Software Testing, Quality Assurance and Maintenance Winter 2015

Lecture 6 — January 16, 2015

Patrick Lam *version 1*

Some binary distinctions

Let's digress for a bit and define some older terms which we won't use much in this course, but which we should discuss briefly.

- *Black-box testing.* Deriving tests from external descriptions of software: specifications, requirements, designs; anything but the code.
- *White-box testing.* Deriving tests from the source code, e.g. branches, conditions, statements.

Our model-based approach makes this distinction less important.

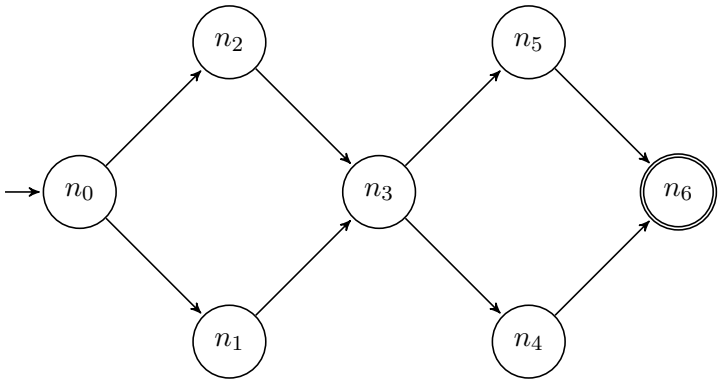
Test paths and cases

We resume our graph coverage content with the following definition:

Definition 1 *A graph is* single-entry/single-exit *(SESE) if N_0 and N_f have exactly one element each. N_f must be reachable from every node in N, and no node in $N \setminus N_f$ may be reachable from N_f, unless $N_0 = N_f$.*

The graphs that we'll be talking about in this course will almost always be SESE.

Here's another example of a graph, which happens to be SESE, and test paths in that graph. We'll call this graph D, for double-diamond, and it'll come up a few times.



Here are the four test paths in D:

$$[n_0, n_1, n_3, n_4, n_6]$$
$$[n_0, n_1, n_3, n_5, n_6]$$
$$[n_0, n_2, n_3, n_4, n_6]$$
$$[n_0, n_2, n_3, n_5, n_6]$$

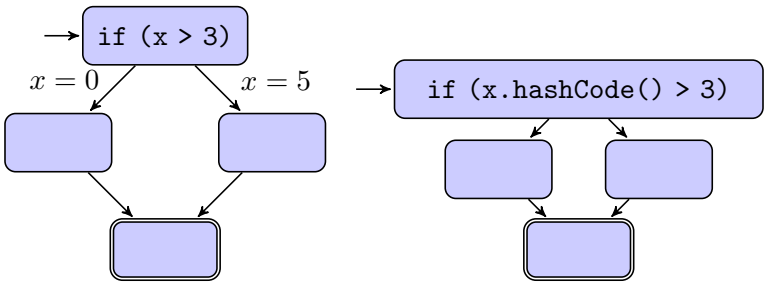
We next focus on the path $p = [n_0, n_1, n_3, n_4, n_6]$ and use it to explain several path-related definitions. We can say that p *visits* node n_3 and edge (n_0, n_1); we can write $n_3 \in p$ and $(n_0, n_1) \in p$ respectively.

Let $p' = [n_1, n_3, n_4]$. Then p' is a *subpath* of p.

Test cases and test paths. We connect test cases and test paths with a mapping path_G from test cases to test paths; e.g. $\text{path}_G(t)$ is the set of test paths corresponding to test case t.

- usually we just write path since G is obvious from the context.
- we can lift the definition of path to test sets T by defining $\text{path}(T) = \{\text{path}(t) | t \in T\}$.
- each test case gives at least one test path. If the software is deterministic, then each test case gives exactly one test path; otherwise, multiple test cases may arise from one test path.

Here's an example of deterministic and nondeterministic control-flow graphs:



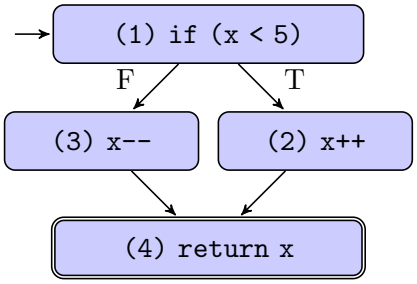
Causes of nondeterminism include dependence on inputs; on the thread scheduler; and on memory addresses, for instance as seen in calls to the default Java `hashCode()` implementation.

Nondeterminism makes it hard to check test case output, since more than one output might be a valid result of a single test input.

Indirection. Note that we will describe coverage criteria with respect to *test paths*, but we always run *test cases*.

Example. Here is a short method, the associated control-flow graph, and some test cases and test paths.

```
int foo(int x) {
  if (x < 5) {
    x ++;
  } else {
    x --;
  }
  return x;
}
```



- Test case: $x = 5$; test path: $[(1), (3), (4)]$.
- Test case: $x = 2$; test path: $[(1), (2), (4)]$.

Note that (1) we can deduce properties of the test case from the test path; and (2) in this example, since our method is deterministic, the test case determines the test path.

Graph Coverage

Having defined all of the graph notions we'll need for now, we apply them to graphs. Recall our previous definition of coverage:

Definition 2 *Given a set of test requirements* TR *for a coverage criterion C, a test set T* satisfies *C iff for every test requirement* tr *in* TR*, at least one t in T exists such that t satisfies* tr*.*

We apply this definition to graph coverage:

Definition 3 *Given a set of test requirements* TR *for a graph criterion C, a test set T satisfies C on graph G iff for every test requirement* tr *in* TR*, at least one test path p in path(T) exists such that p satisfies* tr*.*

We'll use this notion to define a number of standard testing coverage criteria. (At this point, the textbook defines predicates, but mostly ignores them afterwards. I'll just ignore them right away.)

Recall the double-diamond graph D which we saw on page 1. For the *node coverage* criterion, we get the following test requirements:

$$\{n_0, n_1, n_2, n_3, n_4, n_5, n_6\}$$

That is, any test set T which satisfies node coverage on D must include test cases t; the cases t give rise to test paths path(t), and some path must include each node from n_0 to n_6. (No single path must include all of these nodes; the requirement applies to the set of test paths.)

Let's formally define node coverage.

Definition 4 *Node coverage: For each node $n \in reach_G(N_0)$,* TR *contains a requirement to visit node n.*

We will state all of the coverage criteria in the following form:

Criterion 1 *Node Coverage (NC):* TR *contains each reachable node in G.*

We can then write

$$TR = \{n_0, n_1, n_2, n_3, n_4, n_5, n_6\}.$$

Let's consider an example of a test set which satisfies node coverage on D, the double-diamond graph from last time.

Start with a test case t_1; assume that executing t_1 gives the test path

$$\text{path}(t_1) = p_1 = [n_0, n_1, n_3, n_4, n_6].$$

Then test set $\{t_1\}$ does not give node coverage on D, because no test case covers node n_2 or n_5. If we can find a test case t_2 with test path

$$\text{path}(t_2) = p_2 = [n_0, n_2, n_3, n_5, n_6],$$

then the test set $T = \{t_1, t_2\}$ satisfies node coverage on D.

What is another test set which satisfies node coverage on D?

Here is a more verbose definition of node coverage.

Definition 5 *Test set T satisfies* node coverage *on graph G if and only if for every syntactically reachable node $n \in N$, there is some path p in path(T) such that p visits n.*

A second standard criterion is that of edge coverage.

Criterion 2 Edge Coverage *(EC). TR contains each reachable path of length up to 1, inclusive, in G.*

We describe edge coverage this way so that, as far as possible, new criteria in a series will subsume previous criteria.

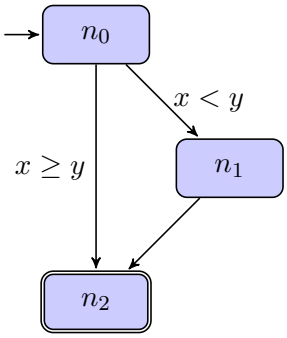
Here are some examples of paths of length ≤ 1:

Note that since we're not talking about *test paths*, these reachable paths need not start in N_0.

In general, paths of length ≤ 1 consist of nodes and edges. (Why not just say edges?)

Saying "edges" on the above graph would not be the same as saying "paths of length ≤ 1".

Here is a more involved example:



Let's define

$$\begin{aligned} \text{path}(t_1) &= [n_0, n_1, n_2] \\ \text{path}(t_2) &= [n_0, n_2] \end{aligned}$$

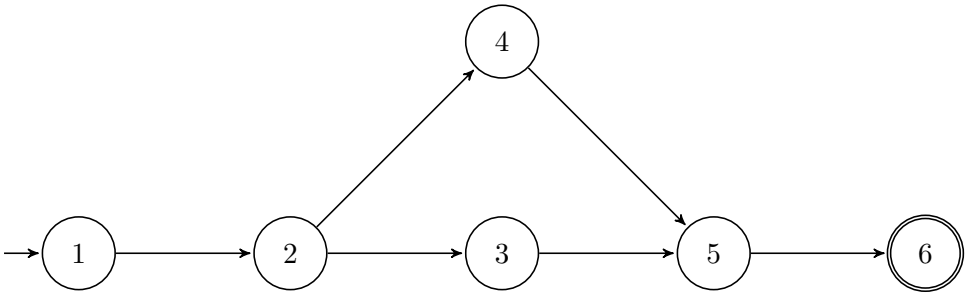
Then

$$\begin{aligned} T_1 &= \text{satisfies node coverage} \\ T_2 &= \text{satisfies edge coverage} \end{aligned}$$

Going beyond 1. So far we've seen length ≤ 0 (node coverage) and length ≤ 1. Of course, we can go to lengths ≤ 2, etc., but we quickly get diminishing returns. Here is the criterion for length ≤ 2.

Criterion 3 Edge-Pair Coverage. *(EPC) TR contains each reachable path of length up to 2, inclusive, in G.*

Here's an example.



- nodes:
- edges:
- paths of length 2: