Ryerson University

Faculty of Engineering & Architectural Science

Department of Biomedical, Computer, and Electrical Engineering

Course Number	COE891
Course Title	Software Testing and Quality Assurance
Semester/Year	Winter2024
Instructor	Dr. Reza Samavi
Section No.	01
Group No.	N/A
Submission Date	Mar 25, 2024
Due Date	Mar 25, 2024 10:00 AM

Lab/Tut Assignment NO.	5

Assignment Title	Logic Coverage

Name	**Student ID**	**Signature***
Astha Patel	501040209	

*By signing above you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: www.ryerson.ca/senate/current/pol60.pdf

Q1

1. Input: `p = a & (!b | c)`

Truth Table:

Row#	a	b	c	P	Pa	Pb	Pc
1	T	T	T	T	T		T
2	T	T				T	T
3	T		T	T	T		
4	T			T	T	T	
5		T	T		T		
6		T					
7			T		T		
8					T		

Figure 1: Truth table for the given predicate p.

2. Condition under which each clause determines p:
   - If c is false then b determines (!b | c)
   - If b is true then c determines (!b | c)
   - Both a and (!b | c) determine p since the predicate involves an & operator.

3. All row pairs for each major clause to satisfy each of the following

   1. GACC

The following result for GACC is based on the truth table on the right:

Major Clause	Set of possible tests
a	(1,5), (1,7), (1,8), (3,5), (3,7), (3,8), (4,5), (4,7), (4,8)
b	(2,4)
c	(1,2)

Figure 2: GACC based on the table provided in Figure 1.

2. CACC

The following result for CACC is based on the truth table on the right:

Major Clause	Set of possible tests
a	(1,5), (1,7), (1,8), (3,5), (3,7), (3,8), (4,5), (4,7), (4,8)
b	(2,4)
c	(1,2)

Figure 3: CACC based on the table provided in Figure 1.

3. RACC

The following result for RACC is based on the truth table on the right:

Major Clause	Set of possible tests
a	(1,5), (3,7), (4,8)
b	(2,4)
c	(1,2)

Figure 4: RACC based on the table provided in Figure 1.

4. GICC

The following result for GICC is based on the truth table on the right:

Major Clause	Set of possible tests	
a	No feasible pairs for P = T	P = F: (2,6)
b	P = T: (1,3)	P = F: (5,7), (5,8), (6,7), (6,8)
c	P = T: (3,4)	P = F: (5,6), (5,8), (7,6), (7,8)

Figure 5: GICC based on the table provided in Figure 1.

5. RICC

The following result for RICC is based on the truth table on the right:

Major Clause	Set of possible tests	
a	No feasible pairs for P = T	P = F: (2,6)
b	P = T: (1,3)	P = F: (5,7), (6,8)
c	P = T: (3,4)	P = F: (5,6), (7,8)

Figure 6: RICC based on the table provided in Figure 1.

Q2

```
int x = y;
while (x < 100) {
  if (x < y) {
    x++;
    break;}
  for (int z = 1; z < x; z++)
    x += z;
  if (x > 5)
    y++;
  else
    y += 2;}
System.out.println(x + "," + y);
```

1. Control Flow Graph (CFG) and Data Flow Graph (DFG)

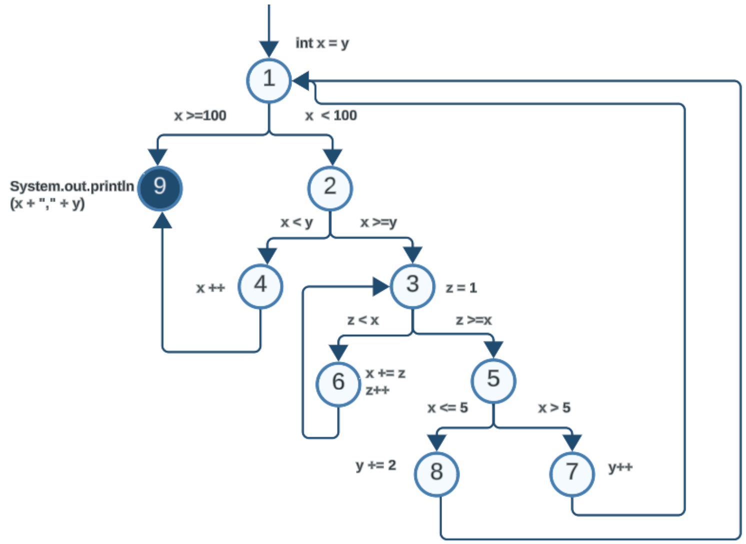

Figure 7: CFG

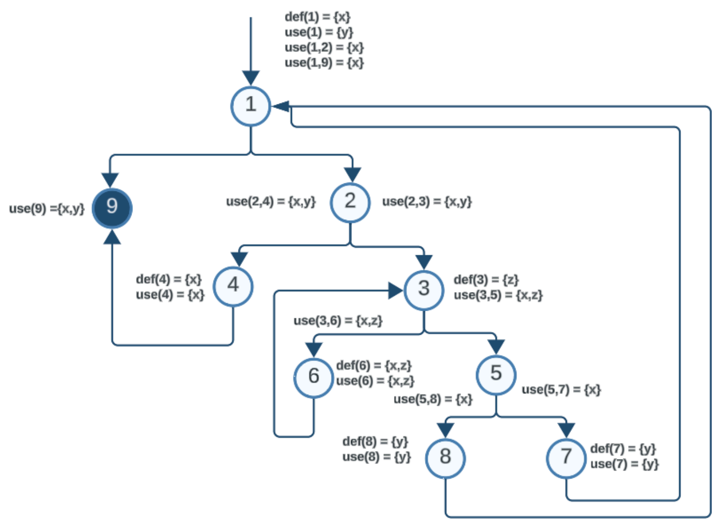

Figure 8: DFG for
CFG in Figure 7.

2. DU pairs for each variable

Table 1: DU pairs for each variable.

Variable	**DU Pair**
x	(1,(1,2)),(1,(1,9)),(1,9),(1,(2,3)),(1,(2,4)),(1,(3,5)), (1,(3,6)),(1,6),(6,6),(1,4),(4,4),(1,(5,7)),(1,(5,8)),(4,(3,5)), (4,(3,6)),(4,6),(4(5,7)),(4,(5,8)),(4,9),(6,(3,6)),(6,(3,5)),(6,(5,7)), (6,(5,8)),(6,9)
y	(1,1),(1,(2,3)),(1,(2,4)),(7,7),(8,8),(8,9),(7,9),(1,9)
z	(3,(3,6)),(6,6),(3,(3,5)),(3,6),(6,(3,5)),(6,(3,6))

3. There is one infeasible path on the CFG, which is the if(x < y) condition. Since x is initialized to y, x = y, we are essentially comparing two of the same values. Hence, the condition will always be false and will never be reachable leading to an infeasible path in the program.
4. DU test paths for def-coverage:

```
Test cases:
  - t1: y = 6, satisfies x < 100 and x > 5
      - defs satisfied: {(1),(3),(6),(7)}
  - t2: y = 100, x = 100
      - defs satisfied: {(1)}
  - t3: y = 2, satisfies x < 100 but not x > 5
      - defs satisfied: {(1),(3),(6),(8)}
  - Def(4) cannot be satisfied because the path is not reachable: It is an
    infeasible path.
```

5. DU test paths for use-coverage. Note: the same paths used for def-coverage can be utilized for use-coverage in this case:

```
Test cases:
   - t1: y = 6, satisfies x < 100 and x > 5
        - uses satisfied: {(1),(1,2),(2,3),(3,6),(3,5),(5,7),(7),(9)}
   - t2: y = 100, x = 100
        - uses satisfied: {(1),(9)}
   - t3: y = 2, satisfies x < 100 but not x > 5
        - uses satisfied: {(1),(1,2),(2,3),(3,6),(3,5),(5,8),(8),(9)}
```

- *use(4) cannot be satisfied because the path is not reachable: It is an infeasible path.*

6. Test set that satisfied all def-use coverage:

```
Test cases:
   - t1: y = 6, satisfies x < 100 and x > 5
        - DU-covered:{(1,2), (1,9), (1,2,3), (1,2,3,5), (1,2,3,5,7), (3,5,7), (3,7),
          (7,7)}
   - t2: y = 100, x = 100
        - DU-paths covered: {(1,9)}
   - t3: y = 2, satisfies x < 100 but not x > 5
        - DU-paths covered: {(1,2)), (1,9), (1,2,3), (1,2,3,5), (1,2,3,5,8),
          (3,5,8), (8,8)}
```

- *DU case (1,2,4,9) cannot be satisfied because the path is not reachable: It is an infeasible path.*

Q3

```java
public class TriangleType {
 /**
  * @param sl, s2, s3: sides of the putative triangle
  *
  * @return enum describing type of triangle
  */
 public Triangle triangle(int sl, int s2, int s3) {
    // Reject non-positive sides
    if (sl <= 0 || s2 <= 0 || s3 <= 0) {
      return Triangle.INVALID;
    }
    // Check triangle inequality
    if (s1 + s2 <= s3 || s2 + s3 <= sl || s1 + s3 <= s2) {
      return Triangle.INVALID;
    }
    // Identify equilateral triangles
    if ((sl == s2) && (s2 == s3)) {
      return Triangle.EQUILATERAL;
    }
    // Identify isosceles triangles
    if ((sl == s2) || (s2 == s3) || (s1 == s3)) {
      return Triangle.ISOSCELES;
    }
    return Triangle.SCALENE;
  }
}
  public enum Triangle {
   SCALENE, ISOSCELES, EQUILATERAL, INVALID
  }
```

A list of predicates in Q3:

PC1: `sl <= 0 || s2 <= 0 || s3 <= 0`

PC2: `s1 + s2 <= s3 || s2 + s3 <= sl || s1 + s3 <= s2`

PC3: `(s1 == s2) && (s2 == s3)`
PC4: `(s1 == s2) || (s2 == s3) || (s1 == s3)`

1. Reachability predicates
   - Line 9: This predicate `s1 <= 0 || s2 <= 0 || s3 <= 0` is always reached.
   - Line 13: The predicate in line 9 must be false to reach the predicate in line 13
     - The following predicate blocks satisfy then the predicate in line 21 will be reached: `!(s1 <= 0 || s2 <= 0 || s3 <= 0)` .
   - Line 17: The predicate in line 9 and line 13 must be false to reach the predicate in line 17.
     - The following predicate blocks satisfy then the predicate in line 21 will be reached: `!(s1 <= 0 || s2 <= 0 || s3 <= 0)` and `!(s1 + s2 <= s3 || s2 + s3 <= s1 || s1 + s3 <= s2)` .
   - Line 21: The predicate in line 9,13, and 17 must be false the predicate in line 21 to be reached.
     - The following predicate blocks satisfy then the predicate in line 21 will be reached: `!(s1 <= 0 || s2 <= 0 || s3 <= 0)` , `!(s1 + s2 <= s3 || s2 + s3 <= s1 || s1 + s3 <= s2)` , and `!((s1 == s2) && (s2 == s3))` .

2. TRs and test cases that satisfy PC
   - TR = To achieve PC, the predicate evaluates to **true**, and predicate evaluates to **false**.

   - PC1: any of the following conditions will satisfy PC1
     - s1 = 3, s2 = 4, s3 = 5 (false)
     - s1 = 0, s2 = 0 , s3 = 0 (true)
     - s1 = -1, s2 = -4 , s3 = -5 (true)

   - PC2: any of the following conditions will satisfy PC2
     - s1 = 3, s2 = 4, s3 = 5 (false)
     - s1 = 1, s2 = 100, s3 = 5 (true)

   - PC3: any of the following conditions will satisfy PC3
     - s1 = 5, s2 = 5, s3 = 5 (true)
     - s1 = 5, s2 = 6, s3 = 5 (false)

- PC4: any of the following conditions will satisfy PC4
    - s1 = 7, s2 = 8, s3 = 10 (true)
    - s1 = 1, s2 = 1, s3 = 1 (false)

3. TRs and test cases that satisfy CC
    - TR = To achieve CC, the clause evaluates to **true**, and clause evaluates to **false**.

- Clauses in PC1: `C1 : s1 <= 0` , `C2 : s2 <= 0` , `C3 : s3 <= 0`
    - Test cases:
        - s1 = - 4, s2 = -7, s3 = -10
            - C1, C2, C3 → true
        - s1 = 1, s2 = 2 , s3 = 2
            - C1, C2, C3 → false

- Clauses in PC2: `C1 : s1 + s2 <= s3` , `C2 : s2 + s3 <= s1` , `C3 : s1 + s3 <= s2`
    - Test cases:
        - s1 = 2, s2 = 3, s3 = 10
            - C1 → true; 2 + 3 <= 10
            - C2 → false; !(3 + 10 <= 2)
            - C3 → false; !(2 + 10 <= 3)
        - s1 = 100, s2 = 4 , s3 = 3
            - C1 → false; !(100 + 4 <= 3)
            - C2 → true; 4 + 3 <= 100
            - C3 → false ; !(100 + 3 <= 4)
        - s1 = 5, s2 = 11, s3 = 6
            - C1 → false; !(5 + 11 <= 6)
            - C2 → false; !(11 + 6 <= 5)
            - C3 → true; 6 + 5 <= 11

- Clauses in PC3: `C1 : (s1 == s2)` , `C2: (s2 == s3)`

- Test cases:
    - s1 = 5, s2 = 5, s3 = 5
        - C1 → true; 5 == 5
        - C2 → true; 5 == 5
    - s1 = 5 , s2 = 6 , s3 = 4
        - C1 → false; 5 != 6
        - C2 → false; 6 != 4

- Clauses in PC4: `c1 : (s1 == s2) , c2 : (s2 == s3) , c3 : (s1 == s3)`
    - Test cases:
        - s1 = 1, s2 = 2, s3 = 2
            - C1 → false; 1 != 2
            - C2 → true; 2 == 2
            - C3 → false; 1 != 2
        - s1 = 2, s2 = 2, s3 = 1
            - C1 → true ; 2 == 2
            - C2 → false; 2 != 1
            - C3 → false; 2 != 1
        - s1 = 2, s2 = 1, s3 = 2
            - C1 → false; 2 != 1
            - C2 → false; 1 != 2
            - C3 → true; 2 == 2

4. Determination predicates (compute and simplify).
    - Predicates PC1, PC2, PC3, and PC4 are all determination predicates.

5. TRs and test cases that satisfy CACC (or RACC).
    - PC1: `s1 <= 0 || s2 <= 0 || s3 <= 0`
        - If C1 is a major clause and C2 and C3 are false then C1 determines PC1.
            - s1 = -1, s2 = 2, s3 = 2
        - If C2 is a major clause and C1 and C3 are false then C1 determines PC1.

    - s1 = 1, s2 = -2, s3 = 1
  - If C3 is a major clause and C1 and C2 are false then C3 determines PC1.
    - s1 = 1, s2 = 1, s3 = -4
  - This CC satisfies RACC since the minor clauses have the same value while the major clause determines the PC1.

- PC2: `s1 + s2 <= s3 || s2 + s3 <= s1 || s1 + s3 <= s2`
  - If C1 is a major clause and C2 and C3 are false then C1 determines PC1.
    - s1 = 2, s2 = 3, s3 = 10
  - If C2 is a major clause and C1 and C3 are false then C1 determines PC1.
    - s1 = 100, s2 = 4, s3 = 3
  - If C3 is a major clause and C1 and C2 are false then C3 determines PC1.
    - s1 = 5, s2 = 11, s3 = 6
  - This CC satisfies CACC but not RACC since the minor clauses do not have the same value while the major clause determines the PC2.

- PC3: `(s1 == s2) && (s2 == s3)`
  - Both C1 and C2 must be true to satisfy the predicate
    - s1 = 5, s2 = 5, s3 = 5

- PC4: `(s1 == s2) || (s2 == s3) || (s1 == s3)`
  - If C1 is a major clause and C2 and C3 are false then C1 determines PC1.
    - s1 = 2, s2 = 2, s3 = 1
  - If C2 is a major clause and C1 and C3 are false then C1 determines PC1.
    - s1 = 1, s2 = 2, s3 = 2
  - If C3 is a major clause and C1 and C2 are false then C3 determines PC1.
    - s1 = 2, s2 = 1, s3 = 2
  - This CC satisfies CACC but not RACC since the minor clauses do not have the same value while the major clause determines PC4.

6. Infeasible requirements are not present in this code for the Triable class because all predicates justify checking for the attributes of a triangle. Here is a breakdown for why these predicates are feasible requirements:
   - PC1: This checks for non-positive sides, which is a valid requirement for a triangle. A triangle cannot have sides with zero or negative lengths.
   - PC2: This enforces the triangle inequality, a fundamental geometric rule. The sum of any two sides in a triangle must be greater than the third side.
   - PC3: This identifies equilateral triangles where all sides are equal. This is a valid triangle classification.
   - PC4: This identifies isosceles triangles where at least two sides are equal. This is also a valid triangle classification