SimAutoGen Tool: Test Vector Generation from Large Scale MATLAB/Simulink Models

Manel Tekaya, Mohamed Taha Bennani, Nedra Ebdelli, Samir Ben Ahmed

SimAutoGen tool: Test vector generation from large scale MATLAB/Simulink models

Manel TEKAYA[1], Mohamed Taha BENNANI[3], Nedra EBDELLI[2], and Samir BEN AHMED[3]

[1] University of Carthage, TELNET Innovation Labs, Tunisia
[2] University of Mannouba, Tunisia
[3] University of Tunis El Manar, Tunisia

Abstract. Safety-critical applications require complete high-coverage testing, which is not always guaranteed by model-based test generation techniques. Recently, automatic test generation by model checking has been reported to improve the efficiency of test suites over conventional test generation techniques. This study introduces our novel tool SimAutoGen, which employs the model checking technique (as a formal verification technique) to derive test vectors from Simulink models of automotive controllers according to structural coverage metrics. Model checking based on test generation is challenging for two reasons. First, the input model to the model checker requires conversion into a formal language. Second, standard tools have limited ability to generate test vectors for large-scale Simulink models because the state-space explodes with increasing model size. Our proposed SimAutoGen avoids the first problem by expressing the properties to be verified, which correspond to a structural coverage metric, in the Simulink language. To solve the state-space explosion problem, we developed a new algorithm that slices the Simulink model into hierarchical levels.

1 Motivation

Apart from providing formal verification, model checking efficiently and automatically derives test sequences from transition system models. Automatic test generation exploits the capabilities of model checkers, generating counterexamples with properties that violate the model [3]. As demonstrated by Gadhari et al. [4], the model checking technique generates test cases from models more efficiently than random generation and guided simulation. Motivated by this study, we began developing SimAutoGen three years ago. We limit our scope to Simulink models because Simulink is the most popular graphical modeling

This research and innovation work is conducted within a MOBIDOC thesis funded by the European Union under the PASRI project. This work is a collaboration between TELNET Innovation Labs and computer science and industrial systems laboratory.

language for embedded automotive software. Several model checking approaches for test case generation from MATLAB/Simulink models have been already proposed, including AutomotGen [4], SmartTestGen [9], and SAL (which integrate the sal–atg tool for automatic test generation) [10] and the V&V Diversity platform [8]. In [5], we compared the performances of SimAutoGen, sal–atg and the SLDV test case generator. Model checkers are recognized for their flexibility and ease of use [3]. However, we identified three main problems with model checkers:

1. Test case generation with model checkers is feasible only when the available model can be handled by the model checker.
2. Model checkers are severely limited by the state-space explosion problem.
3. The properties of model checkers are usually expressed in Linear Temporal Logic or Computational Tree Logic, which differ from the language of the model.

Our tool SimAutoGen corrects these problems in the context of test vector generation from Simulink models. First, SimAutoGen does not transform the Simulink model. Second, we implement a new slicing algorithm inspired by the method described in [7], which solves the state-space explosion problem in large-scale Simulink models. Third, the properties to be verified are expressed in the Simulink language, and specified according to the criterion of the structural coverage model.

2 Structural model coverage criteria

The structural coverage metric can be utilized in two ways, as a test adequacy criterion that decides whether a given test set completely or adequately complies with that criterion, or as an explicit specification for test vector selection. In the second case, the structural coverage metric behaves as a test selection criterion (a generator for white-box tests), because the model and the code generated from it are structurally similar. Thus, we can expect certain interrelations between the attained model and the code coverage. Kirner [11] discussed the preservation of code coverage at the model level. In our work, the structural coverage metrics are employed as the test selection criterion. The test vectors generated from the Simulink models by our model checking technique must conform to the structural coverage criterion. To accomplish this objective, we specify the Simulink properties for three criteria of the control flow coverage (Condition, Decision, and MC/DC), and the criterion of boundary value analysis. These four criteria are briefly described below.

1. **Condition coverage criterion**: This criterion is determined by ensuring the coverage of the Boolean inputs to the logical Simulink blocks.

2. **Branch/Decision coverage criterion**: According to this criterion, a block with conditional behavior is covered provided that all conditional behavior has been exercised at least once. For this purpose, SimAutoGen supports

the following blocks: Logical Operators, Switch, MultiportSwitch, Relational Operator, and Saturation.

3. **MC/DC coverage criterion**: Chilenski [13] investigated three categories of MC/DC: Unique Cause MC/DC, Unique Cause + Masking MC/DC, and Masking MC/DC. Based on [13], we employ masking MC/DC. In masking MC/DC, a basic condition is masked if varying its value cannot affect the outcome of a decision due to structure of the decision and the value of other conditions. Masking MC/DC for logical operator blocks is described in [14]. Besides the properties, each block needs an assumption to ensure generation of the required test vector. In SimAutoGen, the masking MC/DC coverage criterion is applied to the following blocks: Logical Operators, Switch, MultiportSwitch, Relational Operator, and Saturation.

4. **Boundary value analysis**: This criterion ensures data coverage of the numeric type inputs to the mathematical Simulink blocks (Sum, Product, Division, and Subtraction).

3 Software description

We present SimAutoGen, a tool that automatically generates test vectors from MATLAB/Simulink models [2]. Our methodology is based on model checking [6]. The main highlights of the tool, which is designed for automotive controller testing, are listed below:

1. Determines structural coverage metrics at the model level corresponding to the coverage metrics at the code level.
2. Generates test inputs by model checking, thus obtaining the model coverage criteria.
3. Does not convert the Simulink model to an intermediate formal language
4. Specifies the test objectives (properties) as Simulink properties.
5. Avoids the state-space explosion problem during model checking by enhancing an existing solution.
6. Improves the reliability of testing, thus reducing the test phase cost of large-scale Simulink models.

The current implementation of SimAutoGen uses the model checker Prover Plug-In [12] integrated into the Simulink Design Verifier tool (SLDV)[1]. SimAutoGen is implemented in Java (Eclipse Environment) and extracts the relevant information from the Simulink models by a MATLAB script. This information is then used for test generation.

4 Software Architecture

SimAutoGen is developed in the Eclipse and MATLAB environments. The portability of SimAutoGen is ensured by the Java script. A structural overview of SimAutoGen is presented in Figure 1.

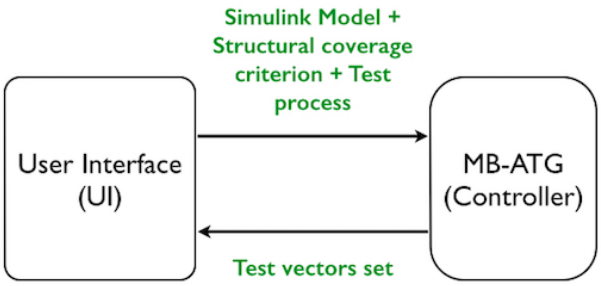


Fig. 1. SimAutoGen overview

User Interface : It is a Java Swing-based application that displays the inputs and outputs of SimAutoGen. The three inputs to SimAutoGen are (1) a Simulink model (a .mdl file), (2) a user-selected structural coverage criterion, and (3) a user-selected process. The three processes, Atomic testing, Unit testing, and Slicing, will be detailed in the appendix. The Atomic testing feature processes tiny Simulink models that require no slicing (i.e., single-output models). This feature is useful for a preliminary implementation testing. The Unit testing feature slices large Simulink models with two or more outputs, and is suitable for testing advanced implementations. The output of SimAutoGen is a set of test vectors or a set of slices. Slicing can be selected for purposes other than test vector generation.

Core elements : SimAutoGen is a new approach called MB–ATG [5], whose structure is described in Figure 2. MB–ATG is implemented in three steps. The first, second, and third steps handle large-scale Simulink models, automatic test vector generation from each slice (according to the structural coverage criterion), and integration of the test vectors generated from each slice, respectively. The second step uses the model checker Prover Plug-In and expresses the properties in the Simulink language. The property ψ and the assumption H as the model M are implemented with Simulink operators called Proof objective and Assumption, labeled P and A, respectively. Both operators are accessible through the SLDV library. In the third step, redundant test vectors are eliminated from the integration.

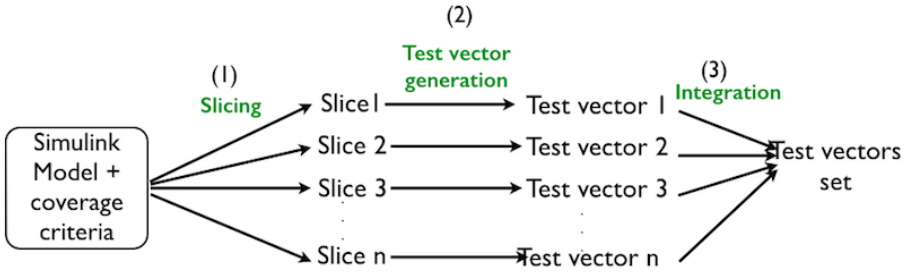
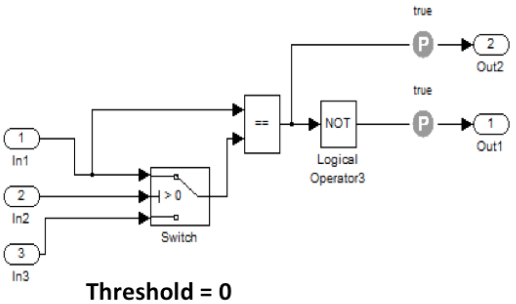


Fig. 2. Structure of MB–ATG

SimAutoGen implements two MB–ATG components: large-scale Simulink model slicing and test vector generation. Large-scale slicing is performed by a new slicing algorithm inspired by the static method described in [7], which constructs dependency graphs based on two dependence relations: Data Dependence and Control Dependence. The Simulink blocks Data-store/Data-read pairs and From/Goto pairs were not treated in the dependence analysis of [7] because they are not connected through explicit links; rather, they communicate remotely through implicit communication protocols (Data-store/Data-read pairs, for example). Our new algorithm models both types of links. The authors of [7] extracted the blocks corresponding to the specific slicing criterion. However, our objective is to slice the whole model into disjoint components (slices). To this end, we trialed two methods; forward slicing and backward slicing. The slicing criteria in forward slicing are the global inputs. This solution is problematic because most of the Simulink models contain Event input variables, which affect all blocks. Consequently, we adopted backward slicing, whose outputs are the slicing criteria. In particular, we compute the slices of the Simulink model by performing a backward reachability analysis and marking the relevant blocks for each output. We then remove the unmarked blocks and all empty subsystems from the model. A subsystem is a set of blocks that you replace with a single Subsystem block. The second MB–ATG component (test vector generation) has two elements: a model transformation protocol and test-vector integration. The model transformation protocol parses each slice and weaves the properties and assumptions according to the block type and the user-selected structural coverage criterion. Before the weaving of properties and assumptions, this protocol locates and calculates ψ and H insertion position. Next, it updates the location of the neighboring blocks. Finally, it weaves P and H over the Simulink model. The model transformation protocol is described in [5]. Figure 3 shows the coverage of the Switch block according to the model decision coverage, with the properties woven on it. The transformed slice is processed by the model checker Prover Plug-In. In this case, a counterexample (equivalent to a test vector) is generated. The test vectors generated and output from each slice are saved in an XL file. All of these test vectors are then integrated while eliminating the repet-

itive and useless elements in the saved XL file. For this purpose, we implement a new algorithm that compares different XL files.



Properties	Test input specification	Test input
Test objective 1 (P1)	In2>Threshold (out = In1)	(In1,In2,In3)(0,1,0)
Test objective 2 (P2)	In2 <= Threshold (out = In3)	(In1,In2,In3)(0,0,0)

Fig. 3. Decision coverage for the Switch block

5 Evaluation and measures

5.1 Model Description

Our tool was evaluated on six automotive industrial models, classified as shown in Table 1. The FastCor and Detection models are large-scale models with 400–800 blocks. AirFlow and AirMPmp have two outputs and between 44 and 75 blocks. ThrAr and AirMnfld are smaller models with 40 blocks and a single output.

Features \ Models	FastCor	Detection	AirFlow	AirMPmp	ThrAr	AirMnfld
Inputs	21	25	9	8	6	8
Blocks number	434	874	75	44	40	38
Implicit signal	12	17	0	0	0	0
Subsystems number	12	19	2	2	2	2

Table 1. Models description

5.2 Output description

Table 2 shows the slicing results of the four large-scale Simulink models described above. The two largest models, FastCor and Detection, are respectively

partitioned into three and five slices, whereas both medium-sized models are divided into two slices. The model splitting decreases the average number of inputs, blocks, and subsystems per slice, thereby avoiding the state-space explosion. The number of implicit connections represents the number of hidden links between the blocks of a single slice.

Models / Slices Features	Fastcor			Detection					AirFlow		AirMPmp	
Slices	S1	S2	S3	S1	S2	S3	S4	S5	S1	S2	S1	S2
Inputs number	11	20	11	8	13	13	6	13	7	6	8	8
Blocks	104	298	90	126	489	449	39	579	40	51	39	37
Implicit connections	3	5	4	3	4	4	2	4	0	0	0	0
Subsystems	7	10	10	7	9	9	2	9	2	2	2	2

Table 2. Slices description

Models / Measures	Fastcor	Detection	AirFlow	AirMPmp	ThrAr	AirMnfld
PST	1.247	4.079	1.073	1.015		
SST	12.892	14.983	11.383	11.003		
WT	3.354	10.14	2.467	2.279	10.661	9.543
IT	1.219	1.936	0.292	0.583		
GT	220.19	1295.009	17.952	20.514	22.134	
TV	5 13 5	8 9 10 4 8	2 2	1 1		
ITV	19	60	3	2	4	3

Table 3. Measures related to the execution time of SimAutoGen

Table 3 shows various measures related to the execution time in milliseconds of the large- and atomic-scale models. Here, WT, IT, and GT denote the execution time of weaving, integration, and generation of all slices, respectively. The variables TV and ITV denote the number of test vectors generated per slice and the number of integrated vectors in the entire model (after removing the redundant input values), respectively. For the slicing action, we determined the parallel slicing time (PST) and sequential slicing time (SST). A comparison of the execution times of the slicing algorithm using sequential and parallel methods shows the improvement because of the use of Parallel Computing Toolbox of MATLAB. Therefore, we have used this toolbox in weaving and test vector generation processes. GT presents the execution time of counterexample generation. It shows that the model checker prover Plug-In consumes a large part of the total execution time.

References

1. Simullink Design Verifier 1 : User' Guide. Mathworks, Inc (2012)
2. Getting Started Guide: R2014b. Mathworks, Inc (2014)
3. Fraser, G. and Wotawa, F. and Ammann, P.: Issues in using model checkers for test case generation. In: Journal of Systems and Software, pp 1403–1418, V82, N°9. Elsevier (2009).
4. Gadkari, A. and Yeolekar, A. and Suresh, J. and Ramesh, S. and Mohalik, S. and Shashidhar, K.: Automatic test case generation from simulink/stateflow models using model checking. In:Software Testing, Verification and Reliability, pp 155-180, V24, N°2. Wiley Online Library (2014)
5. Tekaya, M., and Bennani, M., and Alagui, M., and Ahmed, S.: Aspect-Oriented Test Case Generation from Matlab/Simulink Models. In: Theory and Engineering of Complex Systems and Dependability 2015, pp.495–504. Springer (2015)
6. Clarke, E M. and Grumberg, O. and Peled, D.: Model checking. In: The MIT Press. (2000)
7. Reicherdt, R., and Glesner, S.: Slicing MATLAB simulink models. In: 34th International Conference on Software Engineering (ICSE), pp 551–561. IEEE (2012)
8. Bahrami, D. and Faivre, A. and Lapitre, A.: DIVERSITY-TG : Automatic Test Case Generation from Matlab/Simulink models. In: Embedded real time software and systems, (2012)
9. Peranandam, P. and Raviram, S. and Satpathy, M. and Yeolekar, A. and Gadkari, A. and Ramesh, S. An integrated test generation tool for enhanced coverage of Simulink/Stateflow models. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), pp 308–311. IEEE (2012)
10. Hamon, G. and De Moura, L. and Rushby, J. Automated test generation with SAL. In: CSL Technical Note, pp 15. (2005)
11. Kirner, R. Towards preserving model coverage and structural code coverage. In: EURASIP Journal on Embedded Systems, pp 6. Hindawi Publishing Corp. (2009)
12. Sheeran, Mary. Prover Technology - Prover plug-in documentation (2000)
13. Chilenski, J. and Miller, Steven P. Applicability of modified condition/decision coverage to software testing. In: Software Engineering Journal, pp 193–200, V9, N°5. IET (1994)
14. Rajan, A., Whalen, M and Heimdahl, M.: The effect of program and model structure on MC/DC test adequacy coverage. In: ICSE'08. ACM/IEEE 30th International Conference on Software Engineering, pp 161–170. IEEE (2008)