

Logic Coverage from Source Code

Moonzoo Kim
School of Computing
KAIST

*The original slides are taken from Chap. 8 of Intro. to SW Testing
2nd ed by Ammann and Offutt*



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 the triangle example in the following slides ...

```

30 private static int Triang (int s1, int s2, int s3)
31 {
32     int result;
33
34     // result is output from the routine:
35     // result = 1 if triangle is scalene
36     // result = 2 if triangle is isosceles
37     // result = 3 if triangle is equilateral
38     // result = 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 (s1 <= 0 || s2 <= 0 || s3 <= 0)
43     {
44         result = 4;
45         return (result);
46     }
47
48     result = 0;
49     if (s1 == s2)
50         result = result + 1;
51     if (s1 == s3)
52         result = result + 2;
53     if (s2 == s3)
54         result = result + 3;
55     if (result == 0)
56     { // Confirm it's a legal triangle before declaring
57         // it to be scalene
58
59         if (s1+s2<=s3 | |s2+s3 <= s1
60             | | s1+s3 <= s2)
61             result = 4;
62         else
63             result = 1;
64         return (result);
65     }
66
67     /* Confirm it's a legal triangle before declaring
68     it to be isosceles or equilateral */
69
70     if (result > 3)
71         result = 3;
72     else if (result == 1 && s1+s2 > s3)
73         result = 2;
74     else if (result == 2 && s1+s3 > s2)
75         result = 2;
76     else if (result == 3 && s2+s3 > s1)
77         result = 2;
78     else
79         result = 4;
80     return (result);
81 } // end Triang

```

Ten Triangle Predicates

42: (s1 <= 0 || s2 <= 0 || s3 <= 0)

49: (s1 == s2)

51: (s1 == s3)

53: (s2 == s3)

55: (result == 0)

59: (s1+s2 <= s3 || s2+s3 <= s1 ||
s1+s3 <= s2)

70: (result > 3)

72: (result == 1 && s1+s2 > s3)

74: (result == 2 && s1+s3 > s2)

76: (result == 3 && s2+s3 > s1)

Reachability for Triang Predicates

42: True

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

51: P1

53: P1

55: P1

59: P1 && result = 0

70: P1 && result != 0

72: P1 && result != 0 && result <= 3

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

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

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

Need to solve for the
internal variable *result*

Solving for Internal Variable *result*

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

result = 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**

5 **s1!=s2** && **s1=s3** && **s2=s3**

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

— **Contradiction**

— **Contradiction**

Reachability for Triang Predicates (solved for result – reduced)

42: True

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

51: P1

53: P1

55: P1

59: $P1 \ \&\& \ s1 \neq s2 \ \&\& \ s2 \neq s3 \ \&\& \ s2 \neq s3$

(result = 0)

70: $P1 \ \&\& \ P2 = (s1 = s2 \ || \ s1 = s3 \ || \ s2 = s3)$

(result != 0)

72: $P1 \ \&\& \ P2 \ \&\& \ P3 = (s1 \neq s2 \ || \ s1 \neq s3 \ || \ s2 \neq s3)$

(result <= 3)

74: $P1 \ \&\& \ P2 \ \&\& \ P3 \ \&\& \ (s1 \neq s2 \ || \ s1 + s2 \leq s3)$

76: $P1 \ \&\& \ P2 \ \&\& \ P3 \ \&\& \ (s1 \neq s2 \ || \ s1 + s2 \leq s3)$

$\ \&\& \ (s1 \neq s3 \ || \ s1 + s3 \leq s2)$

Looks complicated, but
a lot of redundancy

Predicate Coverage

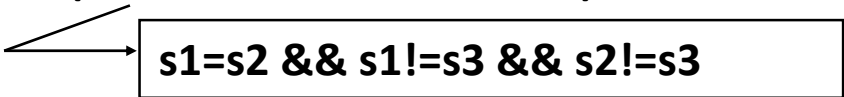
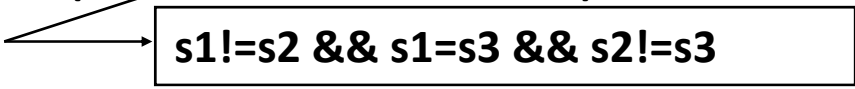
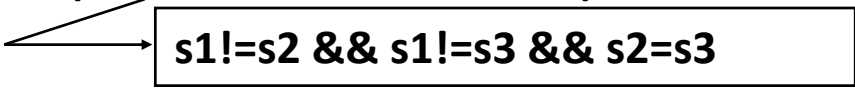
These values are
"don't care", needed
to complete the test.

	T			F		
	s1	s2	s3	s1	s2	s3
p42: (s1 <= 0 s2 <= 0 s3 <= 0)	0	0	0	1	1	1
p49: (s1 == s2)	1	1	1	1	2	2
p51: (s1 == s3)	1	1	1	1	2	2
p53: (s2 == s3)	1	1	1	2	1	2
p55: (result == 0)	1	2	3	1	1	1
p59: (s1+s2 <= s3 s2+s3 <= s1 s1+s3 <= s2)	1	2	3	2	3	4
p70: (result > 3)	1	1	1	2	2	3
p72: (result == 1 && s1+s2 > s3)	2	2	3	2	2	4
p74: (result == 2 && s1+s3 > s2)	2	3	2	2	4	2
p76: (result == 3 && s2+s3 > s1)	3	2	2	4	2	2

Clause Coverage

	T				F			
	S1	s2	s3	EO	s1	s2	s3	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: (result == 1)	2	2	3	2	2	3	2	2
(s1+s2 > s3)	2	2	3	2	2	2	5	4
p74: (result == 2)	2	3	2	2	3	2	2	2
(s1+s3 > s2)	2	3	2	2	2	5	2	4
p76: (result == 3)	3	2	2	2	1	2	1	4
(s2+s3 > s1)	3	2	2	2	5	2	2	4

CACC Coverage (also RACC)

	c1	c2	c3	P	s1	s2	s3	EO
p42: (s1 <= 0 s2 <= 0 s3 <= 0)	T	f	f	t	0	1	1	4
	F	F	F	f	1	1	1	3
	f	T	f	t	1	0	1	4
	f	f	T	t	1	1	0	4
p59: (s1+s2 <= s3 s2+s3 <= s1 s1+s3 <= s2)	T	f	f	t	2	3	6	4
	F	F	F	f	2	3	4	1
	f	T	f	t	6	2	3	4
	f	f	T	t	2	6	3	4
p72: (result == 1 && s1+s2 > s3) 	T	T		t	2	2	3	2
	F	t		f	2	3	3	2
	t	F		f	2	2	5	4
p74: (result == 2 && s1+s3 > s2) 	T	T		t	2	3	2	2
	F	t		f	2	3	3	2
	t	F		f	2	5	2	4
p76: (result == 3 && s2+s3 > s1) 	T	T		t	3	2	2	2
	F	t		f	1	2	2	4
	t	F		f	5	2	2	4

Program Transformation Issues

```
if ((a && b) || c) {  
    s1;  
}  
else {  
    s2;  
}
```


Transform (1)?

```
if (a) {  
    if (b)  
        s1;  
    else {  
        if (c) /* c1 */  
            s1;  
        else  
            s2;  
    }  
}  
else {  
    if (c) /* c2 */  
        s1;  
    else  
        s2;  
}
```


Transform (2)?

```
d = a && b;  
e = d || c;  
if (e) {  
    s1;  
}  
else {  
    s2;  
}
```

Problems with Transformed Programs (1/2)

- Maintenance is certainly harder with Transform (1)
 - Not recommended!
- Coverage on Transform (1)
 - PC on the transform does not imply CACC on original
 - A test suit to satisfy PC on the transform (1):
 - a: any element of $\{1,2,3,4\} \times \{5,6,7,8\}$
 - b: any element of $\{1,2\} \times \{3,4\}$
 - c1: $\{(3,4)\}$
 - c2: any element of $\{5,7\} \times \{6,8\}$
 - ex. $\{1,3,4,5,8\}$
 - CACC on the original does not imply PC on transform
 - Ex. $\{(2,6), (2,4), (3,4)\}$ does not satisfy PC on the transform due to c2

	a	b	c	$(a \wedge b) \vee c$	CACC	PC(1)
1	T	T	T	T		O
2	T	T	F	T	O	
3	T	F	T	T	O	O
4	T	F	F	F	O	O
5	F	T	T	T		O
6	F	T	F	F	O	
7	F	F	T	T		
8	F	F	F	F		O

$$(a \wedge b) \vee c$$

a as major clause: $p_a: b \wedge \neg c$ TR= $\{(2,6)\}$

b as major clause: $p_b: a \wedge \neg c$ TR= $\{(2,4)\}$

c as major clause: $p_c: \neg(a \wedge b)$ TR=
any element of $\{3,5,7\} \times \{4,6,8\}$

Problems with Transformed Programs (2/2)

- Coverage on Transform (2)
 - Structure used by logic criteria is “lost”
 - Hence CACC on the transform 2 only requires 3 tests
- Therefore, it may not be meaningful to transform a program to increase coverage

	a	b	d	c	$(a \wedge b) \vee c$	CACC	PC(1)	CACC(2)
1	T	T	T	T	T		O	
2	T	T	T	F	T	O		O
3	T	F	F	T	T	O	O	O
4	T	F	F	F	F	O	O	
5	F	T	F	T	T		O	
6	F	T	F	F	F	O		O
7	F	F	F	T	T			
8	F	F	F	F	F		O	

$d \parallel c$

d as major clause: $p_d: \neg c$ TR={2,4),(2,6),(2,8)}

c as major clause: $p_c: \neg d$ TR={3,5,7}x{4,6,8}

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