Int J Softw Tools Technol Transfer (2005) / Digital Object Identifier (DOI) 10.1007/s10009-004-0182-5

Computational challenges in bounded model checking*

Edmund Clarke[1], **Daniel Kroening**[2], **Joël Ouaknine**[3], **Ofer Strichman**[4]

[1] Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
e-mail: emc@cs.cmu.edu
[2] Department of Computer Science, ETH Zürich, Switzerland
e-mail: daniel.kroening@inf.ethz.ch
[3] Oxford University Computing Laboratory, Oxford, UK
e-mail: joel@comlab.ox.ac.uk
[4] Information Systems Engineering, Faculty of Industrial Engineering, Technion, Israel
e-mail: ofers@ie.technion.ac.il

Published online: 15 February 2005 –

Abstract. We describe several observations regarding the completeness and the complexity of bounded model checking and propose techniques to solve some of the associated computational challenges. We begin by defining the *completeness threshold* ($\mathcal{CT}$) problem: for every finite model M and an LTL property φ, there exists a number $\mathcal{CT}$ such that if there is no counterexample to φ in M of length $\mathcal{CT}$ or less, then $M \models \varphi$. Finding this number, if it is sufficiently small, offers a practical method for making bounded model checking complete. We describe how to compute an overapproximation to $\mathcal{CT}$ for a general LTL property using Büchi automata, following the Vardi–Wolper LTL model checking framework. This computation is based on finding the *initialized diameter* and *initialized recurrence-diameter* (the longest loop-free path from an initial state) of the product automaton. We show a method for finding a recurrence diameter with a formula of size $O(k \log k)$ (or $O(k(\log k)^2)$ in practice), where k is the attempted depth, which is an improvement compared to the previously known method that requires a formula of size in $O(k^2)$. Based on the value of $\mathcal{CT}$, we prove that the complexity of standard SAT-based BMC is doubly exponential and that, consequently, there is a complexity gap of an exponent between this procedure and standard LTL model checking. We discuss ways to bridge this gap.

Keywords: Bonded-Model-checking – Complexity – Completeness-Threshold

* This article merges and extends two previously published conference proceeding articles [9, 16]. It was supported by the Semiconductor Research Corporation (SRC) under contract no. 99-TJ-684, the National Science Foundation (NSF) under grant nos. CCR-9803774 and CCR-0121547, the Office of Naval Research (ONR) and the Naval Research Laboratory (NRL) under contract no. N00014-01-1-0796, and the Army Research Office (ARO) under contract no. DAAD19-01-1-0485.

1 Introduction

Bounded model checking (BMC) [5, 6] is a method for finding logical errors, or proving their absence, in finite-state transition systems. It is widely regarded as a complementary technique to symbolic BDD-based model checking (see [6] for a survey of experiments with BMC conducted in industry). Given a finite transition system M, an LTL formula φ, and a natural number k, a BMC procedure decides whether there exists a computation in M of length k or less that violates φ. SAT-based BMC is performed by generating a propositional formula, which is satisfiable if and only if such a path exists. BMC is conducted in an iterative process, where k is incremented until (i) an error is found, (ii) the problem becomes intractable due to the complexity of solving the corresponding SAT instance, or (iii) k reaches some precomputed threshold, which indicates that M satisfies φ. We call this threshold the *completeness threshold* and denote it by $\mathcal{CT}$. $\mathcal{CT}$ is any natural number that satisfies

$$M \models_{\mathcal{CT}} \varphi \rightarrow M \models \varphi\,,$$

where $M \models_{\mathcal{CT}} \varphi$ denotes that no computation of M of length $\mathcal{CT}$ or less violates φ. Clearly, if $M \models \varphi$, then the smallest $\mathcal{CT}$ is equal to 0, and otherwise it is equal to the length of the shortest counterexample. This implies that finding the smallest $\mathcal{CT}$ is at least as hard as checking whether $M \models \varphi$. Consequently, we concentrate on computing an overapproximation to the smallest $\mathcal{CT}$ based on some graph-theoretic properties of M (such as the *diameter* of the graph representing it), disregarding the labeling of the states, and an automaton representation of $\neg\varphi$. In particular, we consider as one abstract model all models for which these properties are the same as M's. Thus, this computation corresponds to finding the length

of the longest shortest counterexample to φ in any one of these models, assuming at least one of them violates φ.[1] Thus, when we say *the* value of $\mathcal{CT}$ in the rest of the paper, we refer to the value corresponding to this abstraction.

The value of $\mathcal{CT}$ depends on the model M, the property φ (both the structure of φ and the propositional atoms it refers to), and the exact scheme used for translating the model and property into a propositional formula. The original translation scheme of [5], which we will soon describe, is based on a k-steps *syntactic expansion* of the formula. With this translation, the value of $\mathcal{CT}$ was until now known only for unnested properties such as $\mathbf{G}p$ formulas [5] and $\mathbf{F}p$ formulas [16]. Computing $\mathcal{CT}$ for general LTL formulas has so far been an open problem.

In order to solve this problem we suggest to use instead a *semantic* translation scheme, based on Büchi automata, as was suggested in [10].[2] The translation is straightforward because it follows very naturally the Vardi–Wolper LTL model checking algorithm, i.e., checking for emptiness of the product of the model M and the Büchi automaton $B_{\neg\varphi}$ representing the negation of the property ϕ. Nonemptiness of $M \times B_{\neg\varphi}$, i.e., the existence of a counterexample, is proven by exhibiting a path from an initial state to a *fair loop*. We will describe in more detail this algorithm in Sect. 3.1. Deriving from this product a propositional formula $\Omega_\varphi(k)$ that is satisfiable if and only if there exists such a path of length k or less is easy: one simply needs to conjoin the k-unwinding of the product automaton with a condition for detecting a fair loop. We will give more details about this alternative BMC translation in Sect. 3.2. For now let us just mention that due to the fact that $\Omega_\varphi(k)$ has the same structure regardless of the property φ, it is easy to compute $\mathcal{CT}$ based on simple graph-theoretic properties of the product $M \times B_{\neg\varphi}$. Furthermore, the semantic translation leads to smaller CNF formulas compared to the syntactic translation. There are two reasons for this:

1. The semantic translation benefits from the existing algorithms for constructing compact representations of LTL formulas as Büchi automata. Such optimizations are hard to achieve with the syntactic translation. For example, the syntactic translation for $\mathbf{FF}p$ results in a larger propositional formula compared to the formula generated for $\mathbf{F}p$, although these are two equivalent formulas. Existing algorithms [23] generate in this case a Büchi automaton that corresponds to the second formula in both cases.
2. The number of variables in the formula resulting from the semantic translation is linear in k, compared to a quadratic ratio in the syntactic translation.

Other than the technique for efficient computation of recurrence diameter in Sect. 4, this paper is mainly an exposition of observations about bounded model checking[3] rather than a presentation of new techniques. In particular, we show how to compute $\mathcal{CT}$ based on the semantic translation; prove the advantages of this translation with respect to the size of the resulting formula as mentioned above, both theoretically and through experiments; and, finally, we discuss the question of the complexity of BMC. In Sect. 5 we show that, due to the fact that $\mathcal{CT}$ can be exponential in the number of state variables, solving the corresponding SAT instance is a doubly exponential procedure in the number of state variables and the size of the formula. This implies that there is a complexity gap of an exponent between the standard BMC technique and LTL model checking. We suggest a SAT-based procedure that closes this gap while sacrificing some of the main advantages of SAT. So far our experiments show that our procedure is not better in practice than the standard SAT-based BMC.

Computing $\mathcal{CT}$ is of course not the only way to make BMC complete. Other techniques exist, although always by performing additional steps. Sheeran et al. suggest in [22] to check for convergence by testing if the property is k-inductive (an extension of the traditional induction, which checks if the fact that the property holds in k steps implies that it holds in the $k+1$ step). Although this method can prove correctness for a larger set of cases compared to standard induction, in the worst case k can be as long as $\mathcal{CT}$. Further, it only works for properties that can be reduced to an invariant (a $\mathbf{G}p$ property). McMillan suggested several methods for fixpoint detection in BMC, including a method based on interpolants [18] and a method based on computing the whole set of reachable states [17]. In all of these techniques in the worst case convergence is achieved only when reaching $\mathcal{CT}$, but with additional work for detecting convergence. In these cases simply running BMC up to the bound $\mathcal{CT}$ is easier. McMillan observed, however, that in all the examples he experimented with convergence was reached earlier than $\mathcal{CT}$. We leave to future work the question of whether knowing the value of $\mathcal{CT}$ can help in practice compared to the above-mentioned alternatives. Here we focus only on the general computational problem from a theoretical perspective.

2 A translation scheme and its completeness threshold

2.1 Preliminaries

A *Kripke structure* M is a quadruple $M = (S, I, T, L)$ such that (i) S is the set of states, where states are defined by valuations to a set of Boolean variables (atomic

[1] If this assumption does not hold, e.g., when φ is a tautology, the smallest threshold is of course 0.

[2] The authors of [10] suggested this translation in the context of bounded model checking of infinite systems, without examining the implications of this translation on completeness and complexity as we do here.

[3] Some of these observations can be considered as folk theorems in the BMC community, although none of them, to the best of our knowledge, has been published before.

propositions) At; (ii) $I \subseteq S$ is the set of initial states; (iii) $T \subseteq S \times S$ is the transition relation; and (iv) $L\colon S \to 2^{(At)}$ is the labeling function. Labeling is a way to attach observations to the system: for a state $s \in S$ the set $L(s)$ contains exactly those atomic propositions that hold in s. We write $p(s)$ to denote $p \in L(s)$. The initial state I and the transition relation T are given as functions in terms of At. This kind of representation, frequently called *functional form*, can be exponentially more succinct compared to an explicit representation of the states. This fact is important for establishing the complexity of the semantic translation, as we do in Sect. 3.2.

Propositional *linear temporal logic* (LTL) formulas are defined recursively: Boolean variables are in LTL; then, if $\varphi_1, \varphi_2 \in$ LTL, so are $\mathbf{F}\varphi_1$ (Future), $\mathbf{G}\varphi_1$ (Globally), $\mathbf{X}\varphi_1$ (neXt), $\varphi_1 \mathbf{U} \varphi_2$ (φ_1 Until φ_2), $\varphi_1 \mathbf{W} \varphi_2$ (φ_1 Waiting-for φ_2), $\varphi_1 \vee \varphi_2$, and $\neg\varphi_1$.

2.2 Bounded model checking of LTL properties

Given an LTL property φ, a Kripke structure M, and a bound k, BMC is performed by generating and solving a propositional formula $\Omega_\varphi(k)$: $[\![M]\!]_k \wedge [\![\neg\varphi]\!]_k$, where $[\![M]\!]_k$ represents the reachable states up to step k and $[\![\neg\varphi]\!]_k$ specifies which paths of length k violate φ. The satisfiability of this conjunction implies the existence of a counterexample to φ. For example, for simple invariant properties of the form $\mathbf{G}p$ the BMC formula is

$$\Omega_\varphi(k) \doteq I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} \neg p(s_i)\,,$$

where the left two conjuncts represent $[\![M]\!]_k$ and the right conjunct represents $[\![\neg\varphi]\!]_k$.

There are several known methods for generating $[\![\neg\varphi]\!]_k$ [5, 12]. In the rest of this section we consider the original BMC translation scheme of Biere et al. [5] given below. This translation distinguishes between finite and infinite paths (for the latter it formulates a path ending with a loop). For a given property, it generates both translations and concatenates them with a disjunction.

Here is the translation for several temporal operators:

$$ {}_l[\![P]\!]^i_k := P(S_i)\,, \tag{1}$$

$$ {}_l[\![\mathbf{F}\varphi]\!]^i_k := \bigvee_{j=\min(i,l)}^{k} {}_l[\![\varphi]\!]^j_k\,, \tag{2}$$

$$ {}_l[\![\mathbf{G}\varphi]\!]^i_k := \bigwedge_{j=\min(i,l)}^{k} {}_l[\![\varphi]\!]^j_k\,, \tag{3}$$

$$ {}_l[\![\varphi_1 \mathbf{U} \varphi_2]\!]^i_k := \bigvee_{j=i}^{k} \left({}_l[\![\varphi_2]\!]^j_k \wedge \bigwedge_{n=i}^{j-1} {}_l[\![\varphi_1]\!]^n_k \right) \vee \bigvee_{j=l}^{i-1} \left({}_l[\![\varphi_2]\!]^j_k \wedge \bigwedge_{n=i}^{k} {}_l[\![\varphi_1]\!]^n_k \wedge \bigwedge_{n=l}^{j-1} {}_l[\![\varphi_1]\!]^n_k \right). \tag{4}$$

Each of the expressions of the form ${}_l[\![\cdot]\!]^i_k$ is a variable in the translation. This translation results in a propositional formula with one variable for each subformula of φ, in each position up to k, for each connection point of the loop up to k. This implies that the number of variables in the resulting formula is quadratic in k, as we will prove in Proposition 1. Figure 1 illustrates the idea behind this translation for the **U** operator that appears in Eq. (4). It shows the two possibilities for $p\mathbf{U}q$ to hold in step i, for a given loop to step l.

Constructing a propositional formula that captures finite paths is simpler and requires only a linear number of variables, as there is no loop to consider: each subformula at each location is represented by a new variable.

Finally, in order to capture all possible loops, we generate $\bigvee_{l=0}^{k}({}_lL_k \wedge {}_l[\![\varphi]\!]^0_k)$, where ${}_lL_k \doteq (s_l = s_k)$, i.e., an expression that is true if and only if there exists a back loop from state s_k to state s_l. As mentioned above, the total number of variables introduced by this translation is quadratic in k. More accurately:

Proposition 1. *The syntactic translation results in a propositional formula with $O(k \cdot |v| + (k+1)^2 \cdot |\varphi|)$ variables, where v is the set of variables defining the states of M and $|\varphi|$ is the length of φ.*

Proof. Recall the structure of the formula $\Omega_\varphi(k)$: $[\![M]\!]_k \wedge [\![\neg\varphi]\!]_k$. The subformula $[\![M]\!]_k$ adds $O(k \cdot |v|)$ variables. The subformula $[\![\neg\varphi]\!]_k$ adds, according to the recursive translation scheme, not more than $(k+1)^2 \cdot |\varphi|$ variables, because each expression of the form ${}_l[\![\varphi]\!]^i_k$ is a new variable, and both indices i and l range over $0 \ldots k$. Further, each subformula is unfolded separately, hence leading to the result stated above. □

Note that this result refers to an arbitrary Boolean formula, hence additional variables are needed for transforming it to CNF.

We now investigate the size of the resulting formula. Let $C(M)$ denote the size of a circuit defining M. $[\![M]\!]_k$ is represented with a formula of size $k \cdot C(M)$. The subformula $[\![\neg\varphi]\!]_k$ is represented by a constraint that can be quadratic in k (in the case of the Until operator) for each of the $(k+1)^2 \cdot |\varphi|$ variables (see Proposition 1), as observed by Biere (A. Biere A, 2004, private communication). Thus we can claim:

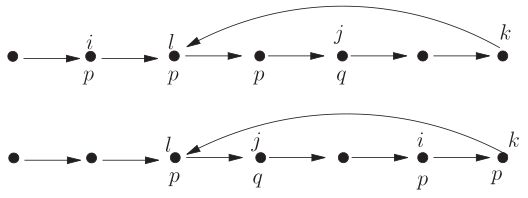


Fig. 1. Demonstrating the two cases in which the Until operator can hold in a path ending with a loop starting from step i. The two cases are captured by the top and bottom parts of Eq. (4), respectively

Proposition 2. *The syntactic translation results in a propositional formula of size $O(k \cdot C(M) + k^4 \cdot |\varphi|)$. If sharing is applied to the formulas of the Until operator, the formula size can be reduced to cubic instead of $O(k^4)$ (A. Biere, 2004, private communication).*

2.3 A completeness threshold for simple properties

There are two known results regarding the value of $\mathcal{CT}$, one for $\mathbf{G}p$ and one for $\mathbf{F}p$ formulas. Their exposition requires the following definitions.

Definition 1. *The* diameter *of a finite transition system M, denoted by $d(M)$, is the longest shortest path (defined by the number of its edges) between any two reachable states of M.*

The diameter problem can be reduced to the "all pair shortest path" problem, and therefore be solved in time polynomial in the size of the graph. In our case, however, the graph itself is exponential in the number of variables. Alternatively, one may use the formulation of this problem as satisfiability of a quantified Boolean formula (QBF), as suggested in [5], and later optimized in [3, 19].

Definition 2. *The* recurrence diameter *of a finite transition system M, denoted by $rd(M)$, is the longest loop-free path in M between any two reachable states.*

Finding the longest loop-free path between two states is NP-complete in the size of the graph. One way to solve it with SAT was suggested in [5]. The number of variables required by their method is quadratic in the length of the longest loop-free path. Hence, the SAT instance may have an exponential number of variables, and finding a solution to this instance is doubly exponential. In Sect. 4 we offer an alternative translation that requires only $O(k \log k)$ variables.

We denote by $d^I(M)$ and $rd^I(M)$ the *initialized* diameter and recurrence diameter, respectively, i.e., the length of the corresponding paths when they are required to start from an initial state.

For formulas of the form $\mathbf{F}p$ (i.e., counterexamples to $\mathbf{G}\neg p$ formulas), Biere et al. suggested in [5] that $\mathcal{CT}$ is less than or equal to $d(M)$ (it was later observed by several researchers independently that in fact $d^I(M)$ is sufficient). For formulas of the form $\mathbf{G}p$ formulas (counterexamples to $\mathbf{F}\neg p$ formulas), it was shown in [16] that $\mathcal{CT}$ is equal to $rd^I(M)$. Computing $\mathcal{CT}$ for general LTL formulas, as was mentioned in the introduction, has so far been an open problem.

These results can be further improved if we take into account the propositions themselves in the property and not just the temporal template of the formula. For example, $\mathcal{CT}$ for $\mathbf{F}p$ is not the initialized diameter of M; rather it is the initialized diameter of the portion of M that satisfies $\neg p$. This is known as the *predicated diameter* [21]. We leave the question of how to further tighten the bounds by considering the propositions for future work.

3 The semantic translation

The *semantic translation* that we present in this section is based on Vardi–Wolper's framework for model checking of LTL properties. We briefly describe this method in Sect. 3.1. In Sect. 3.2 we will show how to adapt it for computing $[\![\neg\varphi]\!]_k$ from LTL formulas. In Sects 3.3 and 3.4 we will show that this new translation is both more efficient (in terms of the size of the resulting formula) and solves the question of computing the value of $\mathcal{CT}$ for arbitrary LTL formulas as defined earlier.

3.1 LTL model checking with Büchi automata

Vardi and Wolper offered in [24] an automata-theoretic view of LTL model checking that we now briefly describe. A labeled Büchi automaton $M = \langle S, S_0, \delta, L, F\rangle$ is a 5-tuple where S is the set of states, $S_0 \subseteq S$ is a set of initial states, $\delta \subseteq (S \times S)$ is the transition relation, L is a labeling function mapping each state to a Boolean combination of the atomic propositions, and $F \subseteq S$ is the set of accepting states. The structure of M is similar to that of a finite-state automaton, but M is used for deciding acceptance of infinite words. Given an infinite word w, $w \in \mathcal{L}(M)$ if and only if the execution of w on M passes an infinite number of times through at least one of the states in F. In other words, if we denote by $inf(w)$ the set of states that appear infinitely often in the path of w on M, then $inf(w) \cap F \neq \emptyset$.

Every LTL formula φ can be translated into a Büchi automaton B_φ such that B_φ accepts exactly the words (paths) that satisfy φ. There are several known techniques to translate φ to B_φ [13].[4] We do not repeat the details of this construction; rather we present several examples in Fig. 2 of such translations.

LTL model checking can be done as follows: Given an LTL formula φ, construct $B_{\neg\varphi}$, a Büchi automaton that accepts exactly those paths that violate φ. Then, check whether $\Psi \doteq M \times B_{\neg\varphi}$ is empty. It is straightforward to see that $M \models \varphi$ if and only if Ψ is empty. Thus, LTL model checking is reduced to the question of Büchi automaton emptiness, i.e., proving that no word is accepted by the product automaton Ψ. In order to prove emptiness, one has to show that no computation of Ψ passes through an accepting state an infinite number of times. Consequently, finding a reachable loop in Ψ that contains an accepting state is necessary and sufficient for proving

[4] Most published techniques for this translation construct a *generalized* Büchi automaton, while in this article we use a standard Büchi automaton (the only difference being that the former allows multiple accepting sets). The translation from generalized to standard Büchi automaton multiplies the size of the automaton by up to a factor of $|\varphi|$. See [11, Sect. 9.2.2].

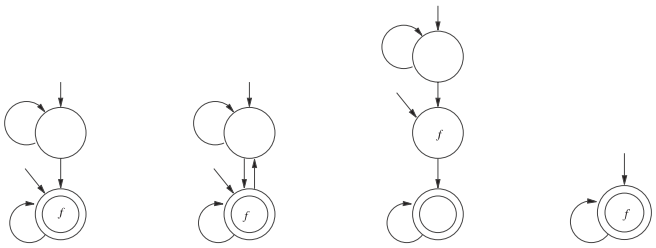


Fig. 2. Several LTL formulas and their corresponding Büchi automata. Accepting states are marked by *double circles*

that the relation $M \not\models \varphi$ holds. Such loops are called *fair loops*.

3.2 A semantic translation method

The fact that emptiness of Ψ is proven by finding a path to a fair loop gives us a straightforward adaptation of the LTL model checking procedure to a SAT-based BMC procedure. This can be done by searching for a witness to the property $\varphi' \doteq \mathbf{G}(\text{TRUE})$ under the fairness constraint $\bigvee_{F_i \in F} F_i$ [8] (that is, $\bigvee_{F_i \in F} F_i$ should be true infinitely often in this path). Thus, given Ψ and k, we can use the standard BMC translation for deriving $\Omega_\Psi(k)$, a SAT instance that represents all the paths of length k that satisfy φ'. Finding such a witness of length k or less is done in BMC by solving the propositional formula:

$$\Omega_\Psi(k) \doteq I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge \bigvee_{l=0}^{k-1} \left((s_l = s_k) \wedge \bigvee_{j=l}^{k} \bigvee_{F_i \in F} F_i(s_j) \right). \quad (5)$$

The rightmost conjunct in Eq. (5) constrains one of the states in F to be true in at least one of the states of the loop. In our case the fairness constraints refer to accepting states in the Büchi automaton. It can be worthwhile to represent the set of accepting states symbolically, as a predicate P, in which case we can replace the last disjunct with P.

Since the Büchi automaton used in this translation captures the semantics of the property rather than its syntactic structure, we call this method a *semantic translation* for BMC.

We continue by proving the two advantages of this translation: the efficiency of the translation and the ease of computing $\mathcal{CT}$.

3.3 The semantic translation is more efficient

The semantic translation has a clear advantage in terms of the number of variables in the resulting formula, as stated in the following proposition (compare to Proposition 1).

Proposition 3. *The semantic translation results in a propositional formula with $O(k \cdot (|v| + |p|))$ variables, where v and p are the sets of variables of M and φ, respectively.*

Proof. The transition relation of the Büchi automaton constructed from φ is defined by p variables. The propositional formula is constructed by unfolding k times the product ψ, hence it uses $O(k \cdot (|v| + |p|))$ variables. It also includes constraints for identifying a loop with a fair state, but these constraints only add clauses, not new variables. □

Not only does the semantic translation require less variables, it also results in a smaller formula, as we now prove. Let $C(\psi)$ be the size of a circuit representing ψ, and let $C(F)$ denote a size of a predicate, or a circuit, that represents all the accepting states in $B_{\neg\varphi}$. We now claim:

Proposition 4. *The semantic translation results in a propositional formula of size $O(k \cdot (C(\psi) + |v| + |p| + k \cdot C(F)))$.*

Proof. Unrolling the circuit k times results in a formula with size $O(k \cdot C(\psi))$. It remains to consider the constraints that we add for detecting fair loops (Eq. (5)). This requires a comparison of states for each l in the range $1 \ldots k$, and for each such comparison a constraint of size $(k-l) \cdot C(F)$ for forcing the loop to be fair. Each comparison requires $O(|v| + |p|)$ bitwise comparisons. So altogether the formula is of size $O(k \cdot (C(\psi) + |v| + |p| + k \cdot C(F)))$. □

The result in Proposition 4 can be further improved by sharing fairness constraints, as in [6], resulting in a translation linear in k.

As before, the analysis here refers to an arbitrary propositional formula, and transforming it to CNF requires the addition of auxiliary variables.

We conducted some experiments in order to check the difference between the translations. We conducted this experiment with NuSMV 2.1, which includes both an optimized syntactic translation [7] and a tool called LTL2SMV that translates LTL formulas to SMV files by describing the transition relation of the corresponding Büchi automata. Since generating the BMC formula requires also a model, and since we only wanted to compare the effect of the translation of the property, we generated a trivial model consisting of a single variable. Given the SMV model representing the Büchi automaton, we checked the property $\mathbf{F}(\text{FALSE})$ under possible fairness constraints, as prescribed by the accepting states of the Büchi automaton.

We experimented with large LTL formulas taken from the WRING benchmark suite [23]. Since some of the formulas are too large to fit in this paper, in Table 1 we only

Table 1. Number of temporal operators for each of the benchmark formulas

Formula	**G**	**X**	**U**	**F**
1	1		5	1
2		8	9	
3		5	9	
4	1	6	8	
5	1	4	7	1
6	7	1	2	2
7		8	1	4
8		70	54	
9	4	66	100	3
10		500	500	

show the number of temporal operators in each one of them.

The results for these formulas appear in Table 2. We checked formula 10 with two different bounds, which appear as Test 10 and Test 11 in the table. Timeout was set to 36 000 s. Run times shorter than a second were taken as 0.5 s for calculating the averages. Although we disregard the instances where the syntactic translation cannot generate formulas for when computing the average, it is rather safe to assume that it would take more time to solve them in comparison with the other instances.

As shown in the table, there are very large differences in the size of the resulting formulas, and hence also in the time it takes NuSMV to generate them. There is not a large difference in the time it takes ZCHAFF to solve the resulting CNF formulas, probably because they are all trivially satisfied when checked against the artificial one-variable model we generated. We do not present results of a real model here, first, because this would skew the comparison, as the differences in the translation are only in the LTL formula, not the model, and second, because all publicly available benchmark models that we are aware of have very short properties (typically not more than two or three nesting levels), unlike what is used in industry.

3.4 Computing $\mathcal{CT}$ for general LTL formulas

A major benefit of the semantic translation is that it implies directly an overapproximation of the value of $\mathcal{CT}$:

Theorem 1. *A completeness threshold for any LTL property φ when using Eq.* (5) *is* $\min(rd^I(\Psi)+1, d^I(\Psi)+d(\Psi))$.

Proof. (a) We first prove that $\mathcal{CT}$ is bounded by $d^I(\Psi)+d(\Psi)$. If $M \not\models \varphi$, then Ψ is not empty. The shortest wit-

Table 2. Comparing results for the Semantic and syntactic translations. Run time is given in seconds. The main bottleneck in the syntactic translation is the generation of the formulas, which are orders of magnitude larger in both the number of variables and clauses in comparison to the semantic translation

Test #	Method	K	# Variables	# Clauses	Formula Generation	Formula Solving (Zchaff)
1	Syn	100	375972	1126666	601	< 1
	Sem	100	13237	37416	< 1	< 1
2	Syn	50	400177	1199772	1821	1.08
	Sem	50	3676	10269	< 1	< 1
3	Syn	70	431787	1294322	25200	1.79
	Sem	70	25959	74901	< 1	< 1
4	Syn	25	65995	197281	10800	0.21
	Sem	25	9697	27620	< 1	< 1
5	Syn	65	586649	1758839	21620	2.2
	Sem	65	18582	53109	< 1	< 1
6	Syn	80	237500	711321	28802	1.74
	Sem	80	23012	65998	< 1	1.92
7	Syn	80	–	–	timeout	–
	Sem	80	21583	61373	< 1	< 1
8	Syn	25	128095	383876	606	< 1
	Sem	25	42702	122800	< 1	< 1
9	Syn	40	–	–	timeout	–
	Sem	40	161822	476036	< 1	< 1
10	Syn	25	–	–	timeout	–
	Sem	25	359057	1053870	< 1	< 1
11	Syn	5	–	–	timeout	–
	Sem	5	72577	210230	< 1	< 1
Avg	Syn		> 318025	> 953153	> 21222	> 1.14
	Sem		68354	199420	0.5	0.62

ness for the nonemptiness of Ψ is a path $s_0, \ldots, s_f, \ldots s_k$, where s_0 is an initial state, s_f is an accepting state, and $s_k = s_l$ for some $l \leq f$. The shortest path from s_0 to s_f is not longer than $d^I(\Psi)$, and the shortest path from s_f back to itself is not longer than $d(\Psi)$. (b) We now prove that $\mathcal{CT}$ is also bounded by $rd^I(\Psi)+1$ (the addition of 1 to the longest loop-free path is needed in order to detect a loop). Falsely assume that $M \not\models \varphi$ but all witnesses are of length longer than $rd^I(\Psi)+1$. Let $W : s_0, \ldots, s_f, \ldots s_k$ be the shortest such witness. By definition of $rd^I(\Psi)$, there exist at least two states, say, s_i and s_j, in this path that are equal (other than the states closing the loop, i.e., $s_i \neq s_k$). If $i, j < f$ or $i, j > f$, then this path can be shortened by taking the transition from s_i to s_{j+1} (assuming, without loss of generality, that $i < j$), which contradicts our assumption that W is the shortest witness. If $i < f < j$, then the path $W' : s_0, \ldots, s_f, \ldots, s_j$ is also a loop witnessing $M \not\models \varphi$, but shorter than W, which again contradicts our assumption. □

The left-hand side drawing below demonstrates a case in which $d^I(\Psi) + d(\Psi) > rd^I(\Psi) + 1$ ($d^I(\Psi) = d(\Psi) = rd^I(\Psi) = 3$), while the right-hand side drawing demonstrates the opposite case (in this case, $d^I(\Psi) = d(\Psi) = 1, rd^I(\Psi)+1 = 5$). These examples justify taking the minimum between the two values.

An interesting special case is invariant properties ($\mathbf{G}p$). The Büchi automaton for the negation of this property ($\mathbf{F}\neg p$) has a special structure (see third drawing in Fig. 2): for all M, any state satisfying $\neg p$ in the product machine Ψ leads to a fair loop. Thus, to prove emptiness, it is sufficient to search for a reachable state satisfying $\neg p$. A path to such a state cannot be longer than $d^I(\Psi)$. More formally:

Theorem 2. *A completeness threshold for* $\mathbf{F}p$ *formulas, where* p *is nontemporal, is* $d^I(\Psi)$.

Recall that we refer to witnesses; hence the theorem refers to counterexamples to $\mathbf{G}\neg p$ formulas.

4 Computing the recurrence diameter with sorting networks

Computing the recurrence diameter efficiently is essential for applying Theorem 1. The currently used technique [5] for computing the recurrence diameter compares all pairs of states. The size of the resulting formula is therefore quadratic in the length of the path k.

We propose the following alternative: first, generate an equation that represents the same set of states but in a sorted order; second, compare the neighbors in the sorted sequence. Since we have to generate the equation without any actual knowledge of the states, the sequence of comparisons performed must be the same for all possible states. This is known as the Bose–Nelson sorting problem. A circuit that solves this problem is called a *sorting network* (see Knuth [15] for a survey).

Ajtai et al. [1] show that sorting networks for n inputs can be built with size $O(n \log n)$. However, there is a very high constant (several thousands) hidden in this complexity result that makes it impractical for our purpose. We therefore use a variant of a *bitonic sorting network* as described by Batcher [2], which has an asymptotic size of $O(n(\log n)^2)$. Bitonic sorting networks have a recursive structure (Fig. 3). The inputs are split into two parts that are sorted independently and then merged. Figure 4 shows a simple sorting network for three input states.

Let $s_0 \ldots s_{n-1}$ denote the n states of the path. Using the sorting network, we obtain an ordered permutation of these states $s'_0 \geq s'_1 \ldots \geq s'_{n-1}$. It is obvious that a sequence of states contains two equal states if and only if its corresponding sorted sequence contains two equal neighboring states, or, formally:

$$\exists i : s'_i = s'_{i+1} \iff \exists l, j : l \neq j \wedge s_l = s_j. \tag{6}$$

Thus, we now only have to compare all neighbors in this sequence. This can be done with $n-1$ comparisons.

4.1 Ordering and swapping

All sorting networks require a *compare* and *swap* operation. Two elements a and b are compared and, if $a > b$, swapped. We implement the ordering operator by computing the last carry bit of the sum $a + (-b)$. Let a denote a bit vector of length β, and let a_i, $0 \leq i < \beta$ denote the ith component of a. Let $\overline{b}$ denote the inverted vector b.

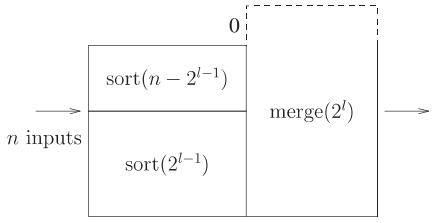


Fig. 3. Block diagram of a bitonic sorting network with n inputs and $l = \lceil \log_2 n \rceil$. If n is not a power of two, the smallest element (denoted by 0) is used for merging

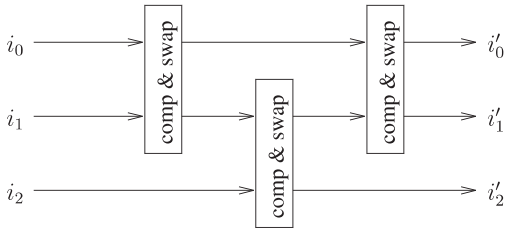


Fig. 4. Sorting network for three inputs. $i_0 \ldots i_2$ are the input unsorted states, and $i'_0 \ldots i'_2$ are the output sorted states

Since $\overline{b}+1$ is equal to $-b$, we can compute $a+(-b)$ by computing $a+\overline{b}+1$. The first carry bit c_0 of this sum is:

$$c_0 := a_0 \vee \overline{b_0}. \tag{7}$$

The ith carry bit of this sum with $i \geq 1$ is:

$$c_i := (a_i \wedge \overline{b_i}) \vee (a_i \wedge c_{i-1}) \vee (\overline{b_i} \wedge c_{i-1}). \tag{8}$$

The value of the last carry bit $c_{\beta-1}$ determines whether we swap a and b. Let b' denote the new value (after swapping) of b. The equation for a' follows the same pattern:

$$b'_i = a_i \wedge c_{\beta-1} \vee b_i \wedge \overline{c_{\beta-1}}. \tag{9}$$

Equation (7) is transformed into CNF using 1 new literal and 3 clauses, Eq. (8) requires 1 new literal and 6 clauses, and Eq. (9) requires 1 new literal and 4 clauses. The swapping has to be done for both a and b. Thus, the total cost of one compare/swap operation with β bits is 3β literals and $14\beta-3$ clauses.

In Fig. 5 we show graphs that present the number of clauses and variables as a function of k, in a formula representing loop-free paths of depth k. The asymptotic advantage of sorting networks is clear from these graphs. The performance of SAT solvers on a particular instance is often not directly related to the size of the CNF. We therefore evaluate the performance of the sorting network and the quadratic encoding on an artificial benchmark with adjustable recurrence diameter. The results are summarized in Table 3. For increasing recurrence diameters k, we show the results for a satisfiable instance (bound k) and an unsatisfiable instance (bound $k+1$) for both methods. The experiments were performed with LIMMAT [4], but we obtained results that are comparable with those of ZCHAFF [20].

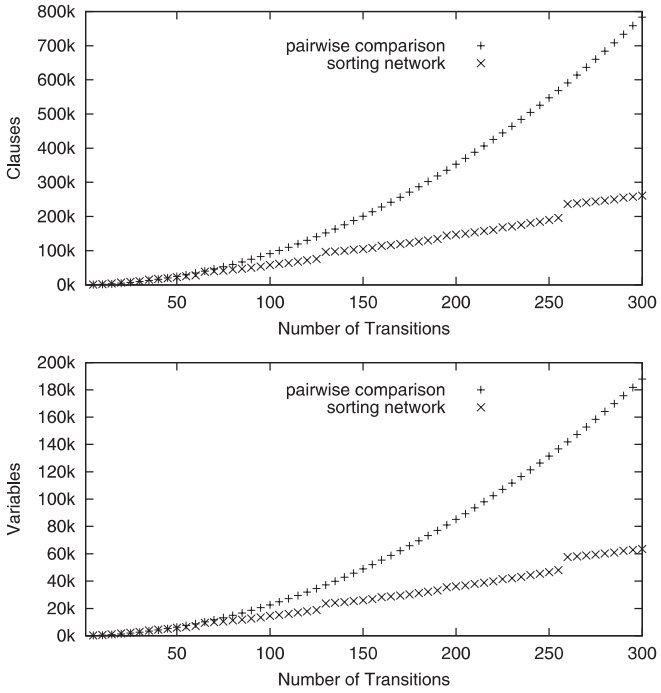


Fig. 5. Comparison of the CNF size (number of literals and clauses) with pairwise comparison and with sorting network

Table 3. Comparison of run time of recurrence diameter test with a sorting network and quadratic encoding. The run time is given in seconds

	quadratic		sorting network	
k	SAT	UNSAT	SAT	UNSAT
10	0.07	0.07	0.11	0.11
20	0.21	0.20	0.29	0.28
30	0.39	0.39	0.51	0.49
40	0.67	0.64	0.81	0.73
50	1.02	0.97	1.07	1.00
60	1.38	1.34	1.31	1.22
70	1.83	1.74	1.79	1.68
80	2.41	2.30	2.16	1.98
90	3.01	2.86	2.41	2.24
100	3.65	4.25	2.78	2.78
Avg.	1.464	1.476	1.324	1.251

For smaller diameters, the quadratic method is faster, while for the larger diameters (> 60), the sorting network is faster. This holds for both satisfiable and unsatisfiable instances. Note that the circuit used is not necessarily representative of circuits used in industry. We were not able to obtain an industrial circuit with a diameter that is both deep enough to make the sorting network efficient and shallow enough to make BMC feasible. We hope that the rapid advancements in SAT solving will eventually make this method more applicable to large circuits as well.

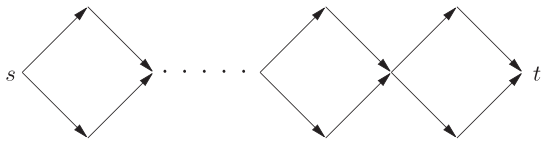
5 The complexity of BMC

According to Theorem 1, the value of $\mathcal{CT}$ can be exponential in the number of state variables. This implies that the SAT instance (as generated in both the syntactic and semantic translations) can have an exponential number of variables, and hence solving it can be doubly exponential. All known SAT-based BMC techniques, including the one presented in this article, have this complexity. Since there exists a singly exponential LTL model checking algorithm in the number of state variables, it is clear that there is a complexity gap of an exponent between the two methods. Why, then, use BMC for attempting to prove that $M \models \varphi$ holds? There are several answers to this question:

1. Indeed, BMC is normally used for detecting bugs, not for proving their absence. The number of variables in the SAT formula is polynomial in k. If the property does not hold, k depends on the location of the shallowest error. If this number is relatively small, solving the corresponding SAT instance can still be easier than competing methods.

2. In many cases the values of $rd^I(\Psi)$ and $d^I(\Psi)$ are not exponential in the number of state variables and can even be rather small. In hardware circuits, the leading cause of exponentially long loop-free paths is counters, and hence designs without counters are much easier to solve. For example, about 25% of the components examined in [3] have a diameter smaller than 20. The authors used a structural analysis of circuits to identify these components and compute their diameters while ignoring the rest of the circuit. This ability to decompose the circuit has led to impressive results in verifying, rather than only falsifying, properties of circuits.
3. For various technical reasons, SAT is not very sensitive to the number of variables in a formula, although theoretically it is exponential in the number of variables. Comparing it to other methods solely based on their corresponding complexity classes is not a very good indicator for their relative success in practice.

We argue that the reason for the complexity gap between SAT-based BMC and LTL model checking (as described in Sect. 3.1) is the following: SAT-based BMC does not keep track of visited states, and therefore it possibly visits the same state an exponential number of times. Unlike explicit model checking it does not explore a state graph, and unlike BDD-based symbolic model checking, it does not memorize the set of visited states. For this reason, it is possible that all paths between two states are explored, and hence a single state can be visited an exponential number of times. For example, an explicit model checking algorithm, such as the double DFS algorithm [14], will visit each state in the graph below not more than twice. SAT-based BMC, on the other hand, will consider in the worst case all 2^n possible paths between s and t, where n is the number of "diamonds" in the graph. In practice no modern SAT solvers would actually be so inefficient because of their ability to prune large parts of the state space, but on the other hand they will not be able to guarantee less than exponential time on this particular example.



A natural question is whether this complexity gap can be closed, i.e., is it possible to change the SAT-based BMC algorithm so it becomes a singly exponential rather than a doubly exponential algorithm. Figure 6 presents a possible singly exponential BMC algorithm for $\mathbf{G}p$ formulas (i.e., reachability) based on an altered SAT algorithm that can be implemented by slightly changing a standard SAT solver. The algorithm forces the SAT solver to follow a particular variable ordering, and hence the main power of SAT (guidance of the search process with splitting heuristics) is lost. Further, it adds constraints for each visited state, forbidding the search process from revisiting it through longer or equally long paths. This potentially adds an exponential number of clauses to the formula.

1. Force a static order, following a forward traversal.
2. Each time a state i is fully evaluated (assigned):
   - Prevent the search from revisiting it through deeper paths, e.g., If $(x_i, \neg y_i)$ is a visited state, then for $i < j \leq CT$ add the following blocking state clause: $(\neg x_j \vee y_j)$.
   - When backtracking from state i, prevent the search from revisiting it in step i by adding the clause $(\neg x_i \vee y_i)$.
   - If $\neg p_i$ holds, stop and return 'Counterexample found'.

Fig. 6. A singly exponential SAT-based BMC algorithm for $\mathbf{G}p$ properties

So far our experiments show that this procedure is worse in practice than the standard BMC.[5] We observed that the SAT solver reaches cycle k quickly and then takes a very long time before it backtracks beyond one or more cycles. Each time it is in cycle k, it simply tries all input combinations, which explains this behavior. Whether it is possible to find a singly exponential SAT-based algorithm that works better in practice than the standard algorithm, is still an open question with a very significant practical importance. The work on SAT instances with bounded cutwidth [25] seems to offer a direction for coping with this problem, because when ordering the variables forward as suggested above the cutwidth does not depend on k.

6 Conclusions

We discussed the advantages of the semantic translation for BMC. We showed that it is in general more efficient, as it results in smaller CNF formulas, and it potentially eliminates redundancies in the property of interest. We also showed how it allows one to compute the completeness threshold $\mathcal{CT}$ for all LTL formulas. This computation is based on computing the recurrence diameter of an automaton. We showed an algorithm for computing the recurrence diameter based on sorting networks, which results in a smaller formula as compared with previously known methods, although we could not locate real examples in which this reduction in the formula size indeed shortens the solving time.

The ability to compute $\mathcal{CT}$ for general LTL enabled us to prove that all existing SAT-based BMC algorithms are doubly exponential in the number of variables. Since LTL model checking is only singly exponential in the number

[5] Biere implemented a similar algorithm in 2001 and reached the same conclusion. For this reason he did not publish this algorithm. Similar algorithms were also used in the past in the context of Automatic Test Pattern Generation (ATPG).

of variables, there is a complexity gap between the two approaches. To close this gap, we suggested a revised BMC algorithm that is only singly exponential but in practice so far has not proved to be better than the original SAT-based BMC.

Acknowledgements. We thank Armin Biere for his suggestions to improve this article, and in particular for helping us get right the results on the size of formulas resulting from the syntactic and semantic translations.

References

1. Ajtai M, Komlós J, Szemerédi S (1983) An $O(N \log N)$ sorting network. In: Proceedings of the 25th ACM symposium on theory of computing, pp 1–9
2. Batcher KE (1968) Sorting networks and their applications. In: Proceedings of the AFIPS spring joint computer conference, 32:307–314
3. Baumgartner J, Kuehlmann A, Abraham J (2002) Property checking via structural analysis. In: Brinksma E, Larsen KG (eds) Proceedings of the 14th international conference on computer aided verification (CAV'02), Copenhagen, Denmark, July 2002. Lecture notes in computer science, vol 2404. Springer, Berlin Heidelberg New York
4. Biere A (2004) The evolution from limmat to nanosat. Technical report, ETH Zürich
5. Biere A, Cimatti A, Clarke E, Zhu Y (1999) Symbolic model checking without BDDs. In: Proceedings of the workshop on tools and algorithms for the construction and analysis of systems (TACAS'99). Lecture notes in computer science, vol . Springer, Berlin Heidelberg New York, pp 193–207
6. Biere A, Cimatti A, Clarke EM, Strichman O, Zue Y (2003) Bounded model checking. In: Advances in computers, vol 58. Academic, New York
7. Cimatti A, Pistore M, Roveri M, Sebastiani R (2002) Improving the encoding of LTL model checking into SAT. In: 3rd international conference on verification, model checking and abstract interpretation (VMCAI), pp 196–207
8. Clarke EM, Grumberg O, Hamaguchi K (1994) Another look at LTL model checking. In: Dill DL (ed) Proceedings of the 6th international conference on computer aided verification. Lecture notes in computer science, vol 818. Springer, Berlin Heidelberg New York, pp 415–427
9. Clarke EM, Kroening D, Ouaknine J, Strichman O (2004) Completeness and complexity of bounded model checking. In: Proceedings of the 5th international conference on verification, model checking and abstract interpretation (VMCAI'04), Venice, Italy, January 2004. Lecture notes in computer science, vol 2937. Springer, Berlin Heidelberg New York, pp 85–96
10. de Moura L, Rueß H, Sorea M (2002) Lazy theorem proving for bounded model checking over infinite domains. In: Proceedings of the 18th international conference on automated deduction (CADE'02), Copenhagen, July 2002
11. Clarke EM, Grumberg O, Peled D (1999) Model checking. MIT Press, Cambridge, MA
12. Frisch A, Sheridan D, Walsh T (2002) A fixpoint based encoding for bounded model checking. In: International conference on formal methods in computer-aided design (FMCAD 2002), Portland, OR, November, pp 238–255
13. Gerth R, Peled D, Vardi M, Wolper P (1995) Simple on-the-fly automatic verification of linear temporal logic. In: Protocol specification testing and verification. Chapman & Hall, London, pp 3–18
14. Holzmann GJ, Peled D, Yannakakis M (1996) On nested depth first search. In: 2nd SPIN workshop, AMS, pp 23–32
15. Knuth DE (1973) The art of computer programming, vol 3: Sorting and searching. Addison-Wesley, Reading, MA
16. Kroening D, Strichman O (2003) Efficient computation of recurrence diameters. In: Proceedings of the 4th international conference on verification, model checking, and abstract interpretation (VMCAI'03), New York, January 2003. Lecture notes in computer science, vol 2575. Springer, Berlin Heidelberg New York, pp 298–309
17. McMillan KL (2002) Applying SAT methods in unbounded symbolic model checking. In: Brinksma E, Larsen K (eds) Proceedings of the 14th international conference on computer aided verification (CAV'02), Copenhagen, July 2002. Lecture notes in computer science, vol 2404. Springer, Berlin Heidelberg New York, pp 250–264
18. McMillan KL (2003) Interpolation and sat-based model checking. In: Hunt WA Jr, Somenzi F (eds) Proceedings of the international conference on computer aided verification (CAV'03), July 2003. Lecture notes in computer science, vol . Springer, Berlin Heidelberg New York
19. Mneimneh M, Sakallah K (2002) SAT-based sequential depth computation. In: Workshop on constraints in formal verification, Ithaca, New York, September
20. Moskewicz M, Madigan C, Zhao Y, Zhang L, Malik S (2001) Chaff: engineering an efficient SAT solver. In: Proceedings of the conference on design automation (DAC'01)
21. Schuppan V, Biere A (2004) Efficient reduction of finite state model checking to reachability analysis. Int J Softw Tools Technol Transfer
22. Sheeran M, Singh S, Stalmarck G (2000) Checking safety properties using induction and a sat-solver. In: Hunt, Johnson (eds) Proceedings of the international conference on formal methods in computer-aided design (FMCAD 2000)
23. Somenzi F, Bloem R (2000) Efficient Büchi automata from LTL formulae. In: Emerson EA, Sistla AP (eds) 12th international conference on computer aided verification (CAV'00), Berlin, July. Springer, Berlin Heidelberg New York, pp 248–263
24. Vardi MY, Wolper P (1986) An automata-theoretic approach to automatic program verification. In: Proceedings of the 1st IEEE symposium on logic in computer science, pp 332–344
25. Wang D, Clarke EM, Zhu Y, Kukula J (2001) Using cutwidth to improve symbolic simulation and boolean satisfiability. In: IEEE International workshop on high level design validation and test (HLDVT 2001), November, p 6