Science of Computer Programming 126 (2016) 94–110

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Parameterised three-valued model checking

Nils Timm *, Stefan Gruner

Department of Computer Science, University of Pretoria, South Africa

ARTICLE INFO

Article history:
Received 20 March 2015
Received in revised form 14 January 2016
Accepted 28 January 2016
Available online 1 February 2016

Keywords:
Three-valued logic
Three-valued abstraction
Parameterisation
Model checking
Counterexample-guided abstraction refinement

ABSTRACT

Three-valued abstraction is an established technique in software model checking. It proceeds by generating a state space model over the values true, false and unknown, where the latter value is used to represent the loss of information due to abstraction. Temporal logic properties can then be evaluated on such models. In case of an unknown result, the abstraction is iteratively refined until a definite result can be obtained. In this paper, we present and extend our work on parameterised three-valued model checking (PMC). In our parameterised three-valued models, unknown parts can be either associated with the constant value unknown or with expressions over boolean parameters. Our parameterisation is an alternative way to state that the truth value of certain predicates or transitions is actually not known and that the checked property has to yield the same result under each possible parameter instantiation. A specific feature of our approach is that it allows for establishing logical connections between parameters: While unknown parts in pure three-valued models are never related to each other, our parameterisation approach enables to represent facts like 'a certain pair of transitions has unknown but complementary truth values', or 'the value of a predicate is unknown but remains unchanged along all states of a certain path'. We demonstrate that such facts can be automatically derived from the system to be verified and that covering these facts in an abstract model can be crucial for the success and the efficiency of checking temporal logic safety and liveness properties. Parameterisation enhances the precision of three-valued models without increasing their state space, but it leads to an exponential increase in time complexity, since any property of interest must be checked for each possible parameter instantiation. In this extended paper, we introduce a novel algorithm for direct parameterised three-valued model checking that straightly explores the parameterised state space and thus avoids to construct all instantiations explicitly. We present example verification tasks where the application of our direct algorithm considerably reduces the time effort of PMC.

1. Introduction

Predicate abstraction [1] is an established technique for reducing the complexity of temporal logic model checking. It proceeds by generating a state space model of the software system to be analysed. In this model, concrete states of the system are mapped to abstract states over a finite set of predicates, and admissible executions of the system are represented

* Corresponding author.
E-mail addresses: ntimm@cs.up.ac.za (N. Timm), sg@cs.up.ac.za (S. Gruner).

http://dx.doi.org/10.1016/j.scico.2016.01.006
0167-6423/

by sequences of transitions between states. Traditional predicate abstraction techniques are based on a boolean domain for predicates and on an over-approximation of the concrete state space. Thus, only universal properties are preserved under this form of abstraction. If checking a universal property for an abstract model yields *false*, it cannot be concluded that the original system violates this property as well. In this case, model checking additionally returns an *abstract counterexample* — a path in the model that refutes the property. In order to gain certainty about whether this counterexample is spurious or corresponds to a real path, it has to be simulated on the original system. The simulation of counterexamples involves a partial exploration of the concrete state space, and thus, can be exceedingly costly. Spurious counterexamples are typically ruled out via *counterexample-guided abstraction refinement* (CEGAR) [2]: Further predicates over the variables of the system are iteratively added to the model until a level of abstraction is reached where the property can be either definitely proved or a real counterexample can be found. The application of CEGAR does, however, not guarantee that eventually a model can be constructed that is both precise enough for a definite outcome and small enough to be manageable with the available computational resources.

More recent approaches [3–5] to abstraction refinement for model checking are based on a domain for predicates with the truth values *true, false* and *unknown*. Corresponding three-valued models with the additional value *unknown* enable to explicitly model the loss of information due to abstraction. In comparison to boolean abstractions, the three-valued approach is capable of preserving universal *and* existential properties. Hence, all definite results in three-valued model checking can be directly transferred to the original system. Only an *unknown* result necessitates iterative refinement. In the latter case, an *unconfirmed counterexample* – a potential error path in the model with *unknown* transitions and predicates – is returned. Unconfirmed counterexamples directly hint at necessary refinement steps. Thus, the costly simulation of counterexamples on the original system is not required in the three-valued setting. Model checking three-valued abstractions can be conducted at the same cost as checking boolean abstractions, but it additionally comes along with the aforementioned advantages.

Continuative work in this field has shown that the precision of model checking three-valued abstractions can be increased by the concept of *generalised model checking* (GMC) [6]. While standard three-valued model checking (3MC) [3–5] is based on a special *three-valued* semantics that enables the direct evaluation of temporal logic formulae on three-valued models, the idea of GMC is to construct *all* boolean concretisations of a three-valued model. Then classical two-valued model checking is applied to each concretisation and it is checked whether the results are consistent, i.e. whether either all results are *true* or whether all are *false*. In case of consistency, the result can be transferred to the original system. GMC generally yields more definite results than 3MC. Hence, the application of GMC instead of 3MC can reduce the number of necessary refinement iterations in abstraction-based verification. However, the 3MC problem is PSPACE-complete [3], whereas the GMC problem is even EXP-complete [6]: Number and size of concretisations can be exponential in the size of the three-valued model. Thus, GMC is rather of theoretical than of practical inter- est. Most existing three-valued abstraction-based verification frameworks, such as [5,7,8], rely on standard 3MC and try to compensate the lack of precision with additional refinement steps.

In this paper, we present and extend our work on *parameterised three-valued model checking* (PMC) [9] which is a hybrid of three-valued and generalised model checking. Predicates and transitions in parameterised three-valued models can be either associated with the values *true, false* or *unknown* – or with expressions over boolean parameters. Parameterisation is an alternative way to state that the truth value of certain predicates or transitions is actually not known and that the checked property has to yield the same result under each possible parameter instantiation. PMC is thus generally conducted via evaluating a temporal logic formula under all parameter instantiations and checking whether the results are consistent. Parameterisation particularly allows to establish logical connections between *unknowns* in the abstract model: While *unknown* parts in 3MC and GMC are never related to each other, our parameterisation approach enables to represent facts like 'a certain pair of transitions has unknown but complementary truth values', or 'the value of a predicate is unknown but remains unchanged along all states of a certain path'. Such facts can be automatically derived from the software system to be verified and covering these facts in an abstract model can be crucial for the success and the efficiency of checking temporal logic properties. We developed an automatic verification framework for concurrent systems based on parameterised three-valued model checking: Starting with pure three-valued abstraction, in each iteration either classical refinement or parameterisation of *unknown* parts is applied until a definite result in verification can be obtained. The decisions for refinement or parameterisation are automatically made based on unconfirmed counterexamples. Such a combination of classical refinement and parameterisation in abstraction-based model checking is highly suited for obtaining the necessary precision for a definite result in verification while keeping the state space small. Our approach so far works for checking safety properties of the form '*always p*' and liveness properties of the form '*always eventually p*', where *p* is an atomic predicate.

Parameterisation does not increase the space complexity but it leads to an exponential growth in time complexity. A definite outcome in verification requires a model checking run for each possible parameter instantiation and a check whether all single results are consistent. In our previous work [9], we performed PMC by constructing all instantiations and considering them separately. Here, we introduce a *direct* algorithm for parameterised three-valued model checking. Our novel algorithm straightforwardly explores the parameterised state space. Its worst-case time complexity is still exponential in the number of parameters – each state might be explored exponentially often. However, we demonstrate that for several application examples our direct approach enables us to keep the number of exploration steps small and thus to achieve a significantly enhanced runtime performance. Our work includes a proven theorem on the correctness of our algorithm.

The remainder of this paper is organised as follows. Section 2 provides the background on three-valued model checking. Section 3 introduces our new concept parameterised three-valued model checking. In Section 4 we demonstrate how PMC