Encodings of Bounded LTL Model Checking in Effectively Propositional Logic

Juan Antonio Navarro-Pérez Andrei Voronkov

* The University of Manchester
School of Computer Science
{navarroj,voronkov}@cs.manchester.ac.uk

Abstract

In this short paper we describe an encoding of LTL bounded model checking within the Bernays-Schönfinkel fragment of first-order logic. This fragment allows a natural and succinct representation of both a software/hardware system and a property to verify. More interestingly, we note that modules in systems can be directly encoded without a preliminary flattening stage which, in standard approaches, could yield an exponential blowup in the size of the result.

1 Introduction

Model checking is a technique used to verify that a hardware or software component follows some formally specified expected behaviour. One often builds a description of the system, often modelled as a finite state machine, and use a temporal logic to specify properties that the system should satisfy. Advances in the theory and practise of model checking, such as the introduction of symbolic (McMillan, 1993) and bounded model checking (Biere et al., 1999), made this technique useful to find simple bugs in industrial settings (Copty et al., 2001).

The idea of bounded model checking (BMC) is to search for counterexamples (violations of the property to verify) within executions of the system that have a bounded length. Typically, a propositional formula is created and a decision procedure for propositional logic is used to find models which represent bugs in the system. We observe, however, that BMC problems can also be compactly encoded within the Bernays-Schönfinkel class of formulae, which are non-propositional but with a finite Herbrand universe. Moreover, this fragment corresponds to the effectively propositional category (EPR) of the CASC competition, giving a motivation to find a new source of benchmarks.

The EPR encoding gives a more succinct and natural description of both the system and the property to verify. It is not needed, for example, to replicate copies of the temporal formula for every step of the execution trace where it has to be checked. Furthermore, system described in a modular way can be directly encoded without needing to expand module definitions beforehand. A prover could potentially use this information to better organise its search.

On the other hand, our encoding may also turn out to be useful for propositional, SAT-based, approaches to bounded model checking. Indeed, it preserves the structure of the original bounded model checking problem in the obtained effectively propositional formula and reduces the problem of finding an optimised propositional encoding to the problem of finding an optimised propositional instantiation of the EPR description.

2 Linear temporal logic encoding

We consider the standard definition of the semantics of linear temporal logic (LTL) over infinite paths, but we also extend the definition to deal with paths of finite lengths: for a finite path $\pi = s_0 \dots s_k$, we say that π *is a model of an LTL formula* ϕ *a state* s_i, with $0 \leq i \leq k$ and denoted as usual by $\pi \models_i \phi$, if every infinite extension of π is also a model of ϕ at s_i.

Moreover, we also make use of the notion of a k*-path* which is either a finite path of the form $\pi = s_0 \dots s_k$, or an infinite path with a loop of the form $\pi = s_0 \dots s_{l-1}(s_l \dots s_k)^\omega$. Paths of this kind are important since they are finitely described and enough to check satisfiability w.r.t. Kripke structures (i.e. described by initial states and a transition relation).

Theorem 1 (Biere et al. (1999))**.** *A linear temporal logic formula ϕ is satisfiable in a Kripke structure M if and only if there is a k-path π in M with $\pi \models \phi$.*

Using this idea, we propose an encoding of the model checking problem where satisfying paths are encoded as finite models of a quantifier-free predicate formula. Given a Kripke structure M an LTL formula ϕ and a bound $k \geq 0$, the encoding consists of three sets of constraints: $|[\,M\,]| \cup |[\,\phi\,]| \cup |[\,k\,]|$. The first set $|[\,M\,]|$ contains a symbolic representation of the system to be verified. It asserts $I(\mathsf{s}_0)$, the initial conditions are satisfied at the state s_0, and that $\mathsf{trans}(X, Y) \rightarrow T(X, Y)$, if the predicate $\mathsf{trans}(\mathsf{s}_i, \mathsf{s}_j)$ is true then the pair (s_i, s_j) must be in the transition relation of the system.

The encoding $|[\,\phi\,]|$ of the temporal formula ϕ, which we assume is given in negation normal form, is performed in a similar way to the renaming approach in clause normal form translations. First, a variable X is added to atoms in ϕ to represent time (e.g. if $p(\mathsf{s}_i)$ then p is true at the state s_i). New names, which we denote by $\Theta_\gamma(X)$, are introduced to represent each temporal subformula; and a set of constraints is used to attach its intended meaning to those names. For example, the *weak until* temporal operator is encoded as:

$$\begin{array}{ll} \psi\mathbf{W}\phi\text{:} & \mathsf{weak}_{\psi,\phi}(X) \rightarrow \Theta_\phi(X) \vee \mathsf{xweak}_{\psi,\phi}(X) \qquad \mathsf{xweak}_{\psi,\phi}(X) \rightarrow \Theta_\psi(X) \\ & \mathsf{xweak}_{\psi,\phi}(X) \wedge \mathsf{trans}(X,Y) \rightarrow \mathsf{weak}_{\psi,\phi}(Y) \quad \mathsf{last}(X) \wedge \mathsf{xweak}_{\psi,\phi}(X) \rightarrow \mathsf{hasloop} \end{array}$$

Similarly, we can also give definitions for the *next* ($\mathbf{X}\phi$) and *eventually* ($\mathbf{F}\phi$) operators. Other standard temporal connectives —such as *until*, *release* and *globally*— can be introduced as abbreviations of existing connectives. The fact $\Theta_\phi(\mathsf{s}_0)$ is also added to make the temporal formula true at the initial state.

Finally the set $|[\,k\,]|$ has a number of constraints to make $\mathsf{trans}(\mathsf{s}_i, \mathsf{s}_{i+1})$ hold for every $i \leq 0 < k$, and $\mathsf{last}(\mathsf{s}_k)$ for the last state in the bounded length. A constraint $\mathsf{hasloop} \rightarrow \mathsf{trans}(\mathsf{s}_k, \mathsf{s}_0) \vee \cdots \vee \mathsf{trans}(\mathsf{s}_k, \mathsf{s}_k)$ is also included to handle the case of infinite paths with a loop, effectively making a transition from the last to some previous state. We can then prove that this encoding indeed captures the semantics of model checking.

Theorem 2. *Let M be a Kripke structure, ϕ an LTL formula and $k \geq 0$. (1) ϕ is satisfiable in M iff $|[\,M, \phi, k\,]|$ is satisfiable for some $k \geq 0$. (2) The size of $|[\,M\,]|$ and $|[\,\phi\,]|$ is linear w.r.t. its input; the size of $|[\,k\,]|$ is linear w.r.t. k.*

3 Encoding of the system description

In the discussion of the previous section we generally assumed that the system (a structure M) was already symbolically described as a pair of formulae $I(X)$ and $T(X,Y)$. But an advantage of the predicate encoding is that important features for component development, such as the ability to describe systems in a modular way, can be directly represented. There is no need, for example, to create and instantiate all modules of a system description before doing the actual encoding. A phase which, in the propositional case, can represent an exponential blowup in the size of the resulting formula.

The system description is fragmented in a number of modules, each specifying how a section of the system works. We now introduce, for each variable in a module, a new predicate symbol of the form $\mathsf{module\_var}(\bar{I}, X)$. The name of the symbol is prefixed with the module name to avoid clashes between modules. Since, moreover, several instances of a single module can be created, a number of variables $\bar{I}$ serve to distinguish among such instances. The last argument X again represents a state in the execution. The definition of a module is then replaced by a predicate formula that states the relations that should hold among its variables. Other features, such as the creation and instantiation of submodules as well as passing module references as parameters, can also be easily implemented. In particular, for example, all of the main features of a system description language such as SMV, can be linearly encoded in a predicate formula using this ideas.

4 Conclusions and future work

We briefly described how instances of bounded model checking can be encoded in the Bernays-Schönfinkel class of formulae. Moreover, the representation is very compact and succinct, linear in the size of the input system, the temporal property and the bound k. Compare with the propositional case where the encoded temporal formula is of size $O(nk)$, with n the size of the formula, and the encoded system of exponential size if the original description was highly modular.

We are also currently working in the development of a tool that —taking as input a smv description, an LTL formula, and a bound k— produces an EPR formula in the tptp format suitable for use with effectively propositional and first-order reasoners. Directions for future work include the extension to more general forms of temporal logics (such as μTL), the inclusion of more features to describe systems (such as arrays and arithmetic) and the application of similar encoding techniques to other suitable application domains. The tool in development and a more detailed version of this ideas is available at `http://www.cs.man.ac.uk/~navarroj/eprbmc`.

References

A. Biere, A. Cimatti, E.M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proc., Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99)*, volume 1579 of *LNCS*, 1999.

F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M.Y. Vardi. Benefits of bounded model checking at an industrial setting. In *CAV'01*, volume 2102 of *LNCS*, pages 436–453. Springer, 2001.

K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.