

CS 130 SOFTWARE ENGINEERING

# TESTING BOUNDED ITERATION INFEASIBLE PATHS

Professor Miryung Kim  
UCLA Computer Science  
Based on Materials from Miryung Kim

# AGENDA

- ▶ The number of paths and Infeasible paths
- ▶ Symbolic execution Test Generation
- ▶ Regression test selection

# MORE SPECIFICALLY

- ▶ Counting the number of loop iterations for bounded programs
- ▶ Identify infeasible paths through symbolic execution

# ESTIMATING THE NUMBER OF PATHS

# ESTIMATING THE NUMBER OF PATHS

- ▶ For a loop-free program with  $k$  branches, the number of paths is  $2^k$  if no infeasible path exists.
- ▶ How about a program with loops?
- ▶ In this exercise, we perform loop unrolling then the number of paths is  $2^{(\# \text{ branches})}$  when each branch can be executed both true or false.

```
public static int fun1(int N) {  
    int sum = 0;  
    for (int i = 1; i <= N; i++) {  
        for (int j = 1; j <= Math.pow(3, i); j+  
+) {  
            System.out.println("HelloWorld");  
            if (new Random().nextInt() % 2 == 0)  
                sum++;  
        }  
    }  
    return sum;  
}
```

# EXAMPLE 1. ESTIMATING LOOP ITERATIONS

- ▶ 1. What is the number of times that “HelloWorld” will be printed?
  - ▶ 2. What is the number of branch executions in the loop-unrolled program?
  - ▶ 3. What is the number of paths?
- 
- ▶ *The answers is the same as the Example 2’s answer shown in the later slides.*

## EXAMPLE 2:

```
public static void fun2(int N) {  
    int sum = 0;  
    Random rand = new Random();  
    for (int i = 1; i <= N; i++) {  
        for (int j = 1; j <= Math.pow(3, i); j++) {  
            if (rand.nextInt() % 2 == 0)  
                sum++;  
        }  
        for (int k = 1; k <= i; k++) {  
            if (Math.pow(2, k) % 2 == 0) sum++;  
        }  
    }  
    System.out.println("N:" + N + "\tSum:" + sum);  
}
```

## EXAMPLE 2

- ▶ 1.What is the number of branch executions in the loop-unrolled program?
- ▶ 2.What is the number of paths?

# ABOUT LOOP J'S ITERATION

When  $i$  is  $1$ , the loop  $j$  executes  $3^1$ .

When  $i$  is  $2$ , the loop  $j$  executes  $3^2$ .

...

When  $i$  is  $N$ , the loop  $j$  executes  $3^N$ .

$$T_1(N) = 3^1 + 3^2 + \dots + 3^N$$

$$3T_1(N) = 3^2 + \dots + 3^N + 3^{(N+1)}$$

Subtract the first from the second

$$2T_1(N) = 3^{(N+1)} - 3$$

$$T_1(N) = (3^{(N+1)} - 3)/2$$

# ABOUT LOOP K'S ITERATION

When i is 1, the loop j executes 1.

When i is 2, the loop j executes 2.

...

When i is N, the loop j executes N.

$$T_1(N) = 1 + 2 + \dots + N$$

$$T_1(N) = N + N - 1 + \dots + 1$$

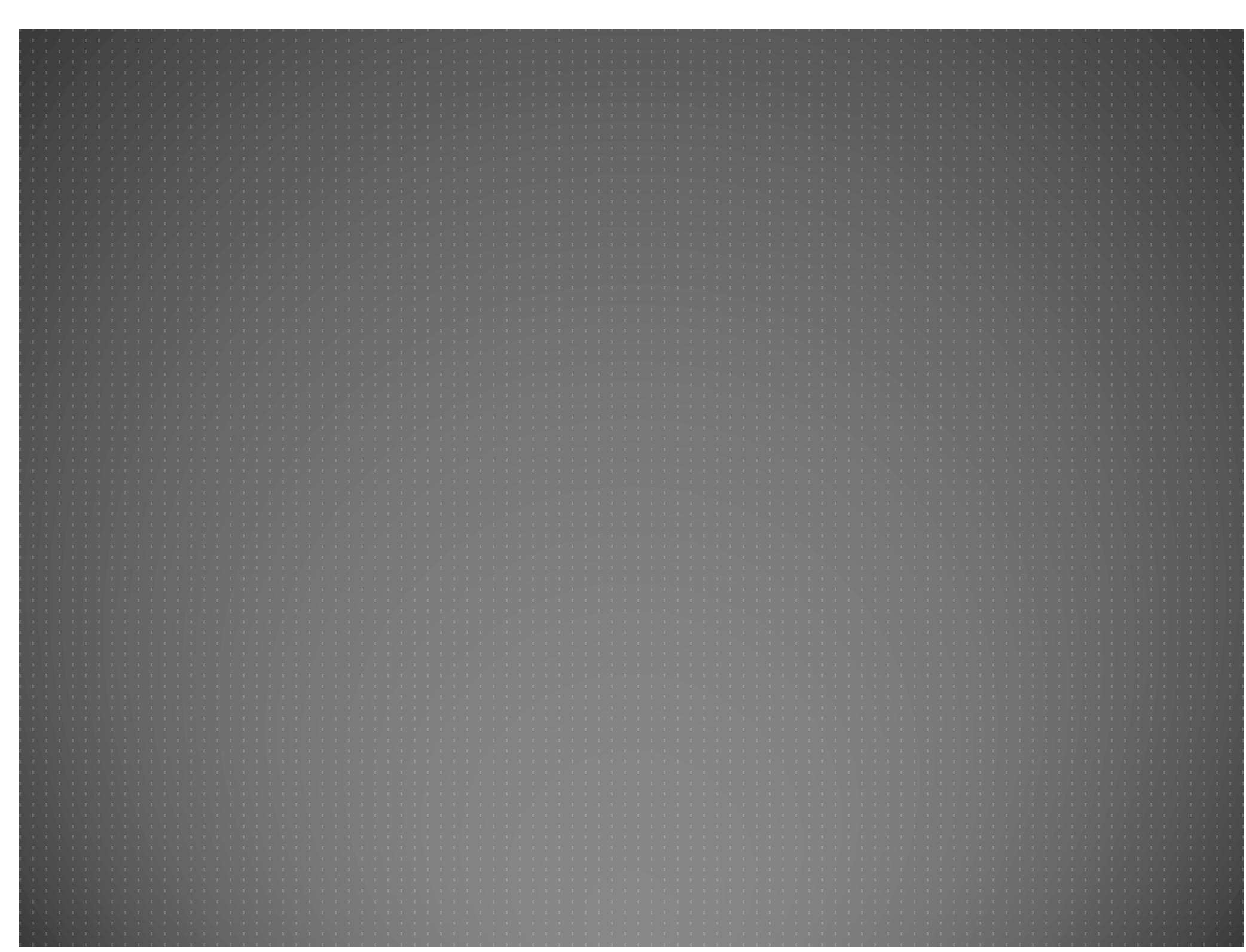
$$2T_1(N) = (N+1) * N // (N+1) appears N times$$

$$T_1(N) = N(N+1)/2$$

- ▶ `rand.nextInt()%2==0` can evaluate true or false randomly.
- ▶ `Math.pow(2,k)%2==0` is always true because  $(2^k)\%2$  is always 0. So this branch is deterministic regardless of inputs provided to the program.
- ▶ The total number of paths is therefore  $2^{\{(3^{(N+1)} - 3)/2\}}$ .

# EXTRA POINT

- ▶ Extra point: what if the second branch was “`rand.nextInt()%2==0`” not “`Math.pow(2,k)%2==0`”
- ▶ If so, the total number of branch executions from loop J and K:  $\{(3^{(N+1)} - 3)/2 + N(N+1)/2\}$
- ▶ The total number of paths  $2^{\{(3^{(N+1)} - 3)/2 + N(N+1)/2\}}$



# PRACTICE QUESTION

```
sum = 0;
for (int i = 0; i < N; i++ ){
    for (int j = 1; j <= pow(2,i); j++){
        if (a[i]<k)
            sum++;
    }
    for (int m=1; m<=i; k++) {
        if (pow(2,m) % 2 == 1) { sum++;} // false
        else {sum--;}
    }
}
```

The second branch always evaluates to false for every possible input. So the branch  $(\text{pow}(2,m)\%2$  does not contribute to increasing the number of path).

For the first branch

When  $i=0$ , the inner j loop iterates  $2^0$

When  $i=1$ , the inner j loop iterates  $2^1$

...

When  $i=n$ , the inner j loop iterates  $2^n$ ,

Let's call  $T(n)$  the total number of times that the inner loop j iterates.

$$T(n) = 2^0 + 2^1 + \dots + 2^n$$

Multiply  $T(n)$  by 2 on both sides of the equation

$$2T(n) = 2^1 + 2^2 + \dots + 2^{n+1}$$

Subtract the first equation from the second,

$$T(n) = 2^{n+1} - 2^0$$

$$T(n) = 2^{n+1} - 1$$

The branch  $(a[i] < k)$  could evaluate to TRUE for some input and FALSE for some other input, however it evaluates to the same value within each  $i$ .

So the total number of paths is  $2^n$ , not  $2^{(2^{n+1}-1)}$ .

# INFEASIBLE PATH

- ▶ Paths that cannot be executed under any inputs are called “infeasible paths” commonly called as, dead code.
- ▶ We are going to teach a symbolic execution to model individual paths in terms of **logical constraints**.

- ▶ In other words, for a given branch  $b$ , we are examining whether there exists an input that evaluates the branch to be true and whether there exists another input that evaluates the branch to be false.
- ▶ This program has only one path, because the branch execution is always FALSE.

```
int x=0;  
if (x>1) x:=x+1 else x:=x-1;
```

Consider

```
if (x>1) x:=x+1 else x:=x-1;
```

# SYMBOLIC EXECUTION

# SYMBOLIC EXECUTION

- ▶ Use fresh variables in the beginning
- ▶ Use fresh variables after the state updates
- ▶ Loop must be unrolled first
- ▶ For each branch, we propagate the constraints for both true and false evaluation of the branch
- ▶ Basically, we are conservatively estimate the effect of taking either path.

# PATH CONDITION AND EFFECT

- ▶ For each path, the condition to exercise the path is called a “path condition.”
- ▶ The effect of executing statements along a possible path is called “effect”

# SIMPLE EXAMPLE

```
if (x>1) {  
    x:= x-2; //stmt 1  
}else {  
    x:= 2*x; //stmt 2  
}  
// stmt 3
```

Path condition for stmt1 is  $x > 1$

Path condition for stmt2 is  $x \leq 1$

Effect for stmt1 ( $x > 1$ ) AND ( $x' = x - 2$ )

Effect for stmt2 ( $x \leq 1$ ) AND ( $x' = 2x$ )

The overall symbolic execution result at  
stmt3

$((x > 1) \text{ AND } (x' = x - 2)) \text{ OR } ((x \leq 1) \text{ AND } (x' = 2x))$

# INFEASIBLE PATH EXAMPLE I.

```
public static int infeasiblepath(int x) {  
    if (x < 4) {  
        return 0;  
    }  
    int value = 0;  
    int y = 3 * x + 1;  
    if (x * x > y) {  
        value = value + 1;  
    } else {  
        value = value - 1;  
    }  
    return value;
```

# INFEASIBLE PATH EXAMPLE I.

- ▶ 1. Draw a control flow graph for the program.
- ▶ 2. What is the number of paths?
- ▶ 3. Use symbolic execution to model each path in terms of logical constraints.

- ▶ There is one path that returns immediate when  $x < 4$ .
- ▶  $x*x > y$  cannot go through "else" side because when  $x \geq 4$ ,  $x*x > 3x + 1$  is always true.
- ▶ The else side is dead code.

- ▶ Going back to the concept of the number of feasible paths, the paths is  $2^{\text{(the number of branch executions that could evaluate to both TRUE for some input or FALSE for some other input)}}$ .
- ▶ In other words, for a given branch  $b$ , we are examining whether there exists an input that evaluates the branch to be true and whether there exists another input that evaluates the branch to be false.

# PRACTICE ON INFEASIBLE PATH

```
public int pathConstraints(int x){  
    int value = 0;  
    int y=x*x;  
    if(x>3) x = x + 1; else x = abs(x)  
; // assume that abs(x) computes the  
absolute value of a.  
    if(2*x-1>y) y= y * 2; else y = y/  
2;  
    System.out.print("130 is a  
software engineering course.");
```

# SOLUTION

Path1 TT:  $x > 3 \text{ AND } 2(x+1) - 1 > x^2$   
 $\Rightarrow x > 3 \text{ AND } 2x + 1 > x^2 \text{ (NOT FEASIBLE)}$

Path2 TF:  $x > 3 \text{ AND } 2(x+1) - 1 \leq x^2$   
 $x > 3 \text{ AND } 2x - 1 \leq x^2 \text{ (FEASIBLE, for example when } x=4)$

Path3 FT:  $x \leq 3 \text{ AND } 2(\text{abs}(x)) - 1 > x^2$   
 $x \leq 3 \text{ AND } (x \geq 0 \text{ AND } 2x - 1 > x^2 \text{ OR } (x < 0 \text{ AND } -2x - 1 > x^2))$   
 $\Rightarrow 0 \leq x \leq 3 \text{ AND } 2x - 1 > x^2 \text{ OR } (x < 0 \text{ AND } 0 > x^2 + 2x + 1) \text{ NOT FEASIBLE}$

Path4 FF:  $x \leq 3 \text{ AND } 2\text{abs}(x) - 1 \leq x^2$   
 $x \leq 3 \text{ AND } (x \geq 0 \text{ AND } 2x - 1 \leq x^2 \text{ OR } (x < 0 \text{ AND } -2x - 1 \leq x^2))$   
 $\text{(FEASIBLE, for example } x=3, 2x - 1 \leq x^2, 5 \leq 9)$

## ► $\text{abs}(x)$

- $x \geq 0$  AND  $\text{abs}(x) = x$
- $x < 0$  AND  $\text{abs}(x) = -x$

# TEST INPUT GENERATION

# HOW TO GENERATE TEST INPUTS?

- ▶ You can use symbolic execution to generate concrete inputs
- ▶ If you want to exercise a particular path, first determine a path condition.
- ▶ Find concrete input assignments for each path

# SIMPLE EXAMPLE

```
if (x>1) {  
    x= x-2; //stmt 1  
}else {  
    x= 2x; //stmt 2  
}  
// stmt 3
```

Path condition for stmt1 is  $x > 1$

Path condition for stmt2 is  $x \leq 1$

Effect for stmt1 ( $x > 1$ ) AND ( $x' = x - 2$ )

Effect for stmt2 ( $x \leq 1$ ) AND ( $x' = 2x$ )

The overall symbolic execution result at stmt3

$((x > 1) \text{ AND } (x' = x - 2)) \text{ OR } ((x \leq 1) \text{ AND } (x' = 2x))$

⇒ The test input you need is any  $x$  where  $x > 1$  and any  $x$  where  $x \leq 1$ .

⇒ Concretely  $x = 2$  exercises stmt 1 and  $x = 1$  exercises stmt 2

# TEST GENERATION: #1

```
public static int generate_tests_for_this1(int x,  
int y, int z) {  
    if (x < y) {  
        z++;  
    } else {  
        z--;  
    }  
    if (z < 2 * x + 5) {  
        x++;  
    } else {  
        y++;  
    }  
    return y;  
}
```

# TEST GENERATION: #1

- ▶ 1. Draw a control flow graph for the program.
- ▶ 2. Use symbolic execution to model each path in terms of logical constraints.
- ▶ 3. Find concrete input assignments for each path condition

- ▶      if ( $x < y$ )  $z++$ ; else  $z--$ ;
- ▶      if ( $z < 2*x+5$ )  $x++$ ; else  $y++$ ;
- ▶ Path Conditions
- ▶ TT:  $x < y$  AND  $z+1 < 2x+5$
- ▶ TF:  $x < y$  AND  $z+1 \geq 2x+5$
- ▶ FT:  $x \geq y$  AND  $z-1 < 2x+5$
- ▶ FF:  $x \geq y$  AND  $z-1 \geq 2x+5$
  
- ▶ See the attached notes for path conditions and corresponding effects.

# TEST GENERATION #2

```
public static int generate_tests_for_this2(int x,
int y, int z) {
    // a bit time consuming, so we will do this
    question if we have a time,
    // otherwise complete it at home.
    for (int i = 0; i < 2; i++) {
        if (x < y) {
            z++;
        } else {
            z--;
        }
        if (z < 2 * x + 5) {
            x++;
        } else {
            y++;
        }
    }
    return y;
}
```

- Q2. The same program with a finite loop

```
► for (int i=0; i<2; i++) {  
    ►     if (x<y) z++; else z--;  
    ►     if (z<2*x+5) x++; else y++;  
    ► }
```

- Loop Unrolling

```
► B1: if (x<y) z++; else z--;  
► B2: if (z<2*x+5) x++; else y++;  
► B3: if (x<y) z++; else z--;  
► B4: if (z<2*x+5) x++; else y++;
```

- 4 branch executions => 16 paths

- ▶ Path Conditions for each path
- ▶ TTTT:  $x < y \text{ AND } z+1 < 2x+5 \text{ AND } x+1 < y \text{ AND } z+2 < 2(x+1)+5$
- ▶ TTTF:  $x < y \text{ AND } z+1 < 2x+5 \text{ AND } x+1 < y \text{ AND } z+2 \geq 2(x+1)+5$
- ▶ TTFT:  $x < y \text{ AND } z+1 < 2x+5 \text{ AND } x+1 \geq y \text{ AND } z < 2(x+1)+5$
- ▶ TFFF:  $x < y \text{ AND } z+1 < 2x+5 \text{ AND } x+1 \geq y \text{ AND } z \geq 2(x+1)+5$
- ▶ TFTT:  $x < y \text{ AND } z+1 \geq 2x+5 \text{ AND } x < y+1 \text{ AND } z+2 < 2(x+1)+5$
- ▶ TFTF: ... (Continue in a similar manner. Note that later branch conditions are affected by prior assignments. This case is for demonstration purposes and the exam question will not ask you to list 16 different path conditions tediously, but may require understanding the concept of path conditions.)
- ▶ TFFT:
- ▶ TFFF:
- ▶ FTTT:
- ▶ FTTF:
- ▶ FTFT:
- ▶ FTFF:
- ▶ FFTT:
- ▶ FFTF:
- ▶ FFFT:
- ▶ FFFF:

# EQUIVALENCE PARTITIONING

- ▶ A good test case covers a large part of the possible input data.
- ▶ The concept of “equivalence partitioning” is a formalization of this idea and helps reduce the number of test cases required.

# RECAP.

- ▶ We have studied three different coverage metrics.
- ▶ We have studied how to count the number of paths when the number of loops is bounded.

# PREVIEW

- We will discuss regression test selection, prioritization, and augmentation methods.

# QUESTIONS?