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.
12         Enter three integers that represent the lengths
13         of the sides of a triangle.
14         The triangle will be categorized as
15         either scalene, isosceles, equilateral or invalid.
16         ";
17
18     public static void main (String[] argv)
19     {  // Driver program for trityp
20         int A, B, C;
21         int T;
22         System.out.println (instructions);
23         System.out.println ("Enter side 1: ");
24         A = getN();
25         System.out.println ("Enter side 2: ");
26         B = getN();
27         System.out.println ("Enter side 3: ");
28         C = getN();
29         T = Triang (A, B, C);
30         System.out.println ("Result is: " + triTypes[T]);
31     }
32     // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

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     }
47
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 4 of 5)
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: \( P_1 = s_1>0 \land s_2>0 \land s_3>0 \)
51: \( P_1 \)
53: \( P_1 \)
55: \( P_1 \)
59: \( P_1 \land (\text{triOut}=0) \)
62: \( P_1 \land (\text{triOut}=0) \land (s_1+s_2 > s_3) \land (s_2+s_3 > s_1) \land (s_1+s_3 > s_2) \)
70: \( P_1 \land (\text{triOut}=0) \)
72: \( P_1 \land (\text{triOut}=0) \land (\text{triOut}=3) \)
74: \( P_1 \land (\text{triOut}=0) \land (\text{triOut}=3) \land (\text{triOut}=1) \land (s_1+s_2<=s_3) \)
76: \( P_1 \land (\text{triOut}=0) \land (\text{triOut}=3) \land (\text{triOut}=1) \land (s_1+s_2<=s_3) \land (\text{triOut}=2) \land (s_1+s_3<=s_2) \)
78: \( P_1 \land (\text{triOut}=0) \land (\text{triOut}=3) \land (\text{triOut}=1) \land (s_1+s_2<=s_3) \land (\text{triOut}=2) \land (s_1+s_3<=s_2) \land (\text{triOut}=3) \land (s_2+s_3 <= s_1) \)

Need to solve for the internal variable \( \text{triOut} \)

Solving for Internal Variable \( \text{triOut} \)

At line 55, \( \text{triOut} \) has a value in the range \((0 .. 6)\)

\[
\text{triOut} = 0 \quad s_1!=s_2 \land s_1!=s_3 \land s_2!=s_3
\]
\[
1 \quad s_1=s_2 \land s_1!=s_3 \land s_2!=s_3
\]
\[
2 \quad s_1!=s_2 \land s_1=s_3 \land s_2!=s_3
\]
\[
3 \quad s_1!=s_2 \land s_1!=s_3 \land s_2=s_3
\]
\[
4 \quad s_1=s_2 \land s_1!=s_3 \land s_2=s_3
\]
\[
5 \quad s_1!=s_2 \land s_1=s_3 \land s_2=s_3
\]
\[
6 \quad s_1=s_2 \land s_1=s_3 \land s_2=s_3
\]

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

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

### Correlated Active Clause Coverage

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>f</th>
<th>f</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>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>T</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>f</td>
<td>f</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>f</td>
<td>T</td>
<td>f</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>f</td>
<td>f</td>
<td>T</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4</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>T</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>f</td>
<td>f</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>f</td>
<td>T</td>
<td>f</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>f</td>
<td>f</td>
<td>T</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>p72: (triOut == 1 &amp;&amp; S1+S2 &gt; S3)</td>
<td>T</td>
<td>t</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>s1==s2 &amp;&amp; s1!=s3 &amp;&amp; s2!=s3</td>
<td>F</td>
<td>t</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>t</td>
<td>F</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>p74: (triOut == 2 &amp;&amp; S1+S3 &gt; S2)</td>
<td>T</td>
<td>t</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>s1==s2 &amp;&amp; s1==s3 &amp;&amp; s2!=s3</td>
<td>F</td>
<td>t</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>t</td>
<td>F</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>p76: (triOut == 3 &amp;&amp; S2+S3 &gt; S1)</td>
<td>T</td>
<td>t</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>s1!=s2 &amp;&amp; s1==s3 &amp;&amp; s2==s3</td>
<td>F</td>
<td>t</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>t</td>
<td>F</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>4</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

- 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