Logic Coverage

Lecture Objectives:

1) Define the terms predicate and clause.
2) Explain the concept of short circuit evaluation
3) Define the concept of predicate coverage and clause coverage.
4) Explain the concept of combinatorial coverage.
5) Define active clause coverage.
6) Apply active clause coverage to create test cases for an appropriate clause.
7) Explain inactive clause coverage.
8) Define the concept of infeasibility.
9) Explain the concpet of predicate transformation.
10) Use predicate transformation to expand the logic of a given program into simpler form.

A brief definition

- A clause ci in predicate p, called the major clause, determines p if and only if the values of the remaining minor clauses cj are such that changing ci changes the value of p

```
uint8_t x;
uint8_t y;
If ((x>=0) || (y>100))
{


}
```

Another clause example

```
uint8_t x;
uint8_t y;
uint8_t z;
if((x>0) || ((y>100) && (z%2==0)))
{

}
```


Predicates

- Where do predicates come from in code?

Branches / decision statements

- What happens to active clause coverage if a predicate has only one clause?

predicate coverage

```
if ( x > 5 )
{
    A();
}
else
{
    B();
}
```

Predicate Coverage
⇒ Every Predicate is true and false

Clause coverage:
⇒ Every clause is true and false.

. . .

Why is logic coverage hard?

- 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
- Internal Variables
  - Variables in the predicates that are not inputs to the program are called internal variables

- The triangle problem
  - With your neighbor, look through the code and tell me where all of the predicates are at in the code by line number.

Handout code

42
49
51
53
55
57
70
72
74
76

Predicates and Reachability

Line Number	Reachability
42	TRUE
49	P1 => side1>0, side2>0, side3>0
51	P1
53	P1
55	P2 => side1 != side2 != side3 != side1
59	P2 False
62	Conditions
70	
72	For example
74	
76	
78	

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

what needs to be true?

Predicate Coverage

- Figure out a set of values to make each predicate be true or false

Predicates and Reachability

	True				False				
Line	A	B	C	EO	A	B	C	EO	
42	0	0	0	4	4	4	5	2	
	4	4	5	2	3	4	5	1	
49	4	5	4	2	3	4	5	1	
51	5	4	4	2	3	4	5	1	
53	3	4	5	1	3	3	3	3	
55	1	1	3	4	2	3	4	1	
59	1	1	1	3	2	2	3	2	
62									
70									
72									
74									
76									
78									

Eo expected output of entire method.

Continue to fill the whole method.

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

	True				False			
Line	A	B	C	EO	A	B	C	EO
42								
59								
72								
74								
76								

Clause Coverage

Lots more testing

	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

Predicate Transformation

- So the answer to all of this is to eliminate multiclause predicates?

Yeah!

Example Code

```
if ((x > 5) && (y>5))
{
      Printf("Both are greater than 5.");
}
else
{
      Printf("Both are not greater than 5.");
}
```

Bad... Multiple predicate.

Predicate transformation

- Change each multi claused predicate into a single predicate
- Add code as is necessary.

Example Code

```
if (x > 5)
{
  if (y>5)
  {
      Printf("Both are greater than 5.");
  }
  else
  {
      Printf("Both are not greater than 5.");
  }
}
else
{
      Printf("Both are not greater than 5.");
}
```

if((x>5)&&(y>5))
{ printf("Both greater than 5");
}
else{ both not greater than 5");

original is much simpler.

Duplicated code...

BAD

7 clauses in a predicate

if

if

if

if

if

if

⋮

Predicate Transformation

- So the answer to all of this is to eliminate multiclause predicates?