User Manual for the Larch Development Environment

Michele J. Morin and Betty H.C. Cheng

Department of Computer Science
Michigan State University
East Lansing, MI 48824
email: {chengb}@cps.msu.edu
ph: (517) 355-8344, fax: (517) 432-1061

April 1994

1 Introduction

The Larch Development Environment, LDE, provides a graphical user interface for the development of formal specifications using the Larch family of specifications languages. Larch uses a two-tiered approach to formal specifications. The first tier, using the Larch Shared Language (LSL), provides programming language independent specifications defining the characteristics and behavior of abstract data types. The second tier, using the Larch Interface Languages (LIL), provides specifications of the components of a software system. The interface language is particular to the programming language used for the software system and defines the interfaces of the components of that system. LDE offers support for constructing and editing specifications, browsing and referencing existing specifications, and proof construction with LSL specifications.

1.1 Hardware Requirements

The Larch Development Environment runs on a Sun SPARCstation 10 using the Solaris2.3 operating system. It may also be run on a SPARCstation 2, but it will run much slower. No other particular hardware is required.

1.2 Software Requirements

To run LDE, the Larch syntax checkers, lsl, lcl, and lcc, and the Larch Prover, LP, must be available. LDE was written to work with versions 2.4 of all three syntax checkers and the Larch Prover. It should run other versions, but if there are major differences between versions, this is not guaranteed. For the LSL syntax checker, the *-lp* option must be available. The full paths for these tools must be put into the *run_ld* file, as discussed in Section 1.3.1.

To compile LDE, the iXBUILD 3.1 libraries, the Motif and Xlib libraries, the Sun compiler CC, and the UNIX *make* and *ar* utilities must be available.
NOTE: The iXBUILD 3.1 libraries must be compiled with the Motif libraries, not the SUNWmotif libraries. If they are not, a link error occurs due to the use of XmStrings in the SUNWmotif libraries.

1.3 Functionality Description

The Larch Development Environment is very useful for both experienced and new Larch specification developers. For experienced developers, it alleviates the awkwardness and tediousness of the command line environment of the existing Larch tools and it aids in lowering the learning curve for new developers. This section provides a description of the functionality of each of LDE's windows.

1.3.1 Getting Started

Before starting LDE, you must first make sure the paths for the syntax checkers, LP, lde, and the specifications are correct in the *run_ld* file. This file is shown in Figure 1. This figure has line numbers added in for reference.

- Line 1 contains the directory path for the file LDE. This is the Xdefaults file for graphical interface. The colors and fonts in this file may be changed by the user as desired.
- Line 2 contains the home directory for the Larch tools.

```
1 setenv XAPPLRESDIR /home/canidae/u20/morinm/cps898/ldt
2 setenv LARCH_HOME /opt/Larch
3 setenv SPEC_PATH .:/home/arctic/u03/morinm/cps814/specs:
$LARCH_HOME/LSL/lib:$LARCH_HOME/LCL/lib:$LARCH_HOME/lcc/speclib/lsl:
4 setenv LPCOMM_PATH /home/canidae/u20/morinm/cps898/ldt
5 setenv LARCH_HOME_BIN $LARCH_HOME/bin
6 setenv LDT_LSL_PATH $LARCH_HOME_BIN
7 setenv LDT_LCL_PATH $LARCH_HOME_BIN
8 setenv LDT_LCC_PATH $LARCH_HOME_BIN
9 setenv LDT_LP_PATH $LARCH_HOME_BIN
10 setenv LDT_LSL_EXE $LARCH_HOME_BIN/lsl2.4
11 setenv LDT_LCL_EXE $LARCH_HOME_BIN/lcl
12 setenv LDT_LCC_EXE $LARCH_HOME_BIN/lcc
13 setenv LDT_LP_EXE $LARCH_HOME_BIN/LP
14 setenv HELP_PATH $LARCH_HOME/LP/lib
15 setenv LDT_PATH /home/canidae/u20/morinm/cps898/ldt
16 setenv LDT_SHELL_PATH $LDT_PATH
17 setenv LDT_EXEC_PATH $LDT_PATH
18 mkdir ./temp
19 mkdir ./temp/lpfiles
20 $LDT_EXEC_PATH/lde &
```

Figure 1: The *run_ld* Startup File

- Line 3 indicates the specification path. This path must include the directories containing the lsl and lcl init-files. The Larch Handbook and any other directories containing specifications should be put in this *SPEC_PATH* so that they can be registered when LDE is started. If a specification file is in two different directories, the one in the directory listed first in the *SPEC_PATH* will be used and the other will be discarded.
- Line 4 has the directory containing the file *lpcomm.lp*. This file is needed for starting LP.
- Lines 5 through 9 are the directories containing the executables for the LSL checker, LCL checker, LCC checker, and LP, respectively.
- Lines 10 through 13 are the executables for the syntax checkers and LP. The LSL checker must be version 2.4 or a newer version having the *-lp* option.
- Line 14 is the directory containing the file *lp.help*.
- Lines 15 through 17 contain the directory paths containing the executable *lde*.
- Lines 18 and 19 create directories for the temporary files created and used by LDE. These lines should not be modified and the local directory from which LDE is run from should not have any file or directory called *temp* previous to running LDE.
- Line 20 contains the command to start LDE.

To start LDE, type *source run_ld* at the command line. The environment variables will then be set appropriately, links will be made to all specifications found in the *SPEC_PATH*, and the opening window will appear. This opening window is shown in Figure 2. By clicking on the *LSL*, *LCL*, or *LCC* buttons, the respective development window will appear.

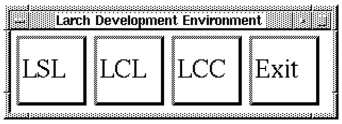

Figure 2: The LDE Opening Window

1.3.2 The Development Window

Figure 3 shows the *LSL Development Window*. The *LCL* and *LCC Development Windows* have basically the same functionality. The differences will be noted.

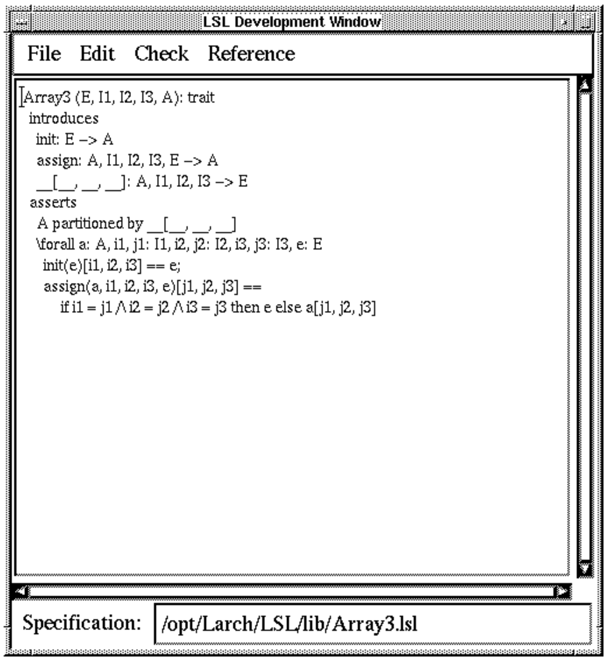

Figure 3: The Development Window

The *Development Window* consists of a menu bar, a scrolling editable specification text area, and a non-editable file path text field. The options on the menu bar are described below. The *Specification* text field indicates which specification file is currently loaded into the *Development Window*. The full directory path is given with the file name. The scrolled text area is where the specification is loaded and edited. This is a very simple editor with no special editing options. When text has been modified and you attempt to quit before saving, you will be notified and asked if you would like to save.

- File Menu: This menu provides the standard file options, as follows.

– New: This option will clear the text area and place *new* in the *Specification* field. If the current text in the text area has been modified, you will be asked if you would like to save first.

– Load: This option will display the *Known Specifications List*, shown in Figure 4. By clicking on the desired specification in the list and then clicking on the *Load* button, or by double-clicking on the desired specification, the selected specification will be loaded into the text area. As with *New*, if the current specification has been modified, you will first be asked if you want to save.

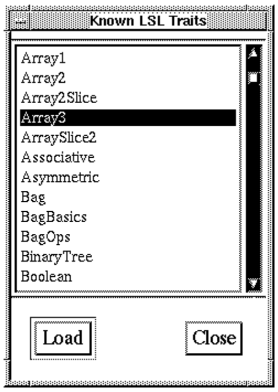

Figure 4: The Known Specifications List

– Save: If the current specification is not a new specification, it will automatically be saved under the same file name. If it is a new specification, the *Save As* option will take over.

– Save As: This option displays a window with text fields for the file name and directory paths. Enter the appropriate information and press *Apply* to save.

– Quit: This option closes the *Development Window*. If the current specification has been modified, you will be asked if you want to save first.

- Edit Menu: This menu provides template options for the specification language of the *Development Window*. Templates are discussed in Section 1.3.3. Only specifications originally created with the templates may be loaded into the templates. If the current specification was not originally created with the template, a window will appear informing you of this, and no template will be displayed.

  – Edit Specification: The current specification is loaded into the editable template and the text area of the *Development Window* becomes non-editable. If there is no current specification (the *Specification* field says *new*), a blank template will be displayed.

  – View Specification: The current specification is loaded into a non-editable template. You may only view the specification, without editing it. The text area of the *Development Window* remains editable.

- Check Menu: This menu offers syntax checking and, for LSL, debugging options.

  – Check Syntax: If the current specification has been modified, you must save first. Then the syntax checker, corresponding to the *Development Window's* respective language, is called. The output of the syntax checker is displayed in a text window.

- Axiom Check (LSL only): The LSL syntax checker is run with the *-lp* option. Then the lsl-created *_Axioms.lp* file is appended to the end of *lpcomm.lp*. You will be asked if you want to view the execution results. *Yes* corresponds to sending the command *execute <trait-name>_Axioms* to LP. *No* corresponds to sending the command *execute-silently <trait-name>_Axioms* to LP. The Larch Prover interface will then appear with the results of the execution displayed. The Larch Prover interface is discussed in Section 1.3.6.
- Full Check (LSL only): This option performs the same function as *Axiom Check*, except that the lsl-created *_Checks.lp* file is used instead of *_Axioms.lp*. The *Checks* file starts the proof of all theorems in the current trait.

- Reference Menu: This menu allows the selection of specifications for display in *Reference Windows*, described in Section 1.3.4.
  - Lookup Selection: When a specification is highlighted in the text area of the *Development Window*, this option will search for the specification with that name and display it in a reference window.
  - Browse XXX Specs: XXX represents be *LSL*, *LCL*, or LCC, depending on which *Development Window* it is selected from. The LCL and LCCL *Reference* menus provide options to browse the respective language specifications and to browse LSL specifications. When this option is selected, the list of specifications for the corresponding language is displayed. This list is the same list used for loading specifications, shown in Figure 4.

1.3.3 Templates

For each of the Larch languages, templates are provided for the development of new specifications. These templates are not as advanced as one would like, so all keywords and punctuation must be entered by the developer, and only specifications originally created by the templates can be reloaded into the templates. The templates break down the specifications into their major sections/clauses. As the templates for each language have the same functionality (the only differences are in the specification clauses), only the *LCC Template* will be discussed here. This template is shown in Figure 5.

When the template is displayed with the *Edit* option, as opposed to the *View* option, any of the scrolled text areas can be edited. Functions are listed in the *Functions* list. The LCL template also has a similar *Claims* list. To edit (or view) an existing function, double-click on that function in the list, and the function template will appear. This template is displayed in Figure 6. After the text has been edited in the function template, press *Apply* to make the change and close the template, press *Reset* to return the text to its original text, or *Close* to close the template without making the change.

To create a new function, click on *New Function* and a blank function template will appear. After entering the function text and pressing *Apply*, the new function will be added at the end of the *Functions* list.

When you are finished editing the specification, press *Apply* to generate the edited specification and load it into the text area of the *Development Window*. You can press *Reset* at any time to return the specification to the state it was in when originally loaded into the template. *Close* will close the window without saving changes.

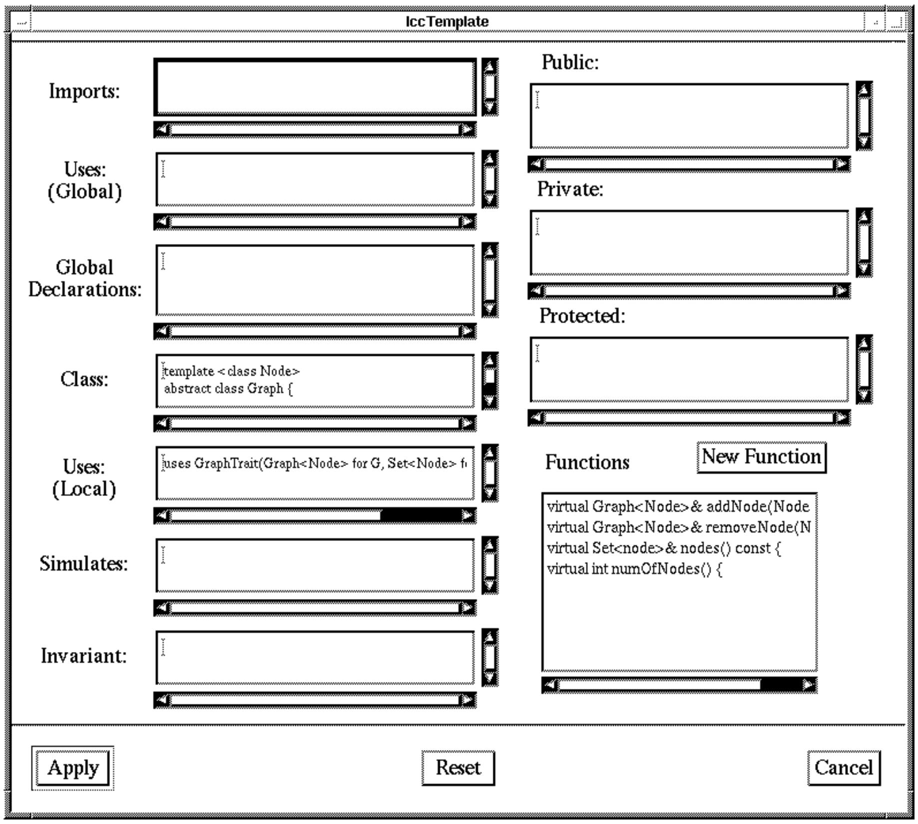

Figure 5: The LCC Template

1.3.4 The Reference Window

Reference Windows may be displayed for as many specifications as you wish. They provide the ability to view the properties of specifications that you may be referencing. Figure 7 shows the *Reference Window*. The LCL and LCC *Reference Windows* are they same, except they do not offer the *Sorts* and *Operators* options.

- Sorts (LSL only): This option uses the LSL syntax checkers *-sorts* option to get the sorts of the trait. The sorts are then displayed in a text window.
- Operators (LSL only): This option does the same as the *Sorts* option, except the *-symbols* option is used instead of *-sorts*.
- Look Up: This is the same feature as in the *Development Window*. If you highlight a used, assumed, included, or imported specification and click on *Look Up*, that specification will be displayed in a new *Reference Window*.
- Expand: This option performs a macro expansion of the used, assumed, included, and imported specifications in the *Reference Window*. The text of these referenced specifications is inserted into the text displayed in the *Reference Window*.
- Compress: If the specification in the *Reference Window* has been expanded, this option will return it to its non-expanded state.

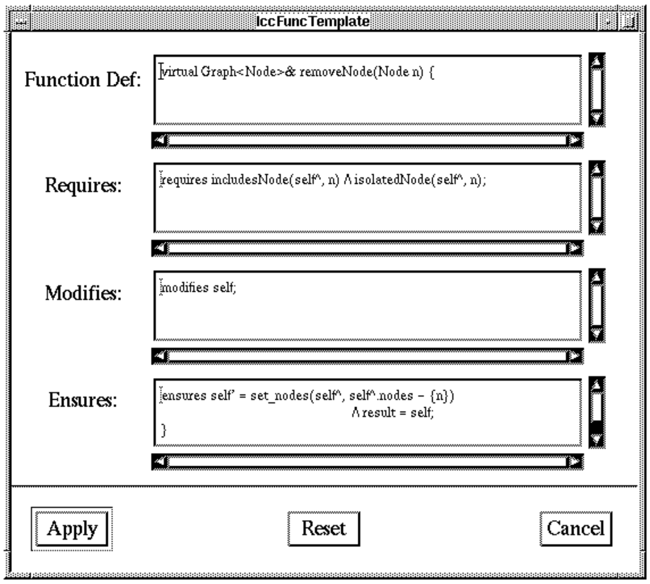

Figure 6: The LCC Function Template

- Dependencies: This option will display the *Dependency Hierarchy Window* for the specification. This *Dependency Window* is discussed in Section 1.3.5.
- Done: This option closes the *Reference Window*.

1.3.5 The Dependency Window

This window, shown in Figure 8 displays a graphical representation of the dependency hierarchy of the specification displayed in the associated *Reference Window*. The root of this hierarchy tree is the specification and only its descendants are displayed. This tree can be traversed by double-clicking on icons and can display *Reference Windows* for dependencies by single-clicking on the respective icons with the middle mouse button.

NOTE: For Specifications with very deep Dependency Hierarchies, the display of the Dependency Window and the display of the Dependency Tree will take a much longer time than for those with shorter hierarchies.

- Middle Mouse Button: When you single click on an icon with the middle mouse button, the specification corresponding to that icon will be displayed in a Reference Window.
- Path Display Area - Left Mouse Button: When you double click on an icon in the Path Display Area, that specification becomes the parent node of the Dependency Tree. It becomes the last icon in the Path Display Area and its immediate children are displayed in the Children Display Area.
- Children Display Area - Left Mouse Button: When you double click on an icon in the Children Display Area, that specification will become the parent node of the Dependency Tree. It is moved to the end of the path in the Path Display Area and its immediate children are displayed in the Children Display Area.

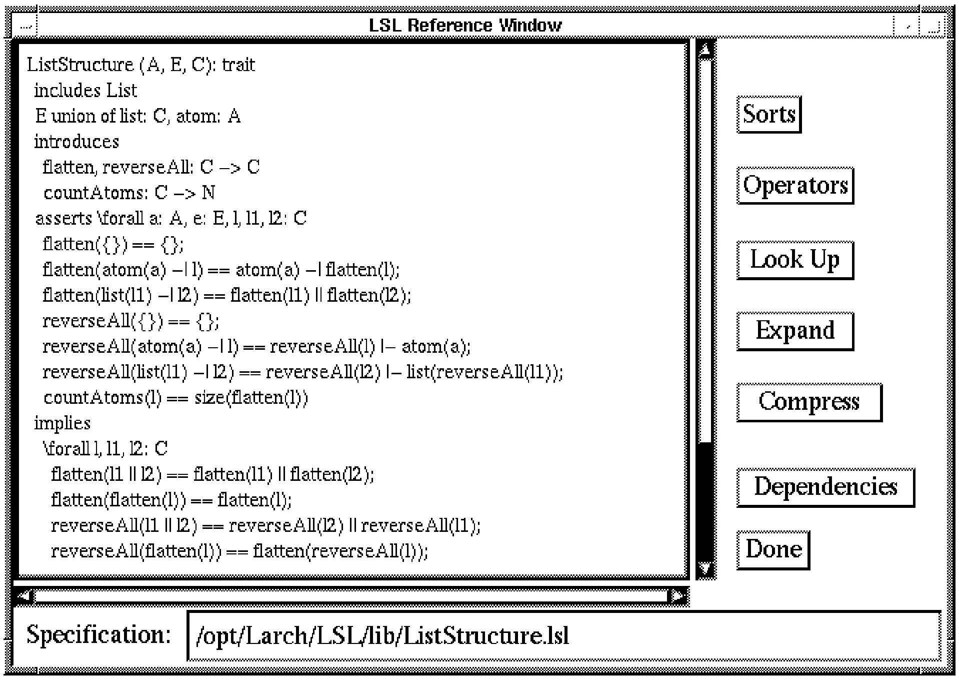

Figure 7: A Reference Window

- Dependency Tree - Left Mouse Button: When you double click on an icon whose children are stacked (when they overlap), the subtree of that icon node will be expanded. When it has been expanded, you can double-click on it again to return it to its stacked state.
- Done: This button closes the *Dependency Window*.
- Help: This button displays a window with the help text given here.

1.3.6 The Larch Prover Interface

The main LP Window, shown in Figure 9, contains a scrollable display area, menu options, and buttons. The menu options and buttons generate commands to be sent to LP for proof development. The resulting output is displayed in various windows. Only proof related output is displayed in the main proof window. All other output is routed to various windows for display. When running LP, a script file and a log file are created. The script file contains all commands entered during a session with LP, and the log file contains the commands with their resulting output. The following list briefly describes the services provided by the menu options and buttons. For more information on LP commands, see *A Guide to LP, The Larch Prover*, by Stephen J. Garland and John V. Guttag, Massachusetts Institute of Technology, March 1993.

- File Menu:
  - Execute: This option displays a window where a script file's name can be entered to be executed in LP. The user must also know the path name of the script file.
  - Stop Execution: This option puts a stop command in the script file. Then when the script file is executed, the execution will stop at this command.

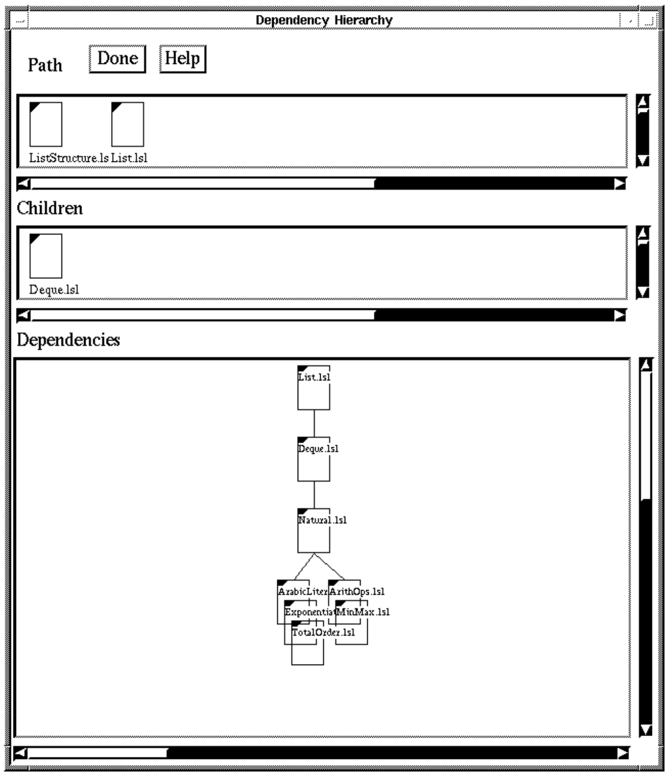

Figure 8: A Dependency Hierarchy Window

  - Write: This option displays a window requesting the file to be written to. All previous commands are then written to this file.
  - Quit: This option sends the *quit* command to LP and closes all LP related windows.

- Display Menu: This menu provides options to display symbols, facts, conjectures, proof status, ordering constraints, rewrite rules, settings, time statistics, history, and version. The LP output for these display commands is displayed in their respective windows.

- Options: This option displays the *LP Settings* window, shown in Figure 10, in which various settings, such as the log file, automatic ordering, and default proof method, may be set. Options are initially set by reading the file produced upon start-up of the LP interface.

- Help: This option displays the *LP Help Window*, shown in Figure 11. To select a help topic, double-click on that topic, single-click and then click on *Search*, or enter the topic into the *Help Topic* field. The help text is then displayed in the text area at the bottom of the window.

- Assert: This option displays a window in which an assertion can be constructed.

- Declare: This options displays a window to declare operators, sorts, or variables, depending on the option chosen from the *Declare* menu.

- Apply: This option displays a window to apply a rule to the term rewriting system of LP.

- Order: This option sends the command "*order*" to LP to order the rewriting system.

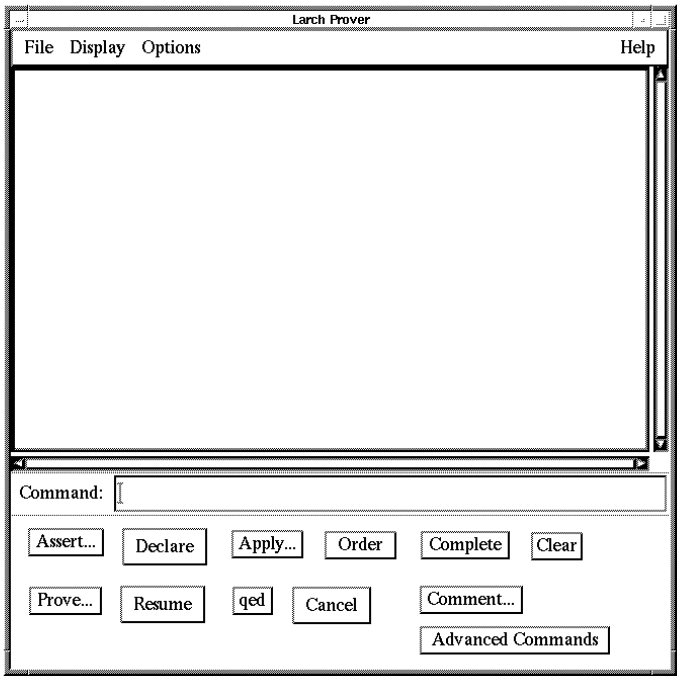

Figure 9: The Larch Prover Main Window

- Complete: This option invokes the completion procedure where LP attempts to produce critical pairs between each pair of rewrite rules in the system.

  NOTE: This may take a very long time if there are many rewrite rules in the system and may cause the LP interface to hang while waiting for LP to finish this procedure. It is recommended to not use this command from the LP interface if there are a large number of rewrite rules.

- Clear: This option clears the state of the proof system.

- Prove: This option displays a window in which a conjecture may be entered and its proof method can be set. When the selected proof method is *Cases* or *Induction*, another window is displayed for the cases to be entered or the induction variables to be set, respectively.

- Resume: This option causes LP to resume its attempt to resolve an outstanding conjecture.

- qed: This option displays the status of the conjectures currently in the LP system.

- Cancel: This cancels the proof of the current conjecture.

- Comment: This option displays a window in which a comment can be entered. This comment will be written to the log and script files.

- Advanced Commands: This option displays a scrolling list of all advanced LP commands. This list is shown in Figure 12. To select a command, double-click in the list or single-click and then click on the *Apply* button.

The *Assert Command Window* is shown in Figure 13. Each command that needs additional information from the user has an associated *Command Window*. The push button options at the

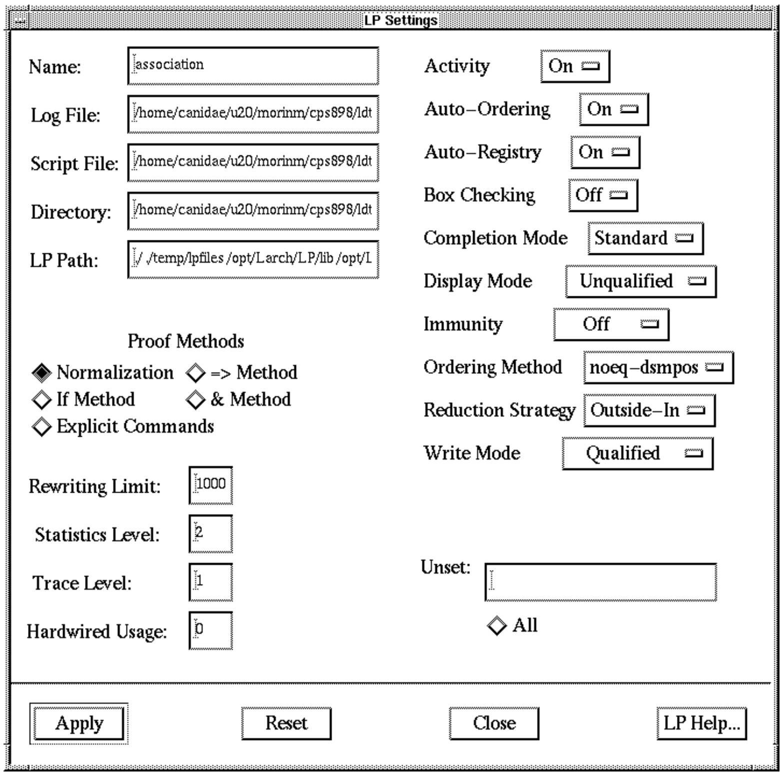

Figure 10: The Larch Prover Settings Window

bottom of all *Command Windows*, including the *Settings Window*, Figure 10, and the *Advanced Commands* list, Figure 12, provide the same options.

- Apply: This button formulates the command and sends it to LP. The text areas are then reset so that another command can be sent.
- Close: This button closes the window without sending the command.
- LP Help: This button displays the *LP Help Window*, shown in Figure 11, and automatically selects the command associated with the *Command Window* as the help topic.

2 Installation Procedure

To install LDE, create a directory called **lde** that the executable and other files needed for execution will be held in. Change directories to **lde**. Untar the tar bundle and uncompress the files in **lde** by typing:
tar xvf LDE.
Uncompress all the files in **lde** and its subdirectories:
uncompress src/*
uncompress ixb/*
uncompress *

In LDE, there is a **README** file containing a list of all the files you should have. Check over this list to ensure that you have all the necessary files. In order to compile lde, you must have

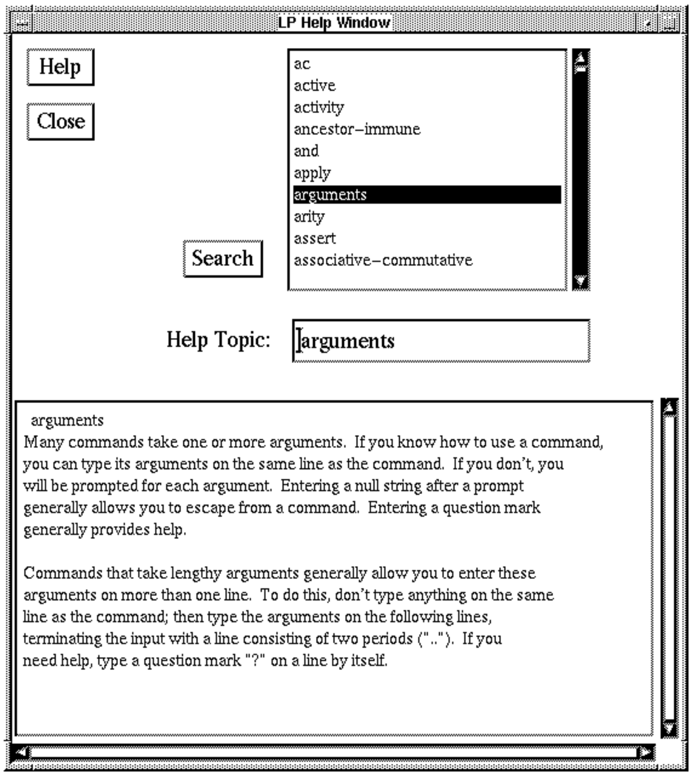

Figure 11: The Larch Prover Help Window

the UNIX make and ar utilities, the iXBUILD3.1, Motif, and XLIB libraries, and the SunOS CC compiler. The **INSTALL** file gives the following procedure for installing lde.

Change to the src directory. First modify any directory paths in Makefile that are different from your file system.
NOTE: The iXBUILD 3.1 libraries must be compiled using the OSF Motif libraries, not the SUNWmotif libraries. Using the SUNWmotif libraries causes a link error due to XmStrings.
Type **setenv XBPRJDIR 'pwd'** to set the project directory environment variable needed by the makefile. Type **make** to compile lde. It will take a while to compile (10-20 minutes). You will see many compiler warnings about unused parameters and case statements. Ignore these warnings.

After it has finished compiling, you can remove the LDE.o and libLDE.a. These are the object files and are no longer needed after the executable has been created. Move the executable, lde, from the src directory to the main lde directory.

Change directories to the main lde directory. To run lde, make sure the paths in the **run_ld** file are correct. See Section 1.3.1 for more details on this file. The Xdefaults file, LDE, should be in the same directory as the executable. Type **source run_ld** to start up lde.

3 References

For more information on formal specifications, the Larch languages and tools, and the Larch Development Environment, see the following [1, 2, 3, 4]:

Figure 12: The LP Advanced Command List

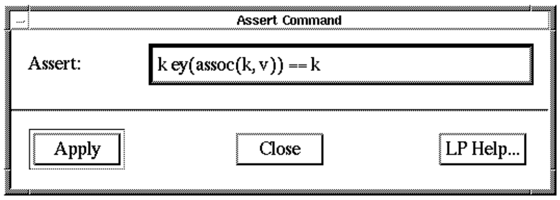

Figure 13: A Command Window

References

[1] Y. Cheon and G. T. Leavens, "A Quick Overview of Larch/C++," Technical Report 93-18, Iowa State University, Department of Computer Science, Iowa State University, Ames, Iowa, 50011, June 1993.

[2] D. Gries, *The Science of Programming.* Springer-Verlag, 1981.

[3] J. V. Guttag, J. J. Horning, K. J. S.J. Garland, A. Modet, and J. Wing, *Larch: Languages and tools for formal specification.* Texts and Monographs in Computer Science, Springer-Verlag, 1993.

[4] M. Morin and B. H. C. Cheng, "Graphical Development Environment for Larch Shared and Interface Languages," Technical Report CPS-94-18, Michigan State University, Department of Computer Science, A714 Wells Hall, East Lansing, MI 48824-1027, April 1994.