Control Flow Testing: Part A

Dr Fani Deligianni,

Fani.Deligianni@glasgow.ac.uk

https://www.gla.ac.uk/schools/computing/staff/fanideligianni/

Learning Objectives

- Understand the main characteristics of control flow testing.
- Discuss the most commonly used techniques of control flow testing
- Understand the main limitations of control flow testing:
  - When to use it
  - How to use it
  - What to expect

White-Box Testing

Fundamental Assumptions:

- Executing a faulty statement is a necessary condition for revealing a fault

White-Box Testing

Characteristics:

- Based on program code
  - Can be measured objectively
  - Can be measured automatically
- Can be used to compare test suites
- Allows for covering the coded behaviour in software

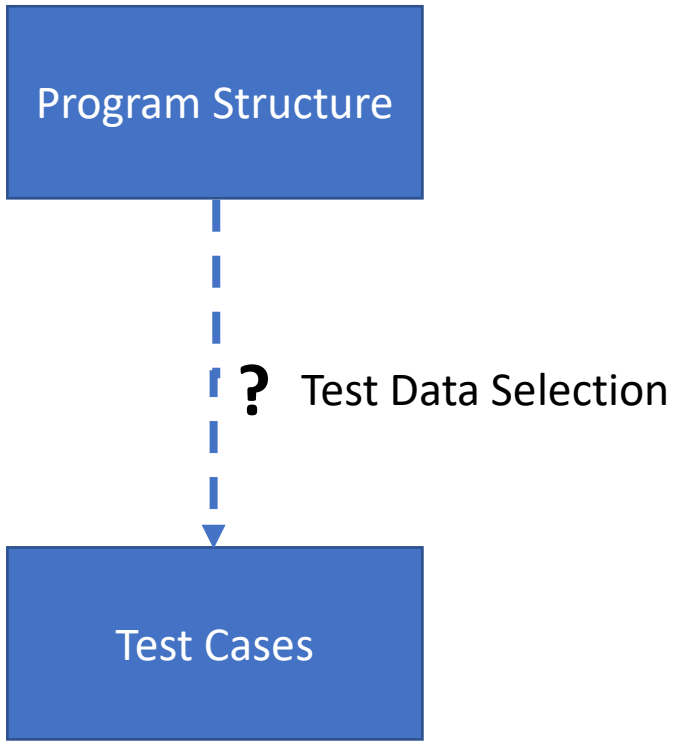
White-Box Testing

What Black-Box cannot test:

- It takes long time to check memory leaks
- Which portion of if condition is evaluated
- What path the program took to achieve the end result
- Is there any dead and unused code
- Is there any extra code that is not needed
- What are the potential breaking points in code
- Is the code compliant to best practices

White-Box Testing

- From Program Structure to Test cases

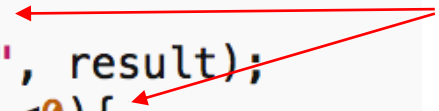


Different Techniques:
- Control-flow
- Data-flow
- Fault based

White-Box Testing

- Example

```
1  public void printSum(int a, int b){
2      int result = a+b;
3      if(result >0){
4          print("red", result);
5      else if(result <0){
6          print("blue, result");
7  }
```



We may decide to test these two program decisions

Coverage Criteria

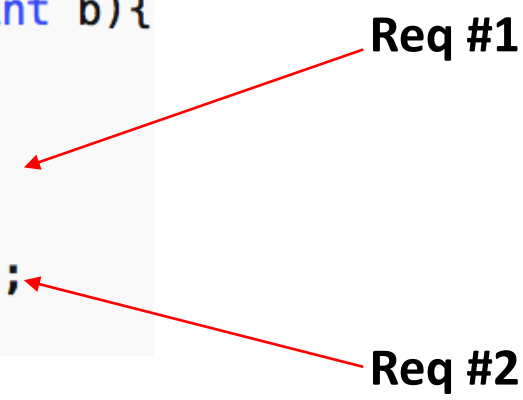
- Defined in terms of **test requirements**
  - Test requirements are the elements or entities in the code that we need to exercise in order to satisfy the criteria.

- Results in a set of **test specifications**
  - These are descriptions/specifications of the test required to satisfy the test requirement.
- The outcome is a set of **test cases** which are instantiations of the test specifications

printSum Test Requirement

- Example

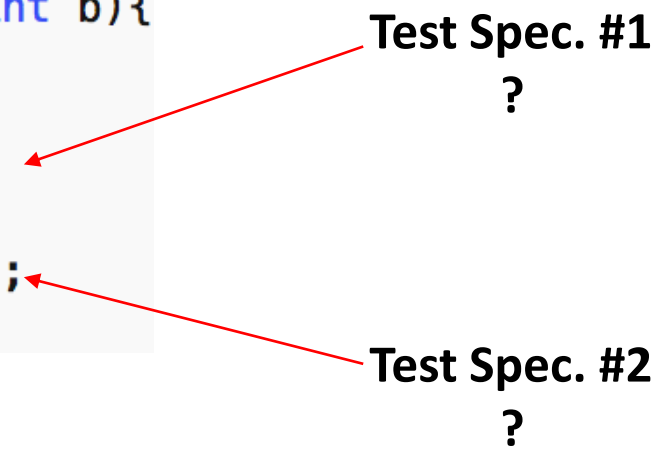
```
1  public void printSum(int a, int b){
2      int result = a+b;
3      if(result >0){
4          print("red", result);
5      else if(result <0){
6          print("blue, result");
7  }
```



printSum Test Specification

- Exercise

```
1  public void printSum(int a, int b){
2      int result = a+b;
3      if(result >0){
4          print("red", result);
5      else if(result <0){
6          print("blue, result");
7  }
```



Question

- What are the possible test specification that will satisfy Req #1 and #2?
  (This should be expressed in terms of constraints on the input)

printSum Test Specification

- Exercise

Answer

```
1  public void printSum(int a, int b){
2      int result = a+b;
3      if(result >0){
4          print("red", result);
5      else if(result <0){
6          print("blue, result");
7  }
```

Test Spec. #1
a + b >0

Test Spec. #2
a + b <0

printSum Test Cases

- Exercise

```
1  public void printSum(int a, int b){
2      int result = a+b;
3      if(result >0){
4          print("red", result);
5      else if(result <0){
6          print("blue, result");
7  }
```

Test Spec. #1
a + b >0

Test Spec. #2
a + b <0

Question

- Write test cases that will implement the test specifications in the format:

#1((a=[ ], b=[ ]), (outputColor=[ ], outputValue=[ ]))
#2((a=[ ], b=[ ]), (outputColor=[ ], outputValue=[ ]))

printSum Test Cases

- Exercise

```
1  public void printSum(int a, int b){
2      int result = a+b;
3      if(result >0){
4          print("red", result);
5      else if(result <0){
6          print("blue, result");
7  }
```

Test Spec. #1
a + b >0

Test Spec. #2
a + b <0

Answer

There are multiple answers (e.g)

#1 ((a=[3], b=[8]), (outputColor=[red], outputValue=[12]))
#2 ((a=[-5], b=[-8]), (outputColor=[blue], outputValue=[-13]))

Summary

- Control flow testing is a white-box approach
- Control flow testing is based on a structural strategy
- Control flow common in initial development cycles
- Control flow can identify bad software practices
- Control flow requires access to the source code
- In control flow testing coverage is the key criterion to define test requirements, specifications and cases

Control Flow Testing: Part B

Dr Fani Deligianni,

Fani.Deligianni@glasgow.ac.uk

https://www.gla.ac.uk/schools/computing/staff/fanideligianni/

Coverage Criteria

- Test coverage attempts to address questions about when to stop testing, or the amount of testing that is enough for a given program.
- Ideal testing is to explore exhaustively the entire test domain, which in general (and in practice) is impossible.

Statement Coverage

- Characterised by two aspects:

(1) Test Requirements

All the statements in the program

(2) Coverage Measure

$$\frac{\text{Number of executed test statements}}{\text{Total number of statements}}$$

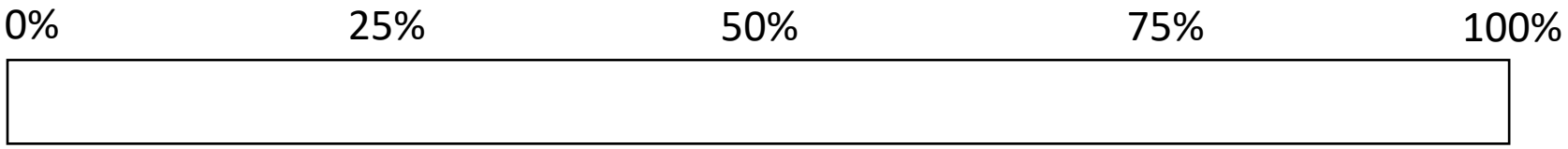
printSum Statement Coverage

- Example

```
1  public void printSum(int a, int b){
2      int result = a+b;
3      if(result >0){
4          print("red", result);
5      else if(result <0){
6          print("blue, result");
7  }
```

TC. #1
a = 3
b = 9

TC. #2
a = -5
b = -8



printSum Statement Coverage

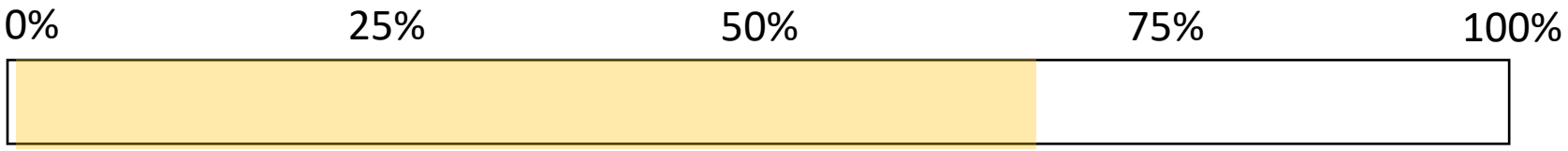
- Example

```
1  public void printSum(int a, int b){
2      int result = a+b;
3      if(result >0){
4          print("red", result);
5      else if(result <0){
6          print("blue, result");
7  }
```

TC. #1
a = 3
b = 9

TC. #2
a = -5
b = -8

Statement coverage = 5/7



printSum Statement Coverage

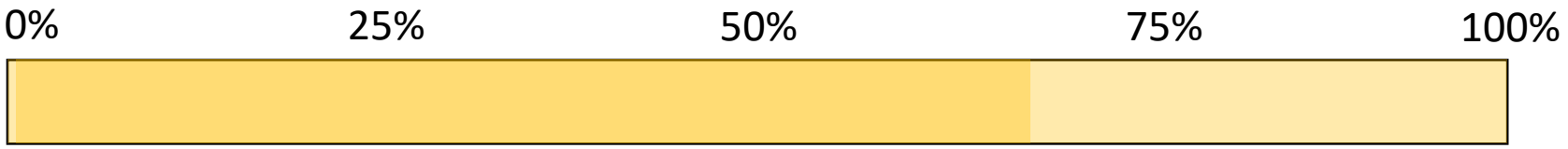
- Example

```
1  public void printSum(int a, int b){
2      int result = a+b;
3      if(result >0){
4          print("red", result);
5      else if(result <0){
6          print("blue, result");
7  }
```

TC. #1
a = 3
b = 9

TC. #2
a = -5
b = -8

Statement coverage = 7/7



Statement Coverage In practice

- Most used in the industry
- Typical coverage target is 80-90%

Why don’t we aim for 100% ?

Control Flow Graphs

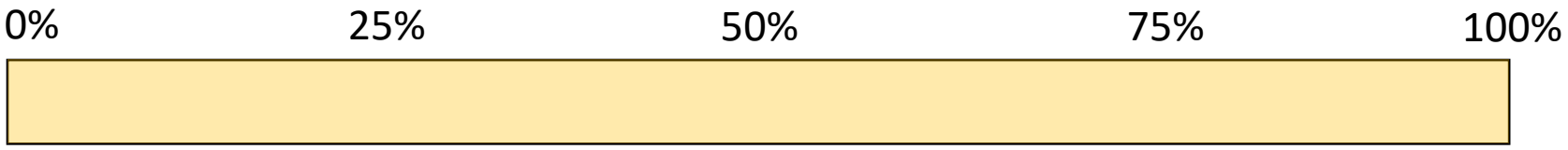
- Example

```
1  public void printSum(int a, int b){
2      int result = a+b;
3      if(result >0){
4          print("red", result);
5      else if(result <0){
6          print("blue, result");
7      else [donothing]
8  }
```

TC. #1
a = 3
b = 9

TC. #2
a = -5
b = -8

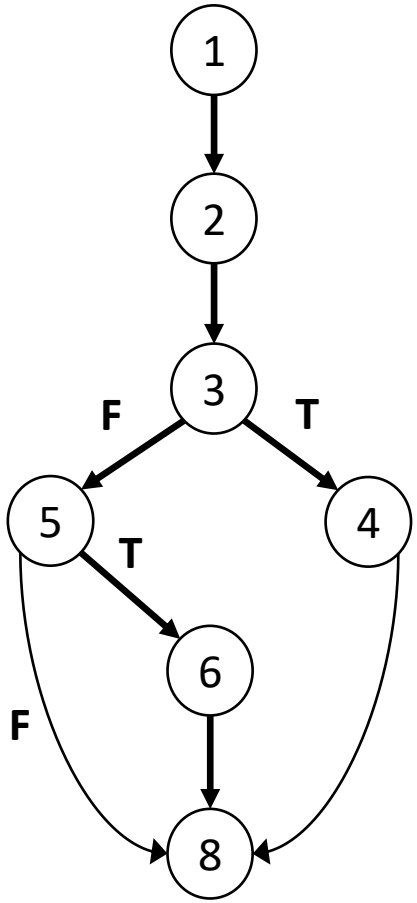
Statement coverage = 7/7



Control Flow Graphs

- Example

```
1  public void printSum(int a, int b){
2      int result = a+b;
3      if(result >0){
4          print("red", result);
5      else if(result <0){
6          print("blue, result");
7      else [donothing]
8  }
```



Branch Coverage

- Characterised by two aspects:

(1) Test Requirements

All branches in the program

(2) Coverage Measure

$$\frac{\text{Number of executed test branches}}{\text{Total number of branches}}$$

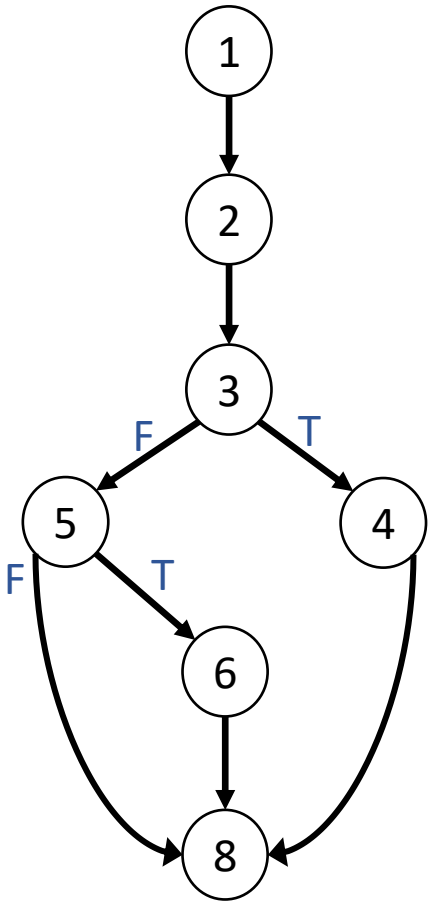
Branch Coverage

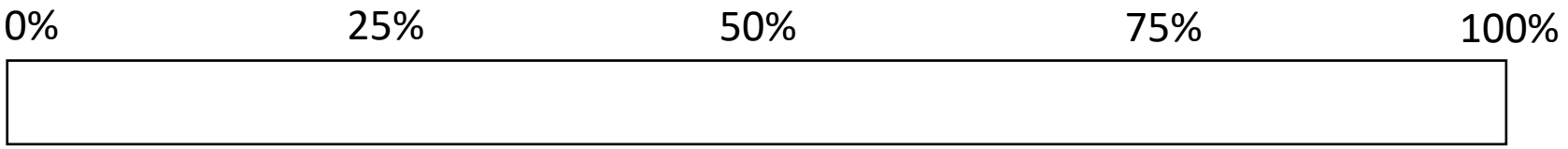
- Example

TC. #1
a = 3
b = 9

TC. #2
a = -5
b = -8

```
1  public void printSum(int a, int b){
2      int result = a+b;
3      if(result >0){
4          print("red", result);
5      else if(result <0){
6          print("blue, result");
7      else [donothing]
8  }
```





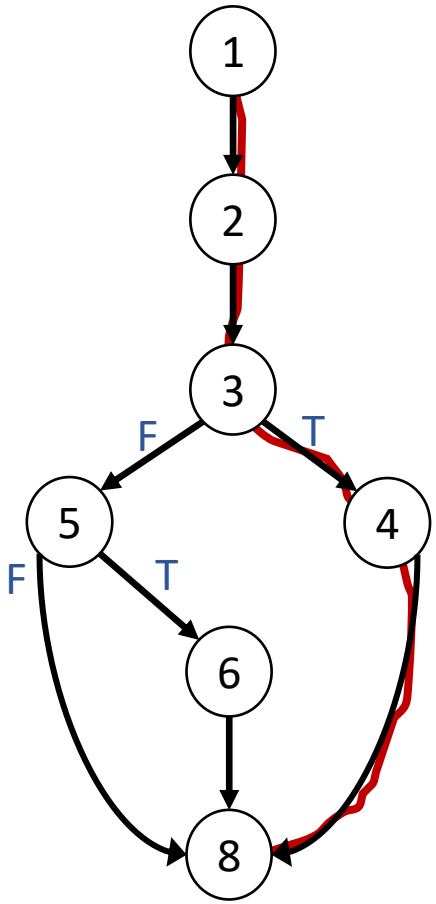
Branch Coverage

- Example

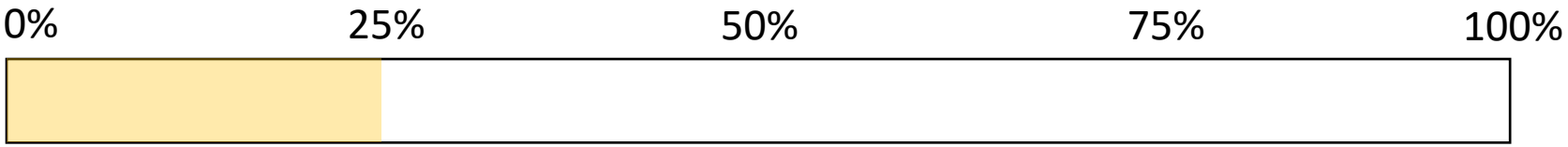
TC. #1
a = 3
b = 9

TC. #2
a = -5
b = -8

```
1  public void printSum(int a, int b){
2      int result = a+b;
3      if(result >0){
4          print("red", result);
5      else if(result <0){
6          print("blue, result");
7      else [donothing]
8  }
```



Branch coverage = 1/4



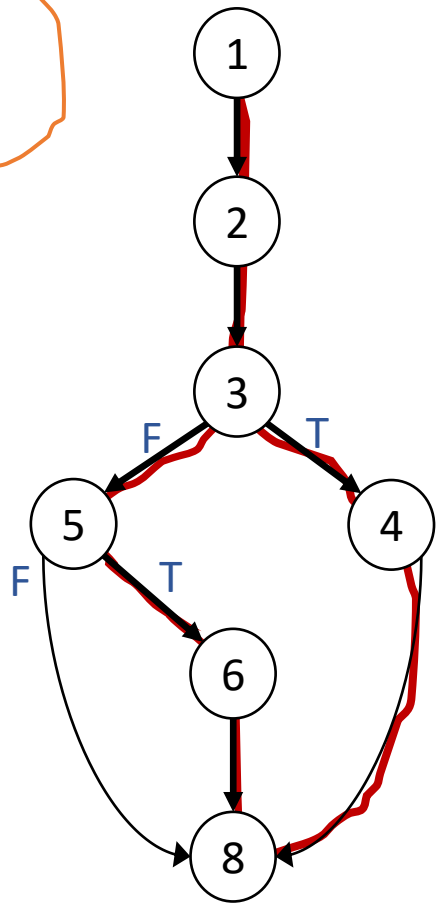
Branch Coverage

- Example

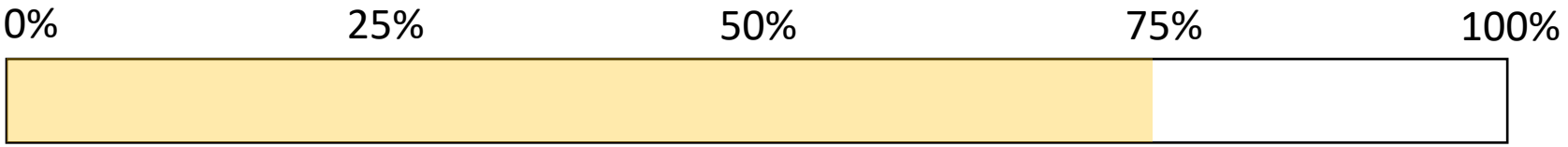
TC. #1
a = 3
b = 9

TC. #2
a = -5
b = -8

```
1  public void printSum(int a, int b){
2      int result = a+b;
3      if(result >0){
4          print("red", result);
5      else if(result <0){
6          print("blue, result");
7      else [donothing]
8  }
```



Branch coverage = 3/4



Branch Coverage

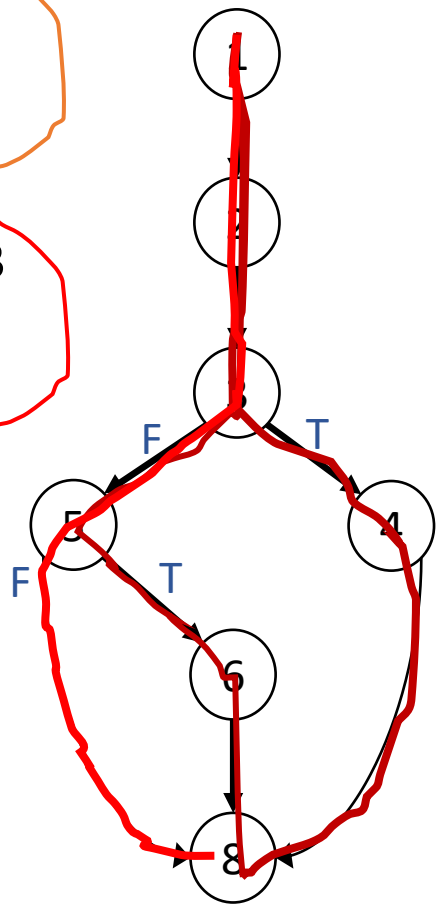
- Example

TC. #1
a = 3
b = 9

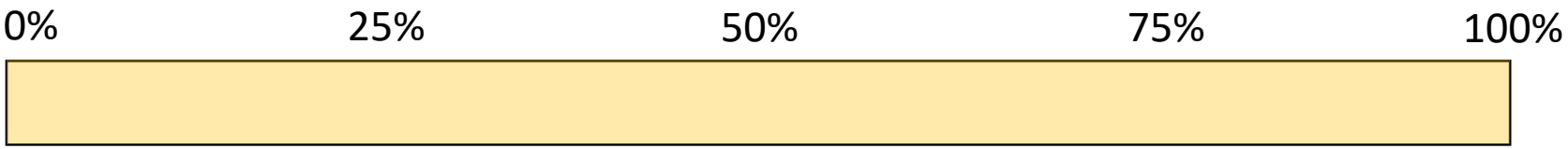
TC. #2
a = -5
b = -8

TC. #3
a = 0
b = 0

```
1  public void printSum(int a, int b){
2      int result = a+b;
3      if(result >0){
4          print("red", result);
5      else if(result <0){
6          print("blue, result");
7      else [donothing]
8  }
```



Branch coverage = 4/4



Summary

- Statement coverage
- How to objectively measure statement coverage
- Branch coverage
- How to objectively measure branch coverage
- Examples based on control flow graphs

Control Flow Testing: Part C

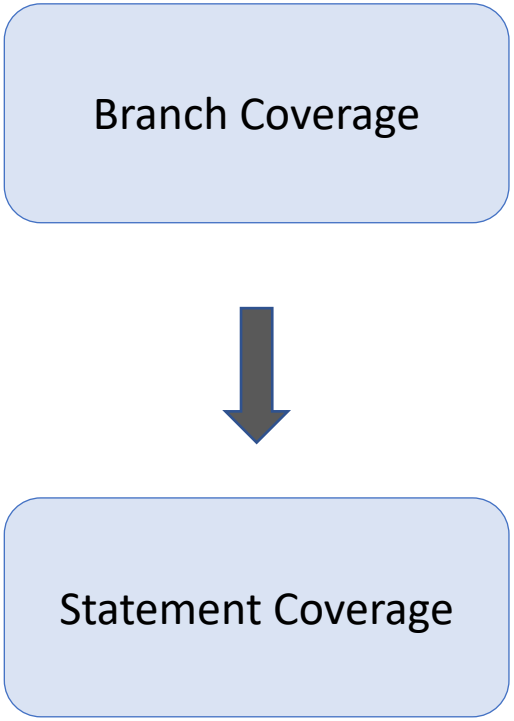
Dr Fani Deligianni,

Fani.Deligianni@glasgow.ac.uk

https://www.gla.ac.uk/schools/computing/staff/fanideligianni/

Test Criteria Subsumption

- One test criteria subsumes another criteria when all the test suites that satisfies that criteria will also satisfy the other



If a test suite achieve a 100% branch coverage, the same test suite will also achieve (necessarily)100% statement coverage

Control Flow Graphs

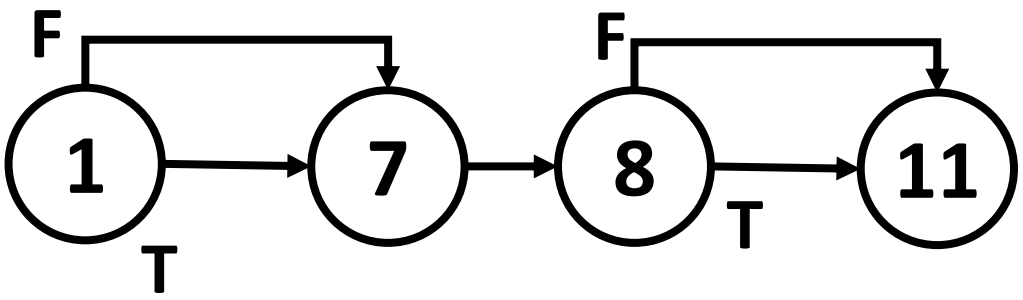
- A graphical representation of a program's control structure
- They consist of nodes
  - **Decision points** – the control can diverge (ie. If statement)
  - **Junction points** – the control flow can merge (ie. End if or for loop)
  - **Process blocks** – sequence of program statements uninterrupted by either decisions or junctions
    - A process has one entry and one exit point.

Control Flow Graphs

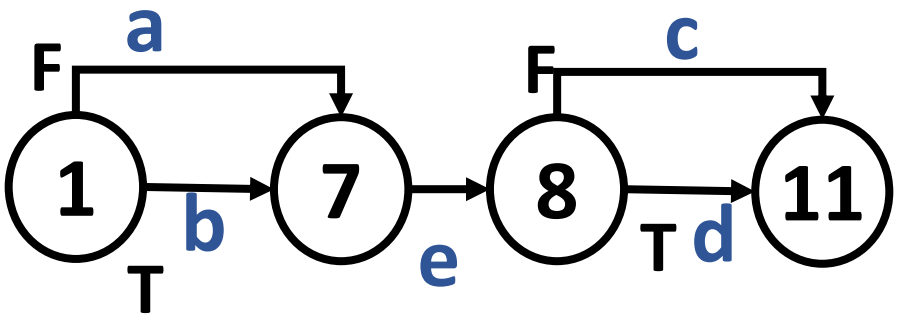
- Multiple entry points
- A path is a sequence of consecutive nodes
- Complete testing
    A. Cover every path from entry to exit
    B. Cover every statement at least once
    C. Cover every branch at least once
- A implies B and C (but impractical)
- B is not equal to C

Control Flow Graph

```
1   if(x!=0){
2       z=x+10;
3   }
4   else{
5       z=0;
6   }
7   System.out.print(x);
8   if(y>0){
9       w = y/z;
10  }
11  return w;
```



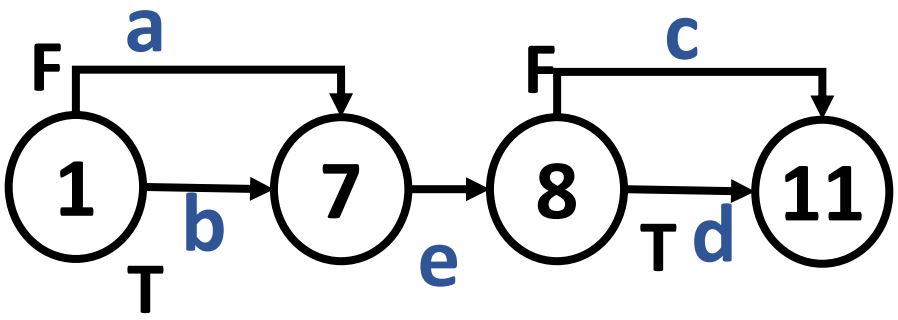
Statement Coverage




```
1   if(x!=0){
2       z=x+10;
3   }
4   else{
5       z=0;
6   }
7   System.out.print(x);
8   if(y>0){
9       w = y/z;
10  }
11  return w;
```

PATHS	PROCESS LINKS					TEST CASES	
	a	b	c	d	e	INPUT	OUTPUT
aec	✓		✓		✓	x=0 y=-1	w
bed		✓		✓	✓	x=1 y= 1	w

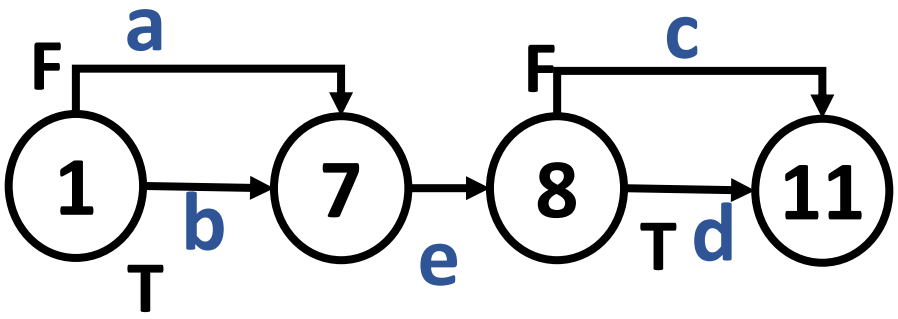
Branch Coverage




```
1   if(x!=0){
2       z=x+10;
3   }
4   else{
5       z=0;
6   }
7   System.out.print(x);
8   if(y>0){
9       w = y/z;
10  }
11  return w;
```

PATHS	DECISIONS		TEST CASES	
	1	8	INPUT	OUTPUT
aec	F	F	x=0 y=-1	w
bed	T	T	x=1 y= 1	w

Branch Coverage




```
1   if(x!=0){
2       z=x+10;
3   }
4   else{
5       z=0;
6   }
7   System.out.print(x);
8   if(y>0){
9       w = y/z;
10  }
11  return w;
```

PATHS	DECISIONS		TEST CASES	
	1	8	INPUT	OUTPUT
aec	F	F	x=0 y=-1	w
bed	T	T	x=1 y= 1	w

Although we have 100% statement and branch coverage, the erroneous computation still has not been found

Summary

- Statement and branch testing dominates control-flow testing.
- If a test suite has 100% branch coverage, the same test suite will also achieve 100% statement coverage
- Even with 100% branch and statement coverage, there are bugs not found.

Control Flow Testing: Part D

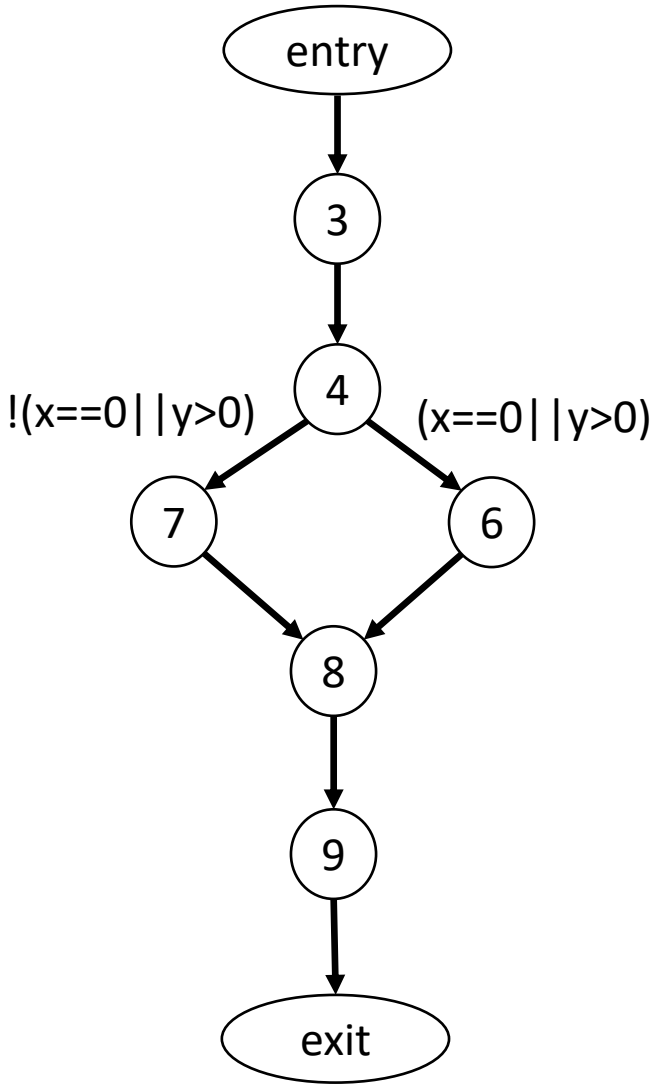
Dr Fani Deligianni,

Fani.Deligianni@glasgow.ac.uk

https://www.gla.ac.uk/schools/computing/staff/fanideligianni/

Condition Coverage

```
1  void main(){
2    float x,y;
3    read(x);
4    read(y);
5    if(((x==0)||(y>0))
6      y = y/x;
7    else x = x+2;
8    write(x);
9    write(y)
10 }
```



Tests: (x=5, y=5)
(x=5, y=-6)

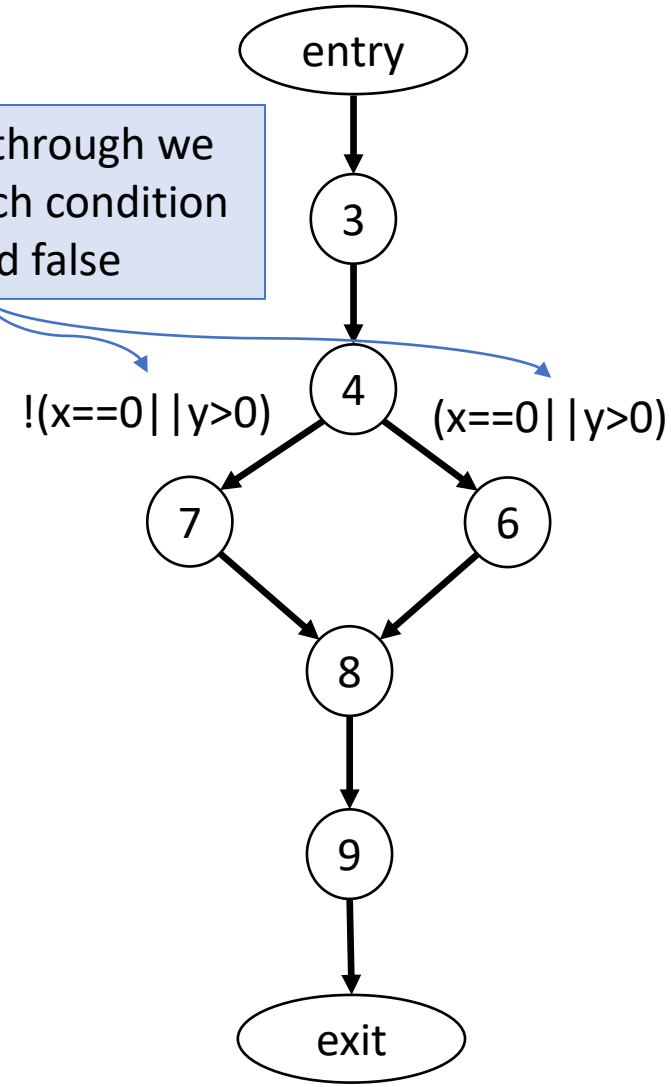
Branch coverage = 100%

But there is still a bug in line 6

Condition Coverage

```
1  void main(){
2    float x,y;
3    read(x);
4    read(y);
5    if(((x==0)||(y>0))
6      y = y/x;
7    else x = x+2;
8    write(x);
9    write(y)
10 }
```

To be more through we can make each condition true and false



Tests: (x=5, y=5)
(x=5, y=-6)

Branch coverage = 100%

But there is still a bug in line 6

Condition Coverage

- Characterised by two aspects:

(1) Test Requirements

Individual conditions in the program

(2) Coverage Measure

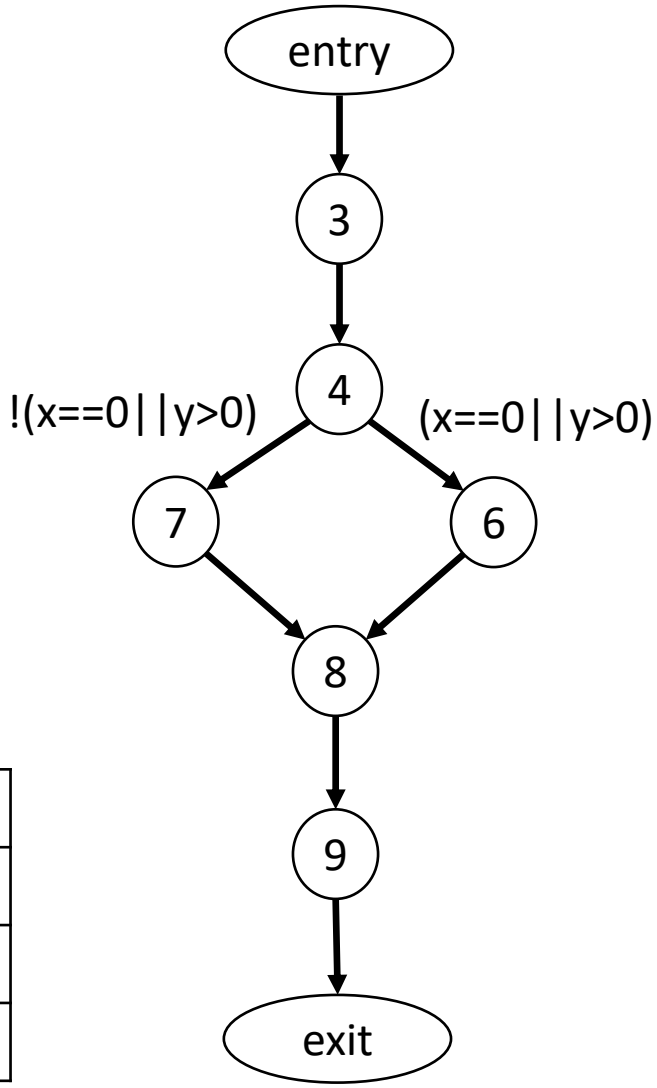
$$\frac{\text{Number of conditions that are both T and F}}{\text{Total number of conditions}}$$

Condition Coverage

```
1  void main(){
2    float x,y;
3    read(x);
4    read(y);
5    if(((x==0)||(y>0))
6      y = y/x;
7    else x = x+2;
8    write(x);
9    write(y)
10 }
```

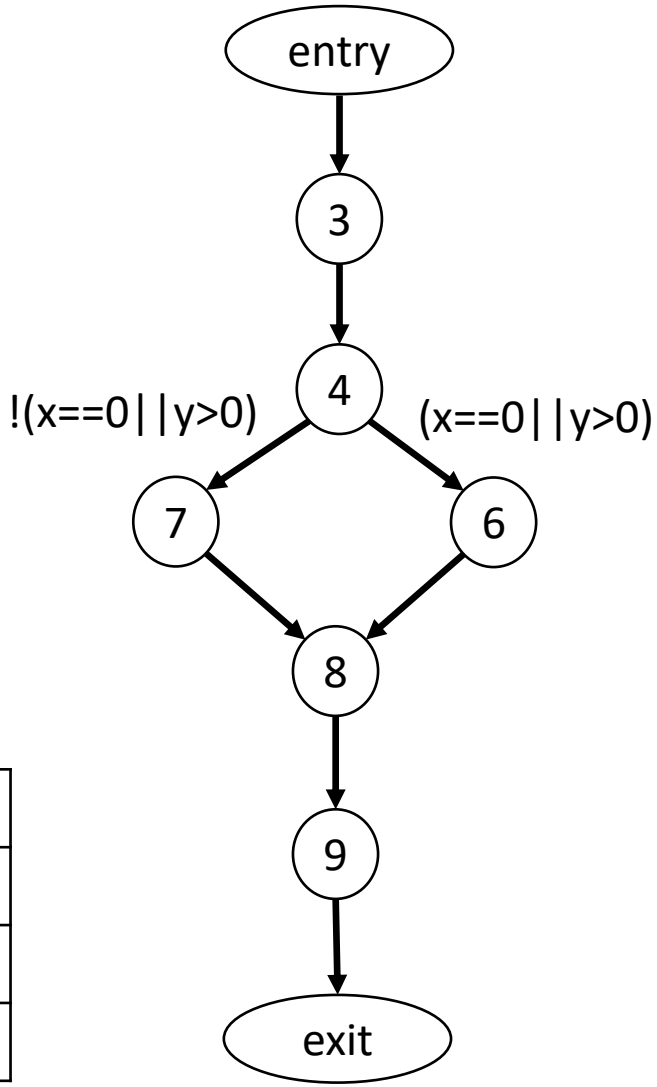
TC #1 (x=0, y=-5)
#2 (x=5, y=5)

		Condition	
		x ==0	y>0
TC	#1		
	#2		



Condition Coverage

```
1  void main(){
2    float x,y;
3    read(x);
4    read(y);
5    if(((x==0)||(y>0))
6      y = y/x;
7    else x = x+2;
8    write(x);
9    write(y)
10 }
```



TC #1 (x=0, y=-5)
#2 (x=5, y=5)

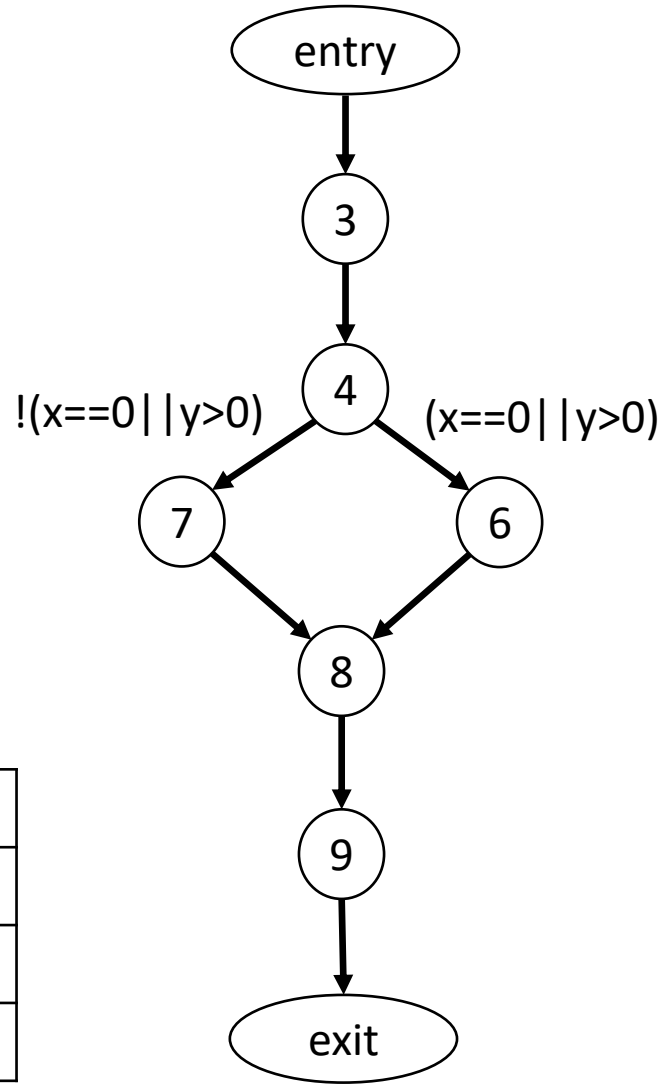
		Condition	
		x ==0	y>0
TC	#1	T	F
	#2		

Condition coverage = 0/2

0% 25% 50% 75% 100%

Condition Coverage

```
1  void main(){
2    float x,y;
3    read(x);
4    read(y);
5    if(((x==0)||(y>0))
6      y = y/x;
7    else x = x+2;
8    write(x);
9    write(y)
10 }
```

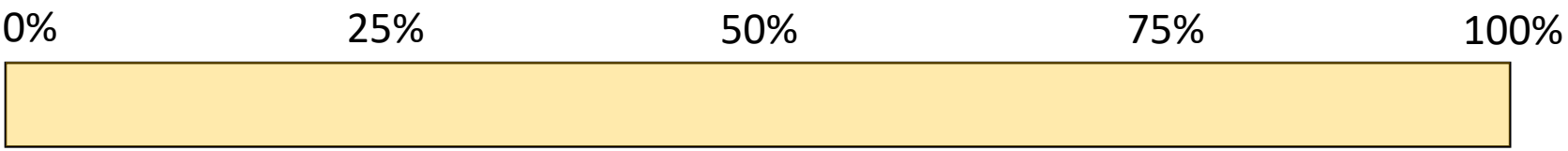


TC #1 (x=0, y=-5)

#2 (x=5, y=5)

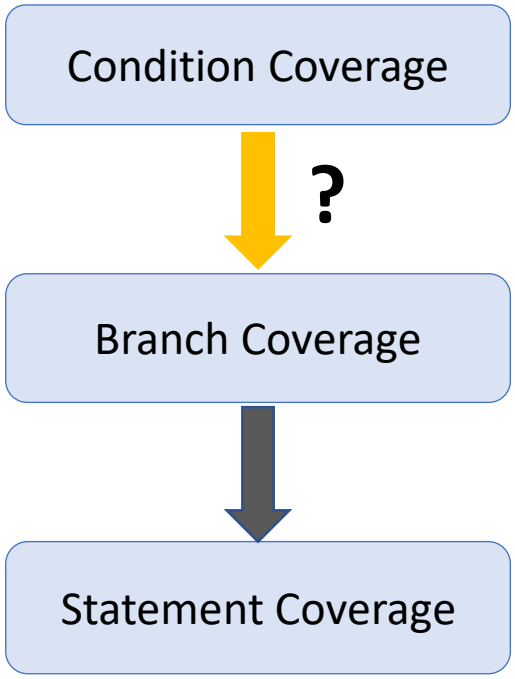
		Condition	
		x ==0	y>0
TC	#1	T	F
	#2	F	T

Condition coverage = 2/2



Subsumption

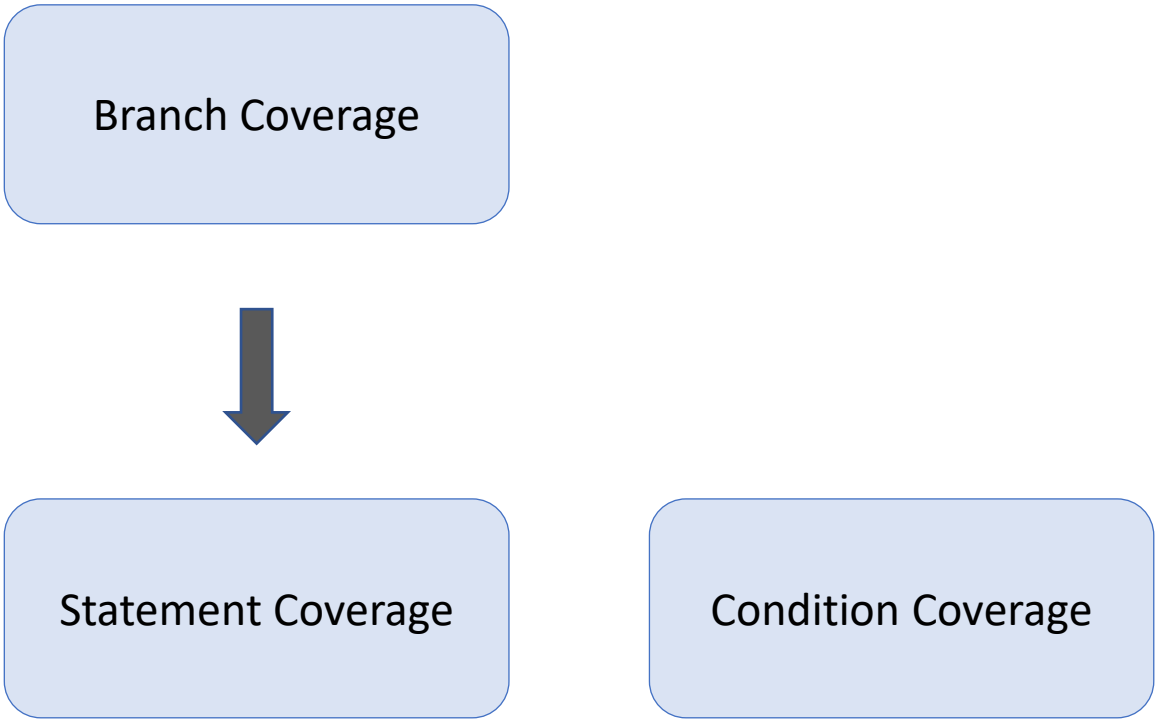
- Does Condition Coverage imply branch coverage?



[ ] YES
[ ] NO

Test Criteria Subsumption

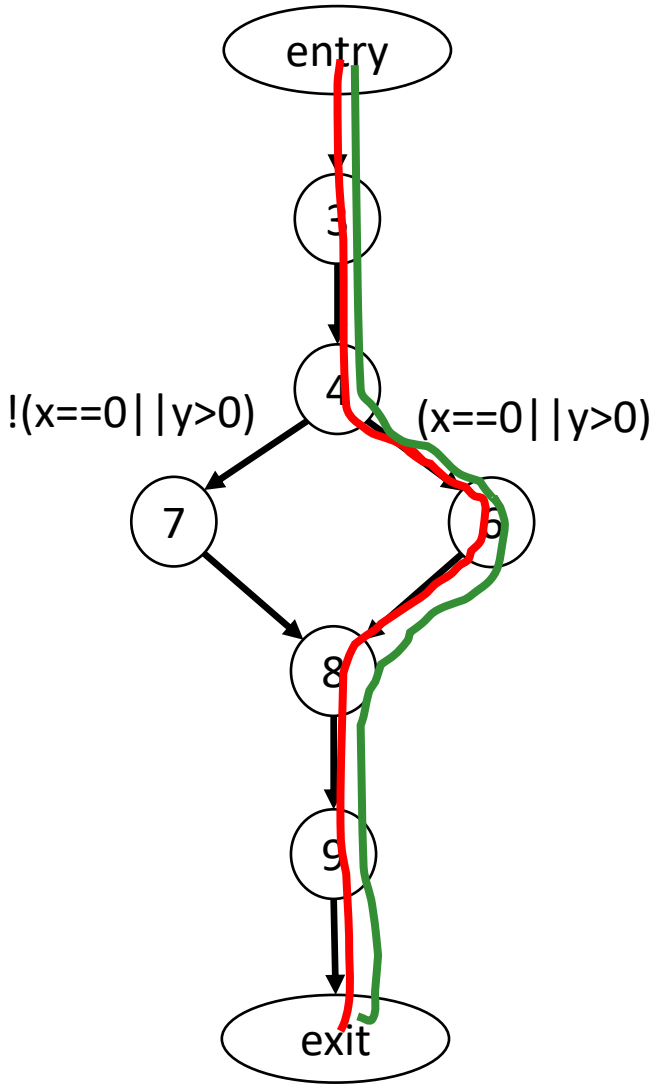
- One test criteria subsumes another criteria when all the test suites that satisfies that criteria will also satisfy the other



Test Criteria Subsumption

```
1  void main(){
2    float x,y;
3    read(x);
4    read(y);
5    if(((x==0)||(y>0))
6      y = y/x;
7    else x = x+2;
8    write(x);
9    write(y)
10 }
```

TC #1 (x=0, y=-5)
#2 (x=5, y=5)



Condition coverage = 100%
Branch coverage = 50%

Hence, condition coverage does not subsume branch coverage and are better to be considered together

Branch and Condition Coverage

- Characterised by two aspects:

(1) Test Requirements

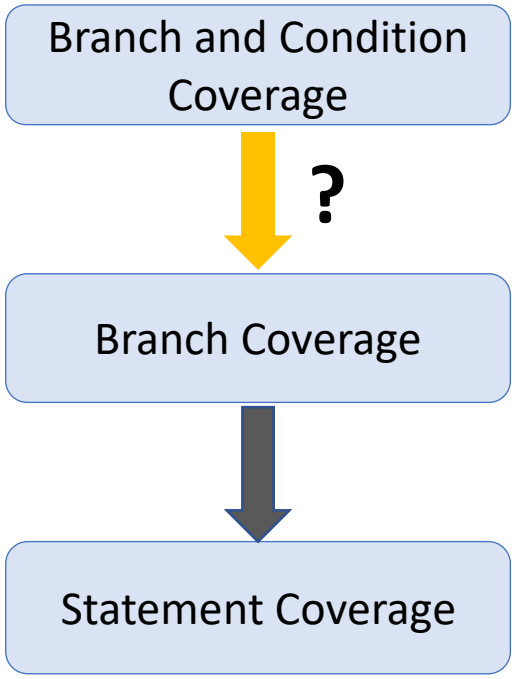
All branches and individual conditions in the program

(2) Coverage Measure

Computed considering both coverage measures

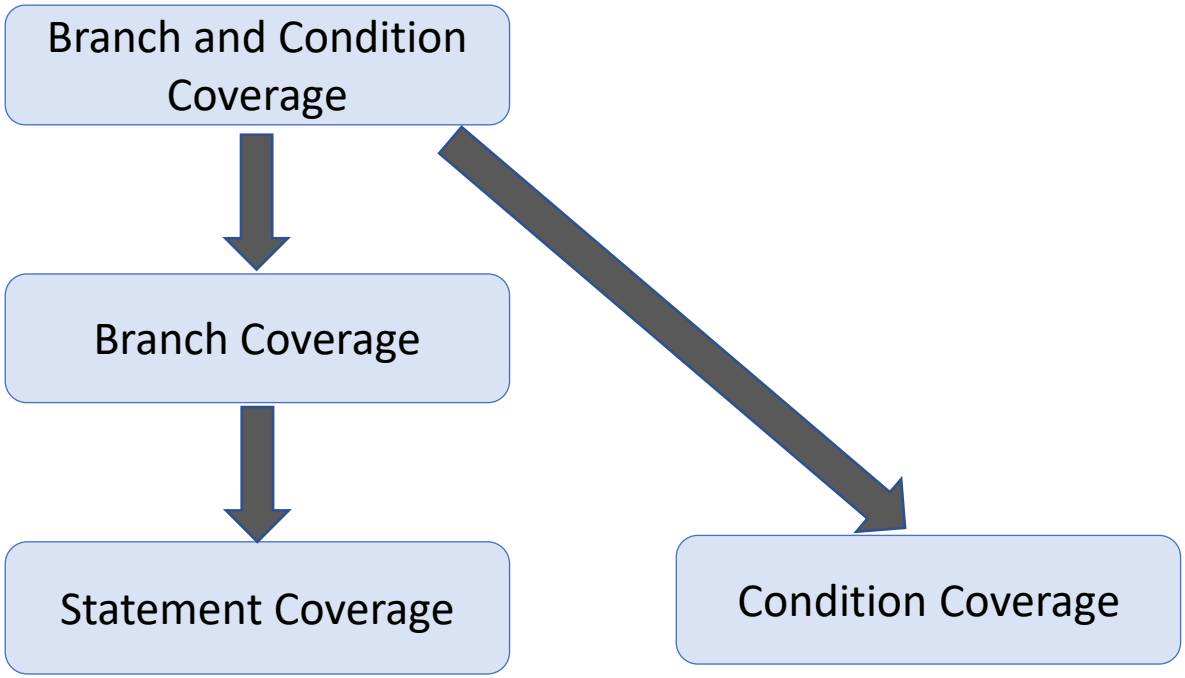
Subsumption

- Does Branch and condition coverage imply branch coverage?



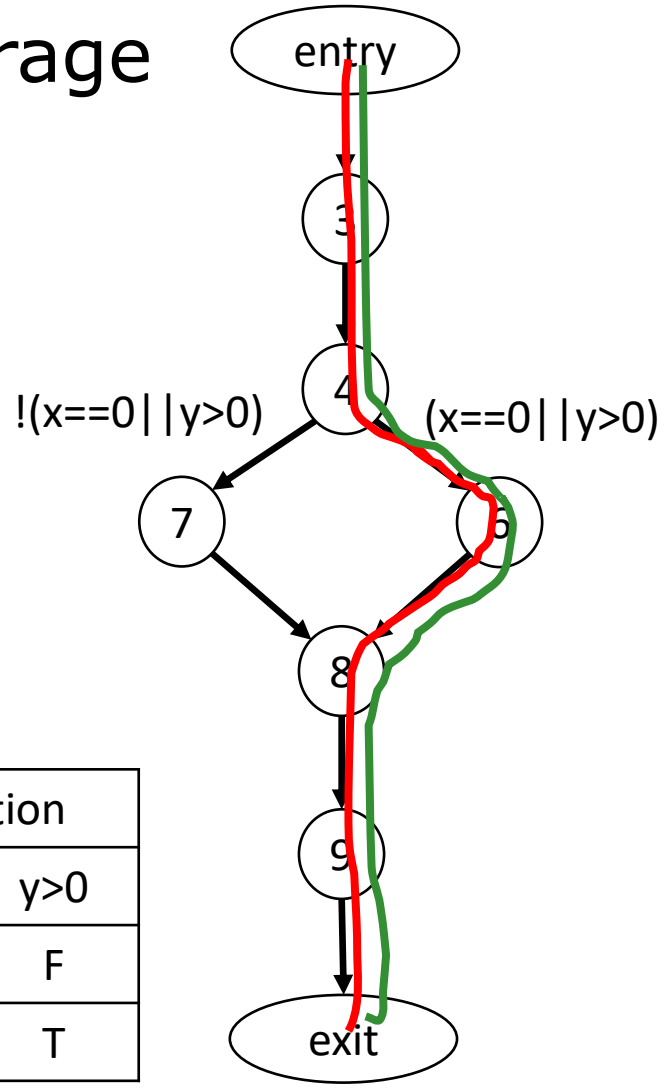
[ ] YES
[ ] NO

Test Criteria Subsumption



Branch and Condition Coverage

```
1  void main(){
2    float x,y;
3    read(x);
4    read(y);
5    if(((x==0)||(y>0))
6      y = y/x;
7    else x = x+2;
8    write(x);
9    write(y)
10 }
```



TC #1 (x=0, y=-5)
#2 (x=5, y=5)

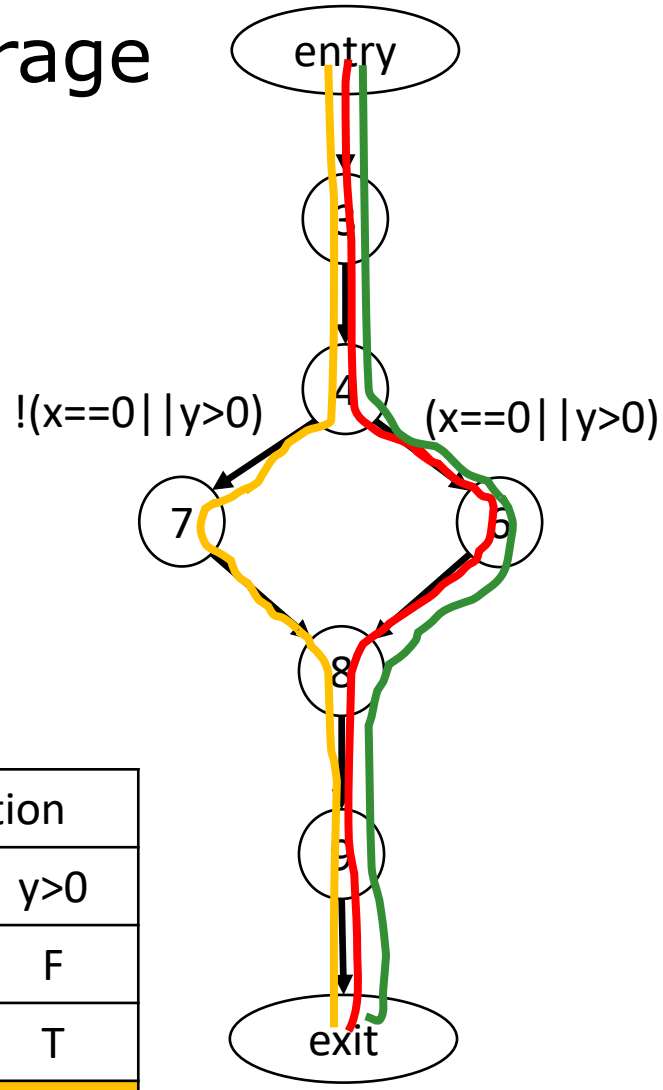
		Condition	
		x ==0	y>0
TC	#1	T	F
	#2	F	T

Condition coverage = 100%
Branch coverage = 50%

Question:
Add a test case that will result in 100% branch and condition coverage?

Branch and Condition Coverage

```
1  void main(){
2    float x,y;
3    read(x);
4    read(y);
5    if(((x==0)||(y>0))
6      y = y/x;
7    else x = x+2;
8    write(x);
9    write(y)
10 }
```



TC #1 (x=0, y=-5)

#2 (x=5, y=5)

#3 (x=3, y=-2)

Condition coverage = 100%

Branch coverage = ?

		Condition	
		x ==0	y>0
TC	#1	T	F
	#2	F	T
	#3	F	F

Summary

- Conditional coverage
- Conditional coverage can complement branch coverage
- We can combine conditional and branch coverage to create more robust tests

Control Flow Testing: Part E

Dr Fani Deligianni,

Fani.Deligianni@glasgow.ac.uk

https://www.gla.ac.uk/schools/computing/staff/fanideligianni/

Modified Condition/Decision Coverage

- Very important criterion as it is often required for safety critical applications.

Modified Condition/Decision Coverage

- **Key Idea:** To test important combinations of conditions and limiting testing cost by excluding non-important combinations

Approach:
Extend branch and decision coverage with requirement that **each condition should affect the decision outcome independently**.

Modified Condition/Decision Coverage

- **Example**: Assuming the predicate a && b && c

Test case	a	b	c	Outcome

How many test cases needed to satisfy multiple condition coverage?

Modified Condition/Decision Coverage

- **Example**: Assuming the predicate a && b && c

Test case	a	b	c	Outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

8 test cases, with each test case having a different combination of values for a,b and c.

Modified Condition/Decision Coverage

- **Example**: Assuming the predicate a && b && c

Test case	a	b	c	Outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

Identifying important combinations:
Combinations in which a single condition independently affects the outcome of the overall predicate

Modified Condition/Decision Coverage

- **Example**: Assuming the predicate a && b && c

Test case	a	b	c	Outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

1	True	True	True	True
5	False	True	True	False

Find two test cases such that the only difference between the two test cases is the value of **a** and the overall outcome of the predicate

Modified Condition/Decision Coverage

- **Example**: Assuming the predicate a && b && c

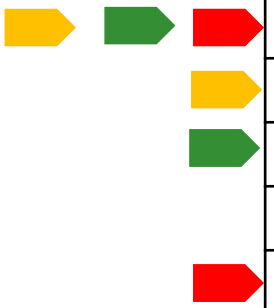
Test case	a	b	c	Outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

1	True	True	True	True
5	False	True	True	False
3	True	False	True	False

Find two test cases such that the only difference between the two test cases is the value of **b** and the overall outcome of the predicate

Modified Condition/Decision Coverage

- **Example**: Assuming the predicate a && b && c



Test case	a	b	c	Outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

1	True	True	True	True
5	False	True	True	False
3	True	False	True	False
2	True	True	False	False

Find two test cases such that the only difference between the two test cases is the value of c and the overall outcome of the predicate

Modified Condition/Decision Coverage

- **Example**: Assuming the predicate a && b && c

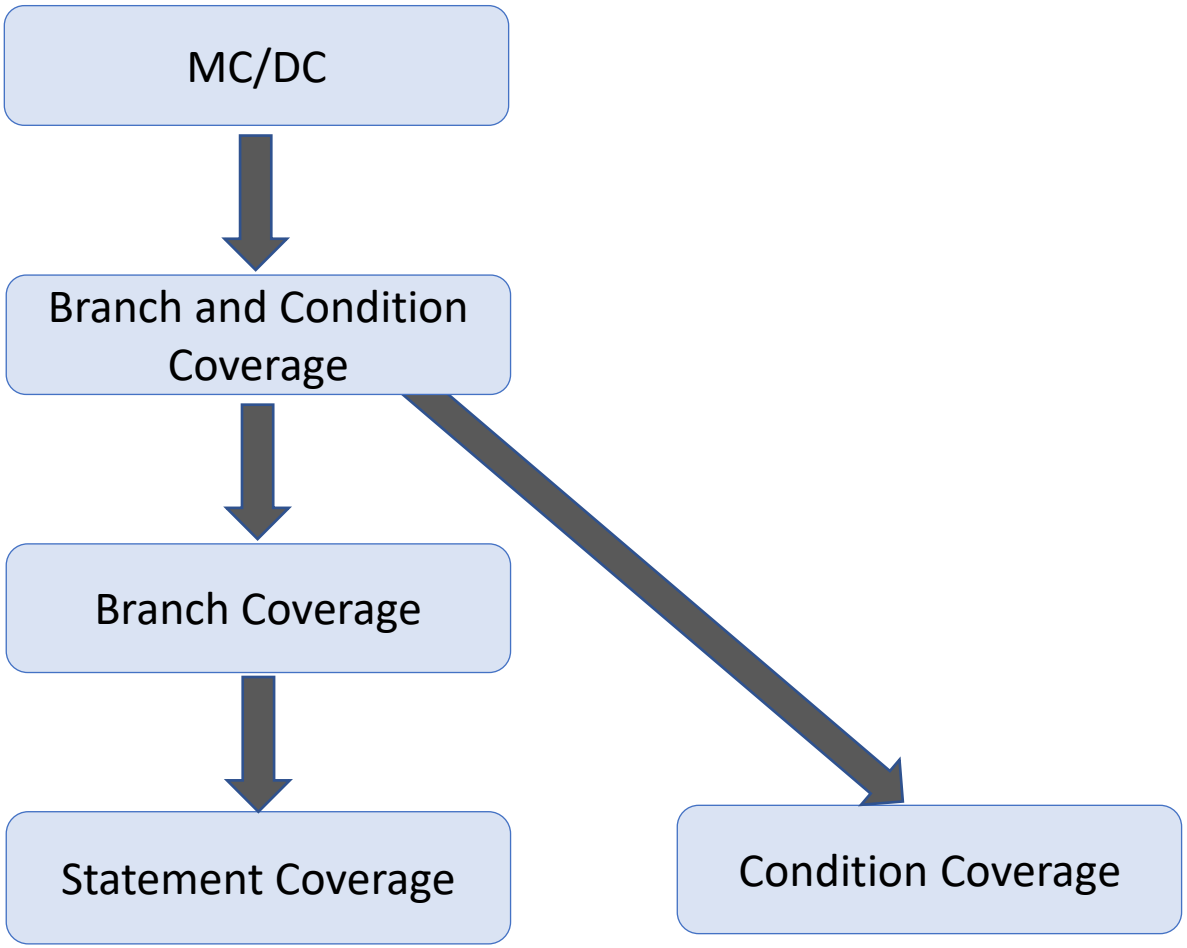
Test case	a	b	c	Outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

8 test cases to cover all possible combinations

1	True	True	True	True
5	False	True	True	False
3	True	False	True	False
2	True	True	False	False

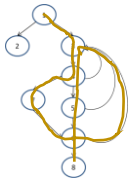
4 test cases to satisfy MC/DC criteria

Test Criteria Subsumption

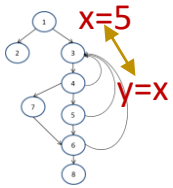


Other Criteria

Path Coverage



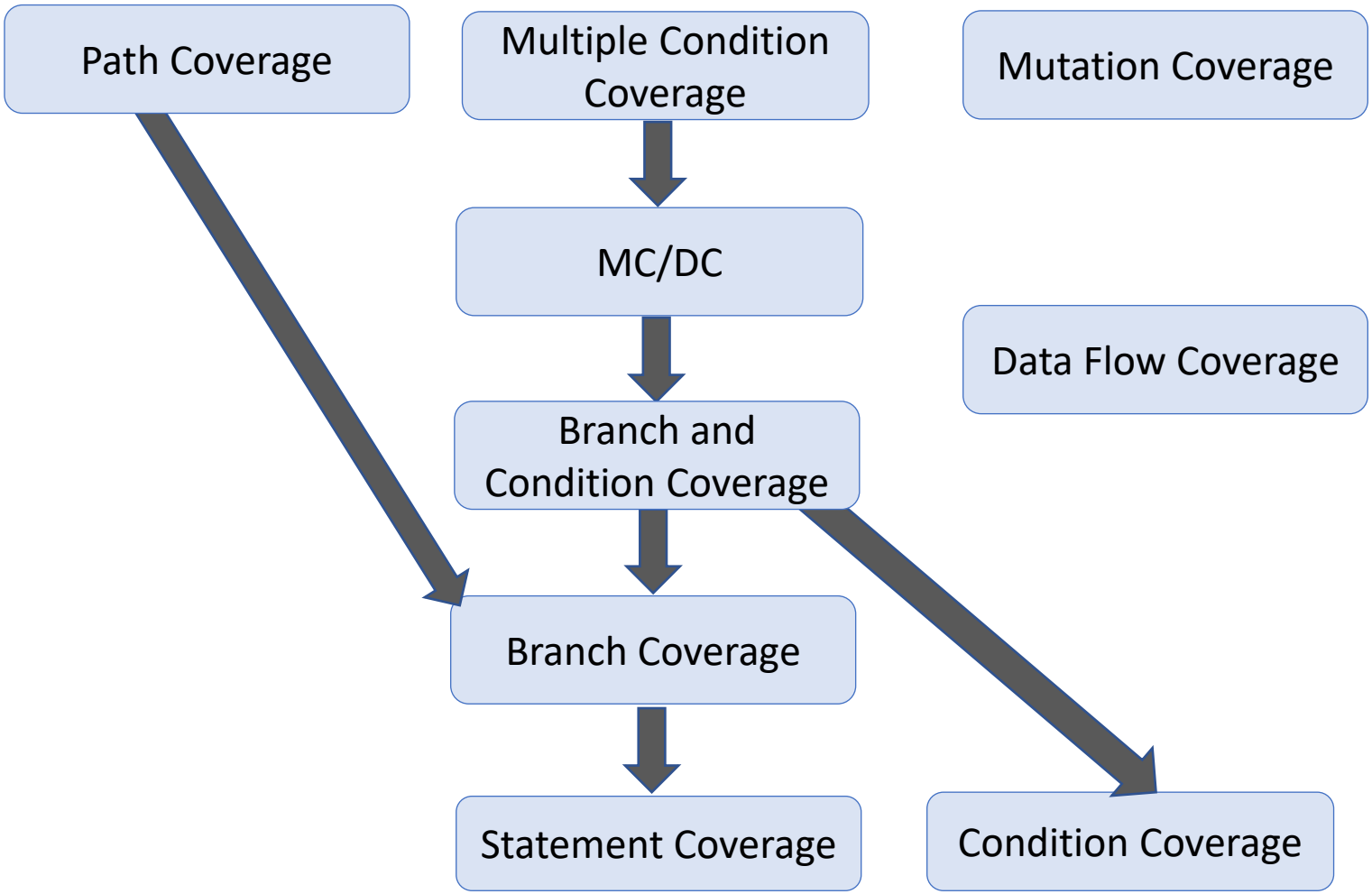
Data-Flow Coverage



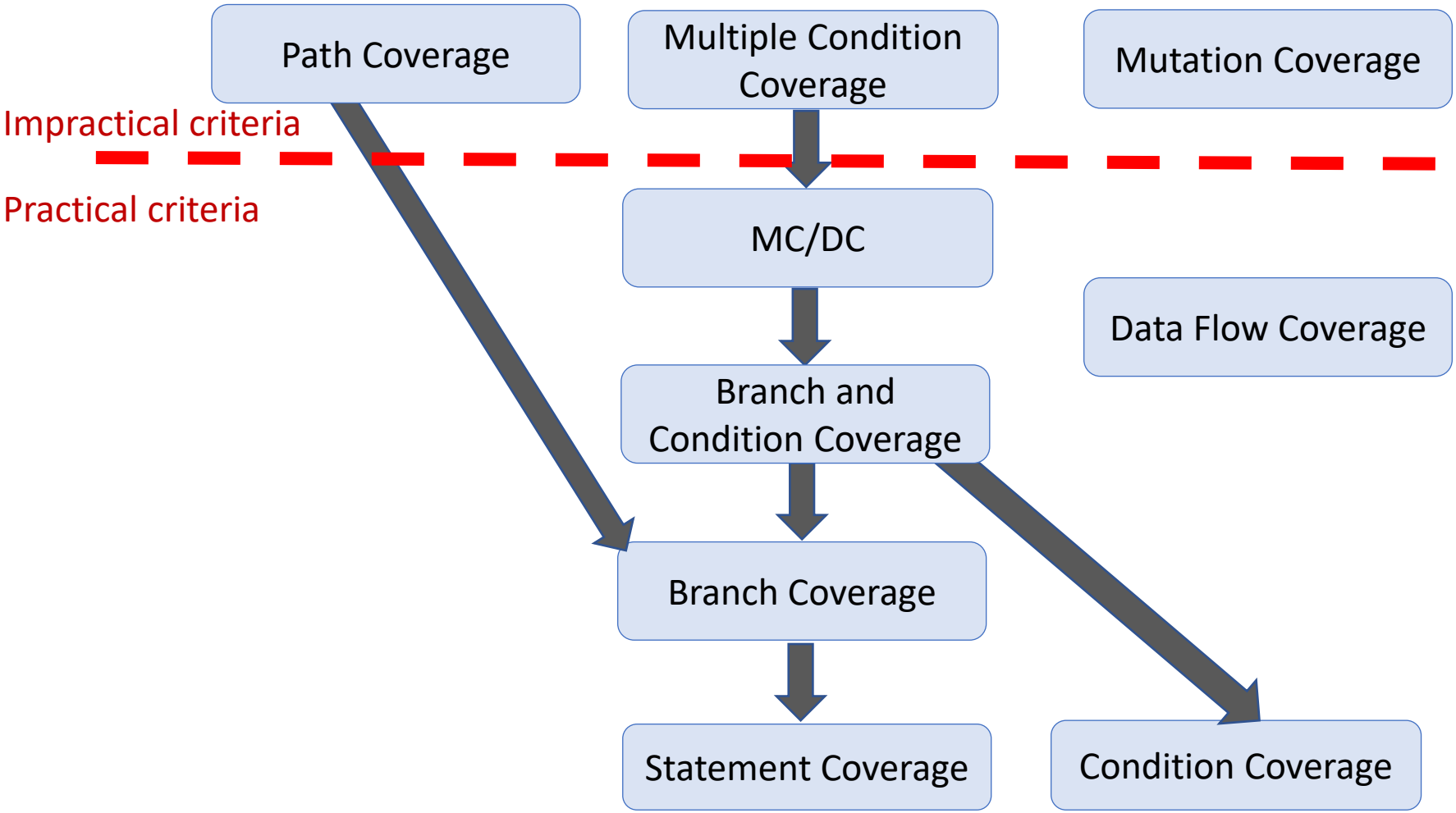
Mutation Coverage

```
if(k  >  9)
if(k  >=9)
```

Test Criteria Subsumption



Test Criteria Subsumption



Summary

- About 65% of all bugs can be caught in unit testing.
- Unit testing is dominated by control-flow testing methods.
- Statement and branch testing dominates control-flow testing.
- Studies show that control-flow testing catches 50% of all bugs caught during unit testing.
  - About 33% of all bugs.