Alex Lundin Testing Final

Use Triangle, a numbered version of which is available here, to answer the questions below. Only the triangle() method is considered.

public class Triangle{

   /\*\*

     \* @param s1, s2, s3:  sides of the putative triangle

     \* @return String describing type of triangle

     \*/

1.   public static String triangle(int s1, int s2, int s3){

      // Reject non-positive sides

2.      if (s1 <= 0 || s2 <= 0 || s3 <= 0)

3.        return "INVALID";

      // Check triangle inequality

4.      if (s1+s2 <= s3 || s2+s3 <= s1 || s1+s3 <= s2)

5.        return "INVALID";

      // Identify equilateral triangles

6.      if ((s1 == s2) && (s2 == s3))

7.        return "EQUILATERAL";

      // Identify isosceles triangles

8.     if ((s1 == s2) || (s2 == s3) || (s1 == s3))

9.        return "ISOSCELES";

10.    return "SCALENE";

11.   }}

Q1. List all predicates in the triangle() method. Index them by the line numbers in the program listing.

**List all predicates**

* + **p1: (s1 <= 0 || s2 <= 0 || s3 <= 0)**
  + **Simplify:**
    - **a = s1 <= 0**
    - **b = s2 <= 0**
    - **c = s3 <= 0**
    - **(a || b || c )**
  + **p2: (s1+s2 <= s3 || s2+s3 <= s1 || s1+s3 <= s2)**
  + **Simplify:**
    - **a = s1+s2 <= s3**
    - **b = s2+s3 <= s1**
    - **c = s1+s3 <= s2**
    - **(a || b || c )**
  + **p3: ((s1 == s2) && (s2 == s3))**
  + **Simplify:**
    - **a = (s1 == s2)**
    - **b = (s2 == s3))**
    - **(a && b)**
  + **p4: ((s1 == s2) || (s2 == s3) || (s1 == s3))**
  + **Simplify:**
    - **a = s1 == s2**
    - **b = s2 == s3**
    - **c = s1 == s3**
    - **(a || b || c )**

Q2. Find values (concrete value for s1, s2 and s3) for each predicate to satisfy predicate coverage (PC).

**Note: When you pick the values first you need to make sure that the predicate is reachable. For example, if the triangle is “INVALID” then no other predicates are reachable or if the triangle is “EQUILATERAL” then you cannot cover “ISOSCELES”.**

**P1**

(s1 <= 0 || s2 <= 0 || s3 <= 0)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | a | b | c | P1 |
| R1 | T | T | T | T |
| R2 | F | T | F | T |
| R3 | T | F | T | T |
| R4 | T | F | F | T |
| R5 | F | T | T | T |
| R6 | F | T | F | T |
| R7 | F | F | T | T |
| R8 | F | F | F | F |

1. **s1 = -1, s2 = -1, s3 = -1 makes entire predicate T**
2. **s1 = 1, s2 = 1, s3 = 1 makes entire predicate F**

**P2**

(s1+s2 <= s3 || s2+s3 <= s1 || s1+s3 <= s2)

Same table as P1, with different clauses in the predicates

1. **s1 = 1, s2 = 1, s3 = 1 makes entire predicate T**
2. **s1 = 2, s2 = 3, s3 = 4 makes entire predicate F**

**P3**

((s1 == s2) && (s2 == s3))

|  |  |  |  |
| --- | --- | --- | --- |
|  | a | b | P3 |
| R1 | T | T | T |
| R2 | T | F | F |
| R3 | F | T | F |
| R4 | F | F | F |

1. **s1 = 1, s2 = 1, s3 = 1 makes entire predicate T**
2. **s1 = 1, s2 = 2, s3 = 3 makes entire predicate F**

**P4**

((s1 == s2) || (s2 == s3) || (s1 == s3))

Same table as P1, with different clauses in the predicates

1. **s1 = 1, s2 = 2, s3 = 1 makes entire predicate T**
2. **s1 = 2, s2 = 3, s3 = 4makes entire predicate F**

Q3. Define the control flow graph for triangle() method.

Note: You need to define the set of edges (E) that defines your graph, the initial nodes (N0) and the final nodes (Nf). To simplify grading, use the line numbers as node numbers.

N={ 1, 2,3,4, 5, 6, 7, 8, 9,10 }

N0={ 1 }

Nf={ 3,5,7,9,10 }

E={ (1,2) , (2,3) , (2,4) , (4,5), (4,6), (6,7), (6,8), (8,9), (8,10) }

Q4. Use the triangle() method and your control flow graph from Q3

Q4.1.    List the Edge-Pair Coverage test requirements

TR = {[1,2,3], [1,2,4], [2,4,5], [2,4,6], [4,6,7], [4,6,8], [6,8,9], [6,8,10]}

Q4.2.    List test paths that achieve Edge-Pair Coverage (EPC)

{[1,2,**3**], [2,4,**5**], [4,6,**7**], [6,8,**9**], [6,8,**10**]}

Q5. Table 1 shows a functionality-based approach that use the semantic information of the geometric classification of triangles.

Table 1: Geometric Partitioning of triangle

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Characteristics** | SCALENE | EQUILATERAL | ISOSCELES | INVALID |
| **Block 1** | True | True | True | True |
| **Block 2** | False | False | False | False |

Do partitions satisfy disjointness and completeness properties? What are the constraints?

Disjointness:

No, by definition an isosceles triangle has at least two sides equal to each other.

This means an isosceles triangle with all three sides equal will overlap the equilateral triangle block. The sets are no disjoint.

Completeness:

Yes, the invalid blocks accounts for negative and zero sided triangles. Which are impossible. The boundary conditions are accounted for.

Q6. How many tests are needed to satisfy the All combinations criterion (ACoC)? Please list all the tests. Mark the infeasible tests based on the constraints from Q5.

2\*2\*2\*2 = 16 tests

A -SCALENE

B- EQUILATERAL

C- ISOSCELES

D- INVALID

Block 1 – true

Block 2 - false

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Characteristics** | A | B | C | D |
| **Block 1** | A1 | B1 | C1 | D1 |
| **Block 2** | A2 | B2 | C2 | D2 |

**Red are infeasible**

**A1, B2, C2, D2**

A1, B1, C2, D2

A1, B2, C1, D2

A1, B2, C2, D1

**A2, B1, C2, D2**A1, B2, C1, D2

A2, B1, C1, D2

A2, B1, C2, D1

**A2, B2, C1, D2**

A1, B2, C1, D2

A2, B1, C1, D2

A2, B2, C1, D1

**A2, B2, C2, D1**

A1, B2, C2, D1

A2, B1, C2, D1

A2, B2, C1, D1