

Introduction to Software Testing

Chapter 3.3

Logic Coverage for Source Code

Paul Ammann & Jeff Offutt

www.introsoftwaretesting.com

Logic Expressions from Source

- Predicates are derived from decision statements in programs
- In programs, most predicates have less than four clauses
 - Wise programmers actively strive to keep predicates simple
- When a predicate only has one clause, COC, ACC, ICC, and CC all collapse to predicate coverage (PC)
- Applying logic criteria to program source is hard because of reachability and controllability:
 - Reachability: Before applying the criteria on a predicate at a particular statement, we have to get to that statement
 - Controllability: We have to find input values that indirectly assign values to the variables in the predicates
 - Variables in the predicates that are not inputs to the program are called *internal variables*
- These issues are illustrated through an example in the following slides ...

Triang (pg 1 of 5)

```
1 // Jeff Offutt -- Java version Feb 2003
2 // The old standby: classify triangles
3 // Figures 3.2 and 3.3 in the book.
4 import java.io.*;
5 class trityp
6 {
7     private static String[] triTypes = { "", // Ignore 0.
8         "scalene", "isosceles", "equilateral", "not a valid
9         triangle"};
10    private static String instructions = "This is the ancient
11    TriTyp program.\nEnter three integers that represent the lengths
12    of the sides of a triangle.\nThe triangle will be categorized as
13    either scalene, isosceles, equilateral\nor invalid.\n";
14    public static void main (String[] argv)
15    { // Driver program for trityp
16        int A, B, C;
17        int T;
```

Triang (pg 2 of 5)

```
16 System.out.println (instructions);
17 System.out.println ("Enter side 1: ");
18 A = getN();
19 System.out.println ("Enter side 2: ");
20 B = getN();
21 System.out.println ("Enter side 3: ");
22 C = getN();
23 T = Triang (A, B, C);
24
25 System.out.println ("Result is: " + triTypes [T]);
26 }
27
28 // =====
```

Triang (pg 3 of 5)

```
29 // The main triangle classification method
30 private static int Triang (int Side1, int Side2, int Side3)
31 {
32     int tri_out;
33
34     // tri_out is output from the routine:
35     //   Triang = 1 if triangle is scalene
36     //   Triang = 2 if triangle is isosceles
37     //   Triang = 3 if triangle is equilateral
38     //   Triang = 4 if not a triangle
39
40     // After a quick confirmation that it's a legal
41     // triangle, detect any sides of equal length
42     if (Side1 <= 0 || Side2 <= 0 || Side3 <= 0)
43     {
44         tri_out = 4;
45         return (tri_out);
46     }
```

Triang (pg 4 of 5)

```
48     tri_out = 0;
49     if (Side1 == Side2)
50         tri_out = tri_out + 1;
51     if (Side1 == Side3)
52         tri_out = tri_out + 2;
53     if (Side2 == Side3)
54         tri_out = tri_out + 3;
55     if (tri_out == 0)
56     { // Confirm it's a legal triangle before declaring
57         // it to be scalene
58
59         if (Side1+Side2 <= Side3 || Side2+Side3 <= Side1 ||
60             Side1+Side3 <= Side2)
61             tri_out = 4;
62         else
63             tri_out = 1;
64         return (tri_out);
65     }
```

Triang (pg 5 of 5)

```
67  /* Confirm it's a legal triangle before declaring */
68  /* it to be isosceles or equilateral */
69
70  if (tri_out > 3)
71    tri_out = 3;
72  else if (tri_out == 1 && Side1+Side2 > Side3)
73    tri_out = 2;
74  else if (tri_out == 2 && Side1+Side3 > Side2)
75    tri_out = 2;
76  else if (tri_out == 3 && Side2+Side3 > Side1)
77    tri_out = 2;
78  else
79    tri_out = 4;
80  return (tri_out);
81 } // end Triang
```

Ten Triang Predicates

```
42: (Side1 <= 0 || Side2 <= 0 || Side3 <= 0)
49: (Side1 == Side2)
51: (Side1 == Side3)
53: (Side2 == Side3)
55: (triOut == 0)
59: (Side1+Side2 <= Side3 || Side2+Side3 <= Side1 ||
     Side1+Side3 <= Side2)
70: (triOut > 3)
72: (triOut == 1 && Side1+Side2 > Side3)
74: (triOut == 2 && Side1+Side3 > Side2)
76: (triOut == 3 && Side2+Side3 > Side1)
```

Reachability for Triang Predicates

42: True

49: P1 = s1>0 && s2>0 && s3>0

51: P1

53: P1

55: P1

59: P1 && (triOut)=0

62: P1 && (triOut)=0

&& (s1+s2 > s3) && (s2+s3 > s1) && (s1+s3 > s2)

70: P1 && (triOut)!=0

72: P1 && (triOut)!=0 && (triOut)<= 3

74: P1 && (triOut)!=0 && (triOut)<= 3 && (triOut)!=1 || s1+s2<=s3)

76: P1 && (triOut)!=0 && (triOut)<= 3 && (triOut)!=1 || s1+s2<=s3)

&& (triOut)!=2 || s1+s3<=s2)

78: P1 && (triOut)!=0 && (triOut)<= 3 && (triOut)!=1 || s1+s2<=s3)

&& (triOut)!=2 || s1+s3 <= s2) && (triOut)!=3 || s2+s3 <= s1)

Need to solve for the
internal variable *triOut*

Introduction to Software Testing (Ch 3), www.introsoftwaretesting.com

9

Solving for Internal Variable *triOut*

At line 55, *triOut* has a value in the range (0 .. 6)

triOut = 0 s1!=s2 && s1!=s3 && s2!=s3

1 s1=s2 && s1!=s3 && s2!=s3

2 s1!=s2 && s1=s3 && s2!=s3

3 s1!=s2 && s1!=s3 && s2=s3

4 s1=s2 && s1!=s3 && s2=s3 — Contradiction

5 s1!=s2 && s1=s3 && s2=s3 — Contradiction

6 s1=s2 && s1=s3 && s2=s3

Introduction to Software Testing (Ch 3), www.introsoftwaretesting.com

© Ammann & Offutt

10

Reachability for Triang Predicates (solved for triOut – reduced)

42: True

49: P1 = s1>0 && s2>0 && s3>0

51: P1

53: P1

55: P1

59: P1 && s1 != s2 && s2 != s3 && s2 != s3 (triOut = 0)

62: P1 && s1 != s2 && s2 != s3 && s2 != s3 (triOut = 0)
&& (s1+s2 > s3) && (s2+s3 > s1) && (s1+s3 > s2)

70: P1 && P2 = (s1=s2 || s1=s3 || s2=s3) (triOut != 0)

72: P1 && P2 && P3 = (s1!=s2 || s1!=s3 || s2!=s3) (triOut <= 3)

74: P1 && P2 && P3 && (s1 != s2 || s1+s2<=s3)

76: P1 && P2 && P3 && (s1 != s2 || s1+s2<=s3)
&& (s1 != s3 || s1+s3<=s2)

78: P1 && P2 && P3 && (s1 != s2 || s1+s2<=s3)

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

Looks complicated, but
a lot of redundancy

Predicate Coverage

These values are
"don't care", needed
to complete the test.

	T				F			
	A	B	C	EO	A	B	C	EO
p42: (S1 <= 0 S2 <= 0 S3 <= 0)	0	0	0	4	1	1	1	3
p49: (S1 == S2)	1	1	1	3	1	2	2	2
p51: (S1 == S3)	1	1	1	3	1	2	2	2
p53: (S2 == S3)	1	1	1	3	2	1	2	2
p55: (triOut == 0)	1	2	3	4	1	1	1	3
p59: (S1+S2 <= S3 S2+S3 <= S1 S1+S3 <= S2)	1	2	3	4	2	3	4	1
p70: (triOut > 3)	1	1	1	3	2	2	3	2
p72: (triOut == 1 && S1+S2 > S3)	2	2	3	2	2	2	4	4
p74: (triOut == 2 && S1+S3 > S2)	2	3	2	2	2	4	2	4
p76: (triOut == 3 && S2+S3 > S1)	3	2	2	2	4	2	2	4

Clause Coverage

	T				F			
	A	B	C	EO	A	B	C	EO
p42: (S1 <= 0)	0	1	1	4	1	1	1	3
(S2 <= 0)	1	0	1	4	1	1	1	3
(S3 <= 0)	1	1	0	4	1	1	1	3
p59: (S1+S2 <= S3)	2	3	6	4	2	3	4	1
(S2+S3 <= S1)	6	2	3	4	2	3	4	1
(S1+S3 <= S2)	2	6	3	4	2	3	4	1
p72: (triOut == 1)	2	2	3	2	2	3	2	2
(S1+S2 > S3)	2	2	3	2	2	2	5	4
p74: (triOut == 2)	2	3	2	2	3	2	2	2
(S1+S3 > S2)	2	3	2	2	2	5	2	4
p76: (triOut == 3)	3	2	2	2	1	2	1	4
(S2+S3 > S1)	3	2	2	2	5	2	2	4

Correlated Active Clause Coverage

		A	B	C	EO
p42: (S1 <= 0 S2 <= 0 S3 <= 0)	T f f	0	1	1	4
	F f f	1	1	1	3
	f T f	1	0	1	4
	f f T	1	1	0	4
p59: (S1+S2 <= S3 S2+S3 <= S1 S1+S3 <= S2)	T f f	2	3	6	4
	F f f	2	3	4	1
	f T f	6	2	3	4
	f f T	2	6	3	4
p72: (triOut == 1 && S1+S2 > S3)	T t	2	2	3	2
	F t	2	3	3	2
	t F	2	2	5	4
p74: (triOut == 2 && S1+S3 > S2)	T t	2	3	2	2
	F t	2	3	3	2
	t F	2	5	2	4
p76: (triOut == 3 && S2+S3 > S1)	T t	3	2	2	2
	F t	1	2	2	4
	t F	5	2	2	4

At least one pair of sides must be equal.

s1=s2 && s1!=s3 && s2!=s3

s1!=s2 && s1=s3 && s2!=s3

s1!=s2 && s1!=s3 && s2=s3

Summary : Logic Coverage for Source Code

- **Predicates appear in decision statements – if, while, for, etc.**
- **Most predicates have less than four clauses**
- **The hard part of applying logic criteria to source is resolving the internal variables**
- **Non-local variables (class, global, etc.) are also input variables if they are used**
- **If an input variable is changed within a method, it is treated as an internal variable thereafter**