

CMPS 290G – Topics in Software Engineering
Winter 2004 – Software Validation and Defect Detection
Homework 2

Due: 12 Feb 2004

1. What is a *race condition*?
2. Write a (short) Java program with a race condition. Briefly describe how the race condition can occur.
3. Write a (short) Java program that has no race conditions, but whose behavior depends on how different threads are interleaved, and where the program behaves incorrectly under some thread interleaving. Briefly describe how this incorrect behavior can occur.
4. Fix the previous program so that it always behaves correctly.
5. What does it mean for a method to be *atomic*? State which of the methods in the programs from questions 2, 3, and 4 are atomic.
6. Using the definitions of lecture 5a, suppose

S_1 has atomicity B
 S_2 has atomicity R
 S_3 has atomicity B
 S_4 has atomicity A
 S_5 has atomicity B
 S_6 has atomicity L
 S_7 has atomicity B
 S_8 has atomicity A

what is the atomicity of the following statements?

$S_2; S_4; S_6$

$S_1; S_2; S_3; S_4; S_5; S_6; S_7$

$S_1; S_2; S_3; S_4; S_5; S_6; S_7; S_8$

7. Write a test harness that reveals a bug in `java.util.StringBuffer` and causes `java.util.StringBuffer` to crash. (Hint: This can be done in 10 lines of code.)

8. Consider the Java method:

```
void m(int a,int b) {
    if (a<0) return 0;
    int i=0, r=0;
    while (i!=a) {
        loop:
            r = r + b;
            i++;
        }
    return r;
}
```

Suppose the following are candidate loop invariants that Daikon conjectures might hold at the label `loop`.

$i > 0$	$i \geq 0$	$b < 0$
$a > 0$	$a \geq 0$	$i < a$
$a < i$	$r = i*b$	$r = i+b$

For each candidate invariant, either

- convince yourself that the invariant is correct (that is, it always hold at the label `loop`, no matter what arguments the method `m` is applied to), or
- state what method arguments would cause Daikon to reject that candidate invariant.

Inspired by these invariants, suggest a useful postcondition for this method.

- List an advantage and disadvantage of automated testing.
- What is regression testing?
- What is code coverage? Suggest how this metric might be used effectively, and one way in which this metric might be mis-used.
- Describe how bug trends could help decide when to release a software product.
- List at least 3 kinds of bugs caught by fuzz testing.
- Suppose we have a (delta-debugging) configuration containing two possible changes: c_1 and c_2 . Suppose $test(\emptyset)$ passes, $test(\{c_1\})$ fails, $test(\{c_2\})$ fails, and $test(\{c_1, c_2\})$ fails. Is this configuration (1) consistent, (2) unambiguous, (3) monotonic? Give examples of configurations that fail each of these three properties.
- Suppose we have a configuration containing four possible changes: c_1, c_2, c_3 , and c_4 , where
 - $test(C)$ is unresolved if $c_1 \in C$ and $c_2 \notin C$,
 - otherwise $test(C)$ fails if $c_1 \in C$ and $c_3 \in C$, and
 - otherwise $test(C)$ passes.

Illustrate the process of applying the algorithm *ddmin* to this problem, using the format of table 2 of the paper "Simplifying Failure-Inducing Input". Is the result a minimal failure-inducing change set?

- List five examples of *checkers* that are complete (at least with high probability) and asymptotically faster than the computation they check.