Using HOL inside EMACS

Phillip J. Windley

Technical Report No. CS-90-01

December, 1990

Department of Computer Science
University of Idaho

ABSTRACT

This report gives a brief tutorial on the *Emacs/HOL* interface for HOL users. The *Emacs/HOL* interface as a convenient means of interacting with HOL. The report describes how to use the interface and gives instructions on how to set up the interface in a UNIX environment.

This work was sponsored, in part, under Boeing Contract NAS1-18586, Task Assignment No. 3, with NASA-Langley Research Center.

1. Introduction.

One of the most confusing aspects of using HOL for the novice is developing an environment in which proofs inside HOL can be done easily. This report aims to guide a new user (or one who has not used the interface program between GNU–Emacs and HOL) through the steps required to get everything set up and working. This section will give a brief introduction to HOL and Emacs. The remainder of the report includes an instruction manual of sorts for using the interface and setting up the interface in a UNIX environment.

1.1 HOL.

HOL is a general theorem proving system developed at the University of Cambridge [1] [2] that is based on Church's higher-order logic. Church developed higher-order logic as a foundation for mathematics, but it is a promising language for describing computational systems of all kinds. HOL can be used for proving properties of anything that can be expressed in higher–order logic including hardware and software. [3] [4]

HOL has a built–in meta–language for manipulating the theorem prover and HOL terms called ML. ML is a powerful programming language in its own right and has been described in [5]. ML is a type polymorphic, lambda calculus–based functional language. It has a powerful type inference mechanism that frees the programmer from having to specify types except in those circumstances where it is impossible for the interpreter to infer them. Because HOL is built on top of ML, it inherits much of ML's power and functionality.

1.2 GNU Emacs.

GNU Emacs is a member of the Emacs family of editors. It is fully customizable and provides a language, *e-lisp,* for extending the editor and providing new functions. This report assumes a general familiarity with Emacs and how it is used. Readers requiring more information are directed to [6] and [7]. GNU Emacs was chosen since it provides a very powerful language for extending the editor and built-in commands for working with processes from within the editor.

In this report, we will use the standard Emacs convention for showing characters typed in Emacs. Normal characters which are assigned graphical symbols such as "a" and ")" will be shown in typewriter type as `a` and `)`. Characters which are not assigned a graphical symbol are called control characters and are written as `C-b`, which means that the "b" key is pressed while the control key is held down. Emacs also makes use of a *meta-key* (usually the escape key). Command key sequences that use the meta-key are written `M-f`, meaning that the "f" key is pressed while the meta-key is pressed (or immediately after in the case where the escape key is used as the meta–key).

Commands are given to Emacs either by typing a sequence of characters that have been bound to a particular command or they can be given by name. For example, to execute the command *forward-char* which moves the cursor forward one character, we can either press the `C-f` key sequence to which it is bound, or we can type `M-x forward-char`. `M-x` is the key sequence for running commands by name.

2. Using the *Emacs/HOL* Interface.

This section is intended as an introduction to using the *Emacs/HOL* interface. A full account of the available commands is included in the next section. Furthermore, it assumes that the set up procedures outlined in later chapters have been followed.

The *Emacs/HOL* interface allows a user to run HOL as a subprocess of the GNU Emacs editor. When HOL is running under Emacs, the user may interact with it by sending it text from the editor. Being able to send arbitrary text strings to HOL from within an editor has a number of advantages:

- Because the text is entered in the editor, it can be easily changed.
- Program fragments that reside in text files can be tested for correctness.
- Responses from the editor are stored in a text buffer that can be browsed and even saved to disk for later review.

Using the *Emacs/HOL* interface is really quite simple. First, start HOL by typing `M-x run-hol`. This starts the interpreter, initializes the communication links, and creates an interaction buffer, where the results returned by HOL are printed. The user interacts with HOL using commands built into the interface.

Suppose that we wish to use HOL to prove that $1 + 1 = 2$. We would normally type the HOL expression

```
set_goal([],"1 + 1 = 2");;
```

at the HOL prompt. Using the *Emacs/HOL* interface we can either type the goal directly in the interaction buffer, or we could type it in a file and then send the definition to HOL using commands from the interface. I like to open a file for each theory and open two windows in EMACS: one showing the theory file I'm currently working on and one showing the HOL interaction buffer. Whether the HOL expressions are typed directly in the interaction buffer or stored in a text file, we must tell Emacs to communicate the expression to HOL. We can do this by placing the cursor before the expression and typing `C-c n` which is bound to the Emacs command `interpreter-send-next-expression`. Alternately, we can place the cursor directly after the expression and type `C-c p` which is bound to the Emacs command `interpreter-send-previous-expression`. The *Emacs/HOL* interface uses blank lines, in part, to determine where expressions start and end, so it is important that every expression be delineated by blank lines.

Since all of the information returned from HOL is put into the interaction buffer, HOL will echo the goal in the interaction buffer after we send the goal to HOL.

We proceed with the proof in the same manner we would directly, except that we type our commands in the file buffer and HOL replies in the HOL interaction buffer. For example, we can prove the previous goal using the following tactic:

```
expand(REWRITE_TAC [num_CONV "2"; num_CONV "1"; ADD_CLAUSES]);;
```

We send the command to HOL using the usual interaction commands after which HOL replies the the goal has been proven.

Note that we can either send expressions to HOL individually using the appropriate interaction commands, or we can type in several HOL expressions and then send the collection to HOL using the Emacs command `interpreter-send-region` which is bound to `ESC C-z`. In any case, the results from HOL will be stored in the HOL interaction buffer.

3. The *Emacs/HOL* Interface.

This section describes the each of the commands available in the *Emacs/HOL* interface.

As mentioned earlier. The interpreter is started by running the function *run-hol*. After HOL has been started, HOL expressions may be either entered in the interaction buffer, or sent to the interpreter from some other buffer. In buffers with an HOL local major mode, the same key bindings available for communicating with the interpreter in the interaction buffer will be available as well.

The following commands are available for sending commands and text to the interpreter:

a.) **Interpreter-send-previous-expression** sends the expression preceding the cursor to HOL. The function that Emacs uses to search for the beginning of an expression is customizable.

b.) **Interpreter-send-next-expression** is similar to interpreter-send-previous-expression, except that the expression following the cursor is sent to HOL.

c.) **Interpreter-send-region** sends a region of text to the HOL. The region can contain any number of expressions.

d.) **Interpreter-send-buffer** sends an entire buffer to HOL.

e.) **Reset-interpreter** kills the inferior process and starts it again. This command can be used when a new HOL session is desired.

f.) **Interpreter-yank-previous-send** yanks the last thing sent to HOL, placing it in the current buffer after the cursor.

g.) **Interpreter-select-process-buffer** selects the HOL interaction buffer and places the cursor in it. If the HOL interaction buffer is not currently displayed, a window is created for it.

h.) **Interpreter-recenter-output-buffer** recenters the HOL interaction buffer so that the last line is in the middle of the interaction buffer.

i.) **Interpreter-toggle history** toggles the history function off and on. When the history function is on, the interface will write both the string sent to HOL and HOL's

Key Binding	*Interpreter command*
C-c p	interpreter-send-previous-expression
C-x C-e	interpreter-send-previous-expression
ESC RET	interpreter-send-previous-expression
C-c n	interpreter-send-next-expression
ESC C-z	interpreter-send-region
ESC o	interpreter-send-buffer
C-c C-y	interpreter-yank-previous-send
C-c C-s	interpreter-select-process-buffer
C-c C-l	interpreter-recenter-output-buffer
C-c C-t	interpreter-toggle-history
C-c C-b	xhol-backup
C-c C-p	xhol-print-top-goal
C-c C-r	xhol-rotate-goal-stack

Figure 1. Key Bindings for the Interpreter Interaction Buffer

reply in the interaction buffer. When off, only HOL's reply is written in the interaction buffer. Intially the history function is off.

In addition to commands for sending expressions to HOL and controlling the interpreter there are also several commands available for controlling the HOL goal stack.

a.) **xhol-backup** causes the last action that affected the goal on the goal stack to be undone.

b.) **xhol-print-top-goal** prints the goal on the top of the goal stack.

c.) **xhol-rotate-goal-stack** rotates the goal stack once. An argument can be specified using the GNU-Emacs `universal-argument` command (`C-u`). The goal stack is rotated by the number in the argument.

These commands are intended primarily as examples of the kinds of things that can be done using the *Emacs/HOL* interface. Using *e-lisp,* many other interesting commands could be written.

Most of the commands mentioned in this section are bound to one or more key sequences for convenience. The table in Figure 1 list the commands available for communicating with the interpreter and the keys to which they are bound.

4. Setting Up the *Emacs/HOL* Interface.

Before the *Emacs/HOL* interface can be used for the first time, there are several things that need to be done.

4.1 The Emacs Load Path.

The *e-lisp* files containing the code for the interface must be placed in a directory that is in the Emacs load path. This is generally done by setting the `load-path` variable in the Emacs start–up file, `.emacs`. A sample `.emacs` file is included in Appendix A.

Suppose that the files which make up the *Emacs/HOL* interface were put in a directory called `emacslib` rooted in the user's home directory. The following s–expression shows how the directory `emacslib` could be added to the load path.

```
(setq load-path (cons (expand-file-name "~/emacslib")
                      load-path))
```

4.2 Autoloading.

The autoload tables in Emacs should be modified to automatically load the appropriate files when certain commands are given. Specifically, the command `hol-mode` should cause Emacs to load the file `hol.el` where the HOL major mode information is stored. Also, the command `run-hol` should automatically load the file `xhol.el` which contains the code necessary to start an HOL session from within Emacs. The following s–expressions show how this is done:

```
(autoload 'hol-mode "hol"
          "HOL mode for GNU Emacs." t)
```

```
(autoload 'run-hol "xhol"
          "Execute hol from within emacs." t)
```

4.3 Emacs Hooks.

An Emacs hook is a function that, if defined, is run at a particular spot in the initialization of some system. The *Emacs/HOL* interface provides two hooks so that the HOL environment can be customized by the user. By defining functions for these hooks in their `.emacs` file, users can affect the binding of variables before HOL is started and the actions taken by HOL after it is started.

The first of hook provided by the *Emacs/HOL* interface is called `hol-var-hook` and is used to change the variables in the interface. Here is the definition of a sample `hol-var-hook` that causes HOL to start up in a particular directory.

```
(setq hol-var-hook '(lambda ()
   (progn
      (setq interpreter-default-directory "~/hol/theories"))))
```

Even though this causes HOL to focus its attention on a particular directory, files can be loaded into Emacs and sent to HOL from any directory. Only the HOL theory files (files that end in a file extension of `.th`) are stored in the directory set in the `hol-var-hook`.

The other hook to the interface allows commands to be sent to HOL after it has been started. For example, the following s–expression defines a hook that causes HOL to automatically load an initialization file after it has started.

```
(setq hol-start-hook '(lambda ()
   (progn
      (interpreter-wait-for-process)
      (interpreter-send-string
          "loadf `~/hol/ml/init`;;\n"))))
```

This s–expression deserves some comment. The first expression in the `progn` sequence, `(interpreter-wait-for-process)`, blocks until HOL has started and is ready to accept commands. The second line sends a string to HOL. Note that it this can be any valid HOL command. This one merely tells HOL to load the file `init.ml` and gives a path to the file. Notice the `\n` at the end of the string. This is the *e-lisp* representation for a newline character and must be included.

4.4 The Initialization File.

The initialization file mentioned in the last paragraph is a good place to put any commands that should be executed whenever HOL is started. HOL contains a flag to turn off the prompt. When HOL is run inside Emacs, the prompt provides very little information and is usually a nuisance, so I like to turn it off. In addition, HOL has an internal search path for libraries and files that it loads that can be set in the initialization file as well. A sample initialization file is contained in Appendix B.

4.5 The Customization File.

The *Emacs/HOL* interface is built on top of a more general interpreter interface for Emacs that is described in [8]. The customization file should not generally have to be changed. This section provides a general description of the customization file; interested readers are referred to [8].

The customization file that allows HOL to work with the interpreter interface is called `xhol.el`. The customization file:

1.) Adds key bindings for the commands in the interpreter interface to the mode-map for HOL.

2.) Defines macros so that Emacs commands for interpreter interface are given names specific to the HOL interpreter.

3.) Initializes the global variables described previously.

4.) Defines functions that are not actually needed to run the interpreter under Emacs, but which provide a convenient way for users to interact with the interpreter.

5.) Tells the interpreter interface how to find the beginning and end of an expression.

The complete text of the customization file, `xhol.el`, is contained in Appendix C.

Several of the variables and functions defined in `xhol.el` deserve further comment:

a.) **interpreter-program-name** is the name of HOL on your system. This can be a hard path or a simple call that relies on the user's `path` variable.

b.) **interpreter-connection-type** determines whether UNIX pipes or pseudoterminals will be used for communication between HOL and Emacs. In general, pseudoterminals are preferred to pipes since pipes tend to become full and lose data.

c.) **interpreter-default-directory** is the directory that you wish HOL to start in. The default is the current directory (nil). This is usually changed in the user's Emacs initialization file using `hol-var-hook`.

d.) **interpreter-default-output** is the function used to print results from HOL in the HOL interaction buffer. If you don't like the way results are formatted, this is the place to start. The interpreter interface has a more sophisticated mechanism for handling interpreter results, but this requires that the interpreter be modified so that it knows it is running under Emacs. [8] describes the operation of the process filter in more detail.

e.) **hol-backward-expr** searches backward from the point (current cursor position) to the front of the last complete HOL expression. A more sophisticated function could undoubtably be constructed, but this one works in most circumstances.

f.) **hol-forward-expr** is similar to `hol-backward-expr`, except that it searches forward to the end of the next complete HOL expression.

g.) **xhol-commands** is the function that initializes the keymap for the local major mode. To add a new function, include a new `define-key` call with the desired key binding and the name of the function.

5. Conclusion.

This paper has described a user interface to HOL that was implemented in GNU–Emacs using *e-lisp*. The interface provides a means of sending HOL expressions to an child HOL process from a file and saves HOL responses in an interaction buffer for browsing and editing.

The *Emacs/HOL* interface provides basic user interface that is extensible since it is written on top of Emacs using *e-lisp*. Many new functions for interfacing with HOL can be defined using *e-lisp* and integrated into the interface. For example, one common means of using HOL proceeds in two steps: (1) proof discovery with HOL's interactive proof commands and (2) writing a batch command so that the proof can be replayed for reverification when other parts of the theory change. One could write a function in *e-lisp* that automates much of the tedious editing that takes place when a batch command is created from an interactive proof.

References

[1] Gordon, Mike, "A Proof Generating System for Higher–Order Logic," University of Cambridge Computer Laboratory Technical Report No. 103, January, 1987.

[2] Camilleri, Albert, Mike Gordon, and Tom Melham, "Hardware Verification using Higher Order Logic" in *From HDL Descriptions to Guaranteed Correct Circuit Designs*, D. Borrione, ed., Elsevier Scientific Publishers, 1987.

[3] Joyce, Jeffery J., "Using Higher–Order Logic to Specify Computer Hardware and Architecture," *Proceedings of the IFIP TC10 Working Conference on Design Methodology in VLSI and Computer Architecture,* D. Edwards, ed. North–Holland, Amsterdam, 1988.

[4] Joyce, Jeffery J., "Formal Verification and Implementation of a Microprocessor" in *VLSI Specification, Verification, and Synthesis,* G. Birtwhistle and P. Subrahmanyam, eds., Kluwer Academic Publishers, Boston, 1988, pp. 129–157.

[5] Gordon, M., R. Milner, and C. Wadsworth, *Edinburgh LCF,* Springer Verlag Lecture Notes in Computer Science No. 78, Chapter 2, 1979.

[6] Stallman, Richard, "GNU Emacs Manual", *Free Software Foundation*

[7] Lewis, Bill *et. al.,* "GNU Emacs Lisp Manual", *Free Software Foundation*

[8] Windley, Phillip J., "Running Interpreters under GNU Emacs," *University of California, Davis, Division of Computer Science Technical Report CSE-89-2*, March, 1989.

Appendix A. The Emacs Initialization File.

This appendix contains a sample `.emacs` file containing the definitions necessary to run HOL using the *Emacs/HOL* interface.

```
; set up the load path
(setq load-path (cons (expand-file-name "~/emacslib")
load-path))

;;; stuff to autoload
(autoload 'hol-mode "hol"
          "HOL mode for GNU Emacs." t)

(autoload 'run-hol "xhol"
          "Execute hol from within emacs." t)

;;; a-list stuff
(setq auto-mode-alist (cons '("\\.hol$" . hol-mode) auto-mode-alist))
(setq auto-mode-alist (cons '("\\.ml$" . hol-mode) auto-mode-alist))


;;; hol hooks

(setq hol-start-hook '(lambda ()
   (progn
      (interpreter-wait-for-process)
      (interpreter-send-string
          "loadf `~/hol/ml/init`;;\n"))))

(setq hol-var-hook '(lambda ()
   (progn
      (setq interpreter-default-directory "~/hol/theories"))))
```

Appendix B. The HOL Initialization File.

This appendix contains a sample `init.ml` file that sets up HOL for execution from within Emacs.

```
%-----------------------------------------------------------------

   File: init.ml
   Author: PJWindley (University of Idaho)

   Purpose: initialize HOL88 for running under emacs.

-----------------------------------------------------------------%

%-----------------------------------------------------------------
Since this file is only called if I'm running HOL from within
Emacs, I want to turn my prompt off.
-----------------------------------------------------------------%
set_flag (`prompt`,false);;

%-----------------------------------------------------------------
Search the system directories first.  Note that the trailing '/'
has to be on each path component for this to work.  Personal
tactics are stored in ~/hol/tactics, personal ML files in
~/hol/ml and theory files in ~/hol/theories.
-----------------------------------------------------------------%
set_search_path (search_path() @ [`~/hol/tactics/`;
                                  `~/hol/ml/`;
                                  `~/hol/theories/`;
                                 ]);;
```

Appendix C. The HOL Customization File.

This appendix contains the customization file that defines *e-lisp* variables needed to run HOL from within Emacs. These definitions will not normally need to be changed.

```
;; xhol -- Run HOL from within emacs
;; Principle author: Phillip J. Windley, University of Idaho
;; November 26, 1988

;; This file must be run in conjunction with interpreter.el

(require 'hol)
(require 'interpreter)

;;; Functions and variables for specifying the interpreter to use

;;; add interpreter key bindings to mode map
(interpreter-evaluation-commands hol-mode-map)

;;; add xhol specific key bindings to mode map
(defun xhol-commands (keymap)
  "Add specialized xhol command key bindings to a keymap.
This may be used on more than one keymap.  They are defined in
interpreter interaction mode and can be added to any other mode map
that is appropriate."

  (define-key keymap "\ C-c\ C-b" 'xhol-backup)
  (define-key keymap "\ C-c\ C-r" 'xhol-rotate-goal-stack)
  (define-key keymap "\ C-c\ C-p" 'xhol-print-top-goal))

(xhol-commands hol-mode-map)
(xhol-commands interpreter-interaction-mode-map)

(fset 'run-hol 'run-interpreter)

(defvar interpreter-program-name "hol88"
   "*Program invoked by the `run-interpreter' command.")

(defvar interpreter-under-emacs-switch " "
   "*Switch to pass to interpreter to tell it that its
running under emacs.")

(defvar interpreter-program-arguments nil
   "*Arguments passed to the interpreter by the `run-interpreter' command.")
```

```
(defvar interpreter-buffer-name "*hol*"
   "*Name of buffer to start interpreter in.")

(defvar interpreter-process-name "hol"
   "*Name to use for process identification.")

(defvar interpreter-mode-line-name "HOL: "
   "*String to print on modeline before process status.")

(defvar interpreter-connection-type t
   "*Set to t to use pty's, nil to use pipes.")

(defvar interpreter-default-directory nil
   "*Default directory to start interpreter in, nil for don't care.")

;;; For HOL, % is comment string, but it also has special meaning
;;; to format in emacs, so we double it.

;;; leave these empty for now.
(defvar interpreter-begin-comment ""
   "*String that begins comments in the interpreter.")

(defvar interpreter-end-comment ""
   "*String that ends comments in the interpreter.")

(defvar interpreter-result-highlight-string ""
   "*Default string to highlight results from the interpreter
in the Interaction buffer.")

;;; var hooks are run before starting the process to change the default
;;; values of variables in the xhol file.
(setq interpreter-var-hook hol-var-hook)

;;; start hooks are run after the process has been started, they can be
;;; used to send messages to the interpreter.
(setq interpreter-start-hook hol-start-hook)



(fset 'interpreter-forward-expr 'hol-forward-expr)
(fset 'interpreter-backward-expr 'hol-backward-expr)

;;; output routines and process filter association list

;;; HOL doesn't supply command characters for emacs, so this isn't used.
(defvar interpreter-process-filter-alist nil
```

```
  "Table used to decide how to handle process filter commands.
Value is a list of entries, each entry is a list of two items.

The first item is the character that the process filter dispatches on.
The second item is the action to be taken, a function.

When the process filter sees a command whose character matches a
particular entry, it calls the handler with two arguments: the action
and the string containing the rest of the process filter's input
stream.  It is the responsibility of the handler to invoke the action
with the appropriate arguments, and to reenter the process filter with
the remaining input.")


(defun interpreter-default-output (string)
   (if (and (not (string-equal string "\n"))
            (not (string-equal "\n " string )))
      (interpreter-write-with-message "" string)))


;;; functions for finding beginning and end of expression.

;;; search for a double ;; and then the preceding blanks line.
(defun hol-backward-expr ()
   "Searches for the blank line before the last two semi-colons."
   (progn
      (re-search-backward ";;" nil t)
      (re-search-backward "^[ ]*$" nil t)))

;;; search for a double ;; and then the following blanks line.
(defun hol-forward-expr ()
   "Searches for the blank line following next two semi-colons."
   (progn
      (re-search-forward ";;" nil t)
      (re-search-forward "^[ ]*$" nil t)))


;;; new key bindings for functions defined here

(defun xhol-backup ()
   (interactive)
   (interpreter-send-string "backup();;\n"))

(defun xhol-print-top-goal ()
   (interactive)
```

```
   (interpreter-send-string "top_goal();;\n"))

(defun xhol-rotate-goal-stack (number)
   (interactive "P")
   (if number (interpreter-send-string (concat "rotate("
                                               (int-to-string number)
                                               ");;\n"))
              (interpreter-send-string "rotate(1);;\n")))
```