272 – Homework Assignment 3

Fall 2008

Due: November 24th, 5:00 PM

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

1. In this problem you will generate test cases for a K-ary heap using Korat. A K-ary heap is defined as follows: A K-ary heap is a heap data structure created using a K-ary tree where each node in the tree has at most K children. A K-ary heap can be seen as a K-ary tree with two additional constraints:

- **The shape property** : All levels of the tree, except possibly the last one (deepest) are fully filled, and, if the last level of the tree is not filled, the nodes of that level are filled from left to right.
- **The heap property**: The data item in each node is greater than or equal to the data items in each of its children according to some comparison predicate which is fixed for the entire data structure.

(a) Provide an array implementation of the K-ary heap where K is passed as an argument to the constructor. Write the repOK method for the K-ary heap based on the above description. Use the finitization method to set the range of values in the array and the value of K. An array implementation for a binary heap is provided in the Korat website which should help you in implementing and testing the K-ary heap.

(b) Generate test cases for a K-ary heap using Korat. Generate test cases for K-ary heaps where K is 3 and 4. Choose 5 different values for the number of nodes and generate all the canonical heaps for each value using Korat. Report the number of heaps generated by Korat and the time and the memory usage.

(c) Give two implementations of the repOK method: 1) One that is efficient for test generation with Korat. 2) One that is inefficient for test generation with Korat. Note that, these two repOK methods should be equivalent in terms of the value they return. Experimentally demonstrate the difference between these two repOK methods by showing the time and the memory usage by Korat for each one. Explain why is one of them more efficient than the other one.

Korat JAR and its dependencies are available at **/cs/class/cs272/cs272/korat/** . You will need to include all of them in your CLASSPATH environment variable to run Korat.

Note: The files for this assignments should be compiled against the latest Korat distribution available at the above mentioned location. Please refer to the tutorial available at http://korat.sourceforge.net/tutorial.html and the manual available at http://korat.sourceforge.net/manual.html

2. Consider the following transition system $T = (S, R, I, L)$ with the set of states $S = \{0, 1, 2, 3\}$, the initial set of states $I = \{0\}$, the transition relation $R = \{(0, 1), (1, 2), (2, 0), (2, 3), (3, 3)\}$, the set of atomic propositions $AP = \{p, q\}$ and the labeling function $L : S \to AP$ where $L(0) = \{q\}$, $L(1) = \{p, q\}$, $L(2) = \{q\}$, and $L(3) = \emptyset$.

For each of the following temporal formulas list the states which satisfy them:

$$EXp,\ AXp,\ EXq,\ AXq,\ EGp,\ EGq,\ AGp,\ AGq,\ EFp,\ EFq, AFp,\ AFq.$$

Which of these properties are satisfied by the transition system T?

3. Given the following pieces of code, draw the control flow graphs and give minimal test sets for the following criteria: statement coverage, branch coverage and path coverage. If there are statements, branches or paths that are not feasible to execute, indicate them. The inputs can take any integer value. Label the basic blocks in your control flow graphs. For each test case show the set of statements, branches or paths it covers using the basic block labels (for example, for the test set for path coverage, show the paths covered for each test case). For path coverage loops should be executed zero and one time.

(a)

```
int test1(int x, int y) {
  if (x > y+1)
     z = 1;
  else
     z = 0;
  if (x > y+3)
     w = 3;
  else
     w = 0;
 return z+w;
}
```

(b)

```
int test2(int x, int y) {
  int i, sum, out;
  sum = 0;
  for (i = 0; i < x; i++) {
    sum = sum + i;
  }
  out = sum;
  if (sum < y)
    out = -1;
  return out;
}
```

4. Consider the following testing tools: TestEra, Korat, and Cute. Characterize these tools using the following terms (one tool can be characterized with more than one term): Black box

testing, white box testing, functional testing, structural testing, specification based testing, random testing, bounded exhaustive testing, symbolic execution, unit testing, integration testing, system testing, branch coverage, statement coverage.

Submission Instructions:

To submit electronic files:

1. Create a directory whose name is your CS account. For example, user John Doe –whose account is jdoe– would do:

   % mkdir jdoe

2. Put in the directory the files for this assignment.

3. Execute the turnin program. For example, user jdoe would execute:

   % turnin hw3@cs272 jdoe

   You can execute turnin up to 10 times per project. Earlier versions will be discarded. The timestamp of turnin has to be before the due date.

To submit answers on paper: Please submit your answers in a stapled bundle with your name on the front sheet.