Software Testing and Quality Analysis

Chapter 4:
Structural Testing Methods: Data Flow Graph

Module Supervisor: Mrs. M. Halassa

Academic year 2024/2025

Contents

- Introduction
- Variable definition and Variable use
- Data Flow Testing Criteria
- Relationship between diff criteria
- Conclusion

Introduction

- **Data flow testing** can be considered to be a form of structural testing: in contrast to functional testing, where the program can be tested without any knowledge of its internal structures, structural testing techniques require the tester to have access to details of the program's structure. Data flow testing focuses on the variables used within a program.

Definition

- Data flow testing identifies paths in the program that go from the assignment of a value to a variable to the use of such variable, to make sure that the variable is properly used.

Variable definition and Variable use

- Variable definition(def) : a location where a value of the variable is changed
  - x appears on the left side of an assignment (x = 4;)
  - x is an input to a program (reading instruction).
- Variable use (use) : a location where variable's value is accessed
  - x appears on the right side of an assignment
  - x appears in a conditional test
  - x is an actual parameter to a method
  - x is an output of the program or writing instruction.
  - x is an output of a method in a return statement.

Variable definition and Variable use

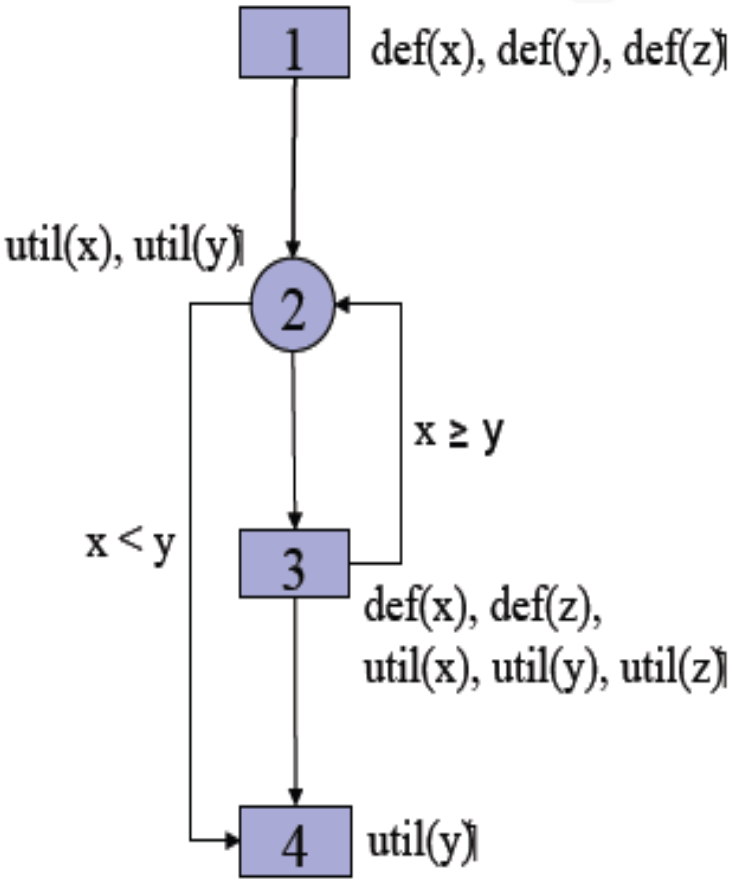
```
open (fichier1) ;
read (x,fichier1) ;
read (y,fichier1) ;
z := 0 ;
-------------------------
while x ≥ y do
-------------------------
    begin
        x := x-y ;
        z := z+1 ;
    end
-------------------------
open (fichier2) ;
print (y,fichier2) ;
close (fichier1) ;
close (fichier2) ;
```

Bloc 1
Définitions : x, y, z
Utilisations : aucune

Bloc 2
Définitions : aucune
Utilisations : x, y

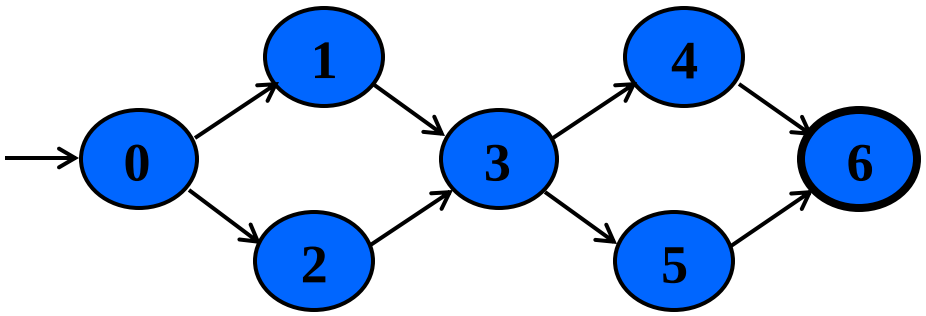
Bloc 3
Définitions : x, z
Utilisations : x, y, z

Bloc 4
Définitions : aucune
Utilisations : y



Test Paths

- Test Path : A path that starts at an initial node and ends at a final node
- Test paths represent execution of test cases
  - Some test paths can be executed by many tests
  - Some test paths cannot be executed by any tests
- Subpath : A subsequence of nodes .



Four test paths
[0, 1, 3, 4, 6]
[0, 1, 3, 5, 6]
[0, 2, 3, 4, 6]
[0, 2, 3, 5, 6]

Visiting and Touring

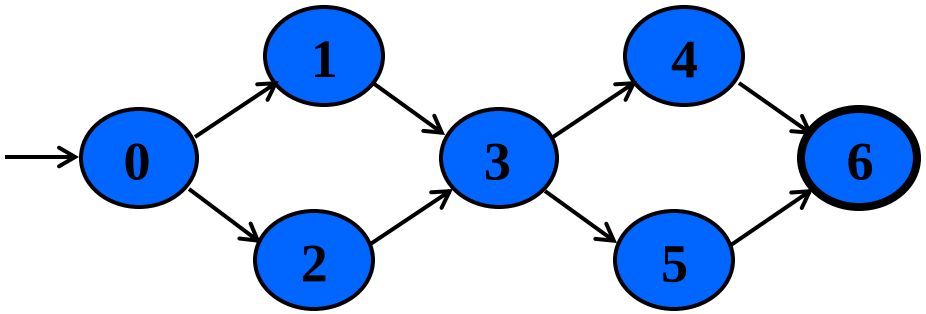
- Visit : A test path *p* *visits* node *n* if *n* is in *p*
  A test path *p* *visits* edge *e* if *e* is in *p*
- Tour : A test path *p* *tours* subpath *q* if *q* is a subpath of *p*

Path [0, 1, 3, 4, 6]

Visits nodes 0, 1, 3, 4, 6

Visits edges (0, 1), (1, 3), (3, 4), (4, 6)

Tours subpaths [0, 1, 3], [1, 3, 4], [3, 4, 6], [0, 1, 3, 4], [1, 3, 4, 6]



C-use and P-use

- C-use : Uses of a variable that occurs within an expression as part of an *assignment statement*, in an *output statement*, as a *parameter within a function call*, and in *subscript expressions*, are classified as c-use, where the “c” in c-use stands for computational.
- P-use: The occurrence of a variable in an expression used as a *condition in a branch statement* such as an *if* and a *while*, is considered as a p-use. The “p” in p-use stands for predicate.

Multiple use of a variable

1- Instruction using a variable in a calculation and in a predicate:

- case of c-usage and p-usage together

  Example: if (x + 5) > 0

  Node labeled by **c-use** and **p-use**

2-Using a variable in a calculation and in an assignment:

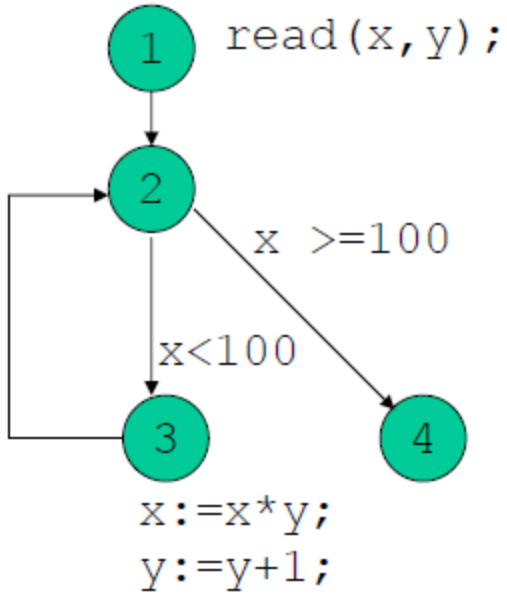
- case of c-use and declaration

  Example: x = x + 5

  Node labeled by **def** and **c-use**

Data Flow Based Criteria

- For example:

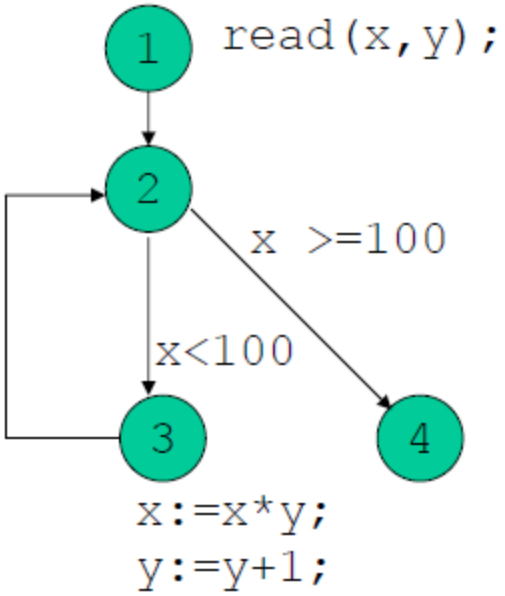


- The branch (2, 4) is **p-user** of x which defined in the node 1.
- The statement 3 ‘x:=x*y’ **is c-user** of y witch defined in the node 1.

Data Flow Based Criteria

A usage path (c-use or p-use) is a path linking the variable definition instruction to a user instruction.

[1,2,4] is a p-use path
for the variable x.

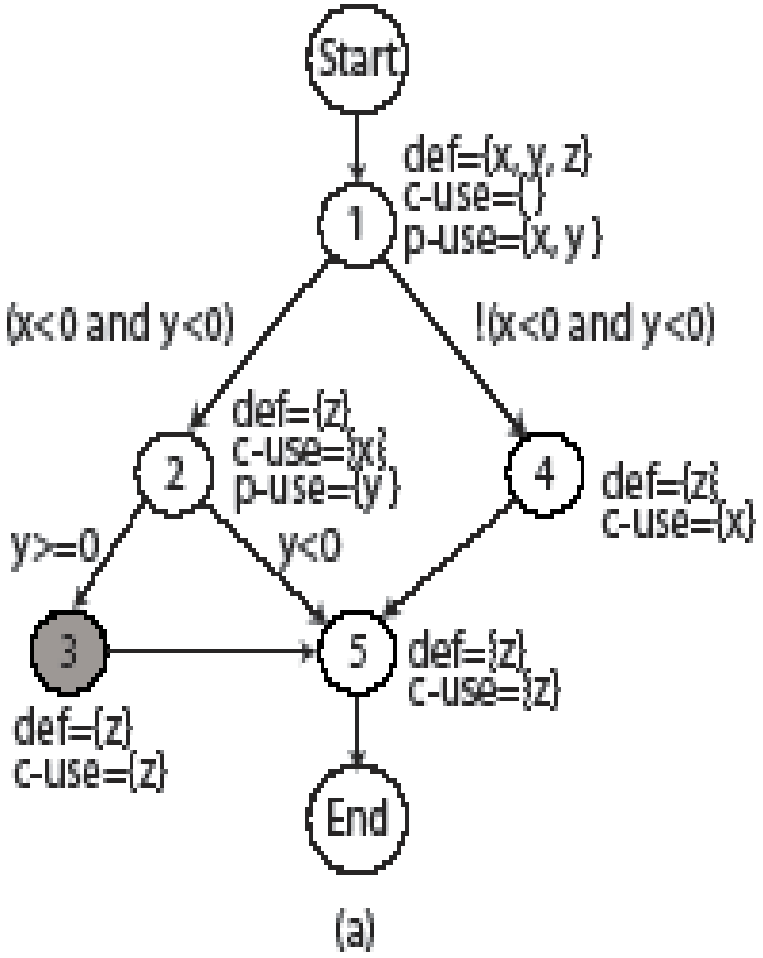


Def-Clear Path

Def-Clear Path (dc-path)

Any path starting from a node at which variable *x* is defined and ending at a node at which *x* is used, *without redefining x anywhere else along the path*, is a def-clear path for *x.*

-Path 2-5 is def-clear for variable *z* defined at node 2 and used at node 5.
-Path 1-2-5 is *NOT def-clear for variable z* defined at node 1 and used at node 5.



Data Flow Testing Criteria

- Six data flow testing criteria
  - All-defs
  - All-uses
  - All DU-Path

  - All-c-uses
  - All-p-uses
  - All-p-uses/some-c-uses
  - All-c-uses/some-p-uses

All-definitions and all-uses criteria

- **All-definitions(**all-defs): for every node i, and every x in def(i) there is a def-clear path

  , i.e.,Each def reaches at least on use.

  - For def of every var, one of its uses (p-use or c-use) must be tested
  - Intuitively: All definitions are used at least once
  - Limit of the criterion: some definition-usage pairs not covered

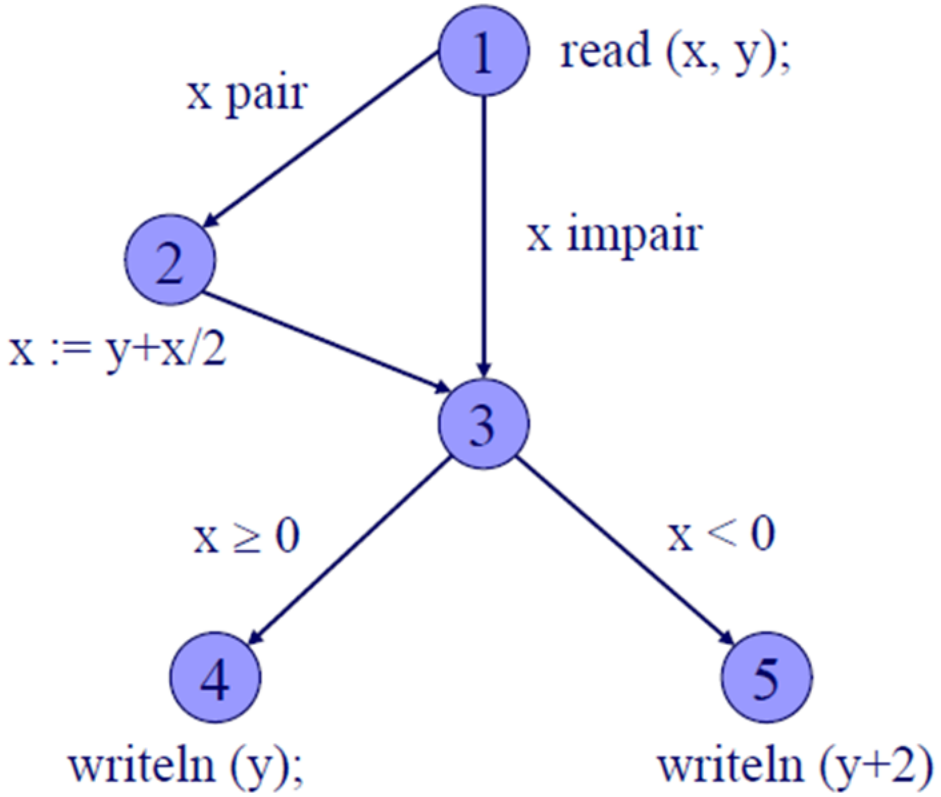
All-definitions and all-uses criteria

- ***All-uses:*** *for each definition and for each reference accessible from this definition, coverage of all users*

(c-user nodes or p-user arcs)(Just cover each use of each definition) *i.e.,Each def reaches all possible uses.*

- *all-users* ➔ *all-definitions*

All-definitions and all-uses criteria



All def test paths

[1,3,5]
[1,2,3,5]

All uses test paths

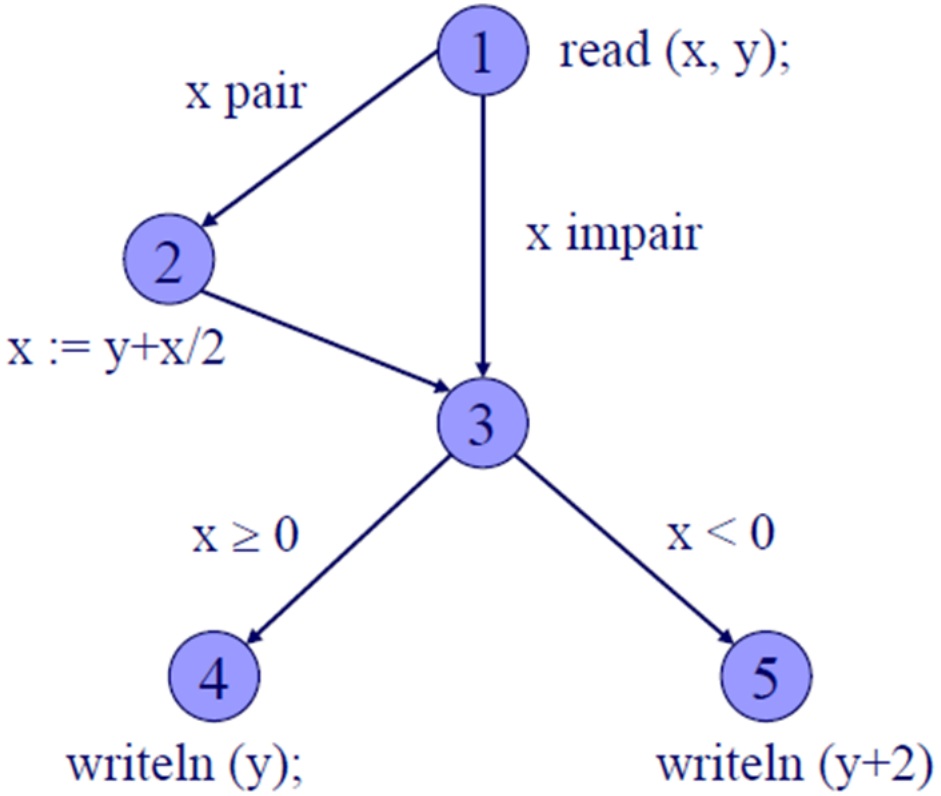
[1,3,4]
[1,2,3,4]
[1,3,5]
[1,2,3,5]

Other criteria based on data flow

- **All-c-uses/some-.p-use :** for each definition, and for each c-usage accessible from this definition, there is a dr-strict path; if there is no c-usage, just have a dr-strict path between the definition and one of the p-usages.

- **All-p-uses/some-c-uses :** for each definition, and for each p-usage accessible from this definition and for each branch derived from the associated condition, there is a dr-strict path extended by the first segment of this branch; if there is no p-usage, it is sufficient to have a dr-strict path between the definition and one of the c-usages,

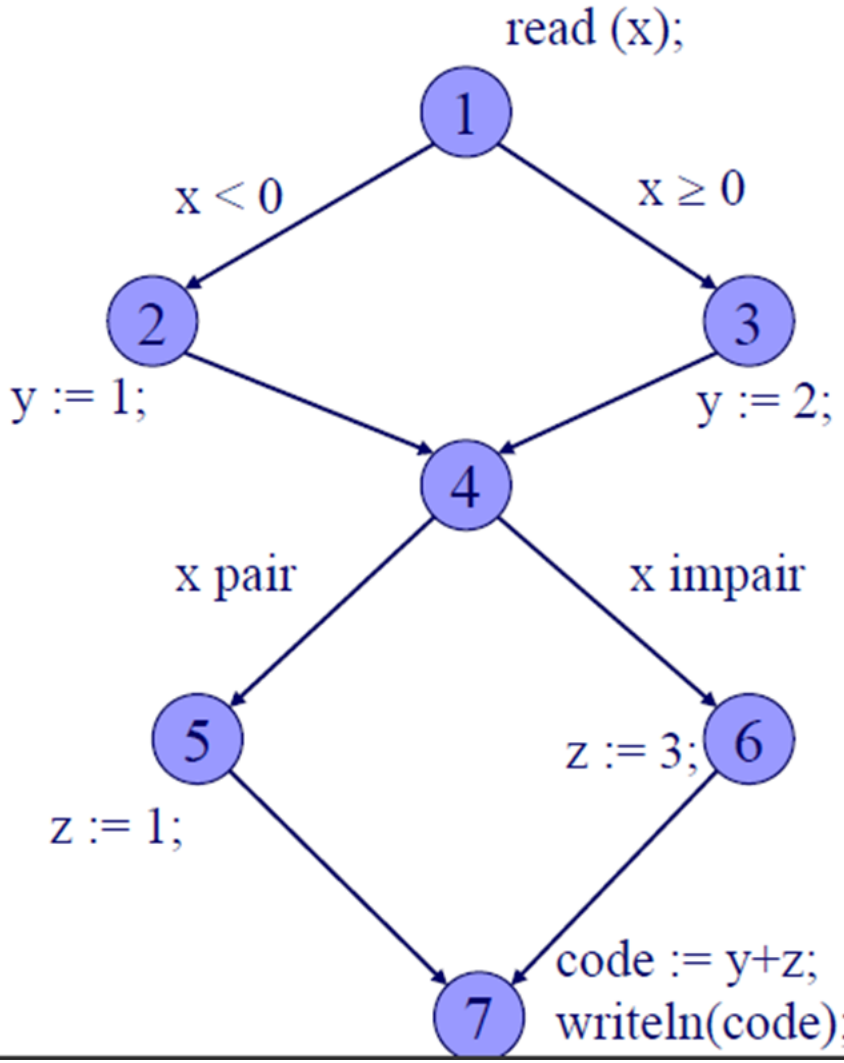
All-p-uses/some-c-uses *-example*



All-p-uses/some-c-uses test paths

[1,3,4]
[1,2,3,5]

All-uses criteria *limit*



All uses test path

[1,2,4,5,7]
[1,3,4,6,7]

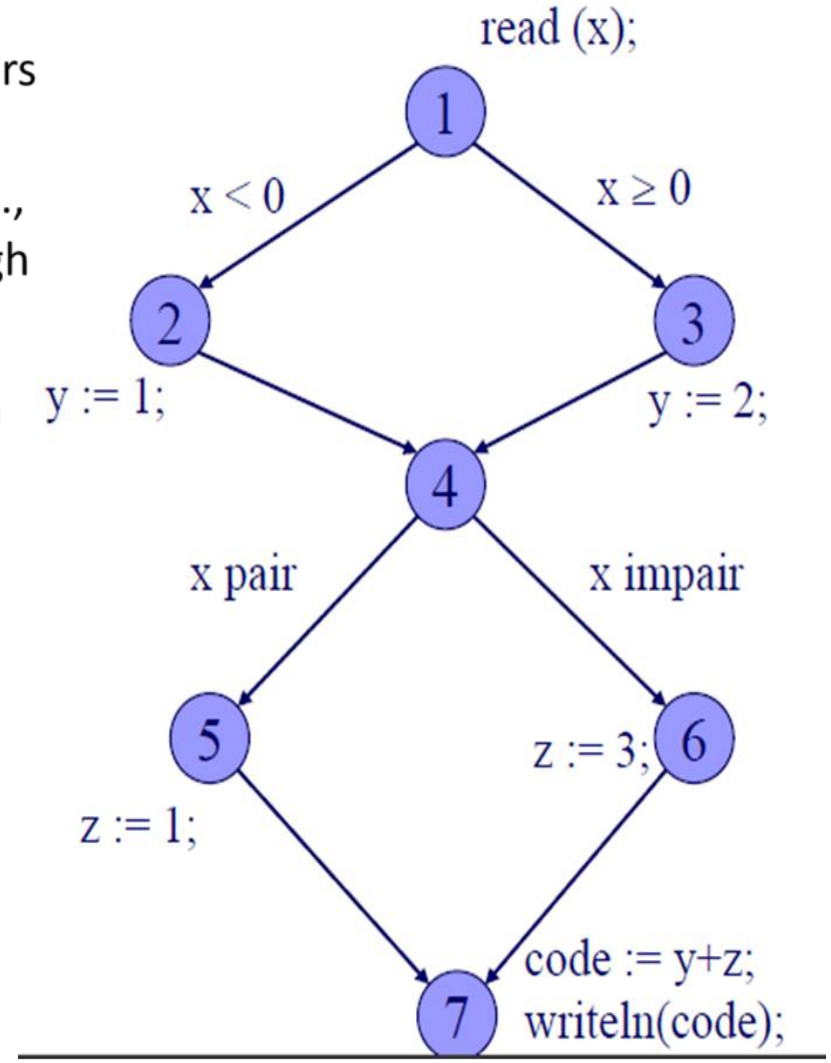
Both of these tests do not cover all uses paths. The node (7) Can be a user of the node (2) for the variable y

=> All DU-Path

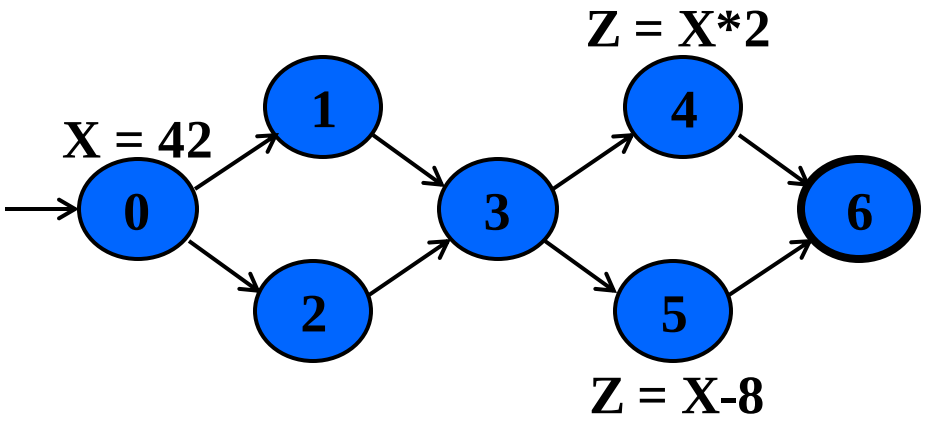
All DU-Path *criterion*

- This criterion adds to the criterion all-users that all possible paths must be covered between the definition and reference- i.e., each def reaches all possible uses through all possible DU paths
- In the previous example, this criterion test paths are :

[1,2,4,5,7]
[1,3,4,6,7]
[1,2,4,6,7]
[1,3,4,5,7]



Data Flow Testing Example

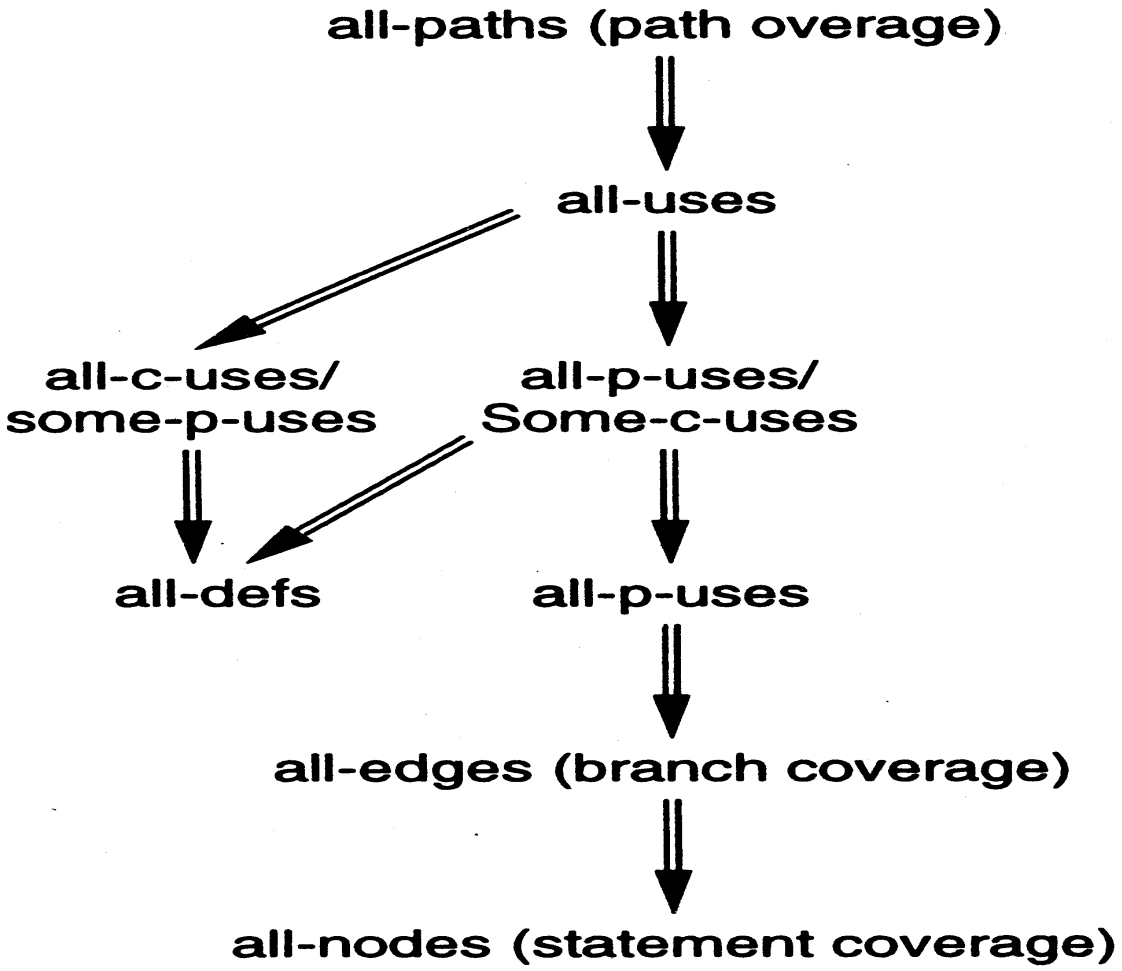


All-defs for *X*
[0, 1, 3, 4]

All-uses for *X*
[0, 1, 3, 4]
[0, 1, 3, 5]

All-du-paths for *X*
[0, 1, 3, 4]
[0, 2, 3, 4]
[0, 1, 3, 5]
[0, 2, 3, 5]

Relationship between diff criteria



Conclution

- Data- ow coverage criteria are claimed to provide a better measure of coverage than control ow because they track dependencies between variables in the fow graph.
- We can use these to discover potential faults in the program,for example:

- If a definition is only followed by definitions of the same variable- is it useful?

- If we use a variable and it is not always preceded by a definition we might - use it when it is undefined?