Non-graded Practice Quiz set for CS 4218 Software Testing and Debugging

Set by Abhik Roychoudhury

Please use these questions for your own practice.

They will not be graded. They are also not an indicator of any exam questions.

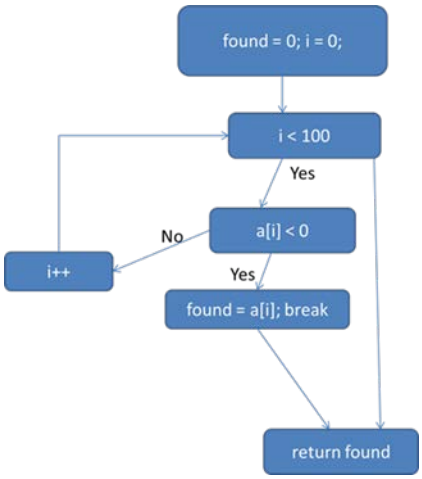
Question 1.

Consider the following program.

1. found = 0;
2. for (i = 0; i < 100; i++){
3. if (a[i] < 0){ found = a[i]; break;}
4. }
5. return found;

Consider an array which has a single negative number -1. When the program is executed with this array as input, what is the dynamic slice with respect to the value of found in line 5? More importantly, do you think the dynamic slice captures all influences in determining the value of found at the end of the program? Give critical comments and arguments. You may find it easier to draw the control flow graph before constructing the dynamic slice.

Answer: Control flow graph is as follows. The array mentioned in the question avoids the loop altogether. Therefore, the only influence in the return value is coming from the first statement, which is included in the dynamic slice. Statements 2,3,4 do not appear in the dynamic slice.



Question 2.

Consider the following program which adds up all elements of a given array. What are the legal execution paths of this program (that is, paths which are executed when a concrete array is fed to the program), and what are the path conditions of those paths?

```
sum = 0;
for (i = 0; i < 100; i++){  sum = sum + a[i];}
return sum;
```

Answer:

There is only one execution path in the program, which is executed on any input.

The path condition is true.

Question 3.

In class we discussed the usage of symbolic execution for test generation. As discussed, we can use symbolic execution to compute a *path condition* – a logical formula which is true for all inputs following a given path. Consider the following program which computes the greatest common divisor of two numbers x and y.

```
scanf("%d", &x); scanf("%d", &y);
while (x != y){ if (x >y) { x = x-y;} else {y = y - x;} }
printf("%d\n", x);
```

What is the path traced by the input (x == 8, y == 6)?
Compute the path condition of this path.

Answer:

The path traced by the input (x == 8, y == 6) is as follows

```
Scanf x; scanf y;
If (x > y)   // yes
     x = x – y    // x is now 8-6 = 2, y is 6
if  (x > y)   // no
     y = y – x   // y is now 6-2=4, x is 2
if  (x > y)   // no
     y = y – x   // y is now  4-2 = 2, x is 2
Exit loop  [x !=y] is false
```

The path condition is

$x > y \ /\backslash \ x - y < y \ /\backslash \ x - y < y - (x - y) \ /\backslash \ (x - y) == y - (x - y) - (x - y)$

$x > y \ /\backslash \ x < 2y \ /\backslash \ 2x < 3y \ /\backslash \ 3x == 4y$

Question 4.

Design a finite state machine (FSM) to model the requirements of a transaction controller software.

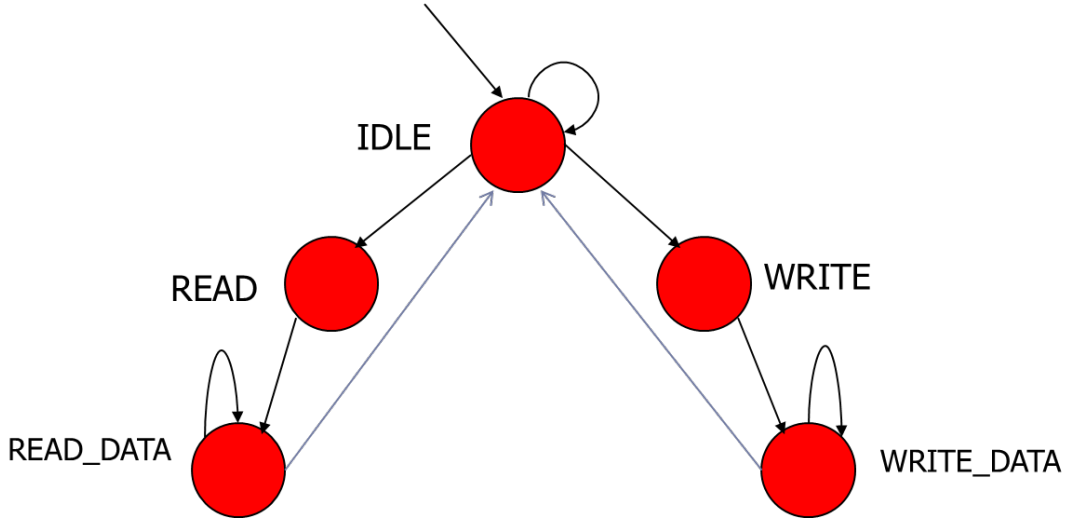
The controller is IDLE as long as a READ or a WRITE instruction does not begin on the bus.
On encountering a READ instruction, the controller moves to the READ state, sets a transaction variable as on-going, and waits for the peripheral to provide the READ data. Once READ is finished, it updates the transaction status to finished and moves back to IDLE state. On encountering a WRITE instruction, the controller has a similar behaviour, except for the fact that data is written to the peripheral once the peripheral is ready.

How would you generate test cases to cover all the states of this FSM? What will your test cases even look like, as far as this controller is concerned?

Answer: Each test case can be a run of the FSM e.g.

(IDLE, WRITE, WRITE_DATA, IDLE, READ, READ_DATA)$^{\omega}$ where ω denotes “repeated forever” - is a test

This one test is sufficient to achieve state coverage.



Question 5.

Consider a function which takes in an integer parameter n, and returns the factorial n!
It returns special values such as 0 for any integer over which factorial is not defined, and returns error values such as -1 for any integer which represents an overflow for the machine's integer representation. Use the category partition technique to construct a test-suite for this function.

Answer:

int factorial(int *n*)

1. **ITFs**: This function does only one thing, so we identify one Independently Testable Feature: *that the method correctly computes and returns the factorial of its parameter, n, or returns an appropriate error code*.

2. **Parameters and environment**: The parameters here are pretty obvious — there's only one, *n*.

What about environment? Is there anything about the containing system (operating system, computer, language, etc.) which might affect the behaviour of this function? I can think of one possibility, which is that int has a limited range.

Value classes for environment — size of int:

1. 16-bit
2. 32-bit
3. 64-bit

Value classes for parameters — *n*:

1. $n < -1$ (undefined: return 0)
2. $n = -1$ (undefined: return 0 — just below boundary)
3. $n = 0$ (just above boundary)
4. $0 < n < \text{max}$
5. $n = \text{max}$ (just below boundary)
6. $n = \text{max} + 1$ (out of range: return -1 — just above boundary)
7. $n > \text{max} + 1$ (out of range: return -1)

Where max is the largest factorial we can compute in the relevant environment

Generate test case specifications: We have three different environments and seven different value classes, so this means 3 × 7 = 21 tests, on the face of things. It's not quite so bad in practice though: we can reasonably expect that $n < -1$, $n = -1$, $n = 0$, $0 < n <$ max (for the smallest max, 7) and $n >$ max + 1 (for the largest max, 20) will behave pretty much the same way in all environments. So we could start our test specification out with these five:

Test case	Size of int	Value class for *n*	Expected result
1	any	Value class $n < -1$	0 (undefined)
2	any	$n = -1$	0 (undefined)
3	any	$n = 0$	1
4	any	$0 < n <$ max (for all environments)	varies (correct answer)
5	any	$n >$ max + 1 (for all environments)	-1 (out of range)

That leaves us trying to cover $n =$ max and $n =$ max + 1. max varies according to the size of int, so we do need 2 × 3 = 6 more cases for these:

Test case	Size of int	Value class for *n*	Expected result
6	16-bit	$n =$ max	(correct answer)
7	16-bit	$n =$ max + 1	-1 (out of range)
8	32-bit	$n =$ max	(correct answer)
9	32-bit	$n =$ max + 1	-1 (out of range)
10	64-bit	$n =$ max	(correct answer)
11	64-bit	$n =$ max + 1	-1 (out of range)