Control Flow Testing

Structural Testing

- In structural testing, the software is viewed as a white box and test cases are determined from the implementation of the software.
- Structural testing techniques include control flow testing and data flow testing.

Control Flow Testing

- Control flow testing uses the control structure of a program to develop the test cases for the program.
- The test cases are developed to sufficiently cover the whole control structure of the program.
- The control structure of a program can be represented by the control flow graph of the program.

Control Flow Graph

- The control flow graph $G = (N, E)$ of a program consists of a set of nodes N and a set of edge E.
- Each node represents a set of program statements. There are five types of nodes. There is a unique entry node and a unique exit node.
- There is an edge from node n_1 to node n_2 if the control may flow from the last statement in n_1 to the first statement in n_2.

Control Flow Graph: Nodes

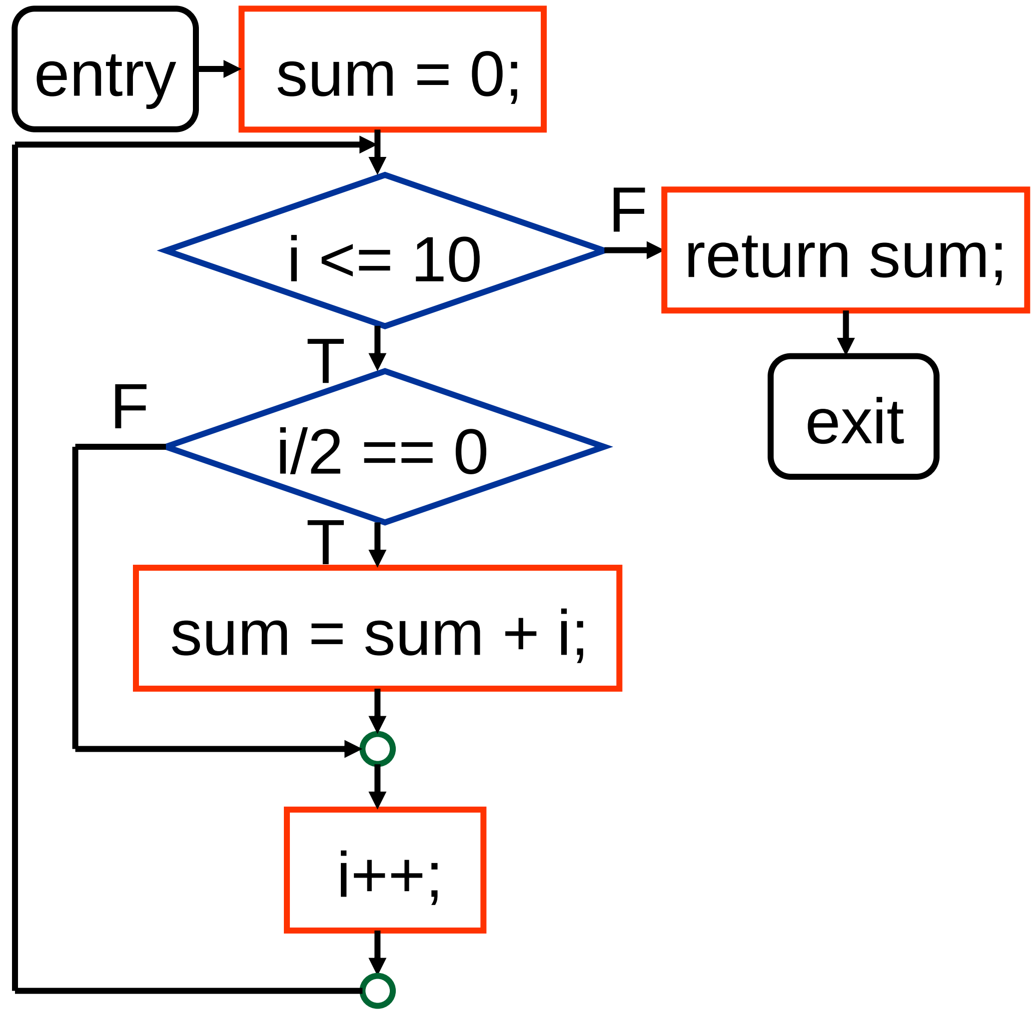
- A decision node contains a conditional statement that creates 2 or more control branches (e.g. if or switch statements).
- A merge node usually does not contain any statement and is used to represent a program point where multiple control branches merge.
- A statement node contains a sequence of statements. The control must enter from the first statement and exit from the last statement.

Control Flow Graph: An Example

```
int evensum(int i)
{
  int sum = 0;

  while (i <= 10) {
    if (i/2 == 0)
      sum = sum + i;
    i++;
  }
  return sum;

}
```



Test Cases

- A test case is a complete path from the entry node to the exit node of a control flow graph.
- A test coverage criterion measures the extent to which a set of test cases covers a program.

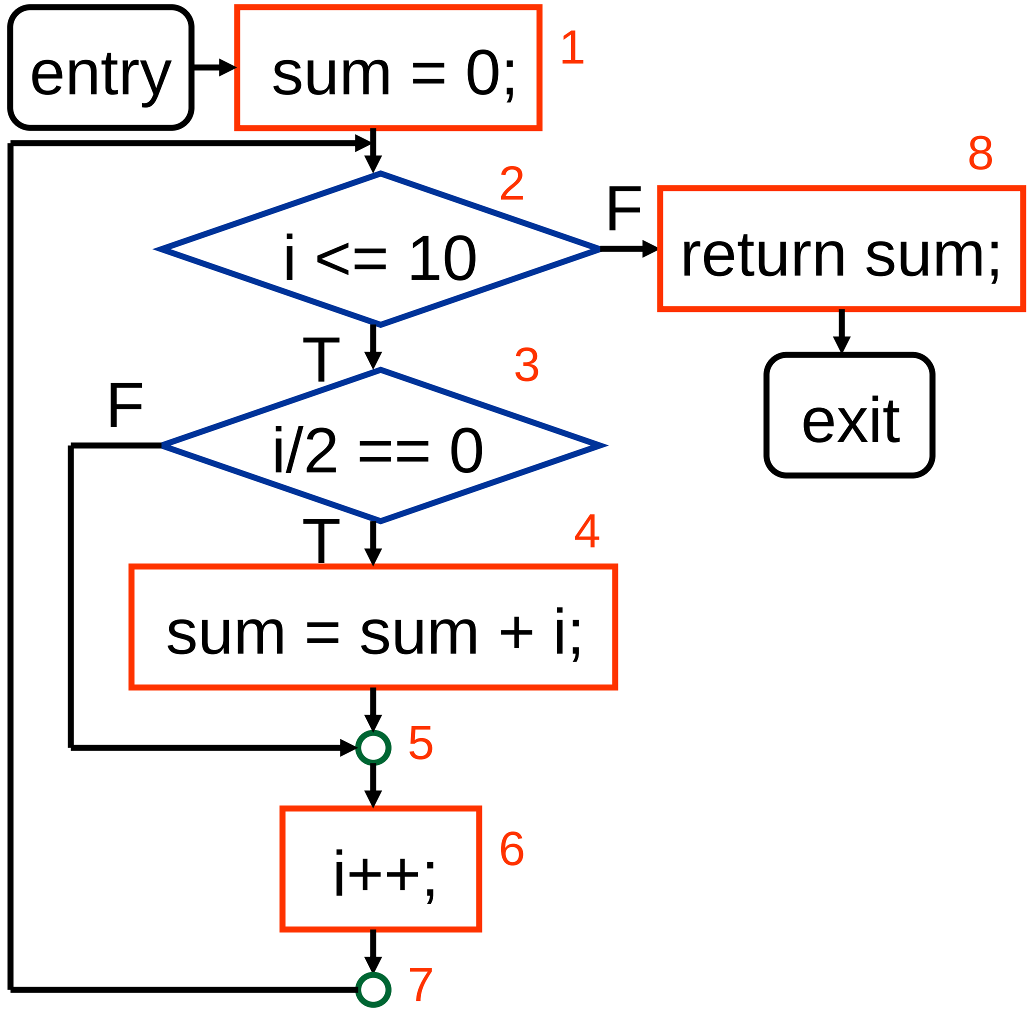
Test Coverage Criteria

- Statement coverage (SC) Node coverage
- Decision coverage (DC) Edge coverage
- Condition coverage (CC)
- Decision/condition coverage (D/CC)
- Multiple condition coverage (MCC)
- Path coverage (PC)

Statement Coverage

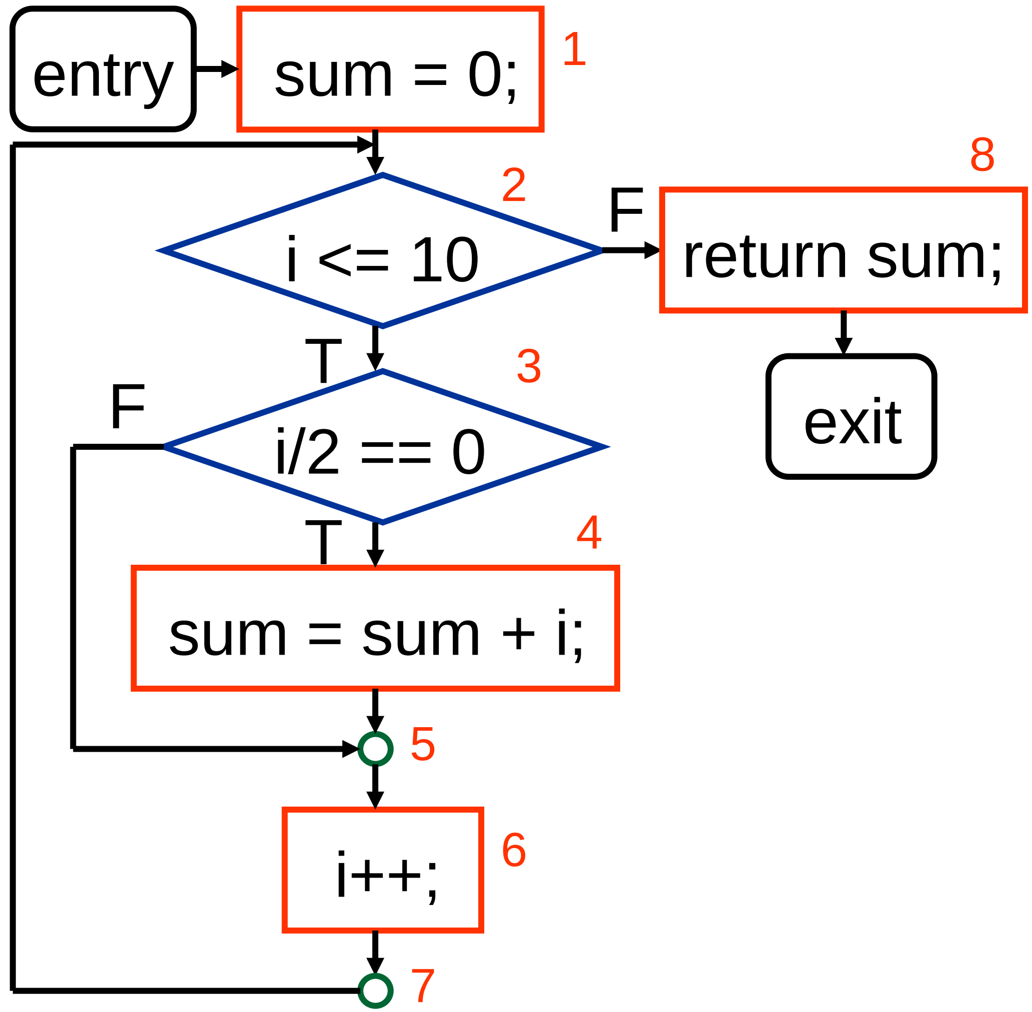
- Every statement in the program has been executed at least once.

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 2 \rightarrow 8$



Decision Coverage

- Every statement in the program has been executed at least once, and every decision in the program has taken all possible outcomes at least once.



$1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 2 \rightarrow$
$3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 2 \rightarrow 8$

Decision Coverage

$d = (A \wedge B) \vee C$

combination number	value				DC	CC	D/CC	MCC
	A	B	C	d				
1	1	1	1	1	1			
2	1	1	0	1	1			
3	1	0	1	1	1			
4	0	1	1	1	1			
5	1	0	0	0	2			
6	0	1	0	0	2			
7	0	0	1	1	1			
8	0	0	0	0	2			

Condition Coverage

- Every statement in the program has been executed at least once, and every condition in each decision has taken all possible outcomes at least once.

Condition Coverage

$d = (A \wedge B) \vee C$

combination number	value A	B	C	d	DC	CC	D/CC	MCC
1	1	1	1	1	1	1_1		
2	1	1	0	1	1	1_2		
3	1	0	1	1	1	1_3		
4	0	1	1	1	1	1_4		
5	1	0	0	0	2	2_4		
6	0	1	0	0	2	2_3		
7	0	0	1	1	1	2_2		
8	0	0	0	0	2	2_1		

Decision/Condition Coverage

- Every statement in the program has been executed at least once, every decision in the program has taken all possible outcomes at least once, and every condition in each decision has taken all possible outcomes at least once.

Decision/Condition Coverage

$d = (A \wedge B) \vee C$

combination number	value				DC	CC	D/CC	MCC
	A	B	C	d				
1	1	1	1	1	1	1_1	1_1	
2	1	1	0	1	1	1_2		
3	1	0	1	1	1	1_3	1_2	
4	0	1	1	1	1	1_4	1_3	
5	1	0	0	0	2	2_4	2_3	
6	0	1	0	0	2	2_3	2_2	
7	0	0	1	1	1	2_2		
8	0	0	0	0	2	2_1	2_1	

Multiple Condition Coverage

- Every statement in the program has been executed at least once, all possible combination of condition outcomes in each decision has been invoked at least once.
- There are 2^n combinations of condition outcomes in a decision with n conditions.

Multiple Condition Coverage

$d = (A \wedge B) \vee C$

combination number	value A B C d	DC	CC	D/CC	MCC
1	1 1 1 1	1	1_1	1	1
2	1 1 0 1	1	1_2		2
3	1 0 1 1	1	1_3	1_2	3
4	0 1 1 1	1	1_4	1_3	4
5	1 0 0 0	2	2_4	2_3	5
6	0 1 0 0	2	2_3	2_2	6
7	0 0 1 1	1	2_2		7
8	0 0 0 0	2	2_1	2_1	8

Path Coverage

- Every complete path in the program has been executed at least once.
- A loop usually has an infinite number of complete paths.

Test Coverage Criteria Hierarchy

path coverage

↓

multiple condition coverage

↓

decision/condition coverage

↙ ↘

decision coverage   condition coverage

↘ ↙

statement coverage

Testing Simple Loops

- Skip the loop entirely
- Go once through the loop
- Go twice through the loop
- If the loop has max passes = n, then go n – 1, n, and n + 1 times through the loop

Testing Nested Loops

- Set all outer loops to their minimal value and test the innermost loop
- Add tests of out-of-range values
- Work outward, at each stage holding all outer loops at their minimal value
- Continue until all loops are tested

Java Code Coverage Tool

- EclEmma is a free Java code coverage tool for Eclipse
  http://www.eclemma.org
- EclEmma adopts the philosophy of the EMMA Java code coverage tool for the Eclipse workbench
  http://emme.sourceforge.net

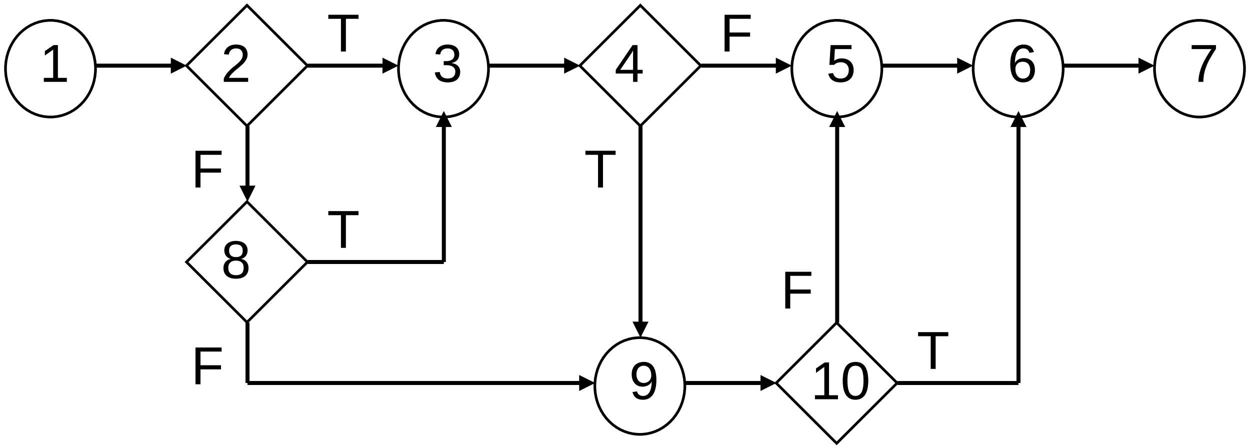
EclEmma

- Fast develop/test cycle: Launches from within the workbench like JUnit and test runs can directly be analyzed for code coverage.
- Rich coverage analysis: Coverage results are immediately summarized and highlighted in the Java source code editors.
- Non-invasive: EclEmma does not require modifying your projects or performing any other setup.

Path Selection

- It is better to take many simple paths than a few complicated ones.
- There is no harm in taking paths that will exercise the same code more than once.
- Select paths as small variations of previous paths.
- Try to change one thing in each path at a time.

An Example

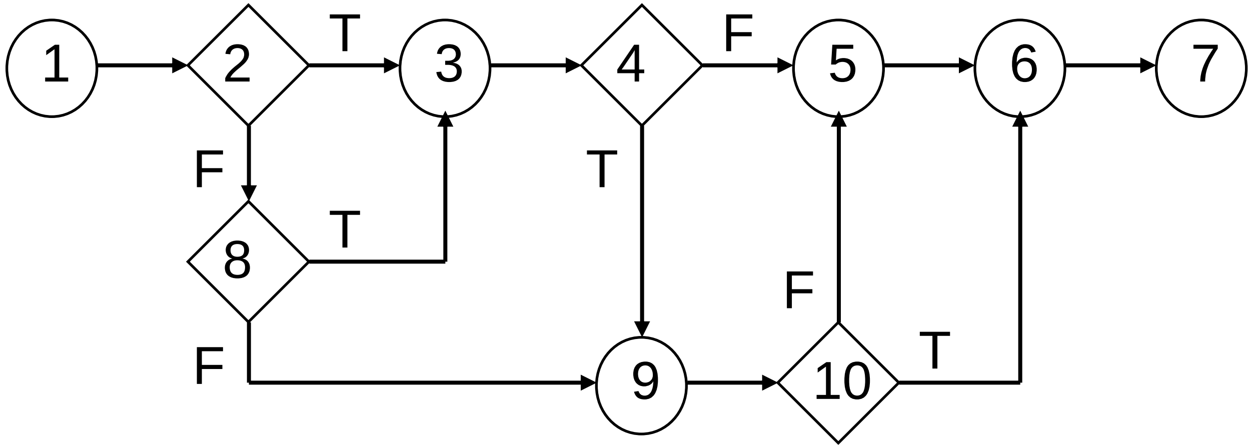


1 → 2 → 3 → 4 → 5 → 6 → 7   2 4

1 → 2 → 8 → 3 → 4 → 9 → 10 → 6 → 7   2 4 8 10

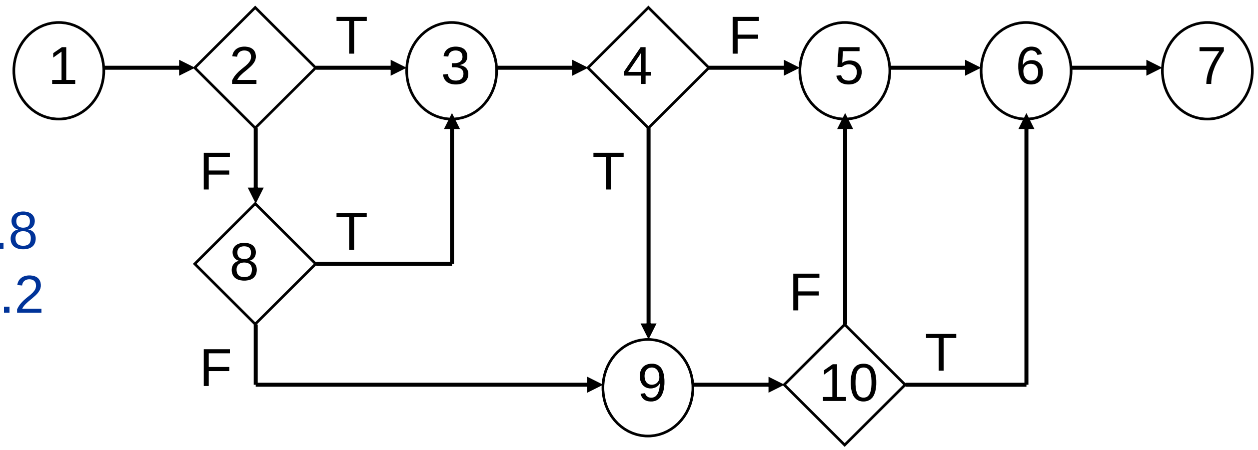
1 → 2 → 8 → 9 → 10 → 5 → 6 → 7   2 8 10

An Example



1 → 2 → 3 → 4 → 5 → 6 → 7	2	4		
1 → 2 → 3 → 4 → 9 → 10 → 5 → 6 → 7	2	4		10
1 → 2 → 3 → 4 → 9 → 10 → 6 → 7	2	4		10
1 → 2 → 8 → 3 → 4 → 9 → 10 → 6 → 7	2	4	8	10
1 → 2 → 8 → 9 → 10 → 6 → 7	2		8	10

An Example: Usage Information



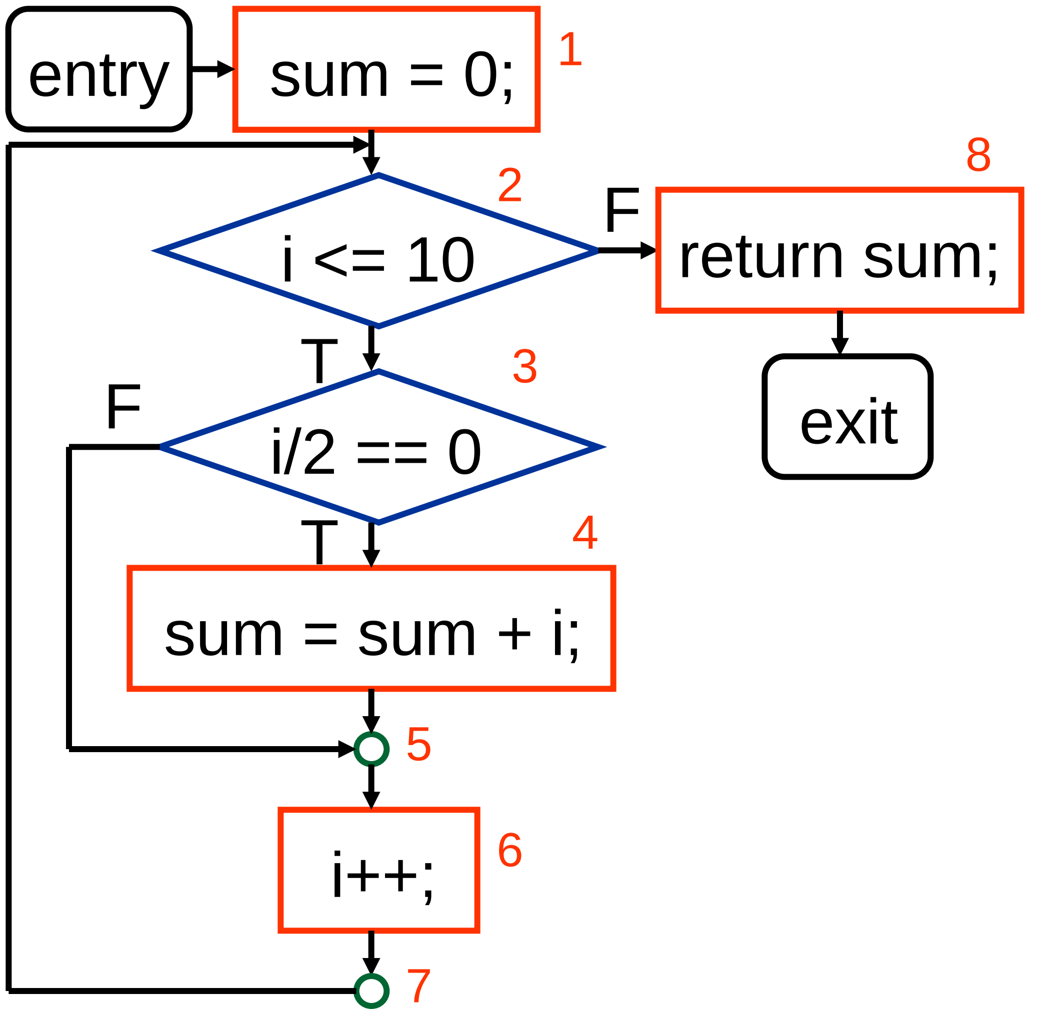
T: 0.8
F: 0.2

1 → 2 → 3 → 4 → 9 → 10 → 6 → 7	2	4		10
1 → 2 → 3 → 4 → 9 → 10 → 5 → 6 → 7	2	4		10
1 → 2 → 3 → 4 → 5 → 6 → 7	2	4		
1 → 2 → 8 → 3 → 4 → 9 → 10 → 6 → 7	2	4	8	10
1 → 2 → 8 → 9 → 10 → 6 → 7	2		8	10

Path Predicate Expression

- A complete path may contain a succession of decisions.
- An input vector is a tuple of values corresponding to the vector of input variables.
- A path predicate expression is a Boolean expression that characterizes the set of input vectors that will cause a complete path to be traversed.

An Example



$1 \to 2 \to 3 \to 4 \to$
$5 \to 6 \to 7 \to 2 \to 8$

{ i <= 10, i/2 == 0,
i + 1 > 10 }

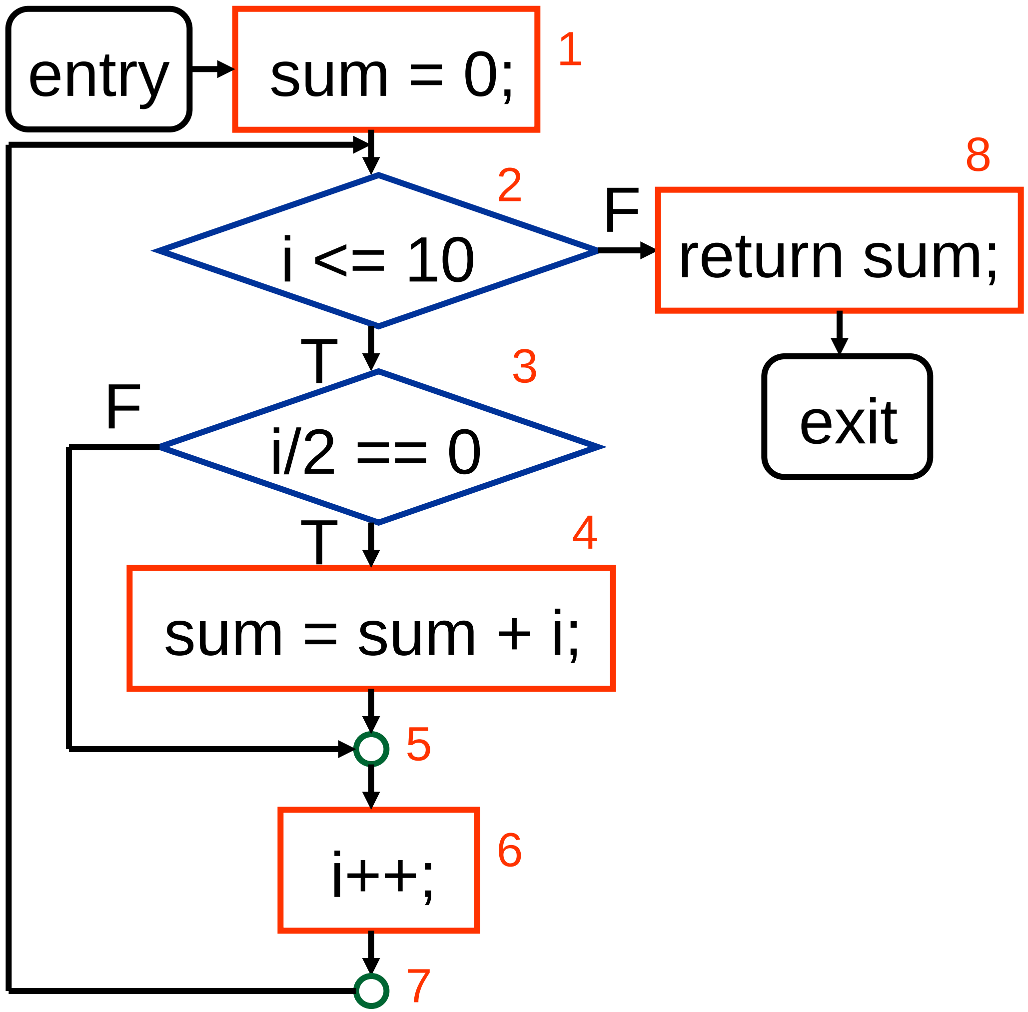
$1 \to 2 \to 3 \to 5 \to$
$6 \to 7 \to 2 \to 3 \to$
$4 \to 5 \to 6 \to 7 \to$
$2 \to 8$

{ i <= 10, i/2 != 0,
i+1 <= 10,
(i+1)/2 == 0,
i+2 > 10 }

Path Sensitization

- Path sensitization is the act of finding a set of solutions to a path predicate expression.
- If a path predicate expression has a solution, then the corresponding path is achievable; otherwise, the corresponding path is unachievable.

An Example: Correlated Decisions



$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow$
$5 \rightarrow 6 \rightarrow 7 \rightarrow 2 \rightarrow 8$

achievable

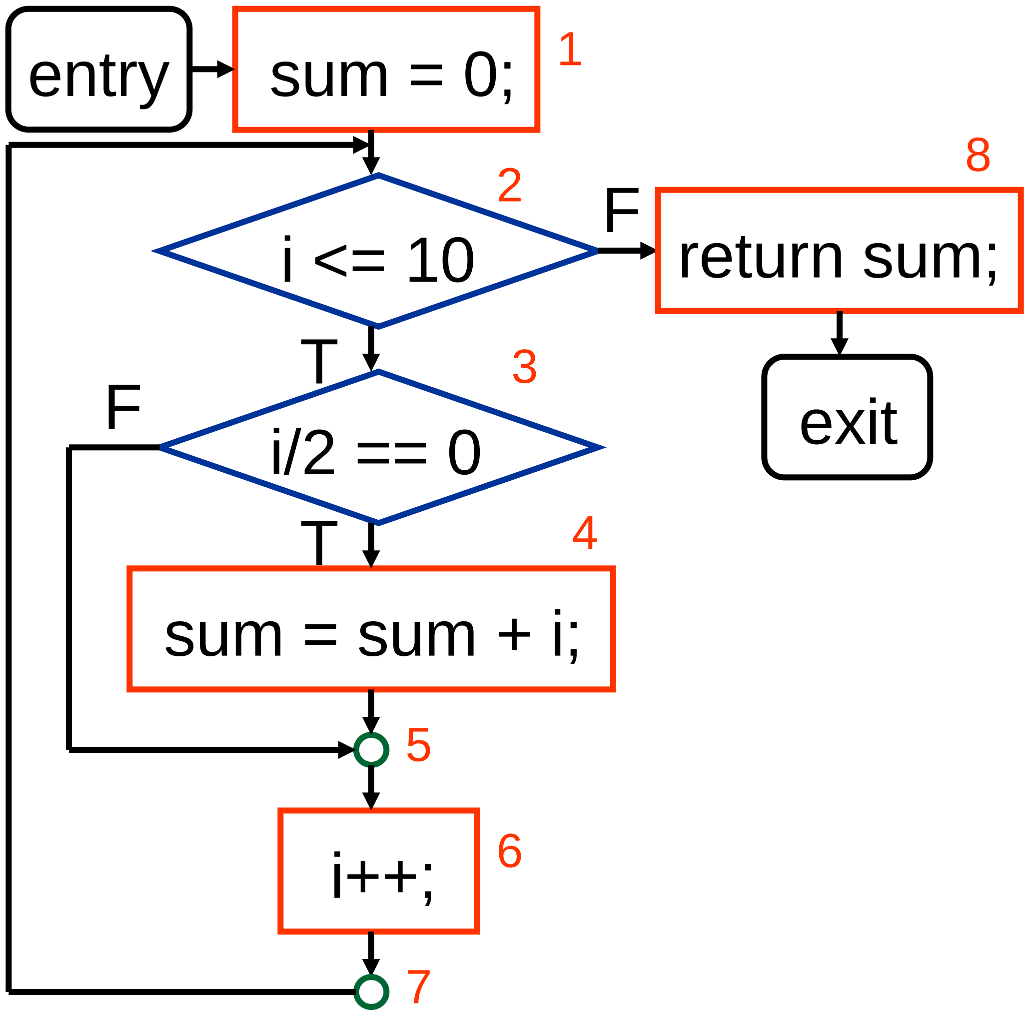
$1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow$
$6 \rightarrow 7 \rightarrow 2 \rightarrow 8$

unachievable

Test Oracle

- To verify the execution is correct, we need to compare the actual outcome with the expected outcome.
- Test oracle is a tool that can return the expected outcome for a given input vector.
- An executable specification of a program can be used as a test oracle for that program.

An Example

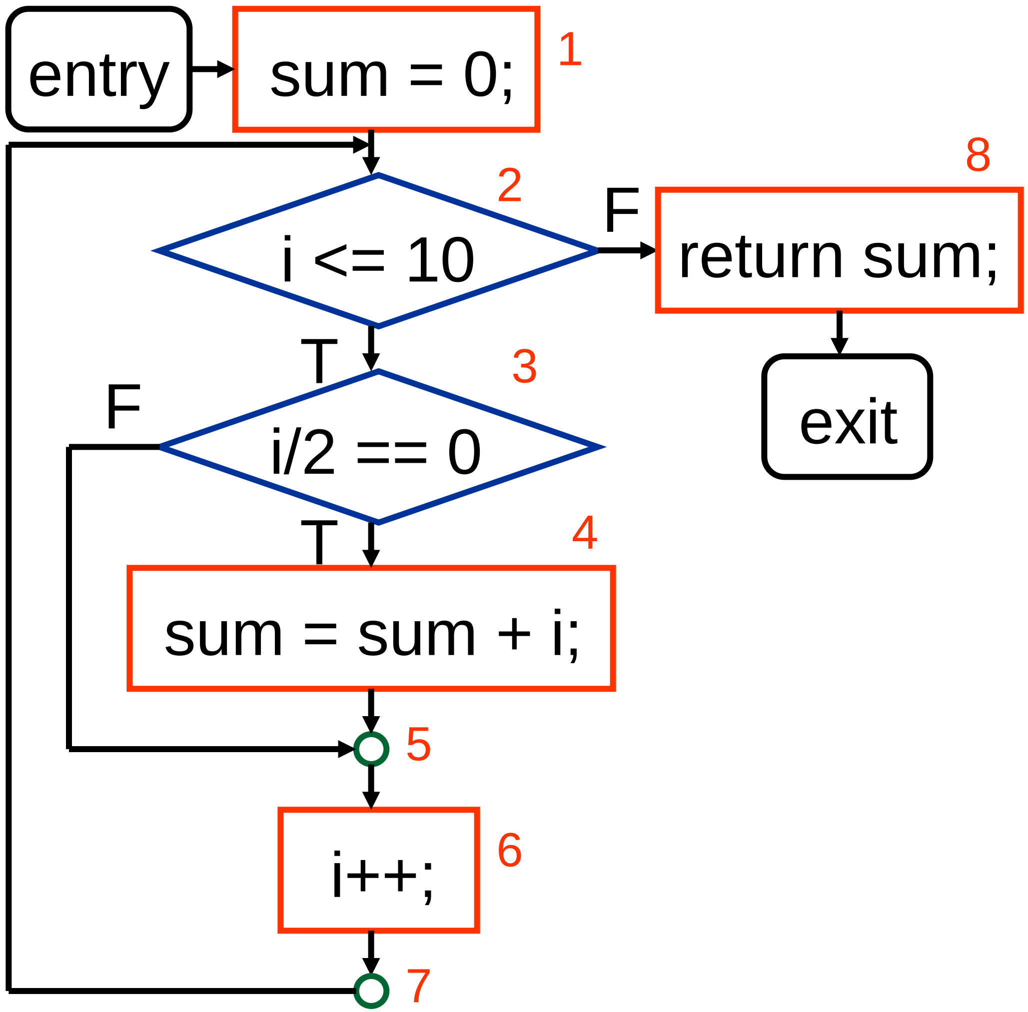


$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow$
$5 \rightarrow 6 \rightarrow 7 \rightarrow 2 \rightarrow 8$

path sensitization:
i = 10

test oracle:
sum = 10

An Example



1 → 2 → 3 → 5 →
6 → 7 → 2 → 3 →
4 → 5 → 6 → 7 →
2 → 8

path sensitization:
i = 9

test oracle:
sum = 10