Introduction to Software Testing
Chapter 3.3
Logic Coverage for Source Code

Paul Ammann & Jeff Offutt

http://www.cs.gmu.edu/~offutt/softwaretest/
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 …
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. Enter three integers that represent the lengths
12     of the sides of a triangle. The triangle will be categorized as
13     either scalene, isosceles, equilateral or invalid."
14
15    public static void main (String[] argv)
16    {  // Driver program for trityp
17        int A, B, C;
18        int T;
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     // ===========================================================================
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);
    }
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    }
/* 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: \( P1 = s1>0 \land s2>0 \land s3>0 \)

51: \( P1 \)

53: \( P1 \)

55: \( P1 \)

59: \( P1 \land triOut = 0 \)

62: \( P1 \land triOut = 0 \land (s1+s2 > s3) \land (s2+s3 > s1) \land (s1+s3 > s2) \)

70: \( P1 \land triOut \neq 0 \)

72: \( P1 \land triOut \neq 0 \land triOut \leq 3 \)

74: \( P1 \land triOut \neq 0 \land triOut \leq 3 \land (triOut \neq 1 \lor s1+s2 \leq s3) \)

76: \( P1 \land triOut \neq 0 \land triOut \leq 3 \land (triOut \neq 1 \lor s1+s2 \leq s3) \land (triOut \neq 2 \lor s1+s3 \leq s2) \)

78: \( P1 \land triOut \neq 0 \land triOut \leq 3 \land (triOut \neq 1 \lor s1+s2 \leq s3) \land (triOut \neq 2 \lor s1+s3 \leq s2) \land (triOut \neq 3 \lor s2+s3 \leq s1) \)

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)

$$triOut = 0 \quad s1!=s2 \quad && \quad s1!=s3 \quad && \quad s2!=s3$$

1. $s1=s2 \quad && \quad s1!=s3 \quad && \quad s2!=s3$
2. $s1!=s2 \quad && \quad s1=s3 \quad && \quad s2!=s3$
3. $s1!=s2 \quad && \quad s1!=s3 \quad && \quad s2=s3$
4. $s1=s2 \quad && \quad s1!=s3 \quad && \quad s2=s3$
5. $s1!=s2 \quad && \quad s1=s3 \quad && \quad s2=s3$
6. $s1=s2 \quad && \quad s1=s3 \quad && \quad s2=s3$

Contradiction

Contradiction
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.

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Truth</th>
<th>T A</th>
<th>T B</th>
<th>T C</th>
<th>T EO</th>
<th>F A</th>
<th>F B</th>
<th>F C</th>
<th>F EO</th>
</tr>
</thead>
<tbody>
<tr>
<td>p42: (S1 &lt;= 0</td>
<td></td>
<td>S2 &lt;= 0</td>
<td></td>
<td>S3 &lt;= 0)</td>
<td></td>
<td>0 0 0 4</td>
<td>1 1 1 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p49: (S1 == S2)</td>
<td></td>
<td>1 1 1 3</td>
<td>1 2 2 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p51: (S1 == S3)</td>
<td></td>
<td>1 1 1 3</td>
<td>1 2 2 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p53: (S2 == S3)</td>
<td></td>
<td>1 1 1 3</td>
<td>2 1 2 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p55: (triOut == 0)</td>
<td></td>
<td>1 2 3 4</td>
<td>1 1 1 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p59: (S1+S2 &lt;= S3</td>
<td></td>
<td>S2+S3 &lt;= S1</td>
<td></td>
<td>S1+S3 &lt;= S2)</td>
<td></td>
<td>1 2 3 4</td>
<td>2 3 4 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p70: (triOut &gt; 3)</td>
<td></td>
<td>1 1 1 3</td>
<td>2 2 3 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p72: (triOut == 1 &amp;&amp; S1+S2 &gt; S3)</td>
<td></td>
<td>2 2 3 2</td>
<td>2 2 4 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p74: (triOut == 2 &amp;&amp; S1+S3 &gt; S2)</td>
<td></td>
<td>2 3 2 2</td>
<td>2 4 2 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p76: (triOut == 3 &amp;&amp; S2+S3 &gt; S1)</td>
<td></td>
<td>3 2 2 2</td>
<td>4 2 2 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Introduction to Software Testing (Ch 3)

© Ammann & Ofutt
<table>
<thead>
<tr>
<th>Clause Coverage</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>EO</th>
<th>T</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>EO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>p42: (S1 &lt;= 0)</strong></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>(S2 &lt;= 0)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>(S3 &lt;= 0)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td><strong>p59: (S1+S2 &lt;= S3)</strong></td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td></td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>(S2+S3 &lt;= S1)</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>(S1+S3 &lt;= S2)</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td></td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td><strong>p72: (triOut == 1)</strong></td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td></td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>(S1+S2 &gt; S3)</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td></td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td><strong>p74: (triOut == 2)</strong></td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>(S1+S3 &gt; S2)</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td><strong>p76: (triOut == 3)</strong></td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>(S2+S3 &gt; S1)</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
### Correlated Active Clause Coverage

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>EO</th>
</tr>
</thead>
<tbody>
<tr>
<td>p42: ((S1 &lt;= 0 \text{ or } S2 &lt;= 0 \text{ or } S3 &lt;= 0))</td>
<td>T</td>
<td>f</td>
<td>f</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>f</td>
<td>f</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>f</td>
<td>T</td>
<td>f</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>f</td>
<td>f</td>
<td>T</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>p59: ((S1+S2 &lt;= S3 \text{ or } S2+S3 &lt;= S1 \text{ or } S1+S3 &lt;= S2))</td>
<td>T</td>
<td>f</td>
<td>f</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>f</td>
<td>f</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>f</td>
<td>T</td>
<td>f</td>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>f</td>
<td>f</td>
<td>T</td>
<td>2</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>p72: ((\text{triOut} == 1 \text{ and } S1+S2 &gt; S3))</td>
<td>T</td>
<td>t</td>
<td></td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>t</td>
<td></td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>t</td>
<td>F</td>
<td></td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>p74: ((\text{triOut} == 2 \text{ and } S1+S3 &gt; S2))</td>
<td>T</td>
<td>t</td>
<td></td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>t</td>
<td></td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>t</td>
<td>F</td>
<td></td>
<td>2</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>p76: ((\text{triOut} == 3 \text{ and } S2+S3 &gt; S1))</td>
<td>T</td>
<td>t</td>
<td></td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>t</td>
<td></td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>t</td>
<td>F</td>
<td></td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

- At least one pair of sides must be equal.
Program Transformation Issues

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

Transform (1)?

Transform (2)?

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

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

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>((a \land b) \lor c)</th>
<th>CACC</th>
<th>PC</th>
<th>CACC(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
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