Control and Data Flow Testing

Contents

Control Flow Testing

(Naik & Tripathy, 2008)

Steps in Control Flow Testing

1. Select the path selection criteria and the SUT
2. Draw a control flow graph
3. Select paths using the path selection criteria
4. Generate test input data
5. If the paths are not feasible, go to 3
6. Perform testing

Path Selection Criteria

All-Path

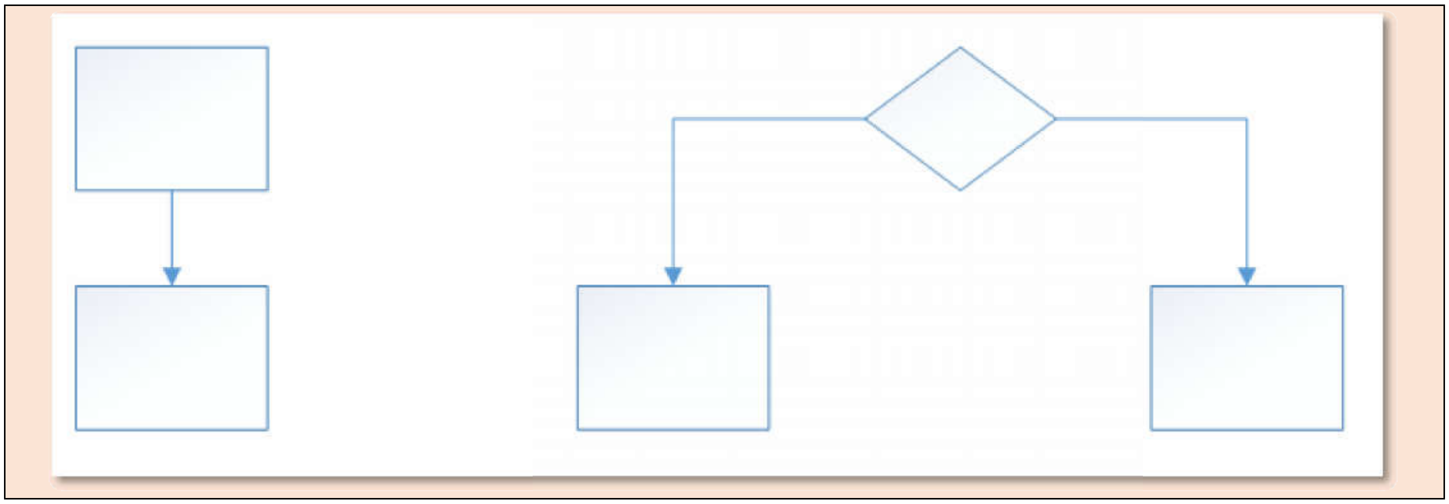
A path is a sequence of steps through the program unit from the start to the end. The all-paths criteria selects all possible path through the program unit. Selecting all paths is ideal but there may a very large number of paths making all-path selection infeasible.

Statement Coverage

Statement coverage refers the executing individual program statements and observing the outcome. If all statements are executed at least once then 100% statement coverage was achieved. Statement coverage is the weakest coverage criterion in program testing.

Branch Coverage

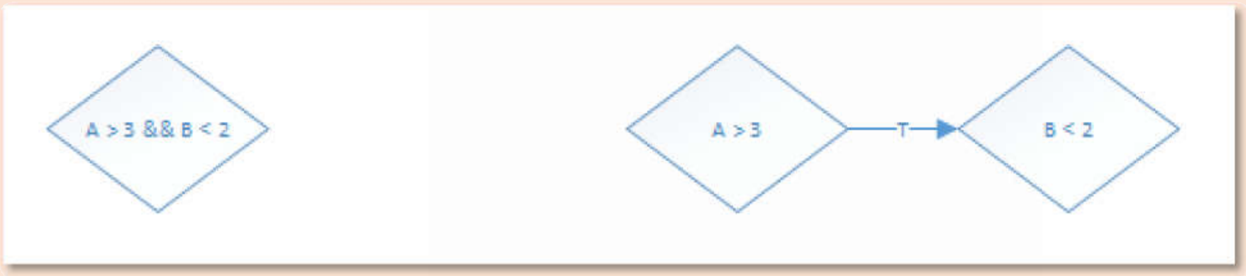
A branch is an outgoing edge of the control flow graph. Process nodes have one branch while decision nodes have two branches.



If all branches are executed at least once then 100% branch coverage was achieved.

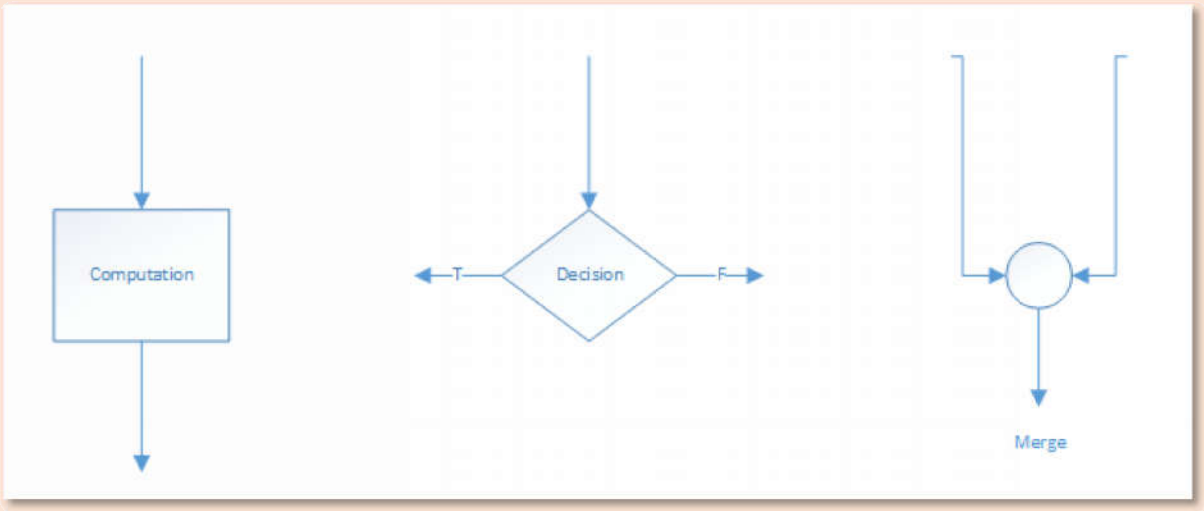
Predicate Coverage

Predicate coverage is a special form of branch coverage where each predicate is given its own decision node.



Control Flow Graph (CFG)

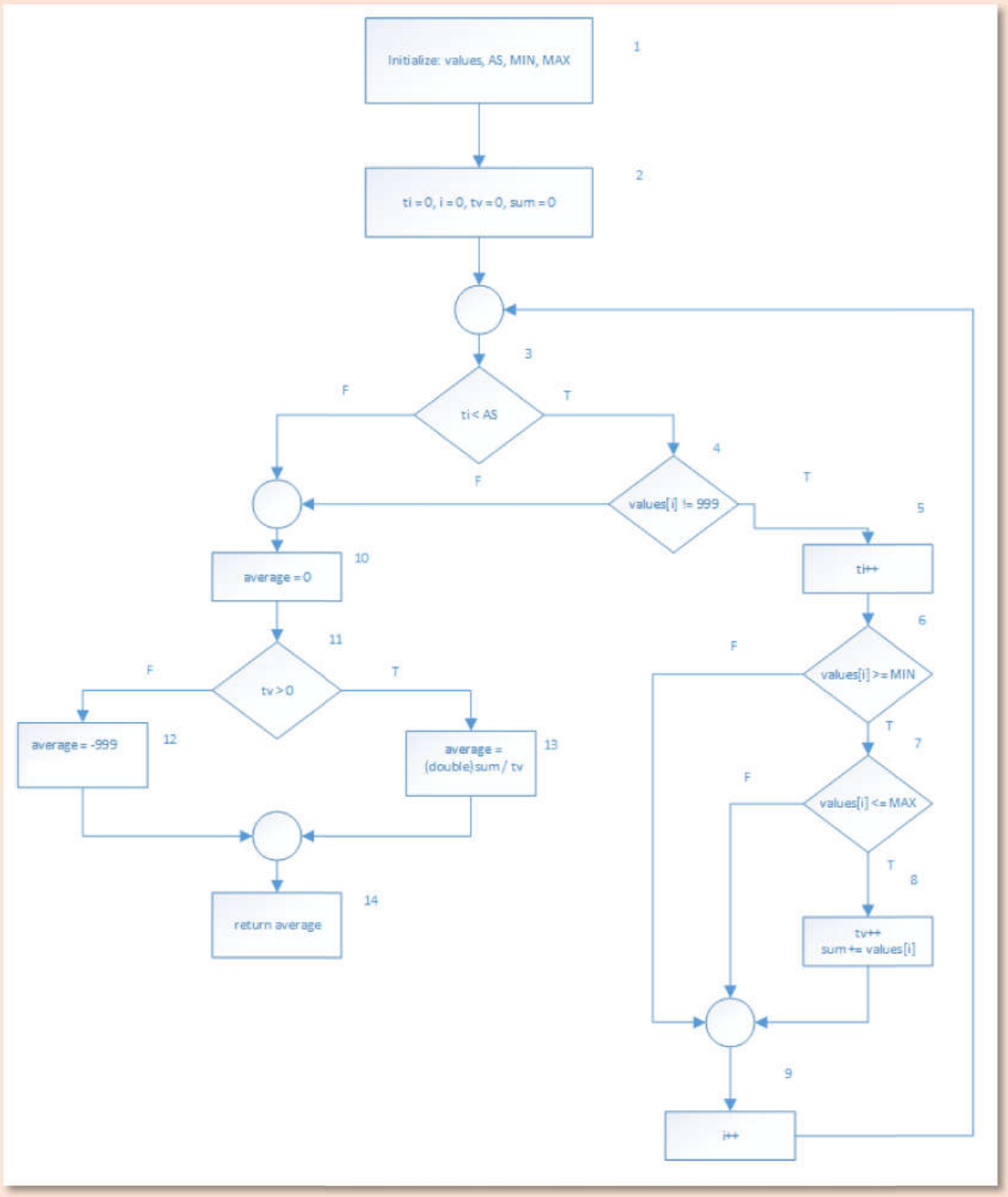
A CFG is a graphical representation of a program unit. Three symbols are used to construct the graph.



Each computation and decision node is given a unique integer. Each branch from a decision node is labeled with a ‘T’ or an ‘F’.

Example of a CFG

```
public static double ReturnAverage(int[] values, int AS, int MIN, int MAX)
{
    int ti = 0, i = 0, tv = 0, sum = 0;
    while(ti < AS && values[i] != 999)
    {
        ti++;
        if(values[i] >= MIN && values[i] <= MAX)
        {
            tv++;
            sum += values[i];
        }
        i++;
    }
    double average = 0;
    if(tv > 0)
    {
        average = (double)sum / tv;
    }
    else
    {
        average = -999;
    }
    return average;
}
```



Selecting Paths

All-Path Criteria

1-2-3(T)-4(T)-5-6(T)-7(T)-8-9-3(F)-10-11(T)-13-14

Others…

Statement Coverage

1-2-3(T)-4(T)-5-6(T)-7(T)-8-9-3(F)-10-11(T)-13-14

1-2-3(T)-4(T)-5-6(T)-7(T)-8-9-3(F)-10-11(F)-12-14

Predicate (Branch) Coverage

1-2-3(F)-10-11(F)-12-14

1-2-3(F)-10-11(T)-13-14

Others…

Generating Test Input Data

Terms

Input Vector: An input vector is a collection of data values sent into the SUT.

Predicate: An expression that evaluates to either true or false.

Path Predicate: A set of predicates associated with a path.

Predicate Interpretation: Symbolically substituting operations along a path in order to express the predicates solely in terms of the input vector and constant values.

Path Predicate Expression: An interpreted path predicate.

Example 1

1-2-3(F)-10-11(F)-12-14

<u>Predicate Interpretation Table</u>

Node	Description	Interpretation
1	`<values, AS, MIN, MAX>`	
2	`ti = 0, i = 0, tv = 0, sum = 0`	
3(F)	`ti < AS`	`0 < AS`
10	`average = 0`	
11(F)	`tv > 0`	`0 > 0`
12	`average = -999`	
14	`return average`	`return -999`

<u>Path Predicate Expression</u>

`0 < AS ≡ false -- (1)`
`0 > 0 ≡ false -- (2)`

<u>Input Vector (Test Data)</u>

`values[0] = don’t care`
`AS = -1`
`MIN = don’t care`
`MAX = don’t care`

Example 2 (infeasible path)

1-2-3(F)-10-11(T)-13-14

Predicate Interpretation Table

Node	Description	Interpretation
1	`<values, AS, MIN, MAX>`	
2	`ti = 0, i = 0, tv = 0, sum = 0`	
3(F)	`ti < AS`	`0 < AS`
10	`average = 0`	
11(T)	`tv > 0`	**`0 > 0`** (Infeasible path – cannot be true)
13	`average = (double)sum/tv`	**`average = (double)0/0`**
14	`return average`	**`return (double)0/0`**

Data Flow Testing

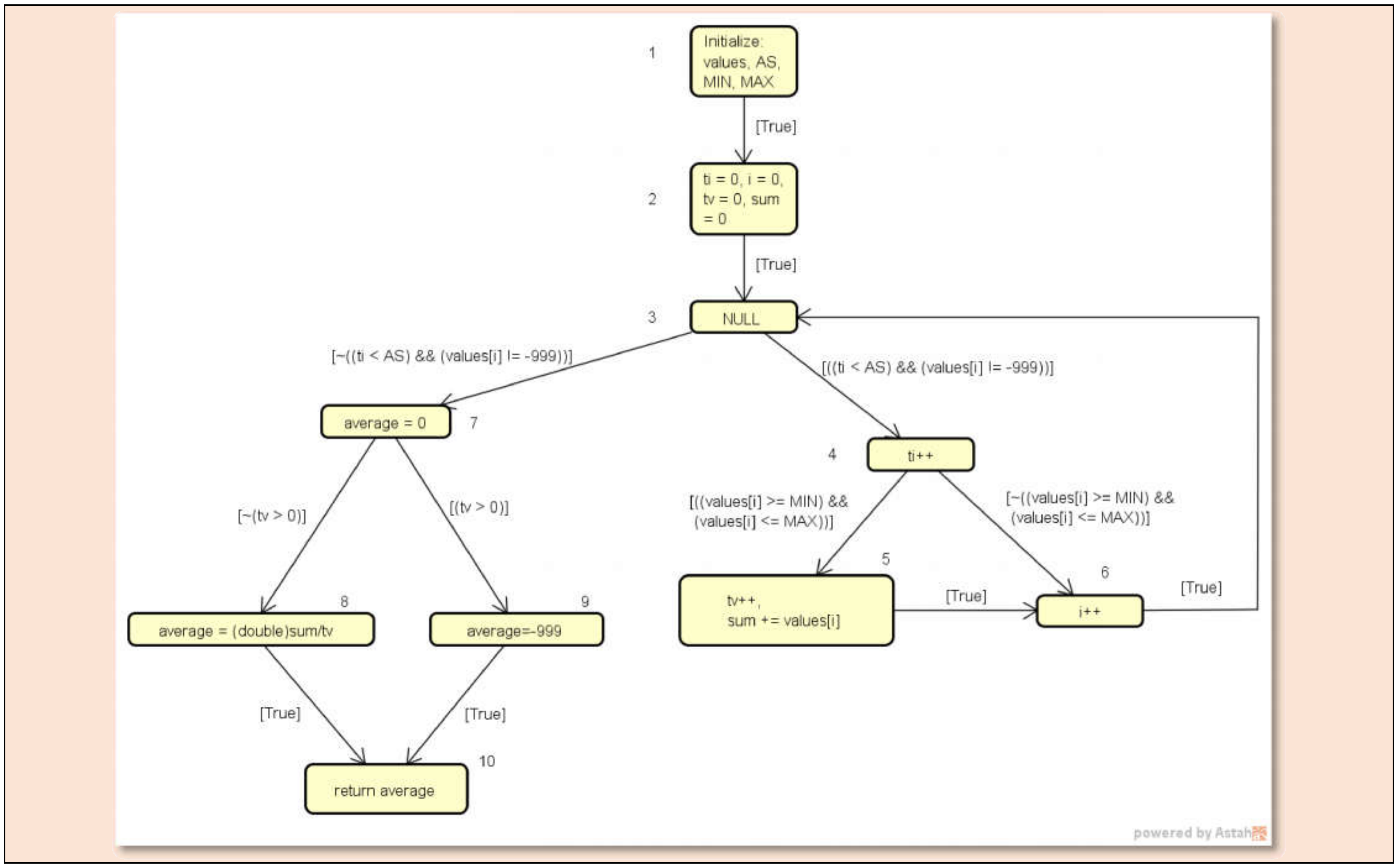
Steps in Data Flow Testing

1. Draw a data flow graph (DFG) from the SUT
2. Select one or more data flow testing criteria
3. Identify paths in the data flow graph satisfying the selection criteria
4. Derive path predicate expressions from the selected paths and solve the expressions to derive test data

The Data Flow Graph

Nodes specify a sequence of ***definitions*** and ***c-uses***. A ***definition*** of a variable occurs when it is assigned a value. A ***c-use*** of a variable occurs when the variable is used in a computation. A variable becomes ***undefined*** or ***killed*** when it becomes deallocated.

Edges specify a set of ***p-uses***. A ***p-use*** of a variable occurs when the variable is used in a predicate.


1
Initialize: values, AS, MIN, MAX
[True]
2
ti = 0, i = 0, tv = 0, sum = 0
[True]
3
NULL
[~((ti < AS) && (values[i] != -999))]
[((ti < AS) && (values[i] != -999))]
average = 0
7
4
ti++
[~(tv > 0)]
[(tv > 0)]
[((values[i] >= MIN) && (values[i] <= MAX))]
[~((values[i] >= MIN) && (values[i] <= MAX))]
5
6
tv++, sum += values[i]
[True]
i++
[True]
8
9
average = (double)sum/tv
average=-999
[True]
[True]
10
return average
powered by Astah

Data Flow Testing Criteria

All-defs

For each variable x and for each node i, such that x has a ***global definition*** in node i, select a ***complete path*** which includes a ***def-clear*** path from node i to:

- Node j having a ***global c-use*** of x or
- Edge (j, k) having a p-use of x

Global Definition: A node has a global definition of a variable if the variable is defined in that node and there is a ***def-clear*** path of that variable to another node or edge where the variable is used.

Complete Path: A path from the entry node to the exit node.

Def-clear path: A path is a def-clear path if there is no definition or undefinition of the given variable.

Global c-use: A variable has global c-use in a node if it has been defined in an earlier node.

All-c-uses

For each variable x and for each node i, such that x has a global definition in node i, select complete paths which include def-clear paths from node i to all nodes j such that there is a global c-use of x in j.

All-p-uses

For each variable x and for each node i such that x has a global definition in node i, select complete paths which include def-clear paths from node i to all edges (j, k) such that there is a p-use of x on edge (j, k).

All-p-uses/Some-c-uses

Identical to all-p-uses except if x has no p-use, then this reduces to ***some-c-uses***.

> ***Some-c-uses***: For each variable x and for each node i such that x has a global definition in node j, select complete paths which include def-clear paths from node i to *some* nodes j such that there is a global c-use of x in node j.

All-c-uses/Some-p-uses

Identical to all-c-uses except if x has no global c-use, then this reduces to ***some-p-uses***.

> ***Some-p-uses***: For each variable x and for each node i such that x has a global definition in node i, select complete paths which include def-clear paths from node i to some edges (j, k) such that there is a p-use of x on edge (j, k).

All-uses

The conjunction of off all-p-uses and all c-uses.

All-du-paths

For each variable x and each node i such that x has a global definition in node i, select complete paths which include all ***du-paths*** from node i:

- To all nodes j such that there is a global c-use of x in j and
- To all edges (j, k) such that there is a p-use of x on (j, k)

Du-paths: A path is a definition-use path w.r.t. variable x if the first node has a global definition of x and either:

- The last node has a global c-use of x and the path is a def-clear ***simple path*** w.r.t. x or
- The last edge has a p-use of x and the path is a def-clear, loop-free path w.r.t. x.

Simple path: A path in which all nodes, expect possibly the first and last node are distinct.

Loop-Free path: A path in which all nodes are distinct.

Def() and c-use() Sets of Nodes

Node i	def(i)	c-use(i)
1	`{values, AS, MIN, MAX}`	`{}`
2	`{ti, i, tv, sum}`	`{}`
3	`{}`	`{}`
4	`{ti}`	`{ti}`
5	`{tv, sum}`	`{tv, i, sum, value}`
6	`{i}`	`{i}`
7	`{average}`	`{}`
8	`{average}`	`{sum, tv}`
9	`{average}`	`{}`
10	`{}`	`{average}`

Predicates and p-use() Set of Edges

Edges (i, j)	predicate(i, j)	p-use(i, j)
(1, 2)	`True`	`{}`
(2, 3)	`True`	`{}`
(3, 4)	`((ti < AS) && (values[i] != -999))`	`{ti, AS, values}`
(4, 5)	`((values[i] >= MIN) && (values[i] <= MAX))`	`{values, MIN, MAX}`
(4, 6)	`~((values[i] >= MIN) && (values[i] <= MAX))`	`{values, MIN, MAX}`
(5, 6)	`True`	`{}`
(6, 3)	`True`	`{}`
(3, 7)	`~((ti < AS) && (values[i] != -999))`	`{ti, AS, values}`
(7, 8)	`~(tv > 0)`	`{tv}`
(7, 9)	`(tv > 0)`	`{tv}`
(8, 10)	`True`	`{}`
(9, 10)	`True`	`{}`

All-defs on `tv`

Global definitions: 2, 5

Global c-use: 5

  Def-clear path: 2-3-4-5

  Complete path: 1-2-3-4-5-6-3-7-9-10

P-uses: (7, 8)

  Def-clear path: 2-3-7-8

  Complete path: 1-2-3-7-8-10

Global c-use: 8

Def-clear path: 5-6-3-7-8

Complete path: 1-2-3-4-<u>5-6-3-7-8</u>-10

All-c-uses on `ti`

Global definitions: 2, 4

Node 2: Global c-use: 4

Node 4: There is no c-use that corresponds

Def-clear path: 2-3-4

Complete paths:

- **1-<u>2-3-4</u>-5-6-3-7-8-10**
- **1-<u>2-3-4</u>-6-3-7-8-10**
- **1-<u>2-3-4</u>-5-6-3-7-9-10**
- **1-<u>2-3-4</u>-6-3-7-8-10**

All-p-uses on `tv`

Global definitions: 2, 5

Node 2: p-uses: (7, 8) and (7, 9)

Def-clear paths: 2-3-7-8, 2-3-7-9

Node 5: p-uses: (7, 8) and (7, 9)

Def-clear paths: 5-6-3-7-8, 5-6-3-7-9

Complete paths:

- **1-<u>2-3-7-8</u>-10**
- **1-<u>2-3-7-9</u>-10**
- **1-2-3-4-<u>5-6-3-7-8</u>-10**
- **1-2-3-4-<u>5-6-3-7-9</u>-10**

All-p-uses/Some-c-uses on `i`

Global definitions: 2, 6

No p-use of i → c-use of i

Node 2: c-use in 6

Def-clear path: 2-3-4-6

Complete path: 1-<u>2-3-4-6</u>-3-7-8-10

All-c-uses/Some-p-use on **AS**

Global definition: 1

No global c-use of AS → p-use of AS

Edges: (3, 4) and (3, 7)

Def-clear paths: 1-2-3-4, 1-2-3-7

Complete path: $\underline{\text{1-2-3-4}}$-5-6-3-7-8-10