Testing and Assurance of Software for Critical Systems

Krishna Sindhuja Kalusani

USC ID – T25568677

Systems in embedded software are to be tested with good test cases and the correctness of the test case result is particularly challenging. For this purpose there are procedures which makes the test cases reliable. Structural coverage criteria is one among them. This measures how well a code structure is exercised during the test execution. This includes statement coverage, branch coverage, condition coverage and decision/modified coverage (MC/DC) in the order of decreasing difficulty of test inputs. The MC/DC is used for critical avionics software in which every base condition in a decision should take on all possible outcomes atleast once. Each basic condition should be shown to independently affect the decision outcome. MC/DC test suite size is linear in the number of the basic conditions in a decision.

Masking problem: An intermediate variable introduced if OR becomes AND by mistake. Here the output may be same but the problem is masked out. The solution for this is that MC/DC requires that the effect of a condition propagation to its enclosing decision's outcome. Observable MC/DC was defined to require that the effect of a condition propagate to some program outcome.

Approximating observability using tags:

1. Assign each condition tag.
2. Track & propagate these tags through program execution.
3. If a tag reaches some output of the corresponding condition.

OMC/DC is more effective and robust. Counter example based test generation is one method in which a model that is doesn't exercise this condition and deriving test obligation from coverage criteria. The counter example produced in the test case meets the obligation. This can be infeasible especially when they are complex criteria. In incremental OMC/DC test generation, tag propagation can be captured naturally using dynamic symbolic execution. Tags are preferred to propagate to unvisited variables. The incremental approach gives good fault finding results. The abstraction may not reflect the exact results that were desired using the simulations. Steering the model would effectively handle the tolerances for getting flawless implementation. High true positives and low false negatives are desirable. Steering is able to account difference between expected and given behaviour for a good reason.