EE360T/382V Software Testing

khurshid@ece.utexas.edu

February 10, 2020

Overview

Today – Chapter 2: Graph coverage criteria

Last class – Chapter 1: Introduction

Next class – Chapter 2: Graph coverage for source code

- Read: Section 2.3

Recall: Coverage criteria for testing

D1.20 Test requirement – specific element of the software artifact that a test case must cover (satisfy)

- Some requirements may be infeasible

D1.21 Coverage criterion – rule or collection of rules that impose test requirements on a test suite

- Criterion defines requirements precisely

D1.22 Coverage – given test requirements *TR* for coverage criterion *C*, test suite *T* satisfies *C* if and only if for each *tr* ϵ *TR*, at least one test in *T* satisfies *tr*

D1.23 Coverage level – given requirements *TR* and suite *T*, coverage level is

Recall: Subsumption

D1.24 Criteria subsumption – criterion C_1 subsumes criterion C_2 if and only if every test suite that satisfies C_1 also satisfies C_2

Note: must be true for every test suite

Example: if a test suite covers every branch, it covers every statement

Black-box or white-box?

D1.25 Black-box testing – deriving tests from external descriptions

- E.g., specs, designs, and requirements

D1.26 White-box testing – deriving tests from the source-code internals

- E.g., branches, conditions, and statements

Modern testing techniques often use both black-box and white-box approaches

Recall: Criteria based on structures

The textbook focuses on four kinds of structures to define criteria:

- Graphs (Chapter 2)
  - E.g., control-flow graphs (CFGs)
- Logical expressions (Chapter 3)
  - E.g., if-conditions
- Input domain characterization (Chapter 4)
  - E.g., sorted array
- Syntactic structures (Chapter 5)
  - E.g., mutation

EE360T/382V Software Testing

khurshid@ece.utexas.edu

Chapter 2*: Graph Coverage

*Introduction to Software Testing by Ammann and Offutt

Overview

Graphs form the basis for many coverage criteria

- Define test requirements using graph elements
- Given an artifact:
  - Create a graph abstraction, e.g., control-flow
  - Evaluate how test execution paths cover test requirements

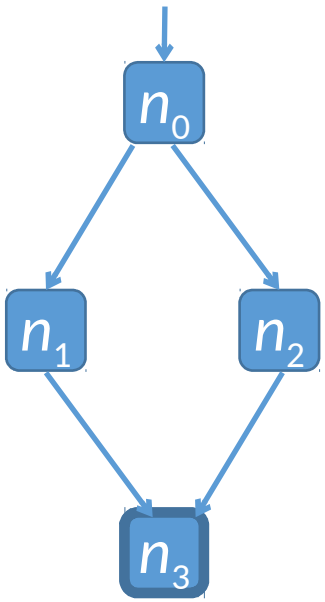
A graph $G = <N, E>$

- N is a set of nodes and E: $N \times N$ is a set of edges

Let $N_0 \subseteq N$ be a set of *initial* nodes

Let $N_f \subseteq N$ be a set of *final* nodes

Example graphs

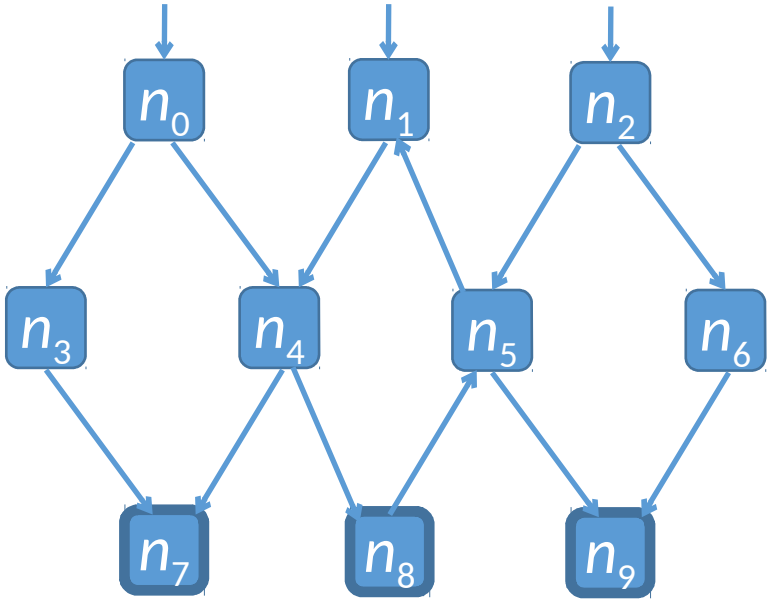


$N = \{ n_0, ..., n_3 \}$

$N_0 = \{ n_0 \}$

$E = \{ (n_0, n_1), (n_0, n_2), (n_1, n_3), (n_2, n_3) \}$

$|E| = 4$

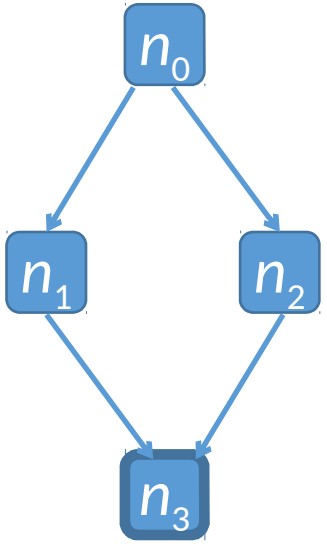


$N = \{ n_0, ..., n_9 \}$

$N_0 = \{ n_0 , n_1 , n_2 \}$

$E = \{ (n_0, n_3), (n_0, n_4), (n_1, n_4), (n_2, n_5), (n_2, n_6), ... \}$

$|E| = 12$



$N = \{ n_0, ..., n_3 \}$

$N_0 = \{\}$

$E = \{ (n_0, n_1), (n_0, n_2), (n_1, n_3), (n_2, n_3) \}$

$|E| = 4$

Path

A path $p = \langle n_1, n_2, \ldots, n_t \rangle$ in graph $G = \langle N, E \rangle$ is a sequence of nodes such that $(n_i, n_{i+1}) \in E$ for $1 \leq i < t$

A sub-path of a path p is a sub-sequence of p

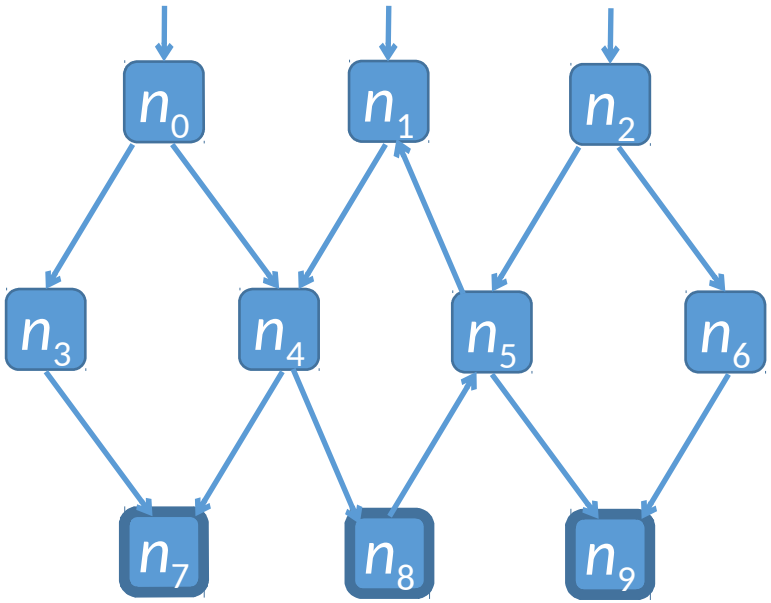
The length of a path is the number of edges it has

Example paths:

- $\langle n_0, n_3, n_7 \rangle$
- $\langle n_1, n_4, n_8, n_5, n_1 \rangle$
- $\langle n_4, n_8, n_5, n_9 \rangle$

Example non-path:

- $\langle n_0, n_7 \rangle$



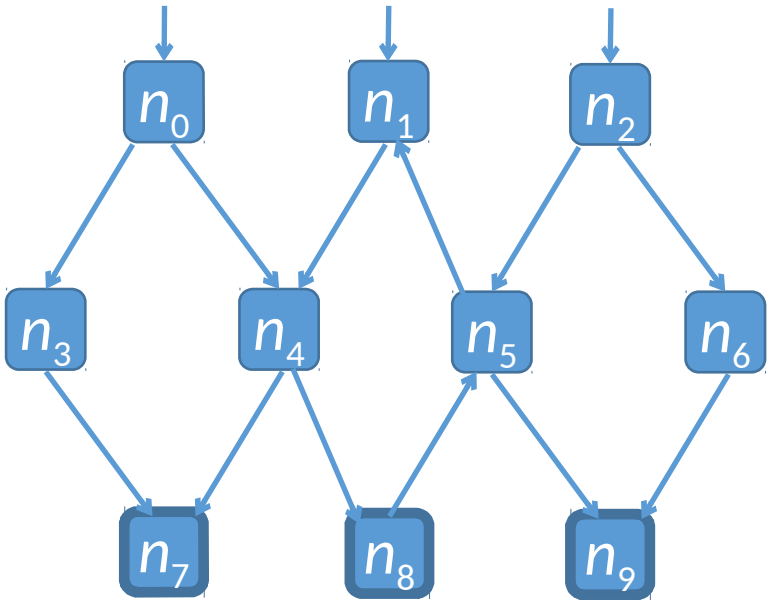
Syntactic reachability

A node *n* (or edge *e*) is syntactically reachable from from node *m* if there exists a path from *m* to *n* (or *e*)

Let $reach_G(x)$ be set of nodes in *G* reachable from *x*

Reachability examples:

- $reach(n_0) = N - \{ n_2, n_6 \}$
- $reach(\{ n_0, n_1, n_2 \}) = N$
- $reach((n_6, n_9)) = \{ n_6, n_9 \}$



Test path

D2.31 Test path – a path *p* (possibly of length 0) that starts at an initial node and ends at a final node

- In deterministic code, each test executes a unique path
- In non-deterministic code, each test may execute multiple paths

Test path *p* visits node *n* if *n* is on *p*

Test path *p* visits edge *e* if *e* is on *p*

For test *t*, write *path*(*t*) for test path executed by *t*

For test suite *T*, *path*(*T*) = { *path*(*t*) | *t* ∈ *T* }

Graph coverage criteria

Graph coverage criteria define requirements on properties of test paths in a graph, e.g., CFG

Two common forms of criteria

- Structural graph coverage criteria
- Data flow coverage criteria

A test requirement is met by *visiting* a particular node or edge or *touring* a particular path

D2.32 Graph coverage – Given test requirements *TR* for a graph criterion *C*, test suite *T* satisfies *C* if and only if $\forall tr \in TR \mid \exists p \in path(T) \mid p$ meets tr

Structural coverage criteria

Let G be the graph that models the software

D2.33 Node coverage – for each node $n \in reach(N_0)$, TR contains the predicate "visit n"

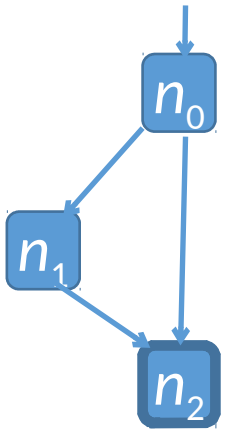
C2.1 Node coverage (*NC*) – *TR* contains each reachable node in *G*

C2.2 Edge coverage (*EC*) – *TR* contains each reachable path of length up to 1 (inclusive) in *G*

Example – Let tests t_1 and t_2 be such that:

- path(t_1) = <n_0, n_1, n_2>
- path(t_2) = <n_0, n_2>

Then { t_1 } gives *NC* and { t_1, t_2 } gives *EC*



EPC, PPC, and CPC

C2.2 Edge-pair coverage (*EPC*) – *TR* contains each reachable path of length up to 2 (inclusive) in *G*

A path *p* is simple if no node appears more than once on *p* except that first and last node may be the same

D2.35 Prime path – a prime path is a simple path that is not a proper subpath of another simple path

C2.3 Prime path coverage (*PPC*) – *TR* contains each prime path in *G*

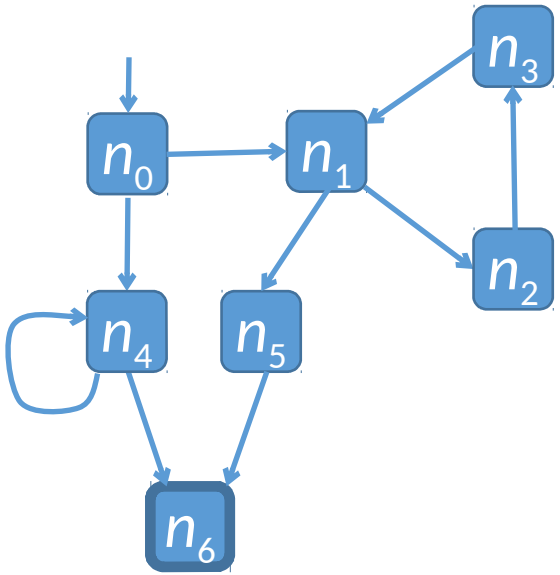
- Provides an elegant way to treat loops

C2.7 Complete path coverage (*CPC*) – *TR* contains all paths in *G*

Finding prime paths

Simple algorithm

- Start with paths of length 0, extend to length 1, …
- Collect all simple paths, and filter for prime



Length 0:
[0]
[1]
[2]
[3]
[4]
[5]
[6]!

Length 1:
[0,1]
[0,4]
[1,2]
[1,5]
[2,3]
[3,1]
[4,4]*
[4,6]!
[5,6]!

Length 2:
[0,1,2]
[0,1,5]
[0,4,6]!
[1,2,3]
[1,5,6]!
[2,3,1]
[3,1,2]
[3,1,5]

Length 3:
[0,1,2,3]
[0,1,5,6]!
[1,2,3,1]*
[2,3,1,2]*
[2,3,1,5]
[3,1,2,3]*
[3,1,5,6]!

Length 4:
[2,3,1,5,6]!

?/!