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)*

```
// Jeff Offutt -- Java version Feb 2003
// The old standby: classify triangles
// Figures 3.2 and 3.3 in the book.
import java.io.*;
class trityp
{
  private static String[] triTypes = { "",    // Ignore 0.
        "scalene", "isosceles", "equilateral", "not a valid
        triangle"};
  private static String instructions = "This is the ancient
TriTyp program.\nEnter three integers that represent the lengths
of the sides of a triangle.\nThe triangle will be categorized as
either scalene, isosceles, equilateral\nor invalid.\n";

public static void main (String[] argv)
{  // Driver program for trityp
   int A, B, C;
   int T;
```

Triang *(pg 2 of 5)*

```
    System.out.println (instructions);
    System.out.println ("Enter side 1: ");
    A = getN();
    System.out.println ("Enter side 2: ");
    B = getN();
    System.out.println ("Enter side 3: ");
    C = getN();
    T = Triang (A, B, C);

    System.out.println ("Result is: " + triTypes [T]);
}

// ==================================
```

Triang *(pg 3 of 5)*

```
// The main triangle classification method
private static int Triang (int Side1, int Side2, int Side3)
{
   int tri_out;

   // tri_out is output from the routine:
   //    Triang = 1 if triangle is scalene
   //    Triang = 2 if triangle is isosceles
   //    Triang = 3 if triangle is equilateral
   //    Triang = 4 if not a triangle

   // After a quick confirmation that it's a legal
   // triangle, detect any sides of equal length
   if (Side1 <= 0 || Side2 <= 0 || Side3 <= 0)
   {
      tri_out = 4;
      return (tri_out);
   }
```

Triang *(pg 4 of 5)*

```
tri_out = 0;
if (Side1 == Side2)
   tri_out = tri_out + 1;
if (Side1 == Side3)
   tri_out = tri_out + 2;
if (Side2 == Side3)
   tri_out = tri_out + 3;
if (tri_out == 0)
{  // Confirm it's a legal triangle before declaring
   // it to be scalene

   if (Side1+Side2 <= Side3 || Side2+Side3 <= Side1 ||
       Side1+Side3 <= Side2)
      tri_out = 4;
   else
      tri_out = 1;
   return (tri_out);
}
```

Triang *(pg 5 of 5)*

```
/* Confirm it's a legal triangle before declaring  */
/* it to be isosceles or equilateral  */

if (tri_out > 3)
   tri_out = 3;
else if (tri_out == 1 && Side1+Side2 > Side3)
  tri_out = 2;
else if (tri_out == 2 && Side1+Side3 > Side2)
  tri_out = 2;
else if (tri_out == 3 && Side2+Side3 > Side1)
  tri_out = 2;
else
  tri_out = 4;
return (tri_out);
} // 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*

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	

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
s1=s2 && s1!=s3 && s2!=s3	F t	2	3	3	2
	t F	2	2	5	4
p74: (triOut == 2 && S1+S3 > S2)	T t	2	3	2	2
s1!=s2 && s1=s3 && s2!=s3	F t	2	3	3	2
	t F	2	5	2	4
p76: (triOut == 3 && S2+S3 > S1)	T t	3	2	2	2
s1!=s2 && s1!=s3 && s2=s3	F t	1	2	2	4
	t F	5	2	2	4

At least one pair of sides must be equal.

Program Transformation Issues

```
if ((a && b) || c) {
    S1;
}
else {
    S2;
}
```

Transform (1)?

```
if (a) {
    if (b)
        S1;
    else {
        if (c)
            S1;
        else
            S2;
    }
}
else {
    if (c)
        S1;
    else
        S2;
}
```

Transform (2)?

```
d = a && b;
e = d || c;
if (e) {
    S1;
}
else {
    S2;
}
```

Problems with Transformed Programs

- Maintenance is certainly harder with Transform (1)
  - Not recommended!
- Coverage on Transform (1)
  - PC on transform does not imply CACC on original
  - CACC on original does not imply PC on transform
- Coverage on Transform (2)
  - Structure used by logic criteria is "lost"
  - Hence CACC on transform 2 only requires 3 tests
  - Note: Mutation analysis (Chapter 5) addresses this problem
- Bottom Line: Logic coverage criteria are there to help you!

a	b	c	$(a \wedge b) \vee c$	CACC	PC	CACC(2)
T	T	T	T		X	
T	T	F	T	X		X
T	F	T	T	X	X	X
T	F	F	F	X	X	
F	T	T	T		X	
F	T	F	F	X		X
F	F	T	T			
F	F	F	F		X	

Summary : Logic Coverage for Source Code

- **Predicates** appear in decision statements
  - if, while, for, etc.
- Most predicates have less than **four clauses**
  - But some applications have predicates with many 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
- To maximize effect of logic coverage criteria:
  - Avoid transformations that hide predicate structure