Software Testing
(Data Flow Testing)

Data flow

- A program unit, such as a function, accepts input values, performs computations while assigning new values to local and global variables, and, finally, produces output values.

Motivations for data flow testing

There are two motivations for data flow testing as follows.

- First, a memory location corresponding to a program variable is accessed in a desirable way. For example, a memory location may not be read before writing into the location.

- Second, it is desirable to verify the correctness of a data value generated for a variable—this is performed by observing that all the uses of the value produce the desired results.

Two conceptual levels

- *Static data flow testing*
- Testing is performed by analyzing the source code, and it does not involve actual execution of source code. Static data flow testing is performed to reveal potential defects in programs.
- *Dynamic data flow testing*

Dynamic data flow testing involves identifying program paths from source code based on a class of *data flow testing criteria*.

CFT and DFT

There is much similarity between control flow testing and data flow testing. Moreover, there is a key difference between the two approaches.

- The similarities stem from the fact that both approaches identify program paths and emphasize on generating test cases from those program paths.
- The difference between the two lies in the fact that control flow test selection criteria are used in the former, whereas data flow test selection criteria are used in the latter approach.

Data Flow Anomalies

The potential program defects are commonly known as data flow anomaly

- *Defined and Then Defined Again (Type 1)*
- *Undefined but Referenced (Type 2)*
- *Defined but Not Referenced (Type 3)*

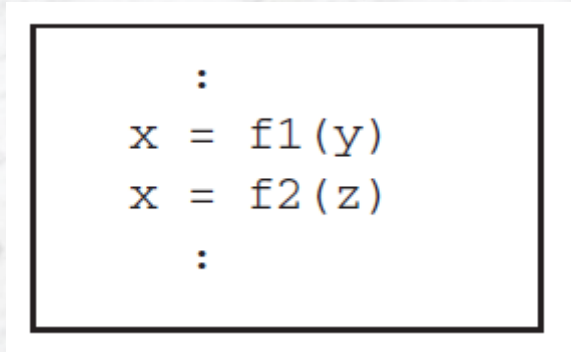
- *Undefined: Declare*
- *Defined: Initialize*
- *Referenced: declare, initialize and utilize*

Defined and Then Defined Again (Type 1)

- The computation performed by the first statement is redundant if the second statement performs the intended computation.
- The first statement has a fault. For example, the intended first computation might be w = f1(y).
- The second statement has a fault.

For example, the intended second

computation might be v = f2(z).



- A third kind of fault can be present in the given sequence in the form of a missing statement between the two. For example, v = f3(x) may be the desired statement that should go in between the two given statements.

Undefined but Referenced (Type 2)

- To use an undefined variable in a computation, such as $x = x - y - w$, where the variable *w* has not been *initialized* by the programmer.
- **For correction: one must eliminate the anomaly either by initializing w or replacing w with the intended variable**

Defined but Not Referenced (Type 3)

Third kind of data flow anomaly is to define a variable and then to undefine it without using it in any subsequent computation.

For example, consider the statement $x = f(x, y)$ in which a new value is assigned to the variable x. If the value of x is not used in any subsequent computation, then we should be suspicious of the computation represented by $x = f(x, y)$. Hence, this form of anomaly is called “defined but not referenced.”

State transition diagram of a program variable

States

U: Undefined

D: Defined but not referenced

R: Defined and referenced

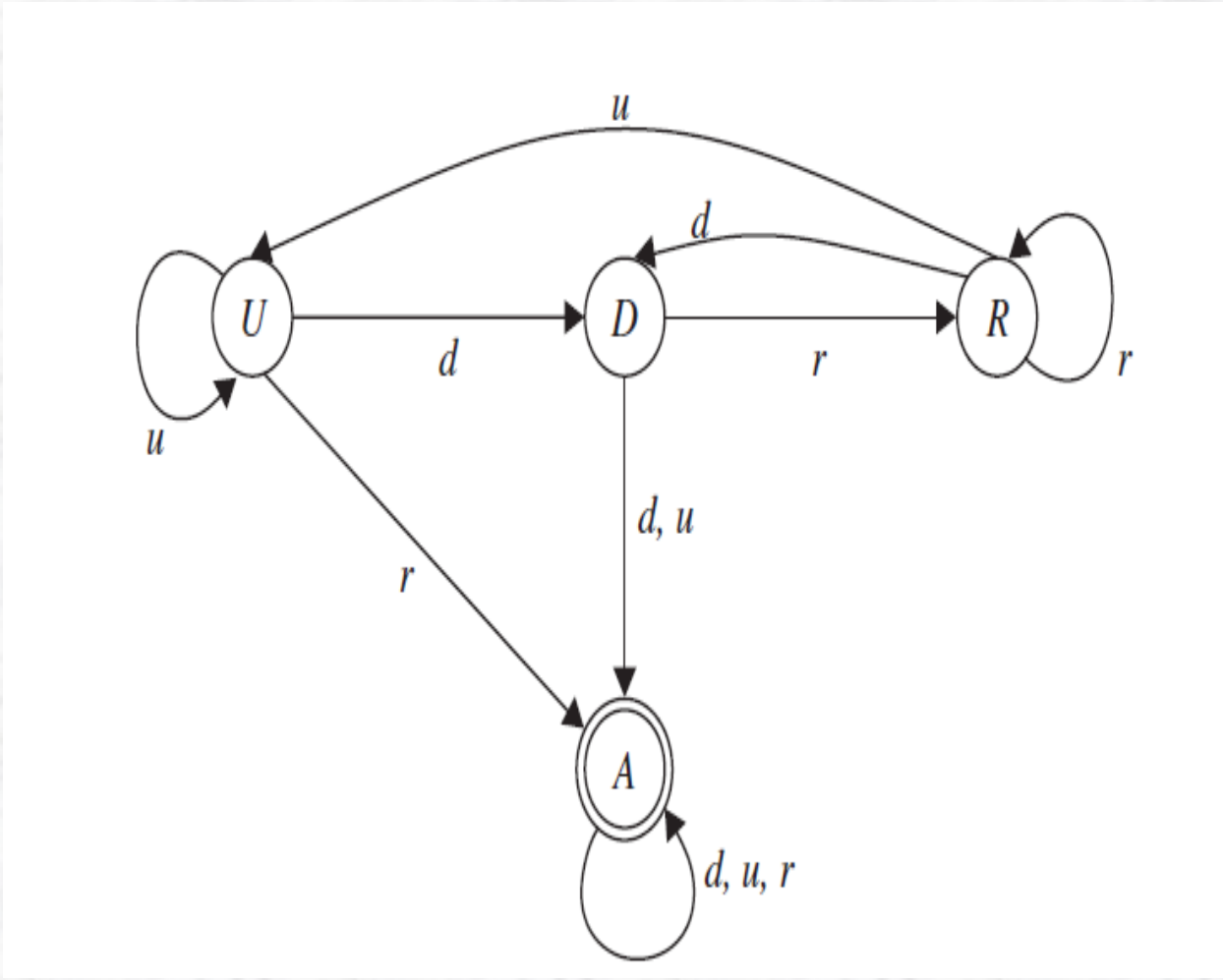
A: Abnormal

Actions

d: Define

r: Reference

u: Undefine



An outline of performing data flow testing

- 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 and solve those expressions to derive test input.

There are two forms of *uses* of a variable

- **Computation use (c-use):** This directly affects the computation being performed. In a c-use, a potentially new value of another variable or of the same variable is produced. Referring to the C function VarTypes(), the statement

*iptr = i + x;

gives examples of c-use of variables *i* and *x*.

- **Predicate use (p-use):** This refers to the use of a variable in a predicate controlling the flow of execution. Referring to the C function VarTypes(), the statement

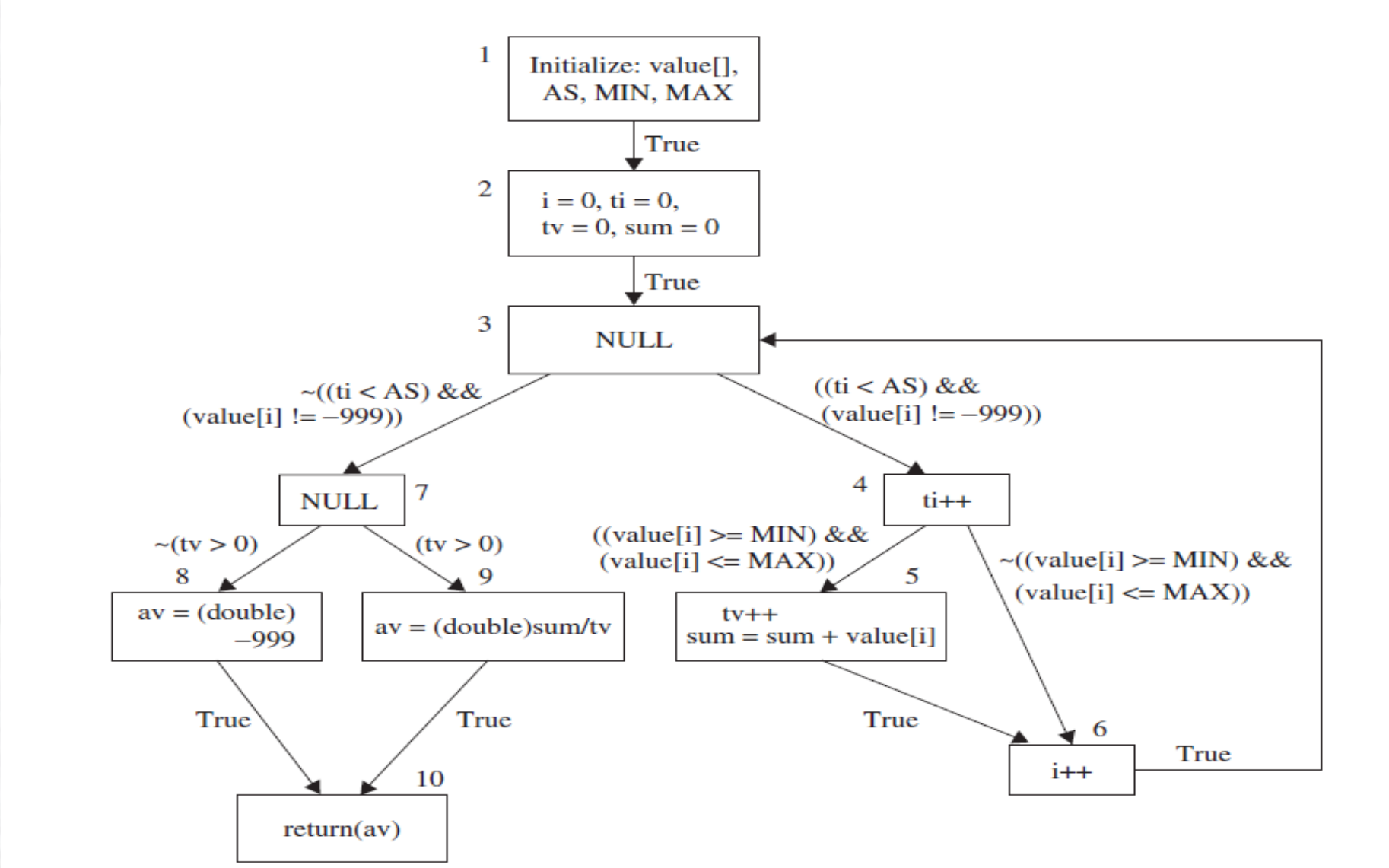
if (*iptr > y) ...

gives examples of p-use of variables *y* and iptr.

Data Flow Graph (Directed Graph)

- 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 parameter and each nonlocal variable which occurs in the subprogram.
- The exit node has an *undefinition* of each local variable.

Data flow graph of ReturnAverage() example.



Simple Path in DFG

- *Simple Path*: A simple path is a path in which all nodes, except possibly the first and the last, are distinct.
- Paths **2-3-4-5** and **3-4-6-3** are simple paths

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

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

Av=double(sum/tv)

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) && (value[i] ! = − 999)	{i, ti, AS, value}
(4, 5)	(value[i] < = MIN) && (value[i] > = MAX)	{i, MIN, MAX, value}
(4, 6)	~((value[i] < = MIN) && (value[i] > = MAX))	{i, MIN, MAX, value}
(5, 6)	True	{}
(6, 3)	True	{}
(3, 7)	~((ti < AS) && (value[i] ! = − 999))	{i, ti, AS, value}
(7, 8)	~(tv > 0)	{tv}
(7, 9)	(tv > 0)	{tv}
(8, 10)	True	{}
(9, 10)	True	{}

References:

- Chapter 5: Software Testing and Quality Assurance (Indian)