Completeness and Complexity of Bounded Model Checking ⋆

Edmund Clarke[1] Daniel Kroening[1] Joël Ouaknine[1] Ofer Strichman[2]

[1] Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
emc|kroening|ouaknine@cs.cmu.edu
[2] Faculty of Industrial Engineering, Technion, Haifa, Israel
ofers@ie.technion.ac.il

Abstract. For every finite model M and an LTL property φ, there exists a number $\mathcal{CT}$ (the *Completeness Threshold*) 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 over-approximation to $\mathcal{CT}$ for a general LTL property using Büchi automata, following the Vardi-Wolper LTL model checking framework. 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.
The article mainly focuses on observations regarding bounded model checking rather than on a presentation of new techniques.

1 Introduction

Bounded Model Checking (BMC) [2, 3] 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 [3] 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 either (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 pre-computed threshold, which indicates that

⋆ This research is supported by the Semiconductor Research Corporation (SRC) under contract no. 99-TJ-684, the National Science Foundation (NSF) under grants no. 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.

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 over-approximation to the smallest $\mathcal{CT}$ based on graph-theoretic properties of M (such as the *diameter* of the graph representing it) and an automaton representation of $\neg\varphi$. In particular, we consider all models with the same graph-theoretic properties of M as one abstract model. Thus, this computation corresponds to finding the length of the longest shortest counterexample to φ by any one of these graphs, 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 computed by 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 [2], which we will soon describe, is based on a k-steps *syntactic expansion* of the formula, using Pnueli's expansion rules for LTL [8, 12] (e.g., $\mathbf{F}p \equiv p \vee \mathbf{XF}p$). With this translation, the value of $\mathcal{CT}$ was until now known only for unnested properties such as $\mathbf{G}p$ formulas [2] and $\mathbf{F}p$ formulas [11]. 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 [6][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. Non-emptiness 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 Section 3. Deriving from this product a propositional formula $\Omega_\varphi(k)$ that is satisfiable iff 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 Section 4. 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 $M \times B_{\neg\varphi}$. Furthermore, the semantic translation leads to smaller CNF formulas comparing to the syntactic translation. There are two reasons for this:

[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 [6] 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.

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 [14] 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, comparing to a quadratic ratio in the syntactic translation.

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 Section 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, although future improvements of this technique may change this conclusion.

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 propositions) At (ii) $I \subseteq S$ is the set of initial states (iii) $T \subseteq S \times S$ is the transition relation and (iv) $L\colon S \rightarrow 2^{(At)}$ 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 comparing to an explicit representation of the states. This fact is important for establishing the complexity of the semantic translation, as we do in Section 4.

Propositional *Linear Temporal Logic* (LTL) formulas are defined recursively: Boolean variables are in LTL; then, if $\varphi_1, \varphi_2 \in$ LTL then 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) and $\varphi_1 \mathbf{W} \varphi_2$ (φ_1 Waiting-for φ_2), $\varphi_1 \vee \varphi_2$ and $\neg\varphi_1$.

[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 was previously published.

2.2 Bounded Model Checking of LTL properties

Given an LTL property φ, a Kripke structure M and a bound k, Bounded Model Checking 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 $[\![\neg\varphi]\!]_k$ [2, 3, 7]. These translations are based on the LTL expansion rules [8, 12] (e.g., $\mathbf{F}p \equiv p \vee \mathbf{XF}p$ and $\mathbf{G}p \equiv p \wedge \mathbf{XG}p$).

In the rest of this section we consider the translation scheme of Biere et al. [3] 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.

Constructing a propositional formula that captures finite paths is simple. The formula is expanded k times according to the LTL expansion rules mentioned above, where each subformula, at each location, is represented by a new variable. For example, for the operator $\mathbf{F}$, the expansion for $i \le k$ is $[\![\mathbf{F}\varphi]\!]_k^i := [\![\varphi]\!]_k^i \vee [\![\mathbf{F}\varphi]\!]_k^{i+1}$ and for $i > k$ $[\![\mathbf{F}\varphi]\!]_k^i := \textsc{false}$. Similar rules exist for the other temporal operators and for the propositional connectives.

To capture paths ending with a loop (representing infinite paths) we need to consider the state s_l $(l \le k)$, which the last state transitions to. The translation for the operator $\mathbf{F}$ for such paths is: ${}_l[\![\mathbf{F}\varphi]\!]_k^i := {}_l[\![\varphi]\!]_k^i \vee {}_l[\![\mathbf{F}\varphi]\!]_k^{\mathrm{succ}(i)}$ where $\mathrm{succ}(i) = i + 1$ if $i < k$, and $\mathrm{succ}(i) = l$ otherwise.

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

Proposition 1. *The syntactic translation results in $O(k \cdot |v| + (k+1)^2 \cdot |\varphi|)$ variables, where v is the set of variables defining the states of M, i.e., $|v| = O(\log |S|)$.*

Proof. Recall the structure of the formula $\Omega_\varphi(k) : [\![M]\!]_k \wedge [\![\neg\varphi]\!]_k$. The sub-formula $[\![M]\!]_k$ adds $O(k \cdot |v|)$ variables. The sub-formula $[\![\neg\varphi]\!]_k$ adds, according to the recursive translation scheme, not more than $(k+1)^2 \cdot |\varphi|$ variables, because each expression of the form ${}_l[\![\varphi]\!]_k^l$ 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.

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 [2], and later optimized in [1, 13].

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 [2]. The number of variables required by their method is proportional to 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.

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 [2] 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 [11] 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.

In the next section we review how LTL model checking can be done with Büchi automata. In Section 4 we will show how a similar method can be used for generating $\Omega_\varphi(k)$.

3 LTL model checking with Büchi automata

In this section we describe how model checking of LTL formulas can be done with Büchi automata, as it was first introduced by Vardi and Wolper in [15]. 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 $\mathit{inf}(w)$ the set of states that appear infinitely often in the path of w on M, then $\mathit{inf}(w) \cap F \neq \emptyset$.

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[9][4]. We do not repeat the details of this construction; rather we present several examples in Fig. 1 of such translations.

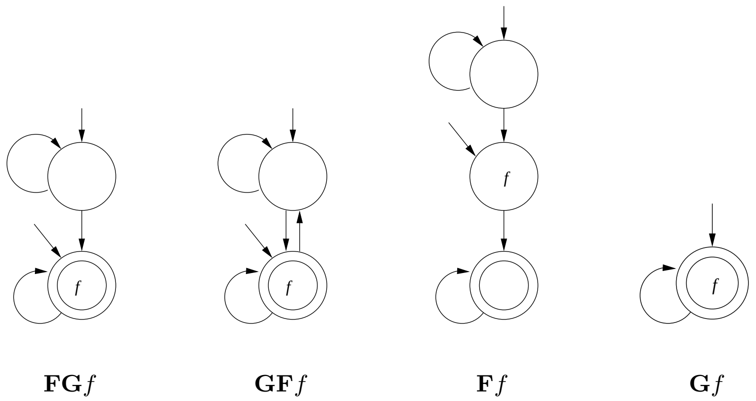


Fig. 1. Several LTL formulas and their corresponding Büchi automata. Accepting states are marked by double circles.

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

4 The semantic translation

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

[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 $|\varphi|$.

BMC procedure. This can be done by searching for a witness to the property $\varphi' \doteq \mathbf{G}(\text{TRUE})$ under the fairness constraint $\bigvee_{F_i \in F} F_i$ [5] (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:

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

The right-most conjunct in Equation 1 constrains one of the states in F to be true in at least one of the states of the loop.

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 $\mathcal{CT}$.

4.1 The semantic translation is more efficient

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

Proposition 2. *The semantic translation results in $O(k \cdot (|v| + |\varphi|))$ variables.*

Proof. The transition relation of the Büchi automaton constructed from φ is defined by $O(|\varphi|)$ variables (the automaton itself is exponential in the size of the formula, but its corresponding representation by a transition relation is as defined above). The SAT formula is constructed by unfolding k times the product Ψ, hence it uses $O(k \cdot (|v| + |\varphi|))$ variables. It also includes constraints for identifying a loop with a fair state, but these constraints only add clauses, not new variables. □

We conducted some experiments in order to check the difference between the translations. We conducted this experiment with NuSMV 2.1, which includes an optimized syntactic translation [4]. To generate the semantic translation, we derived the Büchi automaton with WRING [14] and added the resulting automaton to the NuSMV model. Then the property to be checked is simply $\mathbf{F}(\text{FALSE})$ under possible fairness constraints, as prescribed by the Büchi automaton. The table in Figure 2 summarizes these results.

As can be seen from the table and from Figure 3, there is a linear growth in the number of variables in the resulting CNF formula with the semantic translation, and a quadratic growth with the syntactic translation. Furthermore, the last three formulas have redundancy that is removed by WRING, but is not removed with the syntactic translation (observe that formulas 2 and 3 result in

Property	K	Semantic	Syntactic
$(x_0\mathbf{U}(!x_0 \wedge x_1))\mathbf{U}x_2$	7	1090	986
	15	2298	3098
	30	4563	14073
	45	6828	39898
$\mathbf{FFF}x_2$	7	528	569
	15	1112	1321
	30	2207	3076
	45	3302	5281
$\mathbf{FFFFFF}x_2$	7	528	632
	15	1112	Timeout
	30	2207	
	45	3302	
$\mathbf{GFGF}x_2$	7	586	590
	15	1234	1426
	30	2449	3511
	45	3664	Timeout

Fig. 2. The number of variables in the CNF formula resulting from the semantic and syntactic translation. The former grows linearly with k, while the latter may grow quadratically. The Timeout entry indicates that it takes NuSMV more than 15 minutes to generate the CNF formula.

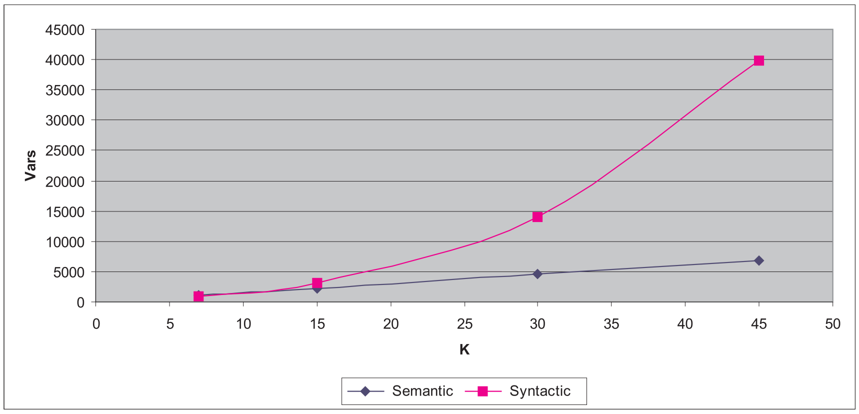


Fig. 3. The number of variables for the formula $(x_0\mathbf{U}(!x_0 \wedge x_1))\mathbf{U}x_2$ with the semantic and syntactic translations.

the same number of variables in the semantic translation, but not in the syntactic translation). The model which we experimented with was a toy example, and hence the resulting CNF was easy to solve with both translations. But it was sufficient for demonstrating the differences between the translations and that in some cases even generating the CNF formulas takes a long time with the syntactic translation.

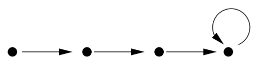
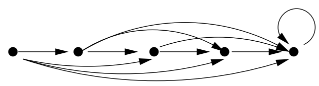
4.2 A calculation of $\mathcal{CT}$ for LTL based on $M \times B_{\neg\varphi}$

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

Theorem 1. *A completeness threshold for any LTL property φ when using Equation 1 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 witness for the non-emptiness 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 exists 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. $\square$

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 1): for all M, any state satisfying $\neg p$ in the product $\Psi : M \times \varphi$ 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 non-temporal, is $d^I(\Psi)$.*

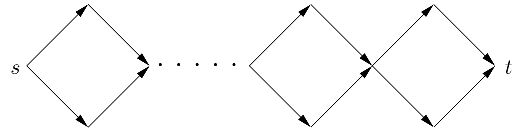
We believe that this theorem can be extended to all safety properties.

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 for 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 [1] have a diameter smaller than 20.
3. For various technical reasons, SAT is not very sensitive to the number of variables in the 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 Section 3), 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 [10], 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.



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 4 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 constrains 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. 4. 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]. 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.

6 Conclusions

We discussed the advantages of the semantic translation for BMC, as was first suggested in [6]. 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 to compute the completeness threshold $\mathcal{CT}$ for all LTL formulas.

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 of variables, there is a complexity gap between the two approaches. In order 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.

[5] A. 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).

References

1. J. Baumgartner, A. Kuehlmann, and J. Abraham. Property checking via structural analysis. In E. Brinksma and K.G. Larsen, editors, *Proc. 14^{th} Intl. Conference on Computer Aided Verification (CAV'02)*, volume 2404 of *LNCS*, Copenhagen, Denmark, July 2002. Springer-Verlag.
2. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proc. of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, LNCS, pages 193–207. Springer-Verlag, 1999.
3. A. Biere, A. Cimatti, E.M. Clarke, O. Strichman, and Y. Zue. *Bounded Model Checking*, volume 58 of *Advances in computers*. Academic Press, 2003.
4. Alessandro Cimatti, Marco Pistore, Marco Roveri, and Roberto Sebastiani. Improving the encoding of LTL model checking into SAT. In *VMCAI*, pages 196–207, 2002.
5. E.M. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. In D. L. Dill, editor, *Proc. 6th Conference on Computer Aided Verification*, volume 818 of *Lect. Notes in Comp. Sci.*, pages 415–427. Springer-Verlag, 1994.
6. L. de Moura, H. Rueß, and M. Sorea. Lazy theorem proving for bounded model checking over infinite domains. In *Proc. 18th International Conference on Automated Deduction (CADE'02)*, Copenhagen, Denmark, July 2002.
7. A. Frisch, D. Sheridan, and T. Walsh. A fixpoint based encoding for bounded model checking. In *Formal Methods in Computer-Aided Design (FMCAD 2002)*, pages 238 – 255, Portland, Oregon, Nov 2002.
8. D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proc. 7th ACM Symp. Princ. of Prog. Lang.*, pages 163–173, 1980.
9. R. Gerth, D. Peled, M. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification Testing and Verification*, pages 3–18. Chapman & Hall, 1995.
10. G.J. Holzmann, D. Peled, and M. Yannakakis. On nested depth first search. In *Second SPIN workshop*, AMS, pages 23–32, 1996.
11. D. Kroening and O. Strichman. Efficient computation of recurrence diameters. In *4th International Conference on Verification, Model Checking, and Abstract Interpretation*, volume 2575 of *Lecture Notes in Computer Science*, pages 298–309, NYU, New-York, January 2003. Springer Verlag.
12. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1991.
13. M. Mneimneh and K. Sakallah. SAT-based sequential depth computation. In *Constraints in formal verification workshop*, Ithaca, New-York, Sep 2002.
14. F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In E. A. Emerson and A. P. Sistla, editors, *Twelfth Conference on Computer Aided Verification (CAV'00)*, pages 248–263, Berlin, July 2000. Springer-Verlag.
15. M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. First IEEE Symp. Logic in Comp. Sci.*, pages 332–344, 1986.