

Software Testing

Structural (White Box) Testing

Lecture 1(control flow testing)

May 29, 2020

KIT
Department of Software Engineering

White Box testing

White-box testing(also called clear box testing, glass box testing, transparent box testing, or structural testing)is a method of testing software that tests internal structures or workings of an application[the tester has access to the internal data structures and algorithms including the code that implement these.

Using the White Box Approach to Test Case Design:

White box testing methods are especially useful for revealing design and code-based control, logic and sequence defects, initialization defects, and data flow defects.

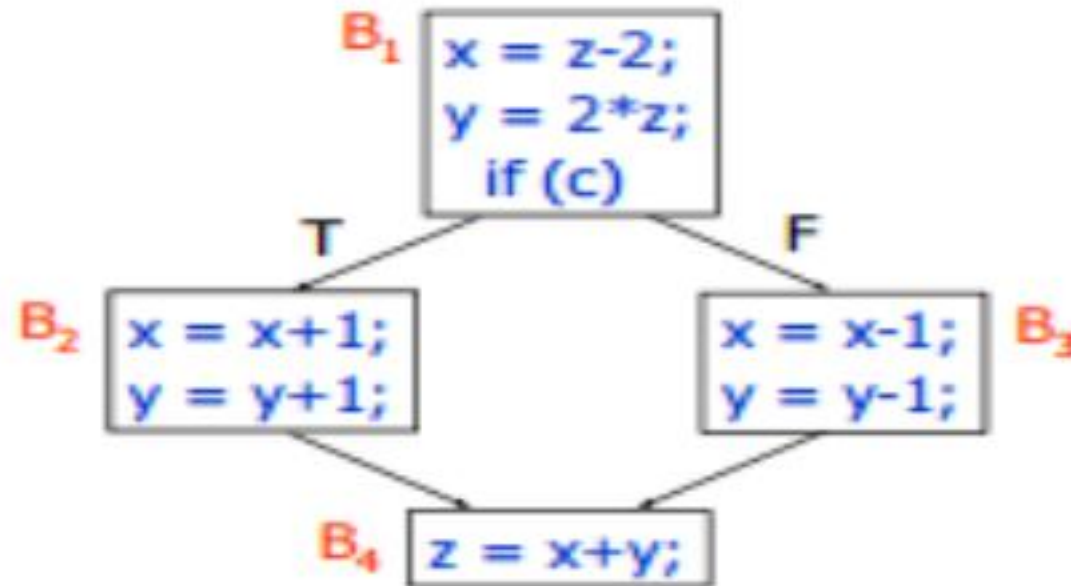
Control flow refers to flow of control from one instruction to another

CFG Example

Program

```
x = z-2 ;  
y = 2*z;  
if (c) {  
    x = x+1;  
    y = y+1;  
}  
else {  
    x = x-1;  
    y = y-1;  
}  
z = x+y;
```

Control Flow Graph



Outline of Control Flow Testing

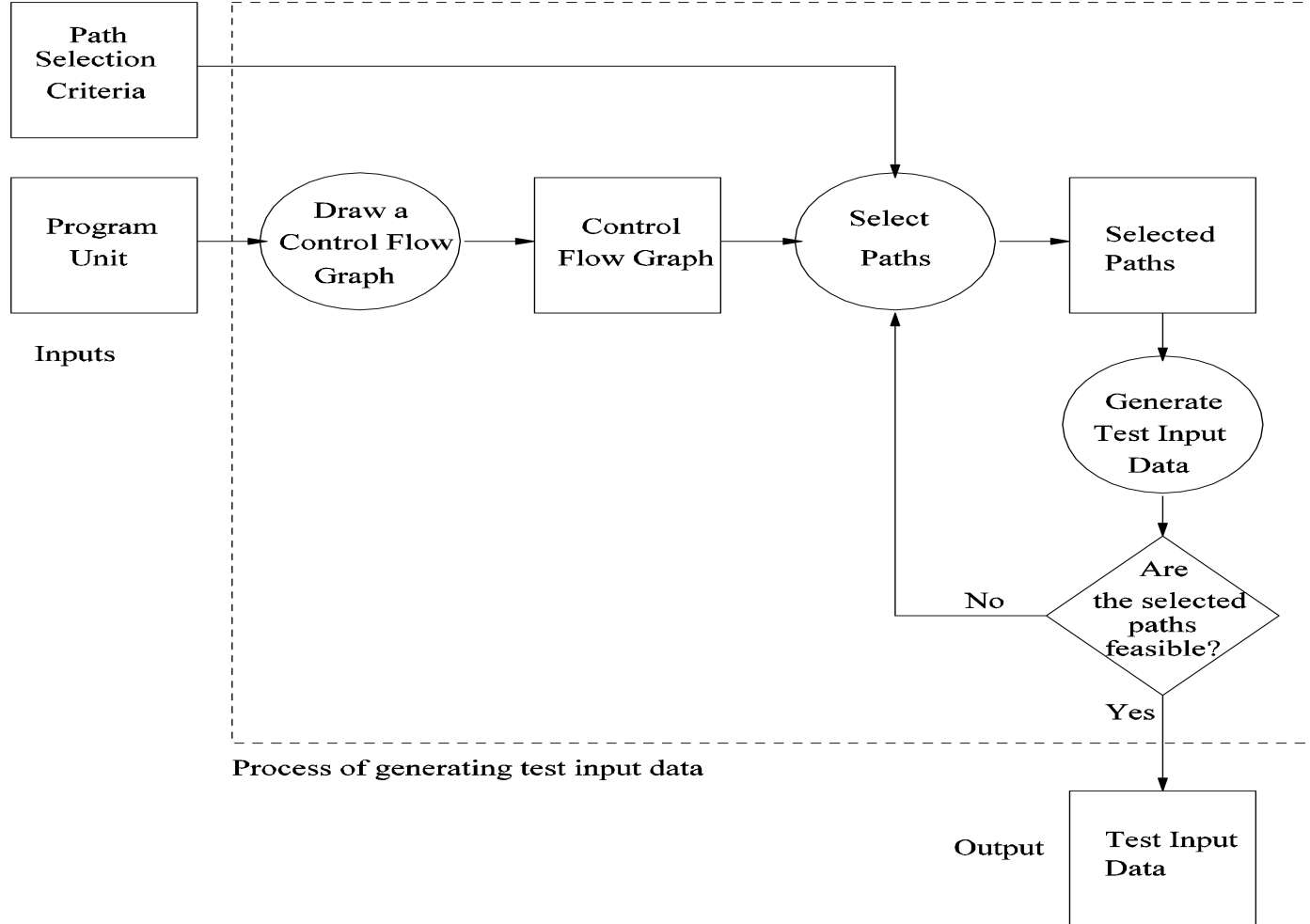


Figure 1: The process of generating test input data for control flow testing.

Basic Idea

- Two kinds of basic program statements:
 - Assignment statements (Ex. $x = 2*y;$)
 - Conditional statements (Ex. `if()`, `for()`, `while()`, ...)
- 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 selection criteria.
- Tools: Automatically generate test inputs from program paths.

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
 - Feasible path/ 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.

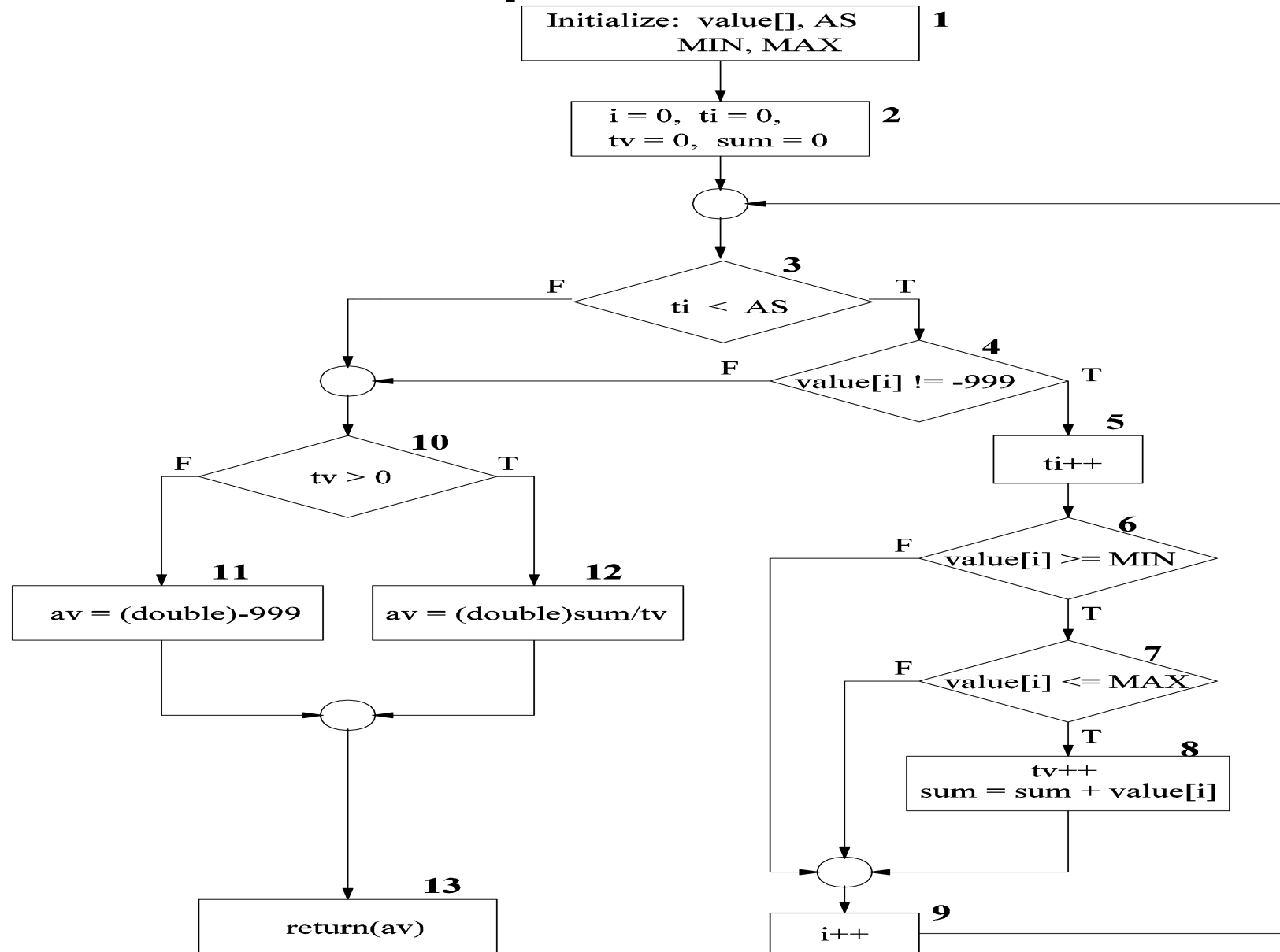
Control Flow Graph

- **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);
}
```

Control Flow Graph



5/29/2020 Figure 2: A CFG representation of `ReturnAverage()`.

Paths in a Control Flow Graph

- A few paths in Figure 2.
 - Path 1: 1-2-3(F)-10(T)-12-13
 - Path 2: 1-2-3(F)-10(F)-11-13
 - Path 3: 1-2-3(T)-4(T)-5-6(T)-7(T)-8-9-3(F)-10(T)-12-13
 - Path 4: 1-2-3(T)-4(T)-5-6-7(T)-8-9-3(T)-4(T)-5-6(T)-7(T)-8-9-3(F)-10(T)-12-13

Path Selection Criteria

- Program paths are selectively executed.
- Question: What paths do I select for testing?
- The concept of *path selection criteria* is used to answer the question.
- Advantages of selecting paths based on defined criteria:
 - Ensure that all program constructs are executed at least once. (The programmer needs to observe the outcome of executing each program construct)
 - Repeated selection of the same path is avoided. (Executing the same path several times is a waste of resources)
 - One can easily identify what features have been tested and what not.
- Path selection criteria
 1. Select all paths.
 2. Select paths to achieve complete statement coverage.
 3. Select paths to achieve complete branch coverage.
 4. Select paths to achieve predicate coverage.

Path Selection Criteria

1. All-path coverage criterion: Select all the paths in the program unit .

Advantage:

- it is desirable since it can detect error.

Disadvantages

- It's difficult to select all possible paths in program in practice.
- Producing all the inputs that exercise all the program paths is difficult process.

Path Selection Criteria

2. 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.
- Example of Paths selected based on statement coverage criterion

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 1: Paths for statement coverage of the CFG of Figure 2.

Path Selection Criteria

3. 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

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

Path Selection Criteria

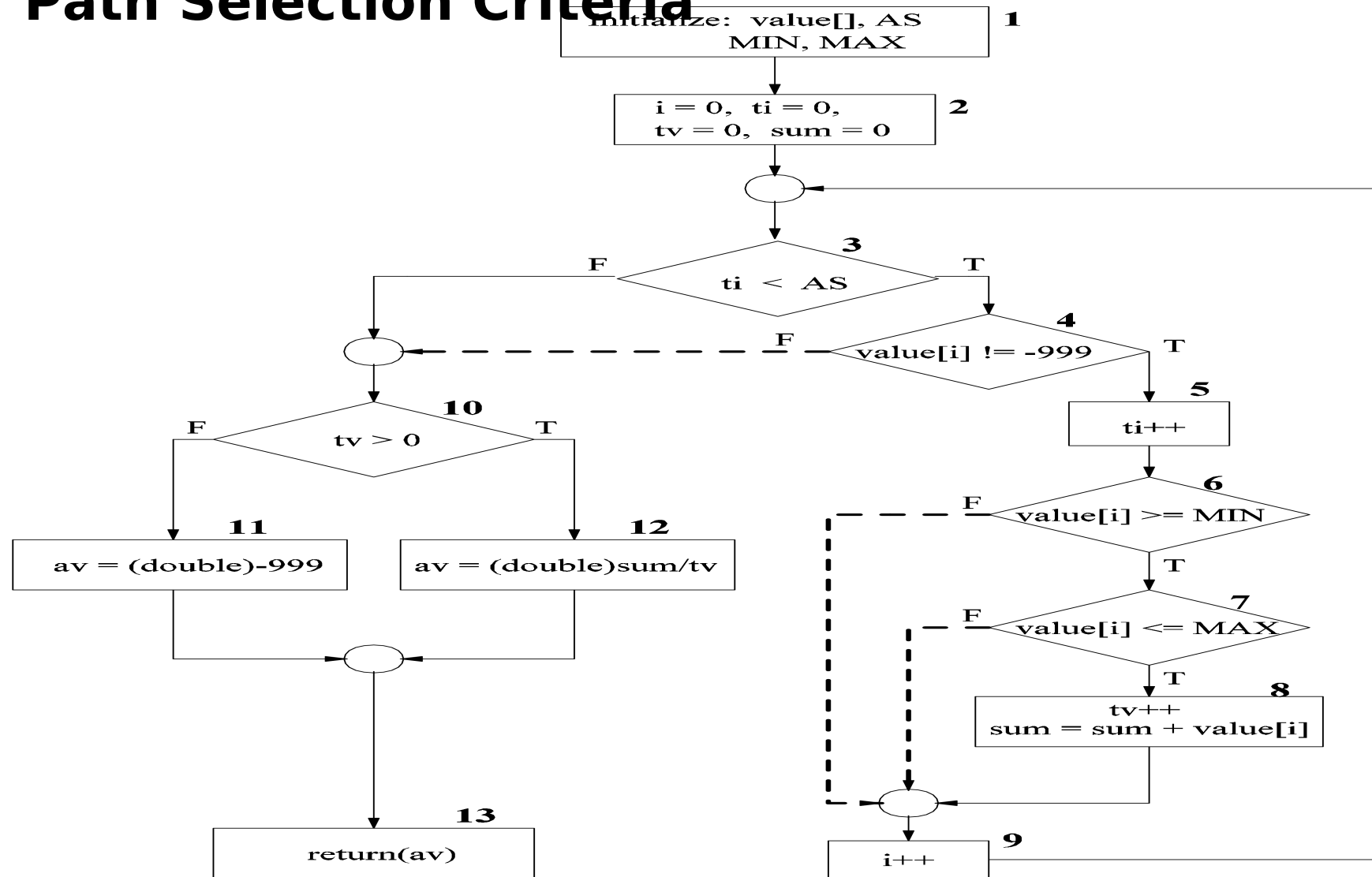


Figure 2: The dotted arrows represent the branches not covered by the statement covering in Table 1.

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
 1. Input vector
 2. Predicate
 3. Path condition
 4. Predicate interpretation
 5. Path predicate expression
 6. Generating test input from path predicate expression

Generating Test Input

1-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 2 is `<value[], AS, MIN, MAX>`.

Generating Test Input

2- Predicate

- A predicate is a logical function evaluated at a decision point.
- Example of a **predicate**: $ti < AS$ is a predicate in node 3 of Figure 2.
- Example of a **predicate**: $Value[i] \leq Max$ is a predicate in node 7 of Figure 2.

3- Path predicate

- A path predicate is the set of predicates associated with a path.
- An example **path** from Fig. 2:
 - 1-2-3(T)-4(T)-5-6(T)-7(T)-8-9-3(F)-10(T)-12-13.
- The example of **path predicate** for the **path** shown in Figure 2.

$ti < AS \equiv \text{True}$

$value[i] \neq -999 \equiv \text{True}$

$value[i] \geq MIN \equiv \text{True}$

$value[i] \leq MAX \equiv \text{True}$

$ti < AS \equiv \text{False}$

$tv > 0 \equiv \text{True}$

Generating Test Input

Predicate interpretation

- A path predicate may contain local variables.
- Example: $\langle i, ti, tv \rangle$ 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 only in terms of the input vector and a constant vector.
- A predicate may have different interpretations depending on how control reaches the predicate.
- Examples of **Predicate interpretation**
 - ♣ Example#1:
The predicate $ti < AS$ can be rewritten as $0 < AS$
 - ♣ Example#2:
The predicate $Value[i] \geq MIN$ can be rewritten as $Value[0] \geq MIN$

Generating Test Input

5- 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.
- example of **Path predicate expression** for the path shown in Figure 2.

$0 < AS \quad \equiv \text{True} \quad \dots\dots (1)$

$\text{value}[0] \neq -999 \quad \equiv \text{True} \quad \dots\dots (2)$

$\text{value}[0] \geq \text{MIN} \quad \equiv \text{True} \quad \dots\dots (3)$

$\text{value}[0] \leq \text{MAX} \quad \equiv \text{True} \quad \dots\dots (4)$

$1 < AS \quad \equiv \text{False} \quad \dots\dots (5)$

$1 > 0 \quad \equiv \text{True} \quad \dots\dots (6)$

Generating Test Input

- Path predicate expression
 - An example of infeasible path
 - Another example of path from Figure 2.
 - 1-2-3(T)-4(F)-10(T)-12-13
 - Path predicate expression for the path shown in Figure 2.
 - $0 < AS \quad \equiv \text{True} \quad \dots\dots (1)$
 - $\text{value}[0] \neq -999 \quad \equiv \text{True} \quad \dots\dots (2)$
 - $0 > 0 \quad \equiv \text{True} \quad \dots\dots (3)$
- 1-2-3(T)-4(F)-10(T)-12-13 is infeasible because the Path predicate expression is unsolved because the constraint $0 > 0 \equiv \text{True}$ is unsatisfiable so we can not generating input data from a path predicate expression

Generating Test Input

6- Generating input data from a path predicate expression

- Consider the path predicate expression that is feasible of Figure 2

$$\begin{array}{llll} 0 < AS & \equiv \text{True} & \text{.....} & (1) \\ \text{value}[0] \neq -999 & \equiv \text{True} & \text{.....} & (2) \\ \text{value}[0] \geq \text{MIN} & \equiv \text{True} & \text{.....} & (3) \\ \text{value}[0] \leq \text{MAX} & \equiv \text{True} & \text{.....} & (4) \\ 1 < AS & \equiv \text{False} & \text{.....} & (5) \\ 1 > 0 & \equiv \text{True} & \text{.....} & (6) \end{array}$$

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

$$\begin{array}{ll} AS & = 1 \\ \text{MIN} & = 25 \\ \text{MAX} & = 35 \\ \text{Value}[0] & = 30 \end{array}$$

- Note: The above set is not unique.

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.