Science of Computer Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Proof assisted bounded and unbounded symbolic model checking of software and system models

Sebastian Krings *, Michael Leuschel

Institut für Informatik, Universität Düsseldorf, Universitätsstr. 1, D-40225 Düsseldorf, Germany

ARTICLE INFO

Article history:
Received 7 October 2016
Received in revised form 26 August 2017
Accepted 28 August 2017
Available online xxxx

Keywords:
B-method
Event-B
Software model
Proof
Symbolic model checking

ABSTRACT

We have implemented various symbolic model checking algorithms, such as BMC, k-Induction and IC3 for B, Event-B and other modeling languages. The high-level nature of software models accounts for complicated constraints arising in these symbolic analysis techniques. In this article we suggest using static information stemming from proof obligations to simplify occurring constraints. We show how to include proof information in the aforementioned algorithms. Using different benchmarks we compare explicit state to symbolic model checking as well as techniques with and without proof assistance. In particular for models with large branching factor, e.g., due to complicated data values being manipulated, the symbolic techniques fare much better than explicit state model checking. The inclusion of proof information results in further performance improvements.

1. Introduction and motivation

Model checking is one of the key techniques used in formal software development. Two variants are currently in use: explicit state model checking and symbolic model checking. In explicit state model checking, every state is computed, the invariant is verified and discovered successor states are queued to be analyzed themselves. Symbolic model checking on the other hand tries to represent the state space and possible paths through it by predicates representing multiple states at once. Instead of stepwise exploration of the state space graph, the model checking problem is encoded as a formula and given to a constraint solver.[1]

So far, existing model checkers for B and Event-B such as ProB [3,4], Eboc [5], pyB [6] or TLC [7] (via [8]) rely on explicit state model checking. Furthermore, aside of ProB, they often only support parts of the B and Event-B languages or do not offer sufficient constraint solving capabilities to handle B efficiently and automatically.

ProB features some symbolic techniques for error detection [9] and test-case generation [10], but not full-blown symbolic model checking. This is mostly due to the high-level nature of B and Event-B. Both the usage of higher-order constructs and the underlying non-determinism accounts for complicated constraints during symbolic model checking. Some complexity can be coped with by relying on SMT solvers [11,12] such as Z3 [13]. Another alternative is to translate to SAT solvers as done in [14] using Kodkod [15].

* Corresponding author.
E-mail addresses: krings@cs.uni-duesseldorf.de (S. Krings), leuschel@cs.uni-duesseldorf.de (M. Leuschel).

[1] BDD-style model checking [1] is also called symbolic model checking. In recent work [2] ProB has been integrated with LTSMin for such kind of model checking.

http://dx.doi.org/10.1016/j.scico.2017.08.013
0167-6423/

This article is the extended version of the conference paper at ABZ 2016 [16], where we have implemented different symbolic model checking algorithms for B and then studied various ways to use proof information to optimize them. The proof information is used to strengthen constraints and reduce the counterexample search space.

For this article we extend our former work [16] in different aspects:

- We have added a formalization of an alternate bounded model checking algorithm (BMC*) in Section 3.2. It is an adaptation of a test-case generation algorithm from [10], improved for using proof information and specialized for bounded model checking.
- We experimented with lazily adding the loop-freeness constraints in the k-Induction algorithm. See Algorithm 3.
- We give a more detailed explanation of how IC3 works on B and Event-B models, including considering abstraction techniques.
- Several models have been added to the empirical evaluation. In particular, we include large models and case studies.
- Old and added models and benchmarks are more thoroughly discussed, including size and characteristics of the different models.
- In addition to comparing symbolic and explicit state model checking we also compare different solver backends.
- The applicability of our technique to state based formal methods other than B is discussed in Section 7. In particular, we apply ProB to a selection of models written in TLA$^+$ and Z.

All techniques used in this article have been implemented both for classical B and Event-B, TLA$^+$ and Z. The implementation is done inside ProB's model checking kernel and reuses the other parts of ProB: the end user can rely on the same user interface, including counterexample presentation and visualization.

Even though the presented algorithms work fully symbolic, they produce concrete counterexamples. In consequence, user interaction happens on the concrete rather than the symbolic level. Additionally, the symbolic representation of a model as well as all predicates discussed in this article can be accessed programmatically through ProB's Prolog APIs.

We will mainly use B and Event-B in our empirical evaluation, as ProB handles those natively. For the sake of brevity, we will only talk about Event-B events in the following sections instead of distinguishing events and operations.

Starting from a discussion of the proof obligations we rely upon in Section 2, we will introduce the model checking algorithms BMC, BMC*, k-Induction and IC3 in Sections 3, 3.2, 4, 5. For each of them, we will show how to include information regarding proven invariant preservation of events into the occurring constraints. Following, in Section 6, we will empirically compare symbolic model checking to explicit state model checking and model checking with and without proof assistance. Applicability to formalisms other than B is outlined in Section 7. Related work is given in Section 8, while our own roadmap and future work is discussed in Section 9. Overall conclusions will be presented in Section 10.

2. Proof obligations used

When using the B method to develop a software or system, one often alternates between different phases. Among those are writing and adapting the specification, manual and automated proof efforts as well as model checking.

These steps usually influence one another: an error detected by model checking is resolved by changing the specification. This again leads to new proof obligations being generated and discharged if possible. Yet, the different steps are only loosely coupled regarding tool support. Model checkers often do not incorporate proof information; provers have no knowledge about model checking results. In this article, we partially bridge the gap by integrating proof information from previous proof attempts into symbolic model checking algorithms.

For Event-B, the information stems from discharged proof obligations exported from Rodin [17]. In particular, we export and use the proof obligations concerned with invariant preservation of single events as defined in [18]:

Definition 1 *(Invariant preservation proof obligation).* For a state vector s, consisting of constants c and variable v, let $Axm(s)$ denote the conjunction of axioms over the constants. In addition, let $Inv(s) = \bigwedge Inv_i(s)$ denote the conjunction of invariants over both constants and variables. For each event e with before-after-predicate $BA_e(s, s')$ connecting variables v and constants c in the state vector s to their successors c and v' in state vector s', the invariant preservation proof obligations for each conjunct of the invariant are

$$Axm(s), Inv(s), BA_e(s, s') \vdash Inv_i(s').$$

If an invariant preservation proof obligation has been discharged, Inv_i is guaranteed to hold after the execution of e. This fact will later be used to strengthen the constraints occurring in the symbolic model checking algorithms.

There are many other proof obligations used to ensure the correctness of Event-B models. The default proof obligations are defined in [18]. Additionally, Rodin plugins can create custom proof obligations [17]. This is done, for instance, to ensure the validity of composition and decomposition of models [19]. So far, we do not take into account other proof obligations than invariant preservation, but plan to do so in future.

Please note that our symbolic model checking algorithms do not create additional proof obligations. We only rely on the fact that certain properties have been proven upfront. Creating custom proof obligations during model checking, e.g., to overcome timeouts by collecting additional knowledge, will be part of our future work.

Please cite this article in press as: S. Krings, M. Leuschel, Proof assisted bounded and unbounded symbolic model checking of software and system models, Sci. Comput. Program. (2017), http://dx.doi.org/10.1016/j.scico.2017.08.013