272 – Homework Assignment 4
Fall 2008
Due: Dec 3rd, 5:00 PM

Do not discuss the problems with anyone other than the instructor or the TA.

Submission Instructions: Send your answers to puneet@cs.ucsb.edu with CS272HW4 in the subject.

1. Generate invariants using Daikon. For this problem you are required to implement a class **BinarySearchable**. This class should have two methods with the following signatures

- **public int[] sort(int[] arr)** : This method should sort the array arr using Bubble sort and return it.
- **public int binarySearch(int[] arr, int val)** : This method should search for the value val in arr using binary search and return its index. The array should only have positive elements. In case the value cant be found it should return -1.

Once you have implemented the above methods write a driver main method in the class that calls these methods with a variety of array lengths and array values and search values (generate the values and the lengths of the arrays and the search values randomly). Now run this class using Daikon and report the invariants you obtain. Do this for two different test sets (i.e., for two different main methods). The first one should be a small test set (say arrays with less than 5 elements, and with calls to the methods only 4-5 times). The second one should be a large enough test set so that Daikon can discover interesting invariants (you should experiment with different test sets until you observe that increasing the test set does not result in new invariants being reported). Report both the small and the large test set (i.e., both main methods), your implementation of the BinarySearchable class, and the invariants you obtain with both test sets using Daikon.

Note that Daikon's invariant generation relies heavily on the amount of data that it will have to analyze, so repeated method calls will give Daikon additional data on the behavior of your method. Also for binarySearch do remember that its important that you have method calls that are going to produce negative results for the binarySearch method.

Running Daikon: Daikon distribution is available at /cs/class/cs272/cs272/daikon. You will need to add daikon.jar available in this directory to your classpath.

Running daikon involves the following steps:

- **Step 1:** Compile your class with the -g command line flag. E.g. %**javac -g myexamples/BinarySearchable.java**

- **Step 2:** Run your program using daikon to generate a trace file. E.g. % **java daikon.Chicory myexamples.BinarySearchable**. This step generates a tracefile BinarySearchable.dtrace.gz
- **Step 3:** Analyze the trace file generated in Step 2 for invariants. E.g % **java daikon.Daikon BinarySearchable.dtrace.gz** . This step will report the invariants that daikon was able to find in your methods.

Note: The last two steps can be combined using the –daikon flag . E.g. % **java daikon.Chicory –daikon myexamples.BinarySearchable**. Also, running Step 3 or running daikon using the above method, generates a file called BinarySearchable.inv.gz in your current directory. This can be used to print out the invariants again without doing the analysis on the trace file again. In order to do this, use the following command : %**java daikon.PrintInvariants BinarySearchable.inv.gz**.

Daikon manual is available here:
http://groups.csail.mit.edu/pag/daikon/download/doc/daikon.html

2. Consider a channel class with four methods: *open*, *send*, *receive*, *close*, and *status*. It is possible to send to or receive from a channel and check its status only after the channel is opened and before it is closed. A channel can be closed only after it is opened. A channel can be reopened only after it is closed. The *status* method is a query method.

(a) Draw the interface machine for this class using the interface machine model from the paper by Whaley et al.

(b) Assume that following constraint is added to the above interface specification: At least two messages have to be sent to a channel before it is closed. Can this be expressed using the state machine model used by Whaley et al.? Draw a state machine for this modified interface specification using the interface machine model from the paper by Alur et al.

3. Assume that a program crashes with the input `(abcde)f`. Also assume that the program crashes with any input string that contains two matching parentheses and it does not crash with any other input. Show an execution of the delta debugging algorithm in this case. Assume that the minimal change is adding one character to the input string, and start with an initial partition where each set adds half of the failure inducing input string given above.