Software Testing, Quality Assurance and Maintenance Winter 2015

Lecture 25 — March 9, 2015

Patrick Lam *version 1*

Logic Coverage

We now shift from graphs to logical expressions, for instance:

```
if (visited && x > y || foo(z))
```

Graphs are made up of nodes, connected by edges. Logical expressions, or predicates, are made up of clauses, connected by operators.

Motivation. Logic coverage is required by standard if you're writing avionics software. The idea is that software makes conditional decisions. While a condition evaluates to true or false, there are a number of ways that it might have done so, depending on its subparts. Logic coverage aims to make sure that test cases explore the subparts adequately.

The standard is called DO-178B. You can find a tutorial about logic coverage, and in particular the coverage called Modified Condition/Decision Coverage (MC/DC), here: `http://shemesh.larc.nasa.gov/fm/papers/Hayhurst-2001-tm210876-MCDC.pdf`.

Predicates. A *predicate* is an expression that evaluates to a logical value. Example:

$$a \wedge b \leftrightarrow c$$

Here are the operators we allow, in order of precedence (high to low):

- $\neg$: negation
- $\wedge$: and (not short-circuit)
- $\vee$: or (not short-circuit)
- $\rightarrow$: implication
- $\oplus$: exclusive or
- $\leftrightarrow$: equivalence

We do not allow quantifiers; they are harder to reason about. Note also that our operators are not quite the same as the ones in typical programming languages.

Clauses. Predicates without logical operators are *clauses*; clauses are, in some sense, "atomic". The following predicate contains three clauses:

```
(x > y) || foo(z) && bar
```

Logical Equivalence. Two predicates may be logically equivalent, e.g.

$$x \wedge y \vee z \equiv (x \vee z) \wedge (y \vee z),$$

and these predicates are not equivalent to $x \leftrightarrow z$. Equivalence is harder with short-circuit operators.

Sources of Predicates: source code, finite state machines, specifications.

Logic Expression Coverage Criteria

We'll use the following notation:

- P: a set of predicates;
- C: all clauses making up the predicates of P.

Let $p \in P$. Then we write C_p for the clauses of the predicate p, i.e.

$$C_p = \{c \mid c \in p\}; \qquad C = \bigcup_{p \in P} C_p$$

Given a set of predicates P, we might want to cover all of the predicates.

Criterion 1 Predicate Coverage *(PC). For each $p \in P$, TR contains two requirements: 1) p evaluates to true; and 2) p evaluates to false.*

PC is analogous to edge coverage on a CFG. (Let P be the predicates associated with branches.)

Example:

$$P = \{(x + 1 == y) \wedge z\}.$$

PC gives a very coarse-grained view of each predicate. We can break up predicates into clauses to get more details.

Criterion 2 Clause Coverage *(CC). For each $c \in C$, TR contains two requirements: 1) c evaluates to true; 2) c evaluates to false.*

Example:

$$(x + 1 == y) \wedge z$$

now needs:

Subsumption. Are there subsumption relationships between CC and PC?

The obvious exhaustive approach: try everything. (This obviously subsumes everything else).

Criterion 3 Combinatorial Coverage *(CoC). For each $p \in P$, TR has test requirements for the clauses in C_p to evaluate to each possible combination of truth values.*

This is also known as multiple condition coverage. Unfortunately, the number of test requirements, while finite, grows _____ and is hence unscalable.

A Heuristic: Active Clauses

So, in general, we can't evaluate every value of every clause, because there are too many of them. The next best thing is to focus on each clause and make sure it affects the predicate. That is, we'll test each clause while it is *active*.

Example. Consider the following clause:

$$p = x \wedge y \vee (z \wedge w)$$

Let's say that we focus on y; we'll call it the major clause. (We may designate any clause as a major clause at our whim.) That makes x, z, w minor clauses.

We want to make y *determine* p with certain minor clause values. That is:

- if we set y to true, then p evaluates to some value X;
- if we set y to false, then p must evaluate to $\neg X$.

The truth assignment:

$$x = \_ \quad z = \_ \quad w = \_$$

will make y determine p; in particular, y true makes p true and t false makes p false.

Definition 1 *Designate c_i as a major clause in predicate p. Then c_i* determines *p if the minor clauses $c_j \in p, j \neq i$, have values such that changing the truth value of c_i changes the truth value of p.*

We do *not* require that c_i has the same truth value as p. That requirement leads to trouble e.g. for the predicate:

Informally, determination tests each clause in a context where that clause has an effect.

Example.

$$p = a \vee b$$

Note that b does not determine p when:

That is, testing b has no effect; the test set { true, false } does not test a or b effectively.

Here is a variant of clause coverage which uses determination.

Definition 2 *Active Clause Coverage (ACC). For each $p \in P$ and making each clause $c_i \in C_p$ major, choose assignments for minor clauses $c_j, j \neq i$ such that c_i determines p. TR has two requirements for each c_i: c_i evaluates to true and c_i evaluates to false.*

This definition is somewhat ambiguous. We'll refine it soon.

Example. For $p = a \vee b$, make a major. We need b to be false for a to determine p. This leads to the TRs:

and similarly for b to determine p we need TRs:

Note the overlap between test requirements; it will always exist, meaning that our set of TRs for active clause coverage are:

In general, for a predicate with n clauses, we need $n + 1$ test requirements. (It might seem that we'd need $2n$ clauses, but we don't. Why?)