AutoMOTGen: Automatic Model Oriented Test Generator for Embedded Control Systems*

Tool Paper

Ambar A. Gadkari, Anand Yeolekar, J. Suresh,
S. Ramesh, Swarup Mohalik, and K.C. Shashidhar

General Motors R&D - India Science Lab, Bangalore
{ambar.gadkari,anand.yeolekar,suresh.jeyaraman,
ramesh.s,swarup.mohalik,shashidhar.kc}@gm.com

1 Introduction

We present AutoMOTGen, a tool for automatic test case generation (ATG) from MATLAB Simulink/Stateflow (SL/SF) models [6] for testing automotive controllers. Our methodology is based on model checking [2]. The main highlights of the tool are:

1. Enhanced coverage of the model elements as well as high-level requirements.
2. A modular design for *plug-and-play* of different model checkers, test data generators and coverage analysis tools for enhancing the test suite quality.
3. Implements sampling time abstraction to generate tests with *lesser* number of (discrete) steps in the intermediate model.
4. Implements coverage dependent instrumentation of the model for the structural coverage criteria.
5. Capability to handle SL/SF blocks commonly used in automotive controllers (including blocks such as integrator, delay, multiplication/division, look-up tables, triggered subsystems and hierarchical and parallel charts).

The current implementation of AutoMOTGen uses SAL [8] as an intermediate representation and uses associated tools such as `sal-atg`, `sal-bmc` and `sal-smc` for generation of test data and proving the unreachability of some of the coverage goals. AutoMOTGen is implemented in Java and C++ (.NET framework) and uses MATLAB scripting language for extracting the relevant information from SL/SF models required for the purpose of test generation.

2 Motivation

Model checking, besides formal verification, has also been shown to provide an efficient technique to automatically derive test sequences from transition system models [1,4,5]. This approach for ATG relies on capabilities of the model

* The opinions expressed in this article are those of the authors, and do not necessarily reflect the opinions or positions of their employers, or other organizations.

A. Gupta and S. Malik (Eds.): CAV 2008, LNCS 5123, pp. 204–208, 2008.

checkers to generate traces for counter-examples of properties that do not hold in the model. Test suites are usually derived to satisfy certain coverage criteria of a model. The coverage criteria are mostly based on structural coverage of the transition system model such as state and transition coverage. The structural elements (states and transitions) are typically associated with Boolean variables called *trap* variables. Structural coverage of a model element is then reduced to model checking the reachability of the state where the associated trap variable is true. Model checking based ATG strives to find the most efficient test suite using directed search techniques. The main advantage of this approach is that one can achieve a systematic coverage of undischarged goals by using various model checking engines employing techniques such as explicit model checking, bounded model checking (based on SAT/SMT solvers), symbolic model checking and others in combination with various model slicing and reduction techniques for covering the deeper goals. Also, whenever certain goals cannot be covered the model checking engines can be invoked to prove the unreachability of those goals. Various other model abstraction techniques such as counter-example guided abstraction refinement, predicate-based abstraction and others can be explored to enhance the coverage, efficiency and scalability. Compared to other techniques such as random generation or guided-simulation used by most of the current commercial ATG tools [9,7] used in automotive controller development, the model checking based approach for ATG holds a greater promise in covering the deep-rooted coverage goals. This motivated the work on development of AutoMOTGen two years back. It serves as an experimental testbed for evaluating various technologies for test generation using industrial case studies. Recently, The Mathworks has introduced a toolbox, Simulink Design Verifier [6] which has ATG capability based on Prover's SAT-solver technology. We believe that our tool with its unique capability to provide an integrated environment for generation of test cases with plug-n-play of diverse tools and combining various techniques can help in addressing the needs of industrial scale designs. Results from our case studies have been encouraging in this regard.

3 Overview of Test Generation Flow

We describe here the generic flow of AutoMOTGen as shown in Figure 1. There are three inputs, namely, SL/SF models, high-level requirements and test specifications including different coverage goals. The output is a test suite of timed input-output sequences that can be used for testing the implementation. The test specification includes different coverage goals based on various structural criteria defined over SL/SF models. The test specification and high-level requirements are translated into formal properties using a subset of Linear Temporal Logic (LTL) by the property generator module. The SL/SF model is translated into a formal language which can be fed to the model checking engine. The model checking engine then verifies the formal model. The generated counter-example traces are converted into test cases consisting of timed input-output sequences.

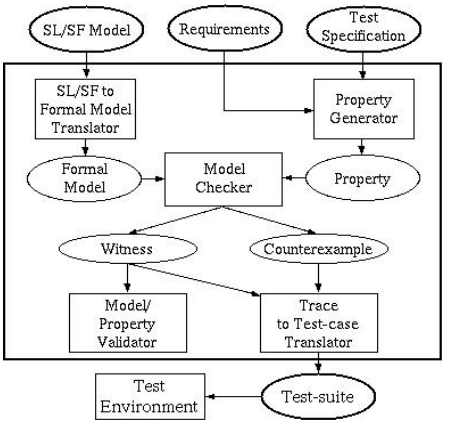

Fig. 1. AutoMOTGen architecture

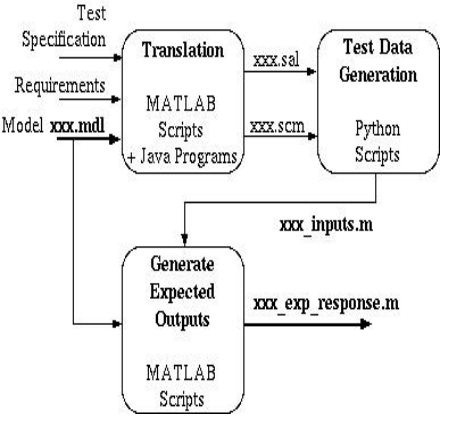

Fig. 2. AutoMOTGen back-end flow

4 AutoMOTGen Implementation

The current implementation of AutoMOTGen uses SAL as an intermediate language. This enables use of associated tools such as sal-atg, sal-bmc, sal-smc, etc. The back-end flow for model translation and test case generation is shown in Figure 2. The translator extracts the relevant information from SL/SF models through scripts built using MATLAB APIs. The SL/SF models are simulated using the generated test data and the corresponding outputs are stored as reference for testing the implementation. During simulation the model coverage information is obtained to assess the completeness of the generated test data. The translation of SL/SF to SAL is non-trivial and involves various steps such as time discretization, type abstractions and captures the simulation semantics of SL/SF. The SAL model is structured such that it retains the hierarchy information and allows the mapping of structural coverage of SL/SF model to the coverage goals. Additionally, monitors are inserted to cover the high-level requirements specified in the form of temporal logic properties. The continuous blocks in Simulink are approximated using linear interpolation. The step-size used for sampling is taken as a user input during the translation step. The user can modify the step-size depending on the coverage information. The trap variables are selectively introduced into the SAL model based upon the user selected coverage options. The use of model-checker such as sal-bmc requires that all the variables should be of type bounded integers. The model-checker sal-inf-bmc can be used in cases where real datatypes are present in the model, however, it puts restrictions on the arithmetic operations which result in nonlinear constraints. In these cases the real variables are approximated by use of look-up-tables for arithmetic operations. The tool provides a capability to easily modify the variable types and their ranges before selecting the appropriate model-checking engine. The uncovered goals are checked for proving unreachability and the results are

reported in the test generation logs. The unreachability of conditions or states in SAL model does not always imply unreachability in the SL/SF model. The tool provides a simple and intuitive GUI.

5 Case Studies

Our methodology has been evaluated using automotive controller case studies viz., Automatic Transmission Controller (ATC) and Adaptive Cruise Controller (ACC). The test results were compared with those obtained from a commercial tool that uses random test data generation techniques. The tests using our method were found to be more efficient in terms of providing model coverage with less number of input injections thus significantly reducing the test execution time, by almost factor of 10 in some cases. Results from these case studies based on a very preliminary implementation with a semi-automated flow are presented in [3]. In AutoMOTGen the entire end-to-end methodology has been fully automated. We have been able to handle various medium-sized controller models (corresponding RTW generated C code ranging between 2000-3000 lines) from real automotive subsystems. Recently, a larger case study has been initiated using controller modules from StabiliTrak^{TM} project (Electronic Stability Control system).

6 Conclusion

We have presented AutoMOTGen for automatic test case generation from SL/SF models of automotive controllers. It uses model checking for efficient generation of test data. It is designed to be modular to enable plug-and-play of different model checkers, test data generators and coverage analysis tools for obtaining efficient test suites. The tool has been evaluated using automotive controller examples and the comparative results with respect to other commercially available tools have been encouraging. Various enhancements are being carried out in the tool such as counter-example based abstraction refinement and other techniques to address scalability issues arising in larger industrial designs. HybridSAL is also being explored as one of the approaches for discretization.

References

1. Ammann, P., Black, P.E., Majurski, W.: Using model checking to generate tests from specifications. In: ICFEM, p. 46 (1998)
2. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge (2000)
3. Gadkari, A., Mohalik, S.K., Shashidhar, K.C., Yeolekar, A., Suresh, J., Ramesh, S.: Automatic generation of test-cases using model checking for SL/SF models. In: 4th International Workshop on Model Driven Engineering, Verification and Validation (MoDeVVa 2007) (2007)

4. Gargantini, A., Heitmeyer, C.L.: Using model checking to generate tests from requirements specifications. In: ESEC / SIGSOFT FSE, pp. 146–162 (1999)
5. Hamon, G., deMoura, L., Rushby, J.: Generating efficient test sets with a model checker. In: 2nd International Conference on Software Engineering and Formal Methods, Beijing, China, September 2004, pp. 261–270. IEEE Computer Society, Los Alamitos (2004)
6. The Mathworks, Inc., `http://www.mathworks.com`
7. Reactis, Reactive Systems, Inc., `http://www.reactive-systems.com`
8. SAL homepage, `http://sal.csl.sri.com/`
9. Safety Test Builder, TNI-Software, `http://www.tni-software.com`