Termination Criteria for Bounded Model Checking: Extensions and Comparison [1]

Mohammad Awedh [2] Fabio Somenzi [3]

University of Colorado at Boulder

Abstract

Increasing attention has been paid recently to criteria that allow one to conclude that a structure models a linear-time property from the knowledge that no counterexamples exist up to a certain length. These termination criteria effectively turn Bounded Model Checking into a full-fledged verification technique and sometimes result in considerable time savings. In [1] we presented a criterion based on the translation of the linear-time specification into a Büchi automaton. BMC can be terminated if no fair cycle is found up to a given length, and one can prove that no fair cycle exists beyond that length. The maximum length for which counterexamples are explicitly checked is called the *termination length*; it obviously depends on the model, the property, and the termination criterion. In this paper we improve the criterion of [1] by adding a check that often substantially reduces termination length. Our previous work employed translation to a *non-generalized* Büchi automaton. Though a well-known technique converts a *generalized* automaton into that form by composing it with a counter, it has the undesirable effect of considerably lengthening the cycles in the graph to be searched. We propose several alternatives to that approach and compare them experimentally. The translation to automata can be accomplished in more than one way, and in this paper we contrast two of them: one based on the algorithms of [18], and one based on the notion of *tight automaton* of [5]. The latter yields shorter counterexamples, but the former often leads to earlier termination. In addition, it can help in identifying safety properties, for which termination checks are much more efficient than for the general case. We finally present results on comparing techniques based on cycle detection to the technique of [13], which converts liveness properties into safety properties by augmentation of the model.

Key words: bounded model checking, Büchi automata, safety and liveness properties, termination conditions.

[1] This work was supported in part by SRC contract 2004-TJ-920.

[2] Email: Awedh@Colorado.EDU

[3] Email: Fabio@Colorado.EDU

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

1 Introduction

Bounded Model Checking (BMC, [3]) is a model checking approach for linear time properties typically expressed in Linear Time Logic (LTL). BMC reduces the search for a counterexample to an LTL property to propositional satisfiability (SAT). Given a Kripke structure $\mathcal{M}$, an LTL formula ψ, and a bound k, BMC tries to refute $\mathcal{M} \models \mathsf{A}\,\psi$ by proving the existence of a witness of length k to $\neg\psi$. That is, BMC tries to find a witness to $\mathcal{M} \models_k \mathsf{E}\,\neg\psi$. The k-bounded witness to $\mathcal{M} \models_k \mathsf{E}\,\neg\psi$ is a path in $\mathcal{M}$ with at most k states. It can be either a finite prefix of a path for a safety property or a looping path (a k-loop) in the general case.

The standard technique to check an LTL property [22,11] constructs a Büchi automaton that accepts all the counterexamples to the LTL formula, and then checks the composition of the property automaton and the original model for language emptiness. The size of the automaton is exponential in the length of the LTL property, and this technique is in PSPACE [17]. Language emptiness is often checked by a BDD-based fixpoint computation.

BMC is known to be a complementary method to the BDD-based LTL model checking: Many problems that are hard for the BDD-based method can be solved easily by BMC [7]. However, it is hard to predict in advance the cases where BMC is more efficient than the BDD-based method [19].

The original Bounded Model Checking [3], although complete in theory, is limited in practice to falsification of LTL properties. BMC can prove that an LTL property ψ passes on a model $\mathcal{M}$ only if a bound, κ, is known such that, if no counterexample of length up to κ is found ($\mathcal{M} \not\models_\kappa \mathsf{E}\,\neg\psi$,) then $\mathcal{M} \models \mathsf{A}\,\psi$. Several methods exist to compute a suitable κ, all of them depending on $\mathcal{M}$, ψ, and the BMC encoding scheme. Some methods are straightforward, but are usually poor. (For an invariant property, one could use an upper bound on the number of reachable states as κ.) The optimum value of κ, however, is usualy very expensive to obtain: Finding it is at least as hard as checking whether $M \models \mathsf{A}\,\psi$ [6].

Several practical approaches that over-approximate the value of κ are based on the recurrence diameter/radius of the model [3]. In [15], the authors use the forward and backward recurrence radii to prove invariant properties. Their approach 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. 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. In this approach the forward and backward recurrence radii can be found by solving a sequence of SAT instances rather than QBF instances. Hence, they are easier to compute. However, the bound based on recurrence radii is not as tight as the one based on the radii.

The approach of [12] does not explicitly compute the backward radius, but it only examines counterexamples of length up to it to prove termination in invariant checking. For a given value of k, the algorithm iterates over approximations of

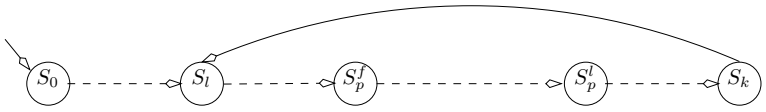

Fig. 1. Path to a fair cycle

the states that are reachable from the initial states, R. The algorithm starts by setting R to be the set of initial states. At each iteration of the algorithm, an over-approximation of the states that are reachable from R in one step is added to R. If no more states are added to R and R does not intersect the target states, the safety property holds in $\mathcal{M}$. It can be shown that k is less than or equal to the backward radius of $\mathcal{M}$.

For safety properties one can compose a suitable automaton with the model to be verified. In this way, model checking is reduced to reachability and the termination criteria for invariants can be applied. General LTL properties, on the other hand, require the ability to prove language emptines (i.e., detect cycles), and therefore call for different approaches to check termination.

One such approach [2] involves a translation that reduces the check for simple liveness to the check for an invariant on an augmented model. The translation adds two components to the model: one for loop detection and the other to monitor the property. This addition doubles the state variables and correspondingly increases the number of reachable states and the length of the longest shortest/simple paths.

A tight bound on proving simple liveness properties is proved in [13]; the upper bound for checking a simple liveness property in BMC is ($\neg p$-predicated radius + $\neg p$-predicated diameter). The $\neg p$-predicated radius (diameter) is the radius (diameter) of the model after deleting all states in which p holds.

In [1], we presented a termination criterion based on the observation illustrated in Figure 1: A counterexample to $\mathsf{A}\,\mathsf{F}\,\mathsf{G}\,\neg p$ consists of two paths: A simple path form an initial state S_0 to a state that satisfies p (S_p^l in Fig. 1), and a simple path leading back to a state that satisfies p (S_p^f in Fig. 1) along which all other states satisfy $\neg p$. If no counterexample is found up to the sum of the bounds on the above simple paths, then $\mathsf{A}\,\mathsf{F}\,\mathsf{G}\,\neg p$ holds. Our termination criterion takes into account the position of the states that satisfy the fairness constraint p and does not augment the model.

After the preliminaries of Section 2, in Section 3 we improve on the criterion of [1] by introducing an additional check that often reduces the length of counterexamples that must be explicitly examined. In Section 4 we study the effect of the translation from LTL to automata on the length of counterexamples and termination. In Section 5 we extend the technique to deal directly with generalized Büchi automata. The results of our experiments are reported in Section 6, and conclusions are offered in Section 7.

2 Preliminaries

LTL Model Checking is the problem of checking whether a specification, expressed by an LTL formula ψ, holds on all paths of a model $\mathcal{M}$, $\mathcal{M} \models \mathsf{A}\,\psi$. The *LTL formulae* over atomic propositions AP are defined as follows

- Atomic propositions, true, and false are LTL formulae.
- If ψ and ϕ are LTL formulae, then so are $\neg\psi$, $\psi \wedge \phi$, $\psi \vee \phi$, $\mathsf{X}\,\psi$, and $\psi\,\mathsf{U}\,\phi$.

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

In LTL model checking, the behavior of the model is usually 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, i.e., 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 model. 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 $p \in L(s)$.

A path π in $\mathcal{K}$, whether finite or infinite, is a non-empty sequence $(s_0, s_1, \ldots)$ of states in $\mathcal{K}$ such that $\delta(s_i, s_{i+1})$ for all $0 \leq i < |\pi|$, where $|\pi|$ is the path length. We let $\pi(i) = s_i$ be the i-th state of π, $\pi_i = (s_0, \ldots, s_i)$ be the prefix of π and $\pi^i = (s_i, s_{i+1}, \ldots)$ be the suffix of π.

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 $i \geq 0$ such that $\pi^i \models g$, and for $j < i$, $\pi^j \models f$.

The *diameter* d of $\mathcal{K}$ is the length of the longest shortest path in $\mathcal{K}$. That is, the diameter is the maximum finite distance between two states. The (forward) *radius* r of $\mathcal{K}$ is the maximum finite distance in $\mathcal{K}$ of a state from the closest states in I. In other words, the radius is the maximum number of forward transitions needed to reach a state reachable from the intitial states. The *backward radius* is the maximum number of backward transitions from a given set of states needed to reach a state backward-reachable from those states.

A simple path between any two states s and t in $\mathcal{K}$ is a cycle free path. The *recurrence diameter* rd of $\mathcal{K}$ is the longest simple path in $\mathcal{K}$. The *recurrence radius* rr of $\mathcal{K}$ is the longest simple path in $\mathcal{K}$ that starts form a state in I.

A finite sequence of states can represent an infinite path if it contains a loop. A path $(s_0, \ldots, s_k)$ is a (k,l)-loop path if there is a transition from state s_k to state s_l for some $l \leq k$. A path is a k-loop path if it is a (k,l)-loop path for some $l \leq k$.

A *safety* property is such that every counterexample to it has a finite prefix that, however extended to an infinite path, yields a counterexample. Sistla [16] provides a syntactic characterization of LTL safety formulae: Every propositional formula

is a safety formula, and if f and g are safety formulae, then so are $f \vee g$, $f \wedge g$, $\mathsf{X}\, f$, $\mathsf{G}\, f$, and $f\, \mathsf{R}\, g$. Not all safety properties are captured by this definition.

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.

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 $i \geq 0$, $(\rho_i, w_i, \rho_{i+1}) \in \Delta$. A run ρ is *accepting* if there exists $q_j \in F$ that appears infinitely often in ρ. Every LTL formula ψ can be translated into a Büchi automaton $\mathcal{A}_\psi$ such that $\mathcal{A}_\psi$ accepts exactly the paths that satisfy ψ [9,18]. In the automata-based approach to LTL Model Checking [20], a Büchi automaton that accepts counterexamples to the LTL formula is constructed. Then, the existence of an initialized fair cycle in the composition of the model and that automaton indicates failure of the specification.

3 An Improved Criterion for Language Emptiness

Let $\mathcal{K} = \langle S, \delta, I, L \rangle$ be a Kripke structure, and let $p \in AP$ be an atomic proposition. Let $path_k$ ($simplePath_k$) be the predicate that is true if $(s_0, \ldots, s_k)$ is a (simple) path in $\mathcal{K}$. The method of [1] is based on the following result.

Theorem 3.1 *Let these predicates over $s_0, \ldots, s_k$ denote sets of paths in $\mathcal{K}$:*

$$\alpha_k = I(s_0) \wedge simplePath_k \wedge p(s_k) \tag{1a}$$

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

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

$$[\![\mathcal{K}, \neg\, \mathsf{F}\, \mathsf{G}\, \neg p]\!]_k = I(s_0) \wedge path_k \wedge \bigvee_{0 \leq l \leq k} [\delta(s_k, s_l) \wedge \bigvee_{l \leq i \leq k} p(s_i)] \ . \tag{1c}$$

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

With reference to Fig. 1, n bounds the length of the path from S_0 to S_p^l, while m bounds the length of the path from S_p^l to S_p^f.

The value of m in Theorem 3.1 can be unnecessarily large in certain structures. As an example, consider Figure 2, in which some unreachable states of a structure are shown. If S_8 is the only state satisfying p, then $m = 8$ because of the simple path from S_0 to S_8. However, such a path cannot be used to close the loop (from S_p^l to S_p^f in Fig. 1) because its initial state does not satisfy p. The longest simple path whose first state satisfies p, such that p does not hold in any subsequent state

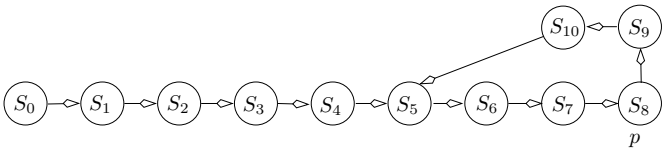

Fig. 2. Capturing the value of m

has length 5; it is therefore sufficient to take $m = 5$. To capture this observation, we add (1b″) to Theorem 3.1.

$$\beta''_k = simplePath_{k+1} \wedge p(s_0) \wedge \bigwedge_{1 \le i \le k+1} \neg p(s_i) \qquad \text{(1b″)}$$

Predicate (1b″) does not replace (1b′): If we move p in Figure 2 from state S_8 to state S_2, β'_k will capture the smallest value of m. In general, if the distance between p-states in the loops is shorter than the stem paths leading to them, β'' will be more effective than β'; conversely, when the p states appear only on the stem paths, β' will often give the smaller value of m. Hence, we let m be the minimum value of k for which either β'_k or β''_k becomes unsatisfiable.

Termination criteria benefit in general from knowledge of what states are unreachable. Performing full reachability analysis would make subsequent BMC redundant and is often too costly. However, reachability analysis of the Büchi automaton is usually quite cheap and identifies a subset of the unreachable states of the composite model. This subset can later be used as a constraint on the paths examined to decide termination by requesting that no state in those paths belong to the unreachable subset.

4 Tight Büchi Automata

In [14], the authors show that the construction of [5] yields a Büchi automaton that is *tight*, and therefore guarantees that shortest counterexamples in the composition of automaton and model will map onto shortest counterexamples in the original model. This property is not shared by the Büchi automata produced by the translation of [18] which we used in [1] to check for termination. For that reason, the algorithm proposed in [1] used the automaton approach only for termination check; for counterexample detection it used the BMC encoding of [3].

It is then natural to ask whether tight automata would benefit the scheme of [1], since they would allow one to unify the models used for proof and falsification of a property. We shall show, however, in Section 6 that most of the times the use of tight automata increases the termination length, often substantially.

One reason for this increase in termination length, which comes from increases of both m and n in Theorem 3.1, is to be found in the sharp increase in the number of states and transitions that the use of tight automata causes. In addition, a tight automaton has exactly one acceptance condition for each until operator in the LTL formula. By contrast, approaches like the one of [18] often reduce the number of acceptance conditions. A final important reason is related to the notion of automa-

ton strength [4]. Tight automata are almost always strong, while the automata of [18] are mostly weak and terminal. It can be shown that for terminal automata $m = 0$ in Theorem 3.1. For weak automata, this property does not hold, but experiments show that $m = 0$ most of the time. (In both cases, it is the β'' predicate that is responsible for these low values.)

Terminal automata accept co-safety languages. Since we detect properties that are syntactically safety properties, we apply the termination criterion for invariants to most safety properties even without resort to the strength of the automaton. However, the occasional safety property will fail the syntactic check, and, more importantly, properties producing weak automata are common, and are handled by our language emptiness criterion. A special criterion for weak properties can be devised; however, its impact on performance is small.

Tight automata are also to be considered as replacement of the traditional BMC encoding of [3] for counterexample detection. In fact, we have found that the encoding of [8,10], which are closely related to the use of tight automata, often improve the speed of counterexample search.

5 Checking Multiple Fairness Conditions

The approach of Theorem 3.1 for checking language emptiness only considers Büchi automata with one fair set. Büchi automata with multiple fair sets are known as *generalized* Büchi automata. The acceptance condition of a generalized Büchi automaton is a set of fair sets $\mathcal{F} \subseteq 2^Q$. A run of a generalized Büchi automaton is accepting if some state p_i of each of the sets $F_i \in \mathcal{F}$ appears infinitely often.

One solution to handle multiple fair sets $\mathcal{F} = \{F_1, \ldots, F_r\}$ is to convert a generalized Büchi automaton $\mathcal{G}$ to an equivalent Büchi automaton $\mathcal{A}$. The standard construction composes a counter with $\mathcal{G}$. The counter has $r = |\mathcal{F}|$ states, is initialized to 1, and is incremented from i to $i + 1$ when a state in F_i is visited. It is reset from r to 1 when a state in F_r is visited. A fair state in $\mathcal{A}$ is a state in which $\mathcal{G}$ satisfies F_1 and the counter's value is 1.

The conversion expands the size of the automaton by a factor related to the number of fair sets. The effectiveness of our termination criterion depends on the order at which the fair sets in $\mathcal{F}$ are visited and in general is reduced as the number of fair sets increases because the fair cycles of the non-generalized automaton tend to grow longer and longer.

To counter this effect, instead of using a counter, we add a state variable x_j for each fair set F_j which keeps track of when a state in F_j is visited.

$$\begin{aligned} x_j(0) &= p_j(0) \\ x_j(t) &= p_j(t) \vee x_j(t-1) \wedge \neg x(t-1) \ . \end{aligned}$$

A fair state in the Büchi automaton is one thata satisfies $x = \bigwedge_{j=1}^{r} x_j$. We can then check the conditions of Theorem 3.1 on x. We call this method the *Flag* method.

Another solution is to extend Theorem 3.1 to handle multiple fair sets. One way

Table 1
Using β''

Model	β'			β''			Both		
	m	n	$T(s)$	m	n	$T(s)$	m	n	$T(s)$
Arbiter	7	-	128.7	6	-	**117.8**	6	-	130.3
Abp	2	-	215.2	2	-	**214.7**	2	-	270.3
D4	11	11	37.37	0	11	29.99	0	11	**28.33**
Fabric1	8	8	2.71	0	8	1.28	0	8	**1.26**
Fabric2	8	8	32.15	0	8	20.69	0	8	**20.33**
Feistel	2	9	2.77	0	9	2.63	0	8	**2.59**
FPMult	3	3	3.26	0	3	2.67	0	3	**2.49**
Huffman1	1	1	21.83	0	1	17.29	0	1	**17.01**
Huffman2	3	3	54.25	0	3	**42.56**	0	3	43.01
Huffman3	10	10	100.57	0	10	14.24	0	10	**13.68**
Miim1	3	3	0.17	0	3	**0.14**	0	3	0.17
Miim2	-	-	109.59	0	-	**108.37**	0	-	108.97
PPC60X_bus	-	-	TO	0	-	**1002.2**	0	-	1010.93
Smult	-	-	8.95	0	-	8.77	0	-	**8.64**
TicTacToe1	11	11	476.22	0	11	**261.75**	0	11	262.02
TicTacToe2	1	1	518.78	0	1	**423.68**	0	1	424.42
Tlc	0	15	**2.07**	0	15	2.13	0	15	2.12
Vsa16	-	-	**283.03**	0	-	296.61	0	-	294.33

to achieve that is to check language emptiness for the fairness condition $\bigcup_{F\in\mathcal{F}} F$. This approach, which we call the *Or* approach, will in general be conservative in estimating the termination length, but does not require any increase in the number of states of the automaton. Besides, taking the union of the fair sets may decrease the distance between fair states along the loops and hence reduce the value of m.

Yet another way to deal with multiple fairness constraints is to apply Theorem 3.1 to each fairness condition in turn. This *One* method also produces conservative values for m and n, but compared to the *Or* approach, will be more effective when one fairness constraint cannot be satisfied in isolation.

6 Experimental Results

The results presented in this section are for models that are from industry, and from the Texas-97 and VIS Verification Benchmark sets [21]. For each model, we count each LTL property as a separate experiment. For all experiments, we set the maximum value of k to 30 and we check for termination at each step. The

Table 2
Using Automaton Reachability analysis

Model	*No*	*Yes*	Model	*No*	*Yes*
CacheCo	29.28	**21.68**	D4	29.31	**26.34**
Ethernet	109.53	**96.7**	Heap	513.12	**421.92**
HourGlass	504.55	**478.87**	Needham	1540.1	**1369.8**
PI_BUS	377.75	**366.92**	ProdCell	**85.47**	118.44
ReqAck	**101.85**	113.27	TwoFifo	95.75	**87.98**
TwoQ	235.37	**225.58**	VsaR1	**353.99**	366.61

Table 3
Comparing tight and non-tight automata

Model	St	Strength	Non-tight			Tight		
			m	n	$T(s)$	m	n	$T(s)$
Coherence	U	strong	17	-	TO	24	-	TO
Ifetch1	P	weak	0	2	**0.31**	-	-	25.77
Ifetch2	P	terminal	0	3	**0.15**	-	-	26.78
FPMult1	P	terminal	0	3	**5.77**	2	-	25.7
FPMult2	P	terminal	0	3	**2.79**	2	-	30.79
Microwave	P	weak	0	0	**0.1**	0	8	0.3
Pathfinder	P	weak	0	0	**0.1**	0	-	22.00
PI_BUS	P	weak	0	1	**0.57**	0	-	945.33
ReqAck	U	weak	0	-	**20.15**	-	-	27.22
s1269-1	P	weak	0	8	**0.22**	0	9	0.39
s1269-2	P	weak	0	8	**0.20**	0	16	1.1
s1269-3	P	terminal	0	1	**0.09**	0	-	81.39
s1423	P	terminal	0	4	0.16	3	4	**0.13**
UsbPhy	P	weak	0	-	143.9	1	-	**88.55**

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.

The first column in every table is the name of the model. The columns labeled m and n in each table give the values of m and n in Theorem 3.1 respectively; if an entry in these columns is a dash, it indicates that no value is captured. The columns labeled T give the runtimes in seconds; boldface is used to highlight best runtimes; a TO in this column indicates a time greater than 1800 s. CPU times in all tables are for both counterexample detection and termination checks.

In Table 1 one sees that applying β'' reduces the value of m. For model *D4*, for instance, the value of m that is captured by β' is 11 while the value that is captured

Table 4
Safety properties

Model	General LTL			Safety		
	St	tl	$T(s)$	St	tl	$T(s)$
Fabric	P	8	20.42	P	8	**17.51**
Huffman1	P	1	17.36	P	1	**11.56**
Huffman2	P	3	43.2	P	3	**31.82**
Huffman3	P	11	17.05	P	10	**9.85**
Lock	U	-	750.45	U	-	**129.34**
Rrobin	U	-	TO	U	-	**160.19**
VsaR	P	5	361.83	P	5	**285.69**

by β'' is 0. Hence, the search for counterexample will stop after k=10 if β'' is used compare to k=21 if only β' is used. In many cases the overhead for checking β'' in addition to β' is well within the noise margin.

Table 2 compares the use of reachability analysis of the automaton when searching for the values of m and n (columns labeled *Yes*) to its omission (columns labeled *No*). Reachability analysis of the automaton usually reduces runtime, but it does not help in reducing the values of m and n.

Table 3 compares tight to non-tight *Büchi* automata when searching for a simple path. The column labeled St in this table indicates whether each property passes (*P*), or remains undecided (*U*). The column labeled $Strength$ is the automaton strength. From this table, we can conclude that using tight automata increases the termination length. The termination length for the model *s1423* increases from 3 to 6 when using a tight automaton. In model *Coherence*, the value of m is increased from 17 to 24 when applying a tight automaton. In model *Ifetch*, using a tight automaton does not even capture the value of m for a given value of k.

Table 4 illustrates the importance of a dedicated criterion for safety properties. All properties in this table are passing properties. The column labeled St has the same meaning as in Table 3; the column labeled tl, when present, reports the termination length.

Tables 5 and 6 show the results of applying different methods when handling multiple fairness conditions. The columns headings identify one of the *Or*, *One*, *Flag*, *Counter*, and *Trans* methods. The last one is the method that applies the translation of liveness into safety [2]. Safety checking is performed using both Bounded Model Checking algorithm (*bmc*) and the BDD-based LTL model checking algorithm in VIS (*ltl*).

Table 5 compares the first three methods. Table 6 compares all methods on a smaller set of examples, because for *Counter* and *Trans* we manually translated the examples. The upper part of Table 6 shows the results of applying the methods *Or*, *One*, and *Flag*. The lower part shows the results of applying the *Counter* and *Translation* methods.

Table 5
Comparison of Or, One, and Flag

Model	# fair	OR			One			Flag		
		m	n	$T(s)$	m	n	$T(s)$	m	n	$T(s)$
Am2910-1	1	0	1	6.02	0	1	6.62	0	1	**5.12**
Am2910-2	1	0	-	**43.24**	0	-	46.35	0	-	45.35
Arbiter	3	6	-	298.77	0	-	105.95	-	-	**32.51**
Chameleon	5	0	-	TO	0	1	0.1	0	0	**0.02**
Cups	7	2	-	TO	0	1	0.19	0	0	**0.04**
D12	6	0	-	22.33	0	3	**0.71**	6	7	1.17
D16	4	-	-	TO	0	1	**1.49**	-	-	TO
Dcnew	2	2	-	TO	0	0	**0.7**	0	-	TO
Nim	2	1	-	86.21	0	1	**0.14**	1	-	49.35
NulMdm	2	-	-	133.11	0	1	196.67	-	-	**107.29**
Philo	11	6	-	TO	0	-	TO	0	0	**0.03**
PnPong1	3	1	9	**1.36**	0	6	1.47	3	11	5.3
PnPong2	3	1	5	0.14	0	1	**0.12**	1	4	0.15
Pong-1	3	-	-	99.32	0	-	287.1	1	5	**0.35**
Pong-2	3	-	-	336.25	0	-	438.25	-	-	**318.42**
ReqAch	2	1	-	3.79	0	-	5.19	2	-	**3.12**
Rether	5	0	-	59.15	0	-	379.02	0	0	**0.07**
Rether	5	0	-	29.16	0	-	392.42	0	0	**0.09**
Short	3	0	2	0	0	0	0	0	0	0
Soap	2	-	-	TO	0	1	**0.35**	-	-	1523.43
Tlc-1	3	0	9	**0.37**	3	14	21.15	2	18	35
Tlc-2	2	0	15	**2.11**	0	15	6.9	4	18	13.95
Vendng1	2	14	-	TO	0	-	TO	13	-	**1041.63**
Vendng2	2	2	2	0.14	0	2	**0.11**	-	-	TO
Vendng3	2	2	19	1240.29	0	2	**0.18**	13	-	TO

In Table 5, 20 out of 25 properties are decided passed by at least one method within the given limit of time and value of k. Both methods *One* and *Flag* are the fastest in 8 experiments; *Or* is the fastest in only 3 experiments. No methods dominates the other in reducing the length of m and n. In model *PnPong1*, the *One* method captures the smallest values of m and n. While in model *Tlc-1*, the *Or* method captures the smallest values of m and n. The *Flag* method proves properties pass in zero value of k in 6 experiments.

Table 6 shows that, the values of m and n that are found by the *Flag* method

Table 6
Comparison of Or, One, Flag, Counter, and Trans.

Model	# fair	Or			One			Flag		
		m	n	$T(s)$	m	n	$T(s)$	m	n	$T(s)$
Crd1	5	0	14	8.5	0	14	25.05	7	18	212.64
Crd2	5	0	26	720.51	0	1	0.05	7	-	659.39
$\mu 1$	2	4	6	0.15	0	6	0.64	0	9	0.39
$\mu 2$	2	0	0	0	0	1	0.01	5	6	0.11
$\mu 3$	2	3	4	0.3	0	3	0.12	5	7	1.84
Pong	3	-	-	101.86	0	-	299.74	1	5	0.35
Short	3	0	2	0	0	0	0	0	0	0

Model	Counter			Trans(bmc)		Trans(ltl)
	m	n	$T(s)$	k	$T(s)$	$T(s)$
Crd1	18	19	66.73	22	493.25	0.21
Crd2	22	-	128.73	30	890.56	0.15
$\mu 1$	9	12	1.6	13	5.32	0.05
$\mu 2$	7	9	0.22	18	31.56	0.05
$\mu 3$	6	11	1.91	19	92.07	0.09
Pong	-	-	36.88	30	48.71	TO
Short	4	4	0.02	10	0.57	0.01

are smaller than those found by the *Counter* method. This is because the *Counter* method depends on the order in which the fair sets are visited, while the *Flag* method does not.

The *Trans* method blows up the state space and increases the diameter and radius of the model. In almost all the experiments in Tables 6, the *Trans* method, using *bmc*, is the slowest one. In almost all these experiments, *ltl* is very fast. We have not tried large models yet with the *Tran* and *Counter* methods because they required considerable manual work to convert them, but we expect that for larger examples *bmc* will prove faster more often.

7 Conclusions

We have presented an improved criterion for termination in Bounded Model Checking, which significantly reduces termination length. An improvement to our termination check could be restricting the search using the *Flag* method to paths that end in a state that satisfies at least one fairness constraint. We are currently evaluating this improvement.

We have also shown that the use of the reachability analysis of the property

automaton speeds up the termination check, but does not reduce the termination length. Even though tight automata find shortest k-loop counterexamples, they increase the termination length.

We have presented different methods for checking multiple fairness conditions when checking language emptiness using BMC. The *Flag* and *One* methods are the best among them. Both *Flag* and *Counter* methods are based on recording the visiting of fair states. However, the performance of the *Counter* method depends on the order of visit of the fair sets. The *Flag* method helps limiting the search for a simple path to the ones satisfy all fairness constraints. Hence, it helps finding small values for m and n, but it may increases the searching time.

The efficiency of the *One* method also depends on the order in which fair sets are checked; in practice, we found that it is more efficient to check the fair states that come from the property automaton before those supplied with the model.

References

[1] M. Awedh and F. Somenzi. Proving more properties with bounded model checking. In R. Alur and D. Peled, editors, *Sixteenth Conference on Computer Aided Verification (CAV'04)*, pages 96–108. Springer-Verlag, Berlin, July 2004. LNCS 3114.

[2] A. Biere, C. Artho, and V. Schuppan. Liveness checking as safety checking. *Electronic Notes in Theoretical Computer Science*, 66(2), July 2002. Formal Methods for Industrial Critical Systems (FMICS'02).

[3] 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.

[4] R. Bloem, K. Ravi, and F. Somenzi. Efficient decision procedures for model checking of linear time logic properties. In N. Halbwachs and D. Peled, editors, *Eleventh Conference on Computer Aided Verification (CAV'99)*, pages 222–235. Springer-Verlag, Berlin, 1999. LNCS 1633.

[5] E. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. In D. L. Dill, editor, *Sixth Conference on Computer Aided Verification (CAV'94)*, pages 415–427. Springer-Verlag, Berlin, 1994. LNCS 818.

[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] F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M. Y. Vardi. Benefits of bounded model checking at an industrial setting. In G. Berry, H. Comon, and A. Finkel, editors, *Thirteenth Conference on Computer Aided Verification (CAV'01)*, pages 436–453. Springer-Verlag, Berlin, July 2001. LNCS 2102.

[8] A. Frisch, D. Sheridan, and T. Walsh. A fixpoint based encoding for bounded model checking. In M. D. Aagaard and J. W. O'Leary, editors, *Formal Methods in Computer Aided Design*, pages 238–255. Springer-Verlag, Nov. 2002. LNCS 2517.

[9] R. Gerth, D. Peled, M. Y. 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] T. Latvala, A. Biere, K. Heljanko, and T. Junttila. Simple bounded LTL model checking. In *Formal Methods in Computer Aided Design*, pages 186–200, Austin, TX, Nov. 2004. Springer. LNCS 3312.

[11] 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.

[12] 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.

[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] V. Schuppan and A. Biere. Shortest counterexamples for symbolic model checking of LTL with past. In *International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'05)*, pages 493–509, Edinburgh, UK, Apr. 2005.

[15] 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.

[16] A. P. Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects in Computing*, 6:495–511, 1994.

[17] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logic. *Journal of the Association for Computing Machinery*, 3(32):733–749, 1985.

[18] 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.

[19] O. Strichman. Accelerating bounded model checking of safety properties. *Formal Methods in System Design*, 24(1):5–24, Jan. 2004.

[20] 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.

[21] URL: http://vlsi.colorado.edu/∼vis.

[22] P. Wolper, M. Y. Vardi, and A. P. Sistla. Reasoning about infinite computation paths. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, pages 185–194, 1983.