DIVERSITY – TG
Automatic Test Case Generation from Matlab/Simulink models

Diane Bahrami, Alain Faivre, Arnault Lapitre

CEA, LIST, Laboratory of Model Driven Engineering for Embedded Systems (LISE),

Point Courrier 174, Gif-sur-Yvette Cedex, F-91191 France

{diane.bahrami; alain.faivre; arnault.lapitre}@cea.fr

Keywords: test generation, Matlab/Simulink, coverage criteria, MC/DC, model-based testing

Topics: Process, methods and tools (topic 2) and Dependability (topic 3)

I. Introduction

Reliability of dynamic systems is a major issue in industry (aeronautics, railway, automotive, nuclear…). Therefore, most development processes require the use of model-based approaches such as UML, Matlab/Simulink®, Statemate®, etc. However, resulting models must be checked and corresponding implementations must be tested with respect to these models. In this paper, we present DIVERSITY, the V&V platform of the LISE laboratory, and more specifically Diversity-TG, which is a test case generator. DIVERSITY deals with several modeling languages whose semantics are based on different paradigms concerning execution and communication.

The LISE laboratory has been or is still involved in several French or European projects (SYSPEO[1], ARAMIS[2], EDONA[3] and CESAR[4]) mainly in collaboration with aeronautical and automotive industrial companies which use the Matlab/Simulink environment. Within these projects, we have added several new functionalities into DIVERSITY allowing to automatically generate test cases from Matlab/Simulink specifications with coverage criteria consistent with expectations of industrials.

This paper details Diversity-TG functionalities in the scope of Matlab/Simulink models dedicated to command and control systems.

Section 2 gives an overview of the V&V DIVERSITY platform. Section 3 details Diversity-TG functionalities used on Matlab/Simulink models and test coverage criteria offered to users. Section 4 illustrates these functionalities on a Matlab/Simulink model, with different test coverage criteria. Section 5 compares our approach with Mathwork's Simulink Design Verifier®. Last section allows us to conclude on future works.

II. DIVERSITY Overview

The DIVERSITY platform developed in the LISE laboratory of the CEA LIST, gathers several tools dedicated to the validation and verification of complex embedded software systems. These tools take as inputs, models of complex systems corresponding to specification level and lower design level.

[1] http://www.systematic-paris-region.org/fr/projets/syspeo
[2] http://www.systematic-paris-region.org/fr/projets/aramis
[3] http://www.edona.fr
[4] http://www.cesarproject.eu/

Models may be described with the help of Stateflow-type languages describing potentially concurrent and communicating automata (Statemate, IF, UML statecharts, ...). They can also be characterized using dataflow languages such as the one used in Matlab/Simulink.

The two main tools offered by DIVERSITY are Diversity-TG and Diversity-MC:

- Diversity-TG (TG for Test Generator) can, in a first step, generate simulation scenarios to validate the input model. These scenarios can optionally be used by a simulator associated with the modeling environment (eg Matlab/Simulink simulator). In a second step, Diversity-TG can generate test cases from the same model to verify the compliance of the implementation (or SUT for System Under Test) with the model.
- Diversity-MC (MC for Model Checker) can automatically validate a property with respect to the input model, provided that this property is expressed with inputs/outputs and state variables of the system in a linear temporal logic form. Contrarily to other concurrent approaches based on a full numerical expansion of the model state space, Diversity-MC carries out a symbolic execution based on model-checking of the input model. Thus, the tool can tackle models whose variables are ranking in large or even infinite domains. This is because each calculated symbolic behaviour, being an intentional (constraints based) representation of numerical behaviours, can denote a huge or even an infinite number of numerical behaviours. Nevertheless there are some limitations with the capacity of Diversity-MC. Typically, Diversity-MC is more efficient with protocol-oriented models (components performing simple data treatment and exchanging signals) than with other types of models.

For both tools, the common DIVERSITY process consists in three steps:

1. Firstly, the input model is analyzed and translated into DIVERSITY's internal representation.
2. Then, an exhaustive symbolic exploration of nominal behaviors is performed (symbolic, in order to avoid numerical combinatorial explosion). Behaviors can be selected according to several coverage criteria described in Section 3.

   Moreover, in order to guarantee termination or to limit the number of generated test cases, several basic structural criteria may be used.
3. Finally, Diversity-TG associates each behavior with a numerical test case, and Diversity-MC returns a verdict, according to the user's properties.

The common kernel of these tools is based on symbolic execution. This kernel generates a symbolic execution graph that characterizes all symbolic behaviors of the system.

Several formal techniques are used for this calculation:

- **Symbolic execution:** The major problem with numerical approaches is the combinatorial explosion due to the value fields associated with system parameters. These value fields can be very large or even infinite. Symbolic computation can handle such fields because it characterizes all behaviors that are not equivalent, but without making redundant calculations when different values of the variables correspond to the same behavior (ie the same execution path). The input parameters are not evaluated numerically, but appear as symbolic constants in guards of executed transitions. The guards of executed transitions in an execution path are integrated in a constraint which is the associated condition of the path. This path condition is simply the logical conjunction of all transition guards of the path and must be satisfied to execute this path. The resolution of this path condition produces a sequence of numeric system inputs which corresponds to a concrete test.
- **Reduction process:** The tricky part of symbolic execution is the reduction of expressions that are generated in path conditions. Symbolic expressions may be of significant size, so it is necessary to simplify them on the fly; in that case the duration of intermediate calculations becomes prohibitive. These simplifications are made with the help of rewriting and constraint solving tools.
- **Constraint solver:** When the execution tree is built, all calculated symbolic behaviors of the system can be accessed by consulting the tree. Thus, livelocks and deadlocks may be detected. Then, constraint solvers may be used to obtain numerical values for the system

parameters by resolution of path conditions. Each condition gives way to an input sequence of the system (by default, DIVERSITY computes one numerical solution for each detected symbolic behaviour, since the other numerical solutions are redundant).

Figure 1 shows the architecture details of the DIVERSITY kernel.

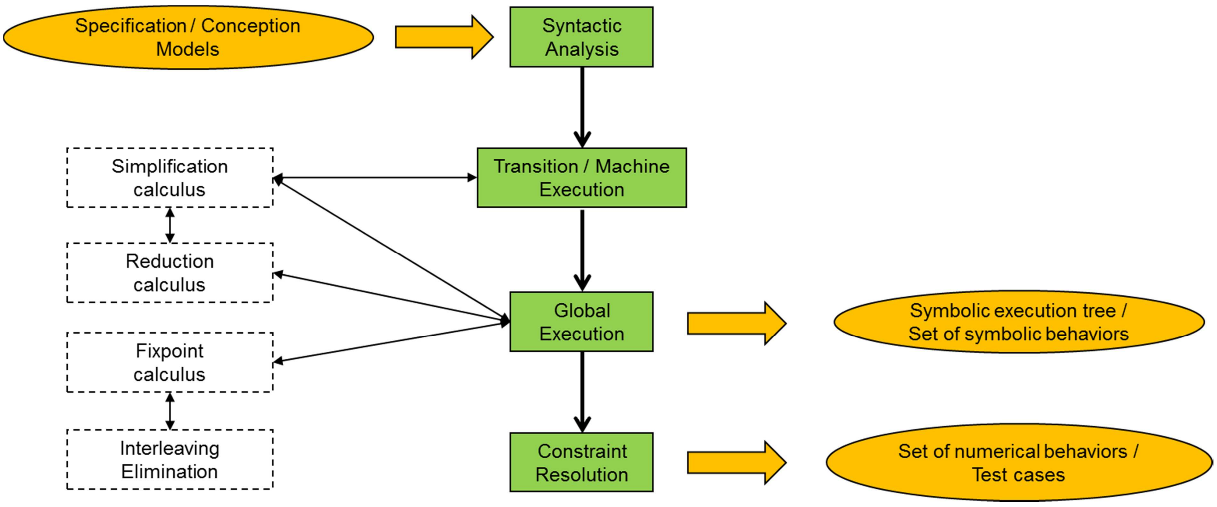

Specification / Conception Models
Syntactic Analysis
Simplification calculus
Transition / Machine Execution
Reduction calculus
Global Execution
Symbolic execution tree / Set of symbolic behaviors
Fixpoint calculus
Interleaving Elimination
Constraint Resolution
Set of numerical behaviors / Test cases

Figure 1: Architecture of the DIVERSITY kernel

In order to deal with combinatory of behaviours and to select the most efficient test cases, Diversity-TG offers several coverage criteria. The first ones are basic coverage criteria which permit to easily control the number of generated test cases with a large coverage of the entire system or selected sub-systems. The last ones, more “intelligent”, comply with industrial V&V standards or are guided by properties given by users to obtain more relevant test cases with regard to system functionalities.

In addition, Diversity-TG selects test cases associated with these coverage criteria using *filters.* Those filters are activated on the fly and associated to pre- or post-treatments during symbolic execution or during analysis of the final symbolic execution tree. On the one hand, existing filters are easily customizable. On the second hand, the architecture of DIVERSITY’s kernel allows to easily add new filters in order to rapidly define new coverage criteria for test generation based on the real needs of various industrial users.

The strategy that we propose is to jointly work with industrial users to ensure that the system models provided initially are compatible with DIVERSITY. If necessary, a first preliminary work consists in improving the models jointly and/or to enlarge DIVERSITY’s corresponding translators in order to achieve this goal. In a second step, a first phase of test generation is performed with the coverage criteria required by the industrial and available in Diversity-TG. After this phase, tests obtained and coverage reached for each criterion are evaluated. If they are not satisfying, the input model can be modified and/or new coverage criteria may be added to Diversity-TG.

III. Test Generation by Diversity-TG from Simulink Models

Matlab/Simulink blocks currently supported by DIVERSITY are all those most frequently used by manufacturers for command and control system models:

- basic mathematical and logical blocks,

- stateflows,
- lookups,
- delays,
- signal routing (mux, demux, switches, from/goto),
- enable, trigger,
- etc.

This list can easily be extended depending on the specificities encountered in the industrial system models. The problem is not usually attached to the translation itself but to the semantic complexity of the block when performing symbolic execution and especially during the phase of constraint solving. A case-by-case study is needed to determine the feasibility of the associated treatment.

Currently, DIVERSITY's main coverage criteria for Matlab/Simulink models are the following:

- Block coverage,
- Transition coverage,
- MC/DC,
- Logical formula coverage,
- Basic structural criteria.

Block coverage

This criterion allows generating test cases which cover a given block. It is interesting to determine under what conditions a block associated with a condition or a trigger is actually executed.

The current version of Diversity-TG generates minimal test cases that cover the execution of the given block. A parameter allows the user to decide when the coverage of a composite system is considered complete: either when the block has been activated independently of its sub-blocks, or only when all of its sub-blocks have been activated.

An immediate possible improvement would be to stop test generation when all symbolic behaviors which allow the block execution are covered by numerical test cases. Another possible improvement would be to stop test generation when all symbolic behaviors of the block have been identified and associated with numerical test cases.

Transition coverage

This criterion identifies all Stateflow charts contained in the Matlab/Simulink model and generates a set of test cases covering as many transitions as possible. At the end of this generation, Diversity-TG gives the rate of transitions covered by the generated test cases and, when appropriate, the list of transitions not covered. Occurrences of transitions not covered by test cases may be due to the fact that these transitions are not really executable. But it may also be due to the setting of limits for depth or number of generated test cases.

A possible improvement would be to select the Stateflow charts that one wants to cover.

MC/DC

Modified condition/decision coverage (MC/DC), was first defined in the DO-178B standard to ensure that a Level A software is tested adequately. Currently, this coverage criterion is referenced in other industrial standards, for example automotive (ISO 26262) where Matlab/Simulink models are used.

In the case of Matlab/Simulink models, this criterion can be applied to any logic block, that is to say, any block associated with a logic function with one or more boolean inputs (the conditions) and one boolean output (the decision). All conditions and the decision must be at least tested once to *true* and once to *false*. Moreover, the test set must show that each condition can affect the decision value independently of the other conditions. This means that, just by changing the value of that condition, while holding the others, the value of decision can change. This is illustrated in Figure 2.

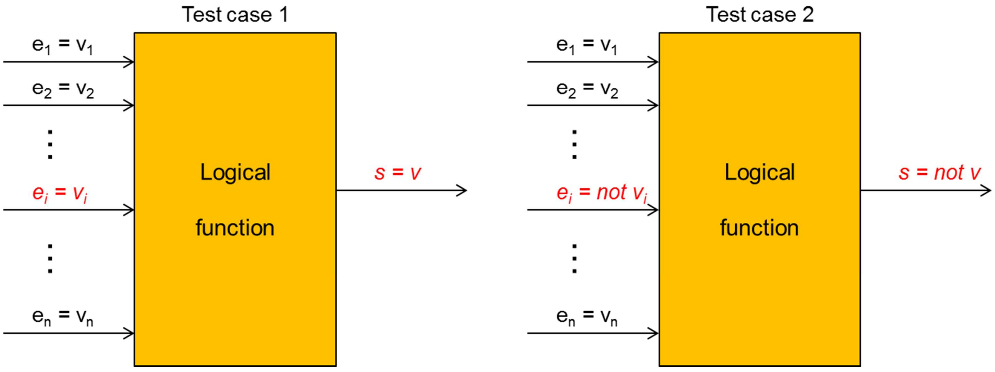


Figure 2: Independence of condition v_i on decision v

Two other logical coverage criteria are also offered by Diversity-TG:

- Decision coverage: outputs of every logical block are tested with the values true and false,
- Decision/Condition coverage: all inputs and outputs of all logical blocks are tested with the values true and false.

Logical formula coverage

This criterion may be used to generate test cases with previous coverage criteria in a particular context defined by properties expressed with the help of the model variables.

Here are examples of properties:

- *mode = normal,* or *mode = weakened*
- $10 \leq speed \leq 100$
- *nb_passengers > 300*

Basic structural criteria

All the criteria mentioned above can be used in conjunction with basic structural criteria which limit the number of generated test cases or the number of simulation cycles for each test case.

DIVERSITY's kernel is designed to add very quickly new customized coverage criteria defined by the industrial user, using « filters ».

Similarly, when setting up the execution of Diversity-TG, it is possible to reference simultaneously these filters to cleverly combine several coverage criteria in test generation.

IV. Industrial Examples

Several industrial Matlab/Simulink models were treated with Diversity-TG, in order to generate test cases according to the coverage criteria mentioned above.

Among these models, we may quote those provided in the following projects:

- The EDONA project
  Gathering at the national level all industrial actors of the automobile embedded software field, EDONA « Environnements de Développement Ouverts aux Normes de l'Automobile » (an Open Development Platform for Automotive Standards) is a project of the pole of competitiveness System@tic Paris-Area.
  In this project, we worked with a Matlab/Simulink model for the « management of external and internal lighting of a car ».
- The CESAR project
  CESAR stands for « Cost-Efficient methods and processes for SAfety-Relevant embedded systems » and is a European funded project from ARTEMIS JOINT UNDERTAKING (JU). In this project, we worked with a Matlab/Simulink model for the « management of opening/closing of airplane doors ».
- The ARAMIS project
  The ARAMIS project is a part of Num@tec Automotive. The main objective is to promote the use of Model-Based design methodologies associated with automatic embedded code generation and to enhance the validation means.
  In this project, we worked with a Matlab/Simulink model for the « management of air conditioning for a hybrid vehicle ».

Table 1 shows the size and complexity of the three previous models.

Model description			
Model	Nb of sub-system blocks	Nb of basic blocks	Nb of inputs / outputs
EDONA	37	167	15 / 6
CESAR	14	41	23 / 11
ARAMIS	12	22	8 / 7

Table 1: Size and complexity of models

We have generated test cases with Diversity-TG from these three models, with different coverage criteria used independently of each other. Table 2 shows that these models do not raise any size or complexity problem for Diversity-TG.

Coverage criteria						
Model	Structural	MC/DC	Decision	Decision/Condition	Block	Transitions
EDONA	Ok	Ok	Ok	Ok	Ok	▨
CESAR	Ok	Ok	Ok	Ok	Ok	Ok
ARAMIS	Ok	Ok	Ok	Ok	Ok	Ok

Table 2: Test case generation verdict

The shaded box corresponds to a model with no Stateflow.

V. Diversity-TG versus Simulink Design Verifier

Currently, very few industrial tools automatically generate test cases from Matlab/Simulink specifications with coverage criteria consistent with the needs of industry. The MathWorks, supplier of Matlab/Simulink, provides such a tool: Simulink Design Verifier. We only found one other tool, called Reactis®, provided by Reactive Systems®.

Since we worked with a short-time evaluation licence of Reactis, we only present here a comparative study between Diversity-TG and Simulink Design Verifier (Version 1.1.1).

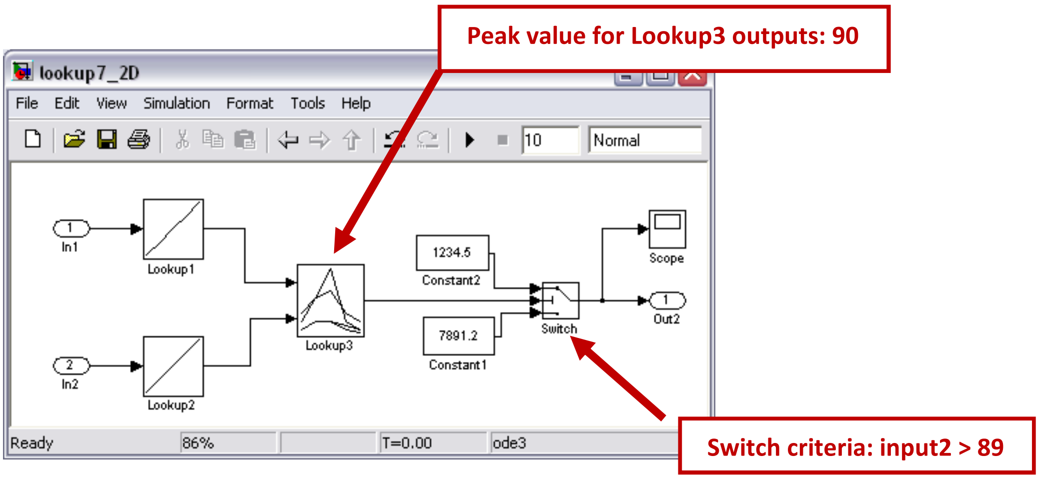
Test generation strategy

Simulink Design Verifier, like Reactis®, uses random test generation, which is quite fast, but has a major drawback: it fails in finding test cases covering specific behaviors (since the probability of finding them randomly is weak). It may be crucial to test these specific behaviors for safety and security reasons.

The following example, shown in Figure 3, illustrates this failure.

It describes a model containing two 1D lookup, a 2D lookup and a switch. The 1D [resp. 2D] lookup block computes an approximation to some function y=f(x) by performing linear interpolation, given a 1D table [resp. two 1D tables] as input and a 1D table [resp. one 2D table] as output.

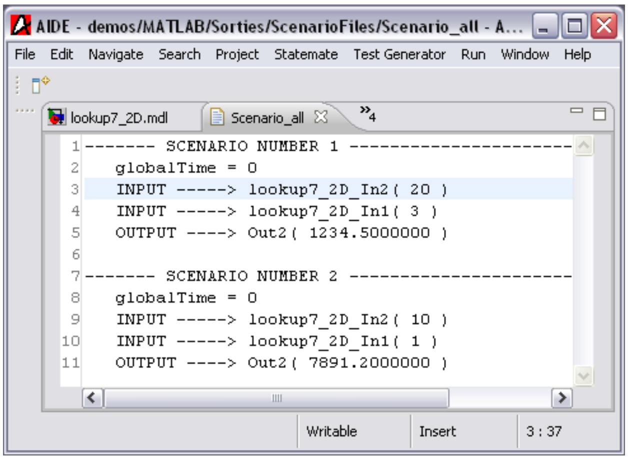
Our aim with this model is to reach both possible outputs of the system, i.e. to cover both possible behaviors of the Switch block.


Peak value for Lookup3 outputs: 90

Switch criteria: input2 > 89

Figure 3: Example of low probability behavior

Diversity-TG needs one step (or cycle) to find the two expected test scenarios shown in Figure 4, whereas Simulink Design Verifier only finds one scenario and goes on computing without finding the other. Its result states that *objective 2* is *undecidable,* with test generation options: condition/decision and max steps: 1000.



```
------- SCENARIO NUMBER 1 ----------------------
   globalTime = 0
   INPUT -----> lookup7_2D_In2( 20 )
   INPUT -----> lookup7_2D_In1( 3 )
   OUTPUT ----> Out2( 1234.5000000 )

------- SCENARIO NUMBER 2 ----------------------
   globalTime = 0
   INPUT -----> lookup7_2D_In2( 10 )
   INPUT -----> lookup7_2D_In1( 1 )
   OUTPUT ----> Out2( 7891.2000000 )
```

Figure 4: Test scenarios generated with Diversity-TG

Coverage criteria

The only test coverage criteria handled by Simulink Design Verifier are Condition, Condition/Decision and the MC/DC criterion, whereas Diversity-TG handles three more criteria and offers facilities to easily add new coverage criteria (see section III).

Moreover, in Diversity-TG, we chose to apply the MC/DC criterion to any logical function (be it a basic logical operator or a whole subsystem containing logical sub-blocks only), whereas Simulink Design Verifier only applies it on basic logical blocks or Fcn blocks in which the user has defined a more complex logical function.

VI. Conclusion

In this paper, we have presented Diversity-TG, a tool which automatically generates test cases and which is based on the symbolic execution kernel of the DIVERSITY platform. This presentation is done in the context of Matlab/Simulink models. This deterministic approach allows us to characterize exhaustively all symbolic behaviors of the model that are consistent with the coverage criteria set for test case generation.

The deterministic approach of Diversity-TG differs from the random approach of the two other equivalent industrial tools that we have tested. The deterministic approach has the advantage of producing, in a systematic way, all test cases associated with the coverage criteria but with variable response times. The probabilistic approach, more efficient in execution time, may not generate all test cases for low probabilistic behaviors. In addition, we have noticed that, in the considered examples, Diversity-TG usually generates a set of tests smaller than those generated by other tools, therefore more optimal, for coverage criteria like MC/DC.

We are currently taking into account new Matlab/Simulink blocks into DIVERSITY, in order to be able to treat new industrial models. A proposed tool evolution is to put back Diversity-TG's generated test cases into Matlab/Simulink's simulator, so that they can allow the user to validate the model by an efficient simulation. Another development will be to connect constraint solving tools better suited to some specific models.