A comparison of SAT-based and SMT-based bounded model checking methods for ECTL*

Extended Abstract ⋆ ⋆⋆

Agnieszka M. Zbrzezny[1] and Andrzej Zbrzezny[1]

IMCS, Jan Długosz University. Al. Armii Krajowej 13/15, 42-200 Częstochowa, Poland.
{agnieszka.zbrzezny,a.zbrzezny}@ajd.czest.pl

Abstract. In this paper we present a comparison of the SAT-based bounded model checking (BMC) and SMT-based bounded model checking methods for ECTL* properties of a parallel composition of transition systems. In the both methods we use the parallel composition (of the transition systems) based on the interleaved semantics. Moreover, the both methods use the same bounded semantics of ECTL* formulae, the compatible encodings of the transition systems and the compatible translations of ECTL* formulae. For the SAT-based BMC we have used the PicoSAT solver and for the SAT-based BMC we have used the Z3 solver. We have implemented the both methods and made some preliminary experimental results which shows that generally the SAT-based method is superior to the SMT-based method. However, in some cases the SMT-method overcomes the SAT-based method.

1 Introduction

The problem of model checking [10] is to check automatically whether a structure $\mathcal{M}$ defines a model for a modal (temporal, epistemic, etc.) formula α. The practical applicability of model checking is strongly limited by the state explosion problem, which means that the number of model states grows exponentially in the size of the system representation. To avoid this problem a number of state reduction techniques and symbolic model checking approaches have been developed, among others, [8, 9, 19, 20].

The SAT-based bounded model checking (BMC) is one of the symbolic model checking technique designed for finding witnesses for existential properties or counterexamples for universal properties. Its main idea is to consider a model reduced to a specific depth. The first BMC method was proposed in [5], and it was designed for linear time properties. Next in [21] the method has been extended to handle branching time properties.

The SMT problem [7] is a generalisation of the SAT problem, where Boolean variables are replaced by predicates from various background theories, such as linear, real,

⋆ Partly supported by National Science Centre under the grant No. 2011/01/B/ST6/05317.

⋆⋆ The study is co-funded by the European Union, European Social Fund. Project PO KL "Information technologies: Research and their interdisciplinary applications", Agreement UDA-POKL.04.01.01-00-051/10-00.

and integer arithmetic. SMT generalises SAT by adding equality reasoning, arithmetic, fixed-size bit-vectors, arrays, quantifiers, and other useful first-order theories.

The SMT-based bounded model checking is quite new technique. It was using to verifying Embedded ANSI-C Software [12], C++ Programs [22], Multi-threaded Software [11], Fixed-Point Digital Controllers [3], timed automata [17], real-time systems [24], LTL Specifications with Integer Constraints [2] and many others.

In order to use the bounded model checking method we need to define a translation from a given temporal logic to the satisfiability modulo theories problem (in short: to SMT). As far as we know, no such translation was given in the literature. However, several translations to SAT from ECTL* and its sublogics were proposed. The first translation from LTL to SAT was introduced in [4] and another ones in [18] and [6]. The first translation from ECTL to SAT was introduced in [21] and then it was substantially improved in [25]. The first correct translation from ECTL* to SAT was introduced in [23] and then it was substantially improved in [26]. The translation from ECTL* to SMT that we use in this paper strictly follows the translation from ECTL* to SAT introduced in [26].

The rest of the paper is organised as follows. In the next section we give some remarks about the syntax and (both the bounded and unbounded) semantics of ECTL*. In Section 3 we give some remarks about our translation from ECTL* to SMT. Preliminary experimental results and some conclusions are presented in Section 4.

2 Syntax and Semantics of ECTL*

In this section we briefly recall the syntax and semantics of the logic ECTL*. The Existential Computation Tree Logic ECTL* is a restriction of a propositional branching-time temporal logic CTL* introduced by Emerson and Halpern in [15] as a specification language for finite-state systems. The restriction consists in using only existential path quantifiers and allowing the negation to be applied to propositional variables only (instead of to arbitrary formulae). For more thorough description one can see [1].

2.1 Syntax of ECTL*

The language of ECTL* consists of two types of formulae: state formulae (interpreted at states) and path formulae (interpreted along paths). The syntax of ECTL* state formulae over the set AP of atomic propositions is defined by the following grammar:

$$\alpha ::= \mathbf{true} \mid \mathbf{false} \mid p \mid \neg p \mid \alpha_1 \wedge \alpha_2 \mid \alpha_1 \vee \alpha_2 \mid \mathbf{E}\varphi$$

where $p \in AP$, α_1, α_2 and α are state formulae, and φ is a path formula. The syntax of ECTL* path formulae over the set AP of atomic propositions is defined by the following grammar:

$$\varphi ::= \alpha \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \mathbf{X}\varphi_1 \mid \varphi_1 \mathbf{U} \varphi_2 \mid \varphi_1 \mathbf{R} \varphi_2$$

where α is a state formula and φ, φ_1 and φ_2 are path formulae. In practice, many interesting temporal properties are formulated by using temporal operators $\mathbf{F}$ (eventually) and $\mathbf{G}$ (always) defined as follows:

$$\mathbf{F}\psi \stackrel{df}{=} \mathbf{true}\,\mathbf{U}\,\psi \qquad\qquad \mathbf{G}\psi \stackrel{df}{=} \mathbf{false}\,\mathbf{R}\,\psi.$$

2.2 Semantics of ECTL*

The semantics of ECTL* formulae is determined with respect to a *transition system* (also called a *model*).

Definition 1. *A* transition system *is a tuple* $\mathcal{M} = (S, Act, \longrightarrow, s^0, AP, L)$*, where* S *is a nonempty finite set of states,* Act *is a set of actions,* $s^0 \in S$ *is the initial state,* $\longrightarrow \subseteq S \times Act \times S$ *is a transition relation,* AP *is a set of atomic propositions, and* $L : S \to 2^{AP}$ *is a* labelling function *that assigns to each state a set of atomic propositions that are assumed to be true at that state.*

Definition 2. *An* ECTL* *state formula* α *is valid in* $\mathcal{M}$*, denoted by* $\mathcal{M} \models \alpha$*, iff* $s^0 \models \alpha$*, i.e.,* α *holds at the initial state of* $\mathcal{M}$*.*

2.3 Bounded Semantics of ECTL*

The *bounded semantics* of ECTL* is defined in [26]. This is done in order to define the *bounded model checking problem* for ECTL* and to translate it into the satisfiability problem. For more thorough description one can see [26].

From now on we assume that models are finite, i.e. the sets S, Act and AP are finite. To define the bounded semantics one needs to represent infinite paths in a model in a special way. To this aim, the notions of k*-paths* and *loops* are defined.

Definition 3. *Let* $\mathcal{M}$ *be a model,* $k \in \mathbb{N}$*, and let* $l \in \mathbb{N}$ *such that* $0 \leqslant l \leqslant k$*. A* k-path *is a pair* (π, l)*, also denoted by* π_l*, where* π *is a finite sequence* $\pi = (s_0, \ldots, s_k)$ *of states such that* $s_j \longrightarrow s_{j+1}$ *for each* $0 \leqslant j < k$*. A* k*-path* π_l *is a* loop *if* $l < k$ *and* $\pi(k) = \pi(l)$*.*

If a k-path π_l is a loop it represents the infinite path of the form uv^ω, where $u = (\pi(0), \ldots, \pi(l))$ and $v = (\pi(l+1), \ldots, \pi(k))$. We denote this unique path by $\varrho(\pi_l)$. Note that for each $j \in \mathbb{N}$, $\varrho(\pi_l)^{l+j} = \varrho(\pi_l)^{k+j}$.

Let s be a state and π_l be a k-path. For a state formula α over AP, the notation $\mathcal{M}, s \models_k \alpha$ means that α k-holds at the state s in the model $\mathcal{M}$. Similarly, for a path formula φ over AP, the notation $\mathcal{M}, \pi_l^m \models_k \varphi$, where $0 \leqslant m \leqslant k$, means that φ k-holds along the suffix $\big(\pi(m), \ldots, \pi(k)\big)$ of π_l.

Lemma 1. *Let* $\mathcal{M}$ *be a model. For every* ECTL* *path formula* φ*, every* k*-path* π_l *in* $\mathcal{M}$*, and every* $0 \leqslant m \leqslant k$*, if* $\mathcal{M}, \pi_l^m \models_k \varphi$*, then*

1. *if* π_l *is not a loop, then for each path* $\rho \in \mathcal{M}$ *such that* $\rho[..k] = \pi$ *it holds* $\mathcal{M}, \rho^m \models \varphi$*.*
2. *if* π_l *is a loop, then* $\mathcal{M}, \varrho(\pi_l)^m \models \varphi$*,*

Theorem 1. *Let* $\mathcal{M}$ *be a model and* α *be an* ECTL* *state formula. Then* $\mathcal{M}, s^0 \models \alpha$ *iff for some* $k \in \mathbb{N}$*,* $\mathcal{M}, s^0 \models_k \alpha$*.*

3 Translation to SMT

We have implemented a translation to SMT strictly following the translation to SAT given in [26]. In our translation to SMT states, loops and actions are represented by natural variables. Since $\mathcal{M}$ is a parallel composition of a finite number n of finite transition system, every state of $\mathcal{M}$ can be encoded as a natural number vector of the length n. Thus, each state of $\mathcal{M}$ can be represented by a valuation of a vector (called a *symbolic state*) of different individual variables called *individual state variables*. Moreover, every action of $\mathcal{M}$ can be represented by a valuation of an individual variable, and the designated positions l of the k-paths used in the translation can be also be represented by valuations of individual variables. Furthermore, k-paths can be represented as vectors of symbolic states.

The details of the translation to SMT will be provided in the full version of the present paper.

3.1 Bounded Model Checking of ECTL* properties

Now let us recall the the BMC method of verifying a given ECTL* state formula α. Let $\mathcal{I}(\mathbf{w}_{0,0})$ be a be a quantifier-free first-order formula representing the initial state, $[M]_k^{F_k(\alpha)}$ be a quantifier-free first-order formula representing transition relation and $\langle\alpha\rangle_k^{[0,0,F_k(\alpha)]}$ be a quantifier-free first-order formula that is the translation of the formula α. In order to verify the formula α one has to check the satisfiability of the following conjunction:

$$[M]_k^{\alpha} := \mathcal{I}(\mathbf{w}_{0,0}) \wedge [M]_k^{F_k(\alpha)} \wedge \langle\alpha\rangle_k^{[0,0,F_k(\alpha)]}$$

starting with $k = 0$. If for a given k the formula $[M]_k^{\alpha}$ is not satisfiable, then k is increased and the resulting formula is to be checked by a SMT-solver again. The method described relies on the following theorem.

Theorem 2. *Let $\mathcal{M}$ be a model and α be an* ECTL* *state formula. Then for every $k \in \mathbb{N}$, $\mathcal{M}, s^0 \models_k \alpha$ if, and only if, the quantifier-free first-order formula $[M]_k^{\alpha}$ is satisfiable.*

4 Experimental Results

In this section we present a comparison of a performance evaluation of two methods: SMT-based BMC and SAT-based BMC for dining philosophers problem.

An evaluation of both BMC algorithms is given by means of the running time and the memory used. In order to compare the translation to SMT with the translation to SAT we have implemented both the algorithms as standalone programs written in the programming language C++. In our SAT-BMC technique we use the state of the art SAT-solver PicoSAT (`http://fmv.jku.at/picosat/`) and in SMT-BMC technique we use the state of the art SMT-solver Z3 [13] (`http://z3.codeplex.com/`).

Our experiments were performed on a computer equipped with I7-3770 processor, 32 GB of RAM, and the operating system Arch Linux with the kernel 3.15.3. As the

benchmark we used the well-known dining philosophers problem [14, 16]. We have modelled this problem by means of communicating finite automata. The system consists of n automata each of which models a philosopher, together with n automata each of which models a fork, together with one automaton which models the lackey. The latter automaton is used to coordinate the philosophers' access to the dining-room. In fact, this automaton ensures that no deadlock is possible. The global system is obtained as the parallel composition of the components, which are shown in Figure 1.

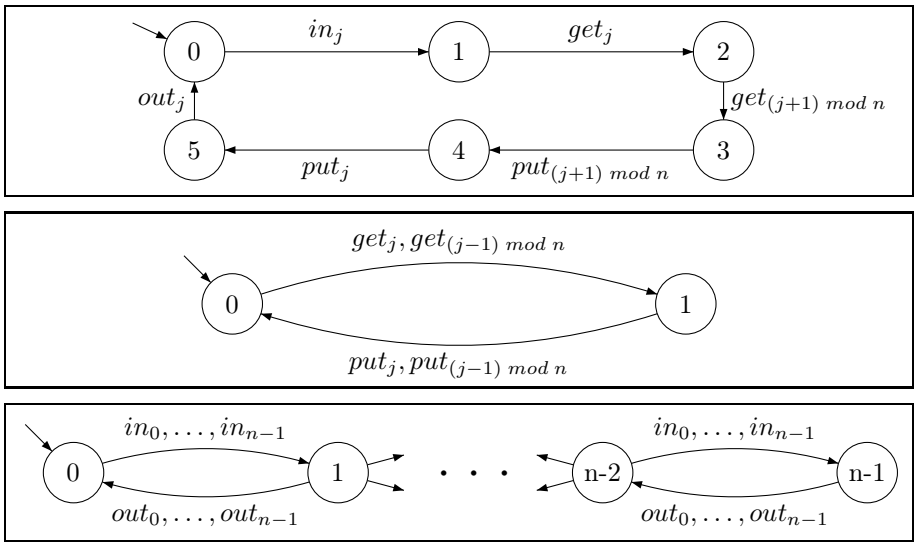


Fig. 1. The automata for the j-th Philosopher, the j-th Fork and the Lackey

Let $AP = \{p_j^i \mid 0 \leqslant j < n, 0 \leqslant i \leqslant 5\}$ and assume that the variable p_j^i is true only at the i-th state of the j-th philosopher. We have tested three ECTL* formulae over AP in order to compare the experimental results for the translation to SAT with the experimental results for the translation to SMT. All the tested formulae are valid in the considered model for every $n \geq 2$.

The first formula

$$\varphi_1 = \bigwedge_{j=0}^{n-1} \mathbf{E}\big(\mathbf{F}\big(p_j^3\big)\big).$$

expresses the following property: *For each philosopher there exists a path on which this philosopher eventually gets his left and right forks.*

In Figures 2(a) and 2(b) we present a comparison of total time usage and total memory usage for the formulae φ_1.

The second formula

$$\varphi_2 = \mathbf{EGF} \bigwedge_{j=0}^{\lfloor \frac{n}{2} \rfloor} \big(p^3_{2\cdot j}\big).$$

expresses the following property: *There exists a path on which always it is the case that eventually every other philosopher is eating.*

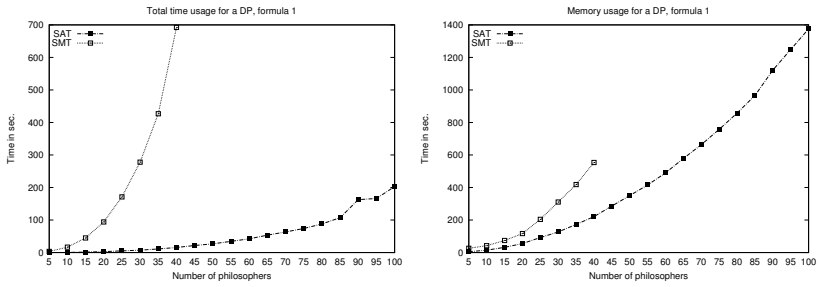
In Figures 3(a) and 3(b) we present a comparison of total time usage and total memory usage for the formulae φ_2.

The third formula

$$\varphi_3 = \mathbf{E} \bigwedge_{j=0}^{n-1} \big(\mathbf{F}\big(p^3_j\big)\big).$$

expresses the following property: *There exists a path on which every philosopher eventually gets his left and right forks.*

In Figures 4(a) and 4(b) we present a comparison of total time usage and total memory usage for the formulae φ_3.

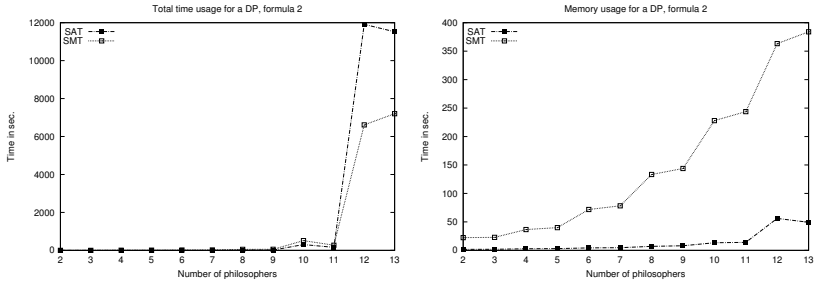


(a) (b)

Fig. 2. A comparison of total time usage and total memory usage for the formulae φ_1.

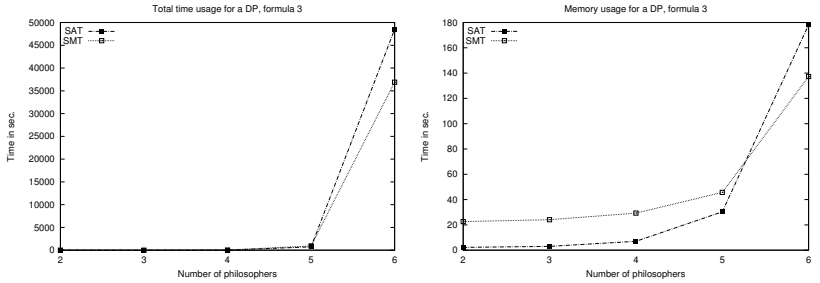
Our preliminary experimental results suggest that generally the SAT-based method is superior to the SMT-based method. However, in some cases the SMT-method overcomes the SAT-based method. An observation of experimental results leads to the conclusion that the SAT-based BMC for ECTL* uses less time and memory comparing to the SMT-based BMC for ECTL*. In particular, using the SAT-based BMC it was possible to verify the formula φ_1 for 100 philosophers, whereas using the SMT-based BMC it was possible to verify the formula φ_1 for 40 philosophers only. However, it was possible to verify the formulae φ_2 and φ_3 for the same numbers of philosophers for the both methods. Moreover, the experimental results for formulae φ_2 and φ_3 lead to the conclusion that the SMT-based BMC method uses less time than the SAT-based BMC method when the number of philosophers increases.

We should stress that the implementation of the SMT-based BMC we used for performing the experiments is our first implementation that uses SMT solvers. Therefore,



(a) (b)

Fig. 3. A comparison of total time usage and total memory usage for the formulae φ_2.



(a) (b)

Fig. 4. A comparison of total time usage and total memory usage for the formulae φ_3.

we hope to improve the implementation in the near future by taking many advantages of possibilities (of SMT-solvers) that we did not use so far.

References

1. C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008.
2. Marcello M. Bersani, Luca Cavallaro, Achille Frigeri, Matteo Pradella, and Matteo Rossi. SMT-based verification of LTL specification with integer constraints and its application to runtime checking of service substitutability. In *SEFM*, pages 244–254, 2010.
3. Iury Bessa, Renato B. Abreu, Joao Edgar Chaves Filho, and Lucas Cordeiro. SMT-based bounded model checking of fixed-point digital controllers. *CoRR*, abs/1403.5172, 2014.
4. A. Biere, A. Cimatti, E. Clarke, M.Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *Proc. of the ACM/IEEE Design Automation Conference (DAC'99)*, pages 317–320, 1999.
5. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proc. of the 5th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *LNCS*, pages 193–207. Springer-Verlag, 1999.

6. A. Biere, K. Heljanko, T. A. Junttila, T. Latvala, and V. Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, 2(5), 2006.
7. Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.
8. R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transaction on Computers*, 35(8):677–691, 1986.
9. E. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
10. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
11. Lucas Cordeiro and Bernd Fischer. Verifying multi-threaded software using SMT-based context-bounded model checking. In *ICSE*, pages 331–340, 2011.
12. Lucas Cordeiro, Bernd Fischer, and João Marques-Silva. SMT-based bounded model checking for embedded ANSI-C software. *IEEE Trans. Software Eng.*, 38(4):957–974, 2012.
13. Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *TACAS*, pages 337–340, 2008.
14. E.W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Inf.*, 1:115–138, 1971.
15. E. Allen Emerson and Joseph Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *J. Comput. Syst. Sci.*, 30(1):1–24, 1985.
16. C.A.R. Hoare. *Communicating sequential processes*. Prentice Hall, 1985.
17. Roland Kindermann, Tommi A. Junttila, and Ilkka Niemelä. Beyond lassos: Complete SMT-based bounded model checking for timed automata. In *FMOODS/FORTE*, pages 84–100, 2012.
18. T. Latvala, A. Biere, K. Heljanko, and T. A. Junttila. Simple bounded LTL model checking. In A. J. Hu and A. K. Martin, editors, *FMCAD*, volume 3312 of *Lecture Notes in Computer Science*, pages 186–200. Springer, 2004.
19. A. Lomuscio, W. Penczek, and Hongyang Qu. Partial order reduction for model checking interleaved multi-agent systems. In *Proceedings of the 9th International Conference on Autonomous Agents and Multi-Agent systems (AAMAS'2010).*, pages 659–666. IFAAMAS Press, 2010.
20. K. McMillan. *Symbolic model checking: An approach to the state explosion problem*. Kluwer Academic Publishers, 1993.
21. W. Penczek, B. Woźna, and A. Zbrzezny. Bounded model checking for the universal fragment of CTL. *Fundamenta Informaticae*, 51(1-2):135–156, 2002.
22. Mikhail Ramalho, Mauro Freitas, Felipe Sousa, Hendrio Marques, Lucas Cordeiro, and Bernd Fischer. SMT-based bounded model checking of C++ programs. In *ECBS*, pages 147–156, 2013.
23. B. Woźna. ACTL* properties and bounded model checking. *Fundamenta Informaticae*, 63(1):65–87, 2004.
24. Liang Xu. SMT-based bounded model checking for real-time systems (short paper). In *QSIC*, pages 120–125, 2008.
25. A. Zbrzezny. Improving the translation from ECTL to SAT. *Fundamenta Informaticae*, 85(1-4):513–531, 2008.
26. A. Zbrzezny. A new translation from ECTL* to SAT. *Fundamenta Informaticae*, 120(3-4):377–397, 2012.