Exam Software Testing & Verification 2011/2012

25 may 2012, 9:00–12:00,BBL-065

Lecturer: Wishnu Prasetya

1. [1.5pt] Consider this program:

```
String foo(c:List<String>) {
  if (c==null)
      c = new List<String>() ;
  String found   = null ;
  int k = 0 ;
  for (s:String in c) {
    if (s.contains("foo")) {
       found = s ;
       k++ ;
    }
  }
  return found + k;
}
```

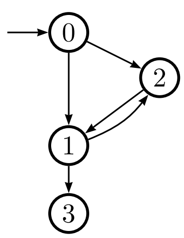
   (a) Give a control flow graph that correponds to the program. It should be made clear in your drawing which group of instructions and expressions is represented by each node.

   (b) Give the definition of Edge-Pair Coverage, and give the smallest possible set of feasible test-paths that would give you maximum Edge-Pair Coverage.

2. [1.5pt] Prime paths.

   (a) Give the definitions of *simple path* and *prime path*.

   (b) Consider the control flow graph below; 1 is the starting node, and 3 is the exit node.



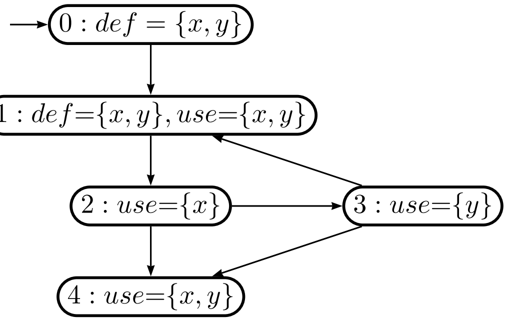
   List all prime paths in the graph.

   (c) A coverage criterion C_1 *subsumes* another criterion C_2 if all test-set that satisfies C_1 also satisfies C_2. Give a test-set on the above graph that shows that the Edge Coverage criterion *does not* subsume the Prime-path Coverage criterion.

   (d) Give a definition of *tour with detour*. That is, when does a test-path t can be said to tour a target path u with *detour*?

   (e) Give a simple program that shows a situation where it is necessary to weaken the concept of 'tour' to 'tour with detour'; specify the problematic prime-path in that example program.

3. [1.5pt] Consider the following control flow graph. The nodes are decorated with def and use information; the edges have no def nor use.



State 0 is the initial-state, and state 4 is the end-state. If no def or use is mentioned on a node, it means the corresponding decoration is empty on that node.

(a) Give all members of $du(1, x)$ and $du(1, 1, x)$.

(b) Give a test-set for the above graph, that shows that full All-Defs Coverage does *not* subsume All-Uses Coverage. Hint: such a test-set gives full coverage of the first, but not the second.

4. [1pt] Consider the classes `A` and `B` below; notice that the method `f` of `A` calls the method `judge` of `B`.

```
1  class A {
2    b : B ;
3    ...
4    String f(int x) {
5        int y = x*x ;
6        if (x<0)
7            y = 0 ;
8        r = b.judge(y) ;
9        return r
10   }
11 }
12
13 class B {
14   ...
15   String judge(int y) {
16      if (reader == null)
17          return null ;
18      int x = reader.readFromFile("data") ;
19      if (y>x)
20          return "ok"
21      return "not ok"
22   }
23 }
```

We want to test the integration between `f` and `judge` (in other words, we want to do integration testing).

(a) List all the *coupling variables* and their corresponding *coupling du-paths* between `f` and `judge`. You can use line numbers to identify the nodes in your paths.

(b) Specify the test-requirements (the TRs) for All-Coupling-Def Coverage and for All-Coupling-Uses Coverage for the above example. You can specify the TRs in terms of paths.

5. [1pt] Consider this function:

$$foo(p : Person, i : Insurance, a : Address)$$

The tester decided to divide the domain of each parameter above into the following blocks:

- $Person : Child, Adult, Senior$
- $Insurance : None, Standard, Premium$
- $Address : Fixed, Moving$

We will only consider abstract test-cases, which are expressed in terms of the blocks above (you don't have to specify the concrete values for p, i, and a).

(a) Give a test-set that would give full Pair Wise Coverage.

(b) Suppose we add a constraint that if the address is $Moving$, then the type of insurance must be $Premium$, is it then still possible to get full Pair Wise Coverage? Explain your answer.

(c) The tester chooses this test-case as a *base test*:

$$foo(Child, Premium, Fixed)$$

Note that this implies that those blocks have been chosen as base blocks. Give a minimalistic test-set that gives maximal Base Choice Coverage. Take the constraint given in (b) into account.

6. [1.5pt] Consider this predicate (it has three clauses):

$$isMasterStudent(x) \;\vee\; (isStudent(x) \wedge (x.name = "foo"))$$

For this predicate, a test-case is just values assigned to the clauses. There is a constraint on the clauses, namely that $isMasterStudent(x) \Rightarrow isStudent(x)$; where $\Rightarrow$ means 'implies'. A test-case is only *feasible* if it satisfies this constraint. A test-set is feasible if it only contains feasible test-cases.

(a) Give a feasible test-set (for the above predicate) that shows that Predicate Coverage *does not* subsume Clause Coverage.

(b) Give a feasible test-set that gives full Correlated Active Clause Coverage, and *indicates for each test-case, which clause(s) it activates.*

Hint: a truth-table can help you.

(c) What is the difference between Correlated Active Clause Coverage and Restricted Active Clause Coverage? Does the test-set you gave in (b) give full Restricted Active Clause Coverage?

(d) Give a feasible test-set that gives maximum Restricted Inactive Clause Coverage, and *indicates for each test-case, which clause(s) it makes inactivate.*

7. [1pt] Mutation.

(a) Suppose we have a program $P(s : String)$ whose input s is described by the following BNF:

(1)	S	$::=$	ϵ
(2)	S	$::=$	$S\ Brace$
(3)	S	$::=$	$S\ Curly$
(4)	$Brace$	$::=$	"()"
(5)	$Brace$	$::=$	"(" $Brace$ ")"
(6)	$Curly$	$::=$	"{}"

where S is the starting symbol. Quoted texts are terminals.

Give as precise as possible a definition of what it means for a test-set on P to have a full Poduction Coverage.

Give such a test-set; indicate for each test-case which production rules it covers.

(b) Suppose we also want to do negative testing on $P(s)$. That is, we want to test how it deals with invalid s. We introduce a single mutation operator o that can be applied to any of the above production rules to mutate the rule. Explain how to do the following:

i. Negative test on P that gives full Mutation Operator Coverage.

ii. Negative test on P that gives full Mutation Production Coverage.

8. [1pt] Consider the following classes. Notice that `A2` is a subclass of `A1`, which is a subclass of `A`, and notice the overriding of the method `g`.

```
class A {
   int x ;
   A a ;
   public f() { g() ; x = a.x }
   public g() { a = new A() ; a.x = ... }
   public h() { x = 0 ; a.x = 0 }
}

class A1 extends A {
  override public g() { super.g() ; a.x = ... }
}

class A2 extends A1 {
  override public g() { h() }
}
```

(a) Draw the Yo-Yo graph of the above classes.

(b) What is the purpose of a Yo-Yo graph?

(c) Give an example of either data flow anomaly or a State Definition Anomaly.