



CHAPTER — 5

DATA FLOW TESTING



OUTLINE OF THE CHAPTER

- The General Idea
- Data Flow Anomaly
- Overview of Dynamic Data Flow Testing
- Data Flow Graph
- Data Flow Terms
- Data Flow Testing Criteria
- Comparison of Data Flow Testing Criteria
- Feasible Paths
- Summary

THE GENERAL IDEA

- A program unit accepts inputs, performs computations, assigns new values to variables, and returns results.
- One can visualize of “flow” of data values from one statement to another.
- A data value produced in one statement is expected to be used later.
 - Example
 - Obtain a file pointer use it later.
 - If the later use is never verified, we do not know if the earlier assignment is acceptable.
- Two motivations of data flow testing
 - The memory location for a variable is accessed in a “desirable” way.
 - Verify the correctness of data values “defined” (i.e. generated) – observe that all the “uses” of the value produce the desired results.
- Idea: A programmer can perform a number of tests on data values.
 - These tests are collectively known as data flow testing.

THE GENERAL IDEA

- Data flow testing can be performed at two conceptual levels.
 - Static data flow testing
 - Dynamic data flow testing
- Static data flow testing
 - Identify potential defects, commonly known as **data flow anomaly**.
 - Analyze source code.
 - Do not execute code.
- Dynamic data flow testing
 - Involves actual program execution.
 - Bears similarity with control flow testing.
 - Identify paths to execute them.
 - Paths are identified based on **data flow testing criteria**.

DATA FLOW ANOMALY

- Anomaly: It is an abnormal way of doing something.
- For example
 - It is an abnormal situation to successively assign two values to a variable without using the first value.
 - It is abnormal to use a value of a variable before assigning a value to the variable.
 - It is abnormal to generate a data value and never use it.
 - Example 1: The second definition of x overrides the first.
 - $x = f1(y);$
 - $x = f2(z);$
- Three types of abnormal situations with using variable.
 - Type 1: Defined and then defined again
 - Type 2: Undefined but referenced
 - Type 3: Defined but not referenced

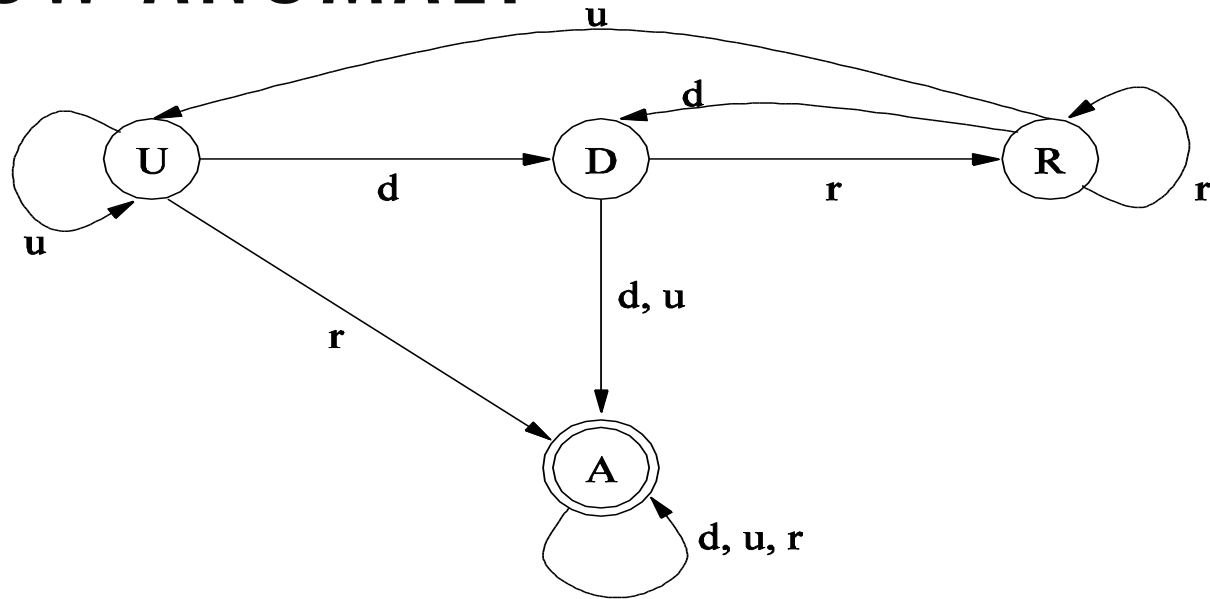
DATA FLOW ANOMALY

- Type 1: Defined and then defined again (Example 1 above)
 - Four interpretations of Example 1
 - The first statement is redundant.
 - The first statement has a fault -- the intended one might be: $w = f1(y)$.
 - The second statement has a fault -- the intended one might be: $v = f2(z)$.
 - There is a missing statement in between the two: $v = f3(x)$.
 - Note: It is for the programmer to make the desired interpretation.
- Type 2: Undefined but referenced
 - Example: $x = x - y - w$; /* w has not been defined by the programmer. */
 - Two interpretations
 - The programmer made a mistake in using w .
 - The programmer wants to use the compiler assigned value of w .
- Type 3: Defined but not referenced
 - Example: Consider $x = f(x, y)$. If x is not used subsequently, we have a Type 3 anomaly.

DATA FLOW ANOMALY

- The concept of a **state-transition diagram** is used to **model a program variable** to identify data flow anomaly.
- Components of the state-transition diagrams
 - The states
 - U: Undefined
 - D: Defined but not referenced
 - R: Defined and referenced
 - A: Abnormal
 - The actions
 - *d*: define the variable
 - *r*: reference (or, read) the variable
 - *u*: undefine the variable

DATA FLOW ANOMALY



Legends:

States

U: Undefined

D: Defined but not referenced

R: Defined and referenced

A: Abnormal

Actions

d: Define

r: Reference

u: Undefine

Figure 1: State transition diagram of a program variable

DATA FLOW ANOMALY

- Obvious question: What is the relationship between the **Type 1**, **Type 2**, and **Type 3** anomalies and Figure 1?
- The three types of anomalies (Type 1, Type 2, and Type 3) are found in the diagram in the form of **action sequences**:
 - Type 1: *dd*
 - Type 2: *ur*
 - Type 3: *du*
- The presence of a data flow anomaly in a program does not necessarily mean that the program's execution will result in failure.
- A data flow anomaly simple means that the program may fail, and therefore the programmer must investigate the cause of the anomaly.
- Bottom line: What to do after detecting a data flow anomaly?
 - Investigate the cause of the anomaly.
 - To fix an anomaly, write new code or modify the existing code.

OVERVIEW OF DYNAMIC DATA FLOW TESTING

- A programmer manipulates/uses variables in several ways.
 - Initialization, assignment, using in a computation, using in a condition
- Motivation for data flow testing?
 - One should not feel confident that a variable has been **assigned the correct value**, if no test case causes the execution of a **path** from the point of assignment to a point where the value is **used**.
- Assignment of correct value means whether or not a value has been correctly generated.
- Use of a variable means
 - Further generation of values for the same or other variables.
 - If the variable is used in a conditional statement to alter the flow of control.
- The above motivation indicates that **certain kinds of paths** are executed in data flow testing.

OVERVIEW OF DYNAMIC DATA FLOW TESTING

- Data flow testing is outlined as follows:
 - Draw a data flow graph from a program.
 - Select one or more data flow testing criteria.
 - Identify paths in the data flow graph satisfying the selection criteria.
 - Derive path predicate expressions from the selected paths (See **Chapter 4.**)
 - Solve the path predicate expressions to derive test inputs (See **Chapter 4.**)

DATA FLOW GRAPH

Occurrences of variables

- **Definition:** A variable gets a new value.
 - `i = x; /* The variable i gets a new value. */`
- **Undefinition or kill:** This occurs if the value and the location become unbound.
- **Use:** This occurs when the value is fetched from the memory location of the variable. There are **two forms** of uses of a variable.
 - **Computation use (c-use)**
 - Example: `x = 2*y; /* y has been used to compute a value of x. */`
 - **Predicate use (p-use)**
 - Example: `if (y > 100) { ... } /* y has been used in a condition. */`

DATA FLOW GRAPH

- A data flow graph is a directed graph constructed as follows.
 - A sequence of **definitions** and **c-uses** is associated with each **node** of the graph.
 - A set of **p-uses** is associated with each **edge** of the graph.
 - The entry node has a definition of each edge parameter and each non-local variable used in the program.
 - The exit node has an undefinition of each local variable.

DATA FLOW GRAPH

```
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);
}
```

DATA FLOW GRAPH

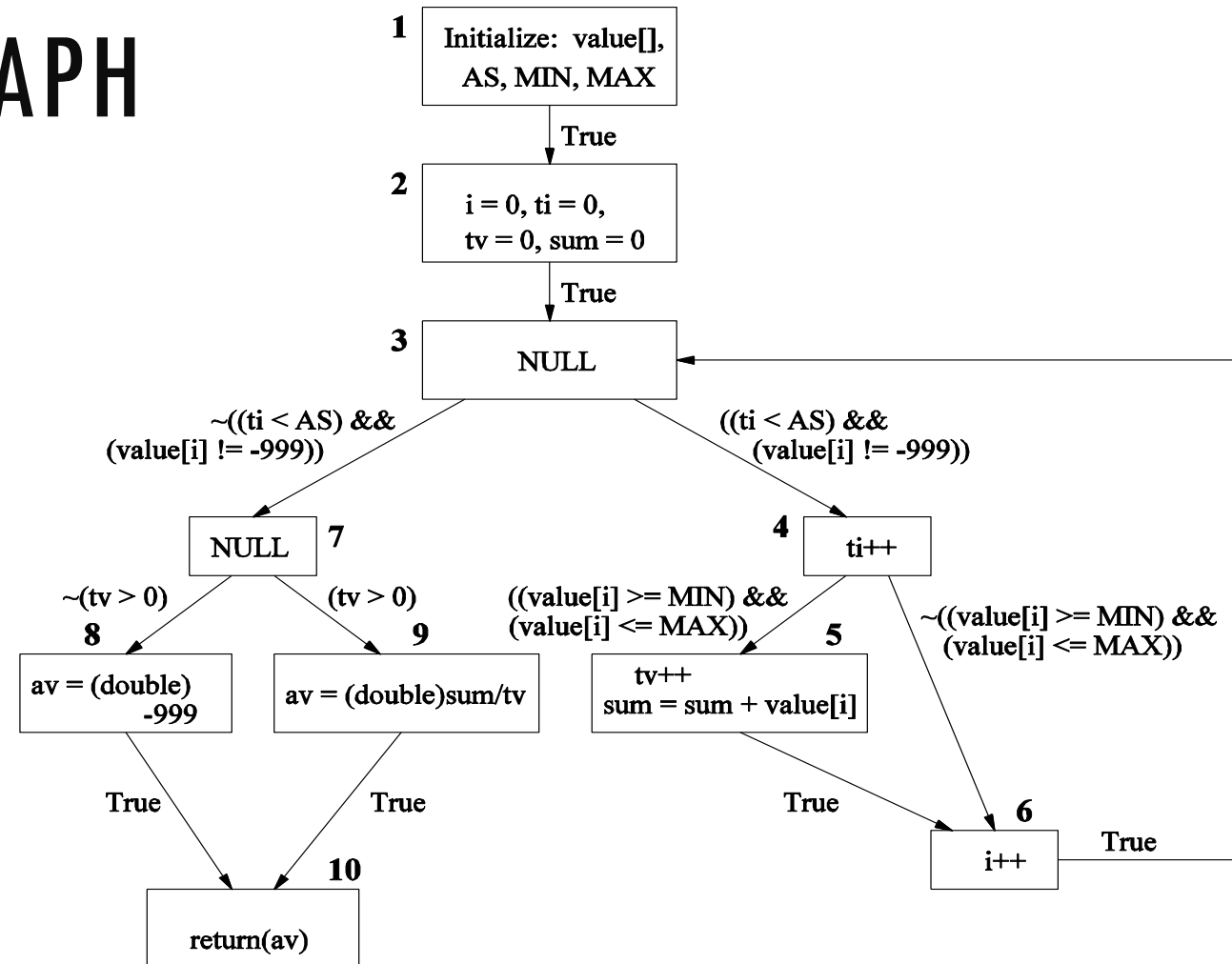


Figure 2: A data flow graph of ReturnAverage() example.

DATA FLOW TERMS

Global c-use: A c-use of a variable x in node i is said to be a global c-use if x has been defined before in a node other than node i .

- Example: The c-use of variable tv in node **9** (Figure 2) is a global c-use since tv has been defined in nodes **2** and **5**.

Definition clear path: A path $(i - n_1 - \dots - n_m - j)$, $m \geq 0$, is called a definition clear path (def-clear path) with respect to variable x

from node i to node j , and

from node i to edge (n_m, j) ,

if x has been neither defined nor undefined in nodes $n_1 - \dots - n_m$.

- Example: $(2 - 3 - 4 - 6 - 3 - 4 - 6 - 3 - 4 - 5)$ is a def-clear path w.r.t. tv in Fig. 2
- Example: $(2 - 3 - 4 - 5)$ and $(2 - 3 - 4 - 6)$ are def-clear paths w.r.t. variable tv from node 2 to 5 and from node 2 to 6, respectively, in Fig. 2

DATA FLOW TERMS

Global definition: A node i has a global definition of variable x if node i has a definition of x and there is a def-clear path w.r.t. x from node i to some node containing a global c-use, or edge containing a p-use of variable x .

Simple path: A simple path is a path in which all nodes, except possibly the first and the last, are distinct.

- Example: Paths $(2 - 3 - 4 - 5)$ and $(3 - 4 - 6 - 3)$ are simple paths.

Loop-free paths: A loop-free path is a path in which all nodes are distinct.

Complete path: A complete path is a path from the entry node to the exit node.

DATA FLOW TERMS

Du-path: A path $(n_1 - n_2 - \dots - n_j - n_k)$ is a du-path path w.r.t. variable x if node n_1 has a global definition of x and either

- node n_k has a global c-use of x and $(n_1 - n_2 - \dots - n_j - n_k)$ is a def-clear simple path w.r.t. x , or
 - Edge (n_j, n_k) has a p-use of x and $(n_1 - n_2 - \dots - n_j - n_k)$ is a def-clear, loop-free path w.r.t. x .
-
- Example: Considering the global definition and global c-use of variable tv in nodes 2 and 5, respectively, $(2 - 3 - 4 - 5)$ is a du-path.
 - Example: Considering the global definition and p-use of variable tv in nodes 2 and on edge $(7, 9)$, respectively, $(2 - 3 - 7 - 9)$ is a du-path.

DATA FLOW TESTING CRITERIA

- Seven data flow testing criteria
 - All-defs
 - All-c-uses
 - All-p-uses
 - All-p-uses/some-c-uses
 - All-c-uses/some-p-uses
 - All-uses
 - All-du-paths

DATA FLOW TESTING CRITERIA

All-defs

- For *each variable* x and *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 .
- Example (**partial**): Consider tv with its global definition in node 2. Variable tv has a global c-use in node 5, and there is a def-clear path $(2 - 3 - 4 - 5)$ from node 2 to node 5. Choose a complete path $(1 - \underline{2 - 3 - 4 - 5} - 6 - 3 - 7 - 9 - 10)$ that includes the def-clear path $(2 - 3 - 4 - 5)$ to satisfy the all-defs criterion.

DATA FLOW TESTING CRITERIA

All-c-uses

- For *each variable* x and *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 .
- Example (**partial**): Consider variable t_i , which has a global definition in 2 and a global c-use in node 4. From node 2, the def-clear path to 4 is $(2 - 3 - 4)$. One may choose the complete path $(1 - \underline{2 - 3 - 4} - 6 - 3 - 7 - 8 - 10)$. (There three other complete paths.)

All-p-uses

- For *each variable* x and *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 (j, k) .
- Example (**partial**): Consider variable t_v , which has a global definition in 2 and p-uses on edges $(7, 8)$ and $(7, 9)$. From node 2, there are def-clear paths to $(7, 8)$ and $(7, 9)$, namely $(2 - 3 - 7 - 8)$ and $(2 - 3 - 7 - 9)$. The two complete paths are: $(1 - \underline{2 - 3 - 7 - 8} - 10)$ and $(1 - \underline{2 - 3 - 7 - 9} - 10)$.

DATA FLOW TESTING CRITERIA

All-p-uses/some-c-uses

- This criterion is identical to the all-p-uses criterion **except** when a variable x has no p-use. If x has no p-use, then this criterion reduces to the some-c-uses criterion.
- Some-c-uses: For *each variable* x and *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* nodes j such that there is a global c-use of x in j .
- Example (**partial**): Consider variable i , which has a global definition in 2. There is no p-use of i . Corresponding to the global definition of I in 2, there is a global c-use of I in 6. The def-clear path from node 2 to 6 is $(2 - 3 - 4 - 5 - 6)$. A complete path that includes the above def-clear path is $(1 - \underline{2 - 3 - 4 - 5 - 6} - 7 - 9 - 10)$.

DATA FLOW TESTING CRITERIA

All-c-uses/some-p-uses

- This criterion is identical to the all-c-uses criterion **except** when a variable x has no c-use. If x has no global c-use, then this criterion reduces to the some-p-uses criterion.
- Some-p-uses: For *each variable* x and *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 (j, k) .

All-uses: This criterion produces a set of paths due to the **all-p-uses** criterion **and** the **all-c-uses** criterion.

All-du-paths: For each variable x and for 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) .

COMPARISON OF DATA FLOW TESTING CRITERIA

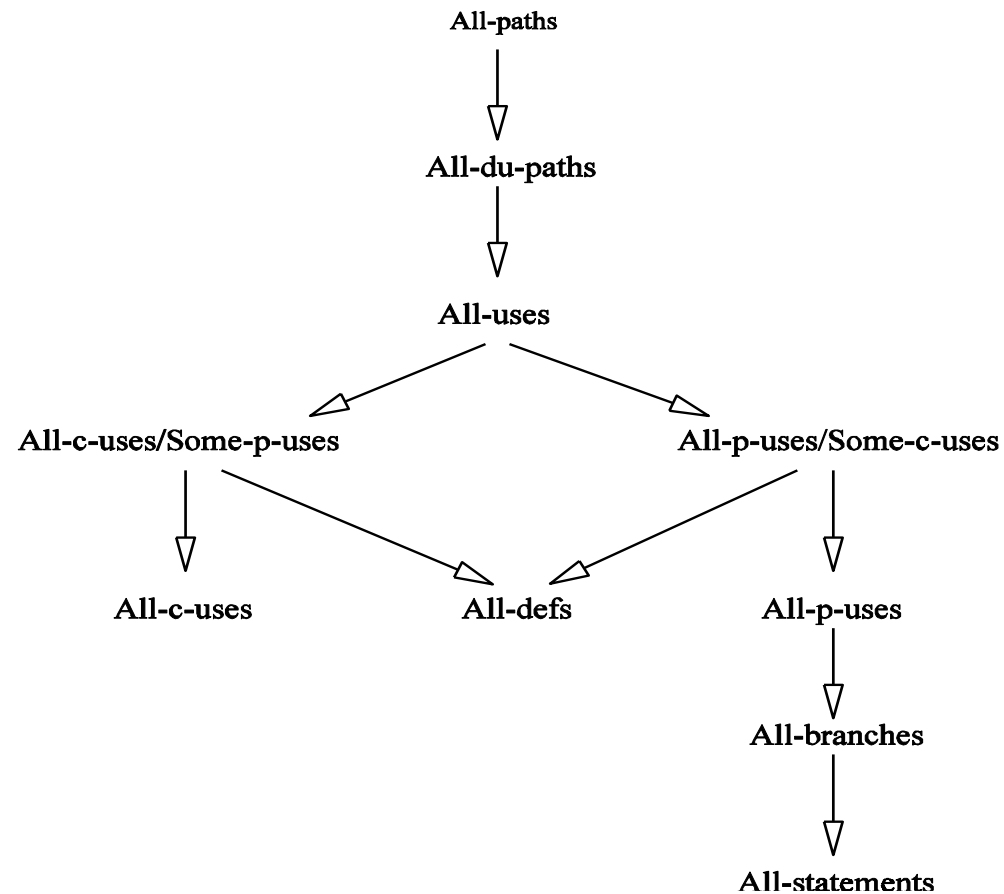


Figure 3: The relationship among DF (data flow) testing criteria

FEASIBLE PATHS

Executable (feasible) path

- A complete path is executable if there exists an assignment of values to input variables and global variables such that all the path predicates evaluate to true, thereby making the path executable.
- Executable paths are also known as feasible paths.
- If no such assignment of values to input variables and global variables exists, then we call the path infeasible or inexecutable.

FEASIBLE PATHS

Let us consider an example to understand the concept of infeasible path. Below given is a program of placing the order and subsequently providing the discount based on the ordered amount.

```
A → Place Order (Amount in Rs.);  
B → If Order Amount < 500;  
C → then Print "No Discount";  
D → Else Print "Applicable for 25% discount";  
E → If Order Amount > 1000;  
F → then Print "Applicable for 40% discount";  
G → End
```

By studying and analysing the above stated program and its corresponding graph, we may find that the path ABCEFG is an infeasible path as Node or the Statement 'F' can't be executed, while traversing the path in the direction A→B→C→E→F, whereas the path ABDEFG and ABDEG may seem to be an appropriate and feasible path.

SUMMARY

- **Data flow** is a readily identifiable concept in a program unit.
- Data flow testing
 - Static
 - Dynamic
- Static data flow analysis
 - Data flow **anomaly** can occur due to programming errors.
 - **Three** types of data flow anomalies
 - (Type 1: *dd*), (Type 2: *ur*), (Type 3, *du*)
 - **Analyze** the code to **identify** and **remove** data flow anomalies.
- Dynamic data flow analysis
 - Obtain a data flow graph from a program unit.
 - Select paths based on DF criteria: *all-defs*, *all-c-uses*, *all-p-uses*, *all-uses*, *all-c-uses/some-p-uses*, *all-p-uses/some-c-uses*, *all-du-paths*.



THANK YOU!!

