Towards Search-based Testing for Event-B Models

Alin Stefanescu, Florentin Ipate, Raluca Lefticaru, Cristina Tudose
Department of Computer Science, University of Pitesti
Str. Targu din Vale 1, 110040 Pitesti, Romania
{*name.surname*}*@upit.ro*

***Abstract*—This position paper discusses the challenges and opportunities of applying search-based techniques to a formal environment of abstract state machines defined using a language called Event-B. Event-B is based on a formal abstract machine notation that has a mature tool support and gets continuous feedback from industry. Although search-based techniques recently developed for extended finite state machines may be adapted to this context, new challenges such as implicit states, non-determinism, non-numerical datatypes and hierarchical models are still to be solved for test data generation for Event-B models.**

***Keywords*-Abstract state machines; Event-B; model-based testing; search-based software testing.**

I. Motivation

The B-method [1] is a formal approach to software engineering introduced in the 90s, aimed at the safety-critical software domain and supported by both academic and commercial tools. B is one of the success stories of the application of formal methods in industry, e.g., major metro systems around the world use nowadays software generated from formally verified B models. The strengths of B-method reside in the rigorous formalism based on abstract state machines and set theory as well as the necessary tool support required by industry. While successful in the area of embedded systems, B is not very suitable for formal system modeling, mainly due to its focus on the lower level implementation aspects.

To address this, the Event-B language [2] was introduced in the 2000s, as an evolution of the B language, having its focus on software systems rather than software implementations and with events as first class citizens rather than operations and methods. Event-B development was supported by two European research projects: RODIN[1] and DEPLOY[2]. RODIN provided a first platform for Event-B called Rodin, whereas DEPLOY further improves the platform based on industrial feedback. There are four industrial partners in DEPLOY (Bosch, SAP, Siemens and Space Systems Finland) that experiment with the latest versions of the platform and provide new requirements and feedback. The core technology behind Rodin platform is theorem-proving, but also model-checking tools have been developed (ProB [3]). Recently, there has been an increasing interest from the Event-B users into test generation based on Event-B models. This in fact provided the main motivation behind our investigations based on search-based testing.

Model-based testing (MBT) is an approach that uses formal models as basis for automatic generation of test cases. There are different types of test models that can be used for MBT, many of them being state-based models, for instance Unified Modeling Language (UML) state diagrams. There exists a first MBT approach that uses Event-B models as test models [4]. The interest in MBT is given by the opportunity of using the Event-B models not only for formal verification of the specification (via theorem proving or model checking) but also to verify using test cases that an existing implementation behaves as specified in Event-B.

For MBT using state-based models, test generation algorithms usually traverse the state space starting in an initial state and are guided by a certain coverage criterion (e.g. state coverage) collecting the execution paths in a test suite. Event-B models do not have an explicit state space, but their state spaces are given by the value of the variables. Then the state is changed by the execution of a given event that is enabled in that state. The ProB tool [3] that is available in the Rodin platform has a good control of the state space, being able to explore it, visualize it and verify various properties using model checking algorithms. Such model checking algorithms can be used to explore the state space of Event-B models using certain coverage criteria (e.g. event coverage) and thus generating test cases along the traversal. Moreover, the input data that triggers the events provides the test data associated with the test cases. Such an approach using explicit model-checking has been applied to models from the business application area by SAP [4]. The algorithms perform well for models with data with a small finite range. However, in case of variables with a large range (e.g. integers), the known state space explosion problem creates difficulties, since the model checker explores the state space enumerating the many possible values of the variables. Search-based techniques may provide a solution to this problem, by guiding the search for a solution in the state space, instead of exhaustively enumerating all values.

In this paper, we discuss the possibility of using meta-heuristic search algorithms in test generation for Event-B models. For one path, one could solve the test data problem by starting with an initial set of data and improving it using

[1]http://rodin.cs.ncl.ac.uk
[2]http://deploy-project.eu

guidance from the constraints on the paths to be satisfied (using certain fitness functions that describe "how far" is the current data from a solution). Meta-heuristic search algorithms can be applied not only to test data generation for one path but also to obtain a set of test cases with the required coverage. This seems similar to the recent work on search-based approaches for Extended Finite State Machines (EFSMs), cf. [5]–[7]. However, there are several problems to be properly understood and tackled for such search-based techniques to work because the Event-B framework has certain characteristics that makes it different from the EFSM domain. Understanding these new research problems to be solved is the main topic of this paper.

The paper is structured as follows. We first describe Event-B in Section II, then we discuss the encountered challenges in Section III. We continue with related work in Section IV and conclude with a discussion on the current state of the affairs in Section V.

II. Formal Modeling with Event-B

Event-B [2] is a language based on the notion of abstract machines having set theory as its mathematical language and using theorem proving as the main tool-supported validation on the Rodin platform. The main constructs of Event-B are *contexts* and *machines*. The contexts specify the static part of a model and contain the following elements: *carrier sets* (i.e. domains), *constants* and *axioms*. The axioms can define relations between the constants or define their domains. Machines represent the dynamic part of a model and can contain *variables, invariants* and *events*.

An event consists of two main elements: *guards* which are predicates to describe the conditions that need to hold for the occurrence of an event and *actions* which determine how specific variables change as a result of the event execution. An event has the form

$$\text{Event} \mathrel{\widehat{=}} \textbf{any } t \textbf{ where } G(t,x) \textbf{ then } S(x,t) \textbf{ end}.$$

It consists of a set of local variables t, a predicate G, called the guard and a substitution $S(x,t)$. The guard restricts possible values for t and x. If the guard of an event is false, the event cannot occur and is called disabled. The substitution S modifies the variables x. It can use the old values of x and the local variables t.

For example, an event that takes a natural number q strictly greater than 100 and adds it to the natural (global) variable $quantity$ could be written as

$$\text{Event} \mathrel{\widehat{=}} \textbf{any } q \textbf{ where } q \in \mathbb{N} \land q > 100 \textbf{ then } \mathit{quantity} := \mathit{quantity} + q \textbf{ end}.$$

Note that if the model has 10 integer variables with their range in [1..10,000], the explicit state space will have 10^{40} states, which is usually too much for a brute-force traversal algorithm of an explicit model checker. Since the theorem proving is mainly used for Event-B, explicit enumerations are generally avoided.

Let us consider another example, involving a set defined as *ITEMS*={*item1, item2, ... , item20*}. Then, one can have an event in which a variable *items* is updated with a new set of items i, assuming that the cardinality of i is greater than 10, and then an item *oneItem* is randomly picked (using the Event-B operator $:\in$) from the set i:

$$\text{Event} \mathrel{\widehat{=}} \textbf{any } i \textbf{ where } i \in \mathbb{P}(\mathit{ITEMS}) \land \text{card}(i) > 10 \textbf{ then } \mathit{items} := \mathit{items} \cup i \ \land \ \mathit{oneItem} :\in i \textbf{ end}.$$

Succinctly, an Event-B model is given by the defined domains, constants, variables, events that change the global variables when enabled and executed, and a set of global invariants specifying the properties required by the specification. The execution of a model starts with a special event that initializes the system, followed by the application of enabled events. At each execution step, all the guards of the events are evaluated and the set of enabled events is computed. Then, one enabled event is non-deterministically chosen and its action is executed. The Rodin platform, built on top of Eclipse, provides different plugins that manage and perform different tasks on the models. For example, there are built-in theorem-provers that ensure that the defined invariants always hold for the model, there is the ProB model-checker that can verify certain properties of the system (e.g. deadlock freedom) and there are many other plugins like composition/decomposition plugin that helps the users to compose or decompose their models into smaller sub-models. The main pattern of constructing larger models is to use the notion of refinement, which allows the user to start with more abstract models and gradually refine them by adding more details and formally proving that the invariants are always preserved along the way.

III. Search-based Testing Challenges

Test generation from an Event-B based specification does not simply reduce to applying one of the search based techniques existing in the literature, since it may pose a number of non-trivial problems as described below:

- **Implicit states:** Event-B specifications are event-based rather than state-based models. Formally, these are abstract state machines [8] in which the (implicit) states are given by the data structures on which the events operate. For large systems, transforming this into an (extended) finite state machine may not be desirable and so test coverage criteria for the original model may need to be defined (e.g. event coverage).
- **Non-determinism:** Often Event-B models are non-deterministic (see the $:\in$ operator and the non-deterministic choice of an enabled event in the previous section). In order to test *non-deterministic* implementations, one usually makes a so-called *complete-testing assumption* [9]: there exists a positive integer N such

that, by applying any input test data set d to the implementation N times, all the paths of the implementation that can be traversed by d will actually be exercised. In practice, this means that, every time the fitness function for a candidate solution d has to be computed, the input d will be applied to the executable model a given number of times N and the resulting paths P will be recorded. Naturally, the expression of the fitness function $f(d)$ for d will depend on the coverage criterion sought. For example, when the test generation algorithm seeks input data to exercise a given path p, it would be natural to evaluate the fitness of the element of P which is "closest" to p, i.e. if the fitness function is minimized then $f(d) = \min_{p \in P} f(p)$.

- **Non-numerical data types:** Very often the data structures used in Event-B models (and hence the event guards) are non-numeric. Most of the models we have considered so far involved relatively simple data structures, involving enumeration types, which could be easily mapped onto numeric types. More complex types like sets or partial functions need further investigations. We currently extend the fitness functions for numerical and boolean types used in the literature [10] to operations on sets. For instance, for the set membership operation $k \in S$, where S is a set and k an element, the fitness function $f(k \in S)$ is defined as $f(s \in S) = \min_{s \in S} f(k == s)$. Moreover, if we use genetic algorithms, efficient encodings of the sets into chromosomes are needed.
- **Hierarchical models:** Complex systems are seldom designed in one step; usually, the design is constructed gradually, through a process of *refinement*. In such instances, generating test sets directly for the final, refined, system may not be feasible. Instead, one would wish to apply test generation methods to the simpler, unrefined version of the system and to develop the resulting test sets in parallel to the refinement of the specification. Furthermore, large systems can be structured into simpler components through *composition-decomposition*. Similarly, a testing strategy in which tests for the components are integrated into the test set of the whole system would be needed. Existing testing strategies for hierarchical and communicating finite state machines [11] could be used as a starting point for devising appropriate testing methods for Event-B models involving refinement and composition-decomposition.

IV. Related Work

The single reference regarding testing based on Event-B is [4]. The paper presents a service integration testing approach using service choreographies models developed by SAP. These models are based on communicating EFSMs. The models are translated into Event-B and processed with ProB model checker, which generates test suites using explicit state space exploration. Although the authors try to address the state explosion problem by e.g. symmetry reduction techniques, this general issue is still present.

ProB was also used for testing based on B models, possibly non-deterministic, in [12]. Since the problem of obtaining a precondition satisfaction predicate out of a set of B constraints is NP-complete, the algorithm works for rather smaller and simpler operations. Non-deterministic models were addressed by introducing additional result parameters that make the internal choices visible.

Model checking tools face a combinatorial blow up of the state-space, known as the state explosion problem. A survey on testing with model checkers is given in [13]. Symbolic and bounded model checking try to alleviate this problem.

Search-based software testing has been applied mostly to white-box or structural testing [10], [14]. Some approaches to evolutionary functional testing tackle model-based testing from state machines and the main directions identified were:

- Generating test data for feasible paths in state machines: A fitness function of the type *approach level + normalized branch level*, inspired from structural search-based testing [10], is proposed in [5], [15] and further used for experimentation in [16], [17]. An empirical study on the efficiency of search-based test generation for EFSMs is given in [17] and one finding is a positive correlation between the test generation cost and some metrics, such as the number of numerical equal operators in the conditions from the path.
- Estimating the feasibility of transition paths and generating transition paths which are more likely to be feasible [6], [18].
- Combining the generation of feasible test paths with the generation of the associated input test data [7]. In this paper a multi-objective evolutionary algorithm is proposed, which considers two objectives: to search for a test sequence that covers a target transition, as well as to minimize the length of this test sequence.

V. Conclusion

In this position paper, we have presented recent efforts towards applying search-based techniques in a new industrial-driven setting provided by the Event-B language. We have sketched some of the main characteristics of Event-B, outlined the existing challenges of applying meta-heuristic search techniques, and performed a literature review of the similar approaches that may be used as a source of inspiration for solving the identified problems. A couple of first solution ideas were also sketched, but we are also seeking feedback or helpful experiences from the search-based testing community.

We are currently implementing our ideas as an Eclipse-plugin integrated in Rodin platform and we are performing the first experiments with models provided by industrial

partners like SAP. We started by using our previous experience in search-based techniques for EFSMs [5], [15] and applied it to small Event-B models that can be easily mapped onto EFSMs with only numerical variables, using fitness functions existing in the literature. In the implementation we make use of the Event-B simulator offered by ProB tool which evaluates the guard satisfiability at runtime. We are now moving forward in addressing non-numerical datatypes such as sets and also addressing the issues related to non-determinism. In parallel, we are systematically evaluating and classifying the existing DEPLOY repository of Event-B models[3], containing a couple of dozens of models, both from academic and industrial partners. The used modeling patterns can help us prioritize which problems to address first and see which of them are suitable for search based techniques, e.g. we have already noticed that many models have an EFSM-like structure (for instance SAP domain-specific models that are translated in Event-B are originally state-based; also Bosch Event-B models of cruise controllers have state-based models behind).

Once our prototype will be able to handle different specific Event-B characteristics, as extracted from the model repository, and once we have implemented different fitness functions, a detailed comparison among the fitness functions but also against the model checking approach will be performed following a sound statistical approach [19]. Regarding model checking, we can also investigate the usage of search-based techniques in guiding the explicit model checker during the state space traversal, cf. [20].

Acknowledgments: This work was partially supported by project Deploy, EC-grant no. 214158, and Romanian Research Grant CNCSIS-UEFISCDI no. 7/05.08.2010 at the University of Pitesti. We thank V. Kozyura and S. Wieczorek from SAP Research for interesting models and inspiring discussions.

REFERENCES

[1] J.-R. Abrial, *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[2] ——, *Modeling in Event-B - System and Software Engineering*. Cambridge University Press, 2010.

[3] M. Leuschel and M. J. Butler, "ProB: an automated analysis toolset for the B method," *Int. J. Softw. Tools Technol. Transf.*, vol. 10, no. 2, pp. 185–203, 2008.

[4] S. Wieczorek, V. Kozyura, A. Roth, M. Leuschel, J. Bendisposto, D. Plagge, and I. Schieferdecker, "Applying model checking to generate model-based integration tests from choreography models," in *Proc. TESTCOM'09*, ser. LNCS, vol. 5826. Springer, 2009, pp. 179–194.

[5] R. Lefticaru and F. Ipate, "Functional search-based testing from state machines," in *Proc. ICST'08*. IEEE Computer Society, 2008, pp. 525–528.

[3] http://deploy-eprints.ecs.soton.ac.uk/view/type/rodin=5Farchive.html

[6] K. Derderian, R. M. Hierons, M. Harman, and Q. Guo, "Estimating the feasibility of transition paths in extended finite state machines," *Autom. Softw. Eng.*, vol. 17, no. 1, pp. 33–56, 2010.

[7] T. Yano, E. Martins, and F. L. de Sousa, "Generating feasible test paths from an executable model using a multi-objective approach," in *Proc. ICSTW'10*. IEEE Computer Society, 2010, pp. 236–239.

[8] E. Börger and R. Stärk, *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer-Verlag, 2003.

[9] G. Luo, G. von Bochmann, and A. Petrenko, "Test selection based on communicating nondeterministic finite-state machines using a generalized Wp-method," *IEEE Trans. Softw. Eng.*, vol. 20, no. 2, pp. 149–162, 1994.

[10] P. McMinn, "Search-based software test data generation: A survey," *Softw. Test. Verif. Reliab.*, vol. 14, no. 2, pp. 105–156, 2004.

[11] F. Ipate, "Test selection for hierarchical and communicating finite state machines," *Comput. J.*, vol. 52, no. 3, pp. 334–347, 2009.

[12] M. Satpathy, M. Butler, M. Leuschel, and S. Ramesh, "Automatic testing from formal specifications," in *Proc. TAP'07*, ser. LNCS, vol. 4454. Springer, 2007, pp. 95–113.

[13] G. Fraser, F. Wotawa, and P. Ammann, "Testing with model checkers: a survey," *Softw. Test. Verif. Reliab.*, vol. 19, no. 3, pp. 215–261, 2009.

[14] K. Lakhotia, P. McMinn, and M. Harman, "An empirical investigation into branch coverage for C programs using CUTE and AUSTIN," *J. Syst. Softw.*, vol. 83, no. 12, pp. 2379–2391, 2010.

[15] R. Lefticaru and F. Ipate, "Automatic state-based test generation using genetic algorithms," in *Proc. SYNASC'07*. IEEE Computer Society, 2007, pp. 188–195.

[16] A. Arcuri, M. Z. Iqbal, and L. Briand, "Black-box system testing of real-time embedded systems using random and search-based testing," in *Proc. ICTSS'10*, ser. LNCS, vol. 6435. Springer, 2010, pp. 95–110.

[17] R. Zhao, M. Harman, and Z. Li, "Empirical study on the efficiency of search based test generation for EFSM models," in *Proc. ICSTW'10*. IEEE Computer Society, 2010, pp. 222–231.

[18] A. Kalaji, R. M. Hierons, and S. Swift, "Generating feasible transition paths for testing from an extended finite state machine (EFSM)," in *Proc. ICST'09*. IEEE Computer Society, 2009, pp. 230–239.

[19] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proc. ICSE'11*, to appear.

[20] F. Chicano and E. Alba, "Searching for liveness property violations in concurrent systems with ACO," in *Proc. GECCO'08*. ACM, 2008, pp. 1727–1734.