

**Visualization and Error Localization
in the UCLID Verification System**

By: Timothy Washington

Summer 2006

UCLID is a system that is used for the modeling and verification of hardware and software systems that are infinite-state. Its application areas include microprocessor verification, software security, and verifying distributed algorithms and protocols. UCLID is an acronym for Uninterpreted functions, Counter arithmetic, and Lambda expressions for Infinite Domains, which are the main features that make up this modeling language. Finding errors previously required looking through the code of a UCLID program along with the text UCLID output which is printed out every time UCLID is run on a given program. This method made it difficult for the user to localize errors in the program. Thus a graphical user interface was built for the UCLID program to make error localization easier for the user.

This graphical user interface was built using the Java programming language. Various classes were constructed to perform and display various processes and features that would be added to the UCLID verification system. The main program is an extension of the Java's JFrame class. These features of the graphical user interface would be integrated with UCLID through another Java class used for running commands that would run UCLID on given command lines.

One important feature which was added to the UCLID verification system was the ability to store command line options. Originally, when running UCLID on a file, the user would have to type that file's path, followed by the name, and all of the verification options which would determine how UCLID would be run on that file. However, after implementing this feature in the graphical user interface, the user has a button that will automatically run UCLID on a file once selected. To determine how UCLID will be run, the user can now select the appropriate check boxes and combo boxes which contain the various verification options. Thus running UCLID was made more convenient.

In order to implement this feature, the corresponding check box and combo box objects were created and added to a panel which appears in the top right corner of the program. Under these options a button object was added to run UCLID. When selected a command string is set equal to the given file's path and name. For all of the check boxes and combo boxes that are selected, the appropriate string is concatenated to the end of this command string. Next this string is entered as an argument in a method from the run-command class which was mentioned earlier.

Next, the text editing feature was added to the UCLID verification system. This would give the user the opportunity to modify the UCLID code. The main feature added for UCLID file modification was a text area that would automatically display a UCLID file's code once opened. This text area was displayed in a tabs which would allow the user to modify various files at one time. Along with this text area was a popup menu that would give the user options such as cutting, pasting, deleting, copying, and selecting UCLID text.

To implement this text editing feature, a separate class was created for the UCLID workspaces. This class would contain all of the text editing components. In order to read these files into the text area of the workspace, an object from the JFileChooser class was instantiated to allow users to select the appropriate file to be displayed. To show multiple files for the user to modify, a tabbed pane was implemented, with tabs that held the components of each UCLID workspace object. In order to implement the popup menu, a mouse listener was added to the text area in the workspace class, thus giving the user the ability for a right click editing menu.

Another feature that was added to the graphical user interface was a display of the UCLID output. The UCLID

output informs the user if a counterexample is found for the program property being verified; if it is then it also displays the values of the UCLID state variables in each state in the counterexample. This text area was added to the bottom right window of the UCLID program and displays the appropriate output each time the user runs UCLID on a file. Also there is an option to view this output in a larger external window which would allow the user to view more variables and states at a time.

In order to display this output in the text area, a string array was set equal to the string array of the UCLID output that is generated with each UCLID run. Next, a loop structure was added to the program. Starting at zero and ending at the length of the array the loop would add the string at the given position in the array to the next line of the text area. The option for the external performed this sequence on a separate text area in a separate window.

Another feature that was added was an error trace table which automatically displays the values of variables through all of the given states. Normally when the user runs UCLID on a file, the UCLID output is displayed and the user must look through all of the given states of this output for any errors in variables or functions. With an error trace table, there are as many columns as there are

states in the counterexample, and as many rows as the number of variables in each state. As with the UCLID output text area, the graphical user interface gives the user the option of an external view of the error trace table. However this view has more options. The user can choose which variables from the error trace table they want to view along with which states will be displayed in the table. Thus, this makes it easier for the user to find errors.

Implementing the error trace table was one of the more complicated parts of building the graphical user interface. For the table to be displayed, a method was made within the error trace class that parsed through the UCLID output and determined the number of states. Another method parsed through the output and determined all of the variables it contained. Next, a table object was created, taking in the number of variables as its row argument and the number of states as its column argument. Next a method was created that took in a state integer as an argument. When called, this method returns all of the variable values for a given state argument. Another method was created to set the table values in a given column number to those of the state variables. After this a looping structure would set these

columns until the last state column of the error trace was set.

For the external view of the error trace table these same methods were used to generate the initial table. For the user to select which variables in the table that they want to use, a list was implemented that displayed each UCLID module's variables. There was also another list implemented that displayed the variables that the user selected to view in the table. Next, the button used to exclusively view these variables in the table was added. When selected it performed the same method as previously mentioned but only on the listed variables. For the user to choose which states would be viewed, check box objects were implemented. When the table button is selected, an array of Boolean values corresponding to the states in chosen by the user. If the corresponding value to a state is true, this it is displayed, if not, then it is hidden.

Another feature that was added was a list of variable dependencies which appears at the bottom right portion of each UCLID file. This displays each of the variables that are defined in the UCLID program, and in a list under each variable, the variables that the variable depends on to exist. For example, if in a given UCLID program, $A = B \& C$, then variable A would appear with B and C in a sub-list

under A. This would also make UCLID use more efficient as the user would just need to look at this structure to see the variable dependencies rather than look through an entire program for a variable and its dependencies.

For the variable dependencies to be displayed, another method was created that would parse through the UCLID file code each time a file is opened and a key listener was implemented so that it would continue to parse each time a key is pressed to keep the variable dependencies updated. When parsing through code, all variables listed under the VAR, STOREVAR, EXTVAR, and INPUT sections of the program were listed for each model. Next code was implemented to create a tree node consisting of the file name. All of the models would be listed as children of this file node. The variables for the given module node would be made into child nodes of the module nodes. Next another parse of the UCLID text was implemented that would get the variable dependencies under the ASSIGN and DEFINE section of a UCLID file for each variable. Once obtained, these variables were added as child nodes of the each variable node.

Variable highlighting was also implemented into the UCLID verification system. Earlier, the user would have to look through the UCLID code and find an instance of a variable. With this feature the user can select from each

module's list of variables and add them to a list of variables to be highlighted. When the highlight button in this window is selected all instances of these variables are highlighted. This visualization feature helps the user localize his/her attention to assignments to specific variables from the error trace.

To implement this feature a highlighting class was created with a primary method that took in the text area as an argument. Next, a method for parsing through the text area to get an integer representation of all the locations of the given words was created. Then a highlighter object is declared which highlights the appropriate words at the location which is determined by the integers.

Another feature that was added to the UCLID verification system was a variable query. This query is viewed in a separate window and gives the user the option of viewing the states in the error trace table that satisfy a predicate. This predicate can be a relational expression that expresses if a variable is equal, not equal, greater than, or less than another variable or constant. When a state of the error trace table holds the appropriate values for these variables the appropriate column will be highlighted.

To implement this query, a separate frame was created. This frame would hold two text fields that represented either a variable and a constant or two constants. Between these text fields a combo box was added that held the following operators: $>$, $<$, $=>$, $<=$, $=$, and \neq . Code was implemented so that when the user selects the button to run the query, only the variables in the text fields are shown in the table. Then code was implemented to get the value of each cell in a given state and see if it is equal to the other value. If so that column was set as selected and if not then that column was left alone.

Another feature that was added to UCLID verification system was a cone of influence computation. A cone of influence is a set of variables that appear in the transitive fan-in of the variables appearing in a UCLID first-order formula. In this case, this would consist of all of the variables in the decide statement that appears at the end of every UCLID program, and which specifies the property to be verified. Each of these variable's dependencies would be listed in a sub-list; under the variables in the sub-list, there would be more sub-lists of variable dependencies and so on. This continues until all of the variables have been listed.

Implementing the cone of influence was similar to implementing the variable dependencies. Both of these graphical user interface elements were trees with the same root for a given file. Also both had key listeners and methods to be updated when a key is pressed or a file is opened. However a method was created once again parse through the code, but this time only get the variables in the decide statement. Once obtained these variables were passed into another method created. In this method these variables are converted into child nodes of the root node and a recursive call is made to get the dependencies of these variables and add them as children accordingly. A Boolean flag was also implemented to prevent redundancy within this tree which could cause an error in the program.

As a result of implementing these various features into the UCLID system, error localization has been made much more efficient for the user. It is easier to run UCLID with specific verification options and work with many files simultaneously. Locating variables in a program and finding their dependencies is now done by selecting the appropriate components of the graphical user interface. Variable queries now give the user information on how variables relate to one another throughout states as well as the error trace table. Finally, the cone of influence

specifies to the user all the variables that the property being verified syntactically depends on.

User Menu

Text Editing on Multiple UCLID Files

Verification Options

Variable Highlighting

Variable Query

Error Trace Table

Cone of Influence

UCLID Output Display

UCLID

File Edit

mainmy-bug.tud | state-bug.tud | pipeline-bug.tud | queue-bug.tud

stage is same as the source of the record instruction *)
 stat = eWST & (src1@SRC = eDST);
 (* Forward the result of the writeback stage if it is valid, and
 its destination is the same as the 2nd source of the exec stage *)
 fwd = case
 when eDST = eSRC2) : wVAL;
 default : eARG1; (* changed to eARG1 from eARG2 by manually
 essac;
 * State assignments *)
 ASSGN

init@SRC = pc0; (* initially arbitrary *)
 next@SRC = case
 when stat = 0 then src1@SRC; (* forward, if it there is a stat *)
 essac;
 init@SRC1 = rd; (* initially arbitrary *)
 next@SRC1 = Lambda(x).
 when e & (a = wDST) : wVAL; (* stores wVAL in wDST if valid *)
 default : src1@SRC;
 essac;
 init@DST = 000;
 next@DST1 = op@SRC;
 init@SRC21 = 00;

	STATE 0	STATE 1	STATE 2	STATE 3	STATE 4
implWRT	false	false	false	false	false
implWRT1	false	false	false	false	true
implWRT2	2	2	2	2	2
implWRT3	64	70	70	70	70
implWRT4	64	70	70	70	70
implWRT5	64	70	70	70	70
implWRT6	64	70	70	70	70
implWRT7	64	70	70	70	70
implWRT8	64	70	70	70	70
implWRT9	64	70	70	70	70
implWRT10	64	70	70	70	70
implWRT11	64	70	70	70	70
implWRT12	64	70	70	70	70
implWRT13	64	70	70	70	70
implWRT14	64	70	70	70	70
implWRT15	64	70	70	70	70
implWRT16	64	70	70	70	70
implWRT17	64	70	70	70	70
implWRT18	64	70	70	70	70
implWRT19	64	70	70	70	70
implWRT20	64	70	70	70	70
implWRT21	64	70	70	70	70
implWRT22	64	70	70	70	70
implWRT23	64	70	70	70	70
implWRT24	64	70	70	70	70
implWRT25	64	70	70	70	70
implWRT26	64	70	70	70	70
implWRT27	64	70	70	70	70
implWRT28	64	70	70	70	70
implWRT29	64	70	70	70	70
implWRT30	64	70	70	70	70
implWRT31	64	70	70	70	70
implWRT32	64	70	70	70	70
implWRT33	64	70	70	70	70
implWRT34	64	70	70	70	70
implWRT35	64	70	70	70	70
implWRT36	64	70	70	70	70
implWRT37	64	70	70	70	70
implWRT38	64	70	70	70	70
implWRT39	64	70	70	70	70
implWRT40	64	70	70	70	70
implWRT41	64	70	70	70	70
implWRT42	64	70	70	70	70
implWRT43	64	70	70	70	70
implWRT44	64	70	70	70	70
implWRT45	64	70	70	70	70
implWRT46	64	70	70	70	70
implWRT47	64	70	70	70	70
implWRT48	64	70	70	70	70
implWRT49	64	70	70	70	70
implWRT50	64	70	70	70	70
implWRT51	64	70	70	70	70
implWRT52	64	70	70	70	70
implWRT53	64	70	70	70	70
implWRT54	64	70	70	70	70
implWRT55	64	70	70	70	70
implWRT56	64	70	70	70	70
implWRT57	64	70	70	70	70
implWRT58	64	70	70	70	70
implWRT59	64	70	70	70	70
implWRT60	64	70	70	70	70
implWRT61	64	70	70	70	70
implWRT62	64	70	70	70	70
implWRT63	64	70	70	70	70
implWRT64	64	70	70	70	70
implWRT65	64	70	70	70	70
implWRT66	64	70	70	70	70
implWRT67	64	70	70	70	70
implWRT68	64	70	70	70	70
implWRT69	64	70	70	70	70
implWRT70	64	70	70	70	70
implWRT71	64	70	70	70	70
implWRT72	64	70	70	70	70
implWRT73	64	70	70	70	70
implWRT74	64	70	70	70	70
implWRT75	64	70	70	70	70
implWRT76	64	70	70	70	70
implWRT77	64	70	70	70	70
implWRT78	64	70	70	70	70
implWRT79	64	70	70	70	70
implWRT80	64	70	70	70	70
implWRT81	64	70	70	70	70
implWRT82	64	70	70	70	70
implWRT83	64	70	70	70	70
implWRT84	64	70	70	70	70
implWRT85	64	70	70	70	70
implWRT86	64	70	70	70	70
implWRT87	64	70	70	70	70
implWRT88	64	70	70	70	70
implWRT89	64	70	70	70	70
implWRT90	64	70	70	70	70
implWRT91	64	70	70	70	70
implWRT92	64	70	70	70	70
implWRT93	64	70	70	70	70
implWRT94	64	70	70	70	70
implWRT95	64	70	70	70	70
implWRT96	64	70	70	70	70
implWRT97	64	70	70	70	70
implWRT98	64	70	70	70	70
implWRT99	64	70	70	70	70
implWRT100	64	70	70	70	70

MODEL PipelineDrapath
 src1@SRC
 src2@SRC
 src3@SRC
 src4@SRC
 src5@SRC
 src6@SRC
 src7@SRC
 src8@SRC
 src9@SRC
 src10@SRC
 src11@SRC
 src12@SRC
 src13@SRC
 src14@SRC
 src15@SRC
 src16@SRC
 src17@SRC
 src18@SRC
 src19@SRC
 src20@SRC
 src21@SRC
 src22@SRC
 src23@SRC
 src24@SRC
 src25@SRC
 src26@SRC
 src27@SRC
 src28@SRC
 src29@SRC
 src30@SRC
 src31@SRC
 src32@SRC
 src33@SRC
 src34@SRC
 src35@SRC
 src36@SRC
 src37@SRC
 src38@SRC
 src39@SRC
 src40@SRC
 src41@SRC
 src42@SRC
 src43@SRC
 src44@SRC
 src45@SRC
 src46@SRC
 src47@SRC
 src48@SRC
 src49@SRC
 src50@SRC
 src51@SRC
 src52@SRC
 src53@SRC
 src54@SRC
 src55@SRC
 src56@SRC
 src57@SRC
 src58@SRC
 src59@SRC
 src60@SRC
 src61@SRC
 src62@SRC
 src63@SRC
 src64@SRC
 src65@SRC
 src66@SRC
 src67@SRC
 src68@SRC
 src69@SRC
 src70@SRC
 src71@SRC
 src72@SRC
 src73@SRC
 src74@SRC
 src75@SRC
 src76@SRC
 src77@SRC
 src78@SRC
 src79@SRC
 src80@SRC
 src81@SRC
 src82@SRC
 src83@SRC
 src84@SRC
 src85@SRC
 src86@SRC
 src87@SRC
 src88@SRC
 src89@SRC
 src90@SRC
 src91@SRC
 src92@SRC
 src93@SRC
 src94@SRC
 src95@SRC
 src96@SRC
 src97@SRC
 src98@SRC
 src99@SRC
 src100@SRC

State 1: |
 ##### Boolean State #####
 implWRT1 = false
 implWRT2 = false
 ##### Term State #####
 implWRT1 = 2
 implWRT2 = 70
 implWRT3 = 70
 implWRT4 = 158
 implWRT5 = 158
 implWRT6 = 65
 implWRT7 = 151
 implWRT8 = 130
 implWRT9 = 64
 ##### Enumerated State #####

References

Randal E. Bryant, Shuvendu K. Lahiri, and Sanjit A. Seshia.
"UCLID: A Verification Tool for Infinite-State Systems."
Carnegie Mellon University. December 30, 2004.
<http://www.cs.cmu.edu/~uclid/>

Randal E. Bryant, Shuvendu K. Lahiri, and Sanjit A. Seshia.
A User's Guide to UCLID version 1.0. Carenegie Mellon
University 2003.

Randal E. Bryant, Shuvendu K. Lahiri, and Sanjit A. Seshia.
Modeling and Verifying Systems using a Logic of Counter
Arithmetic with Lambda Expressions and Uninterpreted
Functions. Copenhagen, Denmark, July 2002.

Shuvendu K. Lahiri and Sanjit A. Seshia. The UCLID Decision
Procedure. July 2004.