

Subsumption

Sometimes one coverage criterion is strictly more powerful than another one: any test set that satisfies C_1 might automatically satisfy C_2 .

Definition 1 *Criteria subsumption: coverage criterion C_1 subsumes C_2 iff every test set that satisfies C_1 also satisfies C_2 .*

Two ways to subsume:

1. test requirements for C_1 are always a superset of the test requirements for C_2 ; e.g. Calculus 2 test requirements are a superset of Calculus 1 test requirements.
2. many-to-one mapping exists between TRs for C_1 and C_2 ; i.e. for any test requirement tr that C_2 imposes, C_1 has at least one test requirement tr' which, when satisfied, also satisfies tr . (key point: tr' may be different from tr).

Software example: branch coverage (“Edge Coverage”) subsumes statement coverage (“Node Coverage”).

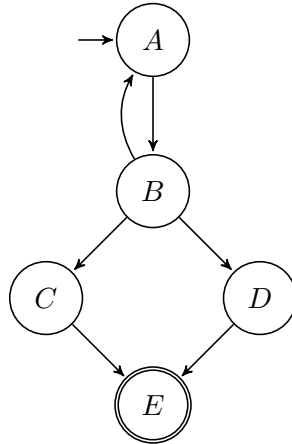
Evaluating coverage criteria. Subsumption is a rough guide for comparing criteria, but it’s hard to use in practice. Consider also:

1. difficulty of generating test requirements;
2. difficulty of generating tests;
3. how well tests reveal faults.

Graph Coverage

We will discuss graph coverage in great detail. Many forms of software testing reduce to graph coverage, so once you understand how graph coverage works, you will have a good understanding of a key software testing topic.

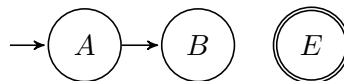
Definition of Graphs. (You should already be familiar with this material, but let's establish the terms we'll use in this class). Let's start with an example graph.



- N : Set of nodes $\{A, B, C, D, E\}$
- N_0 : Set of initial nodes $\{A\}$
- N_f : Set of final nodes $\{E\}$
- $E \subseteq N \times N$: Edges, e.g. (A, B) and (C, E) ;
 C is the predecessor and E is the successor in (C, E) .

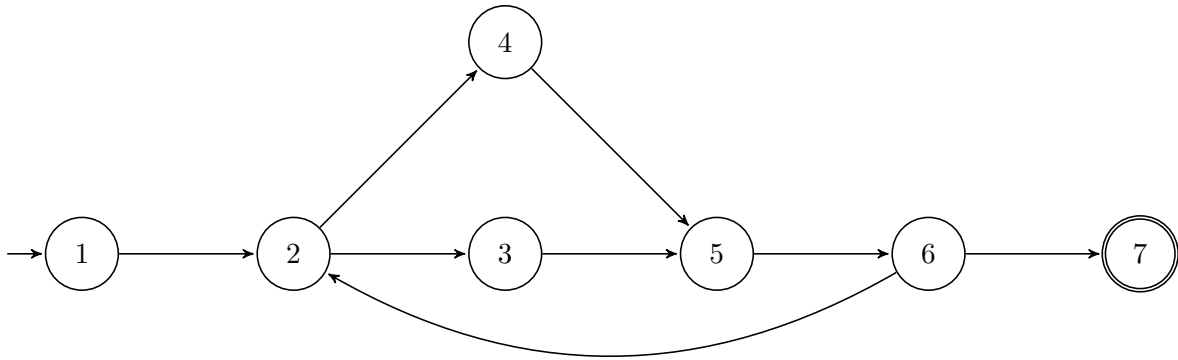
Subgraph: Let G' be a subgraph of G ; then the nodes of G' must be a subset N_{sub} of N . Then the initial nodes of G' are $N_0 \cap N_{\text{sub}}$ and its final nodes are $N_f \cap N_{\text{sub}}$. The edges of G' are $E \cap (N_{\text{sub}} \times N_{\text{sub}})$.

For example, consider the case where we set $N_{\text{sub}} = \{A, B, E\}$. This induces the subgraph:



Note that graphs need not be connected.

Paths. The most important thing about a graph, for testing purposes, is the path through the graph. Here is a graph $G_{\#}$ and some example paths through the graph.



- path 1: $[2, 3, 5]$, with length 2.
- path 2: $[1, 2, 3, 5, 6, 2]$, with length 5.
- not a path: $[1, 2, 5]$.

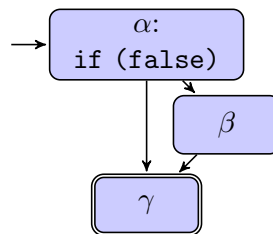
We say that path 1 is *from* node 2 *to* node 5. We can also say that path 1 is *from* edge $(2, 3)$ *to* $(3, 5)$.

Definition 2 A path is a sequence of nodes from a graph G whose adjacent pairs all belong to the set of edges E of G .

Note that length 0 paths are still paths.

Definition 3 A subpath is a subsequence of a path.

Paths and semantics. When a graph is a program’s control-flow graph, some of the paths in the graph may not correspond to program semantics. Consider the following graph.



Clearly β will never execute.

In this course, we will generally only talk about the *syntax* of a graph—its nodes and edges—and not its *semantics*.

However, in the following definition, we’ll talk about both notions.

Definition 4 A node n (or edge e) is syntactically reachable from n_i if there exists a path from n_i to n (or e). A node n (or edge e) is semantically reachable if one of the paths from n_i to n can be reached on some input.

We define $\text{reach}_G(\chi)$ as the portion of the graph reachable from χ . (χ might be a node, an edge, a set of nodes, or a set of edges.) For example:

- $\text{reach}_G(N_0)$ is the set of nodes and edges reachable from the initial node(s);
- $\text{reach}_{G\#}(2)$ in the above graph $G\#$ is $\{2, 3, 4, 5, 6, 7\}$;
- $\text{reach}_{G\#}(7)$ is $\{7\}$.

Note that we include χ in the set of nodes reachable from χ , because paths of length 0 are paths.

When we talk about the nodes or edges in a graph G in a coverage criterion, we'll generally mean $\text{reach}_G(N_0)$; the unreachable nodes tend to (1) be uninteresting; and (2) to frustrate coverage criteria.

Standard graph algorithms, like breadth-first search and depth-first search, can compute syntactic reachability. (Semantic reachability is undecidable; no algorithm can precisely compute semantic reachability for all programs.)

Test cases and test paths. Let's return to the context of test cases which we talked about on Wednesday. Here is a test path from $G\#$:

[1, 2, 3, 5, 6, 7];

here is another one:

[1, 2, 3, 5, 6, 2, 3, 5, 6, 7].

You can easily come up with more paths. Now, test paths are linked to test cases. First, let's define the notion of a test path.

Definition 5 A test path is a path p (possibly of length 0) that starts at some node in N_0 and ends at some node in N_f .

Running the test case on the program or method yields one or more test paths. A test path may represent many test cases (for instance, if a program takes the same branches on all of those test cases); or a test path may represent no test cases (if it is infeasible).