Simple Bounded LTL Model Checking

Timo Latvala[1,⋆], Armin Biere[2], Keijo Heljanko[1,⋆⋆], and Tommi Junttila[3,⋆⋆⋆]

[1] Laboratory for Theoretical Computer Science
Helsinki University of Technology
P.O. Box 5400, FI-02015 HUT, Finland
{Timo.Latvala,Keijo.Heljanko}@hut.fi
[2] ETH Zürich, Computer Systems Institute
CH-8092 Zürich, Switzerland
biere@inf.ethz.ch
[3] ITC-IRST
Via Sommarive 18, 38050 Povo, Trento, Italy
junttila@irst.itc.it

Abstract. We present a new and very simple translation of the bounded model checking problem which is linear both in the size of the formula and the length of the bound. The resulting CNF-formula has a linear number of variables *and* clauses.

Keywords: bounded model checking, LTL, linear translation, NuSMV.

1 Introduction

Bounded model checking [1] (BMC) is a technique for finding bugs in finite state system designs violating properties specified in linear temporal logic (LTL). The method works by mapping a *bounded* model checking problem to the satisfiability problem (SAT). Given a propositional formula encoding a Kripke structure M representing the system, an LTL formula ψ and a bound k, a propositional formula $||[M,\psi,k]||$ is created that is satisfiable if and only if the Kripke structure M contains a counterexample to ψ of length k.

BMC has established itself as a complementary method to symbolic model checking methods based on (ordered) binary decision diagrams (BDDs). The biggest advantage of BMC compared to BDDs is its space efficiency; there are some Boolean functions which cannot be succinctly encoded as a BDD. BMC also produces counterexamples of minimal length, which eases their interpretation and understanding for debugging purposes. However, predicting the cases where BMC is more efficient compared to

⋆ Work supported by the Helsinki Graduate School in Computer Science, the Academy of Finland (project 53695), and the Nokia Foundation.

⋆⋆ Work partially supported by FET project ADVANCE contract No IST-1999-29082, EPSRC grant 93346/01 (An Automata Theoretic Approach to Software Model Checking), and the Academy of Finland (project 53695 and grant for research work abroad).

⋆⋆⋆ This work has been sponsored by the CALCULEMUS! IHP-RTN EC project, contract code HPRN-CT-2000-00102, and has thus benefited of the financial contribution of the Commission through the IHP programme.

A.J. Hu and A.K. Martin (Eds.): FMCAD 2004, LNCS 3312, pp. 186–200, 2004.

BDD-based methods is difficult [2]. Furthermore, BMC is an incomplete method unless we can determine a value for the bound k which guarantees that no counterexample has been missed. Several papers [1, 3, 4] have investigated techniques for computing this bound.

The two main ways of improving the performance of BMC is either to improve solver technology or to modify the encoding of the problem to SAT. Improvements of the second kind usually rely on the appealing idea that simpler is better. The intuition is that an encoding which results in fever variables and clauses is usually easier to solve. We present a new simple encoding for the BMC problem which is linear in the bound, the system description (i.e. the size of the transition relation as a propositional formula) and the size of the specification as an LTL formula. The resulting propositional formula has both a *linear number of variables and clauses.*

We have experimentally evaluated our new encoding. Our experiments compare the sizes of the encodings and the required time to solve the instances.

2 Bounded Model Checking

In bounded model checking we consider finite sequences of states in the system, while LTL formulas specify the infinite behaviour of the system. The key observation by Biere et al. [1] was that a finite sequence can still represent an infinite path if it contains a loop. An infinite path $\pi = s_0 s_1 s_2 \ldots$ is a (k,l)-loop if there exists integers l and k such that $s_{l-1} = s_k$ and $\pi = (s_0 s_1 \ldots s_{l-1})(s_l s_{l+1} \ldots s_k)^\omega$ (we also use the term k-loop). A bounded path $s_0 s_1 \ldots s_k$ of length k can either have $k+1$ unique states or represent an infinite path with a (k,l)-loop if $s_k = s_{l-1}$ for some $1 \leq l \leq k$. This can actually be interpreted in two different ways (corresponding to the same infinite path π). Either the back edge of the loop is from s_{k-1} to s_{l-1} (the dashed back edge in Fig. 1) or the back edge is from s_k to s_l (the solid back edge in Fig. 1). The new *loop shape* depicted on the right side of Fig. 1 requires $k > 0$ for k-loops, which we will silently assume for the rest of the paper. This loop shape allows a more compact translation than the one in [1], replacing the $k+1$ copies in the original translation for closing the loop by k comparisons between bit vectors encoding states.

When k is fixed there are $k+1$ different loop possibilities for a bounded path. There are k different (k,l)-loops and it is of course also possible that no loop exists. The basic idea of Biere et al. [1] was to write a formula which is satisfiable iff the path is a model of the negation of the LTL specification, for each of these cases. The complete translation simply joins the cases in one big disjunction.

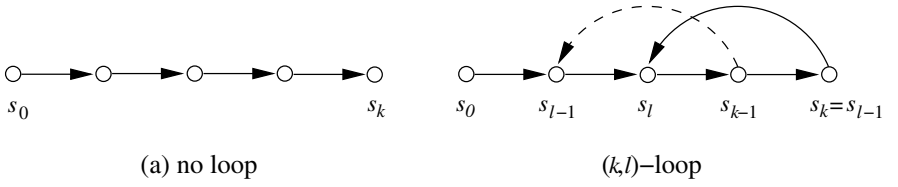

Fig. 1. The two possible cases for a bounded path

Example. Consider a Kripke structure M and the formula $\psi = \mathbf{GF}\neg p$, "infinitely often not p". The negation of the formula is $\mathbf{FG}p$, "eventually always p". We will write a formula which encodes all possible witnesses of length k for the formula $\mathbf{FG}p$. First, we need a formula that captures all paths of length k. Let $T(s,s')$ be the transition relation of M as a propositional formula and $I(s)$ a predicate over the state variables defining the initial states. A path of length k is encoded by the formula:

$$|[M]|_k := I(s_0) \wedge \bigwedge_{i=1}^{k} T(s_{i-1}, s_i). \tag{1}$$

Since the formula we are considering requires an infinite witness we can skip the no loop case. For fixed k and l we use the following rules to build the formula ${}_l|[\neg\psi]|_k$ for capturing witnesses of $\neg\psi$, adapted from [1] to our new loop shape (the dashed back edge):

$${}_l|[\mathbf{F}\phi]|_k^i := \bigvee_{j=min(i,l-1)}^{k-1} {}_l|[\phi]|_k^j \qquad {}_l|[\mathbf{G}\phi]|_k^i := \bigwedge_{j=min(i,l-1)}^{k-1} {}_l|[\phi]|_k^j$$

Thus ${}_l|[\psi]|_k^0 = \bigvee_{i=0}^{k} \bigwedge_{j=min(i,l-1)}^{k-1} p(s_j)$. For each possible (k,l)-loop we must express the condition $L_l := (s_k = s_{l-1})$. Here the states s_i are bit vectors and equality $s_i = s_j$ is defined by $\wedge_{m=1}^{n} s_i[m] \Leftrightarrow s_j[m]$, assuming the vectors have n elements and the m:th element is denoted $s_i[m]$. The final formula which is satisfiable iff there exists a counterexample of length $k > 0$ is:

$$|[M]|_k \wedge \left(\bigvee_{l=1}^{k} \left(L_l \wedge {}_l|[\psi]|_k^0 \right) \right) = I(s_0) \wedge \bigwedge_{i=1}^{k} T(s_{i-1}, s_i) \wedge \underline{\bigvee_{l=1}^{k} \left(L_l \wedge \bigvee_{i=0}^{k-1} \bigwedge_{j=min(i,l-1)}^{k-1} p(s_j) \right)}$$

Without sharing the formula is obviously cubic in k. Let us focus on the LTL part, the big underlined disjunction over $l = 1, \ldots, k$. A first level of sharing can be obtained by associating the inner conjunction to the right, resulting in a quadratic DAG representation. Using the same general idea, the inner disjunction can be associated to the left. The overall size becomes linear. As an example see the circuit in Fig. 2 for $k = 4$. It can still be further optimised by applying $a \vee (a \wedge b) \equiv a$, which essentially results in removing the middle column of or-gates. However, as has been noted in [5], using associativity in synthesis is difficult and in general does not avoid the worst case, which is at least cubic.

As an example for the non-linear behaviour of the original translation [1] consider the (E)LTL formula $G(r \rightarrow (p\mathbf{U}q))$. In the result of the translation we focus on propositional subformulas, which represent the translation of the inner temporal operator at all positions $i = 0, \ldots, k-1$ and all loop starts $l = 1, \ldots, k$. Following Def. 13 in [1] these formulas are sum of product forms. Each product is a cube of the predicates p and q at various states. In Fig. 3 we list all cubes that occur as subformulas for $k = 4$. Each cube

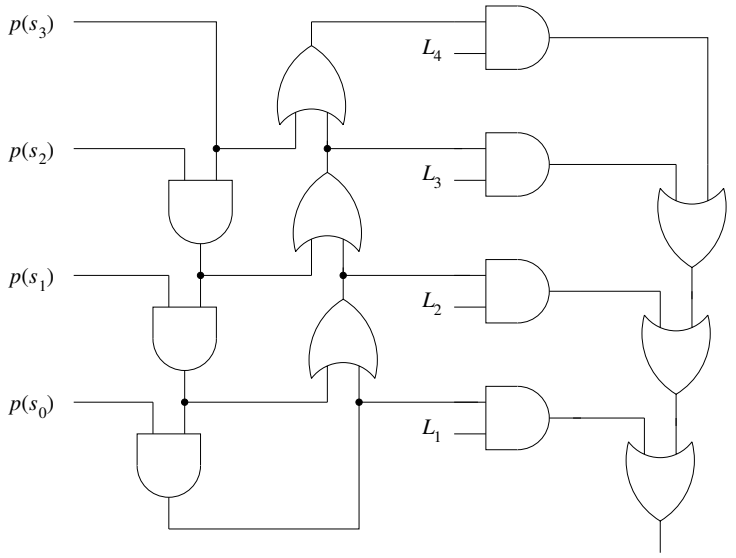

Fig. 2. Circuit encoding for the LTL formula $\mathbf{FG}p$ for $k = 4$

q			
q			p
q		p	p
q	p	p	p

	q		
	q		p
	q	p	p
p	q	p	p
p	q		p
p	q		

		q	
	p	q	
p	p	q	
p	p	q	p
	p	q	p
		q	p

			q
		p	q
	p	p	q
p	p	p	q

Fig. 3. Non-linear number of cubes in the translation of $G(r \rightarrow (p\mathbf{U}q))$ for $k = 4$

is represented by one row of the four matrices in Fig. 3. Each of the matrices collects those cubes where q holds at the same position resp. in the same state.

The number of cubes is at least quadratic in k. For each position j where q holds, the p sequences can be shared. Therefore an upper bound on the overall size is $O(k^3)$ and not $O(k^4)$. The exact size is hard to calculate, but with $\Omega(k^2)$ different cubes in the example, the size has a quadratic lower bound as well.

2.1 LTL

An LTL formula φ is defined over a set of atomic propositions AP. An LTL formula has the following syntax:

1. $\psi \in AP$ is an LTL formula.
2. If ψ and φ are LTL formulae then so are $\neg\psi$, $\mathbf{X}\psi$, $\psi\mathbf{U}\varphi$, $\psi\mathbf{R}\varphi$, $\psi \wedge \varphi$, and $\psi \vee \varphi$.

The operators are the next-time operator **X**, the until operator **U**, and its dual the release operator **R**.

Each formula defines a set of infinite words (models) over 2^{AP}. Let $\pi \in (2^{AP})^\omega$ be an infinite word. We denote the suffix of a word $\pi = \sigma_0\sigma_1\sigma_2\ldots$ by $\pi^i = \sigma_i\sigma_{i+1}\sigma_{i+2}\ldots$ where $\sigma_i \in 2^{AP}$, and π_i denotes the prefix $\pi_i = \sigma_0\sigma_1\ldots\sigma_i$. When a formula ψ defines a word π at time i this is denoted $\pi^i \models \psi$. The set of infinite words defined by a formula ψ is $\{\pi \in (2^{AP})^\omega \mid \pi \models \psi\}$. The relation '$\models$' is inductively defined in the following way.

$$
\begin{array}{ll}
\pi^i \models \psi & \Leftrightarrow \psi \in \sigma_i \text{ for } \psi \in AP.\\
\pi^i \models \neg\psi & \Leftrightarrow \pi \not\models \psi.\\
\pi^i \models \psi \vee \varphi & \Leftrightarrow \pi^i \models \psi \text{ or } \pi^i \models \varphi.\\
\pi^i \models \psi \wedge \varphi & \Leftrightarrow \pi^i \models \psi \text{ and } \pi^i \models \varphi.\\
\pi^i \models \mathbf{X}\psi & \Leftrightarrow \pi^{i+1} \models \psi.\\
\pi^i \models \psi\mathbf{U}\varphi & \Leftrightarrow \exists n \geq i \text{ such that } \pi^n \models \varphi \text{ and } \pi^j \models \psi \text{ for all } i \leq j < n.\\
\pi^i \models \psi\mathbf{R}\varphi & \Leftrightarrow \forall n \geq i, \pi^n \models \varphi \text{ or } \pi^j \models \psi \text{ for some } i \leq j < n.
\end{array}
$$

If $\pi^0 \models \psi$ we simply write $\pi \models \psi$. This presentation of the semantics is intentionally redundant. The additional operators allow us to transform any formula to a *positive normal form*. Formulas in positive normal form have negations only in front of atomic propositions. Using the dualities $\psi\mathbf{U}\varphi \equiv \neg(\neg\psi\mathbf{R}\neg\varphi)$, $\neg\mathbf{X}\psi \equiv \mathbf{X}\neg\psi$ and De Morgan's law, any formula can be transformed without blowup to positive normal form by pushing in the negations. All formulas considered in this paper are assumed to be in positive normal form. We also make use of the standard abbreviations $\top \equiv p \vee \neg p$ for some arbitrary $p \in AP$, $\bot \equiv \neg\top$, $\mathbf{F}\psi \equiv \top\mathbf{U}\psi$ ('finally'), and $\mathbf{G}\psi \equiv \bot\mathbf{R}\psi \equiv \neg\mathbf{F}\neg\psi$ ('globally').

A formula holds in a Kripke structure if all paths of the Kripke structure are accepted by the formula. Formally, a Kripke structure is a tuple $M = (S, T, s_0, L)$, where S is a set of states, $T \subseteq S \times S$ the transition relation, $s_0 \in S$ the initial state, and $L : S \to 2^{AP}$ a function labelling all states with atomic propositions. We require that the transition relation is total. A path of the Kripke structure is a sequence of states $\xi = s_0 s_1 s_2 \ldots$ where s_0 is the initial state and for all $i \geq 0$ we have that $(s_i, s_{i+1}) \in T$. The corresponding word π of a path $\xi = s_0 s_1 s_2 \ldots$ is $\pi = L(s_0)L(s_1)L(s_2)\ldots$. We write $M \models \psi$, if for all paths $\xi = s_0 s_1 s_2 \ldots$ of M the corresponding word π is defined by ψ, i.e. $\pi \models \psi$.

Bounded model checking uses a *bounded semantics of LTL* which safely under approximates the normal semantics. It allows us to use a bounded prefix $\pi_k = s_0 s_1 \ldots s_k$ of an infinite path π to check the formula. The semantics does a case split depending on if the infinite π is a k-loop or not. Biere et al. [1] have shown that if a formula ψ is true in the bounded semantics, denoted $\pi \models_k \psi$, this implies that $\pi \models \psi$. The definition below assumes the formula is in positive normal form.

Definition 1. *([1, 6]) Given an infinite path π and bound $k \in \mathbb{N}$, a formula ψ holds in a path π with bound k iff $\pi \models^0_k \psi$ where*

$$
\begin{array}{ll}
\pi \models^i_k p & \Leftrightarrow p \in s_i \text{ for } p \in AP\\
\pi \models^i_k \neg p & \Leftrightarrow p \notin s_i \text{ for } p \in AP\\
\pi \models^i_k \psi_1 \wedge \psi_2 & \Leftrightarrow \pi \models^i_k \psi_1 \text{ and } \pi \models^i_k \psi_2\\
\pi \models^i_k \psi_1 \vee \psi_2 & \Leftrightarrow \pi \models^i_k \psi_1 \text{ or } \pi \models^i_k \psi_2
\end{array}
$$

$$\pi \models_k^i \mathbf{X}\psi \quad \Leftrightarrow \begin{cases} \pi \models_k^{i+1} \psi & \pi \textit{ is a k-loop} \\ \pi \models_k^{i+1} \psi \wedge (i < k) & \textit{otherwise} \end{cases}$$

$$\pi \models_k^i \psi_1 \mathbf{U} \psi_2 \Leftrightarrow \begin{cases} \exists j \geq i : \pi \models_k^j \psi_2 \wedge \forall n, i \leq n < j : \pi \models_k^n \psi_1 & \textit{k-loop} \\ \exists j, i \leq j \leq k : \pi \models_k^j \psi_2 \wedge \forall n, i \leq n < j : \pi \models_k^n \psi_1 & \textit{otherwise} \end{cases}$$

$$\pi \models_k^i \psi_1 \mathbf{R} \psi_2 \Leftrightarrow \begin{cases} \forall j \geq i : \pi \not\models_k^j \psi_2 \implies \exists n, i \leq n < j : \pi \models_k^n \psi_1 & \textit{k-loop} \\ \exists j, i \leq j \leq k : \pi \models_k^j \psi_1 \wedge \forall n, i \leq n \leq j : \pi \models_k^n \psi_2 & \textit{otherwise} \end{cases}$$

3 A New Translation

Our new translation takes advantage of the fact that for lasso-shaped Kripke structures the semantics of LTL and CTL coincide [7, 8]. The intuition is that when each state has one successor (i.e. the path is lasso-shaped) the semantics of the path quantifiers **A** and **E** of CTL agree. An LTL formula can therefore be evaluated in a lasso-shaped Kripke structure by a CTL model checker by prefixing each temporal operator by an **E** path quantifier [8], which results in a CTL formula[1]. We can thus use the fixpoint characterisation of CTL model checking as a starting point for our translation. The new translation also separates the concern of if the path has a (k,l)-loop from the semantics to an independent part of the translation.

The intuition behind our translation is the following. Following [1], we generate a propositional formula which generates all paths of length k. A part is added to the translation which makes a choice between the following possibilities. Either (a) there is no loop, or (b) there is a loop, i.e. a state s_{l-1} such that $s_k = s_{l-1}$ for some index $1 \leq l \leq k$. The choice and additional constraints under which the choice can be made are implemented as follows. Fresh variables l_i, which do not depend on the state variables in any way, are introduced with appropriate constraints such that if l_i is true then $s_{i-1} = s_k$. We allow at most one l_i to be true in a satisfying truth assignment. This results in a lasso-shaped Kripke structure or a simple finite path if no l_i is true. Allowing simple finite paths is an optimisation and does not affect correctness, but can in some cases (formulas with safety-counterexamples) result in shorter counterexamples. Model checking is accomplished by generating propositional formulas to evaluate the greatest and least fixpoints as required by the implicit CTL formula.

Let M be the Kripke structure of the system and $T(s,s')$ the symbolic transition relation. We consider an unrolling of states $s_0 s_1 \ldots s_k$. Each s_i is a vector of state variables. The unrolling is obtained by equation (1). We require that the Kripke structure is lasso-shaped or a finite path. The variables l_i can seen as selecting one (or possibly none) of the possible (k,l)-loops. This is accomplished by the following constraints.

$$\begin{aligned} |[\mathit{LoopConstraints}]|_k &\Leftrightarrow \mathit{Loop}_k \wedge \mathit{AtMostOne}_k \\ \mathit{Loop}_k &\Leftrightarrow \textstyle\bigwedge_{i=1}^{k} (l_i \Rightarrow (s_{i-1} = s_k)) \\ \mathit{AtMostOne}_k &\Leftrightarrow \textstyle\bigwedge_{i=1}^{k} (\mathit{SmallerExists}_i \Rightarrow \neg l_i) \\ \mathit{SmallerExists}_1 &\Leftrightarrow \bot \\ \mathit{SmallerExists}_{i+1} &\Leftrightarrow \mathit{SmallerExists}_i \vee l_i, \text{ where } 0 < i \leq k \end{aligned}$$

[1] Naturally, we could also use the **A** path quantifier.

In contrast to [1], our definitions also allow the no loop case even if the path has a (k,l)-loop.

The until operator $\mathbf{E}(\psi_1 \mathbf{U} \psi_2)$ can be evaluated by computing the least fixed point $\mathbf{E}(\psi_1 \mathbf{U} \psi_2) = \mu Z.\psi_2 \vee (\psi_1 \wedge \mathbf{EX}Z)$ (see e.g. [9]) while the release operator $\mathbf{E}(\psi_1 \mathbf{R} \psi_2)$ can be evaluated by computing the greatest fixpoint $\mathbf{E}(\psi_1 \mathbf{R} \psi_2) = \nu Z.\psi_2 \wedge (\psi_1 \vee \mathbf{EX}Z)$. The fixpoints are evaluated by first computing an approximation $\langle\langle\cdot\rangle\rangle_i$ for each state and subformula. After this the results of the approximation are used to compute the final result $|[\cdot]|_i$. We evaluate the fixpoints for s_i where $0 \leq i \leq k+1$. The last case $k+1$ is added to make the connections to fixpoints easier to see from the translation.

$:=$	$i \leq k$	$i = k+1$
$\|[p]\|_i$	p_i	$\bigvee_{j=1}^{k} (l_j \wedge p_j)$
$\|[\neg p]\|_i$	$\neg p_i$	$\bigvee_{j=1}^{k} (l_j \wedge \neg p_j)$
$\|[\mathbf{X}\psi]\|_i$	$\|[\psi]\|_{i+1}$	$\bigvee_{j=1}^{k} \left(l_j \wedge \|[\psi]\|_{j+1}\right)$
$\|[\psi \mathbf{U} \varphi]\|_i$	$\|[\varphi]\|_i \vee \left(\|[\psi]\|_i \wedge \|[\psi \mathbf{U} \varphi]\|_{i+1}\right)$	$\bigvee_{j=1}^{k} \left(l_j \wedge \langle\langle \psi \mathbf{U} \varphi \rangle\rangle_j\right)$
$\|[\psi \mathbf{R} \varphi]\|_i$	$\|[\varphi]\|_i \wedge \left(\|[\psi]\|_i \vee \|[\psi \mathbf{R} \varphi]\|_{i+1}\right)$	$\bigvee_{j=1}^{k} \left(l_j \wedge \langle\langle \psi \mathbf{R} \varphi \rangle\rangle_j\right)$
$\langle\langle \psi \mathbf{U} \varphi \rangle\rangle_i$	$\|[\varphi]\|_i \vee \left(\|[\psi]\|_i \wedge \langle\langle \psi \mathbf{U} \varphi \rangle\rangle_{i+1}\right)$	$\bot$
$\langle\langle \psi \mathbf{R} \varphi \rangle\rangle_i$	$\|[\varphi]\|_i \wedge \left(\|[\psi]\|_i \vee \langle\langle \psi \mathbf{R} \varphi \rangle\rangle_{i+1}\right)$	$\top$

The auxiliary translation $\langle\langle\cdot\rangle\rangle$ which computes the approximations for the fixpoints is defined in the last two rows.

Let us consider the case $\psi = \psi_1 \mathbf{R} \psi_2$. We initialise $\langle\langle\psi\rangle\rangle_{k+1}$ to true since we are approximating a greatest fixpoint. When $0 \leq i \leq k$, the auxiliary translation $\langle\langle\psi\rangle\rangle_i$ is the normal fixpoint definition of the release operator. The computed approximation of the fixpoint $\langle\langle\psi\rangle\rangle$ is used to initialise $|[\psi]|_{k+1}$ with the value of $\langle\langle\psi\rangle\rangle_l$, the value of the approximation in the successor of s_k, when we are dealing with a (k,l)-loop. Finally, $|[\psi]|_i$, where $0 \leq i \leq k$, computes the accurate values for each state s_i, again using the standard fixpoint characterisation of release.

Given a Kripke structure M, an LTL formula ψ, and a bound k, the complete encoding as a propositional formula is given by $|[M,\psi,k]|$.

$$|[M,\psi,k]| = |[M]|_k \wedge |[\mathit{LoopConstraints}]|_k \wedge |[\psi]|_0$$

Theorem 1. *Given a finite Kripke structure M, a bound $k \in \mathbb{N}$ and an LTL formula ψ, M has a path π with $\pi \models_k \psi$ iff $|[M,\psi,k]|$ is satisfiable.*

Proof. The proof sketch follows the argument at the beginning of this Section. For both directions we can assume that π is given and is a path of M. Further assume that π is a (k,l) loop. The other case is obvious from the definitions. The bounded semantics on a (k,l) loop coincides with the unbounded semantics. What remains to be proven is that the LTL part of the translation when partially instantiated with π is satisfiable iff $\pi \models \psi$.

Instead of checking whether ψ holds along π we check the corresponding CTL formula ψ' on π interpreted as a Kripke structure itself. The CTL formula ψ' is obtained from ψ by prefixing every temporal operator with the existential path quantifier

E. The ECTL formula ψ' can be translated into an alternation free formula of the modal mu-calculus, which in turn can be transformed into a set of mutual recursive boolean equations with fixpoint semantics as in [10]. The event-driven linear fix point algorithm of [10] is then reformulated symbolically as a non-recursive boolean equation system, which is equivalent to our definition of $|[\cdot]|$. □

As in Theorem 9 of [1] we can lift our Theorem 1 to the unbounded semantics. An upper bound on k would then be of the order $O(|\psi| \cdot |M| \cdot 2^{|\psi|})$. This is easy to show using the automata-theoretic approach to model checking.However, our main result is the following:

Theorem 2. *$|[M,\psi,k]|$ seen as Boolean circuit is linear in $|T|$, $|\psi|$, and k. More precisely, it is of the size $O(|I|+((|T|+|\psi|)\cdot k))$, where $|I|$ and $|T|$ are the sizes of the initial state predicate and the transition relation seen as Boolean circuits, respectively*[2].

Proof. Obviously the translation of $LoopConstraints_k$ is linear w.r.t. k, since both $Loop_k$ and $AtMostOne_k$ loop once over k. We will argue the linearity of $|[\cdot]|$ using the until-case, as it is the most complex. For each $0 \leq i \leq k$, the translation adds a constant number of constraints. The case $i = k+1$ adds k constraints that refer to $\langle\langle \mathbf{U} \rangle\rangle_i$. This does not result in a quadratic formula, even though $\langle\langle \mathbf{U} \rangle\rangle_i$ is linear, because $\langle\langle \mathbf{U} \rangle\rangle_i$ can clearly be shared between the constraints. Linearity of $\langle\langle \mathbf{U} \rangle\rangle_i$ is obvious as only a constant number of constraints are added for each $0 \leq i \leq k+1$. □

3.1 Optimising the Translation

A simple way to optimise the translation is to introduce special translations for certain derived operators. We have developed special translations for $\mathbf{G}\psi, \mathbf{F}\psi, \mathbf{GF}\psi$ and $\mathbf{FG}\psi$. These formulas have similarities which can also be seen in the way they share translations in the case $i = k+1$. Note that the translations of $|[\mathbf{GF}\psi]|_i$ and $|[\mathbf{FG}\psi]|_i$ are only dependent on the case $i = k+1$ since the semantics of the formulas only places requirements on states inside the loop.

$:=$	$i \leq k$	$i = k+1$
$\|[\mathbf{G}\psi]\|_i$	$\|[\varphi]\|_i \wedge \|[\mathbf{G}\psi]\|_{i+1}$	$\bigvee_{j=1}^{k} \left(l_j \wedge \langle\langle \mathbf{G}\psi \rangle\rangle_j \right)$
$\|[\mathbf{F}\psi]\|_i$	$\|[\varphi]\|_i \vee \|[\mathbf{F}\psi]\|_{i+1}$	$\bigvee_{j=1}^{k} \left(l_j \wedge \langle\langle \mathbf{F}\psi \rangle\rangle_j \right)$
$\|[\mathbf{GF}\psi]\|_i$	$\|[\mathbf{GF}\psi]\|_{k+1}$	$\bigvee_{j=1}^{k} \left(l_j \wedge \langle\langle \mathbf{F}\psi \rangle\rangle_j \right)$
$\|[\mathbf{FG}\psi]\|_i$	$\|[\mathbf{FG}\psi]\|_{k+1}$	$\bigvee_{j=1}^{k} \left(l_j \wedge \langle\langle \mathbf{G}\psi \rangle\rangle_j \right)$
$\langle\langle \mathbf{G}\psi \rangle\rangle_i$	$\|[\varphi]\|_i \wedge \langle\langle \mathbf{G}\psi \rangle\rangle_{i+1}$	$\top$
$\langle\langle \mathbf{F}\psi \rangle\rangle_i$	$\|[\varphi]\|_i \vee \langle\langle \mathbf{F}\psi \rangle\rangle_{i+1}$	$\bot$

The translations for the above derived operators can be further optimised at the cost of introducing $k+1$ additional variables. However, the new variables are functionally dependent on the variables l_i and are shared by all subformulas using them. The variables

[2] This bound applies to both to the number of gates and the number of wire connections between the gates of the Boolean circuit in question.

$InLoop_j$, where $0 < j \leq k$, express the fact that the state s_j is in the loop selected by the l_i variables. Additionally, we introduce the variable *LoopExists* which is true iff the path $s_0 s_1 \ldots s_k$ has a (k,l)-loop. In other words, *LoopExists* is false iff π_k is treated as a simple path without a loop. This is encoded by the following definitions.

$$\begin{aligned} InLoop_{i+1} &\Leftrightarrow InLoop_i \vee l_{i+1} \text{ for } 0 < i < k \\ InLoop_1 &\Leftrightarrow l_1 \\ LoopExists &\Leftrightarrow InLoop_k \end{aligned}$$

With the $InLoop_i$ variables we can eliminate the need for the auxiliary translation $\langle\langle\cdot\rangle\rangle$ for the derived operators. This simplifies the translation in most cases. The change in the translation is small as only the case $i = k+1$ changes. Sharing also occurs between the translation for different operators as the translations for $\mathbf{G}\psi$ and $\mathbf{FG}\psi$, and for $\mathbf{F}\psi$ and $\mathbf{GF}\psi$ are the same.

$$\begin{aligned} |[\mathbf{G}\psi]|_{k+1} &= |[\mathbf{FG}\psi]|_{k+1} = LoopExists \wedge \textstyle\bigwedge_{i=1}^{k} (\neg InLoop_i \vee |[\psi]|_i) \\ |[\mathbf{F}\psi]|_{k+1} &= |[\mathbf{GF}\psi]|_{k+1} = \textstyle\bigvee_{i=1}^{k} (InLoop_i \wedge |[\psi]|_i) \end{aligned}$$

3.2 Fairness

In many cases we wish to restrict the possible executions of the system to disallow executions which are unrealistic or impossible in the physical system. The standard way is to add fairness constraints to the model in order to only obtain interesting counterexamples.

There are a few well-known notions of fairness. *Justice* (weak fairness) requires that certain conditions are true infinitely often. *Compassion* (strong fairness) requires that if certain conditions are true infinitely often then certain other conditions must also hold infinitely often.

Let $\{J_1, \ldots, J_j\}$ be a set of Boolean predicates over the state variables which define the conditions that should be true infinitely often. Justice can then be expressed as the LTL formula

$$\mathcal{J} = \bigwedge_{i=1}^{j} \mathbf{GF} J_i.$$

Similarly, compassion can be expressed as an LTL formula. A set of pairs of Boolean predicates $\{(L_1, U_1), \ldots, (L_c, U_c)\}$ over the state variables define the compassion sets. Compassion is defined by the formula

$$\mathcal{C} = \bigwedge_{i=1}^{c} (\mathbf{GF} L_i \Rightarrow \mathbf{GF} U_i).$$

We include the fairness constraints in the specification. Thus, instead of model checking the formula ψ, we check the formula $\mathcal{J} \wedge \mathcal{C} \rightarrow \psi$. Since our propositional encoding of LTL formulas is linear, our overhead for handling fairness is linear in the number of fairness constraints.

4 Related Work

This work can be seen as a continuation of the work done in [11]. There the bounded model checking problem for LTL is translated into the problem of finding a stable model of a normal logic program (another NP-complete problem, see references in [11]) of essentially (modulo a constant) the same size as the translation presented here. The main differences to that work are the following. (i) The translation of [11] uses the close correspondence between the stable model semantics with the notion of a least fixpoint of a set of Boolean equations. The "formula variable dependency graphs" of the translation of [11] are in fact cyclic, while in this work they are acyclic. Seeing the translation of [11] as a propositional formula would result in a translation which is *not* sound. By using the correspondence between least fixpoints and stable models the translation for until and release formulas in [11] do not require the auxiliary translations $\langle\langle\cdot\rangle\rangle$. Thus the translation of [11] had to be significantly changed in order to use SAT. Additionally, the best known automatic translation of the stable model problem to SAT is non-linear [12]. (ii) The translation in [11] employs a different system modelling formalism, which allows for partial order semantics based optimisations. (iii) Moreover, the translation in [11] also allows for deadlocking systems with LTL interpreted over finite paths in the case of a deadlock, a feature left for further work in this paper. (iv) Finally, the implementation presented in this work is new, and based on the NuSMV2 [13] system.

Others have also considered the problem of improving the BMC encoding [5, 6, 4]. Cimatti et al. [5] analyse the original encoding [1] and suggest several optimisations. For instance, they propose a linear encoding for formulas of the form $\mathbf{GF}p$. Their translation is, however, not linear in general. Frisch et al. [6] approach the translation problem by using a normal form of LTL and take advantage of the properties of the normal form. Their procedure modifies the original model and is similar to symbolic tableau-style approaches for LTL model checking. According to their experiments their approach produces smaller encodings than [5]. However, their encoding is also non-linear in the general case. The non-linearity occurs at least in those cases when model checking a formula ψ such that after converting $\neg\psi$ to positive normal form it contains until or finally operators. Closest to their method is the so called *semantic* translation for BMC [14, 4]. The method follows closely the standard automata theoretic approach to model checking and creates a product system $M \times B_{\neg\psi}$, where $B_{\neg\psi}$ is a Büchi automaton representing the negation of the property. The existence of a counterexample is demonstrated by finding a fair loop in the product system. Since only fair loops are accepted the method does not find counterexamples without a loop. This is the main drawback of the method, and is something which could be improved upon in the future. The greatest advantage of the method is that it can leverage the significant amount of research which has been invested in improving the efficiency of LTL to Büchi automata translators. The translation results in a linear number of variables but a quadratic number clauses because of the way fairness is handled. Naturally, the semantic translation could also be improved to linear by e.g. using the translation presented in this work or that of [5]. Furthermore, the approach used in the experiments of [4] results in a translation which is exponential in the LTL formula length as the Wring system used produces explicit state Büchi automata instead of symbolic ones. Related to the semantic translation is the work of [15], which uses a similar product construction. Although

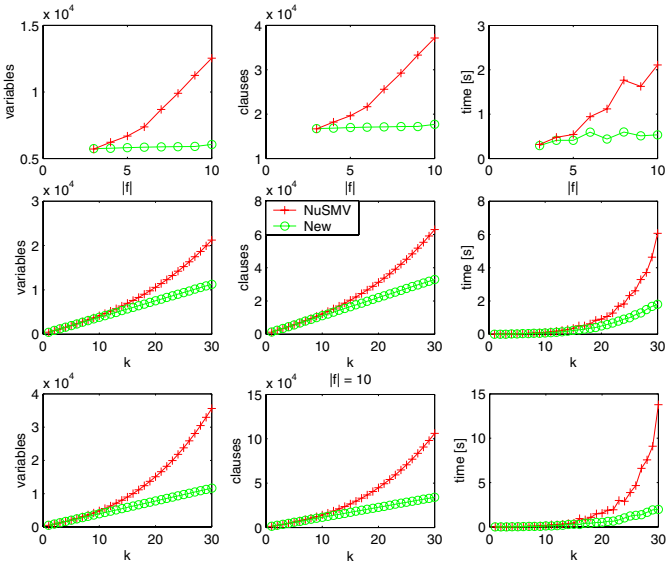

Fig. 4. Plots for NuSMV and New, averages over random formulae

the method is linear in general, the number of state bits in the model is doubled. This blowup does not occur with our method.

Many researchers have also investigated improving SAT solver efficiency. Strichman [2] uses the special properties of the formula $\mathbf{G}p$ to improve solver efficiency of BMC problems. As most safety properties can be reduced to checking invariants, the methods introduced are applicable for safety properties in general. Gupta et al. [16] use BDD model checking runs for training the solvers to achieve better performance.

5 Experiments

The translation has been straightforwardly implemented as a recursive procedure which does case analysis based on the translation. Implementation simplicity is, in our opinion, one of the main strengths of the new translation. The only implementation optimisation used was a simple cache, implemented as a lookup table, for the values of $|[\cdot]|_i$ and $\langle\langle\cdot\rangle\rangle_i$. This avoids a blow up in run time for certain formulas and speeds up the generation of the Boolean formula. All encoding optimisations mentioned in Sect. 3.1 have of course been implemented.

In order to evaluate the practical impact of our new linear translation we have performed two series of experiments. The first series of experiments evaluates the performance of the encoding on random formulae in small random Kripke structures, while the second series of experiments benchmarks the performance on real-life examples. Our implementation is compared against two bounded LTL model checking algorithms. Firstly we compare against the standard NuSMV encoding [13], which includes many of the optimisations of [5]. We also compare against the encoding of [6] which we will call Fixpoint. We do not compare against the SNF encoding also available in [6] since generally the Fixpoint encoding performs better than SNF. In order to make all other

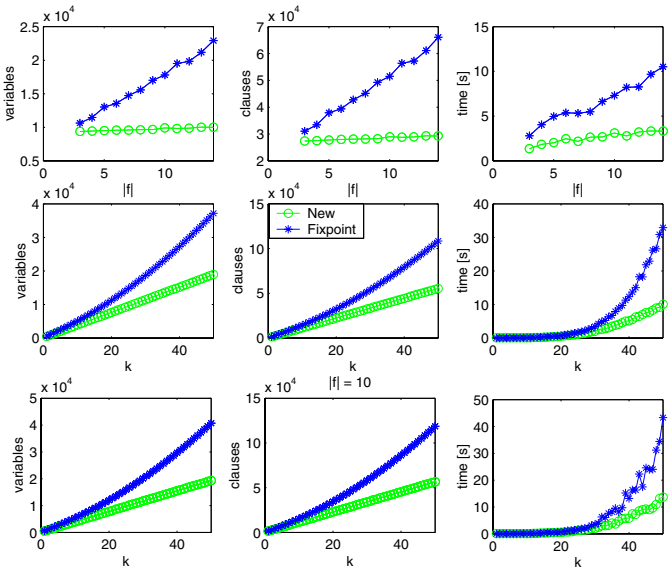

Fig. 5. Plots for Fixpoint and New, averages over random formulae

implementation differences as small as possible, all of the encodings were benchmarked on top of the NuSMV version of D. Sheridan [6] (obtained from his homepage on 18th of March 2004) which contains several BMC related optimisations not included in the standard NuSMV 2.1 distribution. We expect that benchmarking the implementations on top of NuSMV 2.1 would result in larger running times for all the implementations in question at least in the random Kripke structures benchmark. It should be noted that the compact CNF conversion [17] option of the tool was disabled.

In the first series of experiments we generated small random Kripke structures and random formulae using techniques from [8]. The experiments give us some sense of how the implementations scale when the bound or the size of the formula is increased. To demonstrate the cases where the non-linearity of the Fixpoint translation occurs we generated formulas $\neg\psi$ which in positive normal form contains a larger percentage of finally and until operators than other temporal operators. For each formula size we generate 40 formulas, which we then model check by forcing the model checker to look for counterexamples which are of exactly the length specified by the bound. The random Kripke structures we use contain 30 states and one weak fairness constraint which holds in two randomly selected states. We measure the time used to solve the SAT instance and the number clauses and variables in the instance.

When benchmarking against NuSMV default translation we varied the size of the formula from 3 to 10. For each formula size we let the bound grow up to $k = 30$. When benchmarking against Fixpoint translation we were able to increase both the bounds used and the formula sizes to better demonstrate the differences between the two translations. We varied the size of the formula from 3 to 14. For each formula size we let the bound grow up to $k = 50$.

Table 1. Benchmarks

Model	k	NuSMV			Fixpoint			New		
		vars	*clauses*	*time*	*vars*	*clauses*	*time*	*vars*	*clauses*	*time*
abp	16	19,476	57,373	32.3	18,643	54,637	43.7	18,024	52,969	7.4
brp	10	7,599	21,811	1.3	8,550	24,256	1.2	7,471	21,397	1.5
	15	11,494	33,226	18.7	13,150	37,636	22.0	11,116	32,047	17.9
	20	15,514	45,016	471	18,050	51,916	351	14,761	42,697	484
dme	10	53,400	141,438	2.0	54,407	144,022	0.9	53,293	141,087	2.6
	20	104,885	283,733	180	107,527	290,902	263	104,173	281,537	471
	30	156,870	427,528	1,199	161,847	441,382	1,855	155,053	421,987	1,544
pci	10	56,414	167,753	58.3	56,232	167,042	56.6	55,911	166,214	51.5
	15	85,359	254,133	568	84,372	250,947	370	83,756	249,279	382
	20	115,204	343,213	5,921	112,612	335,152	2,216	111,601	332,344	2,102
srg16	20	N/A	N/A	N/A	10,540	28,786	2.3	5,196	14,921	2.7
	40	N/A	N/A	N/A	25,600	71,686	16.6	10,336	29,841	22.3
	60	N/A	N/A	N/A	45,460	128,986	105	15,476	44,761	83.0

In Figures 4 and 5 there are nine plots in each figure which depict the results from tests with random formulae of the new translation against NuSMV and Fixpoint, respectively. The three top plots show the average time, average number of clauses, and average number of variables for each formula size over all bounds. In the second row we have computed the same measures when averaged for each bound over all formula sizes. The last row shows the averages when the size of formula is fixed at ten. The plots clearly show the non-linearity of the competing translations [5, 6] with respect to the bound. Something the plots do not show is time for generating the problems. Our experience is that the new implementation and Fixpoint generated the Boolean formulas almost instantaneously while for the NuSMV encoding there were cases where generation time dominated. In fact, a couple of NuSMV data points had to be omitted from the averages due to the fact that the generation of the SAT instance took several hours.

In the second series of experiments we used real-life examples. As specifications we favoured longer formulas since all implementations can translate simple formulas linearly. The models we used were a model of the alternating bit protocol (abp), a distributed mutual exclusion algorithm (dme), a bounded resource protocol (brp), a model of a pci bus (pci), and a model of a 16-bit shift register (srg16). The results for the real-life examples are summarised in Table 1. We measured the number of variables, cumulative number of clauses and the time used to verify formulas for the reported maximum bound. For the real-life examples, Fixpoint or the new translation are usually the fastest. Our new translation is the most compact one in all cases. However, the differences are small as the model part of the translation dominates the translation size. The shift register example (srg16) shows the strength of a linear translation. NuSMV could not manage $k = 20$ in a reasonable time while Fixpoint displays non-linear growth with respect to k.

All experiments were performed on a computer with an AMD Athlon XP 2000+ processor and 1 GiB of RAM using the SAT solver zChaff [18], version 2003.12.04.

6 Conclusions

We have presented a translation of the bounded LTL model checking problem to SAT which is linear in the bound and the size of the formula. The translation produces a linear number of variables and clauses in the resulting CNF.

Our benchmarks show that our new translation scales better both in size of the bound and the size of the formula than previous implementations [5, 6]. The translation remains linear in all cases. However, in some cases either the size of the formula or the bound must be made large before the benefit shows. One avenue of future work is to include some of the optimisations presented in [5] in order to the improve the performance of our translation for short formulas and small bounds.

Other avenues of future work also exist. One fairly straightforward generalisation of our translation is the ability to handle deadlocking executions. This could probably be done in a manner similar to [11]. Another interesting topic is generalising our translation to include past temporal logic as the translation of [19]. The presented translation could also benefit from specific SAT solver optimisations. When the translation is seen as producing Boolean circuits, all of the circuits are monotonic if the *InLoop* variables and state variables (and their negated versions) are given as inputs. A solver (also CNF-based) could be optimised to take advantage of this.

Acknowledgements

We would like to thank D. Sheridan for sharing his NuSMV implementation with us.

References

1. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In: Tools and Algorithms for the Constructions and Analysis of Systems (TACAS'99). Volume 1579 of LNCS., Springer (1999) 193–207
2. Strichman, O.: Accelerating bounded model checking of safety properties. Formal Methods in System Design **24** (2004) 5–24
3. Kroenig, D., Strichman, O.: Efficient computation of recurrence diameters. In: Verification, Model Checking, and Abstract Interpretation (VMCAI'2003). Volume 2575 of LNCS., Springer (2003) 298–309
4. Clarke, E., Kroenig, D., Oukanine, J., Strichman, O.: Completeness and complexity of bounded model checking. In: Verification, Model Checking, and Abstract Interpretation (VMCAI'2004). Volume 2937 of LNCS., Springer (2004) 85–96
5. Cimatti, A., Pistore, M., Roveri, M., Sebastiani, R.: Improving the encoding of LTL model checking into SAT. In: Verification, Model Checking, and Abstract Interpretation (VMCAI'2002). Volume 2294 of LNCS., Springer (2002) 196–207
6. Frisch, A., Sheridan, D., Walsh, T.: A fixpoint encoding for bounded model checking. In: Formal Methods in Computer-Aided Design (FMCAD'2002). Volume 2517 of LNCS., Springer (2002) 238–255
7. Kupferman, O., Vardi, M.: Model checking of safety properties. Formal Methods in System Design **19** (2001) 291–314
8. Tauriainen, H., Heljanko, K.: Testing LTL formula translation into Büchi automata. STTT - International Journal on Software Tools for Technology Transfer **4** (2002) 57–70
9. Clarke, E., Grumberg, O., Peled, D.: Model Checking. The MIT Press (1999)
10. Cleaveland, R., Steffen, B.: A linear-time model-checking algorithm for the alternation-free modal mu-calculus. Formal Methods in System Desing **2** (1993) 121–147
11. Heljanko, K., Niemelä, I.: Bounded LTL model checking with stable models. Theory and Practice of Logic Programming **3** (2003) 519–550

12. Janhunen, T.: A counter-based approach to translating logic programs into set of clauses. In: Proceedings of the 2nd International Workshop on Answer Set Programming (ASP'03). Volume 78., Sun SITE Central Europe (CEUR) (2003) 166–180
13. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An opensource tool for symbolic model checking. In: Computer Aided Verification (CAV'2002). Volume 2404 of LNCS., Springer (2002) 359–364
14. de Moura, L., Rueß, H., Sorea, M.: Lazy theorem proving for bounded model checking. In: Conference on Automated Deduction (CADE'02). Volume 2392 of LNCS., Springer (2002) 438–455
15. Schuppan, V., Biere, A.: Efficient reduction of finite state model checking to reachability analysis. Software Tools for Technology Transfer (STTT) **5** (2004)
16. Gupta, A., Ganai, M., Wang, C., Yang, Z., Ashar, P.: Learning from BDDs in SAT-based bounded model checking. In: Proceedings of the 40th Conference on Design Automation, IEEE (2003) 824–829
17. Jackson, P., Sheridan, D.: The optimality of a fast CNF conversion and its use with SAT. Technical Report APES-82-2004, APES Research Group (2004) Available from http://www.dcs.st-and.ac.uk/ apes/apesreports.html.
18. Moskewicz, M., Madigan, C., Zhao, Y., L.Zhang, Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the 38th Design Automation Conference. (2001)
19. Benedetti, M., Cimatti, A.: Bounded model checking for past LTL. In: Tools and Algorithms for Construction and Analysis of Systems (TACAS'2003). Volume 2619 of LNCS., Springer (2003) 18–33