Proving More Properties with Bounded Model Checking*

Mohammad Awedh and Fabio Somenzi

University of Colorado at Boulder
{Awedh,Fabio}@Colorado.EDU

Abstract. Bounded Model Checking, although complete in theory, has been thus far limited in practice to falsification of properties that were not invariants. In this paper we propose a termination criterion for all of LTL, and we show its effectiveness through experiments. Our approach is based on converting the LTL formula to a Büchi automaton so as to reduce model checking to the verification of a fairness constraint. This reduction leads to one termination criterion that applies to all formulae. We also discuss cases for which a dedicated termination test improves bounded model checking efficiency.

1 Introduction

The standard approach to model checking an LTL property [17, 9] consists of checking language emptiness for the composition of the model at hand and a Büchi automaton that accepts all the counterexamples to the LTL property. A competing approach consists of encoding the problem as propositional satisfiability (SAT) [2]. In this approach, known as Bounded Model Checking (BMC), a propositional formula is constructed such that a counterexample of bounded length for the LTL formula exists if and only if the propositional formula is satisfiable.

The basic BMC just described is not complete in practice: It often finds a counterexample if it exists, but it cannot prove that a property passes unless a tight bound on the *completeness threshold* [6] of the state graph is known. Such a bound is difficult to obtain. The issue of completeness is addressed by recourse to an induction proof in [14] and [7], or by the use of interpolants in [10] so that BMC can be used for both verification and falsification of invariants.

The approach of [14] is based on the observation that if a counterexample to an invariant exists, then there is a simple path from an initial state to a failure state that goes through no other initial or failure state. Every infinite path that extends this simple path violates the invariant. Therefore, an invariant holds if all states of all paths of length k starting from the initial states satisfy the invariant, and moreover, there is no simple path of length $k+1$ starting at an initial state or leading to a failure state, and not going through any other initial or failure states.

This method can be easily extended to prove LTL safety properties. For full LTL, one can convert the check for a liveness property into the check of a safety property

* This work was supported in part by SRC contract 2003-TJ-920.

following [13]. However, the conversion doubles the number of state variables. An approach that does not incur this penalty is the subject of this paper. We translate the given LTL formula into a Büchi automaton and compose the latter with the model as in [17, 6]. This step reduces the checking of any LTL property to the one of $\mathsf{F}\,\mathsf{G}\,\neg p$ for a propositional formula p, on the composed model.

A counterexample to $\mathsf{F}\,\mathsf{G}\,\neg p$ exists if there is a simple path from an initial state of the composed model followed by a transition to some state on the path.[1] If there is no simple path from an initial state of length k, then there cannot be a counterexample of length $k+1$. This condition is the counterpart of the one for invariants that checks for simple paths from initial states. However, there is no strict analog of the states failing an invariant in the general case. Hence, the check for no simple paths of length k into failure states must be replaced by a criterion that guarantees that no loops satisfying certain acceptance conditions may be closed by extending paths of length k.

In this paper, we present such a criterion to prove LTL properties in general. We also discuss a more efficient criterion for a common special case. The translation of the LTL formula can be accomplished in several ways. In Sect. 4 we discuss the impact of various choices on the efficiency of the model checker.

As in the case of invariants, the effectiveness of our termination criteria depends on the lengths of the simple paths in a state graph. However, our experiments, presented in Sect. 5, show that many properties that defy verification attempts by either standard BMC or BDD-based model checking can be proved by our approach.

2 Preliminaries

The goal of LTL model checking is to determine whether an LTL property is satisfied in a finite model of a sequential system. The behavior of this sequential system is described by a *Kripke structure*. A Kripke structure $\mathcal{K} = \langle S, \delta, I, L\rangle$ consists of a finite set of states S whose connections are described by the transition relation $\delta \subseteq S \times S$. If $(s,t) \in \delta$, then there is a transition form state s to state t in $\mathcal{K}$. The transition relation δ is total: For every state $s \in S$ there is a state $t \in S$ such that $(s,t) \in \delta$. $I \subseteq S$ is the set of initial states of the system. The labeling function $L : S \to 2^{AP}$ indicates what atomic propositions hold at each state. We write $\delta(s,t)$ for $(s,t) \in \delta$; that is, we regard δ as a predicate. Likewise, we write $I(s)$ to indicate that s is an initial state, and, for $p \in AP$, $p(s)$ to indicate that $p \in L(s)$.

Definition 1. *A sequence of states $(s_0, \ldots, s_k)$ forms a* path *of length k of Kripke structure $\mathcal{K}$ if it satisfies*

$$path_k = \bigwedge_{0 \le i < k} \delta(s_i, s_{i+1}) \ .$$

The path is initialized *if $I(s_0)$ holds. A* simple path *of length k satisfies:*

$$simplePath_k = path_k \wedge \bigwedge_{0 \le i < j \le k} (s_i \neq s_j) \ .$$

[1] Precisely, this is the case when BMC starts from paths of length 0, and increases the length by 1 every time.

The simple path condition can be easily expressed with a number of CNF clauses that is quadratic in the length k of the path. Recent work [8] reduces the number of required clauses to $O(k \log^2 k)$.

Definition 2. *A* loop condition L_k *is true of a path of length* k *if and only if there is a transition from state* s_k *to some state of the path.*

$$L_k = \bigvee_{0 \le l \le k} \delta(s_k, s_l) \ .$$

Definition 3. *The* LTL formulae *over atomic propositions* AP *are defined as follows*

- *Atomic propositions,* true*, and* false *are LTL formulae.*
- *if* f *and* g *are LTL formulae, then so are* $\neg f$*,* $f \wedge g$*,* $f \vee g$*,* $\mathsf{X}\, f$*, and* $f \,\mathsf{U}\, g$*.*

An LTL formula that does not contain the temporal operators (X *and* U*) is* propositional*. We write* $f \,\mathsf{R}\, g$ *for* $\neg(\neg f \,\mathsf{U}\, \neg g)$*,* $\mathsf{F}\, f$ *for* $\mathsf{true} \,\mathsf{U}\, f$*, and* $\mathsf{G}\, f$ *for* $\mathsf{false} \,\mathsf{R}\, g$*.*

LTL formulae are interpreted over infinite paths. An atomic proposition p holds along a path $\pi = (s_0, s_1, \ldots)$ if $p(s_0)$ holds. Satisfaction for true, false, and the Boolean connectives is defined in the obvious way; $\pi \models \mathsf{X}\, f$ iff $\pi^1 \models f$, where $\pi^i = (s_i, s_{i+1}, \ldots)$; and $\pi \models f \,\mathsf{U}\, g$ iff there exists $i \ge 0$ such that $\pi^i \models g$, and for $j < i$, $\pi^j \models f$.

A *safety* linear-time property is such that every counterexample to it has a finite prefix that, however extended to an infinite path, yields a counterexample. A *liveness* property, on the other hand, is such that every finite path can be extended to a model of the property. Every linear time property can be expressed as the intersection of a safety and a liveness property [1].

Though in principle a counterexample to a linear-time property is always an infinite sequence of states, for safety properties it is sufficient and customary to present an initialized simple path that leads to a *bad state*—one from which all extensions to infinite paths result in counterexamples. For liveness properties, on the other hand, counterexamples produced by model checkers are ultimately periodical sequences of states. Such sequences can be presented in the form of an initialized path followed by a transition to one of its states. As an example, in a counterexample to the liveness property $\mathsf{F}\, p$ all states of the path satisfy $\neg p$. In a counterexample to $\mathsf{F}\, \mathsf{G}\, \neg p$, the transition from the last state of the path reaches back far enough that a state satisfying p is included in the loop.

Definition 4. *A* Büchi automaton *over alphabet* Σ *is a quadruple*

$$\mathcal{A} = \langle Q, \Delta, q_0, F \rangle \ ,$$

where Q *is the finite set of states,* $\Delta \subseteq Q \times \Sigma \times Q$ *is the transition relation,* $q_0 \in Q$ *is the initial state, and* $F \subseteq Q$ *is a set of accepting states (or fair set).*

A run *of* $\mathcal{A}$ *over an infinite sequence* $w = (w_0, w_1, \ldots) \in \Sigma^\omega$ *is an infinite sequence* $\rho = (\rho_0, \rho_1, \ldots)$ *over* Q*, such that* $\rho_0 = q_0$*, and for all* $i \ge 0$*,* $(\rho_i, w_i, \rho_{i+1}) \in \Delta$*. A run* ρ *is* accepting *if there exists* $q_j \in F$ *that appears infinitely often in* ρ*.*

Boolean satisfiability (SAT) is a well-known NP-complete problem. It consists of computing a satisfying variable assignment for a propositional formula or determining that no such assignment exists.

3 Proving Properties with Bounded Model Checking

Bounded Model Checking (BMC) [2] reduces the search for a counterexample to an LTL property to propositional satisfiability. Given a Kripke structure $\mathcal{K}$, an LTL formula f, and a bound k, BMC tries to refute $\mathcal{K} \models f$ by proving the existence of a witness of length k to the negation of the LTL formula.

BMC generates a propositional formula $[\![\mathcal{K}, \neg f]\!]_k$ that is satisfiable if and only if a counterexample to f of length k exists; $[\![\mathcal{K}, \neg f]\!]_k$ is defined as follows:

$$[\![\mathcal{K}, \neg f]\!]_k = I(s_0) \wedge path_k \wedge [\![\neg f]\!]_k \ , \tag{1}$$

where $[\![\neg f]\!]_k$ expresses the satisfaction of $\neg f$ along that path. Of particular interest to us are three cases:

$$[\![\neg \mathsf{G}\, p]\!] = \bigvee_{0 \le i \le k} \neg p(s_i) \tag{2a}$$

$$[\![\neg \mathsf{F}\, \mathsf{G}\, \neg p]\!] = \bigvee_{0 \le l \le k} (\delta(s_k, s_l) \wedge \bigvee_{l \le i \le k} p(s_i)) \tag{2b}$$

$$[\![\neg \mathsf{F}\, p]\!] = L_k \wedge \bigwedge_{0 \le i \le k} \neg p(s_i) \ , \tag{2c}$$

where p is a propositional formula. The first of the three cases is encountered when checking invariants. The second occurs when checking fairness constraint [4]. It is important because model checking any LTL formula f can be reduced to checking for the satisfaction of a fairness constraint by translating the LTL formula to a Büchi automaton. This translation allows us to deal in a uniform manner with all of LTL. However, common cases may benefit from special treatment. We illustrate these benefits for formulae of the form $\mathsf{F}\, p$, which is our third interesting case.

For an invariant $\mathsf{G}\, p$, no counterexample of length greater than or equal to k exists if $[\![\mathcal{K}, \neg\, \mathsf{G}\, p]\!]_k$ is unsatisfiable, and either of the following predicates is unsatisfiable [14]:

$$\chi(k) = I(s_0) \wedge simplePath_k \wedge \bigwedge_{0 < i \le k} \neg I(s_i) \tag{3a}$$

$$\zeta(k) = simplePath_k \wedge \neg p(s_k) \wedge \bigwedge_{0 \le i < k} p(s_i) \ . \tag{3b}$$

For checking fairness constraints, an unsatisfiable $\chi(k)$ does not guarantee termination because all counterexamples may have to go through more than one initial state. Therefore, a weakened form must be used:

$$\chi'(k) = I(s_0) \wedge simplePath_k \ . \tag{3a$'$}$$

If $\chi'(k)$ is unsatisfiable, then there can be no simple path of length k that can be extended to a counterexample by a transition back to a state along the path. For (3b), dropping the requirement that all states except the last one satisfy p is not sufficient. In the next two sub-sections we develop termination criteria that replace (3b) when $f = \mathsf{F}\, \mathsf{G}\, \neg p$, and when $f = \mathsf{F}\, p$.

3.1 Proving $\mathsf{F}\,\mathsf{G}\,\neg p$

Theorem 1. *Let $\mathcal{K} = \langle S, \delta, I, L\rangle$ be a Kripke structure, let $p \in AP$ be an atomic proposition, and let the following predicates denote sets of paths in $\mathcal{K}$:*

$$\alpha(k) = I(s_0) \wedge simplePath_k \wedge p(s_k) \tag{4a}$$

$$\beta(k) = simplePath_{k+1} \wedge \neg p(s_k) \wedge p(s_{k+1}) \tag{4b}$$

$$\beta'(k) = simplePath_{k+1} \wedge \bigwedge_{0\le i\le k} \neg p(s_i) \wedge p(s_{k+1}) \tag{4b$'$}$$

$$[\![\mathcal{K}, \neg\,\mathsf{F}\,\mathsf{G}\,\neg p]\!]_k = I(s_0) \wedge path_k \wedge \bigvee_{0\le l\le k} \left[\delta(s_k, s_l) \wedge \bigvee_{l\le i\le k} p(s_i)\right] . \tag{4c}$$

Let m be the least value of k for which $\beta'(k)$ is unsatisfiable, and n the least value of k for which $(\alpha \vee \beta)(k)$ is unsatisfiable. Then, $[\![\mathcal{K}, \neg\,\mathsf{F}\,\mathsf{G}\,\neg p]\!]_k$ is unsatisfiable unless it is satisfiable for $k \le n + m - 1$.

Proof. Since $\beta'(k+1)$ is satisfiable only if $\beta'(k)$ is, and $\beta'(|S|+1)$ is unsatisfiable, there is a minimum $m \ge 0$ such that $\beta'(m)$ is unsatisfiable, and for $k > m$, $\beta'(k)$ remains unsatisfiable. A similar argument applies to $\beta(k)$.

If $\alpha(k)$ is unsatisfiable, every initialized simple path of length k in $\mathcal{K}$ ends with a state s_k such that $\neg p(s_k)$. In addition, if $\beta(k)$ is unsatisfiable, no simple path of length k that ends in a state s_k such that $\neg p(s_k)$ can be extended to a simple path of length $k+1$ such that $p(s_{k+1})$. Hence, every initialized simple path of length $k+1$ ends in a state s_{k+1} such that $\neg p(s_{k+1})$. Therefore, $(\alpha\vee\beta)(k+1)$ is satisfiable only if $(\alpha\vee\beta)(k)$ is. Since $\alpha(|S|+1)$ is unsatisfiable, there is a minimum $n \ge m$ for which $(\alpha \vee \beta)(n)$ is unsatisfiable. In addition, for $k > n$ $(\alpha \vee \beta)(k)$ remains unsatisfiable.

If $[\![\mathcal{K}, \neg\,\mathsf{F}\,\mathsf{G}\,\neg p]\!]_k$ is satisfiable for $k = n' \le n$, then the theorem holds for $\mathcal{K}$. Suppose it is satisfiable for $k = n' > n$, but not for any value of k less than or equal to n. Then

$$\gamma(k) = I(s_0) \wedge simplePath_k \wedge \bigvee_{0\le l\le k} \left[\delta(s_k, s_l) \wedge \bigvee_{l\le i\le k} p(s_i)\right] \tag{4c$'$}$$

is also satisfiable for some $k = n''$, $n < n'' \le n'$. Since every initialized simple path of length $n'' \ge n$ satisfies $\neg(p(s_n) \vee \cdots \vee p(s_{n''}))$, if there is a path of length $k > n$ satisfying $\gamma(k)$, no state s_i in (4c$'$) such that $p(s_i)$ holds can have $i \ge n$. Hence, the maximum length of such a path is $m + n - 1$; otherwise, there would be a simple path of length $m' > m$ satisfying (4b$'$) from s_n to a state that satisfies p. Therefore, if there is no path of length at most $m + n - 1$ that satisfies $\gamma(k)$, then $[\![\mathcal{K}, \neg\,\mathsf{F}\,\mathsf{G}\,\neg p]\!]_k$ is unsatisfiable for any $k \ge 0$. □

Theorem 2. *There exists a family of structures $\{\mathcal{K}_i\}$, $i \ge 0$, such that the minimum value of k for which $\gamma(k)$ is satisfiable is $m + n - 1 = 2n - 1$.*

Proof. Structure $\mathcal{K}_i$ is defined as follows:

$$S_i = \{s_0, \ldots, s_{2i+1}\} \qquad I_i = \{s_0\}$$

$$\delta_i = \{(s_j, s_{j+1}) \mid 0 \le j \le 2i\} \cup \{(s_{2i+1}, s_i)\} \qquad L(s_j) = \begin{cases} \{p\} & \text{if } j = i \\ \emptyset & \text{otherwise} \end{cases} .$$

For this structure, $m = n = i + 1$; $\gamma(k)$ is satisfiable for $k = 2i + 1$ and for no other value of k. (Regarding criterion (3a′), $\chi'(k)$ is unsatisfiable for $k > 2i + 1$.) □

As shown in Sect. 5, for many models and properties, the termination criterion based on Theorem 1 is more effective than the one based on (3a′).

The conditions of Theorem 1 can be checked efficiently by observing that (4b) is unsatisfiable only if (4b′) is, and that the satisfiability of (4a) is immaterial until (4b) becomes unsatisfiable. Initially, it is therefore sufficient to check (4b′); when this becomes unsatisfiable, one records the value of m and switches to checking (4b). When the latter also becomes unsatisfiable, then one starts monitoring (4a) until the value of n is found. Naturally, if (4c) becomes satisfiable, the process terminates. It is not required to check one of (4a)–(4b′) for all values of k, though, obviously, skipping some checks may lead to trying larger values of k than strictly necessary.

3.2 Trap States

Suppose that a predicate τ is given such that from a state s that satisfies $\tau(s)$, no state s' satisfying $p(s') \vee \neg\tau(s')$ can be reached. Then, when checking $\mathsf{F}\,\mathsf{G}\,\neg p$, (3a′) can be strengthened as follows:

$$\chi''(k) = I(s_0) \wedge \mathit{simplePath}_k \wedge \neg\tau(s_k) \ . \tag{3a″}$$

The model on which $\mathsf{F}\,\mathsf{G}\,\neg p$ is checked is normally obtained by composition of the given Kripke structure with a Büchi automaton for the negation of an LTL property. The automaton may contain a *trap state*, that is, a non-accepting state with a self-loop as only outgoing transition. Such a state is often introduced when making the transition relation Δ of the automaton complete. In such cases, one can take τ as the predicate that is true of all states of the composition that project on the trap state of the automaton.

3.3 Proving $\mathsf{F}\,p$

Theorem 3. *Let $\mathcal{K} = \langle S, \delta, I, L\rangle$ be a Kripke structure, let $p \in AP$ be an atomic proposition, and let the following predicates denote sets of paths in $\mathcal{K}$:*

$$\theta(k) = I(s_0) \wedge \mathit{simplePath}_k \wedge \bigwedge_{0 \le i \le k} \neg p(s_i) \tag{5a}$$

$$[\![\mathcal{K}, \neg\,\mathsf{F}\,p]\!]_k = I(s_0) \wedge \mathit{path}_k \wedge L_k \wedge \bigwedge_{0 \le i \le k} \neg p(s_i) \ . \tag{5b}$$

Let n be the least value of k such that $\theta(k)$ is unsatisfiable. Then $[\![\mathcal{K}, \neg\,\mathsf{F}\,p]\!]_k$ is unsatisfiable unless it is satisfiable for $k \le n$.

Proof. Since $\theta(|S|+1)$ is unsatisfiable, there exists a minimum k such that $\theta(k)$ is unsatisfiable. Let this minimum be n. Since $\theta(k+1)$ implies $\theta(k)$, if $\theta(n)$ is unsatisfiable, for $k > n$, $\theta(k)$ remains unsatisfiable.

If $[\![\mathcal{K}, \neg \mathsf{F}\, p]\!]_k$ is satisfiable for $k = n' \leq n$, then the theorem holds for $\mathcal{K}$. Suppose it is satisfiable for $k = n' > n$, but not for any value of k less than or equal to n. Then

$$\sigma(k) = I(s_0) \wedge simplePath_k \wedge L_k \wedge \bigwedge_{0 \leq i \leq k} \neg p(s_i) \qquad (5b')$$

is also satisfiable for some $k = n''$, $n < n'' \leq n'$. Since $\sigma(k)$ implies $\theta(k)$, assuming that $\sigma(k)$ is satisfiable for $k = n'' > n$ leads to a contradiction. □

Note that the value of n in Theorem 3 corresponds to the (predicated) recurrence $\neg p$-radius of [13].

4 Minimum-Length Counterexamples

One virtue of the standard BMC algorithm is that it can produce counterexamples of minimum length for all LTL properties if the lengths of the paths whose existence is checked by SAT starts at 0 and is increased by 1 every time. With BDD-based LTL model checking this is not the case for two reasons. The first is that the shortest fair cycle problem is solved only heuristically [5, 12]. The second reason is that in BDD-based LTL model checking, a counterexample is a path in the composition of Kripke structure and property automaton. Such a counterexample may be longer than the shortest counterexamples found in the Kripke structure.

Example 1. Figure 1 shows a Kripke structure $\mathcal{K}$ with $S = \{a, b\}$, $\delta = \{(a, b), (b, a)\}$, $I = \{a\}$, $L(a) = \{r\}$, and $L(b) = \emptyset$. This structure is composed with a Büchi automaton $\mathcal{A}$ for the negation of $\varphi = \mathsf{G}(r \rightarrow \mathsf{F}\, q)$. The alphabet Σ of $\mathcal{A}$ is 2^{AP}. In the figure, an arc of the automaton is annotated with the characteristic function of all the labels for which there is a transition between the two states connected by the arc. Hence, $\mathcal{A}$ can follow an arc into a state if it reads a letter of the input word that satisfies the arc's formula. The doubly-circled states are accepting. The shortest counterexample to $\mathcal{K} \models \varphi$ found in $\mathcal{K} \parallel \mathcal{A}$ includes three states, $a0$, $b1$, and $a1$, even though there is a counterexample consisting of two states, a and b, in $\mathcal{K}$.

Even when it is possible to retrieve a shortest path in the Kripke structure from the shortest path in the composition—as in the case of Example 1—the computation on $\mathcal{K} \parallel \mathcal{A}$ is likely to be more expensive than the one on $\mathcal{K}$ alone because the transition relation is unrolled more times.

An LTL formula may be translated into many distinct Büchi automata. Though they all accept the same language, they may differ in "style" (labels on the states vs. labels on the transitions; one acceptance condition vs. several), or simply in the numbers of states and transitions.

Automata with labels on the transitions react to the evolution of the Kripke structure with which they are composed with a delay of one step. This is an obstacle in producing shortest counterexamples. Automata with labels on the states do not have this disadvantage, but do not guarantee shortest counterexamples either.

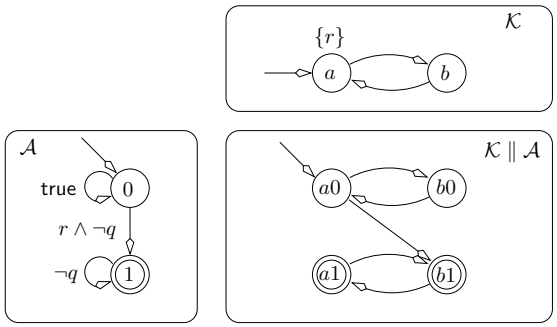

Fig. 1. The composition with the Büchi automaton affects the length of the counterexample

Example 2. Figure 2 shows the Kripke structure of Example 1 composed with a Büchi automaton $\mathcal{A}'$ for the negation of $\varphi = \mathsf{G}(r \rightarrow \mathsf{F}\, q)$ with labels on the states. The

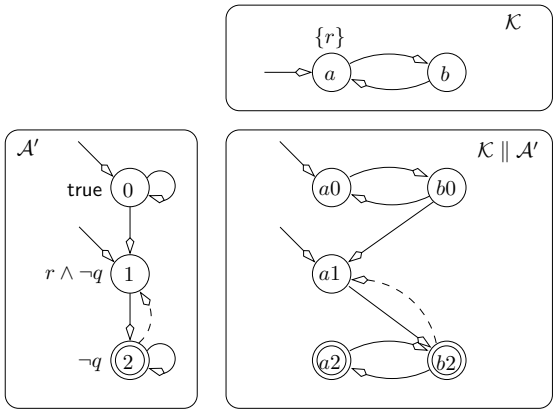

Fig. 2. The position of the labels in the Büchi automaton and its transition relation affect the length of the counterexample

transition drawn with a dashed line from State 2 to State 1 can be added or removed without changing the language accepted by $\mathcal{A}'$ [16]. However, whether it is present or not affects the counterexample found to $\mathcal{K} \models \varphi$. With the optional transition, the language emptiness check applied to $\mathcal{K} \parallel \mathcal{A}'$ returns the cycle $(a1, b2)$, which is of minimal length. By contrast, when the transition is omitted, the shortest counterexample to $\mathcal{K} \models \varphi$ has three states: $a1$, $b2$, and $a2$.

Example 2 shows that adding transitions to an automaton may lead to shorter counterexamples. On the other hand, more transitions may result in longer simple paths which may delay the triggering of the termination conditions. To avoid these problems, the check for existence of counterexamples of length k is performed according to the orig-

inal algorithm of [2], while the termination criteria are applied to the composition of Kripke structure and Büchi automaton. The overhead of building two distinct models for each value of k is more than compensated by the ability to terminate sooner for failing properties.

5 Experimental Results

We have implemented the termination criteria of Theorems 1 and 3, (3a′), and (3a″) in VIS [3, 18]. VIS includes an implementation of the BMC algorithm of [2] that uses zChaff [11] to check for the satisfiability of propositional formulae. VIS detects LTL formulae of special types and treats them accordingly.

- For invariants and formulae that are syntactically safe according to the definition of [15], VIS returns a simple path to a bad state in case of failure. The termination criteria (3a) and (3b) of [14] are applied.
- LTL formulae that contain no temporal operators except X are called *bounded*. The *depth* of a bounded formula is the maximum nesting level of X operators in it. VIS concludes that a bounded formula holds unless it finds a counterexample whose length does not exceed the formula's depth. Propositional formulae are bounded formulae of depth 0.

For formulae that fall into one of the above categories, we use the termination checks already implemented. For the others, we have implemented two approaches.

- The first approach applies the standard BMC algorithm augmented with the termination check of (3a″) to the composition of the original model and a Büchi automaton for the negation of the formula.
- The second approach checks if the given formula is of the form $\mathsf{F}\, p$, in which case, the standard BMC algorithm augmented with the termination check of Theorem 3 is applied. Otherwise, the termination checks of (3a″) and Theorem 1 are applied to the composition of the original model and a Büchi automaton for the negation of the formula, while standard BMC is applied to the original model to check for violations of the property as discussed in Sect. 4.

The results presented in the following tables are for models that are either from industry or from the VIS Verification Benchmark set [18]. For each model, we count each LTL property as a separate experiment. We exclude experiments such that all the methods we compare finish in less than 1 s. Experiments that take more than 1800 s are considered timed out. For all experiments, we set the maximum value of k to 30 and we check for termination at each step. The experiments were run on an IBM IntelliStation with a 1.7 GHz Pentium IV CPU and 2 GB of RAM running Linux. The datasize limit was set to 1.5 GB.

Table 1 shows the results of applying four algorithms to LTL properties for which VIS did not already implement termination criteria. We compare the performance of our method based on Theorems 1 and 3, and the termination criterion (3a″) (*aut_sat*) to the standard BMC algorithm (*bmc*), to the use of the termination criterion (3a″) (*bmc_et*) only, and to the BDD-based LTL model checking algorithm in VIS (*ltl*).

Table 1. Comparison of aut_sat, bmc, bmc_et, and ltl

	state		aut_sat			bmc			bmc_et			ltl	
Model	vars	#	$\models$	k	$Time(s)$	$\models$	k	$Time(s)$	$\models$	k	$Time(s)$	$\models$	$Time(s)$
Am2910	99	1	yes	3	**3.03**	?	30	112.58	?	30	557.48	?	Timeout
Bakery	16	1	no	13	96.86	no	13	78.05	no	13	91.3	no	**0.35**
Blackjack	102	1	yes	1	**1.01**	?	30	148.76	yes	1	1.49	yes	282.6
Chameleon	7	1	yes	3	0.92	?	30	68.81	?	30	639.57	yes	**0.1**
Coherence	1	1	yes	3	2.32	?	30	40.31	?	30	178.46	yes	**0.7**
		2	no	5	4.32	no	5	0.99	no	5	2.19	no	**0.9**
		3	?	21	Timeout	?	30	1231.59	?	19	Timeout	yes	**1.0**
D18	506	1	no	23	1378.21	no	23	**82.68**	no	23	342.52	?	Timeout
		2	yes	0	**0.4**	?	30	123.45	yes	1	13.92	?	Timeout
D24	238	1	no	9	34.53	no	9	**15.12**	no	9	29.35	?	Timeout
Dcnew	10	1	no	6	3.03	no	6	0.88	no	6	1.94	no	**0.26**
		2	no	5	1.62	no	5	0.38	no	5	1.31	no	**0.3**
Dekker	6	1	no	5	2.61	no	5	0.86	no	5	1.31	no	**0.09**
Fabric	85	1	yes	17	**11.21**	?	30	21.57	?	30	116.95	yes	20.9
Feistel	293	1	yes	19	44.43	?	30	39.35	yes	19	35.54	yes	**0.6**
Lock	9	1	yes	7	1.16	?	30	24.4	?	30	359.63	yes	**0.13**
Microwave	4	1	no	2	0.21	no	2	0.05	no	2	0.11	no	**0.01**
		2	yes	3	0.34	?	30	5.87	yes	3	0.29	yes	**0.02**
		3	yes	7	0.95	?	30	16.72	yes	8	1.3	yes	**0.1**
MinMax	27	1	yes	5	13.16	?	30	183.28	?	24	Timeout	yes	**0.41**
Nim	33	1	no	6	19.54	no	6	**4.67**	no	6	11.76	no	464.1
Palu	37	1	no	0	0.1	no	0	**0.05**	no	0	**0.05**	?	Timeout
		2	no	1	0.41	no	1	**0.14**	no	1	**0.14**	?	Timeout
PI_BUS	307	1	yes	5	5.53	?	30	155.11	yes	6	15.68	yes	**1.76**
RetherRTF	43	1	no	2	1.31	no	2	**0.56**	no	2	0.79	no	1.03
		2	?	20	Timeout	?	30	1242.82	?	25	Timeout	no	**1.84**
		3	no	2	1.0	no	2	**0.54**	no	2	0.94	no	0.91
		4	?	25	Timeout	?	30	1014.33	?	28	Timeout	yes	**0.99**
		5	?	30	823.12	?	30	182.49	?	30	332.99	yes	**1.43**
s1269	37	1	yes	9	0.95	?	30	21.37	yes	9	**0.78**	?	Timeout
s1423	74	1	no	9	6.61	no	9	**2.55**	no	9	3.38	?	Timeout
		2	yes	3	0.57	?	30	12.21	yes	3	**0.49**	?	Timeout
Silvermau	17	1	yes	1	**0.14**	?	30	13.72	yes	1	0.16	yes	0.12
Smult	95	1	no	1	0.27	no	1	**0.07**	no	1	0.09	no	37.45
		2	?	30	446.67	?	30	7.52	?	30	382.23	yes	**35.47**
three_processor	48	1	yes	3	**2.5**	?	30	56.44	?	30	934.23	yes	184.6
Timeout	31	1	no	0	**0.06**	no	0	**0.06**	no	0	**0.06**	no	1.13
		2	no	2	0.76	no	2	**0.35**	no	2	0.42	no	1.64
UniDec	18	1	yes	3	2.76	?	30	143.84	yes	10	28.81	yes	**0.16**
		2	yes	8	12.6	?	30	112.87	yes	9	18.42	yes	**0.18**
		3	no	6	5.32	no	6	1.58	no	6	2.33	no	**0.36**
		4	no	6	6.93	no	6	3.47	no	6	3.7	no	**0.15**
UsbPhy	87	1	?	30	380.54	?	30	34.59	?	30	130.72	yes	**192.1**

Table 2. Comparison for special cases

Model	state vars	#	Theorem 1 and (3a″)			Special cases			Property type
			$\models$	k	$Time(s)$	$\models$	k	$Time(s)$	
Arbiter	16	1	?	30	432.6	?	30	**391.95**	Invariant
Blackjack	102	1	yes	1	2.4	yes	1	**1.01**	F p
Bpb	36	1	yes	3	4.72	yes	0	**0.35**	Safety
D4	230	1	?	30	356.7	yes	9	**6.87**	F p
D18	506	2	yes	1	14.66	yes	0	**0.4**	F p
D21	92	1	?	24	Timeout	?	24	Timeout	Invariant
D24	238	2	yes	21	532.24	yes	9	**45.25**	Invariant
Dekker	6	2	?	27	Timeout	yes	18	**266.96**	Invariant
Fabric	85	2	yes	19	15.32	yes	8	**3.76**	Invariant
FPMult	43	1	yes	7	4.36	yes	2	**0.35**	Safety
PI_BUS	307	1	yes	6	18.89	yes	5	**5.53**	F p
Rrobin	5	1	yes	3	0.2	yes	0	**0.02**	Safety
s1269	37	2	yes	5	2.26	yes	1	**0.22**	Invariant
Timeout	31	3	?	30	1023.22	yes	0	**0.07**	Invariant
		4	?	30	923.45	yes	16	**24.91**	Invariant
UniDec	18	2	yes	9	22.09	yes	8	**12.6**	F p
		5	yes	8	17.87	yes	8	**1.62**	Bounded LTL

The first column in Table 1 is the name of the model, the second is the number of state variables, and the third is the property number. The remaining columns are divided into four groups, one for each algorithm. The first column in each group indicates whether each property passes (*yes*), fails (*no*), or remains undecided (*?*); the column labeled k, when present, reports the length of the longest counterexamples that were considered. The columns labeled $Time$ give the times in second for each run. Boldface is used to highlight best run times.

As usual, SAT-based model checking does much better than BDD-based model checking on some examples, and much worse on others. Within the SAT-based approaches, *aut_sat* is the only one to prove (as opposed to falsify) a significant number of properties. In fact, all passing properties in Table 1 are proved by either *aut_sat* or *ltl*.

The termination criterion (3a″) is not very effective by itself. It only proves 11 of the 23 passing properties; 5 of them are proved by Theorems 1 and 3 for smaller values of k. By contrast, Theorems 1 and 3 prove 18 of the 23 passing properties.

Augmenting the standard BMC with the termination criteria of Theorems 1 and 3, and (3a″) helps to prove properties that are hard for the BDD-based method. In Table 1, *ltl* times out before deciding 9 properties, whereas *aut_sat* times out before deciding 3 properties. In addition, *aut_sat* proves some properties faster than *ltl*. For example, for model *Am2910*, *aut_sat* proves the property true in 3.03 s, while *ltl* does not reach a decision in 1800 s. As another example, for model *three_processor*, *aut_sat* proves the property true in 2.5 s, while *ltl* takes 184.6 s to prove it.

Table 2 illustrates the importance of checking for special cases. These include invariants, syntactically safe properties, bounded LTL properties, and liveness properties of the form $\mathsf{F}\, p$, where p is a propositional formula. All properties in this table are passing properties. The column labeled k has the same meaning as in Table 1. If the value of k is 0, the corresponding property is an *inductive* invariant.

In Table 2, the general method is slower. There are two reasons for that: The first is that using the termination criteria of Theorem 1 and (3a$''$) generate more clauses for a given value of k. The second reason is that longer counterexamples are examined. For instance, for *Fabric*, the general method needs 19 steps to prove the property, while the special case takes only 8 steps. As another example, *s1269* has a bounded depth of 1; however, the method based on Theorem 1 and (3a$''$) needs 5 steps to prove the property. The termination check of Theorem 3 is better than the termination check of Theorem 1 when checking properties of the form $\mathsf{F}\, p$. For example, for model *D4*, Theorem 1 fails to prove the property for k up to 30, while Theorem 3 proves it for k equal to 9 in only 6.87 s.

6 Conclusions and Future Work

We have presented an approach to proving general LTL properties with Bounded Model Checking even without prior knowledge of a tight bound on the completeness threshold of the graph [6]. The approach translates the LTL property to a Büchi automaton—as is customary in BDD-based LTL model checking—so as to apply a uniform termination criterion. Experiments indicate that this criterion is significantly more effective than the straightforward generalization of the termination criteria for invariants of [14]. Compared to the completeness threshold of [6], our bound takes into account the position of the fair states in the graph; hence, it may lead to much earlier termination. The experiments also underline the importance of detecting those cases for which special termination criteria are known. Comparison with BDD-based model checking shows a good degree of complementarity. Neither method proved uniformly better than the other, and together, the two could prove all passing properties in our set of experiments.

Our current implementation uses Büchi automata with labels on the transitions. As discussed in Sect. 4, we need to explore the alternative provided by automata with labels on the states as a way to cause earlier termination. Another aspect needing further attention is that our approach only considers Büchi automata with one fair set. Generalized Büchi automata can be converted to non-generalized, but it is not clear that this would be preferable to an extension of Theorem 1 to handle multiple fair sets.

Acknowledgment

The authors thank the referees for their suggestions, including an improved Theorem 3.

References

1. B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21:181–185, Oct. 1985.

2. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Fifth International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99)*, pages 193–207, Amsterdam, The Netherlands, Mar. 1999. LNCS 1579.
3. R. K. Brayton et al. VIS: A system for verification and synthesis. In T. Henzinger and R. Alur, editors, *Eighth Conference on Computer Aided Verification (CAV'96)*, pages 428–432. Springer-Verlag, Rutgers University, 1996. LNCS 1102.
4. A. Cimatti, M. Pistore, M. Roveri, and R. Sebastiani. Improving the encoding of LTL model checking into SAT. In *Proceedings of the Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 196–207, Venice, Italy, Jan. 2002. LNCS 2294.
5. E. Clarke, O. Grumberg, K. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *Proceedings of the Design Automation Conference*, pages 427–432, San Francisco, CA, June 1995.
6. E. Clarke, D. Kröning, J. Ouaknine, and O. Strichman. Completeness and complexity of bounded model checking. In *Verification, Model Checking, and Abstract Interpretation*, pages 85–96, Venice, Italy, Jan. 2004. Springer. LNCS 2937.
7. L. de Moura, H. Rueß, and M. Sorea. Bounded model checking and induction: From refutation to verification. In W. A. Hunt, Jr. and F. Somenzi, editors, *Fifteenth Conference on Computer Aided Verification (CAV'03)*, pages 1–13. Springer-Verlag, Boulder, CO, July 2003. LNCS 2725.
8. D. Kröning and O. Strichman. Efficient computation of recurrence diameters. In *Verification, Model Checking, and Abstract Interpretation*, pages 298–309, New York, NY, Jan. 2003. Springer. LNCS 2575.
9. O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Programming Languages*, pages 97–107, New Orleans, Jan. 1985.
10. K. L. McMillan. Interpolation and SAT-based model checking. In W. A. Hunt, Jr. and F. Somenzi, editors, *Fifteenth Conference on Computer Aided Verification (CAV'03)*, pages 1–13. Springer-Verlag, Berlin, July 2003. LNCS 2725.
11. M. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the Design Automation Conference*, pages 530–535, Las Vegas, NV, June 2001.
12. K. Ravi, R. Bloem, and F. Somenzi. A comparative study of symbolic algorithms for the computation of fair cycles. In W. A. Hunt, Jr. and S. D. Johnson, editors, *Formal Methods in Computer Aided Design*, pages 143–160. Springer-Verlag, Nov. 2000. LNCS 1954.
13. V. Schuppan and A. Biere. Efficient reduction of finite state model checking to reachability analysis. *Software Tools for Technology Transfer*, 5(2–3):185–204, Mar. 2004.
14. M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a SAT-solver. In W. A. Hunt, Jr. and S. D. Johnson, editors, *Formal Methods in Computer Aided Design*, pages 108–125. Springer-Verlag, Nov. 2000. LNCS 1954.
15. A. P. Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects in Computing*, 6:495–511, 1994.
16. 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. Springer-Verlag, Berlin, July 2000. LNCS 1855.
17. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331, Cambridge, UK, June 1986.
18. URL: http://vlsi.colorado.edu/∼vis.