank line separated text regions.

2.2 The Output Window

The output window displays all responses from HOL. An application display control option determines whether input text is also echoed in the output window. Unlike the emacs interface, this window is not editable.

2.3 The Goal Window

The goal window interacts with the tree window described in the next section. The window generally displays the current subgoal unless directed by function requests (tree window or toolbar) to display some other information.

Application display control options determine whether the window should be cleared before satisfying requests to display new information and whether goals are enumerated with a Dewey-style number (e.g. `1.2.1`). If a goal is expanded without multiple subgoals created, the new goal will be labeled with an alphabetic character suffix that distinguishes it from its parent (e.g. `1.2.1a`). Note that this labeling is more compact than the labeling done by the *Prooftree* subgoal package. For example, a *Prooftree* label such as `1.1.1.1.1.1.1`, would be displayed as `1f`.

2.4 The Tree Window

The tree window presents a graphical representation of the current proof state (an example is presented in Figure 2.). As goals are expanded, new subgoal nodes are added below the parent goal with lines indicating parent–child relationships.

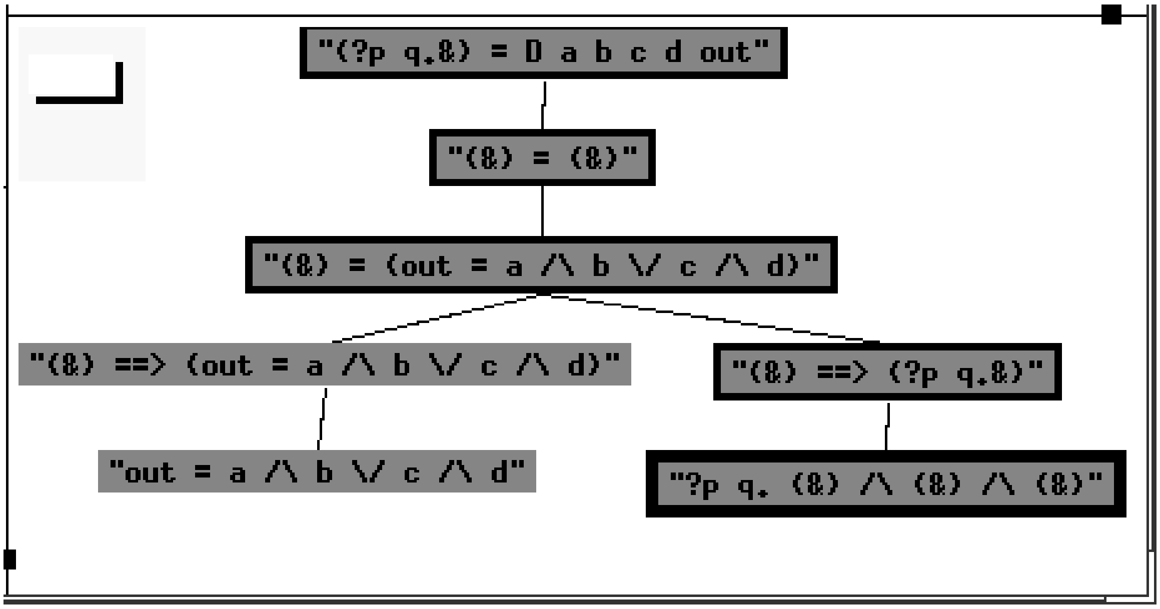

Figure 2: xhol tree example

Tree Node Text: Though the width of the node text (node label) is user settable, large space sizes prevent "the forest from being seen from the trees." Thus, node labels typically display less than the entire goal text. Rather than merely truncating the goal text, several application controls allow insightful information to be displayed. `xhol` provides the user with a compress text function (via the toolbar's *compress tree node icon*) that generates a displayed string with the maximum number of sub-expression levels that will fit in the given space.

The function is similar in spirit to HOL's `max_print_depth`, replacing non-displayed sub-expressions with an ampersand (`&`) holophrast. Several examples of this use can be seen in Figure 2. This mechanism greatly assists the user view the structure of a each goal. This structural view seems to greatly assist novice users in their selection of a potentially appropriate tactic to apply next. Note that the complete goal text is displayed in the *goal window*.

The *application display control icon* provides a number of additional tree node display options:

expand compressed tree nodes: Compressed nodes leave extra node space that can be used to partially fill sub-expression layers. If this option is enabled, the percent character (`%`) indicates the end of a partially filled subexpression.

enumerate tree nodes The Dewey-style numbers described in the *goal window* discussion above, are prepended to the tree node labels.

set tree node size: A popup window requests the user select a node size. Acceptable node sizes are 0–99.

blank fill tree nodes: Figure 2 shows a collection of tree nodes that are of varying size. One result is that the lines connecting nodes are not always vertical. Also note that the child subgoals do not line up evenly under the parent. If a user wishes to impose tree layout conformity, this option ensures that all node labels are the same size.

Tree Node Border: The thickness of the node border is intended to communicate a status of a subgoal status as follows:

Thick border: The node is the active goal.

No border: The node is part of a proven branch

Thin border: The node is part of an unproven branch.

Tree Scrolling: When the tree expands beyond the size of the window, a panner provides scroll access to the tree space (see top left-hand corner of Figure 2). The panner displays a rectangular region (the "canvas") of which only a smaller enclosed rectangular region (the "slider") is visible. The slider may be moved around by pressing, dragging, and then releasing Button 1.

Mouse buttons: Though the goal window displays the current goal, the user may choose to view other goals or the tactics applied to other nodes.

Button 1 pressed: If the mouse is positioned on a tree node, pressing the button toggles the node's "selection" for future toolbar functions. The tool bar functions *rotate*, *present proof log*, *extract proof tactics*, and *extract compressed proof tactic* will use the selected node as an argument to modify the toolbar function behavior. The selected node is displayed with reversed foreground and background colors (inverse video) and remains selected until another node is selected.

If the mouse is positioned in the tree window, but not on a tree node, the "selected" node is "unselected."

Button 2 pressed: If the mouse is positioned on a tree node, the complete goal for that node is displayed in the goal window.

Button 3 pressed: If the mouse is positioned on a tree node, the tactics applied to that goal are displayed in the goal window.

If the mouse is not positioned on a tree node, display of the panner is toggled.

2.5 Tool Bar Functions

Many of the toolbar's functions are common among window based application programs (exit, help, file operations, print operations, clear window operations) and will not be discussed below.

application display control menu: Clicking on this icon[3] causes the option menu to be displayed. Options are available to cause goal window expressions and tree node

[3]The icon is intended to look like a music/video mixer, but thhis has not been obvious to all users.

labels to be enumerated, HOL input to be echoed in the *output window*, the size of tree nodes to be set, and several other similar features.

compress tree node toggle: Clicking on this icon toggles the compression of tree nodes as discussed in the description of the *tree window*.

display current goal: Clicking on the cross-hairs icon causes the tree window to scroll so that the active goal node is displayed and causes the goals window to display the active goal.

present proof log: Clicking the turnstile icon displays a log of the current goal. If no tree node has been selected, the log begins with the root goal. The tree is traversed top to bottom, left to right. For each node, the goal is displayed in the `goal window`, followed by any user input for that goal. User input is indented with a vertical prepended to each line. A simple example output follows:

```
1.2.1 "!a b. ?c. c = AND a b"

     |  e( GEN_TAC );;

1.2.1a "!b. ?c. c = AND a b"

     |  e( GEN_TAC );;

1.2.1b "?c. c = AND a b"

     |  e( REWRITE_TAC[AND_DEF] );;

1.2.1c "?c. c = a /\ b"
```

send region to HOL: Clicking the postal letter icon sends the selected input text to be sent to HOL.

request HOL backup: Clicking the footprints icon sends the string `backup();;` to HOL.

request HOL rotate: Clicking the arrow icon sends the string `rotate();;` to HOL.

extract tactics: Clicking the eyedropper full icon causes the HOL input sent for the selected node and any children to be displayed in the goal window.

extract and compress tactics: Clicking on the eyedropper icon causes the HOL input sent for the selected node and any children to be compressed into a single tactic and displayed in the goal window.

insert extraction: Inserts an extraction into the input window.

send HOL a set_flag() request (menu):

Clicking on the flag icon causes a menu of possible strings a user may wish to send to HOL. Many of these HOL flags are provided by the goals package described in Section 3. The list includes:

- set/unset *show_types* flag.
- set/unset *show_goal* flag.
- set/unset show assumptions flag.
- set/unset show assumption list subset flag.
- set/unset enumerate assumptions flag.
- set assumption sublist
- set `max_print_depth` to 10.
- set `max_print_depth` to 500.
- current state request (sends `print_state 1;;`).

Prooftree: **manipulate/display current tree (menu):** This icon only appears if the *Prooftree* subgoal package will be used. Clicking on the icon will cause a menu of commands that provide information or or manipulate the current (focus) proof. Included are:

- current goal
- all subgoals
- current proof
- show tactic
- compact
- remove proof

Prooftree: **manipulate/display forest (menu):** This icon only appears if the *Prooftree* subgoal package will be used. Clicking on the icon will cause a menu of commands that provide information or or manipulate the entire complete set of proofs. Included are:

- show all proofs
- compact
- complete proof
- switch to another proof

2.6 Other Features

- X_interface flag: so that the same goals package may be used with or without `xhol`, the HOL's `X_interface` flag must be `true`, prior to setting a goal. When the flag is true, the goals package will send extra information to inform `xhol` about the goal state.

- Initialization file: `xhol` looks for a file `.xhol` in the user's home directory. If it is found, the contents is sent to HOL prior to any user input.

- Command line options:

  `-stdIn:` Send user input from `xhol`'s standard input stream to HOL.

`-stdOut:` Send all *output window* output to standard output stream.

By setting both flags on, users may run `xhol` from within emacs.

`-prooftree:` Many functions, such as `backup()` behave differently under *Prooftree*. This option to indicates the *Prooftree* subgoal package will be used.

filename If a filename is present, `xhol` loads it into the input window.

- Customization: All of the application display controls may be set by using the X server resource database utility `xrdb`. The `xhol` utility has been developed for a color display, but also works on non color terminals. Users may also wish to modify the author's default application color scheme.

  Like `xholhelp`, text can be sent to a printer, or any process which accepts input through a pipe. The print function expects the environment variable `HOL_PRINT_CMD` to be set appropriately, as in:

  ```
  setenv HOL_PRINT_CMD "lpr -Plw5"
  ```

3 Goal Package Enhancements

Though HOL supports both forward and goal oriented (backward) proof styles, most proof construction is performed using a backward strategy through the application of *tactics*. In conjunction with a goal package, which acts as a "bookkeeper," tactics allow a user to incrementally develop a proof by breaking a goal down into a (hopefully) simpler goal or set of goals. When all goals are proved, the goal package constructs the complete proof. At any given time during a proof's development, there may be a number of unproven subgoals. Each subgoal consists of a list of assumptions and a term that is to be proved[4].

The application of a tactic to a goal often does not cause a significant textual change to either the assumption list or the goal term. Yet, the entire state is displayed after each tactic's application. For large proofs, this may require literally hundreds of lines to be displayed. If a tactic causes a goal split to occur, thousands of lines may need to be displayed. Not only is the output time consuming, but the system user can find it difficult to interpret the results. Prudent use of the HOL flag `print_all_subgoals` and setting `max_print_depth` to single digit values can be quite helpful. Nonetheless, HOL may still produce more output than desired. To support user control over the type of information that is displayed, several new flags have been added to goal package.

Displaying only the goal or assumption list: Frequently, a user may wish to see only the new goal or assumption list. Two new flags "print_gl" and "print_asl" control whether either of these goal components is displayed. To simplify the setting of these flags a few commands supplement the expand (`e`), command. Using the commands `ea` or `eg` in place of `e`, sets the respective flag to false. The new command `eu` resets both flags to true.

[4]The term part of a goal-assumption list pair is commonly referred to as the "goal," though "goal term" or "goal consequent" might be more correct.

Displaying a subset of the assumption list: Assumption lists can grow rather large and frequently, a user may wish to see a subset of the assumption list—perhaps only the most recently added assumption. A new assignable variable "asl_print_list" and flag "print_asl_list" establish and control the displaying of a subset of the assumption list. For example, the following commands will cause only the last three assumptions to appear.

```
asl_print_list := [3;2;1];;
set_flag(`print_asl_list`,true);;
```

A collection of simple functions is provided to perform these operations.

Displaying assumption list numbers: Setting the flag `print_asl_cnt` to `true` causes an assumption number to be displayed to the left of the assumption. This is particularly useful when a user wishes to limit the number of assumptions displayed.

As it is considered bad practice to access the assumption list directly (e.g. `(el 1 asl)`), we suggest the use of tactics like `FILTER_ASM_REWRITE_TAC[]`. The package includes an improved version of the ML `lines` function, described in the tutorial, is provided. Using `FILTER_ASM_REWRITE_TAC[]` with the new `lines` function, a variety of situations can be easily handled (e.g. rewrite with only facts about the state for time `t+13`). The `lines` function can also be used to delete assumptions from the assumption list. David Shepherd has also suggested using the following tactic:

```
let EXTRACT_ASSUM_THEN ttac t =
UNDISCH_TAC t THEN DISCH_TAC THEN POP_ASSUM ttac;;
```

This tactic extracts assumption `t` from the assumption list and feeds it to theorem tactic ttac. For example the following usage would take an assumption `"0=n"` out of the list and then assume `"n=0"`.

```
EXTRACT_ASSUM_THEN (ASSUME_TAC o SYM) "0=n";;
```

Backup Semantics: To simplify the interface to `xhol`, the semantics of backup has been changed. The original version would undo the result of a rotate. The new version pops the goal stack, eliminating the result of the last tactic applied.

4 Conclusions

A preliminary version of `xhol` was "class tested" and the overall results positive. The visual proof tree display helped students navigate through a proof's development and the ability to quickly reuse successful tactics on siblings was frequently utilized. Several changes have been made based on feedback and the list of desired enhancements has grown fairly long. We hope that the framework developed is general enough to support extensions and the integration of other utilities, such as the Window Library.

We anticipate the release of a version for HOL90 prior to publication of this paper. Other enhancements that will be shortly forthcoming include automatic tree compression, tactic factoring, and enhancing the goals package to send only changes to the current state.

Acknowledgements: The authors would like to thank Jim Alves-Foss and Leslie Young for the assistance in the development of this package.

References

[ALL+93] Tej Arora, Tony Leung, Karl Levitt, Tom Schubert, and Phillip Windley. Report on the UCD microcoded VIPER verification project. *1993 International Workshop on Higher Order Logic Theorem Proving and its Applications*, 1993.

[Cen91] The Centar System, 1991.

[FAY91] George Fink, Myla Archer, and Lie Yang. PM: A proof manager for HOL and other provers. *Proceedings of the 1991 International Workshop on the HOL Theorem Proving System and its Applications*, August 1991.

[Fle90] R. A. Fleming. An experimental new subgoal package, 1990.

[Kal91a] Sara Kalvala. Developing an interface for HOL. *Proceedings of the 1991 International Workshop on the HOL Theorem Proving System and its Applications*, August 1991.

[Kal91b] Sara Kalvala. *A New Subgoal Package for HOL.* University of Cambridge, Computer Lab, 1991.

[KW89] Stephen G. Kochan and Patrick H. Wood, editors. *UNIX Networking.* Haden Books, 1989.

[MAS91] Joel McCormack, Paul Asente, and Ralph Swick. *X Toolkit Intrinsics — C Language Interface, X Window System, X Version 11, Release 5.* MIT X Consortium, 1991.

[Pet91] Chris D. Peterson. *Athena Widget Set — C Language Interface, X Window System, X Version 11, Release 5.* MIT X Consortium, 1991.

[RT89] Thomas Reps and Tim Teitelbaum. *The Synthesizer Generator.* Springer Verlag, 1989.

[TBK92] Laurent Théry, Yves Bertot, and Gilles Kahn. Real theorem provers deserve real user-interfaces. *Proceedings of the Fifth ACM SIGSOFT Symposium on Software Development Environments*, December 1992.

[Thé93] Laurent Théry. A proof development system for HOL. *Proceedings of the 1993 International Workshop on the HOL Theorem Proving System and its Applications*, September 1993.

[Win90] Phillip J. Windley. Using HOL inside EMACS. Technical Report CS-90-01, University of Idaho, 1990.

[Win93] Phillip J. Windley. Documentation for the HOL theorem proving system. http://lal.cs.byu.edu/lal/hol-documentation.html, 1993.

A Edit Window Key Bindings

Movement

Ctrl-f	Forward Char	Ctrl-b	Backward Character
Ctrl-n	Next Line	Ctrl-p	Previous Line
Shift-Ctrl-n	Next Blank Line	Shift-Ctrl-p	Previous Blank Line
Ctrl-v	Next Page	Meta-v	Previous Page
Meta-f	Forward Word	Meta-b	Back Word
Meta-[	Beginning Of Paragraph	Meta-]	End Of Paragraph
Ctrl-a	Beginning Of Line	Ctrl-e	End Of Line
Meta-<	Beginning of File	Meta->	End of File
Ctrl-z	Scroll One Line Up	Meta-z	Scroll One Line Down

Deletion

Ctrl-d	Delete Next Character	Ctrl-h	Delete Previous Character
Meta-d	Delete Next Word	Meta-h	Delete Previous Word
Meta-D	Delete Previous Word	Meta-Bksp	Delete Previous Word
Ctrl-k	Kill To End Of Line	Meta-k	Kill To End Of Paragraph
Ctrl-w	Kill Selection	Meta-D	Kill Word
Ctrl-y	Unkill	Meta-y	Insert Current Selection
Meta-Shift-Del	Kill Previous Word	Meta-Shift-Bksp	Kill Previous Word

Miscellaneous

Ctrl-l	Redraw Display	Meta-q	Form Paragraph
Ctrl-s	Search/Replace Forward	Ctrl-r	Search/Replace Backward
Ctrl-m	Newline	Ctrl-j	Newline and Indent
Ctrl-o	Newline and Backup	Ctrl-t	Transpose Characters
Ctrl-u	Multiply by 4	Ctrl-g	Multiply Reset
Meta-i	Insert File		

HOL templates

Meta-g	Insert set_goal statement	Meta-e	Insert expand statement
Escape	Send selection to HOL	Meta-r	Insert rewrite statement

Button 1 Down/Up	Start/End Selection	Double click	Select Word
Button 2 Down	Insert Selection	Triple click	Select Line
Button 3 Down/Up	Extend/End Selection	Quad click	Select Paragraph
		Quint click	Select Buffer

B xhol Application Window