B Model Abstraction Combining Syntactic and Semantic Methods

J. Julliand[1], N. Stouls[2], P.-C. Bué[1], and P.-A. Masson[1]

[1] LIFC, Université de Franche-Comté
16, route de Gray F-25030 Besançon Cedex
{bue, julliand, masson}@lifc.univ-fcomte.fr

[2] Université de Lyon, INRIA
INSA-Lyon, CITI, F-69621, Villeurbanne, France
nicolas.stouls@insa-lyon.fr

Abstract. In a model-based testing approach as well as for the verification of properties by model-checking, B models provide an interesting solution. But for industrial applications, the size of their state space often makes them hard to handle. To reduce the amount of states, an abstraction function can be used, often combining state variable elimination and domain abstractions of the remaining variables. This paper illustrates a computer aided abstraction process that combines syntactic and semantic abstraction functions. The first function syntactically transforms a B event system M into an abstract one A, and the second one transforms a B event system into a Symbolic Labelled Transition System (SLTS). The syntactic transformation suppresses some variables in M. This function is correct in the sense that A is refined by M. A process that combines the syntactic and semantic abstractions has been experimented. It significantly reduces the time cost of semantic abstraction computation. This abstraction process allows for verifying safety properties by model-checking or for generating abstract tests. These tests are generated by a coverage criteria such as all states or all transitions of an SLTS.

Keywords: Model Abstraction, Syntactic Abstraction, Refinement.

The full version of this short paper is available as a research report: [JSBM09].

1 Introduction

B models are well suited for producing tests of an implementation by means of a *model-based testing* approach [BJK+05,UL06] and to verify dynamic properties by model-checking [LB08]. But model-checking as well as test generation requires the models to be finite, and of tractable size. This usually is not the case with industrial applications, and the search for executions instantiated from the model frequently comes up against combinatorial explosion problems. Abstraction techniques allow for projecting the (possibly infinite or very large) state space of a system onto a small finite set of symbolic states. Abstract models

make test generation or model-checking possible in practice. We have proposed and experimented in [BBJM09] an approach of test generation from abstract models, that computes in finite time a Symbolic Labelled Transition System (SLTS) of all the behaviors of a model (with possibly an infinite concrete state space). However, it appeared that the computation time of the abstraction could be very expensive. We had replaced a problem of search time in a state graph with a problem of proof time. Indeed, computing an abstraction is performed by proving enabledness and reachability conditions on symbolic states [BPS05].

This short paper illustrates on an example our contribution [JSBM09] to reduce this proof time problem, by means of a proof free syntactic abstraction function. It works by suppressing some state variables of a model. When there are domain abstractions on the remaining state variables, a semantic abstraction that requires proof obligation checking is also performed. But it applies to a model that has been syntactically simplified.

2 Electrical System Example

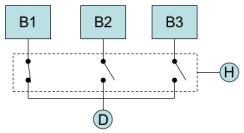

Fig. 1. Electrical System

A device D is powered by one of three batteries B_1, B_2, B_3 as shown in Fig. 1. A switch connects a battery B_i to the device D. A clock H periodically sends a signal that causes a commutation of the switches, i.e. a change of the battery that powers D. The system satisfies the three following requirements:

- Req_1: only one switch is closed at a time (i.e. there is no short-circuit),
- Req_2: there is always one switch closed, connected to a working battery,
- Req_3: a signal from the clock always changes the switch that is closed.

If a failure occurs to the battery that is powering D, the system triggers an exceptional commutation to satisfy Req_2. We assume that there are never more than two batteries down at the same time. When two batteries are down, Req_3 is relaxed and the clock signal leaves unchanged the switch that is closed.

This system is modeled by means of three variables H, Sw and Bat. $H \in \{tic, tac\}$ models the clock: tic means asking for a commutation and tac that the commutation has occurred. Sw models the switches: $Sw = i$ indicates that the switch i is closed while the others are opened. This modelling makes that requirements Req_1 and Req_2 necessarily hold. $Bat \in 1..3 \rightarrow \{ok, ko\}$ models the batteries, with ko meaning that a battery is down. The invariant I expresses the

assumption that at least one battery is not down by stating that $Bat(Sw) = ok$:

$I \mathrel{\widehat{=}} H \in \{tic, tac\} \wedge Sw \in 1..3 \wedge (Bat \in 1..3 \rightarrow \{ok, ko\}) \wedge Bat(Sw) = ok.$

The initial state is defined by $Init$ in Fig. 2. The behavior of the system is described by four events, modeled in Fig. 2 with the primitive forms of substitutions: `Tic` sends a commutation command, `Com` performs a commutation, `Fail` simulates the failure of a battery, and `Rep` simulates the replacement of a battery.

$$
\begin{array}{lll}
Init & \widehat{=} & H,\ Bat,\ Sw\ :=\ tac,\ \{1 \mapsto ok,\ 2 \mapsto ok,\ 3 \mapsto ok\}, 1 \\
Tic & \widehat{=} & H = tac \Rightarrow H := tic \\
Com & \widehat{=} & \mathrm{card}(Bat \rhd \{ok\}) > 1 \wedge H = tic \Rightarrow \\
 & & \quad @ns.(ns \in 1..3 \wedge Bat(ns) = ok \wedge ns \neq Sw \Rightarrow H, Sw := tac, ns) \\
Fail & \widehat{=} & \mathrm{card}(Bat \rhd \{ok\}) > 1 \Rightarrow \\
 & & \quad @nb.(nb \in 1..3 \wedge nb \in \mathrm{dom}(Bat \rhd \{ok\}) \Rightarrow \\
 & & \quad\quad nb = Sw \Rightarrow \\
 & & \quad\quad\quad @ns.(ns \in 1..3 \wedge ns \neq Sw \wedge Bat(ns) = ok \Rightarrow \\
 & & \quad\quad\quad\quad Sw, Bat := ns, Bat <+ \{nb \mapsto ko\}) \\
 & & \quad\quad []nb \neq Sw \Rightarrow Bat := Bat <+ \{nb \mapsto ko\})) \\
Rep & \widehat{=} & @nb.(nb \in 1..3 \wedge nb \in \mathrm{dom}(Bat \rhd \{ko\}) \Rightarrow Bat := Bat <+ \{nb \mapsto ok\})
\end{array}
$$

Fig. 2. B Specification of the Electrical System

3 Syntactic Abstraction

We consider abstractions obtained by observing only a subset of variables, defined as being relevant variables. This set is built as a fixpoint, starting with chosen variables from the property to test, and growing by addition of the variables required for computing the values assigned to the relevant variables.

For example, to test the electrical system in the particular cases where two batteries are down, observing the variable Bat is sufficient. In [JSBM09] we define a set of transformation rules that produce a simplified model A. We prove that A is, by construction, refined by the source model M, so that it is sufficient to verify safety properties on A for them to hold on M. It is also easier to compute test cases from A than from M.

The electrical system is transformed as shown in Fig. 3 for the set of observed variables $\{Bat\}$. It is a correct B event system. The initialization only assigns the observed variable. Its value is the same as in the source model. The event `Tic` is abstracted by `skip` because its guard and its action do not refer to the observed variable. The guard of the events `Com` and `Fail`, that are in the shape of $p(\mathtt{Bat}) \wedge p'(\mathtt{H})$, are transformed in the shape of $p(\mathtt{Bat})$ because the approximation of a proposition $p(x)$ is `true`, when x is a set of non observed variables. The bound variables are considered as observed variables. The action of an event (such as `Com` for example) becomes `skip` if it only assigns non observed variables. For the `Fail` event, we only keep the assignment of the variable `Bat`. Finally, the event `Rep` is unchanged because its guard and its action only assigns the variable `Bat` and depends on the value of the bound variable `nb`.

$$
\begin{array}{l}
Init \ \widehat{=} \ Bat \ := \ \{1 \mapsto ok, \ 2 \mapsto ok, \ 3 \mapsto ok\} \\
Tic \ \widehat{=} \ skip \\
Com \ \widehat{=} \ \mathrm{card}(Bat \rhd \{ok\}) > 1 \Rightarrow @ns.(ns \in 1..3 \wedge Bat(ns) = ok \Rightarrow skip) \\
Fail \ \widehat{=} \ \mathrm{card}(Bat \rhd \{ok\}) > 1 \Rightarrow \\
\qquad @nb.(nb \in 1..3 \wedge nb \in \mathrm{dom}(Bat \rhd \{ok\}) \Rightarrow Bat := Bat <+ \{nb \mapsto ko\}) \\
Rep \ \widehat{=} \ @nb.(nb \in 1..3 \wedge nb \in \mathrm{dom}(Bat \rhd \{ko\}) \Rightarrow Bat := Bat <+ \{nb \mapsto ok\})
\end{array}
$$

Fig. 3. B Syntactically Abstracted Specification of the Electrical System

4 Abstraction Process

In [BBJM09] we have introduced a test generation method based on a semantic abstraction of a B model (see Fig. 4/Process A). The abstraction is computed as an SLTS according to a test purpose. The idea is to observe the state variables that are modified by the operations activated by the test purpose. The domain of the observed variables can be abstracted into a few subdomains. For example, a natural integer n can be abstracted into subdomains $n = 0$ and $n > 0$.

The two main drawbacks of this process are its time cost and the proportion of proof obligations (POs) not automatically proved. Indeed, the semantic abstraction is based on a theorem proving process [BC00]. Each unproved PO adds a transition to the SLTS that is possibly unfeasible. Hence we propose to use a syntactic abstraction in addition to the semantic one. In Fig. 4/Process B, we describe a complete abstraction process in which we combine a syntactic abstraction that eliminates some variables (see Sec. 3), with a semantic abstraction computed by *GeneSyst* [BPS05] that projects the domain of the observed variables onto abstract domains.

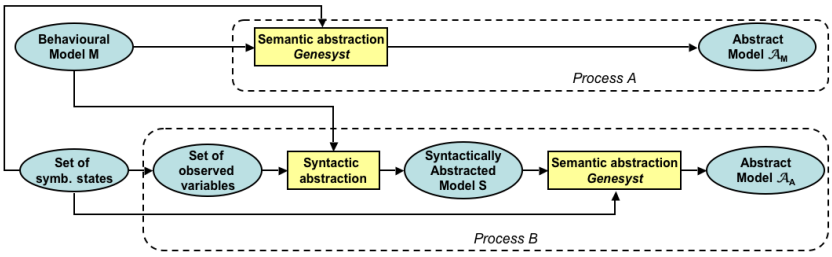

Fig. 4. Abstraction Process

5 Conclusion, Related Works and Further works

We have illustrated a method for abstracting an event system by elimination of some state variables. The abstraction is refined by the source model. It is useful for verifying properties and generating tests. The main advantage of our method is that it first performs syntactic transformations, which reduces the number of

POs generated and facilitates the proof of the remaining POs. This results in a gain of computation time. We believe that the bigger the ratio of the number of state variables to the number of observed variables is, the bigger the gain is. This conjecture needs to be confirmed by experiments on industrial size applications.

Many other works define model abstraction methods to verify properties. The methods of [GS97,BLO98,CU98] use theorem proving to compute the abstract model, which is defined over boolean variables that correspond to a set of *a priori* fixed predicates. In contrast, our method firstly introduces a syntactical abstraction computation from a set of observed variables, and further abstracts it by theorem proving. [CABN97] also performs a syntactic transformation, but requires the use of a constraint solver during a model checking process.

References

[BBJM09] F. Bouquet, P.-C. Bué, J. Julliand, and P.-A. Masson. Test generation based on abstraction and dynamic selection criteria. Research Report RR2009-02, Laboratoire d'Informatique de l'Université de Franche Comté, September 2009.

[BC00] D. Bert and F. Cave. Construction of Finite Labelled Transition Systems from B Abstract Systems. In W. Grieskamp, T. Santen, and B. Stoddart, editors, *Integrated Formal Methods*, volume 1945 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[BJK+05] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, editors. *Model-Based Testing of Reactive Systems*, volume 3472 of *LNCS*. 2005.

[BLO98] S. Bensalem, Y. Lakhnech, and S. Owre. Computing abstractions of infinite state systems compositionally and automatically. In *CAV'98*, volume 1427 of *LNCS*. Springer, 1998.

[BPS05] D. Bert, M.-L. Potet, and N. Stouls. GeneSyst: a Tool to Reason about Behavioral Aspects of B Event Specifications. In *ZB'05*, volume 3455 of *LNCS*, 2005.

[CABN97] W. Chan, R. Anderson, P. Beame, and D. Notkin. Combining constraint solving and symbolic model checking for a class of systems with non-linear constraints. In *CAV'97*, volume 1254 of *LNCS*. Springer, 1997.

[CU98] M.A. Colon and T.E. Uribe. Generating fnite-state abstractions of reactive systems using decision procedures. In *CAV'98*, volume 1427 of *LNCS*, 1998.

[GS97] S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In *CAV'97*, volume 1254 of *LNCS*, 1997.

[JSBM09] J. Julliand, N. Stouls, P.-C. Bué, and P.-A. Masson. B model abstraction combining syntactic and semantics methods. Research Report RR2009-04, LIFC - Laboratoire d'Informatique de l'Université de Franche Comté, November 2009. 15 pages.

[LB08] M. Leuschel and M. Butler. ProB: An automated analysis toolset for the B method. *Software Tools for Technology Transfer*, 10(2):185–203, 2008.

[UL06] M. Utting and B. Legeard. *Practical Model-Based Testing - A tools approach*. Elsevier Science, 2006.