Software Testing, Quality Assurance and Maintenance Winter 2015

Lecture 27 — March 13, 2015

Patrick Lam *version 0*

Last time, we finished with an example showing that GACC does not subsume PC.

Same example with CACC.

So, how can we satisfy CACC on this example from last time?

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

CACC versus RACC. There's no clear reason to prefer RACC over CACC. But we can construct situations where CACC is feasible and RACC is infeasible. These situations typically involve dependencies between clauses, making some combinations of clause variables impossible. See the book for an example.

RACC seems to come from the aviation community misinterpreting a loosely-defined criterion known as "MCDC". RACC then corresponds to "unique-cause MCDC", but the FAA now allows CACC test suites, under the name "masking MCDC".

Inactive Clause Coverage. The book also defines the notion of inactive clause coverage.

Exercise 1. Consider predicate

$$p = a \leftrightarrow b \wedge c.$$

(a) Identify all clauses in predicate p.

(b) Compute and simplify conditions under which each of the clauses determines p. Conditions must use only $\wedge$, $\vee$ and $\neg$.

(c) Write the complete truth table for all clauses. Label rows starting at 1. Row 1 is all-clauses-true. Include columns where each clause determines the predicate, and also a column for the predicate itself.

(d) Identify all pairs of rows from your table that satisfy GACC with respect to each clause.

(e) Identify all pairs of rows from your table that satisfy CACC with respect to each clause.

(f) Identify all pairs of rows from your table that satisfy RACC with respect to each clause.

Exercise 2. Same exercise, but this time $p = a \wedge b$.

Infeasibility and Subsumption

Of course, some test requirements are infeasible. It's easy to make RACC infeasible.

Workaround I. Satisfy feasible TRs, drop infeasible TRs.

Workaround II. If you can't satisfy a particular coverage criterion, use a looser criterion. For instance, if you can't satisfy RACC, settle for CACC.

Causes of infeasibility. From the definition, a logic coverage test requirement t is infeasible if no test case can satisfy test requirement t. Here's an example of a case where predicate coverage is infeasible:

```
void m(char[] c) {
  if (c.length < 0) {
    print ("infeasible");
 }
}
```

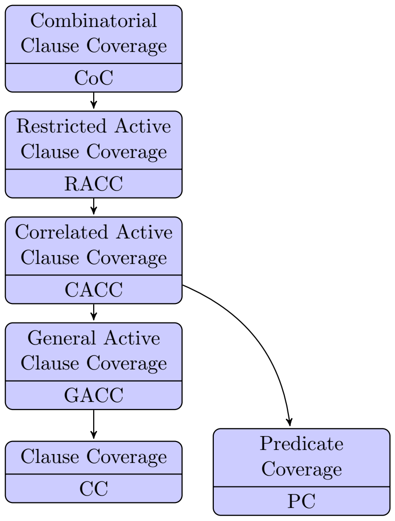
Clearly, no array c can have negative length, so the predicate can't be false.

I can think of the following ways for a coverage test requirement to be infeasible; these ways reflect the steps you need to follow to satisfy test requirements.

- The predicate is unreachable in its context (as in graph reachability); or
- The method containing the predicate never assigns appropriate values to its variables to cause the desired clause values (as in the example above); or
- A clause never determines a predicate and you are trying to achieve an active clause coverage criterion.

Subsumption Chart

Without inactive clause criteria, we can simplify the subsumption chart in the book to:


Combinatorial Clause Coverage
CoC
Restricted Active Clause Coverage
RACC
Correlated Active Clause Coverage
CACC
General Active Clause Coverage
GACC
Clause Coverage
CC
Predicate Coverage
PC

How to get Logic Coverage

To achieve logic coverage:

- identify the predicates $p \in P$ in the program fragment under test;
- figure out how to reach each of the predicates;
- make c determine p (for the active clause criteria); and
- find values for (program) variables to meet various criteria.

We've seen how to make c determine p; let's look at an example where we find values for variables to meet the criteria.

Predicates. We'll use the following predicate, modified from the textbook example `TestPat`:

```
isPat == false && iSub + pL - 1 < sL || subject[iSub] == pattern[0]
```

We can assign names to these clauses, giving the symbolic predicate:

Determination. We analyze the predicate and determine the conditions under which each clause determines the predicate:

Mapping Values to Predicates. Now we need to find program values that correspond to different predicate values. `isPat` is a boolean variable, so it's easy. The code says that `pL` and `sL` are lengths of argument arrays, so we need to assign arrays such that `iSub + pL - 1 < sL`. Finally, we can assign values to `iSub` and `subject` such that `subject[iSub]` is equal to or different from `pattern`, as needed. Examples:

Coverage Criteria. We now list the coverage criteria. For PC, we have:

CC:

CoC:

GACC:

(What are all of the ways to satisfy GACC?)

CACC: additional constraint that each pair must cause p to be both true and false.

RACC:

Concluding Comments on Logic Criteria. GACC doesn't imply predicate coverage, but often does when 3 or more terms are involved. RACC can be unsatisfiable. CACC seems to often be "just right".