Learn and Test for Event-B – a Rodin Plugin

Ionut Dinca, Florentin Ipate, Laurentiu Mierla, and Alin Stefanescu

University of Pitesti, Department of Computer Science
Str. Targu din Vale 1, 110040 Pitesti, Romania
name.surname@upit.ro

Abstract. The Event-B method is a formal approach for reliable systems specification and verification, being supported by the Rodin platform, which includes mature plugins for theorem-proving, model-checking, or model (de)composition features. In order to complement these techniques with test generation and state model inference from Event-B models, we developed a new feature as a Rodin plugin. Our plugin implements a model-learning approach to iteratively construct an approximate automaton model together with an associated test suite. Test suite optimization is further applied according to different optimization criteria.

1 Introduction

This short tool paper presents the implementation in the Rodin platform of the general method "learn-and-test" described in our previous paper [2]. For a given Event-B model [1], the method constructs, in parallel, an *approximate automaton* model and a *test suite* for the system. The approximate model construction relies on a variant of Angluin's automata learning algorithm [3, 4], adapted to finite cover automata [5]. A *finite cover automaton* represents an approximation of the system which only considers sequences of length up to an established upper bound ℓ. Crucially, the size of the cover automaton, which normally depends on ℓ, can be significantly lower than the size of the exact automaton model. In this way, by appropriately setting the value of the upper bound ℓ, the state explosion problem normally associated with constructing and checking state based models can be addressed. The proposed approach also allows for a gradual construction of the model and of the associated test suite (reusing information between iterations), which fits well with the central notion of refinement in Event-B [1].

2 Tool Overview

A bird's eye view of the tool is depicted in Fig. 1. The tool takes as input an Event-B model M and a finite bound ℓ and outputs a finite cover automaton approximating the set of feasible sequences of events of M of length up to ℓ and a test suite, i.e. a set of sequences including test data that make the sequences executable. The core procedure of "Model Learning" generates a cover automaton using a variant of automata learning from queries [3]. Simply put, a cover

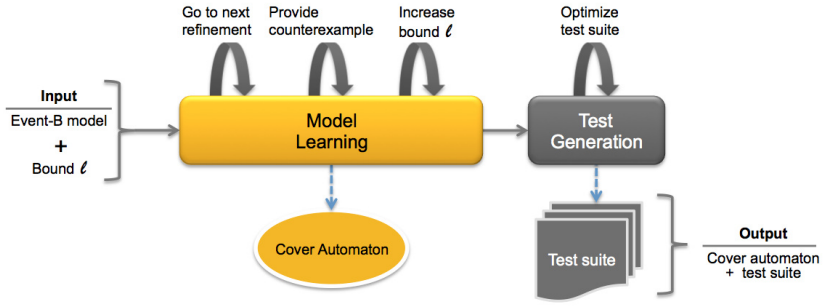

Fig. 1. Overview of the tool features.

automaton for a finite set of words of length up to ℓ, is an automaton accepting all these words but also sequences that may be longer than ℓ. The cover automaton can be incrementally improved by providing more information according to the three loops in the figure. Thus, one can: (a) use the "next refinement" of the Event-B model that contains more information; or (b) "provide a counterexample" by manually or automatically providing sequences that are feasible in the Event-B model, but are not in the cover automaton or vice-versa (the counterexamples are used in the learning procedure); or (c) increase the bound ℓ and implicitly feed the learning engine with longer sequences which again will increase the precision of the finite state approximation. At any point in time, one can use the constructed cover automaton to generate tests that exercise different sequences through the Event-B model. There are many existing methods for test generation from finite state models. In our case, we use internal information from the learning procedure, which maintains a so-called "observation table" that keeps track of the learned feasible sequences. Sets of feasible sequences in this table will provide the desired test suite. Note that during the feasibility check of the sequences in Event-B, test data are also generated. The implementation of feasibility check uses a constraint-solver for Event-B available in ProB [6]. The obtained test suite satisfies strong criteria for conformance testing (usually required in the embedded system domain) and may be large. If weaker test coverage like state-, transition- or event-coverage are desired, optimization algorithms can be applied on the test suite according to the rightmost loop in Fig. 1. We implemented different optimizations as proposed by one of the co-authors in [7] using the jMetal framework which is based on genetic algorithms.

Our tool is a Rodin plugin implemented in Java (with 5,500 LOC) and can be called on any Event-B model with several levels of refinements. Installation instructions and screenshots can be found at: `http://wiki.event-b.org/index.php/MBT_plugin`. Ongoing extensions of the tool tackle not only refinement, but also different types of Event-B decompositions. Experiments with different Event-B models (publicly available on the DEPLOY repository - `http://deploy-eprints.ecs.soton.ac.uk`) produced good results even for large models like BepiColombo [8] (whose third refinement exhibits 17 events

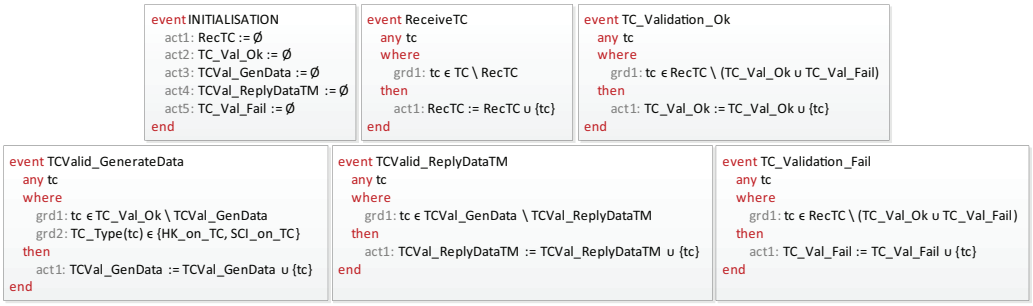

Fig. 2. The events of the abstract machine M_0 in BepiColombo Event-B model [8]

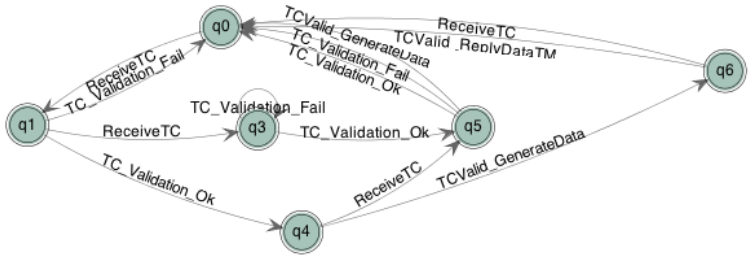

Fig. 3. The generated cover automaton for M_0 and $\ell = 4$

and 18 variables that could induce a large explicit state space for the model). This example is discussed below.

3 The Tool Applied to an Example

An Event-B model has a context providing the data types and an abstract state machine providing the dynamic behavior. The machine has a set of events, which are the first class citizens of Event-B, that operate on a set of global variables. The modeling complexity is addressed using *refinement* as a mechanism to construct a series of more abstract models before reaching a very specific one. For instance, in a refinement step, new variables and new events can be introduced and the existing events can be made more specific.

The BepiColombo aerospace mission is one of the case study used in the DEPLOY project (`http://deploy-project.eu`). In [8], a part of BepiColombo is modeled in Event-B using several levels of refinements (combined with atomic and model decompositions which we do not address here). The main goal of the system is specified at a very abstract level, with a machine M_0. The system specification is concretized through three further refinement levels, M_1, M_2 and M_3. Fig. 2 presents the five events of M_0, plus a special event called 'Initialisation'. Each event has local parameters preceded by the keyword *any*, a guard preceded by the keyword *where*, and an action code preceded by the keyword

then. There exist also global variables (like $RecTC$ of type Set), that are initialized in the event 'Initialisation'. Once the 'Initialisation' event is executed, the modeled system moves from one state to another by choosing one event with its guard true and executing its action code.

Given the BepiColombo Event-B model and an upper bound ℓ, we incrementally construct finite cover automata that will eventually cover all executable event sequences of length less than or equal to ℓ. Fig. 3 (plotted by our tool) illustrates the cover automaton for the first machine M_0 and $\ell = 4$, minimal by construction, having the initial state marked with q_0, transitions labeled with event names and final states marked with a double circle. Starting from the state q_0, the event sequences can be identified by following the transitions with the purpose of reaching the automaton final states, representing a subset of the communication scenarios the spacecraft system may encounter.

A conformance test suite heavily exercising the system would consist of 17 test cases. Conformance testing is a very powerful test type since it covers all states and all transitions of the automaton and also checks each state and the initial and destination states of each transition. However, for a lighter test coverage like event coverage, a test suite consists of only 2 test cases (of length up to 4): (a) `ReceiveTC(tc1), TC_Validation_Ok(tc1), TCValid_GenerateData(tc1), TCValid_ReplyDataTM(tc1)` and (b) `ReceiveTC(tc2), TC_Validation_Fail(tc2)`.

Acknowledgments. This work was supported by project DEPLOY, FP7 EC grant no. 214158, and Romanian National Authority for Scientific Research (CNCS-UEFISCDI) grant no. PN-II-ID-PCE-2011-3-0688 (project MuVet) and grant no. 7/05.08.2010.

References

1. Jean-Raymond Abrial. *Modeling in Event-B – System and Software Engineering.* Cambridge University Press, 2010.
2. Florentin Ipate, Ionut Dinca, and Alin Stefanescu. Model learning and test generation using cover automata. Submitted to IEEE Trans. on Software Engineering., 2012.
3. Florentin Ipate. Learning finite cover automata from queries. *Journal of Computer and System Sciences*, 78:221–244, 2012. In press.
4. Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.
5. Cezar Câmpeanu, Nicolae Sântean, and Sheng Yu. Minimal cover-automata for finite languages. *Theoret. Comput. Sci.*, 267(1–2):3–16, 2001.
6. Michael Leuschel and Michael J. Butler. ProB: an automated analysis toolset for the B method. *Int. J. Softw. Tools Technol. Transf.*, 10(2):185–203, 2008. Tool webpage: `http://www.stups.uni-duesseldorf.de/ProB`.
7. Ionut Dinca. Multi-objective test suite optimization for Event-B models. In *Proc. of ICIEIS'11*, volume 251 of *CCIS*, pages 551–565. Springer, 2011.
8. Asieh Salehi Fathabadi, Abdolbaghi Rezazadeh, and Michael Butler. Applying atomicity and model decomposition to a space craft system in Event-B. In *Proc. of NFM'11*, volume 6617 of *LNCS*, pages 328–342. Springer, 2011.