EE360T/382V Software Testing
khurshid@ece.utexas.edu

February 12, 2020

Overview

Today – Chapter 2: Continue with graph coverage for source code

Last class – Chapter 2: Graph coverage for source code

Next class – Continue with graph coverage

EE360T/382V Software Testing
khurshid@ece.utexas.edu

Chapter 2*: Graph Coverage

*Introduction to Software Testing by Ammann and Offutt

Data flow criteria

Focus: flow of data values

Definition (*def*) – location where value of a variable is stored in memory, e.g., assignment statement

Use – location where a variable's value is accessed

Let V be a set of variables w.r.t. the program modeled

For node *n:*

- $def(n) \subseteq V$ is set of variables defined at n
- $use(n) \subseteq V$ is set of variables used at n

For edge *e:*

- $def(e) \subseteq V$ is set of variables defined at e
- $use(e) \subseteq V$ is set of variables used at e

du-path

A definition of a variable may or may not *reach* a use

- No path from def to use
- Value may change by another def before reaching the use

A path from location n_i to n_j is def-clear w.r.t. variable v if for every node n_k (and edge e_k) on the path where $k != i$ and $k != j$, v is not in $def(n_k)$ or in $def(e_k)$

The def of v at l_i reaches the use at l_j if there is a def-clear path from l_i to l_j

A du-path w.r.t. variable v is a simple path that is def-clear w.r.t. v from node n_i s.t. $v \in def(n_i)$ to node n_j s.t. $v \in use(n_j)$

Grouping du-paths

Def-path set $du(n_i, v)$ – set of du-paths w.r.t. variable v, which start at node n_i

Def-pair set $du(n_i, n_j, v)$ – set of du-paths w.r.t. variable v, which start at node n_i and end at node n_j

$$du(n_i, v) = \bigcup_{n_j} du(n_i, n_j, v)$$

Data flow criteria

C2.9 All-defs coverage (ADC) – for each def-path set $S = du(n, v)$, TR contains at least one path d in S

C2.10 All-uses coverage (AUC) – for each def-pair set $S = du(n_i, n_j, v)$, TR contains at least one path d in S

C2.11 All-du-paths coverage (ADUPC) – for each def-pair set $S = du(n_i, n_j, v)$, TR contains every path d in S
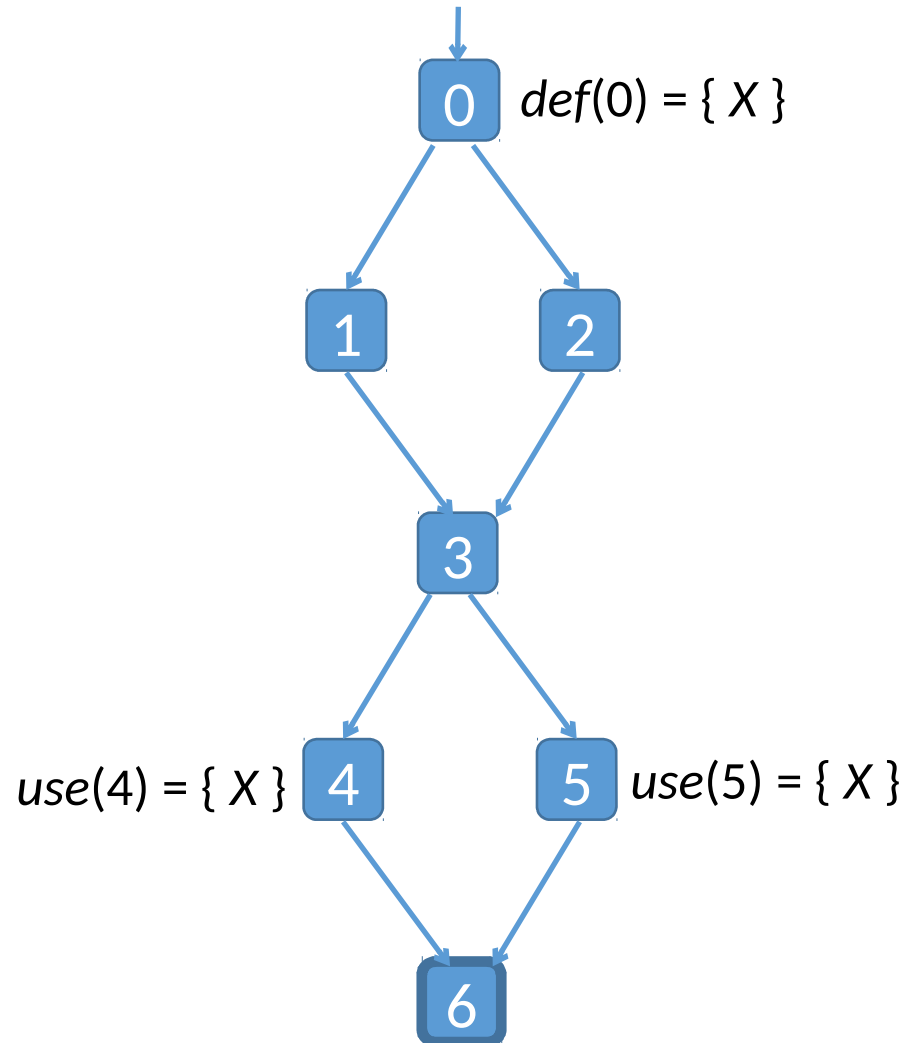
Example: data flow criteria

All-defs

- <0, 1, 3, 4>

All-uses

- <0, 1, 3, 4>
- <0, 1, 3, 5>

All-du-paths

- <0, 1, 3, 4>
- <0, 1, 3, 5>
- <0, 2, 3, 4>
- <0, 2, 3, 5>

$def(0) = \{ X \}$

$use(4) = \{ X \}$ $use(5) = \{ X \}$

Subsumption: graph coverage criteria

Complete Path Coverage
(*CPC*)

Prime Path Coverage
(*CPC*)

All-du-Paths Coverage
(*ADUPC*)

All-uses Coverage
(*AUC*)

All-defs Coverage
(*ADC*)

Edge-pair Coverage
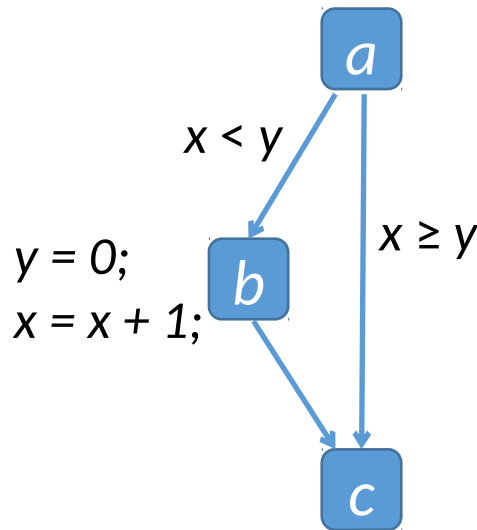(*EPC*)

Edge Coverage
(*EC*)

Node Coverage
(*NC*)
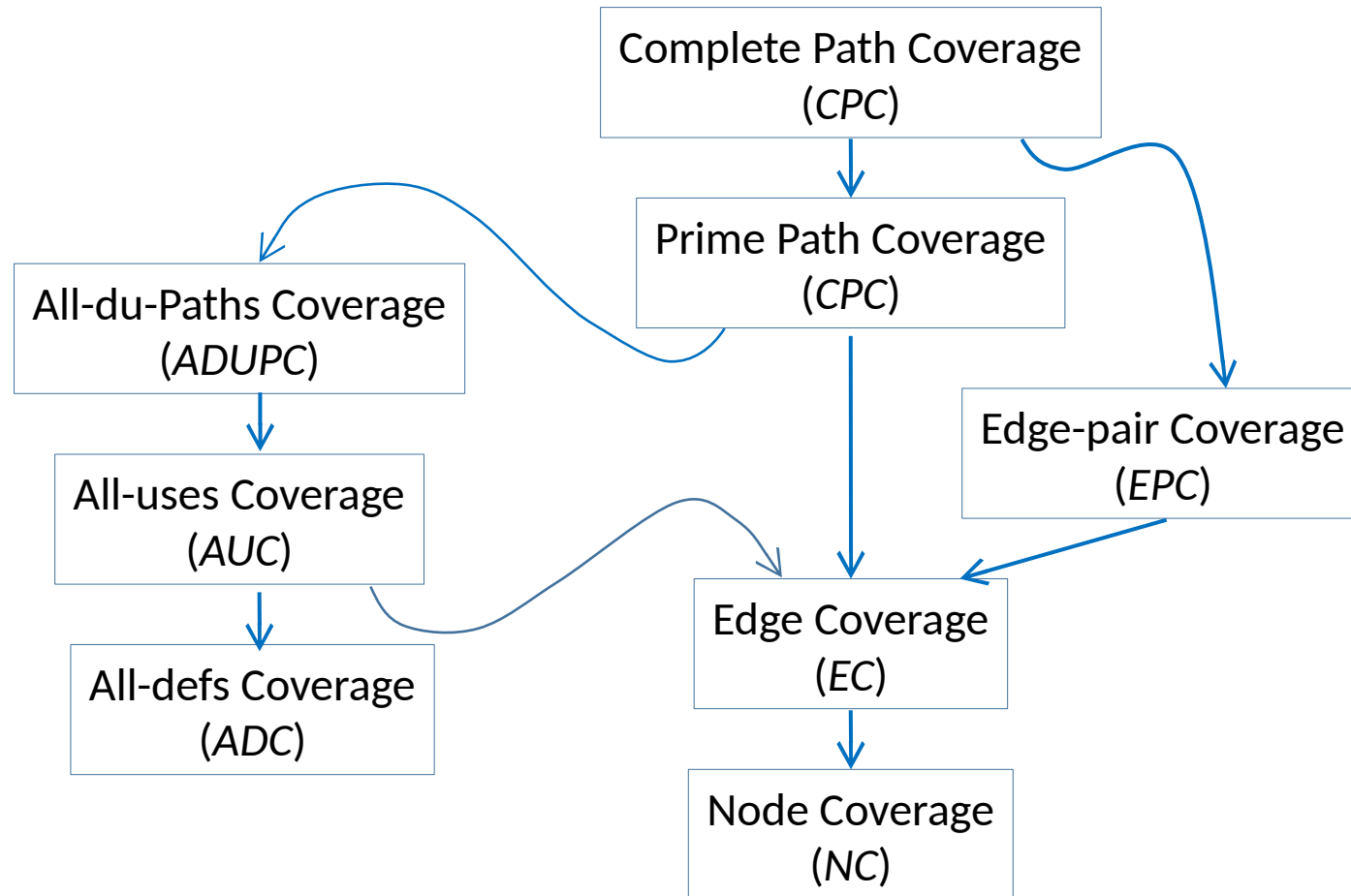
Building CFGs

Nodes are basic blocks (statement sequence such that if the first statement executes, all execute)
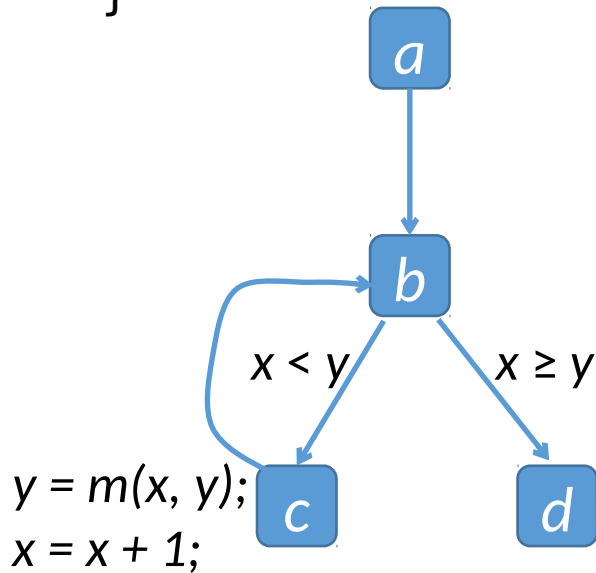
Edges are (conditional) branches

Example *if* statement (with no else block):

```
if (x < y) {
    y = 0;
    x = x + 1;
}
```
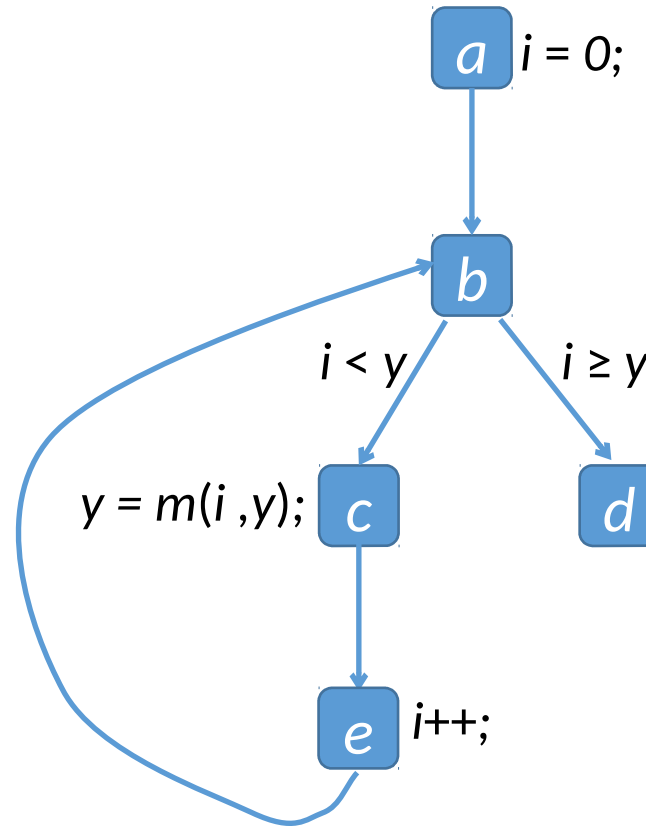

a

x < y

x ≥ y

y = 0;
x = x + 1;

b

c

Example *while* and *for* loops

x = 0;

while (x < y) {

    y = m(x, y);

    x = x + 1;

}

for (int i = 0; i < y; i++) {

    y = m(i, y);

}

Left diagram:

a → b

b → c (x < y)

b → d (x ≥ y)

c → b (y = m(x, y); x = x + 1;)

Right diagram:

a (i = 0;)

a → b

b → c (i < y)

b → d (i ≥ y)

c (y = m(i ,y);)

c → e (i++;)

e → b

def

A *def* occurs for variable *x* if for example:

- *x* appears on the lhs of an assignment
- *x* is a formal parameter
    - Implicit def when the method executes
- *x* is an input (e.g., from the console)

Simple when variable is of a primitive type

Can be complex for arrays and references

If a variable has multiple definitions in the same basic block, the last one matters in data flow analysis

use

A *use* occurs for variable *x* if for example:
- *x* appears on the rhs of an assignment
- *x* appears in a condition
- *x* is an actual parameter in method invocation
- *x* is an output (e.g., to console)
- *x* appears in the body of the return statement

If *def* and *use* for a variable appear on the same node *n*, (*n*, *n*) is a *du-pair* for *v* if *def* occurs after the *use* and the node is in a loop

Example from textbook authors' slides

?/!