Session 4
White-Box Testing

———3.5 Data Flow Testing

苏州大学计算机科学与技术学院 程宝雷

Dataflow Coverage

- Based on the idea that program paths along which variables are defined and then used should be covered.
- A family of path selection criteria has been defined, each providing a different degree of coverage.

Data Flow Testing

- definition-use path
- definition-clear path

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))

Variable Definitions and Uses (cont'd)

- 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...)

Variable Definitions and Uses (cont’d)

- 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 (cont'd)

- 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))

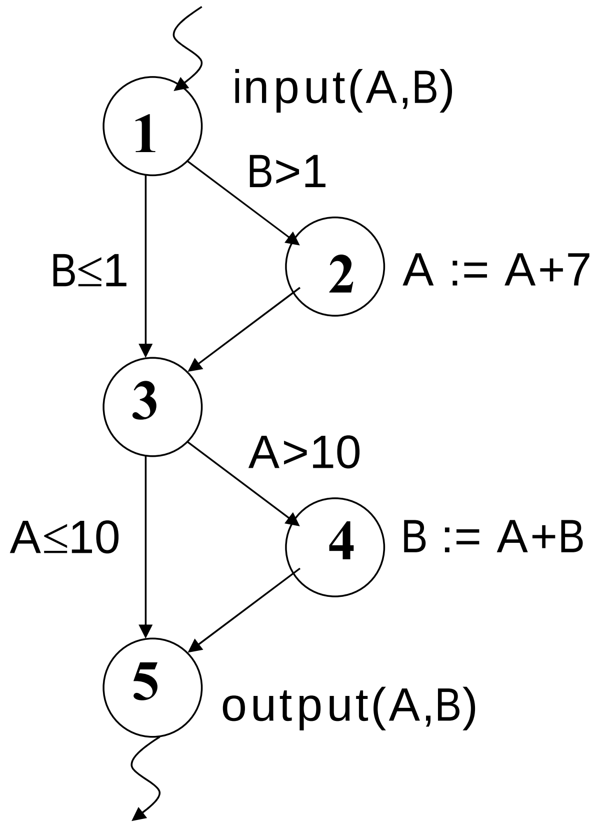
Other Dataflow Terms and Definitions

- A *definition-use pair* **(“du-pair”)** with respect to a variable v is a double (d,u) such that d is a node in the program’s flow graph at which v is defined, u is a node or edge at which v is used。

- Note: Broad definition.

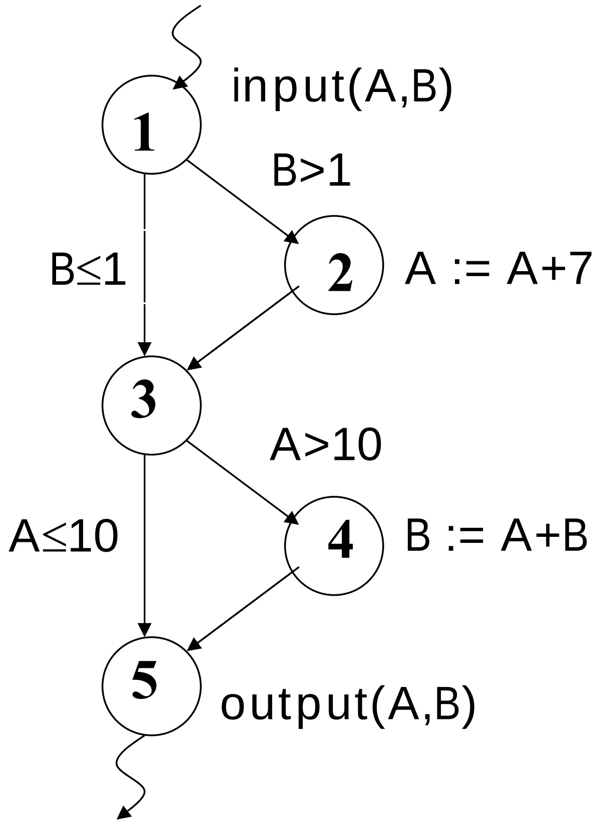
Example 1

```
1. input(A,B)
   if (B>1) then
2.     A := A+7
   end_if
3. if (A>10) then
4.     B := A+B
   end_if
5. output(A,B)
```



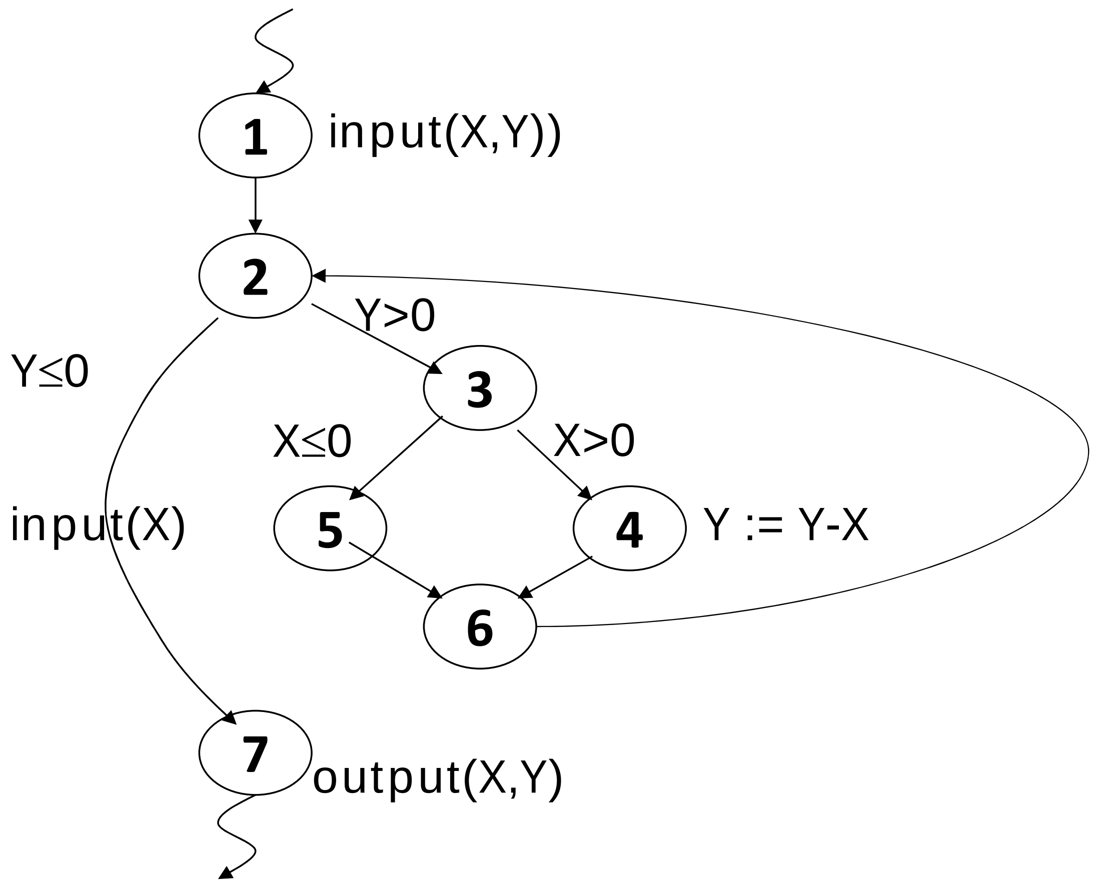
Identifying DU-Pairs – Variable A

du-pair	path(s)
(1,2)	<1,2>
(1,4)	<1,3,4>
(1,5)	<1,3,4,5>
	<1,3,5>
(1,<3,4>)	<1,3,4>
(1,<3,5>)	<1,3,5>
(2,4)	<2,3,4>
(2,5)	<2,3,4,5>
	<2,3,5>
(2,<3,4>)	<2,3,4>
(2,<3,5>)	<2,3,5>



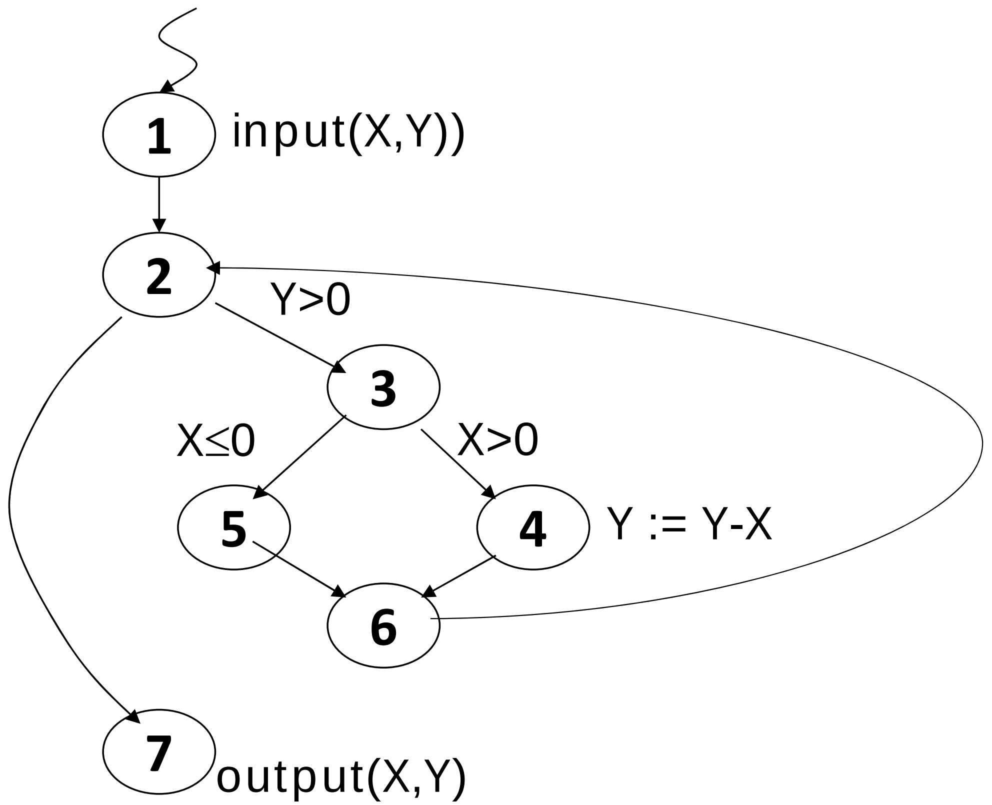
Exercise

```
1. input(X,Y)
2. while (Y>0) do
3.   if (X>0) then
4.      Y := Y-X
     else
5.      input(X)
    end_if_then_else
6. end_while
7. output(X,Y)
```



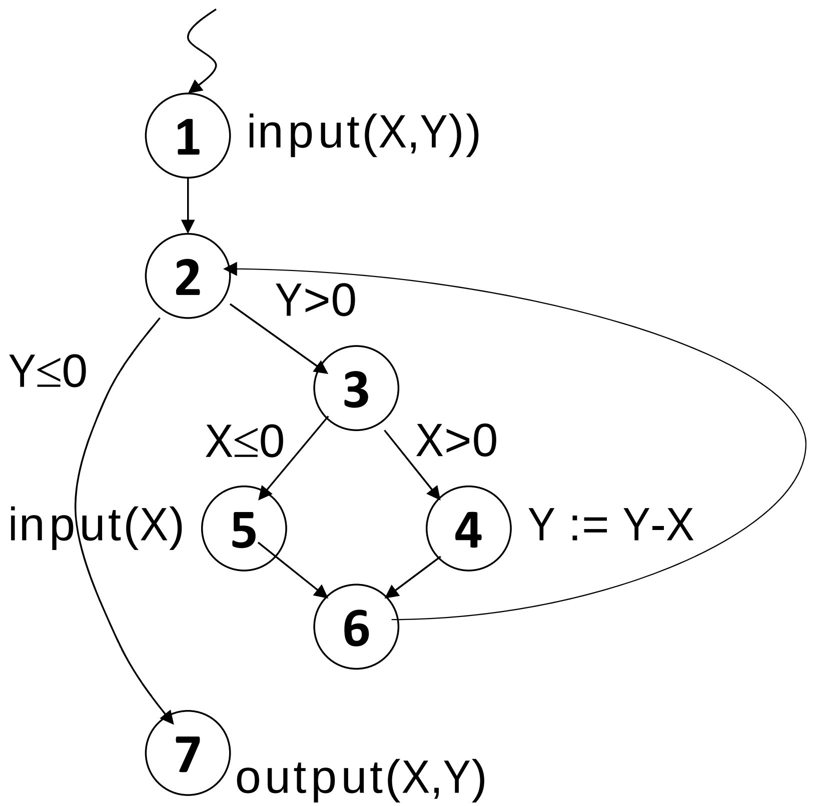
Identifying DU-Pairs – Variable X

du-pair	path(s)
(1,4)	<1,2,3,4>
	<1,2,3,4,(6,2,3,4)*>
(1,7)	<1,2,7>
	<1,2,3,4,6,2,7>
	<1,2,3,4,6,(2,3,4,6)*,2,7>
(1,<3,4>)	<1,2,3,4>
	<1,2,3,4,(6,2,3,4)*>
(1,<3,5>)	<1,2,3,5>
(5,4)	<5,6,2,3,4>
	<5,6,2,3,4,(6,2,3,4)*>



Identifying DU-Pairs – Variable X

du-pair	path(s)
(5,7)	<5,6,2,7>†
	<5,6,2,3,4,6,2,7>
	<5,6,2,3,4,6,(2,3,4,6)*,2,7>
(5,<3,4>)	<5,6,2,3,4>
	<5,6,2,3,4,(6,2,3,4)*>
(5,<3,5>)	<5,6,2,3,5>
	<5,6,2,3,4,6,2,3,5>†
	<5,6,2,3,4,6,(2,3,4,6)*,2,3,5> †
† infeasible	



Data Flow Testing

- definition-use path
- definition-clear path

def-clear path

- *A path is definition clear **("def-clear")** with respect to a variable v if there is no **re**-definition of v within the path.*
- A *definition-use pair* **("du-pair")** with respect to a variable v is a double (d,u) such that d is a node in the program's flow graph at which v is defined, u is a node or edge at which v is used, and there is a def-clear path with respect to v from d to u.
  - Rigorous definition 严谨版本定义

More Dataflow Terms and Definitions

- A path (either partial or complete) is **simple** if all edges within the path are distinct (i.e., different).
- A path is **loop-free** if all nodes within the path are distinct (i.e., different).

Simple and Loop-Free Paths

path	Simple?	Loop-free?
<1,3,4,2>	Yes	Yes
<1,2,3,2>	Yes	No
<1,2,3,1,2>	No	No
<1,2,3,2,4>	Yes	No

Simple and Loop-Free Paths (cont’d)

Which is *stronger*, **simple** or **loop-free**?

环路为1

无环路

More Dataflow Terms and Definitions

A path <n1,n2,...,nj,nk> is a **du-path** with respect to a variable v if v is defined at node n1 and either:

1. there is a **c-use** of v at node nk and <n1,n2,...,nj,nk> is a def-clear **simple** path,

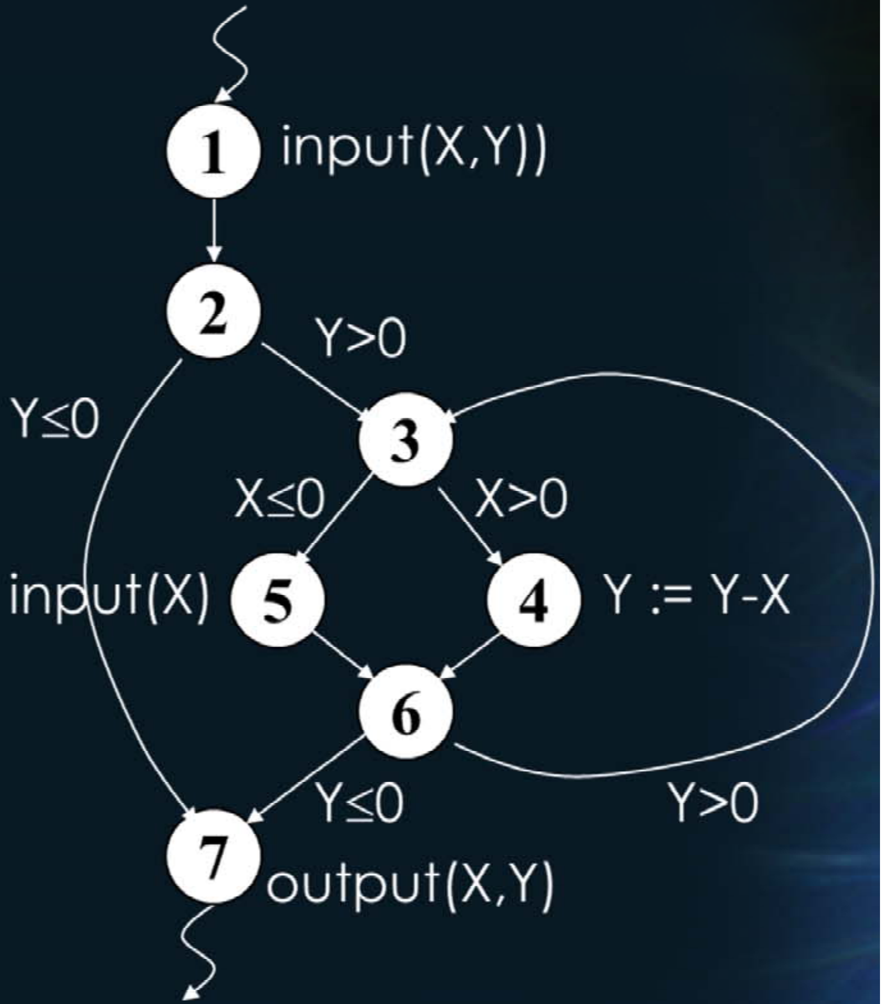
or

1. there is a **p-use** of v at edge <nj,nk> and <n1,n2,...nj> is a def-clear **loop-free** path.

NOTE!

Identifying du-paths

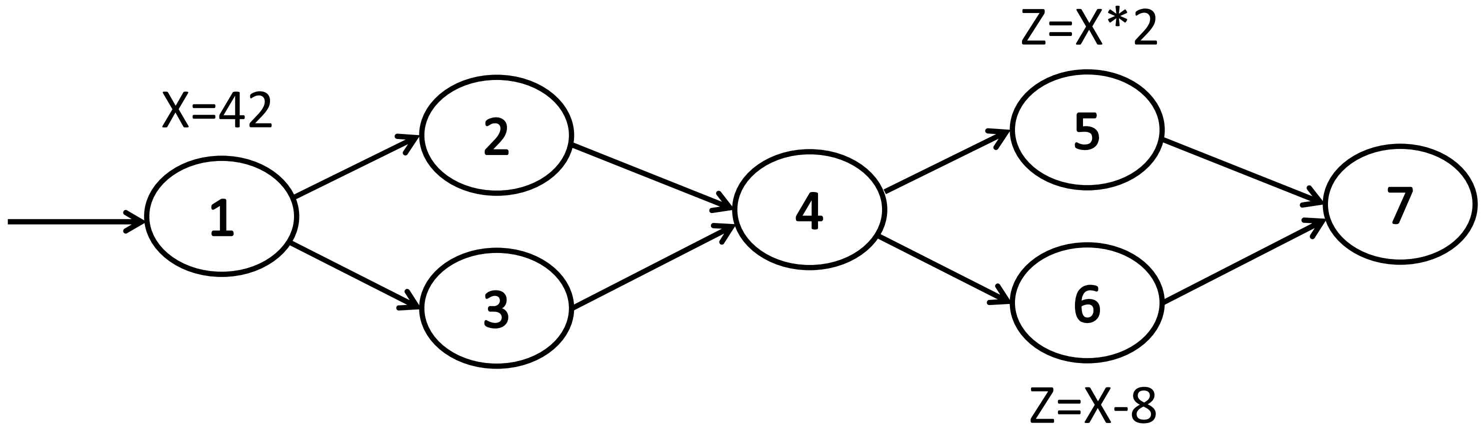
du-pair	path(s)	du-path?
X: (5,7)	<5,6,7> †	Yes
	<5,6,3,4,6,7>	Yes
	<5,6,3,4,6,(3,4,6)*,7>	No
X: (5,<3,4>)	<5,6,3,4>	Yes
	<5,6,3,4,(6,3,4)*>	No
X: (5,<3,5>)	<5,6,3,5>	Yes
	<5,6,3,4,6,3,5> †	No
	<5,6,3,4,6,(3,4,6)*,3,5> †	No
† infeasible		



Dataflow Test Coverage Criteria

- ***All-Defs:*** for **every program variable v**, **at least one def-clear path** from **every definition** of v to **at least one c-use or one p-use** of v must be covered.

Example

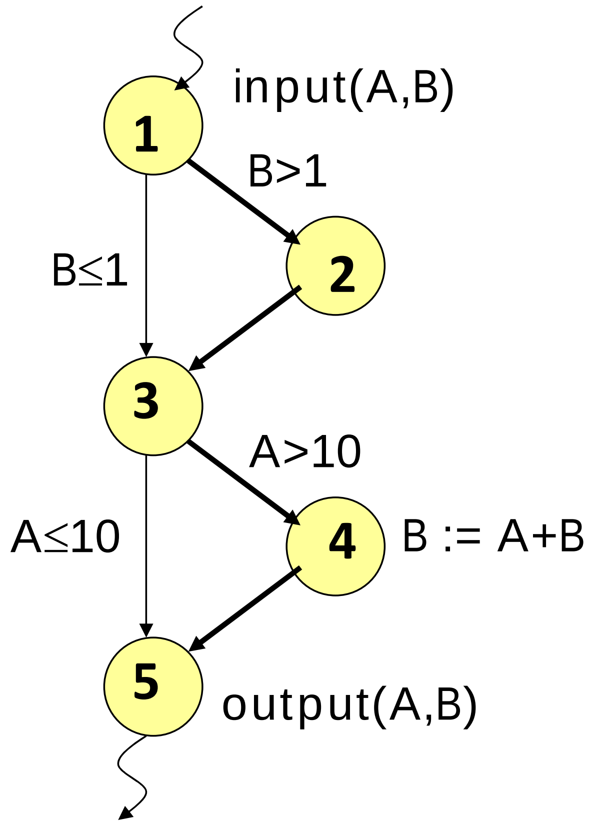


All-defs for *X*

<1,2,4,5>

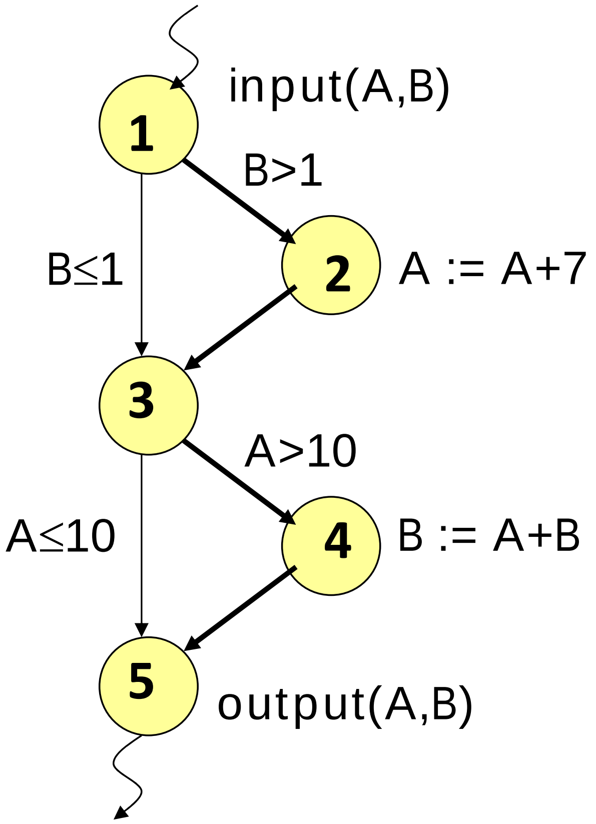
Dataflow Test Coverage Criteria (cont'd)

- Consider a test case executing path:
  1. **<1,2,3,4,5>**
- Identify all def-clear paths covered (i.e., subsumed) by this path for each variable.
- Are all definitions for each variable associated with at least one of the subsumed def-clear paths?



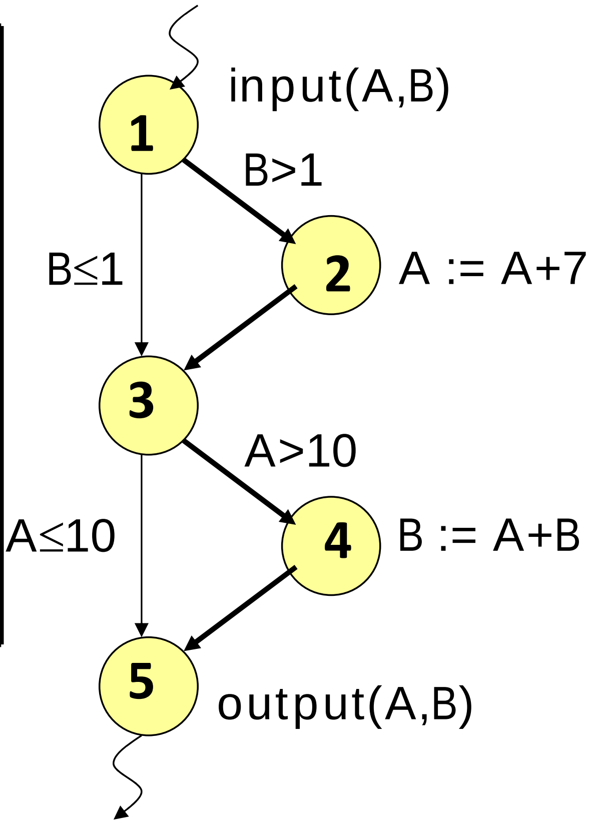
Def-Clear Paths Subsumed by <1,2,3,4,5> for Variable A

du-pair	path(s)
(1,2)	<1,2> ✓
(1,4)	<1,3,4>
(1,5)	<1,3,4,5>
	<1,3,5>
(1,<3,4>)	<1,3,4>
(1,<3,5>)	<1,3,5>
(2,4)	<2,3,4> ✓
(2,5)	<2,3,4,5> ✓
	<2,3,5>
(2,<3,4>)	<2,3,4> ✓
(2,<3,5>)	<2,3,5>



Def-Clear Paths Subsumed by <1,2,3,4,5> for Variable B

du-pair	**path(s)**
(1,4)	<1,2,3,4> ✓
	<1,3,4>
(1,5)	<1,2,3,5>
	<1,3,5>
(4,5)	<4,5> ✓
(1,<1,2>)	<1,2> ✓
(1,<1,3>)	<1,3>



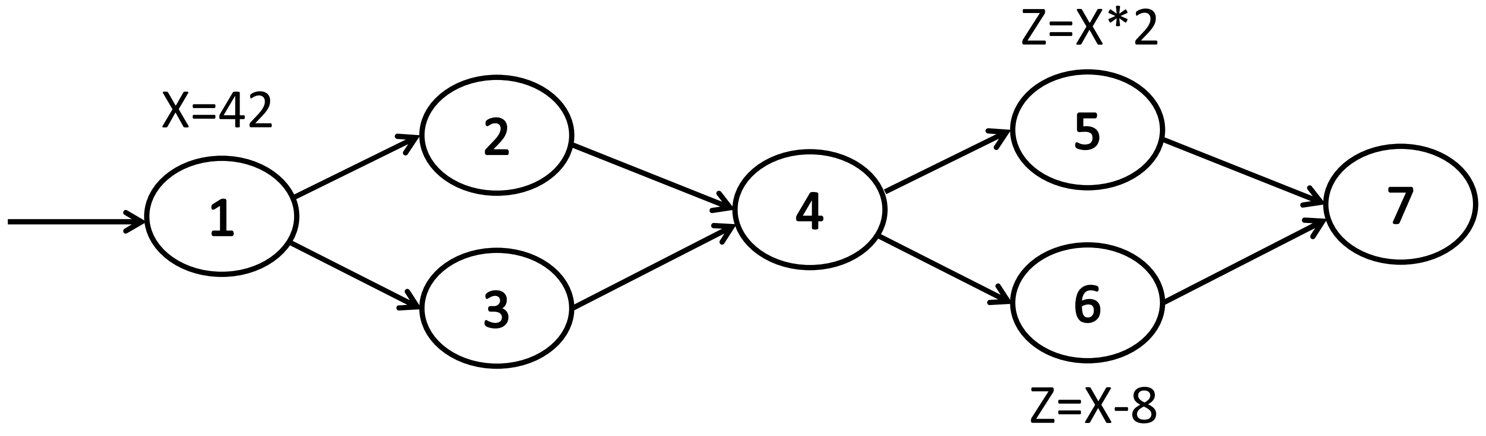
Dataflow Test Coverage Criteria (cont'd)

- Since **<1,2,3,4,5>** covers at least one def-clear path from every definition of A/B to at least one c-use or p-use of A/B, All-Defs coverage is achieved.

Dataflow Test Coverage Criteria (cont'd)

- ***All-Uses:*** for **every program variable v, at least one def-clear path** from **every definition** of v to **every c-use** and **every p-use** of v must be covered.

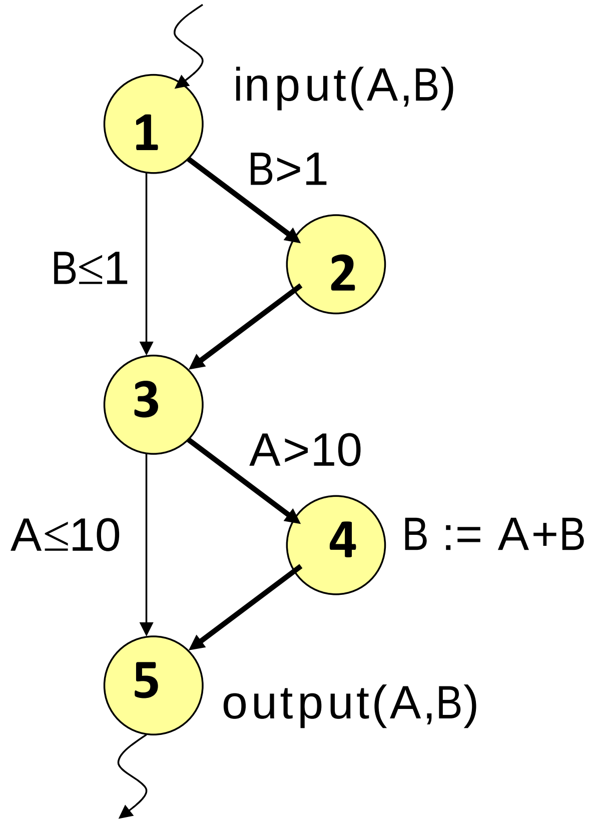
Example



All-defs for *X*	**All-uses for *X***
<1,2,4,5>	<1,2,4,5>
	<1,2,4,6>

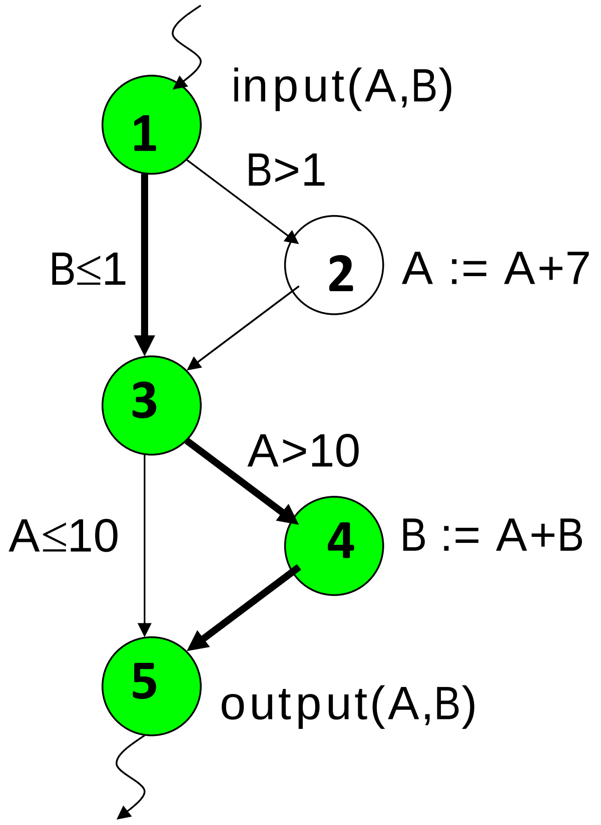
Dataflow Test Coverage Criteria (cont’d)

- Consider additional test cases executing paths:
  1. **<1,2,3,4,5>**
  2. **<1,3,4,5>**
  3. **<1,2,3,5>**
- Do all three test cases provide All-Uses coverage?



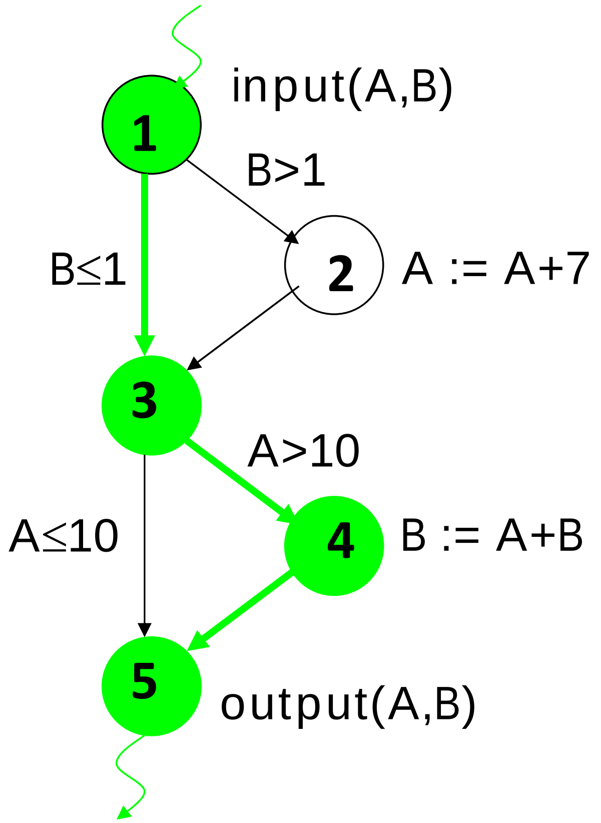
Def-Clear Paths Subsumed by <1,3,4,5> for Variable A

du-pair	path(s)
(1,2)	<1,2> ✓
(1,4)	<1,3,4> ✓
(1,5)	<1,3,4,5> ✓
	<1,3,5>
(1,<3,4>)	<1,3,4> ✓
(1,<3,5>)	<1,3,5>
(2,4)	<2,3,4> ✓
(2,5)	<2,3,4,5> ✓
	<2,3,5>
(2,<3,4>)	<2,3,4> ✓
(2,<3,5>)	<2,3,5>



Def-Clear Paths Subsumed by <1,3,4,5> for Variable B

du-pair	path(s)
(1,4)	<1,2,3,4> ✓
	<1,3,4> ✓
(1,5)	<1,2,3,5>
	<1,3,5>
(4,5)	<4,5> ✓ ✓
(1,<1,2>)	<1,2> ✓
(1,<1,3>)	<1,3> ✓



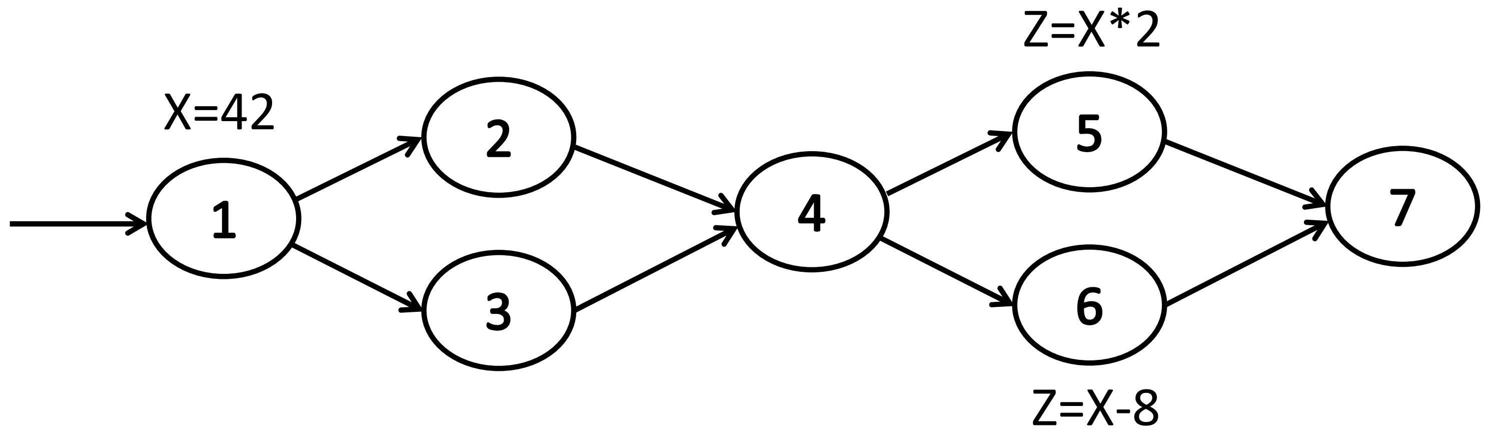
Dataflow Test Coverage Criteria (cont’d)

- Since none of the three test cases covers the du-pair (1,<3,5>) for variable A, All-Uses Coverage is not provided.

Another Dataflow Test Coverage Criterion

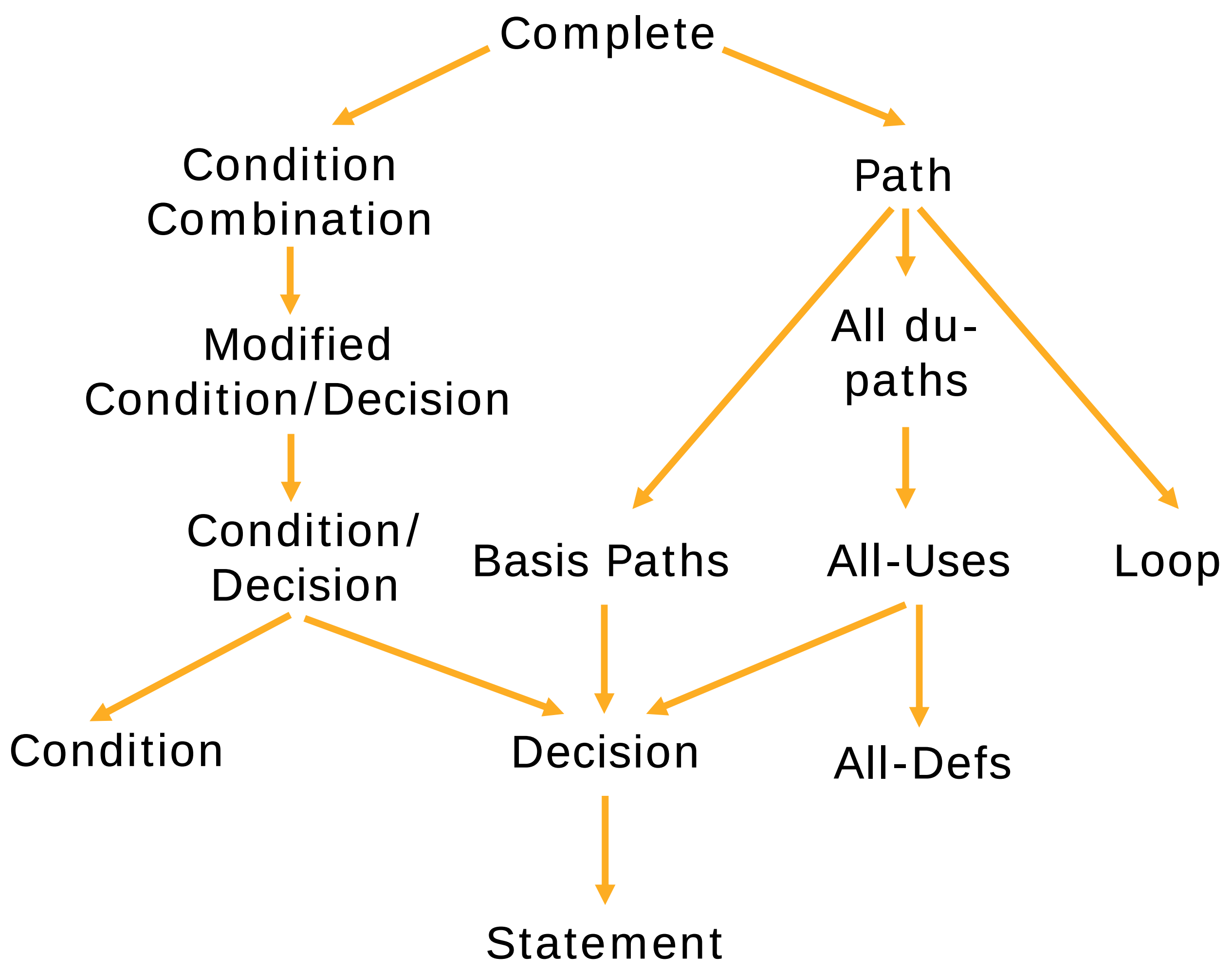
- ***All-DU-Paths:*** for **every program variable v, every du-path** from **every definition** of v to **every c-use** and **every p-use** of v must be covered.

Example



All-defs for *X*	All-uses for *X*	All-DU-Paths for *X*
<1,2,4,5>	<1,2,4,5>	<1,2,4,5>
	<1,2,4,6>	<1,3,4,5>
		<1,2,4,6>
		<1,3,4,6>

Summary of White-Box Coverage Relationships


Complete
Condition Combination
Path
Modified Condition/Decision
All du-paths
Condition/ Decision
Basis Paths
All-Uses
Loop
Condition
Decision
All-Defs
Statement