Session 5

White-Box Testing (cont.)

chengbaolei@suda.edu.cn

3.6 Data Flow Testing

- The data flow testing method selects test paths of a program according to the locations of definitions, uses and deletions of variables in the program
- Data flow testing is a powerful tool to detect improper use of data values due to coding errors
  - Incorrect assignment or input statement
  - Definition is missing (use of null definition)
  - Predicate is faulty (incorrect path is taken which leads to incorrect definition)

Variable Definitions and Uses

- A program variable is **DEFINED** when it appears:
  - on the *left* hand side of an assignment statement (e.g., Y := 17)
  - in an input statement (e.g., input(Y))
  - as an OUT parameter in a subroutine call (e.g., DOIT(X:IN,Y:OUT))

  - 将数据存储起来, 存储单元的内容改变

Variable Definitions and Uses

* A program variable is **USED** when it appears:
  * on the *right* hand side of an assignment statement (e.g., Y := X+17)
  * as an IN parameter in a subroutine or function call (e.g., Y := SQRT(X))
  * in the predicate of a branch statement (e.g., if X>0 then...)

  * 将数据取出来，存储单元的内容不变

Variable Definitions and Uses

* Use of a variable in the predicate of a branch statement is called a *predicate-use* **(“p-use”).** Any other use is called a *computation-use* **(“c-use”).**
* For example, in the program statement:

```
If (X>0) then
    print(Y)
end_if_then
```

there is a p-use of X and a c-use of Y.

Variable Definitions and Uses

- A variable can also be used and then re-defined in a single statement when it appears:
  - on *both* sides of an assignment statement (e.g., Y := Y+X)
  - as an IN/OUT parameter in a subroutine call (e.g., INCREMENT(Y:IN/OUT))

3.6 Data Flow Testing

- Variables that contain data values have a defined life cycle: defined, used, killed (destroyed).
- The "scope" of the variable

```
{           // begin outer block
   int x;     // x is defined as an integer within this outer block
  ...;        // x can be accessed here
     {        // begin inner block
     int y;   // y is defined within this inner block
  ...;        // both x and y can be accessed here
     }        // y is automatically destroyed at the end of this block
     ...;     // x can still be accessed, but y is gone
 }    // x is automatically destroyed
```

3.6 Data Flow Testing

- Three possibilities exist for the first occurrence of a variable through a program path:
  - ~d - the variable does not exist (indicated by the ~), then it is defined (d)
  - ~u - the variable does not exist, then it is used (u): c-use / p-use
  - ~k - the variable does not exist, then it is killed or destroyed (k)

3.6 Data Flow Testing

* List 9 pairs of defined (d), used (u), and killed (k):
  * dd - Defined and defined again—not invalid but suspicious. Probably a programming error.
  * du - Defined and used—perfectly correct. The normal case.
  * dk - Defined and then killed—not invalid but probably a programming error.
  * ud - Used and defined—acceptable.
  * uu - Used and used again—acceptable.
  * uk - Used and killed—acceptable.
  * kd - Killed and defined—acceptable. A variable is killed and then redefined.
  * ku - Killed and used—a serious defect. Using a variable that does not exist or is undefined is always an error.
  * kk - Killed and killed—probably a programming error

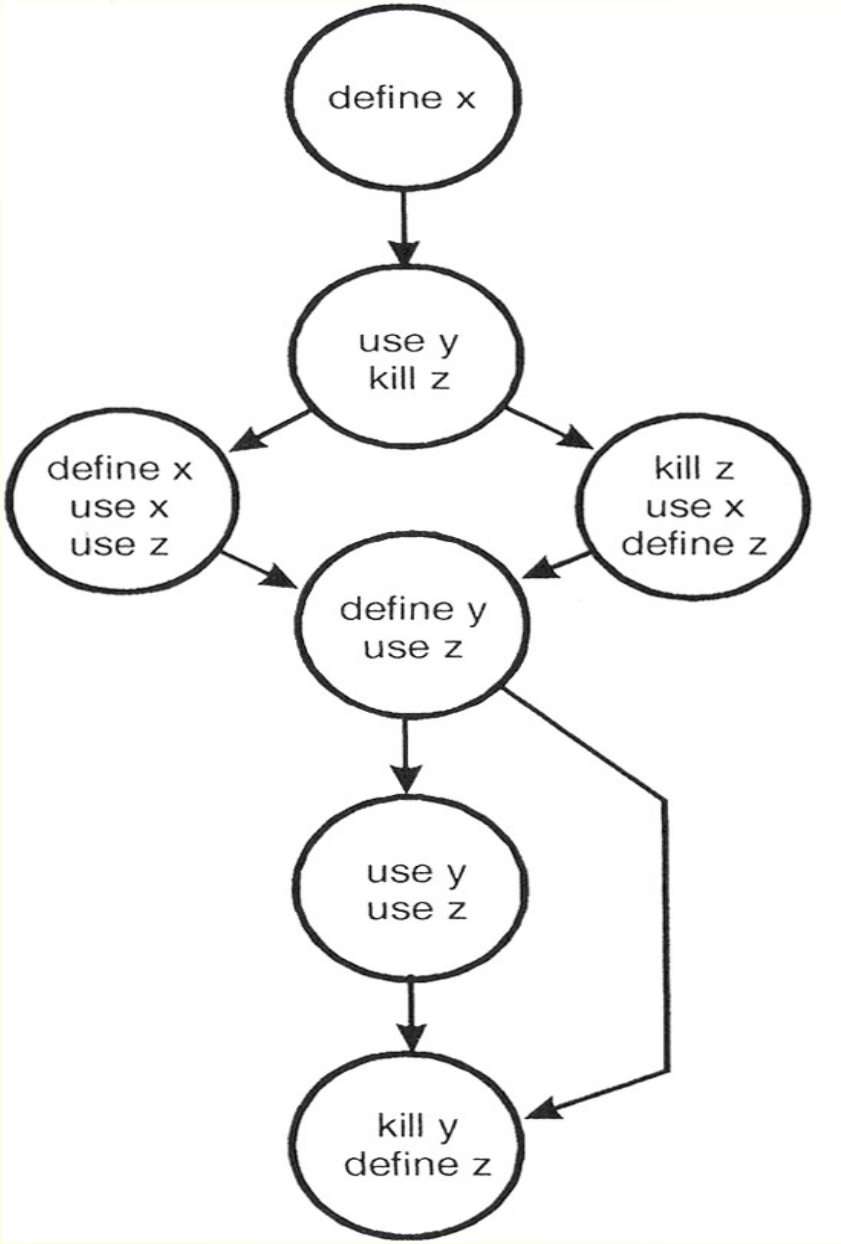
3.6 Data Flow Testing

* A data flow graph is similar to a control flow graph in that it shows the processing flow through a module.

  In addition, it details the definition, use, and destruction of each of the module's variables.
* Technique
  * Construct diagrams
  * Perform a static test of the diagram
  * Perform dynamic tests on the module

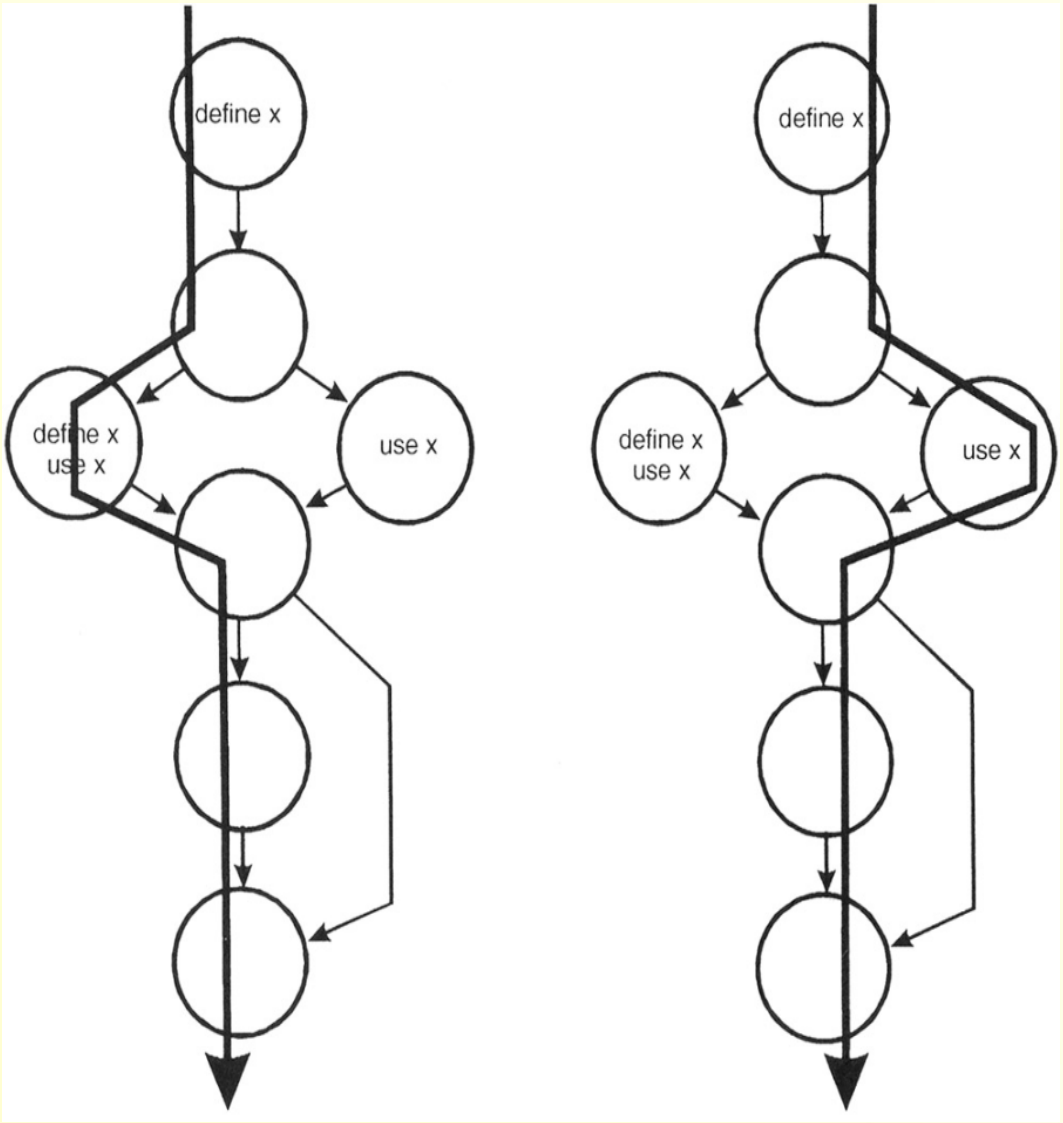
3.6 Data Flow Testing

* For each variable within the module we will examine define-use-kill patterns along the control flow paths.



3.6 Data Flow Testing

* The define-use-kill patterns for x (taken in pairs as we follow the paths) are:
  * ~define - correct, the normal case
  * define-define - suspicious, perhaps a programming error
  * define-use - correct, the normal case

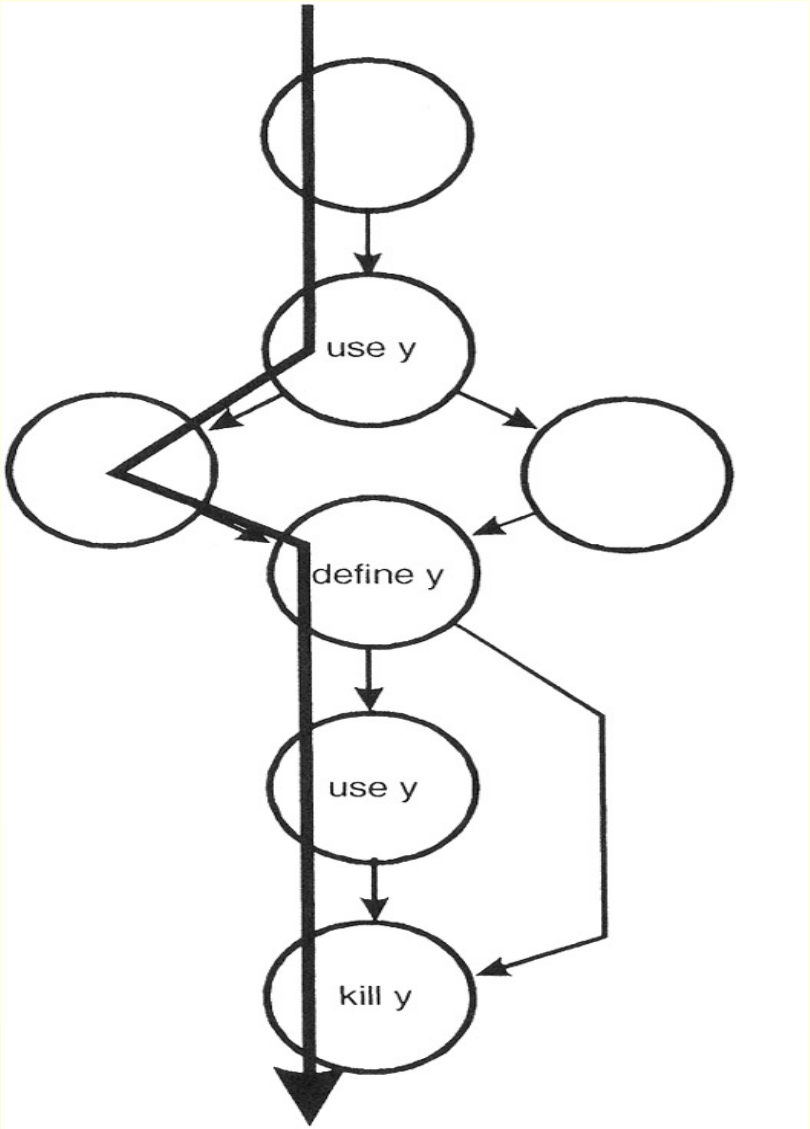


3.6 Data Flow Testing

* Exercise:
  * List the define-use-kill patterns for y and z (taken in pairs as we follow the paths)

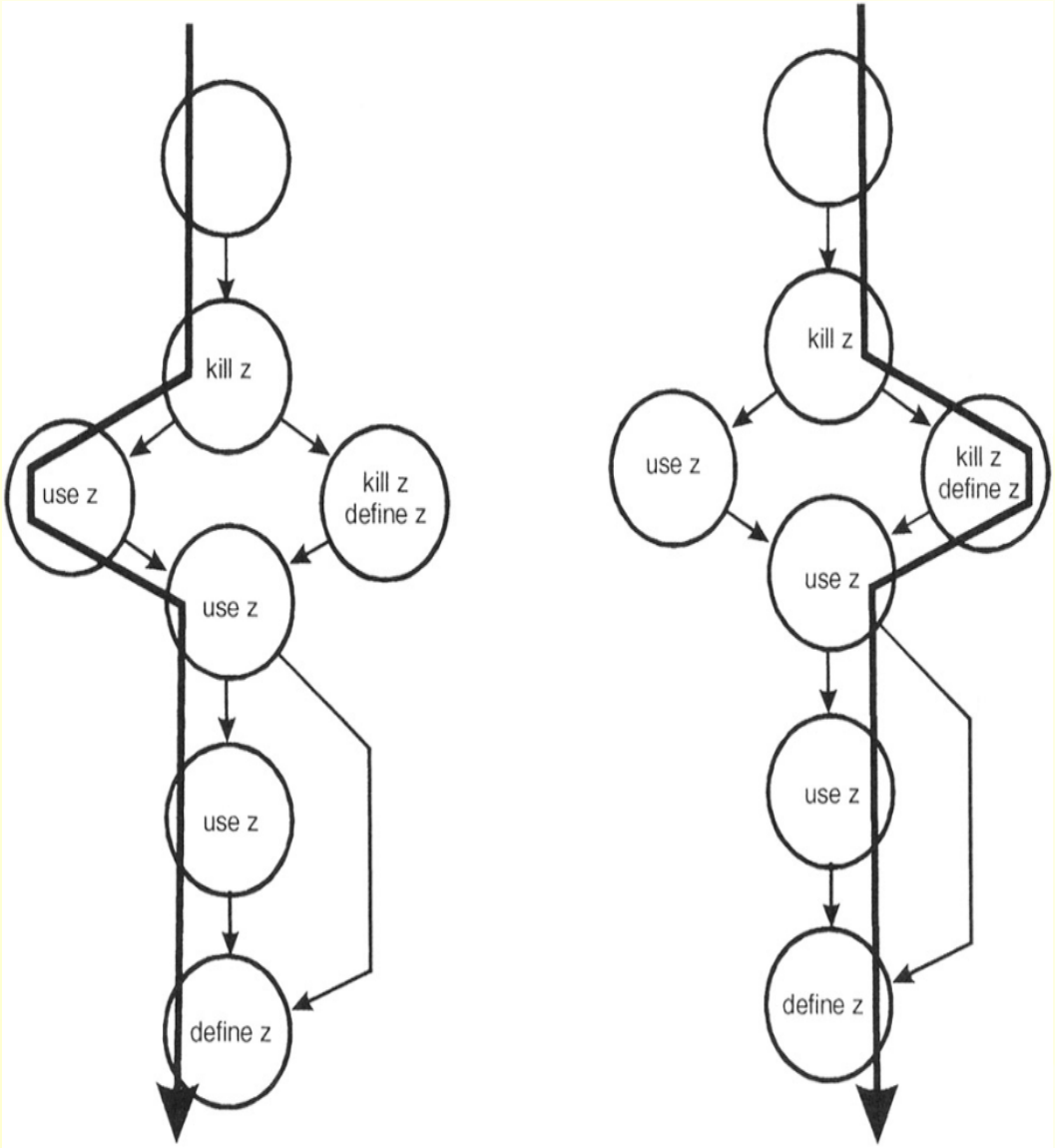
3.6 Data Flow Testing

* The define-use-kill patterns for y (taken in pairs as we follow the paths) are:
  * ~use - major blunder
  * use-define - acceptable
  * define-use - correct, the normal case
  * use-kill - acceptable
  * define-kill - probable programming error



3.6 Data Flow Testing

* The define-use-kill patterns for z (taken in pairs as we follow the paths) are:
  * ~kill - programming error
  * kill-use - major blunder
  * use-use - correct, the normal case
  * use-define - acceptable
  * kill-kill - probably a programming error
  * kill-define - acceptable
  * define-use - correct, the normal case



3.6 Data Flow Testing

* In performing a static analysis on this data flow model the following problems have been discovered:
  * x: define-define
  * y: ~use
  * y: define-kill
  * z: ~kill
  * z: kill-use
  * z: kill-kill