Generating MC/DC Adequate Test Sequences Through Model Checking[*]

Sanjai Rayadurgam
Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455
rsanjai@cs.umn.edu

Mats P.E. Heimdahl
Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455
heimdahl@cs.umn.edu

ABSTRACT

We present a method for automatically generating test sequences to satisfy MC/DC like structural coverage criteria of software behavioral models specified in state-based formalisms. The use of temporal logic for characterizing test criteria and the application of model-checking techniques for generating test sequences to those criteria have been of interest in software verification research for some time. Nevertheless, criteria for which constraints span more than one test sequence, such as the Modified Condition/Decision Coverage (MC/DC) mandated for critical avionics software, cannot be characterized in terms of a single temporal property.

This paper discusses a method for recasting two-sequence constraints in the original model as a single sequence constraint expressed in temporal logic on a slightly modified model. The test-sequence generated by a model-checker for the modified model can be easily separated into two different test-sequences for the original model, satisfying the given test criteria. The approach has been successful in generating MC/DC test sequences from a model of the mode-logic in a flight-guidance system.

1. INTRODUCTION

Software development for critical control systems, such as the software controlling aeronautics applications and medical devices, is a costly and time consuming process. When developing the most critical software, the validation and verification phase (V&V) consume approximately 50%–70% of the software development resources. Thus, if the process of deriving test cases for V&V could be automated and provide requirements-based and code-based test suites that satisfy the most stringent standards (such as DO-178B–the standard governing the development of flight-critical software for civil aviation [14]), dramatic time and cost savings would be realized.

[*]This work has been partially supported by NASA grant NAG-1-224 and NASA contract NCC-01-001.

If a software artifact can be represented as a finite state transition system, *model checking* techniques [6] can be used to generate test cases [12]. Model checkers are tools that explore the reachable state space of a model and report if properties of interest are violated in some state. If a violation is detected, the tool will report a sequence of inputs that brought the system to the violating state. With this capability we can, for example, assert that the "true" branch out of a decision point cannot be taken. If this branch in fact can be taken, the model checker will generate a sequence of inputs (with associated outputs) that forces the system to execute this branch—we have found a test case that covers this branch.

While the model-checking approach has been successful in generating test sequences to various structural criteria [13], coverage criteria similar to Modified Decision/Condition Coverage [5] (MC/DC) cannot be directly used in this approach. MC/DC coverage is designed to demonstrate the independent effect of atomic Boolean conditions on the Boolean expressions in which they occur. Thus, MC/DC test cases come in pairs, one where the atomic condition evaluates to false and one where it evaluates to true, but no other atomic conditions in the Boolean expression are changed. The model checking approach to test case generation is incapable of capturing such constraints over two test sequences. To work around this problem, we proposed to predetermine the Boolean vectors needed for an MC/DC pair and using those Boolean vectors in the specification of temporal properties to generate the test sequences [12]. Other researchers [1] have relaxed the MC/DC criterion to decouple the test sequences, thus making model-checking based test generation feasible.

In this paper we describe a novel alternative that leverages a model checker for complete and accurate MC/DC test case generation. We automatically rewrite the system model by introducing a small number of auxiliary variables to capture the constraints that span more than one test-sequence. We also introduce a special system reset transition to restore a system to its initial state. With these small modifications, a test constraint spanning two sequences in the original model can be expressed as a constraint on a single test-sequence in the modified model. Model-checking techniques can then be employed to generate this single test sequence which can be later factored into two separate test-sequences for the original model satisfying the actual test criteria. The modification to the system model can be performed by simple structural examination of the model without requiring any detailed analysis. The trade-off is the slightly increased state-space caused by the auxiliary variables. The benefit is that test cases can now be generated for the original test criteria using model-checking based approach, without the need for either an elaborate pre-processing analysis step or a relaxed test criterion. We have applied this approach to generate MC/DC test sequences from a realistic model of the flight guidance mode logic for business and regional jets.

2. BACKGROUND

Model checkers build a finite state transition system and exhaustively explore the reachable state space searching for violations of the properties under investigation [6]. Should a property violation be detected, the model checker will produce a counter-example illustrating how this violation can take place. In short, a counter-example is a sequence of inputs that will take the finite state model from its initial state to a state where the violation occurs.

A model checker can be used to find test cases by formulating a test criterion as a verification condition for the model checker. For example, we may want to test a transition (guarded with condition C) between states A and B in the formal model. We can formulate a condition describing a test case testing this transition—the sequence of inputs must take the model to state A; in state A, C must be true, and the next state must be B. This is a property expressible in the logics used in common model checkers, for example, the logic LTL. We can now challenge the model checker to find a way of getting to such a state by negating the property (saying that we assert that there is no such input sequence) and start verification. The model checker will now search for a counterexample demonstrating that this property is, in fact, satisfiable; such a counterexample constitutes a test case that will exercise the transition of interest. By repeating this process for each transition in the formal model, we use the model checker to automatically derive test sequences that will give us transition coverage of the model. This approach has been successfully used to generate test cases to various criteria and from specifications in a variety of formal languages [7, 4, 3, 12, 10].

Nevertheless, not all useful test criteria can be expressed in an appropriate temporal logic that is amenable to model-checking [10]. One such criterion applicable to source code is Modified Condition/Decision Coverage [5]. Offutt *et.al* [11] define an analogous (but not equivalent) criterion for state-transition systems called *full-predicate coverage.* Black *et.al* [1] discuss the difficulty in using a model-checking approach to generate test cases for full-predicate coverage and define a less strict criterion called *uncorrelated full-predicate coverage* using the notion of Boolean derivatives and demonstrate how test cases to satisfy this criterion can be obtained using a model-checker.

Here we present an alternative approach that does not require relaxing the coverage criterion. Rather, by modifying the system model in a simple and systematic way, we obtain test cases that are fully MC/DC (or full-predicate coverage) adequate. These modifications involve adding Boolean variables recording the truth values of predicates in a decision point.

3. MC/DC COVERAGE CRITERION

In [12] we formulated various coverage criteria in temporal logic and demonstrated how model-checking could be used to derive test sequences for those criteria.

One of the important structural coverage criteria that is used in the avionics software domain is the MC/DC criterion. It was developed to meet the need for extensive testing of complex boolean expressions in safety-critical applications [5]. MC/DC was developed as a practical and reasonable compromise between decision coverage and multiple condition coverage. It has been in use for several years in the commercial avionics industry. The important aspect of this criterion is the requirement that testing should demonstrate the independent effect of atomic boolean conditions on the boolean expressions in which they occur. In [12] we defined a similar test criterion for our transitions systems, called *clause-wise guard coverage* that captures this notion of *independent effect* of clauses on predicates.

A test suite is said to satisfy MC/DC (or claus-wise coverage) if executing the test cases in the test suite will guarantee that:

- every point of entry and exit in the model has been invoked at least once,
- every basic condition in a decision in the model has taken on all possible outcomes at least once, and
- each basic condition has been shown to independently affect the decision's outcome

where a basic condition is an atomic Boolean valued expression that cannot be broken into Boolean sub-expressions. A basic condition is shown to independently affect a decision's outcome by varying only that condition while holding all other conditions at that decision point fixed. Thus, a pair of test cases must exist for each basic condition in the test-suite to satisfy MC/DC. However, test case pairs for different basic conditions need not necessarily be disjoint. In fact, the size of MC/DC adequate test-suite can be as small as $N + 1$ for a decision point with N conditions.

4. MC/DC THROUGH MODEL CHECKING

As mentioned earlier, the basic approach to test case generation using model-checking is to recognize that a test case is simply a finite execution trace of a system. If one can characterize this execution trace as a temporal logic property to be verified, model-checking techniques can be used to produce a witness trace for the property. It has also been observed [1] that criteria such as MC/DC that have constraints involving more than one test case cannot be handled in this fashion. The crux of the problem is that each execution trace, i.e., each test case, is characterized by a separate temporal formula and therefore constraints involving two or more execution traces must necessarily span multiple temporal formulas, and, therefore, multiple runs of the model-checker. This means that one must employ some mechanism outside model-checking to ensure satisfaction of those constraints across multiple executions of the model-checker. In our earlier work, we assumed such a mediating mechanism that would resolve the MC/DC coverage constraints between the pair of test cases and provided specific Boolean valued vectors which were used to formulate two separate temporal properties. Nevertheless, such mediation involves boolean constraint satisfaction, an analysis as complex as model-checking itself. Further, it worked only under an assumption of *independence of atomic conditions*, which may not be satisfied in realistic systems, thus leading to temporal formulas that do not have corresponding execution traces in the system.

To overcome this problem, we here propose to augment the original model with auxiliary system variables and transitions in such a fashion that the pair of execution traces (test cases) in the original model can be seen as a single concatenated execution trace in the augmented model. This execution trace can be characterized by a single temporal formula and, therefore, model-checking can be used to produce the concatenated test case. The result can then be easily post-processed and the single execution trace can be split into a pair of execution traces of the original model—a pair of execution traces satisfying the MC/DC criterion.

4.1 Augmenting the System Model

Informally, our approach is based on the idea that to get MC/DC coverage of a specific decision point it would be enough to (1) find a test sequence from the initial state to the decision point of interest, (2) remember the truth values of the conditions in the decision point, and then (3) find a continuation of this test sequence that takes us *back* to the decision point, but this time one of the conditions has a different truth value and the outcome of the decision is different. To achieve this, we must augment our model to address two concerns:

Concatenation: The two test sequences in an MC/DC pair must somehow be concatenated into a single test sequence.

Propagation: Truth values of conditions and decision of interest must be propagated from one test sequence to the other in the pair, so that MC/DC constraint can be enforced.

Since reactive system models typically assume an implicit control loop at the outermost level we can typically find these cycles without modifying the model in any way. Nevertheless, this cannot be guaranteed. To address this issue, we propose to augment the model with a special *hard reset* transition that restores the system to its initial state. Now, two finite execution traces of the original model can be viewed as a single execution trace containing the two traces separated by a *hard reset*. Given this extension, we can achieve concatenation of two sequences by stating that we want to (1) reach a decision point, (2) see a hard reset, and then (3) reach the same decision point again. Figure 1 pictorially illustrates the idea.

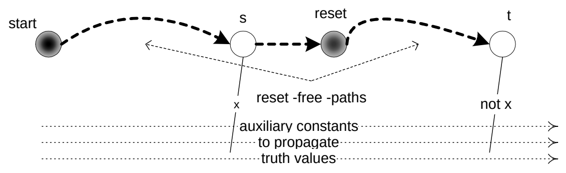

Figure 1: Concatenated test sequence.

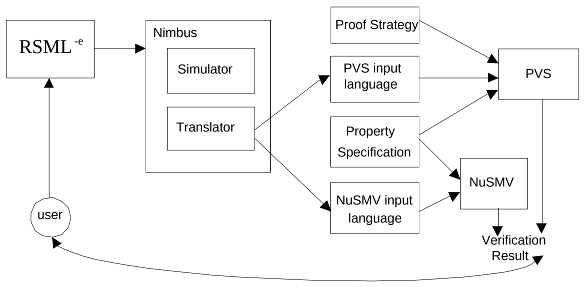

Figure 2: Verification Framework.

Once we can concatenate two test cases into one, what remains is to propagate information containing the values of the clauses and the guard predicate from the decision point in the first segment to the same decision point in the next segment, so that the MC/DC pair constraint can be enforced. The information to be passed is a Boolean vector containing the values of the clauses in the guard and so the number of additional variables required to augment the model is proportional to the number of clauses in the guard.

Assuming that there are n clauses in the guard in the decision point (transition) of interest, the values can be propagated through n auxiliary Boolean variables $u_1, ...u_n$ that have unknown but fixed values, i.e., their value in each execution trace is fixed but may vary from one trace to another. These variables may assume either Boolean value at the beginning of the execution of the augmented model. The next state relation for these variables can be simply specified as $u'_i = u_i$, i.e., the variables retain their initial values through the execution trace.

These variables serve the purpose of *propagating* the truth values of clauses from the decision point in the first segment to the same decision point in the next segment. What we must do is to make this connection in the temporal formula characterizing our concatenated test sequence. The values of the n clauses in the decision in first segment are asserted to be equivalent to the boolean vector $(u_1, ..., u_n)$.

5. CASE STUDY

To validate this approach to generating test cases for MC/DC coverage, we implemented it in our NIMBUS toolset for analyzing and executing RSML^{-e} specifications. We then applied our tool to a realistic flight guidance system provided by one of our industrial partners.

Figure 2 shows an overview of our tools framework. The user builds a behavioral model of the system in the fully formal and executable specification language RSML^{-e}. The specification is then fed to the NIMBUS simulator which checks that the specification is well formed and type correct, and allows the analyst a flexible execution environment for specification validation. After the specification is validated, the analyst can translate the specification to the PVS or NuSMV input languages for verification.

As part of this toolset we implemented a test-case generation framework that uses the NuSMV model-checker as the test-case generation engine. The tool automatically generates trap properties systematically from the specification for various structural coverage criteria, such as state, transition, condition, or, of particular interest for this report, MC/DC coverage. The model is then translated to the input language of NuSMV using our verification infrastructure discussed above, and the trap properties are used as a property specification to generate counterexamples which become the test set satisfying the given criteria. This process is fully automated and requires no manual intervention.

Our results are not in any way limited to the research langauge RSML^{-e} on which NIMBUS operates—our approach is applicable to a wide spectrum of languages, for example, SCR [9], Lustre [8], and Esterel [2].

5.1 Results

To evaluate the scalability of the approach, we generated suites of MC/DC trap properties for our collection of flight guidance models. In the case study, we used the NIMBUS tool to generate property specifications for the MC/DC test-criterion which were then provided to the NuSMV model-checker along with the translated RSML^{-e} specification. We used both the symbolic model checker and the bounded model checker provided in NuSMV. The results of generating test sequences based on the trap properties in Table **??** are included in Table 1. The symbolic model checker failed to complete the test case generation for FGS-05.

As evident from our case study, there are clear limitations to the capabilities of the symbolic model checker. The majority of the time was spent on counterexample generation—each property generates a counterexample and the time necessary to generate hundreds of counterexamples quickly became unacceptable.

The bounded model checker, on the other hand, scaled beyond our expectations. The main problem with a bounded model checker is that we have to put a limit on the search depth—we can only find counterexamples shorter than the maximum search depth we have selected. In verification tasks, this is a serious problem—if we fail to find a counterexample within our search depth we will not know if the property is true or if the counterexample is longer than the search depth allows. Nevertheless, in test case generation, this is not a serious issue. If we fail to find a test case within our predetermined search depth we are no worse off than we were before we had the automated tools—we would simply have to either (1) find the test case by hand or (2) determine that a test case for this MC/DC condition does not exist.

To summarize, our approach to MC/DC test case generation from formal specifications seems to scale well, at least when we use a bounded model checker. The ability to generate hundreds of complete test sequences in less than a minute is a revolutionary improvement over current state of the practice. Our experiences with bounded model checking in this domain have been generally positive and we see this as the ideal tool for this problem. Two characteristics of test case generation leads us to this conclusion; (1) the test cases (counterexamples) are typically short so most of them can be found within a relatively small search depth and (2) the consequences of not finding a counterexample are merely a nuisance, not catastrophic.

6. CONCLUSION

We have demonstrated how model-checking techniques can be used to generate truly MC/DC adequate test sequences. The method is amenable to complete automation and we have implemented the test generation approach in our NIMBUS tools. More importantly, it suggests a general approach for handling test criteria that impose constraints spanning multiple test-sequences. By augmenting the original model with auxiliary information it is possible reduce such constraints to single sequence constraints, which can be expressed in an appropriate temporal logic. The method is simple and practical as has been indicated in our case-studies and we expect this to scale well to even larger systems.

7. REFERENCES

[1] P. Ammann, P. E. Black, and W. Ding. Model checkers in software testing. Technical Report NIST-IR 6777, National Institute of Standards and Technology, 2002.

[2] G. Berry and G. Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.

[3] P. E. Black. Modeling and marshaling: Making tests from model checker counterexamples. In *Proc. of the 19th Digital Avionics Systems Confrence*, October 2000.

[4] J. Callahan, F. Schneider, and S. Easterbrook. Specification-based testing using model checking. In *Proceedings of the SPIN Workshop*, August 1996.

[5] J. J. Chilenski and S. P. Miller. Applicability of modified condition/decision coverage to software testing. *Software Engineering Journal*, pages 193–200, September 1994.

[6] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[7] A. Gargantini and C. Heitmeyer. Using model checking to generate tests from requirements specifications. *Software Engineering Notes*, 24(6):146–162, November 1999.

[8] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[9] C. Heitmeyer, A. Bull, C. Gasarch, and B. Labaw. SCR*: A toolset for specifying and analyzing

	FGS-00	FGS-01	FGS-02	FGS-03	FGS-04	FGS-05
NuSMV Symbolic	1.4	3.8	11.9	91.5	4760	NA
NuSMV Bounded	1.8	1.7	3.0	5.0	17.2	45.4

Table 1: Execution time to generate test sequences. All times are given in seconds.

requirements. In *Proceedings of the Tenth Annual Conference on Computer Assurance, COMPASS 95*, 1995.

[10] H. S. Hong, I. Lee, O. Sokolsky, and H. Ural. A temporal logic based theory of test coverage and generation. In *Proceedings of the International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS '02)*, Grenoble, France, April 2002.

[11] A. J. Offutt, Y. Xiong, and S. Liu. Criteria for generating specification-based tests. In *Proceedings of the Fifth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS '99)*, October 1999.

[12] S. Rayadurgam and M. P. Heimdahl. Coverage based test-case generation using model checkers. In *Proceedings of the 8th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2001)*, pages 83–91. IEEE Computer Society, April 2001.

[13] S. Rayadurgam and M. P. Heimdahl. Test-Sequence Generation from Formal Requirement Models. In *Proceedings of the 6th IEEE International Symposium on the High Assurance Systems Engineering (HASE 2001)*, Boca Raton, Florida, October 2001. To appear.

[14] RTCA. *Software Considerations In Airborne Systems and Equipment Certification.* RTCA, 1992.