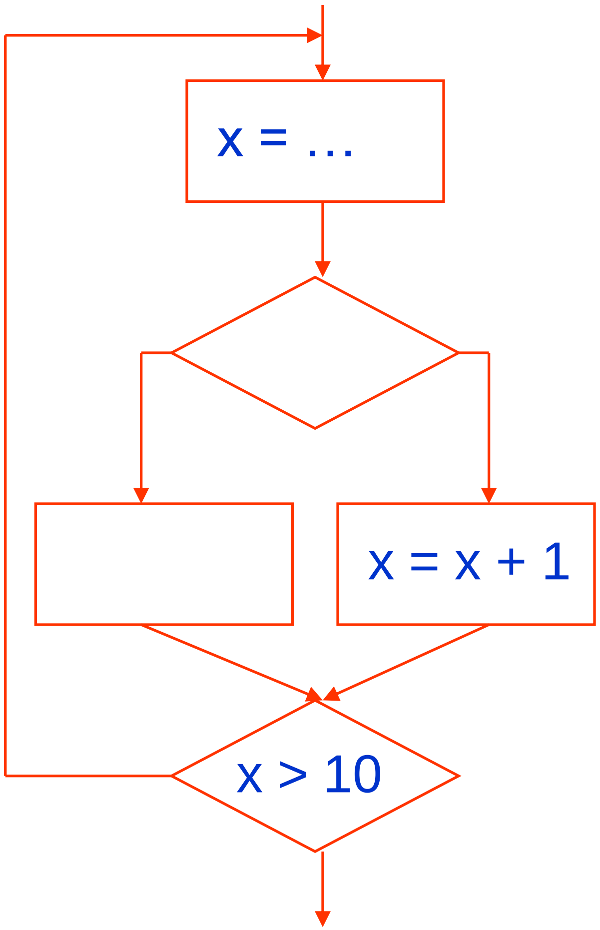
Data Flow Testing

Data Flow Testing

- Data flow testing uses the control flow graph to explore the unreasonable things that can happen to data (data flow anomalies).
- Data flow anomalies are detected based on the associations between values and variables.
  - Variables are used without being initialized.
  - Initialized variables are not used once.

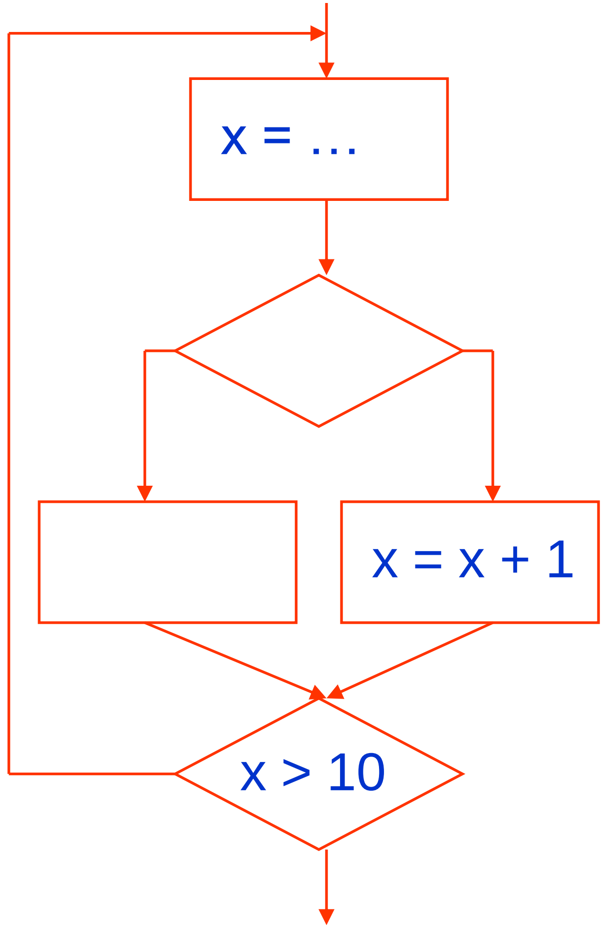
Definitions and Uses of Variables

- An occurrence of a variable in the program is a definition of the variable if a value is bound to the variable at that occurrence.
- An occurrence of a variable in the program is a use of the variable if the value of the variable is referred at that occurrence.



Predicate Uses and Computation Uses

- A use of a variable is a predicate use (p-use) if the variable is in a predicate and its value is used to decide an execution path.
- A use of a variable is a computation use (c-use) if the value of the variable is used to compute a value for defining another variable or as an output value.

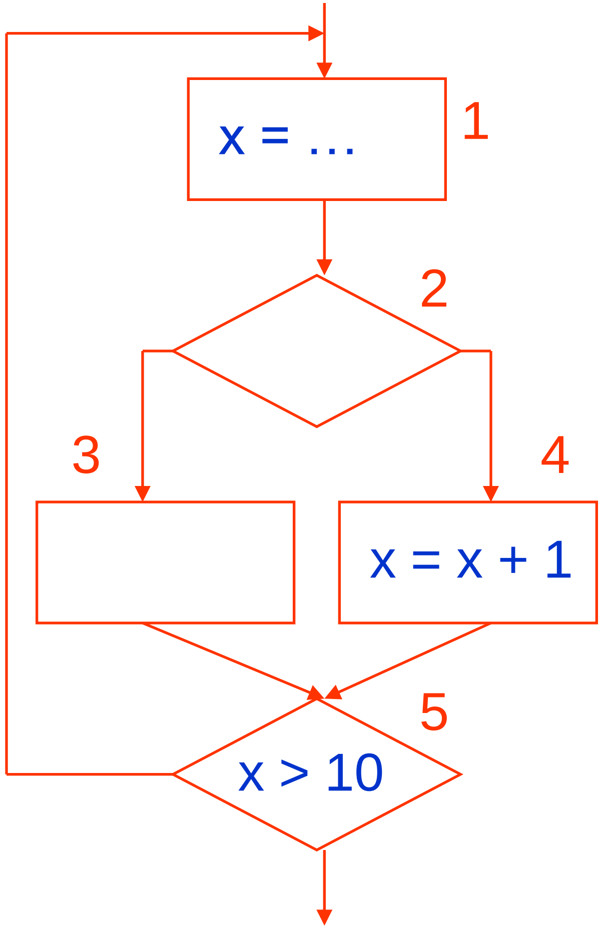


Definition Clear Paths

- A path (i, n_1, n_2, …, n_m, j) is a definition-clear path for a variable x from i to j if n_1 through n_m do not contain a definition of x.

(1, 2, 4)

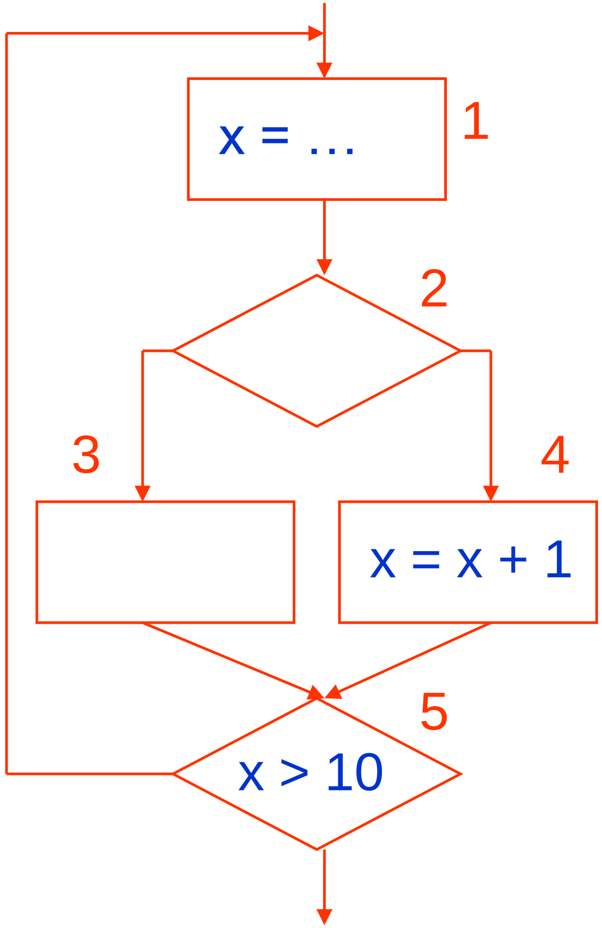
(1, 2, 3, 5)



Definition-C-Use Associations

- Given a definition of x in node n_d and a c-use of x in node $n_{c\text{-}use}$, the presence of a definition-clear path for x from n_d to $n_{c\text{-}use}$ establishes the definition-c-use association (n_d, $n_{c\text{-}use}$, x).

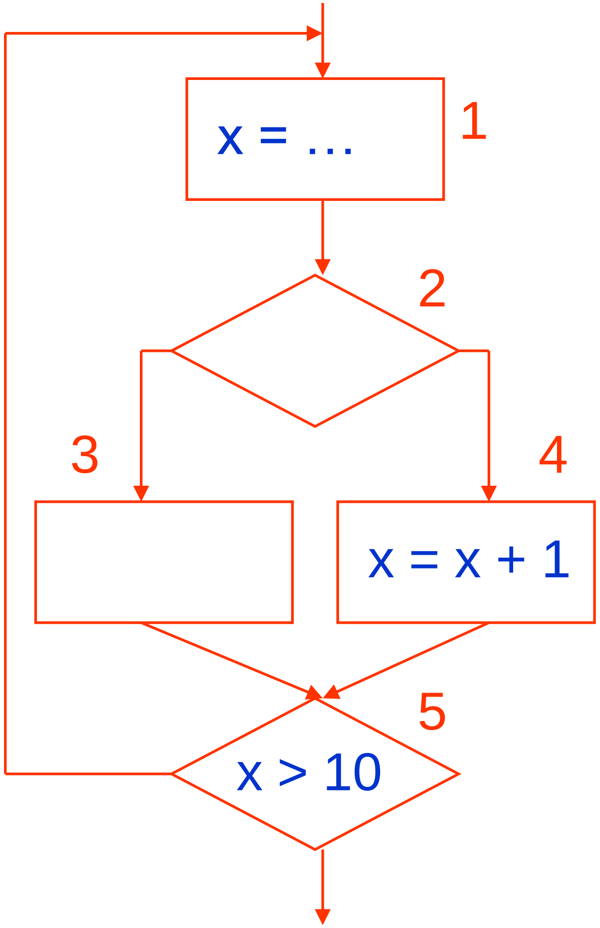
  (1, 4, x)



Definition-P-Use Associations

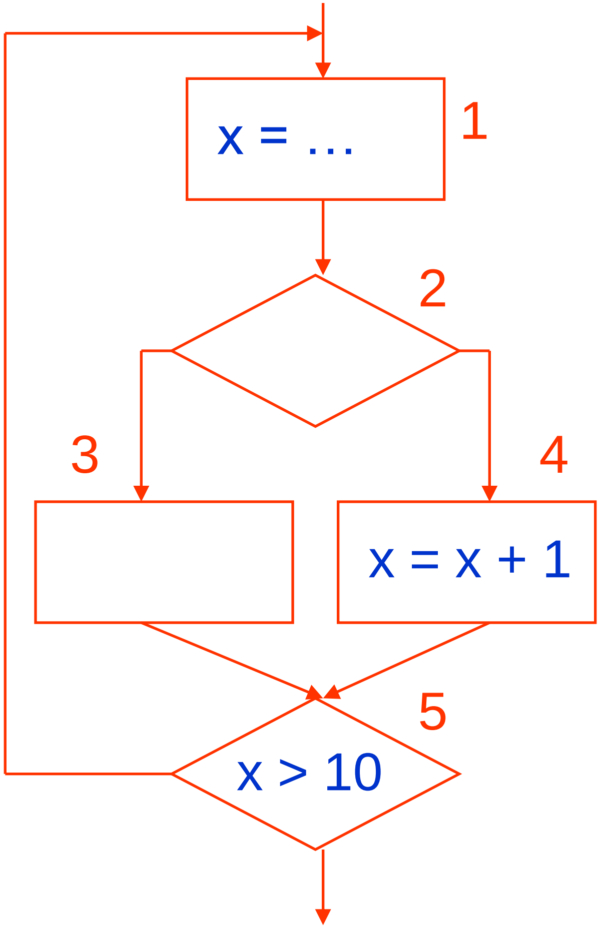
- Given a definition of x in node n_d and a p-use of x in node $n_{p\text{-}use}$, the presence of a definition-clear path for x from n_d to $n_{p\text{-}use}$ establishes a pair of definition-p-use associations (n_d, ($n_{p\text{-}use}$, *t*), x) and (n_d, ($n_{p\text{-}use}$, *f*), x) .

(1, (5, *t*), x)   (1, (5, *f*), x)



DU-Paths

- A path (n_1, …, n_m) is a du-path for variable x if n_1 contains a definition of x and either n_m has a c-use of x and (n_1, …, n_m) is a definition-clear simple path for x (all nodes, except possibly n_1 and n_m, are distinct) or is a p-use of x and is a definition-clear loop-free path for x (all nodes are distinct) .

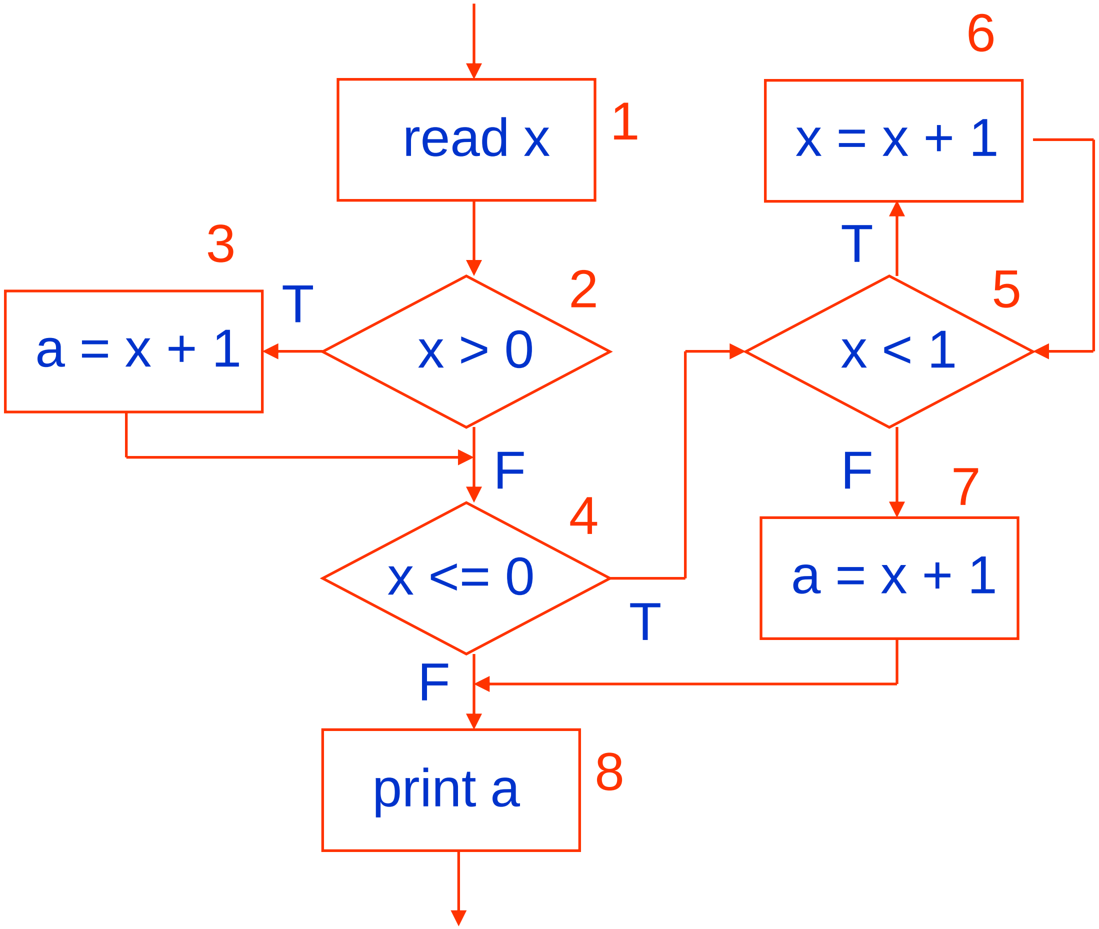


(1, 2, 4)   (1, 2, 3, 5)

Test Coverage Criteria

- All-defs coverage
- All-c-uses coverage
- All-c-uses/some-p-uses coverage
- All-p-uses coverage
- All-p-uses/some-c-uses coverage
- All-uses coverage
- All-du-paths coverage

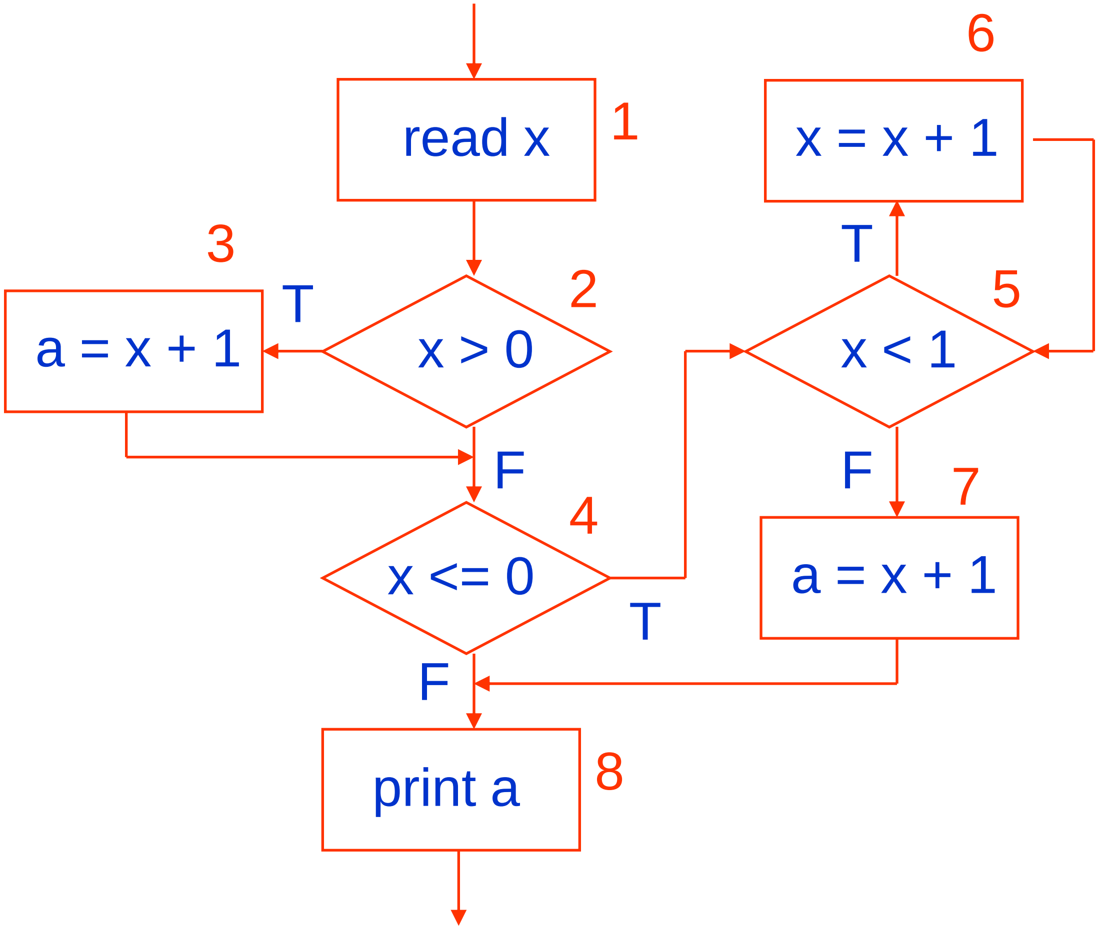
A Running Example



x = 1
P_1: (1, 2, 3, 4, 8)
a = 2

x = -1
P_2: (1, 2, 4, 5, 6, 5, 6, 5, 7, 8)
a = 2

A Running Example

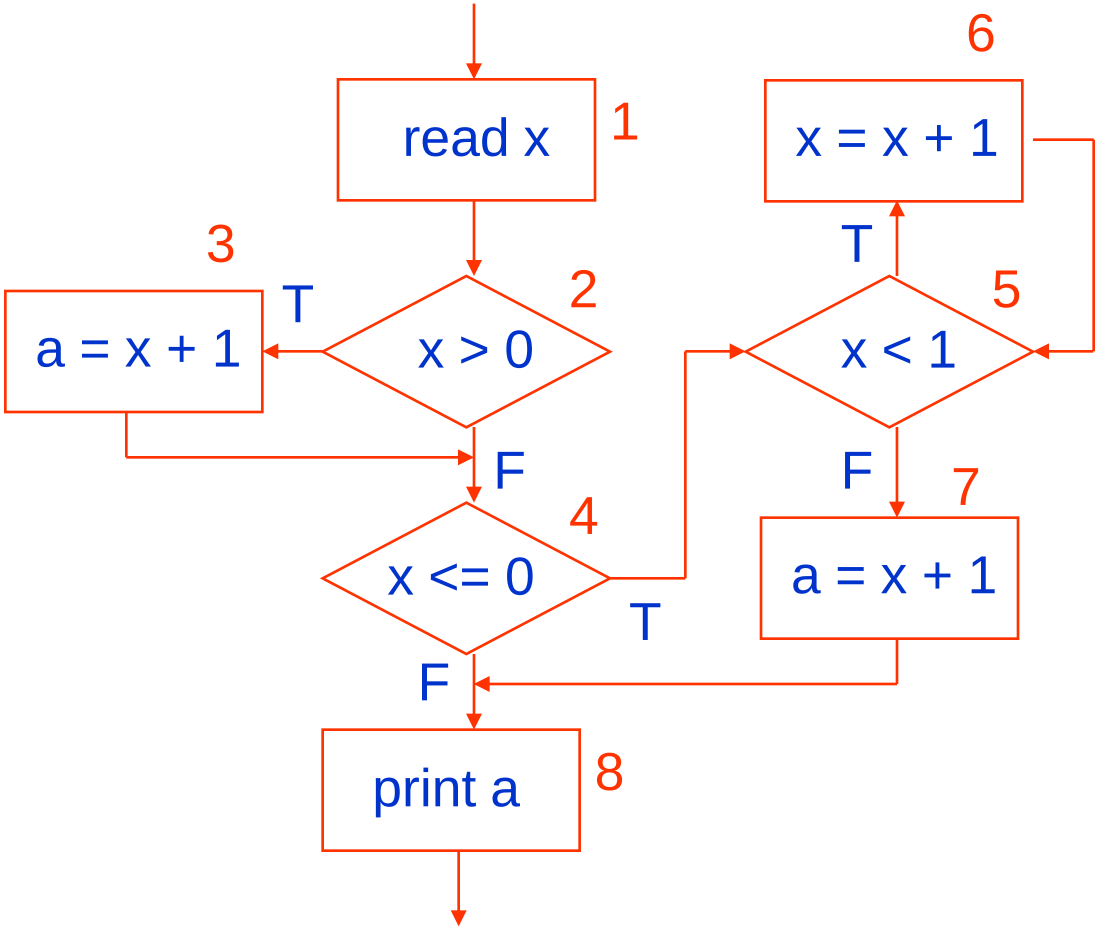


Associations:
(1, (2, *t*), x)
(1, (2, *f*), x)
(1, 3, x)
(1, (4, *t*), x)
(1, (4, *f*), x)
(1, (5, *t*), x)
(1, (5, *f*), x)
(1, 6, x)
(1, 7, x)
(3, 8, a)
(6, 6, x)
(6, 7, x)
(6, (5, *t*), x)
(6, (5, *f*), x)
(7, 8, a)

All-Defs Coverage

- Test cases include a definition-clear path from every definition to some corresponding use (c-use or p-use).

All-Defs Coverage

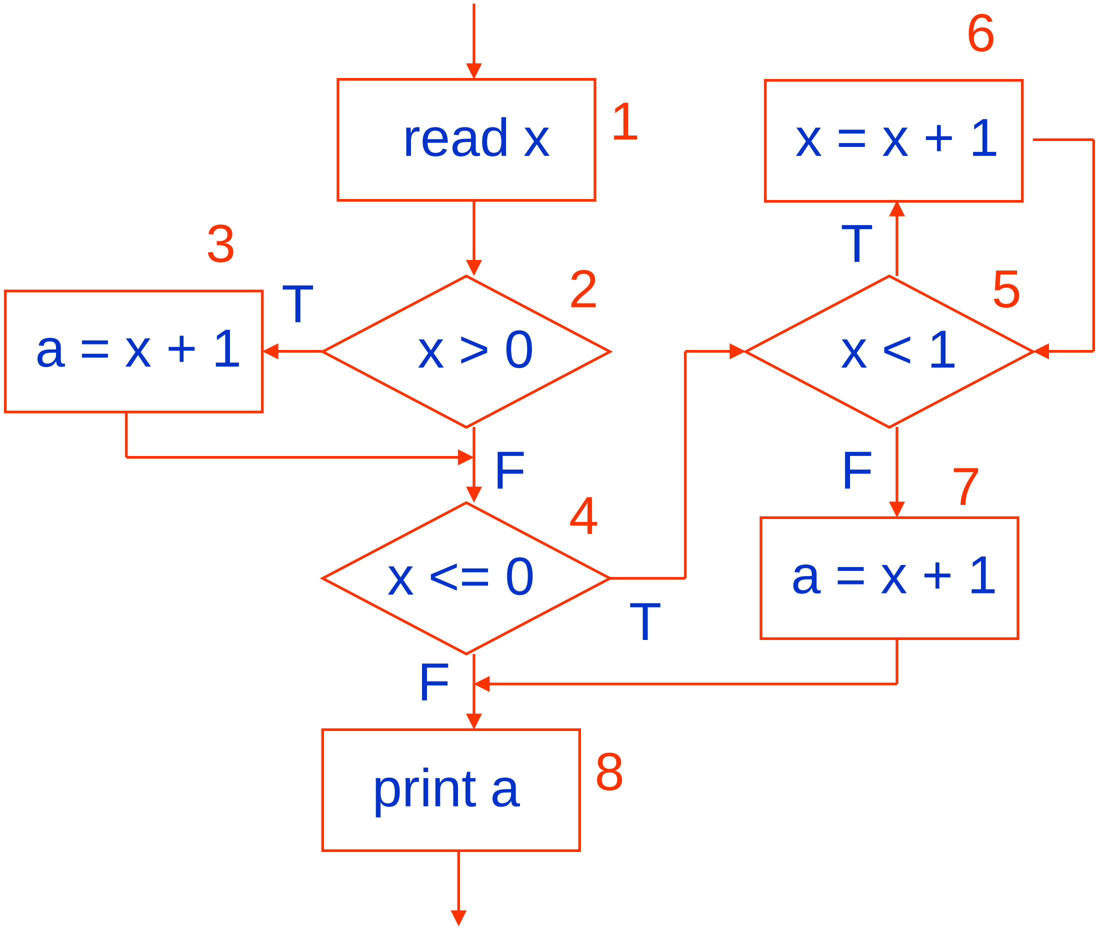


Associations	all-defs
(1, (2, *t*), x)	√
(1, (2, *f*), x)	
(1, 3, x)	
(1, (4, *t*), x)	
(1, (4, *f*), x)	
(1, (5, *t*), x)	
(1, (5, *f*), x)	
(1, 6, x)	
(1, 7, x)	
(3, 8, a)	√
(6, 6, x)	√
(6, 7, x)	
(6, (5, *t*), x)	
(6, (5, *f*), x)	
(7, 8, a)	√
Paths	$\{P_1, P_2\}$

All-C-Uses Coverage

- Test cases include a definition-clear path from every definition to all of its corresponding c-uses.

All-C-Uses Coverage

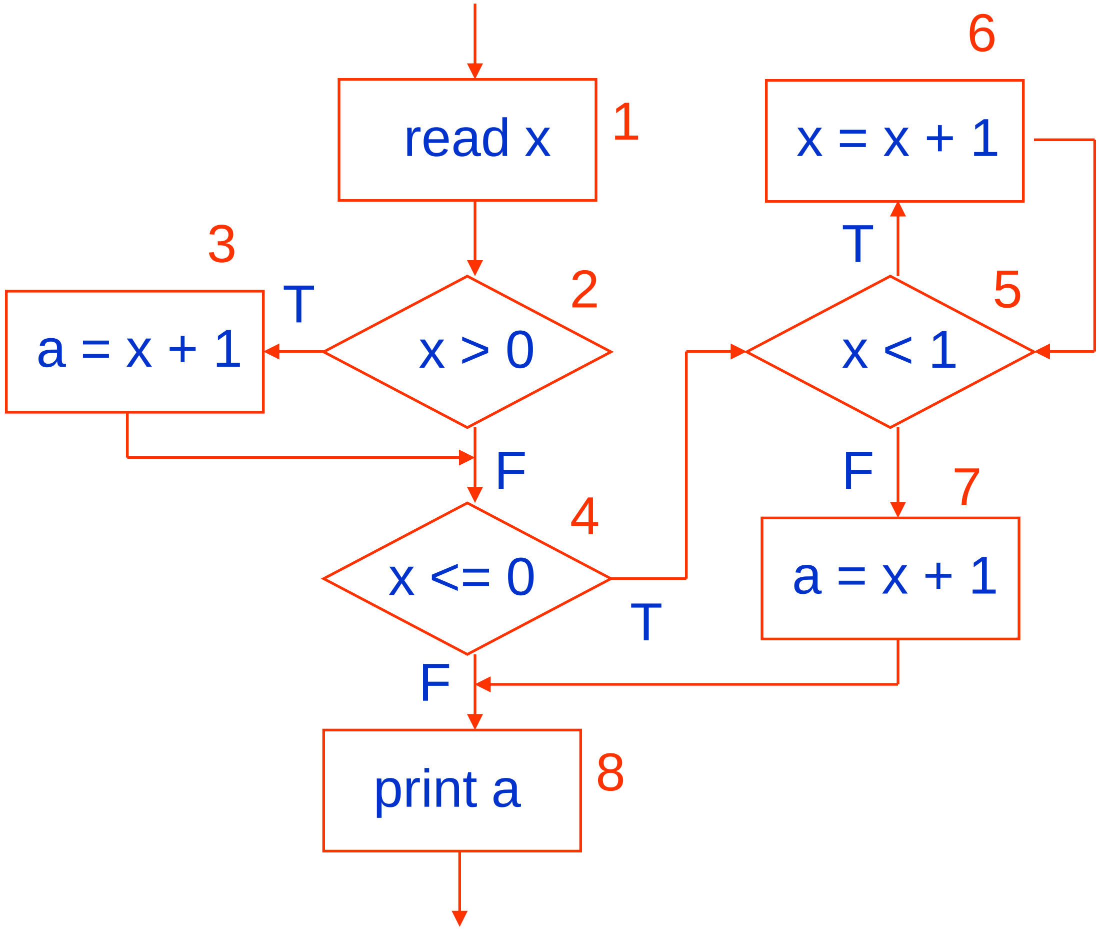


Associations	all-c-uses
(1, (2, *t*), x)	
(1, (2, *f*), x)	
(1, 3, x)	√
(1, (4, *t*), x)	
(1, (4, *f*), x)	
(1, (5, *t*), x)	
(1, (5, *f*), x)	
(1, 6, x)	√
(1, 7, x)	√
(3, 8, a)	√
(6, 6, x)	√
(6, 7, x)	√
(6, (5, *t*), x)	
(6, (5, *f*), x)	
(7, 8, a)	√
Paths	$\{P_1, P_2\}$

All-P-Uses Coverage

- Test cases include a definition-clear path from every definition to all of its corresponding p-uses.

All-P-Uses Coverage

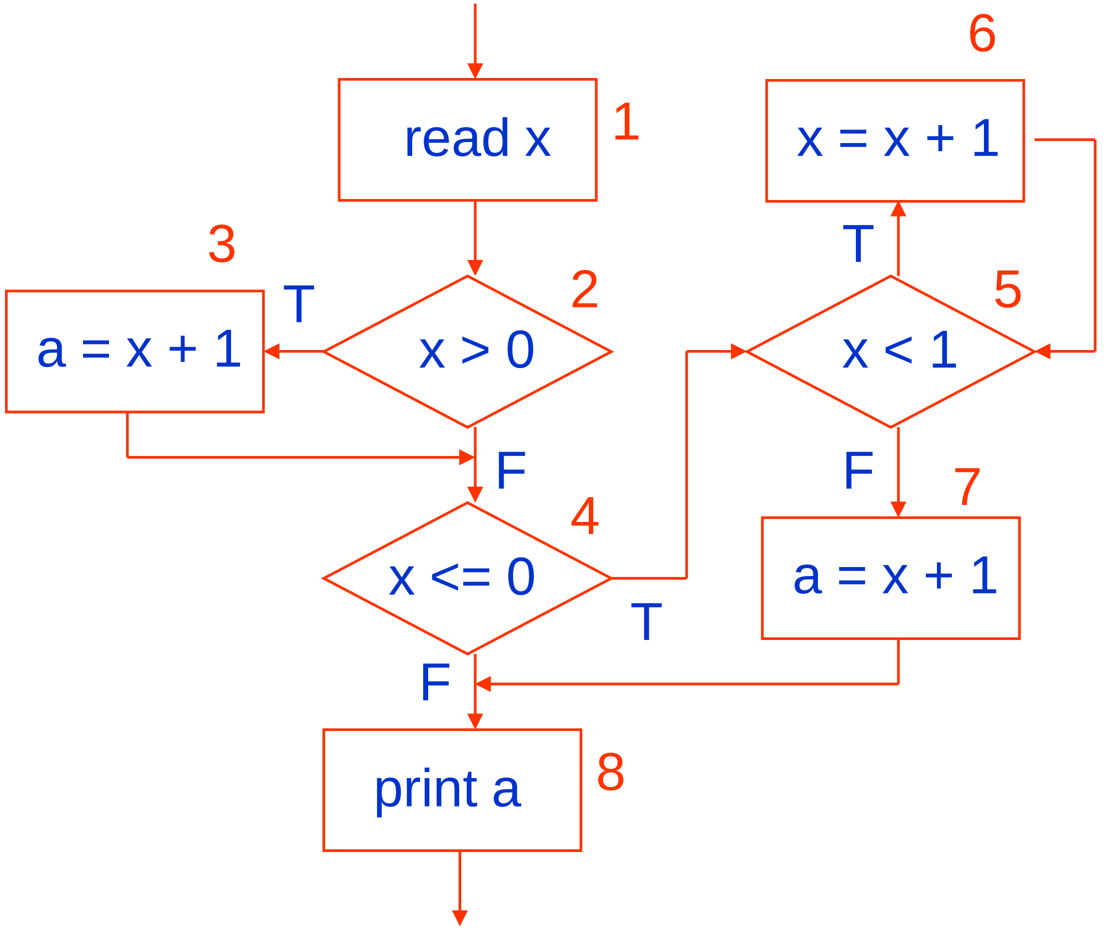


Associations	all-p-uses
(1, (2, *t*), x)	√
(1, (2, *f*), x)	√
(1, 3, x)	
(1, (4, *t*), x)	√
(1, (4, *f*), x)	√
(1, (5, *t*), x)	√
(1, (5, *f*), x)	√
(1, 6, x)	
(1, 7, x)	
(3, 8, a)	
(6, 6, x)	
(6, 7, x)	
(6, (5, *t*), x)	√
(6, (5, *f*), x)	√
(7, 8, a)	
Paths	$\{P_1, P_2\}$

All-C-Uses/Some-P-Uses Coverage

- Test cases include a definition-clear path from every definition to all of its corresponding c-uses. In addition, if a definition has no c-use, then test cases include a definition-clear path to some p-use.

All-C-Uses/Some-P-Uses Coverage

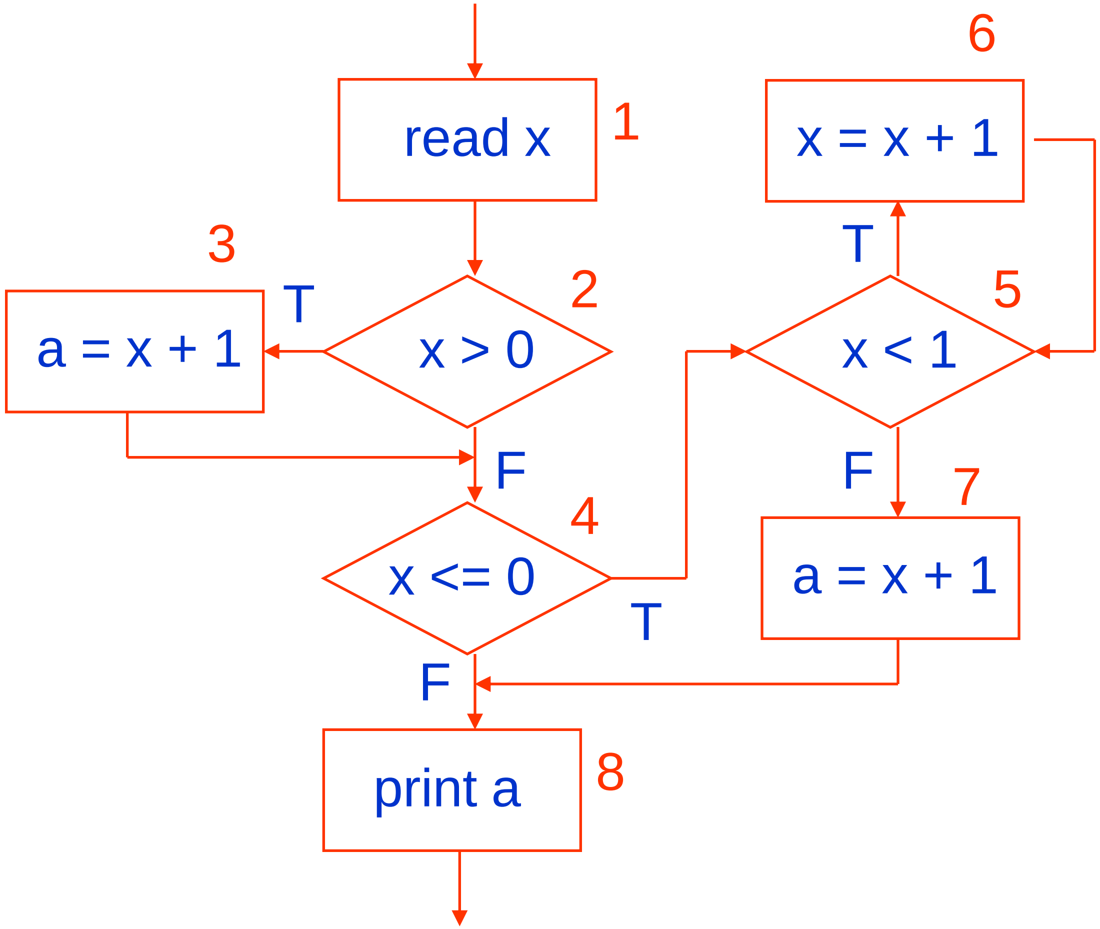


Associations	all-c-uses/ some-p-uses
(1, (2, *t*), x)	
(1, (2, *f*), x)	
(1, 3, x)	√
(1, (4, *t*), x)	
(1, (4, *f*), x)	
(1, (5, *t*), x)	
(1, (5, *f*), x)	
(1, 6, x)	√
(1, 7, x)	√
(3, 8, a)	√
(6, 6, x)	√
(6, 7, x)	√
(6, (5, *t*), x)	
(6, (5, *f*), x)	
(7, 8, a)	√
Paths	$\{P_1, P_2\}$

All-P-Uses/Some-C-Uses Coverage

- Test cases include a definition-clear path from every definition to all of its corresponding p-uses. In addition, if a definition has no p-use, then test cases include a definition-clear path to some c-use.

All-P-Uses/Some-C-Uses Coverage

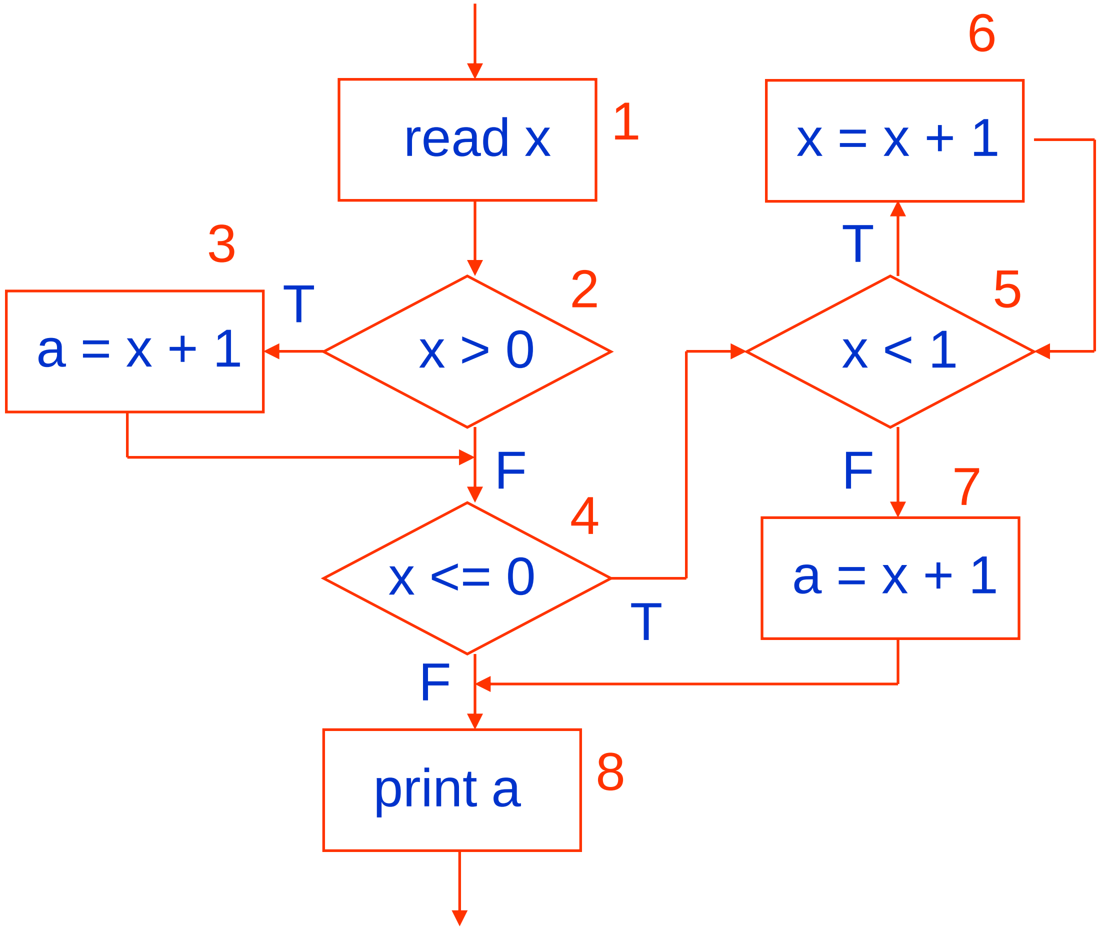


Associations	all-p-uses/ some-c-uses
(1, (2, *t*), x)	√
(1, (2, *f*), x)	√
(1, 3, x)	
(1, (4, *t*), x)	√
(1, (4, *f*), x)	√
(1, (5, *t*), x)	√
(1, (5, *f*), x)	√
(1, 6, x)	
(1, 7, x)	
(3, 8, a)	√
(6, 6, x)	
(6, 7, x)	
(6, (5, *t*), x)	√
(6, (5, *f*), x)	√
(7, 8, a)	√
Paths	$\{P_1, P_2\}$

All-Uses Coverage

- Test cases include a definition-clear path from every definition to each of its uses including both c-uses and p-uses.

All-Uses Coverage

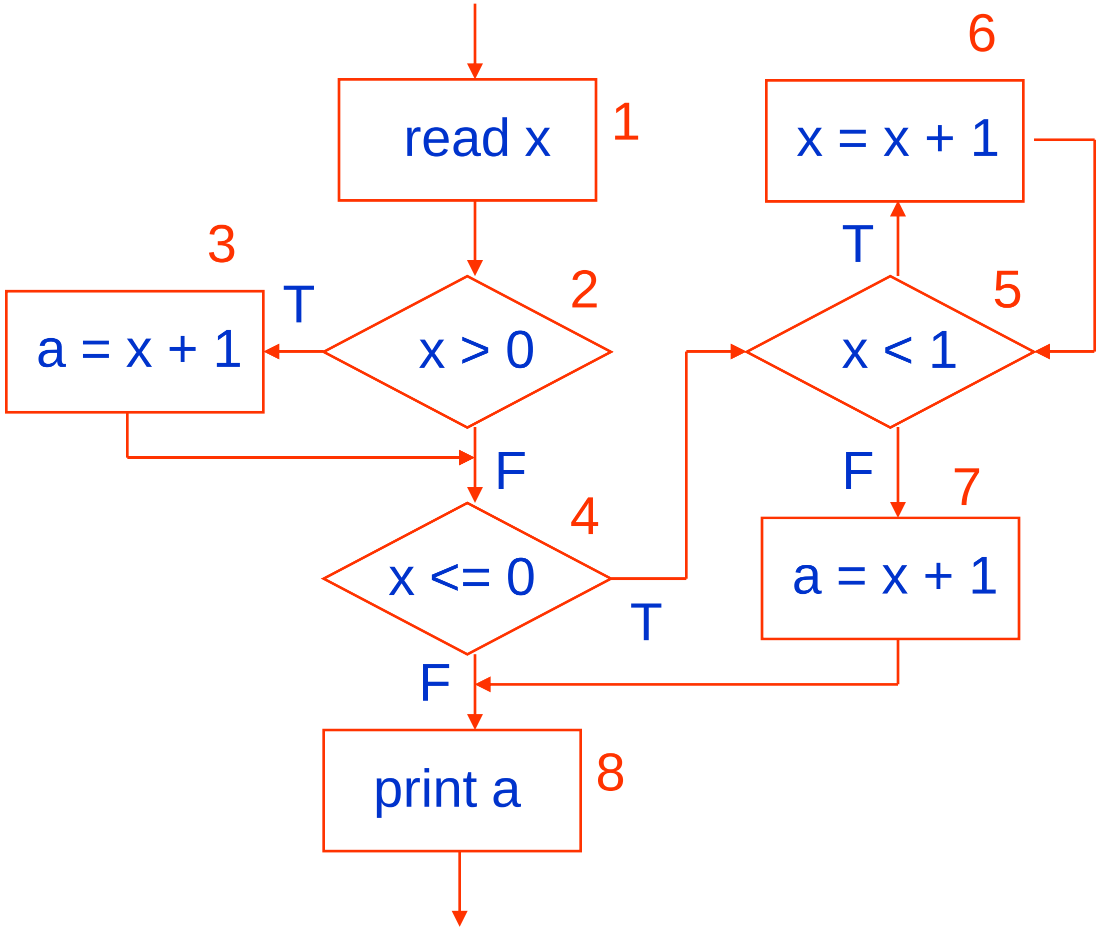


Associations	all-uses
(1, (2, *t*), x)	√
(1, (2, *f*), x)	√
(1, 3, x)	√
(1, (4, *t*), x)	√
(1, (4, *f*), x)	√
(1, (5, *t*), x)	√
(1, (5, *f*), x)	√
(1, 6, x)	√
(1, 7, x)	√
(3, 8, a)	√
(6, 6, x)	√
(6, 7, x)	√
(6, (5, *t*), x)	√
(6, (5, *f*), x)	√
(7, 8, a)	√
Paths	$\{P_1, P_2\}$

All-DU-Paths Coverage

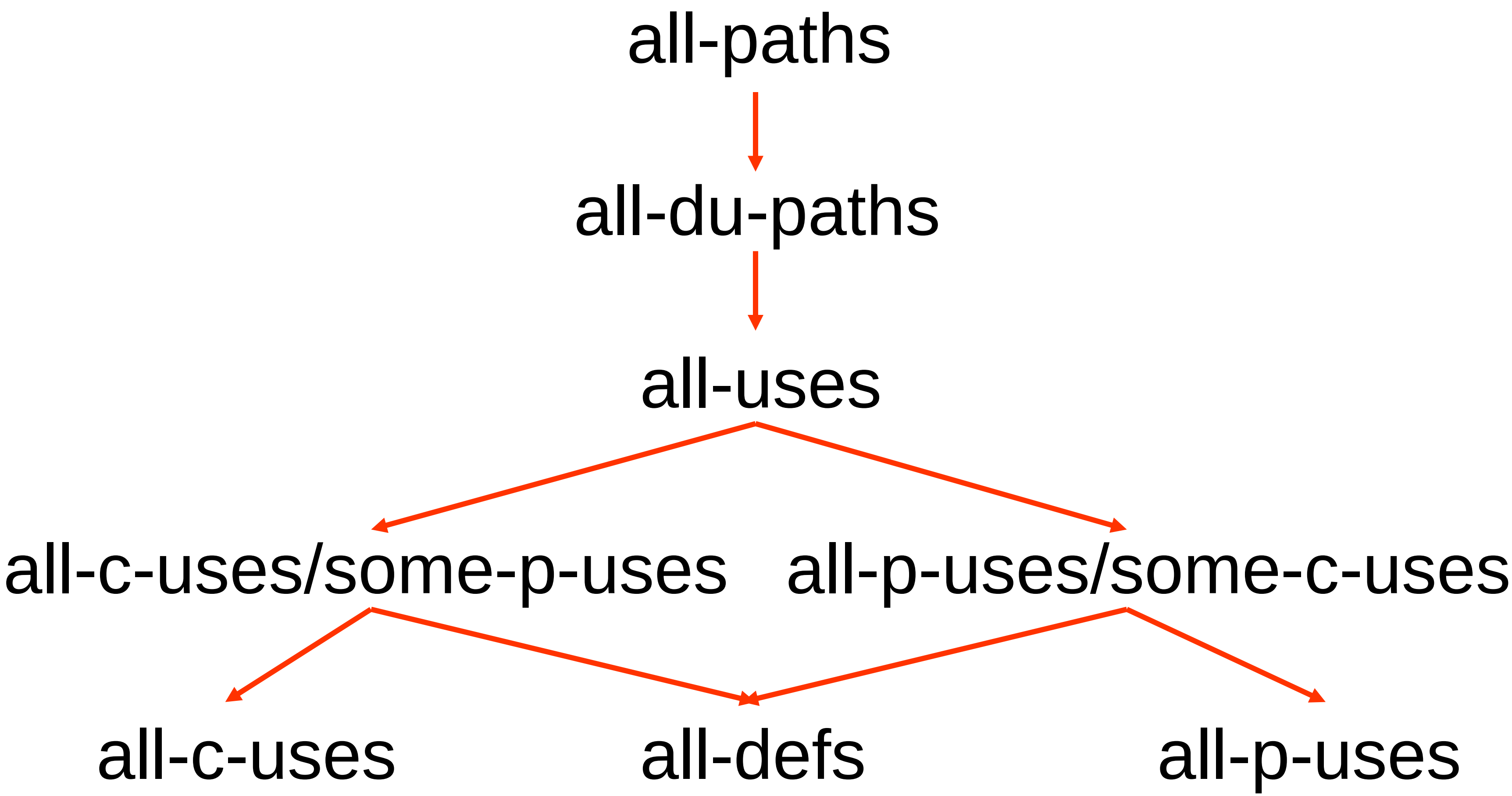
- Test cases include all du-paths for each definition. Therefore, if there are multiple paths between a given definition and a use, they must all be included.

All-DU-Paths Coverage



Associations	all-du-paths
(1, (2, *t*), x)	√
(1, (2, *f*), x)	√
(1, 3, x)	√
(1, (4, *t*), x)	√
(1, (4, *f*), x)	√
(1, (5, *t*), x)	√
(1, (5, *f*), x)	√
(1, 6, x)	√
(1, 7, x)	√
(3, 8, a)	√
(6, 6, x)	√
(6, 7, x)	√
(6, (5, *t*), x)	√
(6, (5, *f*), x)	√
(7, 8, a)	√
Paths	$\{P_1, P_2\}$

Test Coverage Criteria Hierarchy



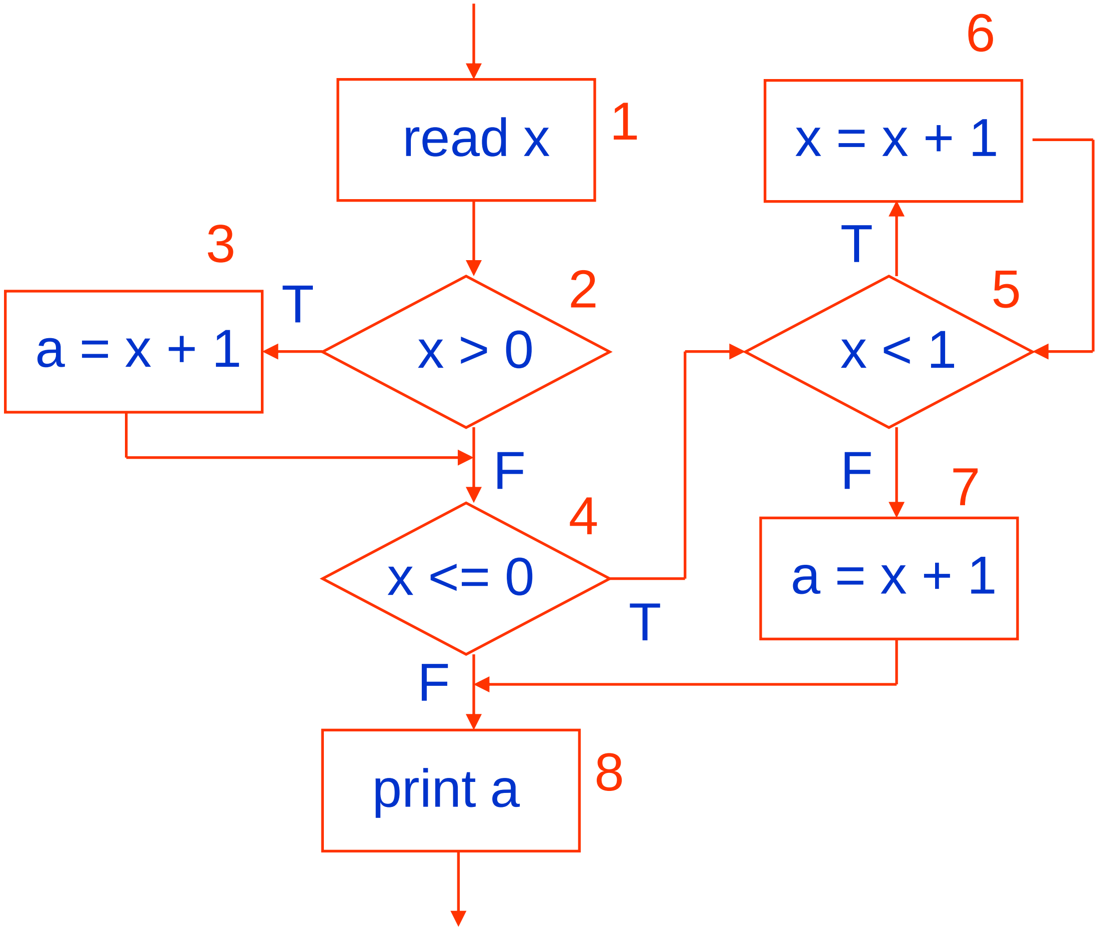
Slices

- A slice is a subset of a program.
- When testing a program, most of the code in the program is irrelevant to what you are interested in. Slicing provides a convenient way of filtering out irrelevant code.
- Slices can be computed automatically by statically analyzing the control flow and data flow of the program.

Slices

- A slice with respect to a variable v at a certain point p in the program is the set of statements that contributes to the value of the variable v at p.
- We use S(v, n) to denote the set of nodes in the control flow graph that contributes to the value of the variable v at node n.

An Example



S(x, 1) = {1}

S(x, 2) = {1}

S(x, 3) = {1, 2}

S(x, 4) = {1, 2}

S(x, 5) = {1, 2, 4}

S(x, 6) = {1, 2, 4, 5, 6}

S(x, 7) = {1, 2, 4, 5, 6}

S(a, 3) = {1, 2, 3}

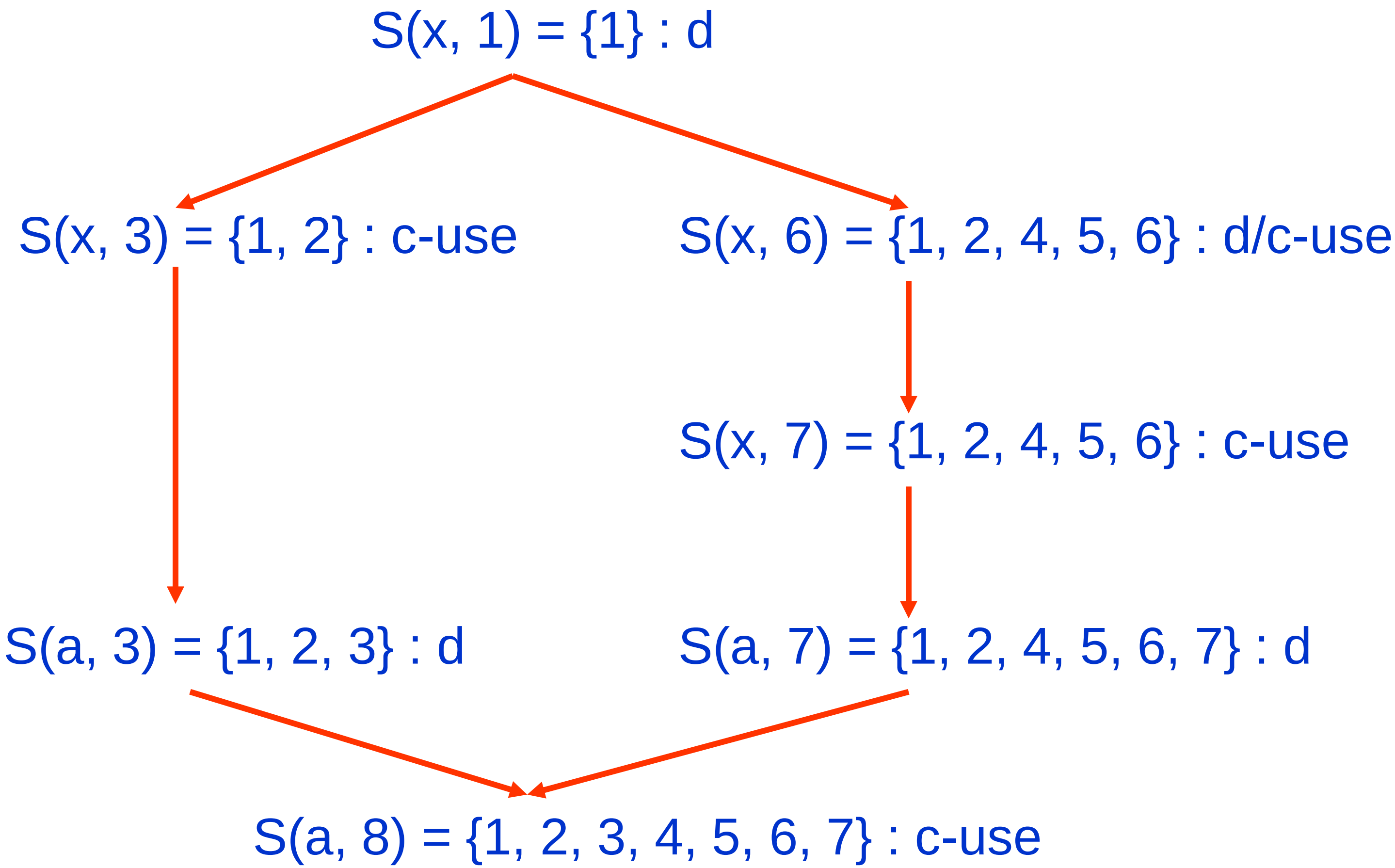
S(a, 7) = {1, 2, 4, 5, 6, 7}

S(a, 8) = {1, 2, 3, 4, 5, 6, 7}

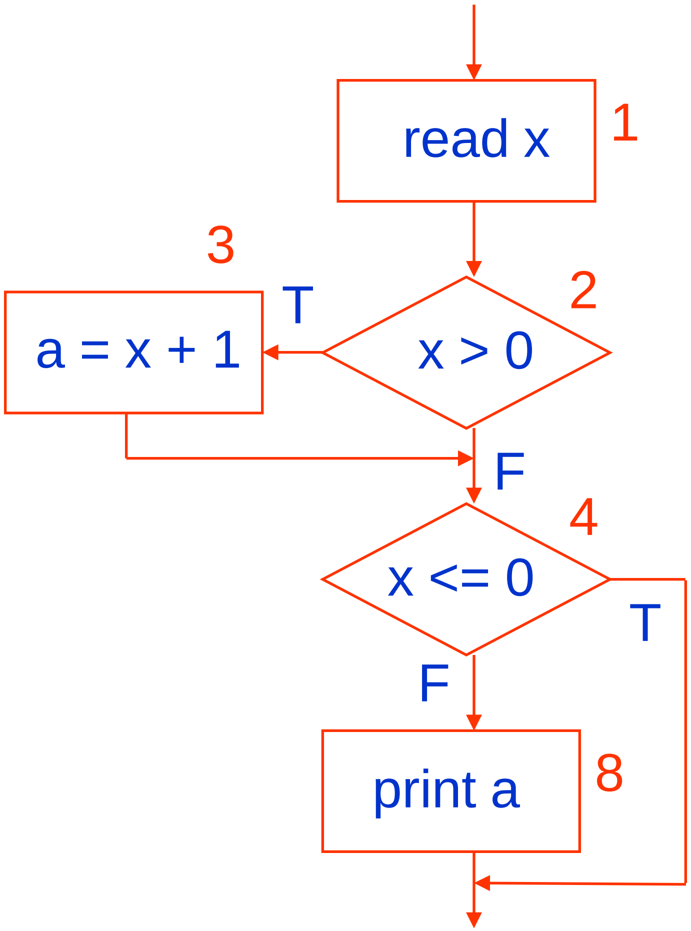
Lattices of Slices

- A definition of a variable v_n at node n usually uses the values of several variables $v_1, \ldots, v_m$.
- The slice $S(v_n, n)$ will contain the slices $S(v_1, n)$, …, $S(v_m, n)$.
- These subset relationships induce a lattice on slices of different variables.

An Example


S(x, 1) = {1} : d
S(x, 3) = {1, 2} : c-use
S(x, 6) = {1, 2, 4, 5, 6} : d/c-use
S(x, 7) = {1, 2, 4, 5, 6} : c-use
S(a, 3) = {1, 2, 3} : d
S(a, 7) = {1, 2, 4, 5, 6, 7} : d
S(a, 8) = {1, 2, 3, 4, 5, 6, 7} : c-use

Test Case I



Test Case II