CS130: Software Engineering (Fall 2019)

Assignment 2 - Solutions

Due Date: October 23, 11:59 PM
Late Policy: There is a 10% penalty per every late day. Late assignments are not accepted after 48 hours. Once a solution is released, there's no credit.

Submission Instruction: Please submit **a single typed-out** report file (.pdf, .doc, .docx) to CCLE before the deadline.

Q1: Symbolic Execution

Alice wrote the following Java function and would like to reason about test adequacy of her function. Please perform symbolic execution on the Java function first and then answer the following sub-questions about testing.

```
static int compute(int x, int y, int z) {
S1: if (x < y) {
S2:     if (y >= z) {
S3:         z = 135;
        } else {
S4:         z = x;
S5:         y = z;
        }
    } else {
S6:     z = -1 * y;
    }

S7: if (z > 0 && z < x + y) {
S8:     x = z + 10;
    } else {
S9:     y = z - 10;
    }
S10:return x + y + z;
}
```

(a) Please identify the feasibility of each path,their corresponding path condition and effect by filling in the following table. The effect for the first path is given as an example.

S1	S2	S7	Path as statement labels	Feasible?	Path condition	Effect
T	T	T	{S1, S2, S3, S7, S8, S10}	Yes	X < Y && Y >= Z && X + Y > 135	x' = 145 y' = Y z' = 135
T	T	F	{S1, S2, S3, S7, S9, S10}	Yes	X < Y && Y >= Z && X + Y < 135	x' = X y' = 125 z' = 135
T	F	T	{S1, S2, S4, S5, S7, S8, S10}	Yes	X < Y && Y < Z && X > 0	x' = X + 10 y' = Z z' = X

F	T	T	{S1, S6, S7, S8, S10}	Yes	X >= Y && Y < 0 && -2Y < X	x' = -Y + 10 y' = Y z' = -Y
T	F	F	{S1, S2, S4, S5, S9, S10}	Yes	X < Y && Y < Z && (X < 0)	x' = X y' = X - 10 z' = X
F	F	T	{S1, S6, S7, S8, S10}	Yes	X >= Y && Y < 0 && -2Y < X	x' = -Y + 10 y' = Y z' = -Y
F	T	F	{S1, S6, S7, S9, S10}	Yes	X >= Y && (Y > 0 OR -2Y > X)	x' = X y' = -Y - 10 z' = -Y

F	F	F	{S1, S6, S7, S9, S10 }	Yes	X >= Y && (Y > 0 OR -2Y > X)	x' = X y' = -Y - 10 z' = -Y

(b) Please cluster the following test cases based on whether the test cases exercise the same path. In your answer, please identify each path using statement labels.

Example: if Si, Sj and Sk are the statements corresponding to path P1, you must represent path P1as P1 = {Si, Sj, Sk}. If test cases Ti, Tj and Tk cover the same path P1, then you must represent your answer as the following: **{Ti, Tj, Tk} covers path {Si,Sj,Sk}.**

T1: x = 10 AND y = 7 AND z = 5
T2: x = 15 AND y = 11 AND z = 7
T3: x = 3 AND y = 4 AND z = 9
T4: x = 100 AND y = -190 AND z= 10
T5: x = 99 AND y = 100 AND z = 47

{T1, T2, T4} covers {S1, S6, S7, S9, S10}

{T3} covers {S1, S2, S4, S5, S7, S8, S10}

{T5} covers {S1, S2, S3, S7, S8, S10}

Q2: Test Coverage

In the following Java program, each statement is marked with labels:

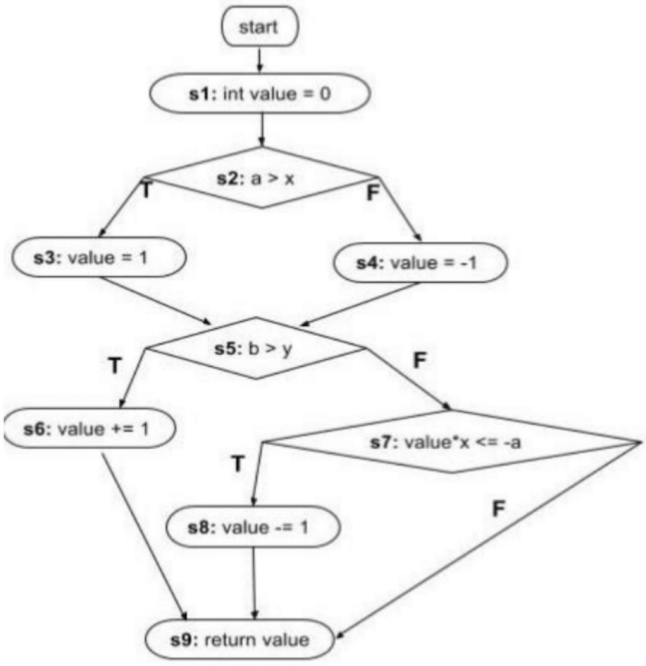
```
     int coverage(int a, int b, int x, int y){
s1     int value = 0;

s2     if (a > x) {
s3          value = 1;
         } else {
s4          value = -1;
         }

s5     if (b > y) {
s6          value += 1;
s7     } else if (value * x <= -a) {
s8          value -= 1;
         }

s9     return value;
}
```

A) Please draw the control flow graph of this Java program. Please annotate each CFG node with the corresponding statement labels **[5pts]**



B) Please create a minimum number of tests to achieve 100% statement coverage. Please write your tests in the format of JUnit test cases and explain which statements each test covers using the labels. **[5pts]**

test1 + test2

```
@Test
public void test1() {
    assertEquals(2, coverage(1, 1, 0, 0)); // TT, s1,s2,s3,s5,s6,s9
}

@Test
public void test2() {
    assertEquals(-2, coverage(0, 0, 1, 1)); // FFT, s1,s2,s4,s5,s7,s8,s9
}
```

C) Please augment your tests in (b) with a minimum number of additional tests to achieve 100% branch coverage. **[5pts]**

```
@Test
public void test3() {
    assertEquals(1, compare(1, 0, 0, 1)); // TFF, s1,s2,s3,s5,s7,s9
}
```

D) Please augment your tests in (b) and (c) with a minimum number of additional tests to achieve 100% coverage of all feasible paths. **[5pts]**

```
public void test4() {
    assertEquals(0, compare(0, 1, 1, 0)); // FT, s1,s2,s4,s5,s6,s9
}

@Test
public void test5() {
    assertEquals(0, compare(1, 0, -10, 1)); // TFT, s1,s2,s3,s5,s7,s9
}
```

E) Identify all infeasible paths (if any) and explain why. **[5pts]**

There is only one infeasible path.
s1,s2,s4,s5,s7,s9 Because if the first if predicate is false, then a <= x and value = -1. So value * x <= -a is always true and the third if statement will never step to the else branch.

Q3 Bounded Loop Iteration [20pts]

Consider the following Java program.

```
int loop(int N) {
    Random rand = new Random();
    for(int i=0; i<N; i++)
    {
        for(int j=0; j<Math.pow(2,3*(i+2)); j++)
        {
            if(rand.nextInt() % 2 == 0)  //s1
            {
                System.err.println("Reached here!");
            }
        }
    }
}
```

(a) Given N, how many times is the “if condition” (labelled as s1) examined? Please include your derivation in your answer. [10pts]

T(N) = 2^6 + ...+ (2^3)^(N+2) = 8^2 + 8^3 + ...+ 8^(N+1)

8*T(N) = 8^6 + +.....+ 8^(N+2)

---------- (subtract)

7*T(N) = (8^(N+2) - 64)

=> ans: (8^(N+2) - 64)/7

(b) How many feasible paths do you have in this program? The answer should only contain N as a variable. [10pts]

2^[(8^(N +2) - 64)/7]

Q4 Who's the Weakest? [20pt]

Assume that the post-condition of the following Java program is TRUE, meaning that it should terminate without throwing any exceptions. Through the weakest precondition reasoning, your goal is to identify the program inputs and intermediate program states such that the program produces the post-condition of TRUE. While the sub-questions are multiple choice questions, please include derivation of weakest precondition reasoning in your answer.

```
    private int getSomeElement(MyObject o, int len, int index){
        int[] s;
        s = new int[len];
        if (o != null) {
P1:
            len = o.s;
            s = o.arr;
        else {
P2:
            len= len - index;
        }
P3:
        return s[len-5];
    }

    class MyObject{
        int s;
        int[] arr;
    }
```

(a) What is the weakest pre-condition at the program point P1?

A. o.arr != null AND o.s < o.arr.length + 5

B. o.s >= 5 AND o.arr != null AND o.s < o.arr.length + 5

C. o.s > 5 AND o.arr != null AND o.s < o.arr.length + 5

D. o.s >= 5 AND o.arr != null AND o.s < o.arr.length + 5 AND o.s != null

E. None of the above

B or D

2 for right answer, 3 for proof

(b) Which of the following is a precondition at the program point P2 but is NOT the weakest? Choose all that apply.

A. index + 10 <= len AND len < s.length + index + 5

B. index <= len AND len < s.length + index + 5

C. index + 5 <= len AND len < s.length + index + 5

D. index + 5 <= len AND len < s.length AND index > -5

E. None of the above

A, D

2 for both options correct, 1 for just one option correct, 3 for proof, 2 for partially correct proof

(c) What is the weakest pre-condition at the program point P3?

A. 5 < len AND len < s.length + 5

B. 5 <= len AND len < s.length

C. 5 <= len AND len < s.length + 5

D. 5 <= o.s AND len < o.arr.length + 5

E. None of the above

C

(d) What is the weakest precondition of the entire program (at the beginning of the program)?

A. (o != null AND o.s >= 0 AND o.arr != null AND o.s < o.arr.length)
OR (o == null AND index + 5 <= len AND index > -5 AND len >= 0)

B. (o != null AND o.s >= 5 AND o.arr != null AND o.s < o.arr.length + 5)
OR (o == null AND index <= len AND len >= 0)

C. (o != null AND o.s >= 5 AND o.arr != null AND o.s < o.arr.length)
OR (o == null AND index <= len + 5 AND index > -5 AND len >= 0)

D. (o != null AND o.s >= 5 AND o.arr != null AND o.s < o.arr.length + 5)
OR (o == null AND index + 5 <= len AND index > -5 AND len >= 0) // len>=0 should be applied to the first part as well.

E. None of the above

D

Proof

```
private int getSomeElement(MyObject o, int len, int index){
    {(o != null AND o.s >= 5 AND o.arr != null AND o.s - 5 < o.arr.length)
    OR (o == null AND (len - index) >= 5 AND (len - index -5) < len)} AND (len >= 0)
    int[] s;
    wp(s = new int[len], ((o != null AND o.s >= 5 AND o.arr != null AND o.s - 5 < o.arr.length)
    OR (o == null AND (len - index) >= 5 AND (len - index -5) < s.length)))
```

```
        === {(o != null AND o.s >= 5 AND o.arr != null AND o.s - 5 < o.arr.length)
    OR (o == null AND (len - index) >= 5 AND (len - index -5) < len)} AND (len >= 0)
    // previous write up of the solution had an error with precedence. (X OR Y) AND (len>=0)
    not (X OR Y AND len>=0)
    s = new int[len];
    wp(if(o != null) S1 else s2, len >= 5 AND len - 5 < s.length)
        === (o != null AND wp(s1, len >= 5 AND len - 5 < s.length)) OR (o == null AND
    wp(s2, len >= 5
        AND len - 5 < s.length)) === (o != null AND o.s >= 5 AND o.arr != null AND o.s -
    5 < o.arr.length)
    OR (o == null AND (len - index) >= 5 AND (len - index -5) < s.length)
    if (o != null) {
        P1:
        wp(len = o.s, len >= 5 AND o.arr != null AND len-5 < o.arr.length)
        == o.s >= 5 AND o.arr != null AND o.s - 5 < o.arr.length
    //It is fine to add o!=null, but because o.s is accessed and read, one may argue that it is
    already implied by o.s >= 5
    len = o.s;
    wp(s = o.arr, len >= 5 AND len-5 < s.length)
    == len >= 5 AND o.arr != null AND len - 5 < o.arr.length
    s = o.arr;
    wp: len >= 5 AND len-5 < s.length
    else {
    P2:
    wp(len = len - index, len >= 5 AND len - 5 < s.length) == (len - index >= 5) AND (len -
    index -5 < s.length)
    len= len - index;
    wp: len >= 5 AND len - 5 < s.length
    }
    P3:
    wp: len >= 5 AND len - 5 < s.length
    return s[len-5];
    postcondition is true
}
```

Q5: Mutants [10pts]

The following function taken an integer array and length of an array as input, and sorts the array in ascending order. In order to assess the adequacy of existing test suites, Tom leverages the mutation analysis to create three variants of the function and see if existing tests can kill such mutants as shown below. *Mutant 1* replaces the constant '1' in the if

condition in line 4 with '2'. *Mutant 2* replaces the greater than operator in line 9 with less than. *Mutation 3* deletes the statement in line 14.

```
1    void bubble_sort(int list[], int n)
2    {
3      int i, j, temp;
4      if( n == 0 || n == 1) return; // mutant 1: n == 1 is changed to n == 2
5      for (i = 0 ; i < ( n - 1 ); i++)
6      {
7         for (j = 0 ; j < n - i - 1; j++)
8         {
9            if (list[j] > list[j+1]) // mutant 2: > is changed to <
10           {
11              /* Swapping */
12              temp = list[j];
13              list[j] = list[j+1];
14              list[j+1] = temp; // mutant 3: this statement is deleted
15           }
16        }
17    }
18   }
```

Please answer the following multiple choice questions. Each choice in the questions indicates a test case in the format of (input, expected output). For example, (list = [1,3,2] ^ n=3, list = [1,2,3]) means that given an array [1,3,2] and the array length is 3, the function will sort the array to [1,2,3].

A. **[5pts]** Please select all tests that kill mutant 1:
   a. (list = [] ^ n=0, list=[])
   b. (list = [1] ^ n=1, list=[1])
   c. (list = [2,1] ^ n=2, list=[1,2])
   d. (list = [1,3,2] ^n=3, list=[1,2,3])

B. **[5pts]** Please select all tests that kill Mutant 1, Mutant 2, Mutant 3:
   a. (list=[-1,4]^ n=2, list=[-4,-1])
   b. (list=[1,2]^ n=2, list=[1,2])
   c. (list=[3,2,1,0]^ n=4, list=[0,1,2,3])
   d. (list=[]^n=0, list=[])