# Topics in Software Dynamic White-box Testing Part 2: Data-flow Testing

[Reading assignment: Chapter 7, pp. 105-122 plus many things in slides that are not in the book …]

# Data-Flow Testing

- **Data-flow testing** uses the control flowgraph to explore the unreasonable things that can happen to data (*i.e.*, anomalies).
- Consideration of data-flow anomalies leads to test path selection strategies that fill the gaps between complete path testing and branch or statement testing.

# Data-Flow Testing (Cont'd)

- **Data-flow testing** is the name given to a family of test strategies based on selecting paths through the program's control flow in order to explore sequences of events related to the status of data objects.
- *E.g.,* Pick enough paths to assure that:
  - Every data object has been initialized prior to its use.
  - All defined objects have been used at least once.

# Data Object Categories

- (d) Defined, Created, Initialized
- (k) Killed, Undefined, Released
- (u) Used:
  - (c) Used in a calculation
  - (p) Used in a predicate

# (d) Defined Objects

- An object (*e.g.*, variable) is **defined** when it:
  - appears in a data declaration
  - is assigned a new value
  - is a file that has been opened
  - is dynamically allocated
  - ...

# (u) Used Objects

- An object is **used** when it is part of a computation or a predicate.
- A variable is used for a computation **(c)**
  A variable is used in a predicate **(p)**

# Example: Definition and Uses

What are the *definitions* and *uses* for the program below?

```
1.  read (x, y);
2.  z = x + 2;
3.  if (z < y)
4       w = x + 1;
    else
5.      y = y + 1;
6.  print (x, y, w,
   z);
```

# Example: Definition and Uses

```
1.   read (x, y);
2.   z = x + 2;
3.   if (z < y)
4          w = x + 1;
     else
5.         y = y + 1;
6.   print (x, y, w, z);
```

| *Def* | *C-use* | *P-use* |
|---|---|---|
| x, y | | |
| z | x | |
| | | z, y |
| w | x | |
| y | y | |
| | x, y, w, z | |

# Data-Flow Modeling

- Data-flow modeling is based on the control flowgraph.
- Each link is annotated with:
  - symbols (*e.g.,* **d**, **k**, **u**, **c**, **p**)
  - sequences of symbols (*e.g.,* **dd**, **du**, **ddd**)
- that denote the sequence of data operations on that link with respect to the variable of interest.

# **du** Path Segments

- A **du Path** is a path segment such that if the last link has a use of **X**, then the path is simple and definition clear.

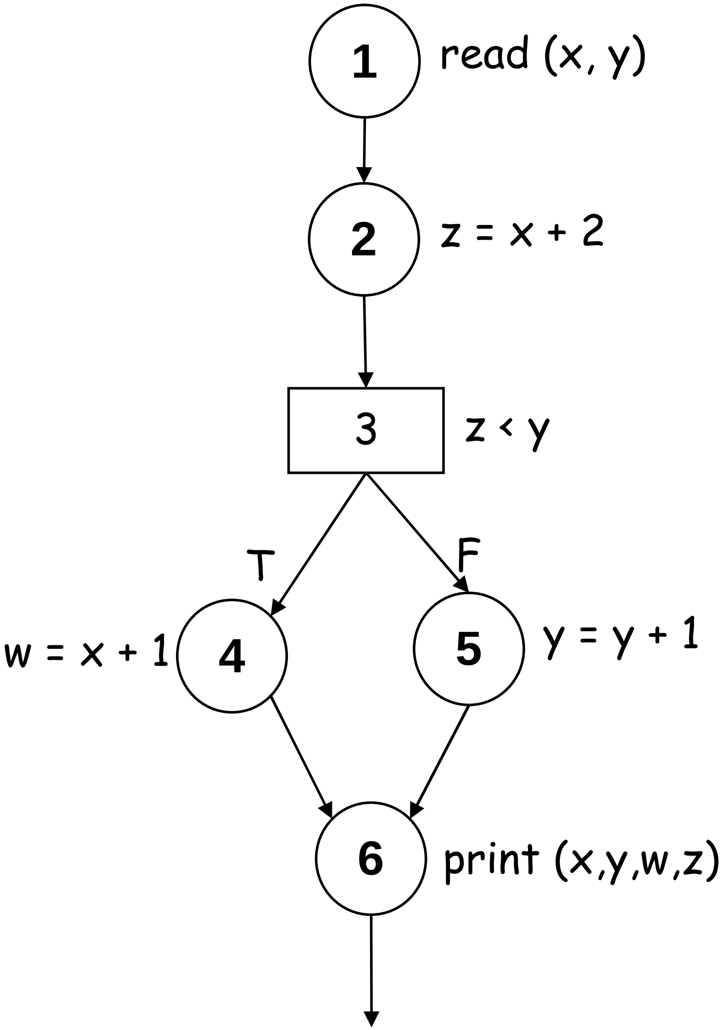
# def-use Associations

- A def-use association is a triple (x, d, u,), where:

  x is a variable,
  d is a node containing a definition of x,
  u is either a statement or predicate node containing a use of x,

  and there is a sub-path in the flow graph from d to u with no other definition of x between d and u.

# Example: Def-Use Associations




*Some* Def-Use Associations:

(x, 1, 2), (x, 1, 4), …

(y, 1, (3,t)), (y, 1, (3,f)), (y, 1, 5), …

(z, 2, (3,t)),...

# Example: Def-Use Associations

def-use associations for variable z.

```
read (z)
x = 0
y = 0
if (z ≥ 0)
{
      x = sqrt (z)
      if (0 ≤ x && x ≤ 5)
            y = f (x)
      else
            y = h (z)
}
y = g (x, y)
print (y)
```

# Example: Def-Use Associations

```
read (z)
x = 0
y = 0
if (z ≥ 0)
{
    x = sqrt (z)
    if (0 ≤ x && x ≤ 5)
            y = f (x)
    else
            y = h (z)
}
y = g (x, y)
print (y)
```

def-use associations for variable x.

# Example: Def-Use Associations

```
read (z)
x = 0
y = 0
if (z ≥ 0)
{
        x = sqrt (z)
        if (0 ≤ x && x ≤ 5)
                y = f (x)
        else
                y = h (z)
}
y=g (x, y)
print (y)
```

def-use associations for variable y.

# Data-Flow Testing Strategies

- All **du** Paths (ADUP)
- All **Uses** (AU)
- Others not covered in this course …

# All du Paths Strategy (ADUP)

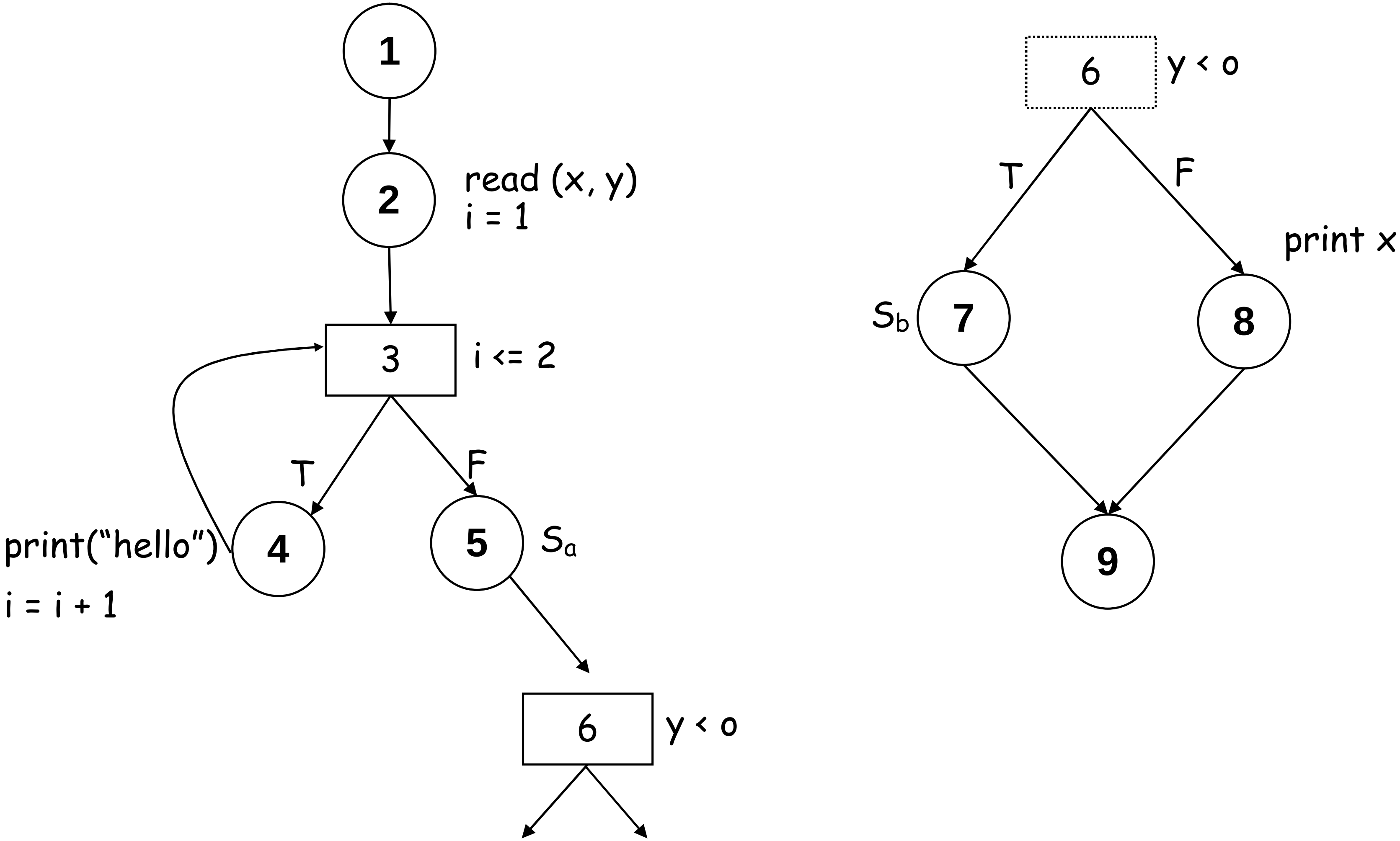
- **ADUP** is one of the strongest data-flow testing strategies.
- **ADUP** requires that every **du** path from every definition of every variable to every use of that definition be exercised under some test All **du** Paths Strategy (ADUP).

# An example: All-du-paths

What are all the du-paths in the following program ?

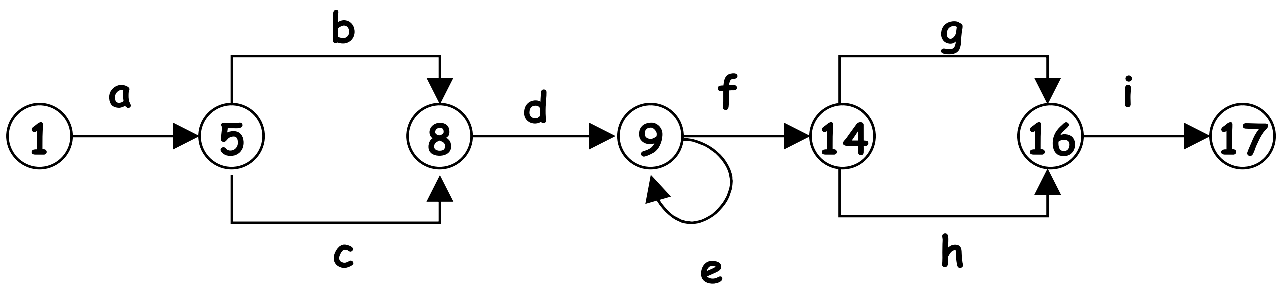
```
read (x,y);
for (i = 1; i <= 2; i++)
        print ("hello");
Sa;
if (y < 0)
        Sb;
else
        print (x);
```

# An example: All-du-paths



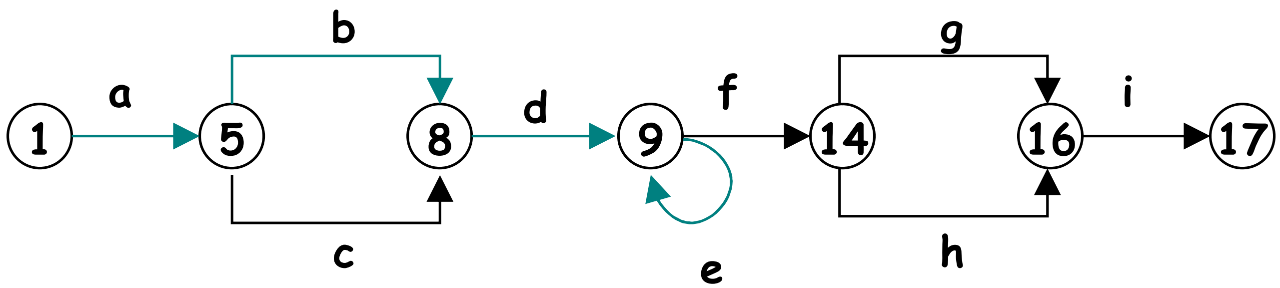
# Example: pow(x,y)

```
/* pow(x,y)
   This program computes x to the power of y, where x and y are integers.
    INPUT:     The x and y values.
    OUTPUT: x raised to the power of y is printed to stdout.
 */
void pow (int x, y)
{
float z;
int p;
if (y < 0)
    p = 0 – y;
else p = y;
z = 1.0;
while (p != 0)
    {
    z = z * x;
    p = p – 1;
    }
if (y < 0)
    z = 1.0 / z;
printf(z);
}
```



# Example: pow(x,y) du-Path for Variable x

```
/* pow(x,y)
   This program computes x to the power of y, where x and y are integers.
    INPUT:     The x and y values.
    OUTPUT: x raised to the power of y is printed to stdout.
 */
        void pow (int x, y)
        {
        float z;
        int p;
        if (y < 0)
           p = 0 – y;
        else p = y;
        z = 1.0;
        while (p != 0)
           {
           z = z * x;
           p = p – 1;
           }
        if (y < 0)
           z = 1.0 / z;
        printf(z);
        }
```



# Results of 2 of the 14 Ntafos Experiments

| Strategy | Mean Number of Test Cases | Percentage of Bugs Found |
|---|---|---|
| Random | 35 | 93.7 |
| Branch | 3.8 | 91.6 |
| All Uses | 11.3 | 96.3 |

| Strategy | Mean Number of Test Cases | Percentage of Bugs Found |
|---|---|---|
| Random | 100 | 79.5 |
| Branch | 34 | 85.5 |
| All Uses | 84 | 90.0 |

# Summary

- Data are as important as code.
- Data-flow testing strategies span the gap between **all paths** and **branch** testing.填补路径和分支测试的缝隙