Formal Methods in Software Engineering

Engr. Madeha Mushtaq

Department of Computer Science

Iqra National University

Why Formal Methods?

- Can we properly/thoroughly test the software to find the bugs in advance before the program goes into execution??
- Testing can detect bugs but we need to make sure absence of bugs.
- Suppose we have a theorem that states that all numbers are less than 1000.
- In order to prove this theorem, we take some examples.
- We say 1 is less than 1000, theorem holds true.
- We say 2 is less than 1000, theorem holds true.
- We say 3 is less than 1000, theorem holds true.
- And so on, so forth…

Why Formal Methods?

- We can have 100s of examples to show that the theorem holds true, but actually not all numbers are less than 1000.
- This looks like a trivial example but this is the simplest example that shows, even if we take 100s examples, we can not make sure whether the theorem holds true or not.
- The only way to prove the theorem is by using a formal proof.
- Same is the case with testing, for testing we write test cases, a test case is nothing more but an example scenario.
- If one particular scenario works that does not mean all scenarios will work.

Example

- Lets look at an example:
- In this program we have to test two strings and see if they are equal or not.
- We have to write test cases to test this program.
- Name of the function that tests the equivalence is “is equal”.
- It takes 2 parameters(strings) as arguments.

Example

- Program to test for two equal strings
- Test cases:
  - isEqual (“cat”, “dog”) - expected **false**
  - isEqual (“Testing”, “Testing”) - expected **true**
  - isEqual (“house”, “home”) - expected **false**

Example

- We have written 3 test cases.
- We can write many test cases as that and test the program, but will that actually help us?
- It may or it may not find an error.
- Lets have a look at the code and see if the code will work or not.

Example

```
equal = strlen(string1) == strlen(string2);
if (equal)
        for (i = 0; i < strlen(string1); i++)
                equal = string1[i] == string2[i];
return equal;
```

Example

- By looking closely at the code, we found out that the code is incorrect.
- What is the error? For two strings of equal length, if only the last character of the string is same. The program will conclude that the strings are equal.
- For example house and mouse.
- So by looking at this example, we again see that even if we write 100s of test cases, we may still not be able to find the error.

Black Box Testing

- In this string equivalence program, we were doing black box testing.
- BLACK BOX TESTING, also known as Behavioral Testing, is a software testing method in which the internal structure/implementation of the item being tested is not known to the tester.

White Box Testing/Structural Testing

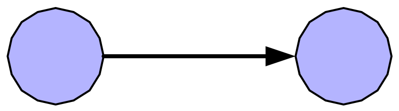
- WHITE BOX TESTING also known as Code-Based Testing or Structural Testing.
- It is a software testing method in which the internal structure/implementation of the item being tested is known to the tester.
- We make a flow graph, analyze it and develop test cases to test the different components of the program.

Flow Graphs

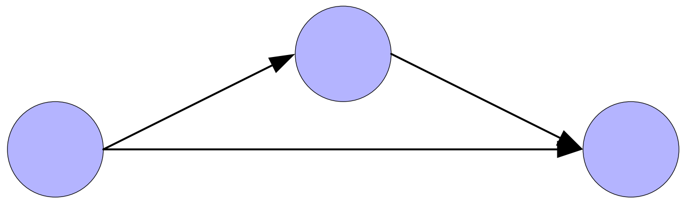
- We have different components of programs:
- Sequence
- Selection
  - Simple IF Statement
  - IF/Else Statement
  - Case Statement
- Loop

Flow Graphs

- Sequence:
  - Sequence is represented by two nodes and an arrow.
  - It is a directed graph.

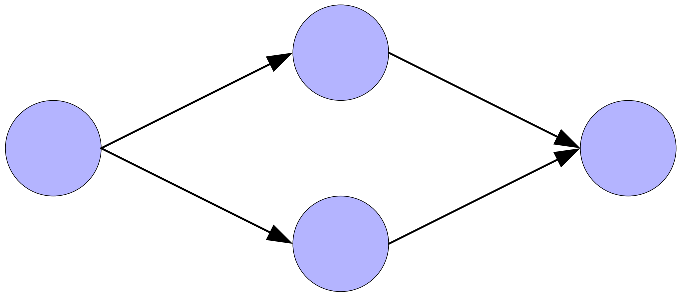

Sequence

Flow Graphs



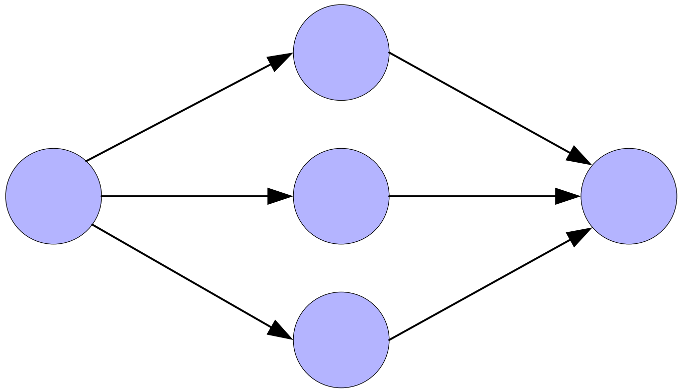
Selection – if statement

Flow Graphs



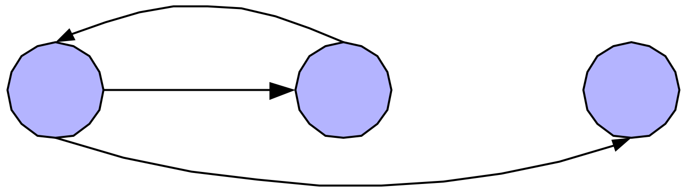
Selection – if-else statement

Flow Graphs



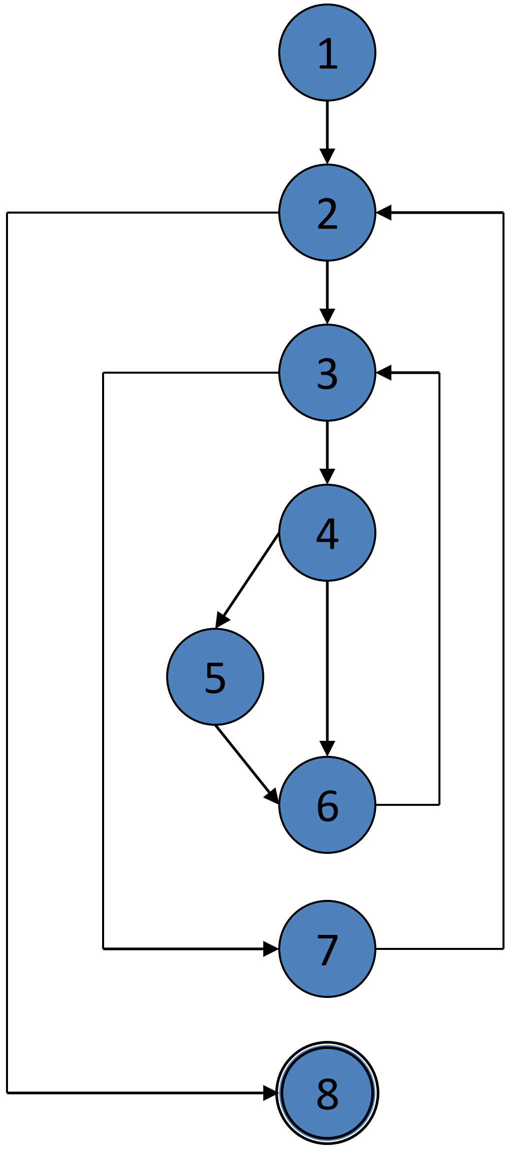
Selection – case statement

Flow Graphs


Loop

Flow graph for bubble sort

```
sorted = false;                              // 1
while (!sorted) {                                    // 2
    sorted = true;
    for (int i = 0; i < SIZE-1; i++) {               // 3
        if (a[i] > a[i+1]) {                         // 4
            swap(a[i], a[i+1]);                      // 5
            sorted = false;
        }
    }                                                //6
}                                                    //7
                                                     //8
```


$$\begin{tikzcd}
1 \arrow[d] \\
2 \arrow[d] \arrow[dddddd, bend right] \\
3 \arrow[d] \arrow[dddd, bend right] \\
4 \arrow[d, "" left] \arrow[dd, bend left] \\
5 \arrow[d] \\
6 \arrow[uuu, bend right] \\
7 \arrow[uuuuu, bend right] \\
8
\end{tikzcd}$$

Limitation of Flow Graphs

- Flow graph helps us in understanding the structure of the program.
- Based upon the structure of the program, we write test cases to test the various components of the program.
- But there is a limitation to this type of testing.
- Given the structure of a program in terms of a flow graph, we can have what is known as coverage.
- There are three types of coverage.
  – Statement Coverage
  – Branch Coverage
  – Path Coverage

Statement Coverage

- Statement coverage basically means that we write a set of test cases and ensure that each statement is executed at least once.
- This means there is no part of the program which is not tested.
- This is the lowest level of coverage.

Branch Coverage

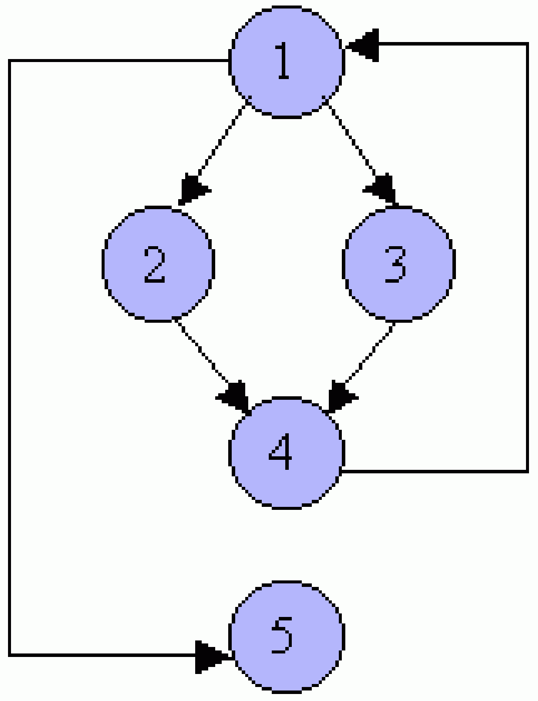
- This is the second level of coverage.
- In this case we try to identify all the branches in the program.
- And write test cases for all the branches.

Path Coverage

- This is the highest level of coverage.
- Here we have to find all the paths in the program.
- And test all the paths by writing test cases for each path.
- Let us see if we can actually do it or not with the help of an example.

Path Coverage Example

```
for (i = 0; i < N; i++) {                    //1
  if (condition1)
          // do something here               //2
  else
          // do something here               //3
  // something here                          //4
}                                            //5
```



2^N Paths

Path Coverage Example

- The total number of paths in this program are $2^{N.}$
- If the value of N is 10, we will have to run 1024 test cases.
- If the value of N is 20, we will have to run 1 million test cases.
- Can we actually run 1 million test cases??
- And this is one of the simplest program with only one If statement and only one For loop.

Only Solution FM

- Hence we are back to the same statement that no number of test cases can actually make sure that the program is bug free.
- The only solution to this is Formal Methods.
- In FM we develop models.
- These models help us argue about the correctness of the program.

End of Slides