XASLANTEST USER'S MANUAL

Jeff Douglas
Reliable Software Group
Department of Computer Science, UCSB
(Revised 7/19/93)

I. Introduction

This document serves as an introduction to Xaslantest, the X-based graphical user interface version of the symbolic execution tool, Aslantest.

There are six sections in this document. Section II provides a brief overview of the ASLAN language. Section III focuses on Aslantest itself, describing operating requirements and usage of the tool. Section IV provides a detailed listing of available commands for Aslantest. Section V provides an overview of Xaslantest. Section VI lists and describes commands available via the graphical user interface.

II. Overview of the ASLAN language

The following is a brief overview of the ASLAN specification language. For a more detailed description, consult the ASLAN User's Manual[1].

The ASLAN specification language takes a state machine approach to specification. A system can be thought of as a collection of states, with two states differing by the values of the state variables. Changes in state variables may only occur through the application of state transitions. Given a state variable X and a transition T, ASLAN uses X' to denote the old value of X(the value of X before the application of T) and X to denote the resulting value.

For example, consider a system consisting of a single red-green traffic light. This system can be characterized using one state variable, Light, and two states, Light=red and Light=green. Thus a valid transition to change the value of Light might be:

```
Transition  Switch_To_Red
Entry
  Light = green
Exit
  Light = red
```

This transition asserts that in the resulting state, the new value of the state variable light is red. The entry assertion serves as a guard to this transition; this transition will only execute if, in the current state, the value of Light is green.

In guaranteeing "correctness" of a system specification, ASLAN takes an inductive approach, attempting to show that the specified system always satisfies certain critical requirements. There are two kinds of system requirements. Requirements that must hold in every state are known as state invariants. For example, a savings account specification would most likely have the following invariant:

(amount_of_money_withdrawn) <= (amount_deposited)

where amount_of_money_withdrawn and amount_deposited are state variables. The other class of requirements describe relationships between the values of state variables as they change across state transitions. These are known as constraints. For example, consider a soda machine that dispenses soda at 50 cents a can. Such a system would want to have the requirement that, if a soda was dispensed, then there was at least fifty cents deposited before hand. This is best expressed using a constraint:

(No_of_sodas = No_of_sodas' - 1) -> (amount_deposited' >= 50)

where No_of_sodas and amount_deposited are state variables, and '->' is logical implication.

In performing the induction, the base step is to show that the system begins only in states that satisfy the state invariant. In the specification, any assertions regarding possible starting states are expressed in the Initial Conditions. So, the base step is to show:

initial_conditions -> state_invariant.

The inductive step is to show that, for every transition T, if the state invariant is satisfied before application of T, the resulting state(after application of T) will satisfy the invariant and constraint:

invariant' & T -> invariant & constraint

where invariant' means applying the 'old value' operator to every variable in the invariant, "&" is logical conjunction, and T represents the entry assertion and the effects of applying transition T.

III. ASLANTEST

Operating Requirements

Aslantest has been developed to run on Sun workstations, under the UNIX operating system. Although not required, to use its X-based graphical user interface, Xwindows is required(Athena Widget Set; X11R4 libraries; R3 or R4 server). It also requires the use of the Lex lexical analyzer generator, the YACC compiler generator and the Austin Kyoto Common Lisp(AKCL) interpreter environment.

There are two parts to the tool, the ASLAN translator and the symbolic executor. The translator (developed using Lex and YACC) takes as input an ASLAN specification and produces a file containing an AKCL source program. The symbolic executor is a collection of AKCL utility routines that, when run by the AKCL interpreter along with the translator-produced program, becomes interactive symbolic executor of the original ASLAN specification.

Features of Aslantest

Aslantest allows the user to symbolically execute a system specification, expressed in the ASLAN specification language. The user can display current information about the

specification, execute transitions, and generate correctness conjectures, either from the command line or from a file.

Overview of Operation

Invoking Aslantest

Aslantest is invoked using a shell script in /fs/cs-grad0/weenie/thesis/aslantest. If the user sets up an alias in the following fashion(% is the command line prompt):

% alias aslantest '/fs/cs-grad0/weenie/thesis/aslantest/aslantest'

then, given an ASLAN specification, spec.asl, it could be executed with the command:

% aslantest spec.asl

The specification would be loaded and translated, the interpreter run, and the utility routines loaded. The '#' prompt signals that the interactive session has begun. At this point, the user can get a list of commands by typing **h**<enter>. **q**<enter> exits the program.

Display Commands

There are a number of commands that enable the user to view aspects of the specification. **LS** lists the specification; **LT** lists its transitions. **Vars** allows the user to view the current contents of any or all of the state variables. **Path** lists the current path of execution, listing all transitions executed and any assumptions that have been made. **Display** shows the current state predicate* , the initial conditions of the system, or the expected resultant state of the system.

Execution Commands

Once in interactive mode, the user may execute transitions providing parameters, save and restore states for later execution, add to the current state predicate, check correctness conjectures, and re-initialize the executor. **Exec** executes a transition. If parameters are required and not provided, an error message will appear. Using **Add**, the user can add a boolean expression to the current state predicate, which lists any restrictions on the values of state variables. **Init** enables the user to specify a start state or a result state. The user may enter the new state from a file or the keyboard, or use the system's default values. **Restart** effectively re-initializes the executor, putting the system in its default initial state (the original initial state of the system), resetting the execution path to nil and state variables to their initial values, and flushing all temporary constants and variables.

State commands

At any time during the execution of the specification, the user may **save** the current state or **restore** a previous state. **States** lists the states that have been saved. **Remove** removes a currently saved state from the list.

---

* the state predicate lists any current restrictions on the values of state variables.

Correctness Conjectures

Check allows the user to view correctness conjectures using current values. **Check** or **Check current** checks to see if the current state predicate implies the state invariant. **Check result** checks to see if the current state predicate implies the resultant state.

The Reduces_To Operator

Aslantest allows the user to provide additional information to the simplifier to assist it in proving the correctness conjectures described above. Using the 'Reduces_to' operator, the user can state that a particular predicate 'reduces to' another predicate. The simplifier can then use this information in its simplification. A reduces_to operation is add via the addpred command, which then stores as part of the state predicate. An example of its use:

```
# addpred
(instructions from symbolic executor)
a reduces_to b
.
#
```

'a reduces_to b' is now part of the state predicate. Whenever a conjecture is generated through the 'check' command, the simplifier will search for any instances of 'a' and replace them with 'b'. One note on this operator. If there are multiple instances currently in the system, there is no guarantee regarding the order of application of the reduces_to operations on a predicate. The reductions may be applied in any order.

Bound Variables

Bound variables in Aslantest are presented to the user as an underscore('_') followed by three consecutive digits. Should the user have occasion to enter a bound variable(through init or addpred), the bound variable should be entered as displayed on the screen. Example:

```
# addpred
(instructions from symbolic executor)
balance(_003) = 100
.
#
```

The value of balance with bound variable _003 as parameter is now 100.

Commands from a file

There are two commands that allow entry of commands from a file. **Seq** executes a sequence of transitions listed in a file. **File** executes a sequence of symbolic execution commands stored in a file. Also, the user has the option with the **Add** and **Init** commands to have predicates entered from a file.

Restrictions of Aslantest

It assumes that the specification is correct with respect to typing.
It assumes the specification is consistent, as defined in the ASLAN User's Manual[1].
It cannot execute mappings to lower level specifications.

IV. Listing of Aslantest Commands

The following is a list of Aslantest commands. Capital letters in commands denote an abbreviated form of the command. Material in brackets ('[]') is optional.

Display Commands

LS

Lists the specification. If the specification is longer than the length of the screen, only a screenful is shown at a time. The user hits the space bar to continue the listing.

LT [trans]

Lists the specification's transitions. They are listed in the order given in the specification, and given a number. When executing, transitions may be referred to by name or number. If a transition(trans) is given (either by name or by number), the code in that particular transition will be listed.

Path

Displays the current path of execution. Shows transitions executed and assumptions made on boolean expressions within the transitions, or via adding a predicate. A single entry from the output of the command is of the form:

(Added Conditions)
Start: trans(args)
(Executed Except #)
(Assumptions)
Finished(Aborted)

Where 'Added Conditions' lists any assumptions made between execution of two transitions, either through Addpred or via assumptions made while evaluating an Entry Condition; Start: trans(args) indicates that the execution of the body of 'trans' was begun with actual parameters 'args'; 'Executed Except #' tells the user that the '#' (number) Except in the listing for 'trans' was executed; 'Assumptions' lists any assumptions made while executing the body of the transition; 'Finished(Aborted)' indicates whether or not the body was executed and terminated successfully.

If an attempt was made to execute a transition and the current state did not satisfy the preconditions for that transition, then the following entry would be added for that transition:

(Added Conditions)
NOP: trans(args)
Preconditions for trans(args) not satisfied

indicating that the preconditions for the transition were not satisfied and the transition was not applied.

STATES

Displays the states saved during execution. Lists them by number, with the comment submitted by the user when the state was saved.

Display [current | Start | Result | Axioms]

Displays system state information. 'Display' or 'Display current' will show the current state predicate. 'Display start' will show the initial conditions of the system. 'Display result' will display the expected resultant state of the system. 'Display axioms' will list any axioms stated in the specification.

Vars [id]

Display variable(s) values. If given with no arguments, lists all state variables and their current values. If given 'id', displays that variable's value.

Execution Commands

Addpred

Allows the user to add a boolean expression to the current state predicate. The state predicate lists any restrictions placed on the values of state variables. The user may choose to type in the predicate or have it entered from a file. If entered from the keyboard, a '**.**' on a line by itself indicates to the executor that the user is finished entering the predicate. If entered from a file, no '**.**' is necessary.

Exec [trans]

Executes the named transition, trans. If no transition is given in the command, the user will be prompted for one. If parameters are required by the transition, and aren't provided, an error message will appear. The transition may be given by name or by its number, shown in the listing of the transitions (**LT**).

FIle [fileid]

Executes the sequence of commands given in 'fileid'. If no file is given, the user is prompted for a filename. Commands in the file should be in the same form as if given at the command line. A '**.**' on a line by itself indicates the end of the file.

SEQ [fileid]

Executes the sequence of transitions given in 'fileid'. Transitions may be given by name or number. If parameters are required and aren't provided, an error message will appear and the sequence will be aborted. If no fileid is given, the user is prompted to enter the sequence from the keyboard. Whether using a file or keyboard, a '**.**' on a line by itself indicates the end of the sequence. Upon receiving the sequence, the user will be asked: 'Show the sequence (y)es or (n)o?'. If the user responds 'y', the executor will list the sequence it has received. The user will then be asked: 'Pause between transitions (y)es or (n)o?'. If the user responds 'y', the executor will pause between executions of consecutive transitions. When the user is ready to continue, typing 'go' will begin execution of the next transition. If the user responds 'n', the executor will

execute all the transitions in the sequence without pausing. If any errors are encountered, execution of the sequence may be aborted or the user may be asked if they wish to continue.

Check [current | Result]

Produces correctness conjectures for the user to verify. 'Check' or 'Check current' produces a conjecture to show that the current state predicate implies the state invariant. 'Check result' produces a conjecture to show that the current state predicate implies the resultant state.

Init [start | Result]

Allows the user to initialize the executor to specific states. 'Init' or 'Init start' allows the user to specify a starting state. The user is allowed to use the default (the initial state given by the specification), enter it via the keyboard or from a file. 'Init Result' allows the user to type in their own expected resultant state, enter it from a file, or use the default('true').

RESTART

Reinitializes the executor. Reinstalls the original initial state, as given by the system. Resets state variables to their initial values, as given in the initial state. Sets the execution path to nil, flushes all temporary variables and constants, and resets execution flags.

Save [comment]

Saves the current state, associating with it a number (for reference) and 'comment' (if provided). If no comment is provided, the user will be prompted for one.

REStore [N]

Restore state number N. If no number is provided, the user is prompted for one.

ReMove [N]

Remove state number N. As with restore, if the user does not provide N, he is prompted for one.

Miscellaneous Commands

Load [fileid]

Loads in a new specification from the file 'fileid'. Reinitializes the executor to process the new specification. If no fileid is given, the user is prompted for one.

Help

Displays a list of available commands.

Quit

Exits the program and returns to UNIX. User is asked to confirm that they wish to exit the session. If he responds in the affirmative, the program exits. If the user responds no, he is asked if he wishes to load another specification. If yes, then a filename prompt is provided and the new specification is loaded. If no, control returns to the original specification.

Comments

The user may enter comments from the command line by beginning them with the character combination '/*'. Comments may be up to one line long, and are treated as no-ops by the executor. Ex:

```
# /* This is a comment
#
```

Execution flags

There are three execution flags that can be set by the user: simplify, verbose, and autosave. To determine the current value of a flag, simply type the name of the flag, and <enter>. To set a flag, type the name of the flag followed by 'on' or 'off'. Ex:

```
# simplify
simplifier is on
# simplify off
simplifier is off
#
```

Simplify

When set, all predicates and values of variables are simplified as much as possible by the simplifier. When off, no simplification is done; values are simply returned. By default, the simplify flag is set to on.

Verbose

When set, the user gets full message display. Turning off this flag results in suppression of certain help messages. By default, this flag is set to on.

Autosave

When set, the state of the system is automatically saved after each execution of a transition. States saved via autosave begin their comment with the keyword: '(AUTO)'. By default, this flag is set to off.

V. XASLANTEST

Xaslantest is the X-based, graphical user interface component of Aslantest. The following subsections describe the layout and operation of the GUI tool.

Overview of Operation

Xaslantest is invoked using the executable file Xaslantest located in /fs/cs-grad0/weenie/thesis/aslantest. The user will need to add this directory to their current path. If your operating environment is cshell, the following command:

```
% setenv PATH /fs/rsl/pkgs/aslantest:$PATH
```

will add the necessary directory to your path. Then, to run Xaslantest, simply type:

```
% xaslantest spec.asl
```

where spec.asl is an ASLAN specification to be symbolically executed. The interface will be set up, and Aslantest will be called to translate and load the specification, and initialize the executor. When the system is ready for interaction, the display will appear as in Figure 1.

Description of Display

The Xaslantest display breaks down into five components, the file label, the source window, message window, command panel, and dialog window. The overall display may be resized using the resize icon in the upper left corner. Also, by clicking and dragging the square box on the lower right of the source window, the length of the source window may be increased or decreased. This is a proportional resize; increasing (decreasing) the length of the source window will decrease (increase) the length of the dialog window. The other components of the display may not be resized.

The file label

Across the very top of the display is the file label. It consists of two components. Centered in the file label is the filename of the currently loaded specification. In the right hand corner is the current line number of the source window positional arrow (described below).

The Source Window

The source window is the upper most text window in the display. The source window is a read-only text window that contains a numbered listing of the source code of the specification. If the source code is longer than the length of the source window, as much of the source code will be displayed as possible, and the user may use the scroll bar provided along the left of the source window to bring other parts of the source code into view. If a line of visible source code extends beyond the width of the source window, another scroll bar will appear along the bottom of the source window. This scroll bar may be used to display the portion of the line that extends beyond the width of the window.

Additionally, in the left margin of the source window is a positional arrow that points to specific lines of the specification. Initially, upon loading the specification, the arrow will be positioned at line 1; subsequently, at each execution of a transition, the arrow will point at the

first line of the transition being executed, and the source code will be adjusted in the window such that the first line of the transition is in the top line of the source window.

The Message Display

Just below the source window, above the command panel, is a one line message display. It will display various messages and prompts in conjunction with the current mode or action being taken. After initialization, when the system is ready for interaction, the message display will show "Ready: Normal mode".

The Command Panel

The command panel is where all command buttons for Xaslantest are located. Using the command panel, the user has the ability to initiate any Aslantest option using the mouse buttons. The command panel consists of two rows which contain buttons. Buttons which appear in the top row of the command panel are all click and release activation buttons, whereas the buttons which appear in the bottom row activate pull down menus, showing other options. The bottom row buttons are activated by clicking and holding on the button, which will produce a pull down menu. The user may then move to the desired option and release the mouse button, activating the option.

In all, eight different buttons can appear in the top row of the command panel. Two, the help and quit buttons, remain in the upper left and right of the command panel at virtually all times. The others are 'response' buttons, and only appear when appropriate. These include the true, false, neither, yes, no and go buttons. The true, false and neither buttons will appear when the executor is asking the user for help with a predicate; figure 2 shows an example of this. The yes and no buttons appear when the executor is querying the user, and 'yes' or 'no' are appropriate responses. 'Go' is a command used when executing a sequence of transitions, so the go button will appear in those situations.

Bottom row buttons include options, display, execute, predicate, states and program. All, when pressed, activate pull down menus revealing various Aslantest options. Many of the options activated from the pulldown menu activate popup shells which get more information from the user. Appendix B describes these popup shells in more detail and gives examples. Specific discussion of the various bottom row buttons will ensue in section VI.

The Dialog Window

The dialog window simulates the normal teletype interface of Aslantest. The window echos responses and messages from the executor, and enables the user to enter commands using the keyboard. The user has full editing capabilities in this window; the following table shows a list of editing key combinations:

Key Combination	**Resulting Action**
<Ctrl>-w	Delete previous Word
<Ctrl>-x	Delete current line
<Ctrl>-u	Delete current line
<Ctrl>-h	Delete previous character
<Delete>	Delete previous character
<Backspace>	Delete previous character
<Return>	Enter a newline

The Xaslantest system is set such that, if the mouse pointer is within the bounds of the Xaslantest screen layout, all keyboard input is directed to the dialog window. A scrollbar on the left of the window enables the user to scroll up to see previous I/O that has gone off the screen. As with the source window, if any input or output continues past the width of the display window, a scroll bar will appear along the bottom of the window, that can be used to view the portion of the text outside the window.

VI. The Command Buttons

The following section is a detailed discussion of the options available via the command panel.

The Top Row

Help

Clicking and releasing on the help button will list in the dialog window the currently available Aslantest commands.

Quit

Executes the Aslantest quit command. The user will be presented with a popup shell, asking him is he wishes to return to UNIX, load a new specification, or cancel the request.

True, False, Neither, Yes, No, Go

Clicking and releasing on these buttons will send the labelled response to the symbolic executor.

The Bottom Row

The Options Button

The options button is used for setting various Aslantest and Xaslantest execution options. Clicking and holding down on the button will reveal a five label pull down menu. The labels are the three execution options for Aslantest: simplify, verbose, and autosave; and two Xaslantest options, autoscroll and script. If the label has a check mark next to it, then that execution flag is set to 'on'. The three Aslantest options are as described previously; script, when on, maintains a hard copy script entitled 'xaslantest.log' in the execution directory. By default, the script flag is set to off. If the autoscroll flag is set, Xaslantest will do whatever scrolling is necessary to keep the positional arrow in view in the source window. If it is set to off, the position arrow will be updated, but if it is taken off the screen, it will not be scrolled into view. The default setting is on. To toggle the settings, simply move the mouse pointer over the appropriate option, and release the button.

The Display Button

The display button provides the user the ability to point and click to display various components of the system. Using this pulldown menu the user may view: the execution path, current state predicate, current start state, current result state, one or all variables, one or all transitions, or the axiom list.

The Execute Button

The Execute pull down menu reveals the various Aslantest execution options. The user may execute a transition, execute a sequence of transitions, initiate the file command, or enter a comment using the execute pull down menu. Each of these options produce

appropriate pull down menus to prompt the user for necessary information to execute the requested command.

The Predicate Button

Using this button, the user may add a predicate, and initialize either the start or result state. They may also initiate the 'check' commands available in Aslantest, 'check current' or 'check result'.

The States Button

The States button enables the user to save, restore or remove states. If there are currently no states, the pull down menu will consist solely of the button 'save state'. If there currently states saved, following the save state button will be a listing of currently saved states. The user can then select one. Upon selection, the state will be displayed and the user can choose to restore, remove, or cancel.

The Program Button

Via the program button, the user can restart the system or load a new specification.

VII. Bibliography

1. B. Auernheimer and R. Kemmerer, *Aslan User's Manual*, Department of Computer Science, University of California at Santa Barbara, TRCS 84-10, April 1992.

Appendix A -- Explanation of Miscellaneous Messages

A (not complete) listing of possible terminal messages:

1. "Could not open file "fileid"

Indicates that the executor could not access the filename 'fileid' provided by the user.

2. "Unknown option - "op"

Indicates that the user entered an option that was unrecognizeable to the command interpreter.

3. "Option ignored - "op"

Indicates the user entered an inappropriate option for that point in the execution. Note: during certain times in the execution, some sets of commands can become restricted, meaning they are ignored until the completion of the current operation. For example, 'File' and 'Seq' are ignored when in the process of executing or evaluating a transition.

4. "Syntax error in wf_formula"

An error was found while parsing a predicate entered by the user, either from a file or from the keyboard. The predicate is discarded and control is returned to the command interpreter.

5. "Incorrect format for transition call"

Most likely, an incorrect number of parentheses were entered when entering the arguments of a transition.

6. "Variables as actual parameters not supported"

At this time, variables are not supported as arguments to transition calls. The execution is aborted, and control is returned to the command interpreter.

7. "Eval_COND - cannot evaluate expression"

Indicates there were references the new value of variables in the decision predicate of an If-Then-Else statement. Execution of the transition is aborted.

8. "Transition reduces to True!"

The transition currently being executed reduces to true, indicating that there is no changing of any state variables within the transition.

9. "Transition reduces to False!"

The transition currently being executed reduces to false, which would indicate an error state. The execution is aborted, and control is returned to the command interpreter. All state variables and the state predicate are left as if the execution didn't occur, and the execution path marks the False transition as 'Aborted'.

10. "Near Fatal Error: Caught in a Signal or error: Returning to top level Command Interpreter"

This is the most serious system error. This message indicates that something occurred to cause the system to fall through to the Kyoto Common Lisp interpreter. Such an error is the result of the system being unable to process some aspect of the given command or user input. The Aslantest error handling system is equipped to handle one occurrence of such an error. A second occurrence causes the system to fall through to the KCL interpreter.

Appendix B -- Explanation of Selected Xaslantest PopupShells

Xaslantest uses popup shells to attain necessary command specific information from the user. These popup shells help direct the user, providing with information on what information to provide and how to provide it. There are three major types of popup shell used by Xaslantest. They are pushbutton popups, selection box popups, and dialog popups.

Pushbutton Popups

Figure 3 shows the pushbutton popup for the Quit command. Pushbutton popups consist of an informational prompt and a collection of buttons for the user to choose from. The user selects an option by moving the mouse pointer to the button and clicking within the button boundary. In the displayed example, the popupshell has three buttons, UNIX(quit and return to UNIX), Load(load new specification), and Cancel(cancel the command). In the case of the quit popup shell, hitting the return key will effectively cancel the command, causing the popup shell to disappear.

Selection Box Popups

Selection box popups provide the user with a list of options of which the user can choose one using their mouse pointer. Figure 4 shows the filemenu selection box. The filemenu selection box is invoked when the user is accessing the file system to load in a file. The user is presented with a list of files in a directory. He may choose a displayed file by clicking on it, or, by clicking on a directory, view another set of files. Selection box popups are also used in selecting variables for display and transitions for display or execution.

Dialog Shells

Dialog shells enable the user to enter information through the keyboard. Dialog shells consist of a text prompt, a single line text window, and a collection of buttons. The dialog shell for entering transition parameters is provided as an example in figure 5. The user may enter text into the text window. The window expands to hold whatever text is entered on the line. When finished, the user may continue processing of the command by hitting the enter key, or clicking on the appropriate command button at the bottom of the display.