UT ECE | THE UNIVERSITY OF TEXAS AT AUSTIN
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

# EE360T/382V Software Testing

khurshid@ece.utexas.edu

February 14, 2020

# Overview

Now – Chapter 2: Graph coverage for source code

Last time – Chapter 2: Graph coverage criteria

Next time – Continue with graph coverage

UT ECE THE UNIVERSITY OF TEXAS AT AUSTIN
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

# EE360T/382V Software Testing

khurshid@ece.utexas.edu

## Chapter 2[*]: Graph Coverage

[*] Introduction to Software Testing by Ammann and Offutt

# Touring, sidetrips, and detours

Sometimes satisfying a requirement exactly is hard

- Sidetrips and detours allow some flexibility in testing

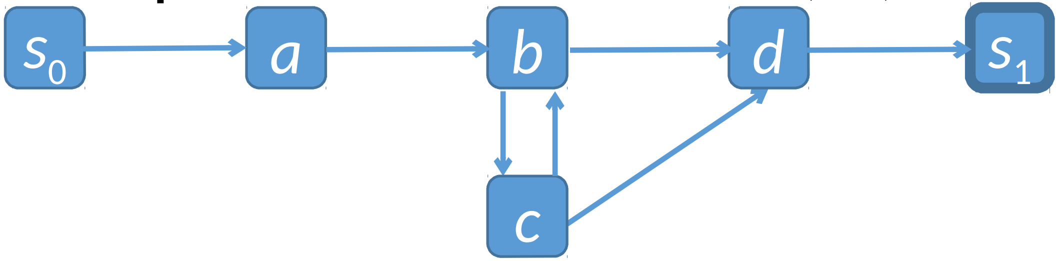
D2.36 Tour – test path *p tours* path *q* if *q* is a subpath of *p*

D2.37 Tour with sidetrips – test path *p tours* path *q* with *sidetrips* if every edge in *q* is also in *p in order*

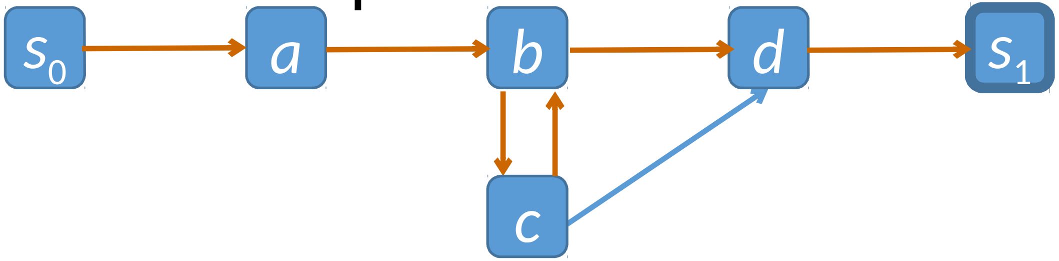
D2.38 Tour with detours – test path *p tours* path *q* with detours if every node in *q* is also in *p in order*

# Example: touring, sidetrips, detours

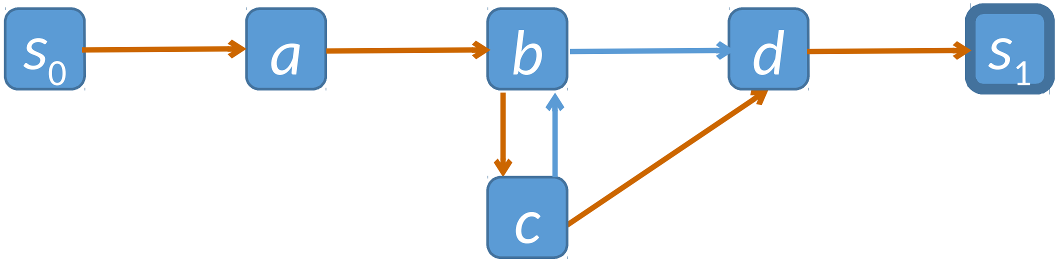
Assume a test path needs to tour <a, b, d>:



Touring with a sidetrip:



Touring with a detour:



# 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*) ⊆ *V* is set of variables defined at *n*
- *use*(n) ⊆ *V* is set of variables used at *n*

For edge *e*:

- *def*(*e*) ⊆ *V* is set of variables defined at *e*
- *use(e)* ⊆ *V* is set of variables used at *e*

# du-path

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

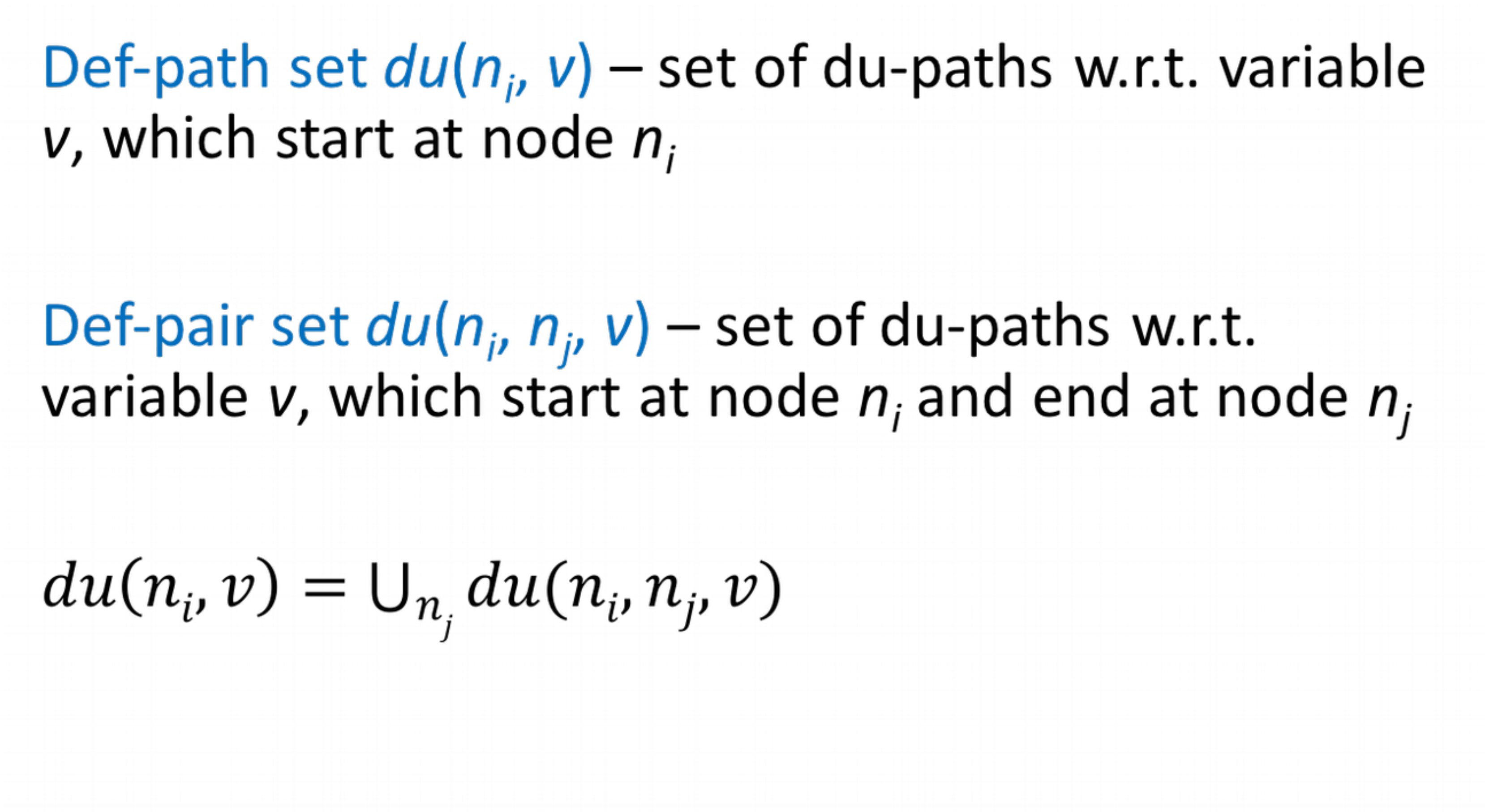
- No path from def to use
- Value may change by another def before reaching the use

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. *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 $n_j$

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

# Data flow criteria

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

C2.10 All-uses coverage (A*U*C) – 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 (AD*UP*C) – 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

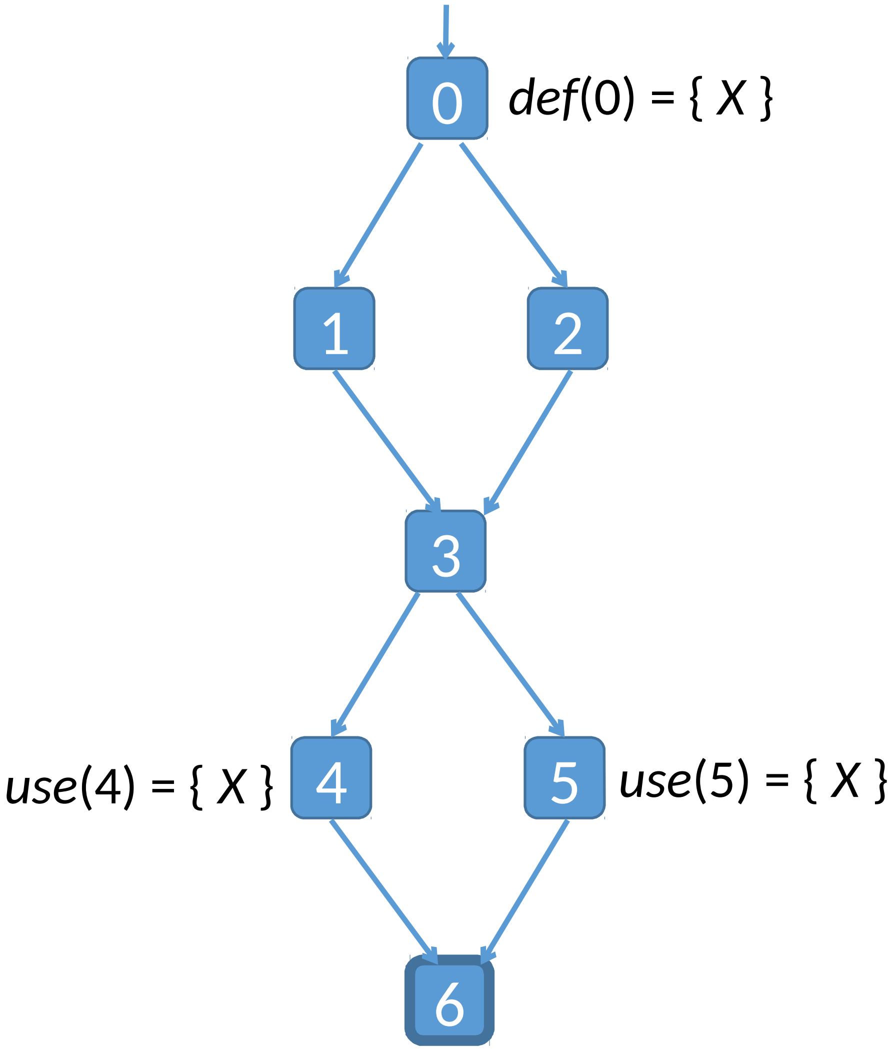
- <0, 1, 3, 4>

All-uses

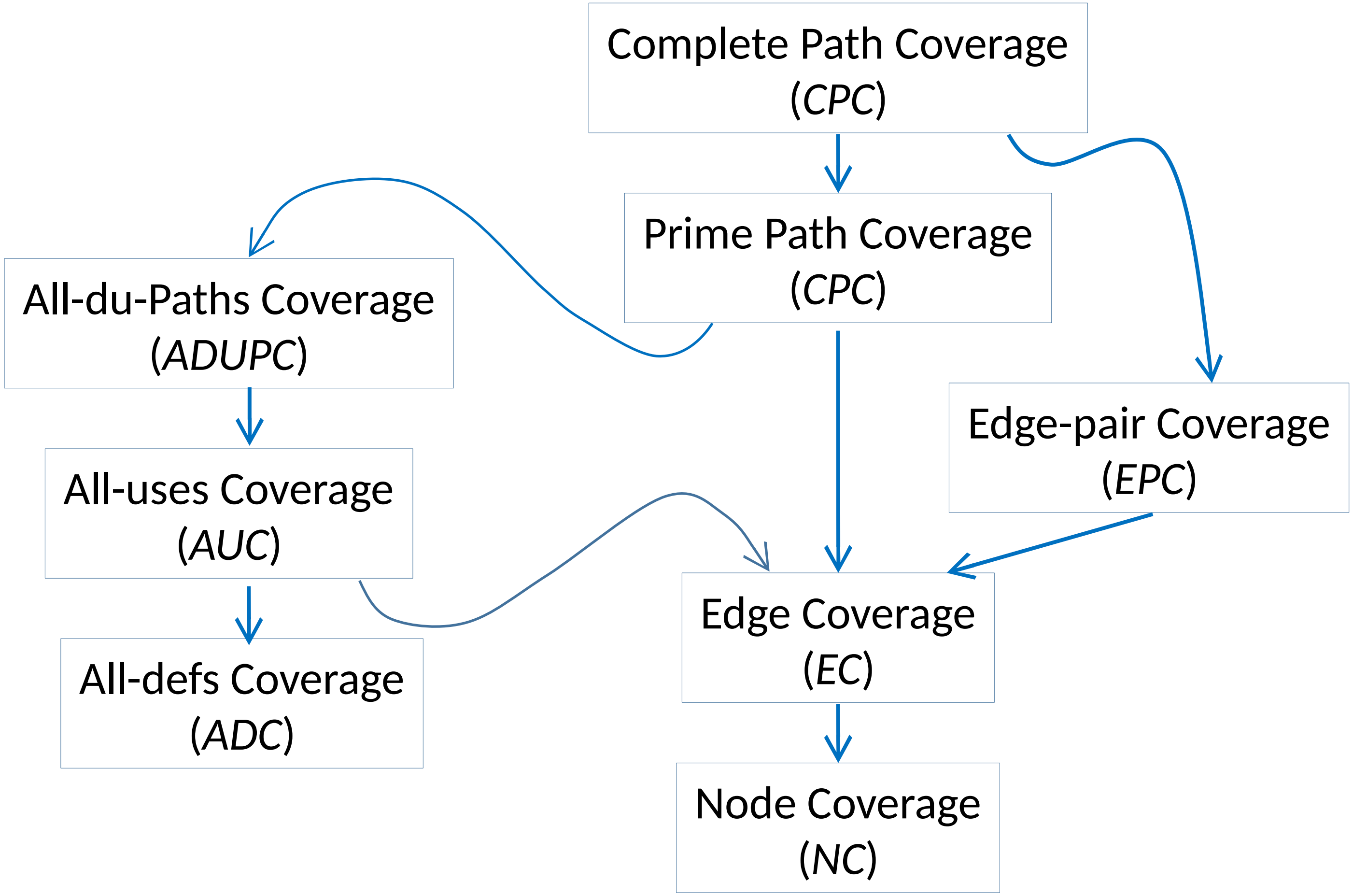
- <0, 1, 3, 4>
- <0, 1, 3, 5>

All-du-paths

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



# Subsumption: graph coverage criteria



Complete Path Coverage (*CPC*)

Prime Path Coverage (*CPC*)

All-du-Paths Coverage (*ADUPC*)

Edge-pair Coverage (*EPC*)

All-uses Coverage (*AUC*)

Edge Coverage (*EC*)

All-defs Coverage (*ADC*)

Node Coverage (*NC*)

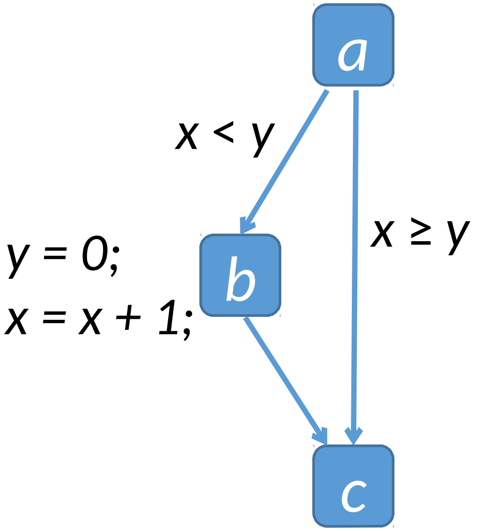
# Building CFGs

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

Edges are (conditional) branches

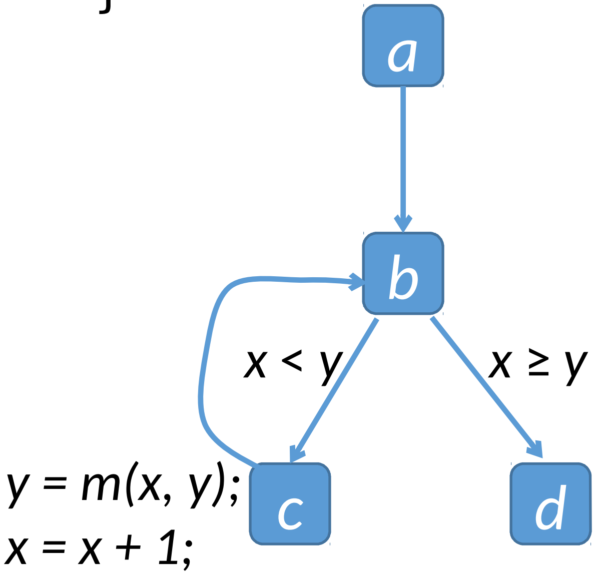
Example *if* statement (with no else block):

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

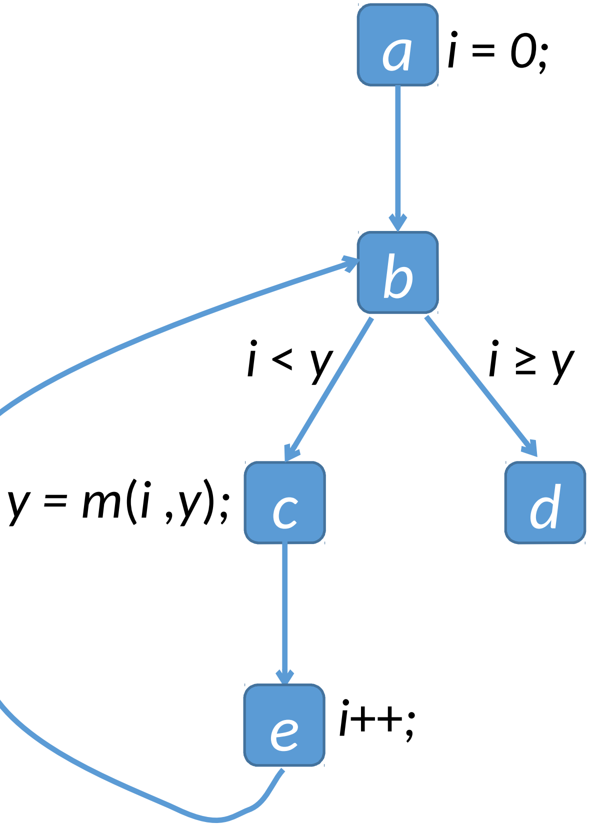


# Example *while* and *for* loops

```
x = 0;
while (x < y) {
      y = m(x, y);
      x = x + 1;
}
```



```
for (int i = 0; i < y; i++) {
      y = m(i, y);
}
```



# *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

# ?/!