Software Testing, Quality Assurance and Maintenance Winter 2015

Lecture 26 — March 11, 2015

Patrick Lam *version 1*

Determination Example. Let's look at one of the examples from last time again.

$$p : (\neg a \oplus b) \rightarrow c$$

We construct a truth table as follows:

a	b	$\neg a \oplus b$	c	p
T	T	T	T	T
T	T	T	F	F
T	T	F	T	F
T	F	F	F	T
F	T	F	T	F
F	T	F	F	T
F	F	T	T	T
F	F	T	F	F

Note that we use certain patterns to see whether a clause determines p in the truth table. For c, we check adjacent rows in the truth table. For b, we check pairs of rows separated by one row; and for a we check one from the top half and one from the bottom half.

Symbolically Making a Clause Determine a Predicate

We've seen the brute-force way to figure out when a clause determines a predicate. It wouldn't scale, since truth tables grow exponentially. There are also symbolic ways of figuring out when a clause determines a predicate; here's one way that the textbook presents.

For predicate p and clause c,

- let $p_{c=\textsf{true}}$ represent p with c replaced by $\textsf{true}$;
- and $p_{c=\textsf{false}}$ represent p with c replaced by $\textsf{false}$.

Assume that c occurs exactly once in p, hence 0 times in $p_{c=\textsf{true}}$ and $p_{c=\textsf{false}}$.

Construct

$$p_c = p_{c=\textsf{true}} \oplus p_{c=\textsf{false}}.$$

Claim 1 *p_c describes conditions under which c determine p. That is, if p_c evaluates to* $\textsf{true}$*, then c determines p; if p_c evaluates to* $\textsf{false}$*, then c does not determine p.*

Let's look at some examples.

1. $p = a \vee b$:

$$\begin{aligned} p_a = p_{a=\mathsf{true}} \oplus p_{a=\mathsf{false}} &= (\mathsf{true} \vee b) \oplus (\mathsf{false} \vee b) \\ &= \mathsf{true} \oplus b \\ &= \neg b \end{aligned}$$

   We see that a determines p precisely when b is false. Symmetrically, b determines p when a is false.

2. $p = a \wedge b$:

$$\begin{aligned} p_a = p_{a=\mathsf{true}} \oplus p_{a=\mathsf{false}} &= (\mathsf{true} \wedge b) \oplus (\mathsf{false} \wedge b) \\ &= b \oplus \mathsf{false} \\ &= b \end{aligned}$$

   We need b to be true for a to determine p; symmetrically for a.

3. $p = a \leftrightarrow b$:

$$\begin{aligned} p_a = p_{a=\mathsf{true}} \oplus p_{a=\mathsf{false}} &= (\mathsf{true} \leftrightarrow b) \oplus (\mathsf{false} \leftrightarrow b) \\ &= b \oplus \neg b \\ &= \mathsf{true} \end{aligned}$$

   In this case, a always determines p regardless of b.

4. $p = a \wedge (b \vee c)$:

$$\begin{aligned} p_b &= p_{b=\mathsf{true}} \oplus p_{b=\mathsf{false}} \\ &= (a \wedge (\mathsf{true} \vee c)) \oplus (a \wedge (\mathsf{false} \vee c)) \\ &= (a \wedge \mathsf{true}) \oplus (a \wedge c) = a \oplus (a \wedge c) \\ &= a \wedge \neg c \end{aligned}$$

Why does this construction work? Intuitively:

- c determines p when setting c to true causes one value of p and setting c to false causes the other value.
- Everything else should be free to change.
- Exclusive-or is true when its inputs differ.
- $p_{c=\mathsf{true}}$ is basically setting c to true, $p_{c=\mathsf{false}}$ is setting c to false.

ACC is feasible for a clause c if there is a truth assignment to p which makes p_c true. If c can never determine p, then c is redundant with respect to p; that means that we could write p without using c, so something is probably wrong somewhere.

Active Clause Coverage Criteria

Minor Clauses. The ambiguity in our definition of active clause coverage stems from our looseness about the minor clauses. All we've said so far is that the major clause has to determine the predicate in each test requirement. Consider the following example:

$$p = a \wedge (b \vee c),$$

and let a be the major clause. Then does the following test suite,

$$\langle a : \mathsf{true}, b : \mathsf{false}, c : \mathsf{true}\rangle \qquad \langle a : \mathsf{false}, b : \mathsf{true}, c : \mathsf{true}\rangle$$

satisfy ACC? Only on a (but you need more cases for b and c).

We will briefly describe three flavours of ACC: General, Correlated and Restricted. The textbook goes into a lot of detail about these flavours, but the differences don't seem important in practice.

Criterion 1 General Active Clause Coverage *(GACC). For each $p \in P$ and letting each clause $c_i \in C_p$ be a major clause, choose minor clause values c_j such that c_i determines p. TR contains two test requirements for each c_i: c_i evaluates to true and c_i evaluates to false. Note that the c_js may be different when c_i evaluates to true and c_i evaluates to false—there are no restrictions on the c_j.*

The above test suite does not satisfy GACC; it does not meet the requirements for c.

Unfortunately, GACC does not subsume PC; it is more like CC (and obviously subsumes it).

Criterion 2 Correlated Active Clause Coverage *(CACC). For each $p \in P$ and letting each clause c_i be major, choose minor clause values for c_j such that c_i determines p. TR contains two requirements for each c_i: c_i evaluates to true and c_i evaluates to false. Furthermore, the values chosen for c_j must cause p to be true for one value of c_i and false for the other.*

To satisfy CACC, you have to satisfy GACC, plus, for each clause, you must make p true and false.

Last variation: Restricted Active Clause Coverage: ensure that c_js are always the same for each c_i.

Criterion 3 Restricted Active Clause Coverage *(RACC). For each $p \in P$ and letting each clause c_i be major, choose minor clause values for c_j such that c_i determines p. TR contains two requirements for each c_i: c_i evaluates to true and c_i evaluates to false. The values for each c_j must be the same when c_i is true and when c_i is false.*

Key point: must have the same minor clause values in the test set for each major clause. It is therefore harder to satisfy RACC than CACC.

Example where GACC does not subsume PC.