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 …
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 triangle"};
9     private static String instructions = "This is the ancient TriTyp program.
10     Enter three integers that represent the lengths of the sides of a triangle.
11     The triangle will be categorized as either scalene, isosceles, equilateral
12     nor invalid."
13
14     public static void main (String[] argv)
15     { // Driver program for trityp
16         int A, B, C;
17         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
28  // =======================================================
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  

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     }

© Ammann & Offutt
/* 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)

\[
triOut = 0 \quad s1!=s2 \quad && \quad s1!=s3 \quad && \quad s2!=s3 \\
1 \quad s1=s2 \quad && \quad s1!=s3 \quad && \quad s2!=s3 \\
2 \quad s1!=s2 \quad && \quad s1=s3 \quad && \quad s2!=s3 \\
3 \quad s1!=s2 \quad && \quad s1!=s3 \quad && \quad s2=s3 \\
4 \quad s1=s2 \quad && \quad s1!=s3 \quad && \quad s2=s3 \quad \text{--- Contradiction} \\
5 \quad s1!=s2 \quad && \quad s1=s3 \quad && \quad s2=s3 \quad \text{--- Contradiction} \\
6 \quad s1=s2 \quad && \quad s1=s3 \quad && \quad s2=s3
\]
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.

<table>
<thead>
<tr>
<th>Predicate</th>
<th>T</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>p42: ((S1 \leq 0 ; \text{or} ; S2 \leq 0 ; \text{or} ; S3 \leq 0))</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>p49: ((S1 = S2))</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>p51: ((S1 = S3))</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>p53: ((S2 = S3))</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>p55: ((\text{triOut} = 0))</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>p59: ((S1+S2 \leq S3 ; \text{or} ; S2+S3 \leq S1 ; \text{or} ; S1+S3 \leq S2))</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>p70: ((\text{triOut} &gt; 3))</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>p72: ((\text{triOut} = 1 ; \text{and} ; S1+S2 &gt; S3))</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>p74: ((\text{triOut} = 2 ; \text{and} ; S1+S3 &gt; S2))</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>p76: ((\text{triOut} = 3 ; \text{and} ; S2+S3 &gt; S1))</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Introduction to Software Testing (Ch 3), www.introsoftwaretesting.com © Ammann & Offutt
# Clause Coverage

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

| p42: (S1 <= 0 || S2 <= 0 || S3 <= 0) | A  | B  | C  | EO |
|-------------------------------------|----|----|----|----|
| 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) | A  | B  | C  | EO |
|---------------------------------------------------|----|----|----|----|
| 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  |

<table>
<thead>
<tr>
<th>p72: (triOut == 1 &amp;&amp; S1+S2 &gt; S3)</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>EO</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1=s2 &amp;&amp; s1!=s3 &amp;&amp; s2!=s3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T t</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>F t</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>t F</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>p74: (triOut == 2 &amp;&amp; S1+S3 &gt; S2)</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>EO</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1!=s2 &amp;&amp; s1=s3 &amp;&amp; s2!=s3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T t</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>F t</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>t F</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>p76: (triOut == 3 &amp;&amp; S2+S3 &gt; S1)</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>EO</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1!=s2 &amp;&amp; s1!=s3 &amp;&amp; s2=s3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T t</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>F t</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>t F</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

© Ammann & Offutt
Program Transformation Issues

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

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

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!
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