NEC
Empowered by Innovation

University of Waterloo

Software Testing and Quality Assurance
Theory and Practice
Chapter 4
Control Flow Testing

Outline of the Chapter

- Basic Idea
- Outline of Control Flow Testing
- Control Flow Graph
- Paths in a Control Flow Graph
- Path Selection Criteria
- Generating Test Input
- Containing Infeasible Paths
- Summary

Basic Idea

- Two kinds of basic program statements:
  - Assignment statements (Ex. x = 2*y;)
  - Conditional statements (Ex. if(), for(), while(), …)
- Control flow
  - Successive execution of program statements is viewed as flow of control.
  - Conditional statements alter the default flow.
- Program path
  - A program path is a sequence of statements from entry to exit.
  - There can be a large number of paths in a program.
  - There is an (input, expected output) pair for each path.
  - Executing a path requires invoking the program unit with the right test input.
  - Paths are chosen by using the concepts of path <u>selection criteria</u>.
- Tools: Automatically generate test inputs from program paths.

Outline of Control Flow Testing

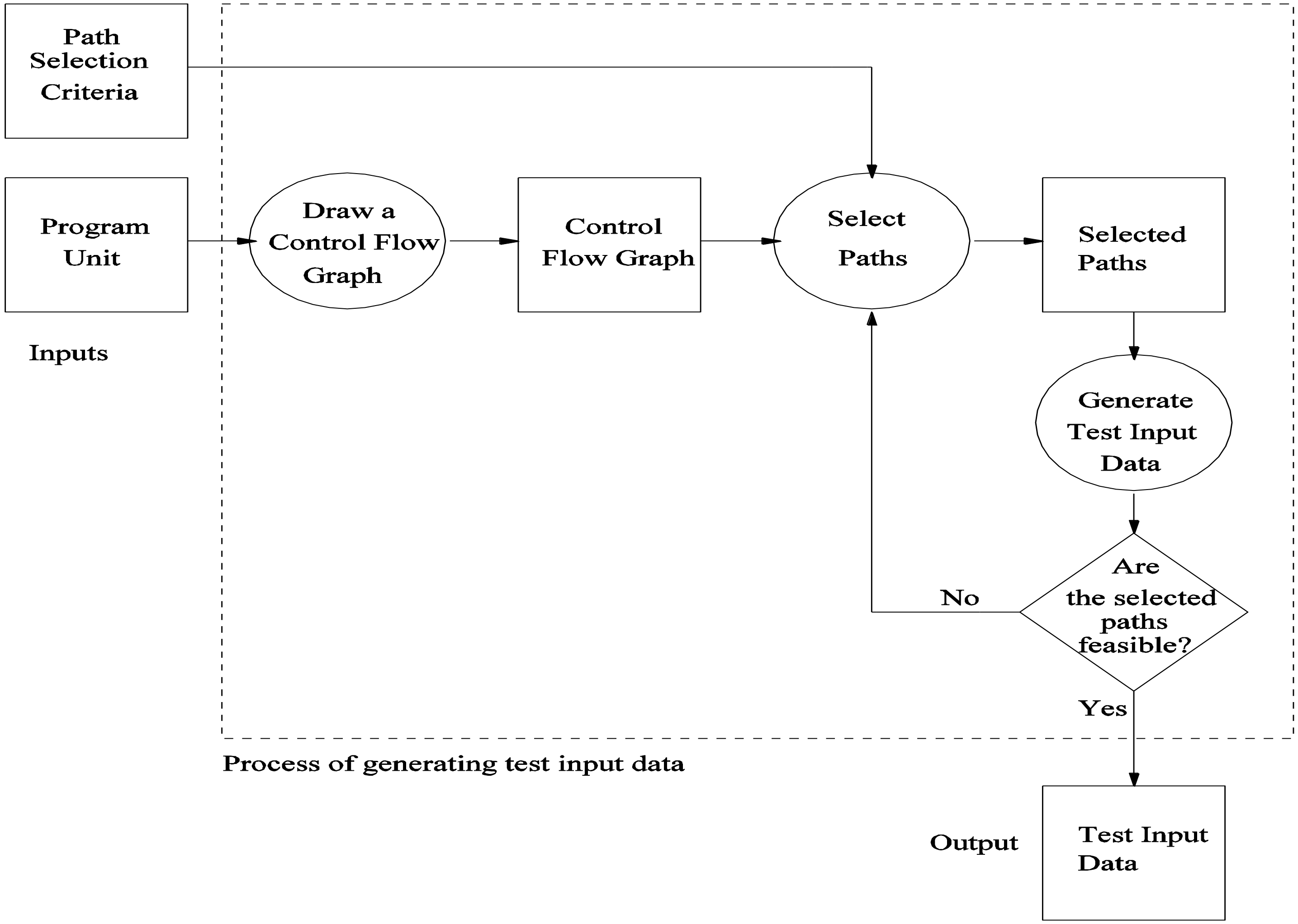

Figure 4.1: The process of generating test input data for control flow testing.

Outline of Control Flow Testing

- Inputs to the test generation process
  - Source code
  - Path selection criteria: statement, branch, …
- Generation of control flow graph (CFG)
  - A CFG is a graphical representation of a program unit.
  - Compilers are modified to produce CFGs. (You can draw one by hand.)
- Selection of paths
  - Enough entry/exit paths are selected to satisfy path selection criteria.
- Generation of test input data
  - Two kinds of paths
    - Executable path: There exists input so that the path is executed.
    - Infeasible path: There is no input to execute the path.
  - Solve the path conditions to produce test input for each path.

Control Flow Graph

- Symbols in a CFG

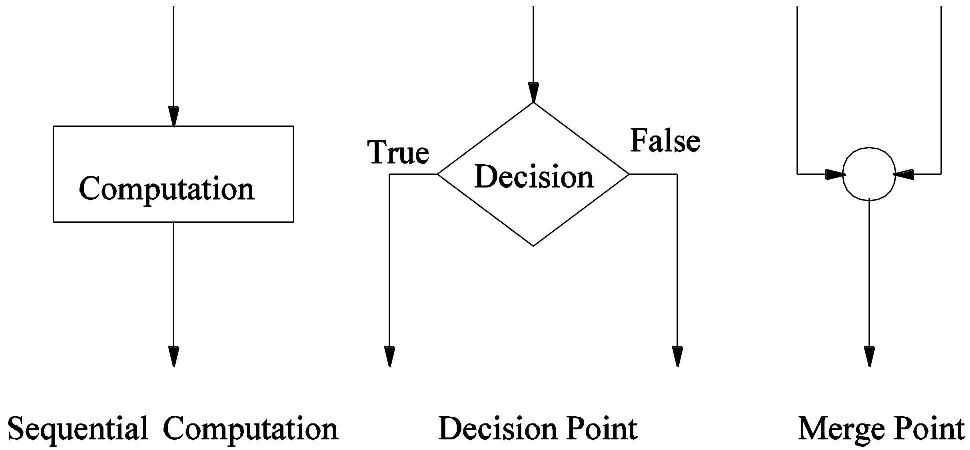

Figure 4.2: Symbols in a control flow graph

Control Flow Graph

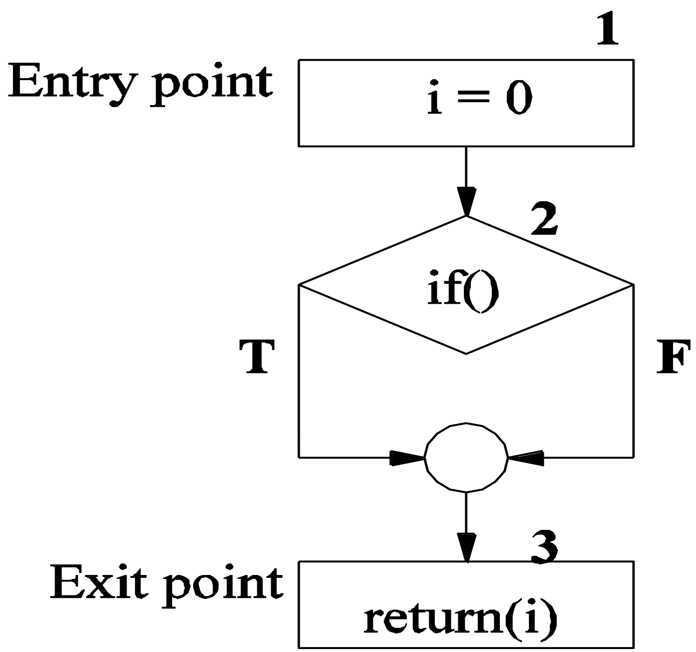

Figure 4.4: A high-level CFG representation of openfiles().

Control Flow Graph

- **Example code: ReturnAverage()**

```
public static double ReturnAverage(int value[],  int AS, int MIN, int MAX){
   /* Function: ReturnAverage  Computes the  average of all  those  numbers in the  input array  in
      the  positive  range  [MIN, MAX]. The  maximum size  of the array is AS. But, the  array size
      could be smaller than AS in which case the end of input is represented by -999. */

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

Figure 4.6: A function to compute the average of selected integers in an array.

Control Flow Graph

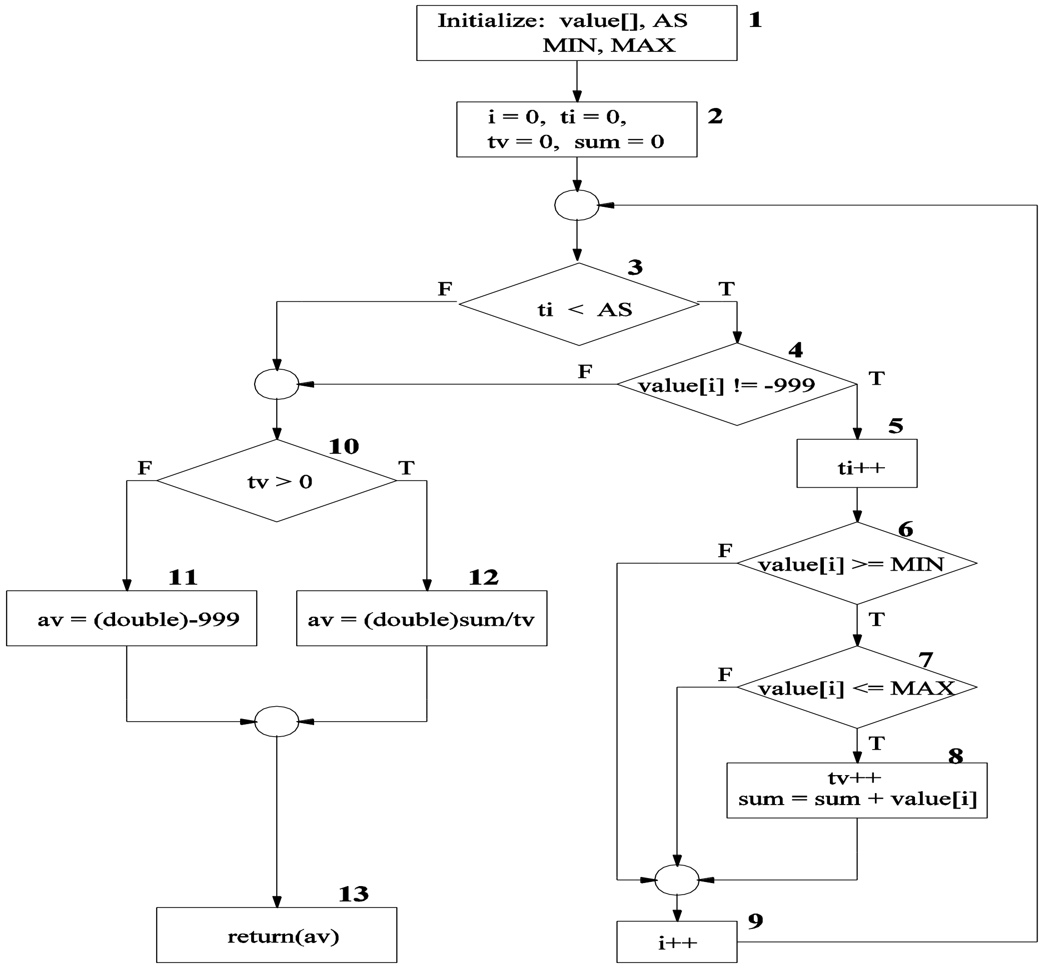

Figure 4.7: A CFG representation of ReturnAverage().

Path Selection Criteria

- Statement coverage criterion
  - Statement coverage means executing individual program statements and observing the output.
  - 100% statement coverage means all the statements have been executed at least once.
    - Cover all assignment statements.
    - Cover all conditional statements.
  - Less than 100% statement coverage is unacceptable.

SCPath1	1-2-3(F)-10(F)-11-13
SCPath2	1-2-3(T)-4(T)-5-6(T)-7(T)-8-9-3(F)-10(T)-12-13

Table 4.4: Paths for statement coverage of the CFG of Figure 4.7.

Path Selection Criteria

- Branch coverage criterion
  - A branch is an outgoing edge from a node in a CFG.
    - A condition node has two outgoing branches – corresponding to the True and False values of the condition.
  - Covering a branch means executing a path that contains the branch.
  - 100% branch coverage means selecting a set of paths such that each branch is included on some path.

Path Selection Criteria

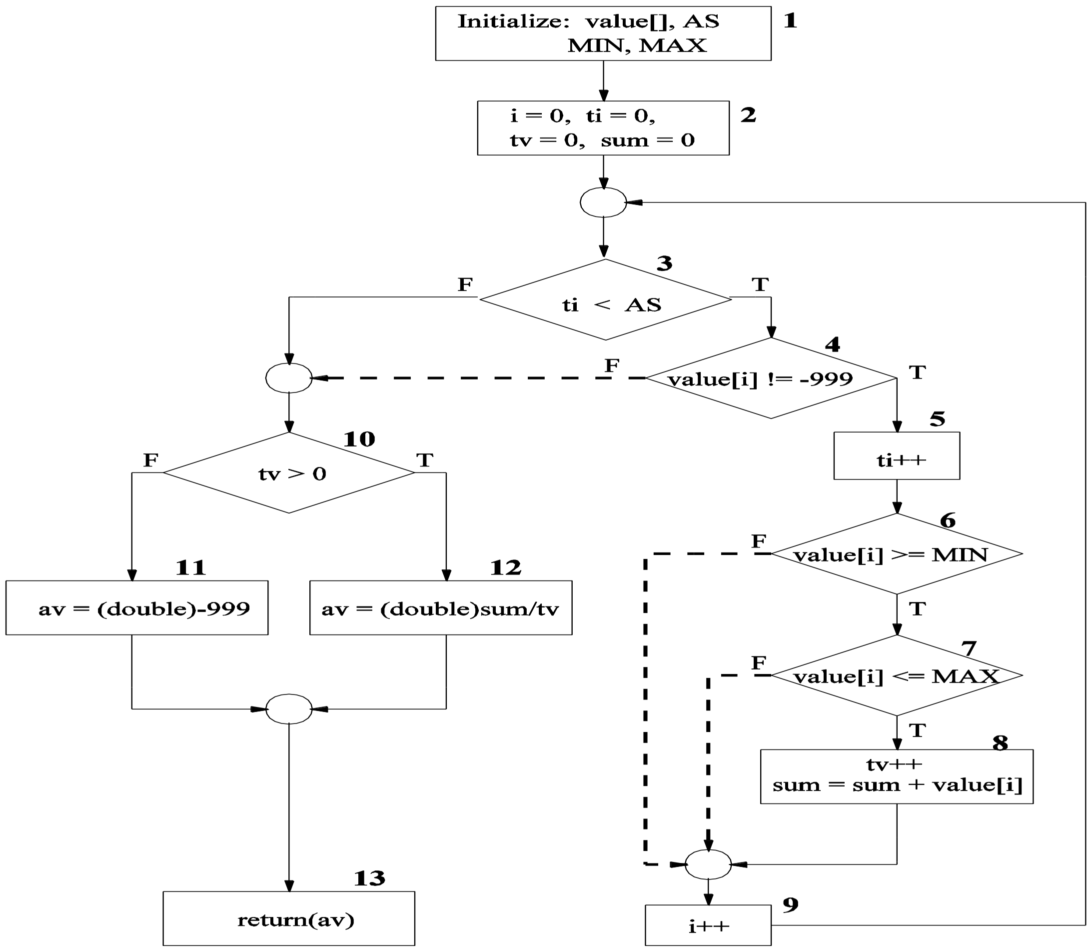

Figure 4.8: The dotted arrows represent the branches not covered by the statement covering in Table 4.4.

Path Selection Criteria

- Branch coverage criterion
  - A branch is an outgoing branch (edge) from a node in a CFG.
    - A condition node has two outgoing branches – corresponding to the True and False values of the condition.
  - Covering a branch means executing a path that contains the branch.
  - 100% branch coverage means selecting a set of paths such that each branch is included on some path.

BCPath 1	1-2-3(F)-10(F)-11-13
BCPath 2	1-2-3(T)-4(T)-5-6(T)-7(T)-8-9-3(F)-10(T)-12-13
BCPath 3	1-2-3(T)-4(F)-10(F)-11-13
BCPath 4	1-2-3(T)-4(T)-5-6(F)-9-3(F)-10(F)-11-13
BCPath 5	1-2-3(T)-4(T)-5-6(T)-7(F)-9-3(F)-10(F)-11-13

Table 4.5: Paths for branch coverage of the flow graph of Figure 4.7.

Path Selection Criteria

- Predicate coverage criterion
  - If all possible combinations of truth values of the conditions affecting a path have been explored under some tests, then we say that predicate coverage has been achieved.

Path Selection Criteria

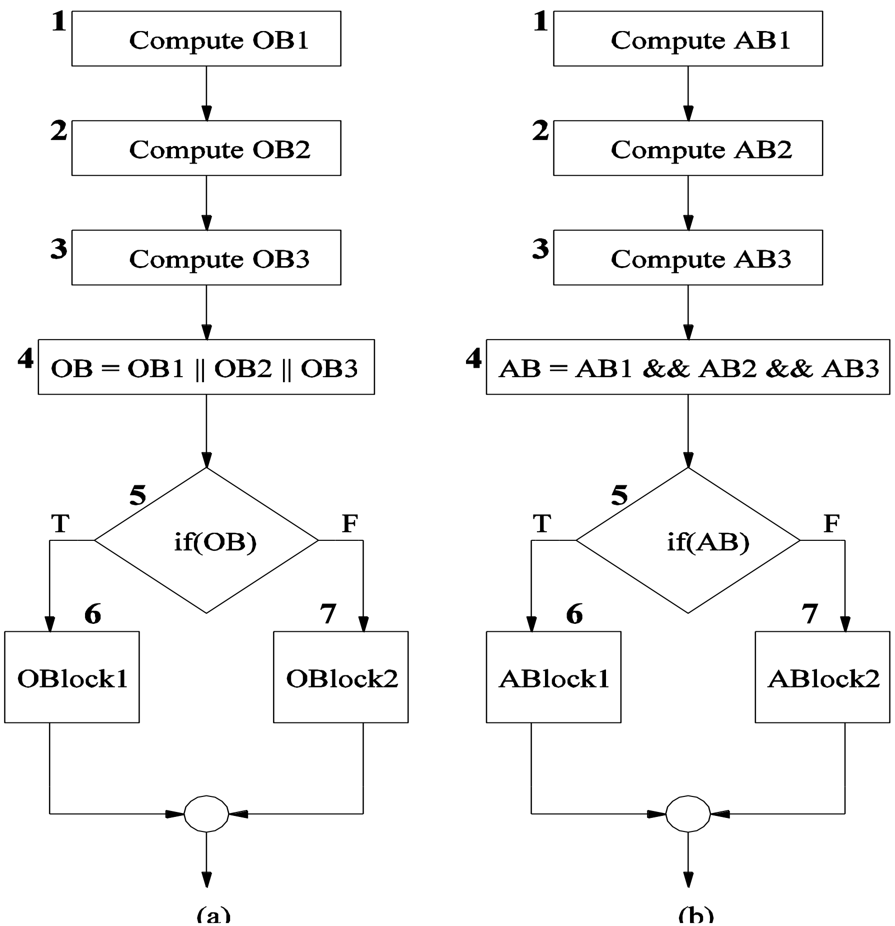

Figure 4.9: Partial control flow graph with (a) OR operation and (b) AND operation.

Generating Test Input

- Having identified a path, a key question is how to make the path execute, if possible.
  - Generate input data that satisfy all the conditions on the path.
- Key concepts in generating test input data
  - Input vector
  - Predicate
  - Path condition
  - Predicate interpretation
  - Path predicate expression
  - Generating test input from path predicate expression

Generating Test Input

- Input vector
  - An input vector is a collection of all data entities read by the routine whose values must be fixed prior to entering the routine.
  - Members of an input vector can be as follows.
    - Input arguments to the routine
    - Global variables and constants
    - Files
    - Contents of registers (in Assembly language programming)
    - Network connections
    - Timers
  - Example: An input vector for openfiles() consists of individual presence or absence of the files "files1," "file2," and "file3."
  - Example: The input vector of ReturnAverega() shown in Figure 4.6 is <value[], AS, MIN, MAX>.

Generating Test Input

- Predicate
  - A predicate is a logical function evaluated at a decision point.
  - Example: ti < AS is a predicate in node 3 of Figure 4.7.
  - Example: The construct OB is a predicate in node 5 in Figure 4.9.
- Path predicate
  - A path predicate is the set of predicates associated with a path.
  - **Figure 4.10:** An example path from Fig. 4.7:
    - 1-2-3(T)-4(T)-5-6(T)-7(T)-8-9-3(F)-10(T)-12-13.
  - **Figure 4.11:** The path predicate for the path shown in Figure 4.10.

ti < AS	≡ True
value[i] != -999	≡ True
value[i] >= MIN	≡ True
value[i] <= MAX	≡ True
ti < AS	≡ False
tv > 0	≡ True

Generating Test Input

- Predicate interpretation
  - A path predicate may contain local variables.
  - Example: <i, ti, tv> in Figure 4.11 are local variables.
  - Local variables play no role in selecting inputs that force a path to execute.
  - Local variables can be eliminated by a process called **symbolic execution**.
  - Predicate interpretation is defined as the process of
    - symbolically substituting operations along a path in order to express the predicate solely in terms of the input vector and a constant vector.
  - A predicate may have different interpretations depending on how control reaches the predicate.

Generating Test Input

- Path predicate expression
  - An interpreted path predicate is called a path predicate expression.
  - A path predicate expression has the following attributes.
    - It is void of local variables.
    - It is a set of constraints in terms of the input vector, and, maybe, constants.
    - Path forcing inputs can be generated by solving the constraints.
    - If a path predicate expression has no solution, the path is infeasible.
  - **Figure 4.13:** Path predicate expression for the path shown in Figure 4.10.

0 < AS	≡ True	…… (1)
value[0] != -999	≡ True	…… (2)
value[0] >= MIN	≡ True	…… (3)
value[0] <= MAX	≡ True	…… (4)
1 < AS	≡ False	…… (5)
1 > 0	≡ True	…… (6)

Generating Test Input

- Path predicate expression
  - An example of infeasible path
  - **Figure 4.14:** Another example of path from Figure 4.7.
    - 1-2-3(T)-4(F)-10(T)-12-13
  - **Figure 4.15:** Path predicate expression for the path shown in Figure 4.14.

0 < AS	≡ True	…… (1)
value[0] != -999	≡ True	…… (2)
0 > 0	≡ True	…… (3)

Generating Test Input

- Path predicate expression (An example of infeasible path)

1-2-3(T)-4(F)-10(T)-12-13.

Figure 4.14 Another example path from Figure 4.7.

TABLE 4.8 Interpretation of Path Predicate of Path in Figure 4.14.

Node	Node Description	Interpreted Description
1	Input vector: < value[], AS, MIN, MAX >	
2	i = 0, ti = 0, tv = 0, sum = 0	
3(T)	ti < AS	0 < AS
4(F)	value[i]! = −999	value[0]! = −999
10(T)	tv > 0	0 > 0
12	av = (double)sum/tv	av = (double)value[0]/0
13	return(av)	return((double) value[0]/0)

Note: The bold entries in column 1 denote interpreted predicates.

Generating Test Input

- Generating input data from a path predicate expression
  - Consider the path predicate expression of Figure 4.13 (reproduced below.)

0 < AS	≡ True	…… (1)
value[0] != -999	≡ True	…… (2)
value[0] >= MIN	≡ True	…… (3)
value[0] <= MAX	≡ True	…… (4)
1 < AS	≡ False	…… (5)
1 > 0	≡ True	…… (6)

  - One can solve the above equations to obtain the following test input data

AS	= 1
MIN	= 25
MAX	= 35
Value[0]	= 30

  - Note: The above set is not unique.

Containing Infeasible Paths

- A program unit may contain a large number of paths.
  - Path selection becomes a problem. Some selected paths may be infeasible.
  - Apply a path selection strategy:
    - Select as many short paths as possible.
    - Choose longer paths.
  - There are efforts to write code with fewer/no infeasible paths.

Summary

- Control flow is a fundamental concept in program execution.
- A program path is an instance of execution of a program unit.
- Select a set of paths by considering path **selection criteria**.
    - Statement coverage
    - Branch coverage
    - Predicate coverage
    - All paths
- From source code, derive a CFG (compilers are modified for this.)
- Select paths from a CFG based on path selection criteria.
- Extract path predicates from each path.
- Solve the path predicate expression to generate test input data.
- There are two kinds of paths.
    - feasible
    - infeasible