CSE647: Testing and Verification (Fall 2000)
Scott Stoller
Exam

For every problem, you must justify your answer.

Problem 1 (10 points)

(a) Describe the differences between a recursive (also called reentrant) lock and a non-recursive (also called non-reentrant) lock.
(b) Are Java's built-in locks recursive or non-recursive?
Answer: (a) If a thread holds a recursive lock and tries to acquire it again, it succeeds and increments a counter associated with the lock. If a thread holds a non-recursive lock and tries to acquire it again, something undesirable happens (e.g., an error, or the thread blocks). (b) Recursive.

Problem 2 (10 points)

Consider a concurrent system $\mathcal{S}$ with two threads θ_0 and θ_1 and one communication object q, which is a bounded FIFO queue with capacity 2. The queue supports exactly two operations, namely, insert and remove. Consider a state s_0 in which q contains exactly one element. $enabled(s_0, \theta_i) = \{t_i\}$ for $i \in \{0, 1\}$. t_0 performs insert, and t_1 performs remove. Static analysis shows that the remove operation does not appear in any transition of thread θ_0. In (a) and (b) below, decide whether it is possible, *based only on the given information about* $\mathcal{S}$, to conclude that the specified set is persistent in the specified state. Your answer must be yes or no, followed by a justification. Use the best conditional dependency relation for queues that you can think of.
(a) Is $\{t_0\}$ persistent in s_0?
(b) Is $\{t_1\}$ persistent in s_0?
Answer: (a) No. If θ_1 does a remove and then two inserts, the second insert disables t_0. (b) Yes.

Problem 3 (10 points)

Consider state-less search (without partial-order methods) of a system $\mathcal{S}$ with a finite acyclic state space. Suppose a search with depth bound d terminates without exploring a state containing a specified control point ℓ. Then we conclude that ℓ is not reachable within d steps (in other words, there does not exist an execution σ of $\mathcal{S}$ such that length$(\sigma) \leq d$ and σ contains a state containing ℓ).
(a) Consider the same situation as above, except that the state-less search is optimized with sleep sets. Is the above conclusion still correct?
(b) Same as (a), except for state-less search optimized with sleep sets and persistent sets.
Answer: (a) Yes. Sleep sets may reduce the number of explored transitions, but not the number of explored states. Intuitively, the answer is yes because a transition is "put to sleep" only after it has been executed in some branch. (b) No. Persistent sets might delay a transition that could be executed within d steps so that it is never executed during the depth-bounded search.

Problem 4 (10 points)

What is substructure sharing? (Briefly describe the general idea; you do not need to give a detailed description of how to implement it.) Explain why substructure sharing is useful in Java PathFinder but not in standard JVMs.

Answer: The idea is that subgraphs of the heap that are the same in multiple explored states are stored only once. This is not useful in standard JVMs, because they store only the current state.

Problem 5 (10 points)

This problem is about predicate abstraction, as described in Model Checking Java (by Brat *et al.*), and as discussed in lecture. Consider the method

```
m(int x, int y, boolean b) {
  if (b) { x = x+1; }
  boolean p = b;
  if (p) { y = y+1; }
  return (x==y);
}
```

Suppose we abstract this method using the directives

```
Abstract.addBoolean("xeqy", x==y);
Abstract.remove(x);
Abstract.remove(y);
```

The result is a method like

```
m1(boolean xeqy, boolean b) {
  if (b) { if (xeqy) xeqy = false;
           else xeqy = Verify.randomBool(); }
  boolean p = b;
  if (p) { if (xeqy) xeqy = false;
           else xeqy = Verify.randomBool(); }
  return xeqy;
}
```

If `x==y`, then `m(x,y,b)` returns `true`. However, `m1(true,true)` non-deterministically returns `true` or `false`. Thus, `m1` is a sound but approximate (not exact) abstraction of `m`. Is it possible to obtain a more accurate abstraction of `m` by introducing one or more additional predicates? If so, give the appropriate abstraction directive(s) and the code for the resulting abstracted method. If not, explain why it is impossible.

Answer: A more accurate abstraction of `m` is obtained by adding the directive `Abstract.addBoolean("xeqyp1", x==y+1)`. The result is a method like

```
m2(boolean xeqy, boolean xeqyp1, boolean b) {
  if (b) { if (xeqy) { xeqy = false; xeqyp1 = true; }
           else if (xeqyp1) { xeqy = false; xeqyp1 = false; }
           else xeqy = Verify.randomBool(); xeqp1 = Verify.randomBool(); }
  boolean p = b;
  if (p) { if (xeqy) { xeqy = false; xeqyp1 = false; }
           else if (xeqyp1) { xeqy = true; xeqyp1 = false; }
           else xeqy = Verify.randomBool(); xeqp1 = Verify.randomBool(); }
  return xeqy;
}
```