SE465/ECE453/ECE653/CS447/CS647 Midterm Sample Solutions

Question 1 Solutions

(a) *Fault*: A static defect in software (incorrect lines of code). It is often referred to as bug. For example, the expected behavior is to calculate the sum of `a` and `b`. The line `return a-b;` in the following code snippet is a fault.

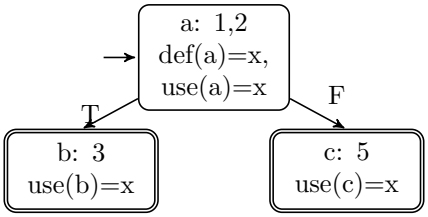
```
1 // should return the sum of a and b
2 int calSum(int a, int b) {
3   return a-b;
4 }
```

Failure: Incorrect behaviour that can be observed. For example, in the code snippet above, when $a = 2$ and $b = 1$, the expected output is 3, while the actual output is 1, which is a failure.

Note that a test case must include expected output. If not, deduct 0.5 mark. However, do not deduct marks for the same mistake twice.

(b)

```
1 int foo(int x) {
2   if (x>0)
3     return 100-x;
4   else
5     return 100+x;
6 }
```



With respect to x: TR_{ADC} = { [a, b]}, and TR_{AUC} = { [a, b], [a, c] }.

The test set that contains a test case $x = 1$ with expected output 99 satisfies ADC, but not AUC because the path $[a, c]$ is not covered.

It must go by the definition of subsumption. For example, it must use a test case to dhow that ADC doesn't subsume AUC. If no test case is given, deduct 1 mark. In addition, a test case must contain input and expected output.

(c) Valgrind is a dynamic tool, while all others are static tools.

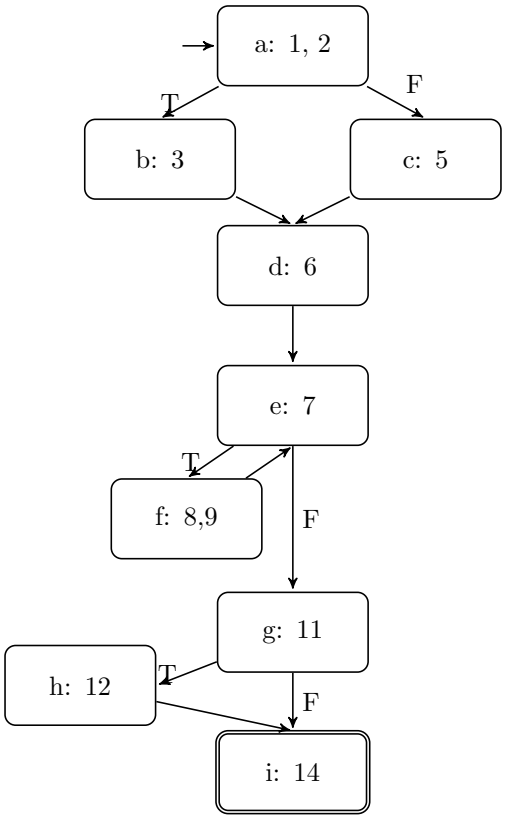
We should accept other reasonable answers.

(d) It can lead to deadlock because the lock is not released before executing `return -EBUSY;`. This leaves the lock unreleased after the method returns, making it impossible for other threads to acquire the lock.

A fix is to add `unlock_kernel();` before `return -EBUSY;`.

Question 2 Solutions

(a)



(b) TR_{EC} = { [a,b],[a,c],[b,d],[c,d],[d,e],[e,f],[e,g],[f,e],[g,h],[g,i],[h,i] } There are 11 edges.

Test set $T = \{t1, t2\}$ is a test set that satisfies EC.

Test case $t1$'s mapping test path is $path(t1)$= [a,c,d,e,f,e,g,i]. $t1$'s input is (2 , 1, _) and its expected output is 2, where _ denotes any valid integer.

Test case $t2$'s mapping test path is $path(t2)$= [a,b,d,e,f,e,g,h,i]. $t2$'s input is (2 , -1, _) and its expected output is 0.5.

(c) No, T doesn't trigger the division by zero bug.

Test case $t3$'s input is (0 , -1, _) and its expected output is infinity.

$t3$ triggers a division by zero bug.

Depending on the answer in b), the answer of c) can be yes. If $t3$ is part of test set given b), then the answer for c) is yes.

Question 3 Solutions

Similar to the assignment quesiton. No sample solutions provided.

Marking Scheme: 1 mark for mutation operator, 3 marks for each non-equivalent mutation, 2 marks for test case for each mutation, and 2 marks for mutant and original output (per mutation).

Question 4 Solutions

(a)

```
1  private void visitRegularCall(Node node,int type,Node child, boolean firstArgDone) {
2      while(firstArgDone && (child !=null)) {
3          child = child.getNext();
4      }
5      if(firstArgDone && (child !=null)) {
6          child = child.getNext();
7      }
8      boolean isSpecialCall = node.getProp(Node.SPECIALCALL_PROP) != null;
9      String simpleCallName = null;
10     simpleCallName = getSimpleCallName(node);
11     if (simpleCallName != null && !isSpecialCall) {
12         child = child.getNext().getNext();
13   }
14 }
```

The line numbers in the original code snippet is: 1876, 1878 (optional), 1897, 1899, 1900 (optional), 1906, 1907, 1911 (optional), 1918, 1920, 1922, 1923, 1927, 1931 (optional), and 1953 (optional).

Please pay attention to the lines that are simply {, or }. No need to be strict on them. These lines are optional.

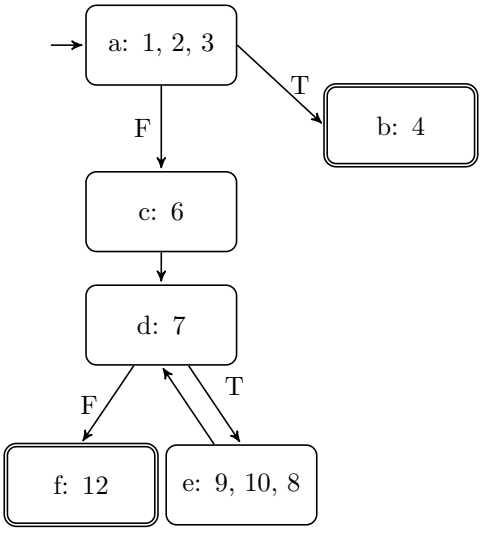
(b) The ansewrs are not unique.

Line 3 in the slice is a definition node for `child` (`child = child.getNext()`). Line 2 is a use node of `child` (`while([...] (child !=null)`). The path from the node of line3 to the node of line 2 is a du-path with respect to `child` because of the while loop. This path is a du-path because it is def-clear with respect to `child` and it is a simple path.

Another correct answer: The def of `child` in the first `child=child.getNext()` (Line 3) statement reaches the use of `child` of the second `if (child !=NULL)` (Line 5) statement. This path is a du-path because it is def-clear with respect to `child` and it is a simple path.

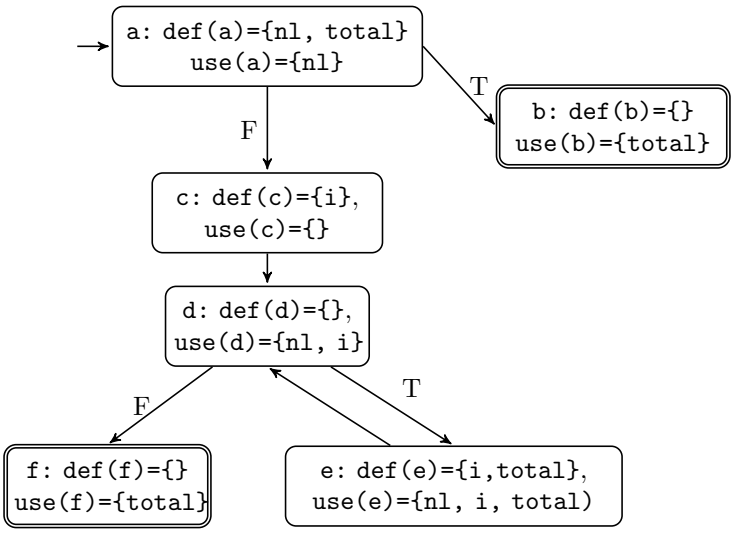
Question 5 Solutions

(a)



Marking Scheme: 1 point for each correct node including outgoing edges.

(b)



Note that `n` is omitted per question (b)'s requirement. Also, a detailed & ordered def-sets & use-sets for node `e` is as follows:

```
// Line 9:  use(e) : {i, nl}
            def(e) : {n}
// Line 10: use(e) : {n, total}
            def(e) : {total}
// Line 8:  use(e) : {i}
            def(e) : {i}
```

It is not required to have this latter presentation of the def and use sets for node `e`, however the order shown here is the correct set for determining AUC requirements in part d).

Marking Scheme: 1 point for each correct node, assuming graph in part a) is correct.

(c) TR_{PPC}= { `[a,b]`, `[a,c,d,e]`, `[a,c,d,f]`, `[d,e,d]`, `[e,d,e]`, `[e,d,f]` }

Marking Scheme: 1 point for each correct test requirement, + 1 additional point if all 6 TRs are correct as a set.

(d)

- `du(nl, a) = { [a,c,d], [a,c,d,e] }` Note that it is unnecessary to list [a] since the def and use are in the same node, but do not deduct points if students list [a]. So [a] is optional.
- `du(i, c) = { [c,d], [c,d,e] }` Note that TR `[c,d,e]` ensures the initial def of `i = 0` reaches the first execution of line 9 and line 8.
- `du(i, e) = { [e,d], [e,d,e] }` Note that TR `[e,d,e]` ensures the value from `i++` reaches the next occurrence of line 9 and `i++` via looping.

Therefore, $TR_{AUC} = du(nl, a) \cup du(i, c) \cup du(i, e)$ = { [a,c,d], [a,c,d,e], [c,d], [c,d,e], [e,d], [e,d,e] }

Marking Scheme: 1 point for each correct test requirement, + 1 additional point if all 6 TRs are correctly enumerated.

(e) Notation: `Last Def -> First Use`

Listing any of the following 3 excluding the one that was provided in the original question (marked below) receive full marks.

Return from CountLeaves

`(CountLeaves, total, 6) -> (CountLeavesInNodeList, total, 10)` for return values (This one was given in the question.)

`(CountLeaves, total, 3) -> (CountLeavesInNodeList, total, 10)` for return values

CountLeavesInNodeList calling CountLeaves

`(CountLeavesInNodeList, n, 9) -> (CountLeaves, m, 2)` for parameter passing

Return from CountLeavesInNodeList

`(CountLeavesInNodeList, total, 2) -> (CountLeaves, total, 8)` for return values (Note that the actual use is at line 8 not line 6).

`(CountLeavesInNodeList, total, 10) -> (CountLeaves, total, 8)` for return values (Note that the actual use is at line 8 not line 6).

CountLeaves calling CountLeavesInNodeList

`(CountLeaves, children, 5) -> (CountLeavesInNodeList, nl, 3)` for parameter passing

Marking Scheme: 0.5 points for each component of an inter-procedural DU pair. Each pair must be a last-def & first-use pair related to a variable passed as a parameter, or a variable used in a return; has 6 components, and is worth 3 points. If more than 3 pairs were listed, a maximum of 9 points was awarded.