Simple is Better: Efficient Bounded Model Checking for Past LTL

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

[1] Laboratory for Theoretical Computer Science
Helsinki University of Technology
P.O. Box 5400, FI-02015 HUT, Finland
{Timo.Latvala, Keijo.Heljanko, Tommi.Junttila}@hut.fi
[2] Institute for Formal Models and Verification
Johannes Kepler University
Altenbergerstrasse 69, A-4040 Linz, Austria
biere@jku.at

Abstract. We consider the problem of bounded model checking for linear temporal logic with past operators (PLTL). PLTL is more attractive as a specification language than linear temporal logic without past operators (LTL) since many specifications are easier to express in PLTL. Although PLTL is not more expressive than LTL, it is exponentially more succinct. Our contribution is a new more efficient encoding of the bounded model checking problem for PLTL based on our previously presented encoding for LTL. The new encoding is *linear* in the bound. We have implemented the encoding in the NuSMV 2.1 model checking tool and compare it against the encoding in NuSMV by Benedetti and Cimatti. The experimental results show that our encoding performs significantly better than this previously used encoding.
Keywords: Bounded Model Checking, Past LTL, NuSMV

1 Introduction

Bounded model checking [1] is an efficient method of implementing *symbolic model checking*, a way of automatically verifying system designs w.r.t. properties given in a temporal logic. Symbolic model checking allows verification of designs with large state spaces by representing the state space implicitly. In bounded model checking (BMC) the system is represented as a propositional logic formula, and only the bounded paths of the system are considered. Given a system model, a temporal logic specification and a bound k, a formula in propositional logic is generated which is satisfiable if and only if the system has a counterexample of length k to the specification. A satisfiability (SAT) solver is used to check if the generated formula is satisfiable. By letting the bound

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

⋆⋆ Work supported by the Academy of Finland (project 53695, grant for research work abroad, research fellow post).

⋆⋆⋆ Work partially done while visiting ITC-IRST, Trento, Italy. This work has been sponsored by the CALCULEMUS! IHP-RTN EC project, contract code HPRN-CT-2000-00102.

grow incrementally we can prove that the system has no counterexample for the given property. Although basic BMC is an incomplete method in practice (it is difficult to a priori determine a reasonably small bound k which guarantees completeness) it has been very successful in industrial context [2,3,4]. The success of BMC is mostly based on that propositional logic is a compact representation for Boolean functions and that BMC allows leveraging the vast improvements in SAT solver technology made in recent years.

Linear temporal logic (LTL) is one of the most popular specification languages used in model checking tools and many model checking tools support some variant of it. However, in most of its incarnations only the so called future fragment of the language (which we will denote by LTL) is considered. This fragment includes only temporal operator which refer to future states. Recently, several papers [5,6,7] have also considered supporting LTL with past operators (PLTL). The main argument for adding support for past operators is motivated by practice: PLTL allows more succinct and natural specifications than LTL. For instance, the specification "if the discharge valve is open, then the pressure alarm must have gone off in the past" can easily be expressed in PLTL while expressing it in LTL is not as straightforward. We believe that an intuitive specification language reduces the probability of a model checking effort failing because of an erroneous specification. The usefulness of PLTL has already been argued earlier in [8].

PLTL also has theoretical advantages compared to LTL. Although PLTL and LTL are equally expressive [9,10], PLTL is exponentially more succinct than LTL [11]. This succinctness comes for free in the sense that model checking for LTL and PLTL are both PSPACE-complete in the length of the formula [12]. In practice, however, PLTL model checking algorithms are more difficult and complex to implement.

The first to present a reasonable solution for doing BMC with PLTL were Benedetti and Cimatti [7]. They showed how the standard BMC encoding [1] can be extended to handle PLTL. Our main contribution is a new encoding for BMC with PLTL based on our LTL encoding [13]. Unlike the encoding in [7], the size of our new encoding is linear in the bound k. The new encoding is quadratic in the size of the formula. When the number of nested past operators is fixed, the encoding becomes linear in the size of the formula. The new encoding has been implemented in the NuSMV 2 model checker [14] and we have experimentally evaluated our encoding. The results clearly show that the new encoding has better running times and that it generates smaller SAT instances than the current encoding in NuSMV. Since the new encoding is also very simple, it allows a straightforward implementation.

2 Bounded Model Checking

The main idea of bounded model checking [1] is to search for *bounded witnesses* for a temporal property. A bounded witness is an infinite path on which the property holds, and which can be represented by a finite path of length k. A finite path can represent infinite behaviour, in the following sense. Either it represents all its infinite extensions or it forms a *loop*. More formally, an infinite path $\pi = s_0 s_1 s_2 \ldots$ of states contains a (k,l)-loop, or just an k-loop, if $\pi = (s_0 s_1 \ldots s_{l-1})(s_l \ldots s_k)^\omega$. The two cases we consider are depicted in Fig. 1.

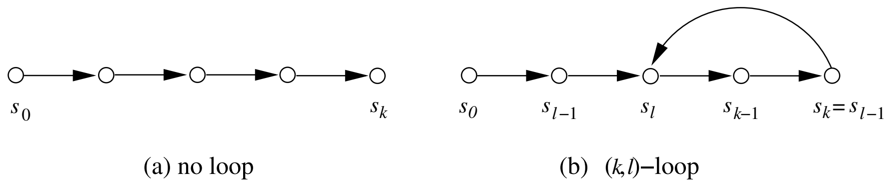

Fig. 1. The two possible cases for a bounded path

In BMC all possible k-length bounded witnesses of the *negation* of the specification are encoded as a SAT problem. The bound k is increased until either a witness is found (the instance is satisfiable) or a sufficiently high value of k to guarantee completeness is reached.

Note that as in [7,13,15] the shape of the loop and accordingly the meaning of the bound k is slightly different from [1]. In this paper, a finite path of length k for representing an infinite path with a loop contains the *looping state* twice, at position $l-1$ and at position k.

2.1 LTL

LTL is a commonly used specification logic. The syntax is defined over a set of atomic propositions AP. Boolean operators we use are negation, disjunction and conjunction. Regarding temporal connectives, we concentrate on the next time ($\mathbf{X}$), the until ($\mathbf{U}$) , and the release ($\mathbf{R}$) operators. The semantics of an LTL formula is defined along infinite paths $\pi = s_0 s_1 \ldots$ of states s_i. The states are part of a model M with transition relation T and initial state constraint I. Further, let π^i denote the suffix of π starting from the i:th state. The semantics can then be defined recursively as follows:

$$\begin{array}{ll}
\pi^i \models \psi & \Leftrightarrow \psi \text{ holds in } s_i \text{ for } \psi \in AP. \\
\pi^i \models \neg\psi & \Leftrightarrow \pi^i \not\models \psi. \\
\pi^i \models \psi_1 \vee \psi_2 & \Leftrightarrow \pi^i \models \psi_1 \text{ or } \pi^i \models \psi_2. \\
\pi^i \models \psi_1 \wedge \psi_2 & \Leftrightarrow \pi^i \models \psi_1 \text{ and } \pi^i \models \psi_2. \\
\pi^i \models \mathbf{X}\psi & \Leftrightarrow \pi^{i+1} \models \psi. \\
\pi^i \models \psi_1 \mathbf{U} \psi_2 & \Leftrightarrow \exists n \geq i \text{ such that } \pi^n \models \psi_2 \text{ and } \pi^j \models \psi_1 \text{ for all } i \leq j < n. \\
\pi^i \models \psi_1 \mathbf{R} \psi_2 & \Leftrightarrow \forall n \geq i, \pi^n \models \psi_2 \text{ or } \pi^j \models \psi_1 \text{ for some } i \leq j < n.
\end{array}$$

Commonly used abbreviations are the standard Boolean shorthands $\top \equiv p \vee \neg p$ for some $p \in AP$, $\bot \equiv \neg\top$, $p \Rightarrow q \equiv \neg p \vee q$, $p \Leftrightarrow q \equiv (p \Rightarrow q) \wedge (q \Rightarrow p)$, and the derived temporal operators $\mathbf{F}\psi \equiv \top \mathbf{U} \psi$ ('finally'), $\mathbf{G}\psi \equiv \neg\mathbf{F}\neg\psi$ ('globally').

It is always possible to rewrite any formula to *positive normal form*, where all negations only appear in front of atomic proposition. This can be accomplished by using the dualities $\neg(\psi_1 \mathbf{U} \psi_2) \equiv \neg\psi_1 \mathbf{R} \neg\psi_2$, $\neg(\psi_1 \mathbf{R} \psi_2) \equiv \neg\psi_1 \mathbf{U} \neg\psi_2$ and $\neg\mathbf{X}\psi \equiv \mathbf{X}\neg\psi$. In the following we assume all formulas are in positive normal form.

2.2 Bounded Model Checking for LTL

We briefly review our simple and compact encoding for bounded model checking LTL given in [13]. This encoding has been shown to outperform previous encodings and in addition is much simpler to implement. Moreover, it forms the basis for our new encoding of PLTL in this paper. It consists of three types of constraints. *Model constraints* encode legal initialised finite paths of the model M of length k:

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

where $I(s)$ is the initial state predicate and $T(s, s')$ is a total transition relation. The *loop constraints* are used to detect loops. We introduce k fresh *loop selector variables* $l_1, \ldots, l_k$ that have the following constraint: if l_i is true then $s_{i-1} = s_k$. In this case the LTL encoding treats the bounded path as a (k,i)-loop. If no loop selector variable is true the LTL encoding treats the path as a simple path without a loop. At most one loop selector variable is allowed to be true. Thus, the loop selector variables show where the bounded path loops. This is accomplished with the following constraints:

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

The constraints select a (k,l)-loop (also called *lasso-shaped* path) from the model, when a loop is needed to find a counterexample. Finally, *LTL constraints* restrict the bounded path defined by the model constraints and loop constraints to witnesses of the LTL formula. The encoding utilises the fact that for lasso-shaped Kripke structures the semantics of CTL and LTL coincide [16,17]. Essentially, the encoding can be seen as a CTL model checker for lasso-shaped Kripke structures based on using the least and greatest fixpoint characterisations of $\mathbf{U}$ and $\mathbf{R}$. The computation of the fixpoints for $\mathbf{U}$ and $\mathbf{R}$ is done in two parts. An auxiliary translation $\langle\langle\cdot\rangle\rangle$ computes an approximation of the fixpoints that is refined to exact values by $|[\cdot]|$. The presentation of the constraints differs slightly from [13] to allow easier generalisation to the PLTL case.

φ	$0 \leq i < k$	$i = k$
$\|[p]\|_i$	p_i	p_i
$\|[\neg p]\|_i$	$\neg p_i$	$\neg p_i$
$\|[\psi_1 \wedge \psi_2]\|_i$	$\|[\psi_1]\|_i \wedge \|[\psi_2]\|_i$	$\|[\psi_1]\|_i \wedge \|[\psi_2]\|_i$
$\|[\psi_1 \vee \psi_2]\|_i$	$\|[\psi_1]\|_i \vee \|[\psi_2]\|_i$	$\|[\psi_1]\|_i \vee \|[\psi_2]\|_i$
$\|[\mathbf{X}\psi_1]\|_i$	$\|[\psi_1]\|_{i+1}$	$\bigvee_{j=1}^{k}\left(l_j \wedge \|[\psi_1]\|_j\right)$
$\|[\psi_1 \mathbf{U} \psi_2]\|_i$	$\|[\psi_2]\|_i \vee \left(\|[\psi_1]\|_i \wedge \|[\psi_1 \mathbf{U} \psi_2]\|_{i+1}\right)$	$\|[\psi_2]\|_i \vee \left(\|[\psi_1]\|_i \wedge \left(\bigvee_{j=1}^{k}\left(l_j \wedge \langle\langle\psi_1 \mathbf{U} \psi_2\rangle\rangle_j\right)\right)\right)$
$\|[\psi_1 \mathbf{R} \psi_2]\|_i$	$\|[\psi_2]\|_i \wedge \left(\|[\psi_1]\|_i \vee \|[\psi_1 \mathbf{R} \psi_2]\|_{i+1}\right)$	$\|[\psi_2]\|_i \wedge \left(\|[\psi_1]\|_i \vee \left(\bigvee_{j=1}^{k}\left(l_j \wedge \langle\langle\psi_1 \mathbf{R} \psi_2\rangle\rangle_j\right)\right)\right)$
$\langle\langle\psi_1 \mathbf{U} \psi_2\rangle\rangle_i$	$\|[\psi_2]\|_i \vee \left(\|[\psi_1]\|_i \wedge \langle\langle\psi_1 \mathbf{U} \psi_2\rangle\rangle_{i+1}\right)$	$\|[\psi_2]\|_i$
$\langle\langle\psi_1 \mathbf{R} \psi_2\rangle\rangle_i$	$\|[\psi_2]\|_i \wedge \left(\|[\psi_1]\|_i \vee \langle\langle\psi_1 \mathbf{R} \psi_2\rangle\rangle_{i+1}\right)$	$\|[\psi_2]\|_i$

The conjunction of these three sets of constraints forms the full encoding of the bounded model checking problem into SAT:

$$|[M,\varphi,k]| = |[M]|_k \wedge |[LoopConstraints]|_k \wedge |[\varphi]|_0.$$

The LTL formula φ has a witness in M that can represented by a finite path of length k iff the encoding is satisfiable. For more details on the encoding and how it can be used for model checking please refer to [13].

3 Bounded Model Checking with Past Operators

Benedetti and Cimatti [7] were the first to consider bounded model checking for PLTL. Their approach is based on the original encoding of Biere et al. [1]. The approach is such that it generates constraints separately for each possible bounded path with a loop (for all values of $0 \leq l \leq k$). This makes sharing structure in the formula difficult. Our encoding is based on a different solution where the concerns of evaluating the formula and forming the bounded path have been separated. As we shall see, this allows for a simple and compact encoding for PLTL.

3.1 PLTL

Extending LTL with past operators results in a logic which is more succinct than LTL and arguably more intuitive for some specifications. The simplest past operators are the two previous state operators $\mathbf{Y}\,\psi$ and $\mathbf{Z}\,\psi$. Both are true if ψ was true in the previous time step. The semantics of the operators differ at the origin of time: $\mathbf{Y}\,\psi$ is always false while $\mathbf{Z}\,\psi$ is always true. Similar to the derived future operators $\mathbf{F}\,\psi$ and $\mathbf{G}\,\psi$ are $\mathbf{O}\,\psi$ ('once') and $\mathbf{H}\,\psi$ ('historically') that hold if ψ holds once in the past or ψ holds always in the past, respectively. The binary operator $\psi_1\,\mathbf{S}\,\psi_2$ ('since') holds if ψ_2 was true once in the past and ψ_1 has been true ever since. Note that ψ_2 must have been true at some point in the past in order for $\psi_1\,\mathbf{S}\,\psi_2$ to hold. The other past binary operator $\psi_1\,\mathbf{T}\,\psi_2$ ('trigger') holds when ψ_2 holds up until the present starting from the time step where ψ_1 was true. If ψ_1 never was true ψ_2 must have been true always in the past.

We define the semantics of PLTL by extending the formal semantics of LTL. Only semantics for the new operators are given.

$$\begin{array}{lll}
\pi^i \models \mathbf{Y}\,\psi & \Leftrightarrow & i > 0 \text{ and } \pi^{i-1} \models \psi. \\
\pi^i \models \mathbf{Z}\,\psi & \Leftrightarrow & i = 0 \text{ or } \pi^{i-1} \models \psi. \\
\pi^i \models \mathbf{O}\,\psi & \Leftrightarrow & \pi^j \models \psi \text{ for some } 0 \leq j \leq i. \\
\pi^i \models \mathbf{H}\,\psi & \Leftrightarrow & \pi^j \models \psi \text{ for all } 0 \leq j \leq i. \\
\pi^i \models \psi_1\,\mathbf{S}\,\psi_2 & \Leftrightarrow & \pi^j \models \psi_2 \text{ for some } 0 \leq j \leq i \text{ and } \pi^n \models \psi_1 \text{ for all } j < n \leq i. \\
\pi^i \models \psi_1\,\mathbf{T}\,\psi_2 & \Leftrightarrow & \text{for all } 0 \leq j \leq i : \pi^j \models \psi_2 \text{ or } \pi^n \models \psi_1 \text{ for some } j < n \leq i.
\end{array}$$

Useful dualities which hold for past operators are $\neg(\psi_1\,\mathbf{S}\,\psi_2) \equiv \neg\psi_1\,\mathbf{T}\,\neg\psi_2$, $\neg\mathbf{H}\,\psi \equiv \mathbf{O}\,\neg\psi$, $\neg(\psi_1\,\mathbf{T}\,\psi_2) \equiv \neg\psi_1\,\mathbf{S}\,\neg\psi_2$, $\neg\mathbf{O}\,\psi \equiv \mathbf{H}\,\neg\psi$, $\neg\mathbf{Z}\,\psi \equiv \mathbf{Y}\,\neg\psi$, and $\neg\mathbf{Y}\,\psi \equiv \mathbf{Z}\,\neg\psi$. Examples of simple PLTL formulas are $\mathbf{G}\,(p \Rightarrow \mathbf{O}\,q)$ ('all p occurrences are preceded by an occurrence of q') and $\mathbf{F}\,\mathbf{G}\,(p\,\mathbf{S}\,\neg q)$ ('eventually p will stay true after q becomes false'). Recall that we assume that all formulas are in positive normal form.

The maximum number of nested past operators in PLTL formula is called the *past operator depth*.

Definition 1. *The past operator depth for a PLTL formula ψ is denoted by $\delta(\psi)$ and is inductively defined as:*

$$\begin{array}{lll}
\delta(\psi) & = 0 & \textit{for } \psi \in AP, \\
\delta(\circ\psi) & = \delta(\psi) & \textit{for } \circ \in \{\neg, \mathbf{X}, \mathbf{F}, \mathbf{G}\}, \\
\delta(\psi_1 \circ \psi_2) & = max(\delta(\psi_1), \delta(\psi_2)) & \textit{for } \circ \in \{\vee, \wedge, \mathbf{U}, \mathbf{R}\}, \\
\delta(\circ\psi) & = 1 + \delta(\psi) & \textit{for } \circ \in \{\mathbf{Y}, \mathbf{Z}, \mathbf{O}, \mathbf{H}\}, \textit{ and} \\
\delta(\psi_1 \circ \psi_2) & = 1 + max(\delta(\psi_1), \delta(\psi_2)) & \textit{for } \circ \in \{\mathbf{S}, \mathbf{T}\}.
\end{array}$$

PLTL has features which impact the way model checking can be done. We illustrate these features through examples. As a running example we use an example from [7] adapted to better suit our setting. In this example the system to be model checked is a counter which uses a variable x to store the counter value. The counter is initialised to 0 and the system adds one to the counter variable x at each time step until the highest value 5 is reached. After this the counter is reset to the value 2 in the next time step and the system starts looping as illustrated in Fig. 2. Thus the system is deterministic and the counter values can be seen as an infinite sequence $(012)(3452)^\omega$ corresponding to a $(6,3)$-loop of the system.

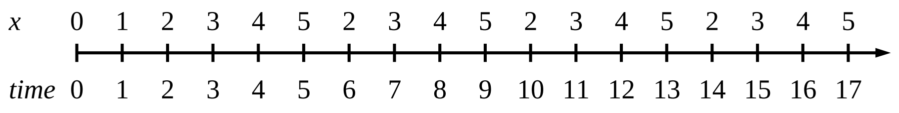

Fig. 2. Execution of the counter system

Consider the $(6,3)$-loop of the counter system. The formula

$$((x=3) \wedge \mathbf{Y}\,\mathbf{Y}\,\mathbf{Y}\,(x=0))$$

holds only at time point 3 but not at any later time point. This demonstrates the (quite obvious) fact that unlike pure future LTL formulas, the PLTL past formulas can distinguish states which belong to different unrollings of the loop. We introduce the notion of a time point belonging to a d-unrolling of the loop to distinguish between different copies of each state in the unrolling of the loop part.

Definition 2. *For a (k,l)-loop π we say that the* period $p(\pi)$ *of π is $(k-l)+1$, i.e., the number of states the loop consists of. We define that a time point $i \geq 0$ in π belongs to the* d-unrolling of the loop *iff $d \geq 0$ is the smallest integer such that $i < l + ((d+1) \cdot p(\pi))$.*

At the time point 3, which belongs to the 0-unrolling of the loop, the formula $\mathbf{Y}\,\mathbf{Y}\,\mathbf{Y}\,(x=0)$ holds. However, at the time point 7 belonging to the 1-unrolling of the loop the formula $\mathbf{Y}\,\mathbf{Y}\,\mathbf{Y}\,(x=0)$ does not hold even though they both correspond to the first state in the unrolling of the loop.

Benedetti and Cimatti [7] observed that encoding the BMC problem for PLTL when the bounded path has no loop was fairly straightforward. It is simple to generalise the no loop case of Biere et al. [1] to include past operators, as they have simple semantics. In the no loop case our encoding reduces to essentially the same as [7]. This case is an optimisation that can sometimes result in shorter counterexamples but is not needed for correctness. When loops are allowed the matter is more complicated and therefore we will focus on this part in the rest of the paper. The fact which enables us to do bounded model checking of PLTL formulas (containing past operators in the loop case) is the following property first observed by [11] and later independently by [7]: for (k,l)-loops the ability to distinguish between time points in different d-unrollings in the past is limited by the past operator depth $\delta(\varphi)$ of a formula φ.

Proposition 1. *Let φ be a PLTL formula and π be a (k,l)-loop. For all $i \geq l$ it holds that if the time point i belongs to a d-unrolling of the loop with $d \geq \delta(\varphi)$ then: $\pi^i \models \varphi$ iff $\pi^j \models \varphi$, where $j = i - ((d-\delta(\varphi)) \cdot p(\pi))$.*

Proof. The proposition directly follows from Theorem 1 and Lemma 2 of [7].

The proposition above can be interpreted saying that after unrolling the loop $\delta(\varphi)$ times the formula cannot distinguish different unrollings of the loop from each other. Therefore if we want to evaluate a formula at an index i belonging to a d-unrolling with $d > \delta(\varphi)$, it is equivalent to evaluate the formula at the corresponding state of the $\delta(\varphi)$-unrolling.

Consider again the running example where we next want to evaluate whether the formula

$$\mathbf{F}\,((x=3) \wedge \mathbf{O}\,((x=4) \wedge \mathbf{O}\,(x=5))) \tag{1}$$

holds in the counter system. The formula expresses that it is possible to reach a point at which the counter has had the values $3,4,5$ in decreasing order in the past. By using the semantics of PLTL it is easy to check that this indeed is the case. The earliest time where the subformula $((x=3) \wedge \mathbf{O}\,((x=4) \wedge \mathbf{O}\,(x=5)))$ holds is time 11 and thus the top-level formula holds at time 0. In fact the mentioned subformula holds for all time points of the form $11+i \cdot 4$, where $i \geq 0$ and $4 = p(\pi)$ is the period of the loop 3452. The time point 11 corresponds to a time step which is in the 2-unrolling of the loop 3452. This stabilisation at the second unrolling is guaranteed by the past operator depth of two of the formula in question. The subformula $((x=4) \wedge \mathbf{O}\,(x=5))$ has past operator depth $\delta(\varphi) = 1$ and it holds for the first time at time step 8 which is in the 1-unrolling of the loop. Again the stabilisation of the formula value is guaranteed by the past operator depth of one of the formula in question. It will also hold for all time steps of the form $8+i \cdot 4$, where $i \geq 0$. Thus, if we need to evaluate any subformula at a time step which belongs to a deeper unrolling than its past operator depth, e.g. if we want to evaluate $((x=4) \wedge \mathbf{O}\,((x=5)))$ at time step 16 in 3-unrolling, we can just take a look at the truth value of that formula at the time step corresponding to the unrolling of the formula to its past operator depth, in this case at time step $8 = 16-(3-1) \cdot 4$.

3.2 Translation

At this point we are ready to present the propositional encoding of the BMC problem for PLTL. From the previous discussion it is fairly obvious that an efficient encoding requires that we encode the unrolling of the loop in a sensible manner and encode the semantics of the operators succinctly.

The basic idea of the encoding is to virtually unroll the path by making copies of the original k step path. A copy of the original path corresponds to a certain d-unrolling. If all loop selector variables l_i are false the encoding collapses to the original path without a loop. The number of copies of the path for a PLTL formula φ is dictated by the past operator depth $\delta(\varphi)$. Since different subformulas have different past depths, the encoding is such that subformulas with different past depths see different Kripke structures. Fig. 3 shows the running example unwound to depth $d = 2$, for evaluating formula (1).

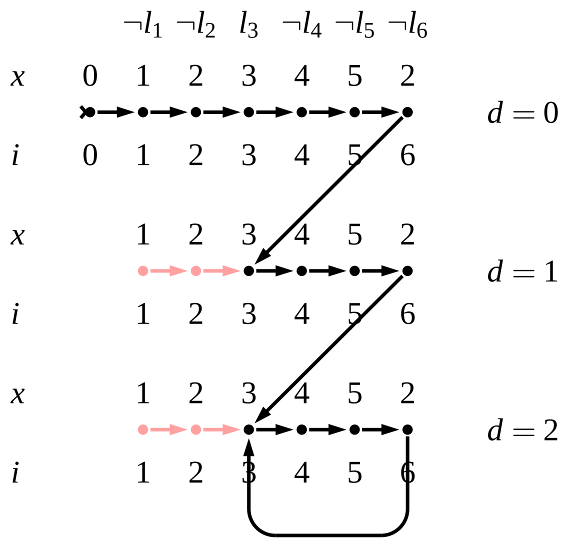

Fig. 3. Black arcs show the Kripke structure induced by virtual unrolling of the loop for $k = 6$ up to depth 2 (i.e., $\delta(\varphi) = 2$) when l_3 holds

The PLTL encoding $|[\varphi]|_i^d$ has two parameters: d is the current d-unrolling and i is the index in the current d-unrolling. The case where $d = 0$ corresponds to the original k-step path. Subformulas at virtual unrolling depth beyond their past operator depth can by Prop. 1 be projected to the depth corresponding to the past operator depth. From this we get our first rule:

$$|[\varphi]|_i^d = |[\varphi]|_i^{\delta(\varphi)}, \text{ when } d > \delta(\varphi).$$

The rest of the encoding is split into cases based on the values of i and d. Encoding atomic propositions and their negation is simple. We simply project the atomic propositions onto the original path. The Boolean operators $\vee$ and $\wedge$ are also easy to encode since they are part of standard propositional logic.

φ	$0 \leq d \leq \delta(\psi), 0 \leq i \leq k$
$\vert[p]\vert_i^d$	p_i
$\vert[\neg p]\vert_i^d$	$\neg p_i$
$\vert[\psi_1 \wedge \psi_2]\vert_i^d$	$\vert[\psi_1]\vert_i^d \wedge \vert[\psi_2]\vert_i^d$
$\vert[\psi_1 \vee \psi_2]\vert_i^d$	$\vert[\psi_1]\vert_i^d \vee \vert[\psi_2]\vert_i^d$

The translation of the future operators is a fairly straightforward generalisation of our pure future encoding of Sect. 2.2 published in [13]. The path is copied as many times as required by the past depth. When $d < \delta(\psi)$ the translation is essentially identical to the pure future encoding with the exception of the case $i = k$. As only the loop part of the copy of the path is relevant (see Fig. 3), the successor for $i = k$ must be encoded to select the correct state in the next d-unrolling. This is accomplished by using the loop selector variables l_i.

φ	$0 \leq d < \delta(\varphi), 0 \leq i < k$	$0 \leq d < \delta(\varphi), i = k$
$\vert[\mathbf{X}\psi_1]\vert_i^d$	$\vert[\psi_1]\vert_{i+1}^d$	$\bigvee_{j=1}^k \left(l_j \wedge \vert[\psi_1]\vert_j^{d+1}\right)$
$\vert[\psi_1 \mathbf{U} \psi_2]\vert_i^d$	$\vert[\psi_2]\vert_i^d \vee \left(\vert[\psi_1]\vert_i^d \wedge \vert[\psi_1 \mathbf{U} \psi_2]\vert_{i+1}^d\right)$	$\vert[\psi_2]\vert_i^d \vee \left(\vert[\psi_1]\vert_i^d \wedge \left(\bigvee_{j=1}^k \left(l_j \wedge \vert[\psi_1 \mathbf{U} \psi_2]\vert_j^{d+1}\right)\right)\right)$
$\vert[\psi_1 \mathbf{R} \psi_2]\vert_i^d$	$\vert[\psi_2]\vert_i^d \wedge \left(\vert[\psi_1]\vert_i^d \vee \vert[\psi_1 \mathbf{R} \psi_2]\vert_{i+1}^d\right)$	$\vert[\psi_2]\vert_i^d \wedge \left(\vert[\psi_1]\vert_i^d \vee \left(\bigvee_{j=1}^k \left(l_j \wedge \vert[\psi_1 \mathbf{R} \psi_2]\vert_j^{d+1}\right)\right)\right)$

When $d = \delta(\varphi)$ we have reached the d-unrolling where the Kripke structure loops back. At this depth we can guarantee that the satisfaction of the subformulas has stabilised (see Prop. 1). Therefore we call the auxiliary translation $\langle\langle\varphi\rangle\rangle_i^d$, which is needed to correctly evaluate until- and release-formulas along the loop (see [13]), at this depth.

φ	$d = \delta(\varphi), 0 \leq i < k$	$d = \delta(\varphi), i = k$
$\vert[\mathbf{X}\psi_1]\vert_i^d$	$\vert[\psi_1]\vert_{i+1}^d$	$\bigvee_{j=1}^k \left(l_j \wedge \vert[\psi_1]\vert_j^d\right)$
$\vert[\psi_1 \mathbf{U} \psi_2]\vert_i^d$	$\vert[\psi_2]\vert_i^d \vee \left(\vert[\psi_1]\vert_i^d \wedge \vert[\psi_1 \mathbf{U} \psi_2]\vert_{i+1}^d\right)$	$\vert[\psi_2]\vert_i^d \vee \left(\vert[\psi_1]\vert_i^d \wedge \left(\bigvee_{j=1}^k \left(l_j \wedge \langle\langle\psi_1 \mathbf{U} \psi_2\rangle\rangle_j^d\right)\right)\right)$
$\vert[\psi_1 \mathbf{R} \psi_2]\vert_i^d$	$\vert[\psi_2]\vert_i^d \wedge \left(\vert[\psi_1]\vert_i^d \vee \vert[\psi_1 \mathbf{R} \psi_2]\vert_{i+1}^d\right)$	$\vert[\psi_2]\vert_i^d \wedge \left(\vert[\psi_1]\vert_i^d \vee \left(\bigvee_{j=1}^k \left(l_j \wedge \langle\langle\psi_1 \mathbf{R} \psi_2\rangle\rangle_j^d\right)\right)\right)$
$\langle\langle\psi_1 \mathbf{U} \psi_2\rangle\rangle_i^d$	$\vert[\psi_2]\vert_i^d \vee \left(\vert[\psi_1]\vert_i^d \wedge \langle\langle\psi_1 \mathbf{U} \psi_2\rangle\rangle_{i+1}^d\right)$	$\vert[\psi_2]\vert_i^d$
$\langle\langle\psi_1 \mathbf{R} \psi_2\rangle\rangle_i^d$	$\vert[\psi_2]\vert_i^d \wedge \left(\vert[\psi_1]\vert_i^d \vee \langle\langle\psi_1 \mathbf{R} \psi_2\rangle\rangle_{i+1}^d\right)$	$\vert[\psi_2]\vert_i^d$

The starting point for the encoding for the past operators is using their one-step fixpoint characterisation. This enables the encoding of the past operators to fit in with the future encoding. Since past operators look backwards, we must encode the move from one copy of the path to the previous copy efficiently. To save space we do not give the encodings for the derived operators $\mathbf{H}\psi \equiv \bot \mathbf{T} \psi$ and $\mathbf{O}\psi \equiv \top \mathbf{S} \psi$ since they are easily derived from the encodings of the binary operators $\psi_1 \mathbf{T} \psi_2$ and $\psi_1 \mathbf{S} \psi_2$.

The simplest case of the encoding for past operators occurs at $d = 0$. At this depth, the past is unique in the sense that the path cannot jump to a lower depth. We do need to take into account the loop edge, so the encoding follows from the recursive characterisation $\psi_1 \mathbf{S} \psi_2$ and $\psi_1 \mathbf{T} \psi_2$. Encoding $\mathbf{Y}\psi$ and $\mathbf{Z}\psi$ is trivial.

φ	$d=0, i=0$	$d=0, 1 \leq i \leq k$
$\|[\psi_1 \mathbf{S} \psi_2]\|_i^d$	$\|[\psi_2]\|_i^d$	$\|[\psi_2]\|_i^d \vee \left(\|[\psi_1]\|_i^d \wedge \|[\psi_1 \mathbf{S} \psi_2]\|_{i-1}^d\right)$
$\|[\psi_1 \mathbf{T} \psi_2]\|_i^d$	$\|[\psi_2]\|_i^d$	$\|[\psi_2]\|_i^d \wedge \left(\|[\psi_1]\|_i^d \vee \|[\psi_1 \mathbf{T} \psi_2]\|_{i-1}^d\right)$
$\|[\mathbf{Y} \psi_1]\|_i^d$	$\bot$	$\|[\psi_1]\|_{i-1}^d$
$\|[\mathbf{Z} \psi_1]\|_i^d$	$\top$	$\|[\psi_1]\|_{i-1}^d$

When $d > 0$ the key ingredient of the encoding is to decide whether the past operator should consider the path to continue in the current unrolling of the path or in the last state of the previous unrolling. The decision is taken based on the loop selector variables, which indicate whether we are in the loop state. In terms of our running example, we need to traverse the straight black arrows of Fig. 3 in the reverse direction. We implement the choice with an if-then-else construct $(l_i \wedge \psi_1) \vee (\neg l_i \wedge \psi_2)$. The expression encodes the choice if l_i is true then the truth value of the expression is decided by ψ_1, otherwise ψ_2 decides the truth value of the expression.

φ	$1 \leq d \leq \delta(\varphi), 2 \leq i \leq k$
$\|[\psi_1 \mathbf{S} \psi_2]\|_i^d$	$\|[\psi_2]\|_i^d \vee \left(\|[\psi_1]\|_i^d \wedge \left(\left(l_i \wedge \|[\varphi]\|_k^{d-1}\right) \vee \left(\neg l_i \wedge \|[\varphi]\|_{i-1}^d\right)\right)\right)$
$\|[\psi_1 \mathbf{T} \psi_2]\|_i^d$	$\|[\psi_2]\|_i^d \wedge \left(\|[\psi_1]\|_i^d \vee \left(\left(l_i \wedge \|[\varphi]\|_k^{d-1}\right) \vee \left(\neg l_i \wedge \|[\varphi]\|_{i-1}^d\right)\right)\right)$
$\|[\mathbf{Y} \psi_1]\|_i^d, \|[\mathbf{Z} \psi_1]\|_i^d$	$\left(l_i \wedge \|[\psi_1]\|_k^{d-1}\right) \vee \left(\neg l_i \wedge \|[\psi_1]\|_{i-1}^d\right)$

The only case left, which actually can be seen as an optimisation w.r.t. the above case, occurs at $i = 1$. The encoding has the general property that if l_j is true all constraints generated by the encoding for $i < j$ will not affect the encoding for $d > 0$. At $i = 1$ we can thus ignore the choice of continuing backwards on the path and always proceed to the previous d-unrolling.

φ	$1 \leq d \leq \delta(\varphi), i = 1$
$\|[\psi_1 \mathbf{S} \psi_2]\|_i^d$	$\|[\psi_2]\|_i^d \vee \left(\|[\psi_1]\|_i^d \wedge \|[\psi_1 \mathbf{S} \psi_2]\|_k^{d-1}\right)$
$\|[\psi_1 \mathbf{T} \psi_2]\|_i^d$	$\|[\psi_2]\|_i^d \wedge \left(\|[\psi_1]\|_i^d \vee \|[\psi_1 \mathbf{T} \psi_2]\|_k^{d-1}\right)$
$\|[\mathbf{Y} \psi_1]\|_i^d, \|[\mathbf{Z} \psi_1]\|_i^d$	$\|[\psi_1]\|_k^{d-1}$

Combining the tables above we get the full encoding $|[\varphi]|_i^d$. Given a Kripke structure M, a PLTL formula φ, and a bound k, the complete encoding as a propositional formula is given by:

$$|[M,\varphi,k]| = |[M]|_k \wedge |[LoopConstraints]|_k \wedge |[\varphi]|_0^0.$$

The correctness of our encoding is established by the following theorem.

Theorem 1. *Given a PLTL formula φ, a bound k and a path $\pi = s_0 s_1 s_2 \ldots$ which is a (k,l)-loop, $\pi \models \varphi$ iff $|[M,\varphi,k]|$ is satisfiable.*

Proof. (sketch) The proof proceeds as an induction on the structure of the formula. All future cases follow a similar pattern. As an example, consider the case $\varphi = \psi_1 \mathbf{U} \psi_2$. By appealing to the induction hypothesis we can assume that $|[\psi_1]|_i^d$ and $|[\psi_2]|_i^d$ are correct. The future encoding replicates the path $\delta(\varphi)$ times, which ensures that at $d = \delta(\varphi)$ all subformulas have stabilised (see Prop. 1). Let $k' = k + p(\pi) \cdot \delta(\varphi)$ denote the index of π which corresponds to the final index of the unrolled model. We first prove that the encoding is correct at $\pi^{k'}$ corresponding to $|[\varphi]|_k^{\delta(\varphi)}$. We will make use of the equivalence: $\pi^i \models \psi_1 \mathbf{U} \psi_2$ iff $\pi^i \models \psi_2$ or $\left(\pi^i \models \psi_1 \text{ and } \pi^{i+1} \models \psi_1 \mathbf{U} \psi_2\right)$.

First assume that $|[\varphi]|_k^{\delta(\varphi)}$ holds. The encoding has the following property: for a (k,l)-loop π, whenever $|[M,\varphi,k]|$ has a satisfying truth assignment where no loop selector variable l_i is true another satisfying truth assignment exists where l_l is true. Thus we only need to consider the case where l_l is true. From $|[\varphi]|_k^{\delta(\varphi)}$ it follows that either $|[\psi_2]|_k^{\delta(\varphi)}$ holds, or that $|[\psi_1]|_k^{\delta(\varphi)}$ and $\langle\langle\varphi\rangle\rangle_l^{\delta(\varphi)}$ hold. In the former case we can appeal to the induction hypothesis and we are done. In latter case we can argue by the definition of $\langle\langle\psi_1 \mathbf{U} \psi_2\rangle\rangle$ that $|[\psi_2]|_j^{\delta(\varphi)}$ must hold for some $l \leq j \leq k$. Let j be the smallest such index. Since $\langle\langle\psi_1 \mathbf{U} \psi_2\rangle\rangle_l^{\delta(\varphi)}$ holds and the definition of $\langle\langle\psi_1 \mathbf{U} \psi_2\rangle\rangle$ forces $|[\psi_1]|_i^{\delta(\varphi)}$ to hold until j, we can conclude that $|[\psi_1]|_i^{\delta(\varphi)}$ holds for all $l \leq i < j$. By the induction hypothesis and the semantics of $\mathbf{U}$ we can then conclude $\pi^{k'+1} \models \varphi$. Combining this with $\pi^{k'} \models \psi_1$, we get $\pi^{k'} \models \varphi$.

Now assume that $\pi^{k'} \models \varphi$. By the equivalence above and the semantics of until we can split the proof into two cases. In the case $\pi^{k'} \models \psi_2$ we can by the induction hypothesis conclude that $|[\psi_2]|_k^{\delta(\varphi)}$ and thus $|[\varphi]|_k^{\delta(\varphi)}$. In the other case we have that $\pi^{k'} \models \psi_1$ and that ψ_2 is satisfied at some later index. Let j' be the smallest such index and denote $j = l + j' - (k'+1)$. Then we know that $|[\psi_2]|_j^{\delta(\varphi)}$ must hold (Prop. 1 and induction hypothesis) and therefore also $\langle\langle\varphi\rangle\rangle_j^{\delta(\varphi)}$ holds. By the semantics of $\mathbf{U}$ we have that $\pi^i \models \psi_1$ for all $k' \leq i < j'$. This fact together with $|[\psi_2]|_j^{\delta(\varphi)}$ implies that $\langle\langle\varphi\rangle\rangle_i^{\delta(\varphi)}$ holds for all $l \leq i \leq j$. Consequently, $|[\varphi]|_k^{\delta(\varphi)}$ holds since we know that $|[\psi_1]|_k^{\delta(\varphi)}$ holds.

Once the correctness of the case $d = \delta(\varphi), i = k$ has been established, the correctness of the remaining cases are easily established. Since the encoding $|[\psi_1 \mathbf{U} \psi_2]|_i^d$ directly follows the recursive semantic definition of $\mathbf{U}$ to compute all the other cases of i and d, and these cases ultimately depend on the proven case we can conclude the encoding is correct for these as well.

Proving correctness for the past operators follows a similar pattern. Consider $\varphi = \psi_1 \mathbf{S} \psi_2$. By the induction hypothesis we can assume that $|[\psi_1]|_i^d$ and $|[\psi_2]|_i^d$ are dealt with correctly. For the past operators the case $i = 0, d = 0$ initialises the encoding while the other cases are computed using the recursive semantic definition of $\mathbf{S}$. The correctness of the initialisation can be argued using the semantics of $\mathbf{S}$ and the induction hypothesis. Again by Prop. 1 we do not need to go deeper than $i = k, d = \delta(\varphi)$. With these ingredients we can establish the correctness of the translation for $|[\psi_1 \mathbf{S} \psi_2]|_i^d$. Performing a case analysis for the rest of the past operators completes the proof. □

The following result can be proved as a straightforward generalisation of the no loop case of [1] to PLTL.

Theorem 2. *If $|[M, \varphi, k]|$ has a satisfying truth assignment where all loop selector variables l_i are false then no matter how the corresponding finite path is extended to an infinite path π, it holds that $\pi \models \varphi$.*

The new encoding is very compact. Let $|I|$ and $|T|$ denote the size of the initial state predicate and the size of the transition relation seen as Boolean circuits.

Theorem 3. *Given a model M, a PLTL formula φ, a bound k, the size of $|[M, \varphi, k]|$ seen as a Boolean circuit is of the order* $O(|I| + k \cdot |T| + k \cdot |\varphi| \cdot \delta(\varphi))$.

Proof. The unrolling of the transition relation and the loop constraints contribute the term $O(|I| + k \cdot |T|)$. For each subformula of φ we add a constant number of constraints at each time point and k constraints at time points $i = k$. Although k constraints that refer to other linear sized constraints ($|[\cdot]|$ and $\langle\langle\cdot\rangle\rangle$) are added at $i = k$, the circuit remains linear because $|[\cdot]|$ and $\langle\langle\cdot\rangle\rangle$ can easily be shared among the constraints. As the loop is virtually unrolled there are $O(k \cdot \delta(\varphi))$ time points for a subformula in the worst case. Combining these two we get $O(|I| + k \cdot |T| + k \cdot |\varphi| \cdot \delta(\varphi))$. $\square$

The translation is linear in all components but since $\delta(\varphi)$ can be $O(|\varphi|)$, it can be seen as worst case quadratic in the formula length. Usually, however, linearity w.r.t. the bound k is the most critical as finding deeper bugs is considered more important than handling very large formulas. When dealing with formulas of fixed $\delta(\varphi)$, e.g. pure LTL formulas, the encoding is linear in $|\varphi|$.

4 Experiments

We have implemented the new encoding in version 2.1.2 of the NuSMV 2 model checker [14]. This facilitates easy comparison against NuSMV, currently the only published PLTL bounded model checker, which is based on the encoding given in [7]. For our implementation of the new PLTL encoding we have adapted the optimisations for the future LTL encoding presented in [13].

We have performed two different sets of experiments. In order to asses how the encodings scale in general, we model checked randomly generated formulas on small randomly generated models. This lets us evaluate how the encodings scale when the size of the formulas is increased or the length of the bound is increased. We also tested the encodings on a few real-life examples to corroborate our findings from the random experiments. In both experiments we measured the size of the generated conjunctive normal form (CNF) expressions. Specifically, we measured the number of variables, clauses and literals (the sum of the lengths of the CNF-clauses) in the generated CNF, and the time to solve the CNF instance. All experiments were run 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. Our implementation and files related to the experiments are available at `http://www.tcs.hut.fi/˜timo/vmcai2005/`.

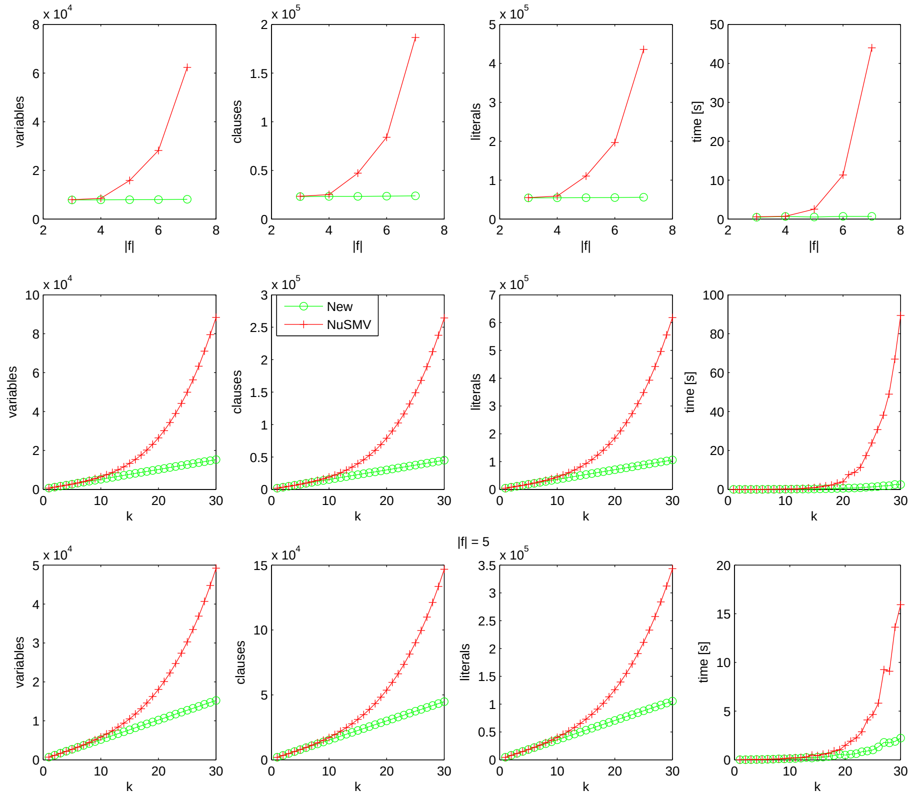

Fig. 4. Random formulae benchmarks.

4.1 Random Formulae

The experiments with random formulae were performed in the following way. Random formulae were generated with the tool described in [17]. We generated 40 formulas for each formula size between three and seven. For each formula we generated a BMC problem for all bounds up to $k = 30$. The BMC problem is constructed using a random Kripke model with 35 states that was generated with techniques described in [17]. The Kripke models have a fairness requirement that requires that at least one of two randomly selected states should appear infinitely often in a counterexample path. This eliminates many short counterexamples and makes the measurement more meaningful for larger values of k.

Figure 4 has twelve plots depicting the results of the benchmarks. In the first row, all results are averaged over the bound and show how the procedures scale with increasing formula size. In the second row, all results are averages over the formula size and show

how the procedures scale in the bound k. For the third row the size of the formula is fixed at 5 and the plots show the average over the 40 formulas. The plots in the first column show the number of variables in the generated CNF. Plots in the second column show the number of clauses and plots in the third column the number of literals in the CNF. The last column has plots which show the time to solve the CNF instances.

From the plots it is clear that the new encoding scales much better than the encoding implemented in NuSMV. This is the case both when considering scaling w.r.t. the size of the formula and the length of the bound.

4.2 Real-life Examples

The second set of experiment were performed on a few real-life examples. We used five models of which four are included in the NuSMV 2 distribution. The models were an alternating bit protocol (*abp*), a bounded resource protocol (*brp*), a distributed mutual exclusion algorithm (*dme*), a pci bus (*pci*) and a 5-bit shift-register (*srg5*). For *abp* and *pci* we checked a property with a counterexample while the properties for *brp*, *dme* and *srg5* were true properties. The template formulae are collected in Table 1.

Table 1. Properties used in real-life benchmarks

Model	Property
abp	$\mathbf{G}\,(p \Rightarrow \mathbf{Y}\,\mathbf{H}\,q)$
brp	$\mathbf{F}\,\mathbf{G}\,(p \Rightarrow \mathbf{O}\,(q \Rightarrow \mathbf{O}\,r))$
dme	$\mathbf{G}\,(p \Rightarrow p\,\mathbf{T}\,(\neg p\,\mathbf{T}\,q))$
pci	$\mathbf{G}\,p \Rightarrow \mathbf{G}\,(q \wedge \mathbf{Y}\,(\neg q \wedge \mathbf{O}\,(r \wedge \mathbf{O}\,(s \wedge \mathbf{O}\,t))) \Rightarrow \mathbf{O}\,(u \wedge \mathbf{O}\,(v \wedge \mathbf{G}\,w)))$
srg5	$\mathbf{F}\,\mathbf{G}\,p \wedge \mathbf{G}\,\mathbf{F}\,q \wedge \mathbf{G}\,\mathbf{F}\,r \Rightarrow \mathbf{F}\,(s\,\mathbf{S}\,(t\,\mathbf{S}\,(u\,\mathbf{S}\,(v\,\mathbf{S}\,w))))$

We measured the number of variables, clauses, and literals in the generated CNF, and the time used to solve an instance at a specific bound k. We also report the cumulative time ($\Sigma\, time$) used to solve all instances up to the given k. The results of the runs can be found in Table 2.

The new encoding was always the fastest. In all cases the new encoding produced the smallest instances w.r.t. all size measures. For *srg5*, NuSMV was not able to proceed further than $k = 18$ because the computer ran out of memory. The reason for this can be found in the apparently at least *cubic* growth w.r.t. the bound k of the encoding for nested binary past operators.

5 Discussion and Conclusions

We have presented an encoding of the BMC problem for PLTL. The encoding is linear in the bound k unlike the encoding by Benedetti and Cimatti [7]. In the general case the encoding is quadratic in the size of the formula but if we fix the past operator depth, the encoding is also linear in the size of the formula. Experiments confirm that the

Table 2. Real-life benchmarks.

Model	k	NuSMV					New				
		vars	*clauses*	*literals*	*time*	Σ*time*	*vars*	*clauses*	*literals*	*time*	Σ*time*
abp	16	25,175	74,208	174,644	104	342	22,827	67,116	158,096	52.5	269
brp	10	14,115	41,228	98,304	0.9	2.5	8,961	25,736	62,156	0.7	2.2
	15	30,225	89,218	211,334	4.6	15.9	13,346	38,536	93,076	1.5	7.5
	20	56,935	169,008	398,564	19.2	75.6	17,731	51,336	123,996	3.2	19.7
dme	10	49,776	139,740	338,752	10.3	15.1	28,855	76,947	192,235	6.3	17.5
	15	139,071	404,485	962,837	98.9	171	42,685	115,282	288,030	15.5	70.2
	20	346,166	1,022,630	2,411,522	1,017	1,812	56,515	153,617	383,825	41.2	214
pci	10	81,285	242,133	567,029	96.7	188	60,456	179,616	421,156	69.8	151
	15	159,885	477,358	1,116,914	2,441	5,408	90,611	269,491	631,891	888	2,422
	18	227,357	679,429	1,589,029	2,557	19,119	108,704	323,416	758,332	867	11,992
srg5	10	137,710	412,952	963,900	53.6	90.7	1,655	4,757	11,445	0.0	0.1
	18	1,264,988	3,794,698	8,854,918	14,914	33,708	2,999	8,677	20,869	0.2	0.9
	30	N/A	N/A	N/A	N/A	N/A	5,015	14,557	35,005	0.7	6.6

encoding is more compact and efficient than the original encoding. In the experiments our encoding scales better both in the bound k and in the size of the formula.

After having independently discovered our new encoding we very recently became aware of a manuscript [19] discussing an alternative approach to bounded model checking of PLTL. Our approach differs in many ways from that of [19], the main differences being that their approach does not perform any virtual unrolling at all and that their starting point is the so called SNF encoding for BMC [15]. It is easy to modify our encoding not to virtually unroll (k,l)-loops by defining the past operator depth function $\delta(\varphi)$ to return the constant 0 for all formulas irregardless of their past operator depth. However, in this case the encoding would *not* remain sound for formulas with looping counterexamples. For example, verifying the formula $\neg\mathbf{G}\,\mathbf{F}\,\mathbf{Y}\,\mathbf{Y}\,\mathbf{Y}\,(x=0)$ on our running example would result in a counterexample at $k=6$ even though the formula holds. We do not see how soundness for full PLTL could be achieved without performing virtual unrolling.

If we restrict ourselves to searching for non-looping counterexamples (not all PLTL formulas have non-looping counterexamples) or to specifications in some subset of full PLTL, the virtual unrolling could be discarded while maintaining soundness. However, although virtual unrolling has a small overhead it also has benefits. For example, model checking formula (1) on our running example requires the transition relation to be unrolled 6 times with our encoding but the encoding of [19] requires the transition relation to be unrolled 11 times before the first witness is found. Due to the efficiency of our encoding the overhead of virtual unrolling is small and the potential gain in using smaller bounds can be significant. We argue that our approach can be more efficient than [19], at least in the cases where the encoding is dominated by the system transition relation size ($|T| \gg |\varphi|$) and the counterexample can be detected earlier by virtual unrolling. In our opinion the new encoding is also easier to understand and implement than that of [19].

There are still possibilities for improving the performance of our encoding and extending it to other uses. The bounded satisfiability problem asks if there is *any* model represented as a bounded path of length k for a given PLTL formula ψ. The new encoding can easily be extended to solve this problem by removing all constraints set by the transition relation on the state variables. If the encoding is viewed as a Boolean circuit, the loop selector variables l_i and the atomic propositions (and their negations) are viewed as input gates, then the encoding generates a monotonic Boolean circuit. This could be exploited in specific SAT solver optimisations. Another possible topic for future research is considering incremental encodings for BMC in the spirit of [20].

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. Biere, A., Clarke, E.M., Raimi, R., Zhu, Y.: Verifying safety properties of a Power PC microprocessor using symbolic model checking without BDDs. In: Computer Aided Verification (CAV 1999). Volume 1633 of LNCS., Springer (1999) 60–71
3. Copty, F., Fix, L., Fraer, R., Giunchiglia, E., Kamhi, G., Tacchella, A., Vardi, M.Y.: Benefits of bounded model checking at an industrial setting. In: Computer Aided Verification (CAV 2001). Volume 2102 of LNCS., Springer (2001) 436–453
4. Strichman, O.: Accelerating bounded model checking of safety properties. Formal Methods in System Design **24** (2004) 5–24
5. Havelund, K., Rosu, G.: Synthesizing monitors for safety properties. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2002). Volume 2280 of LNCS., Springer (2002) 342–356
6. Gastin, P., Oddoux, D.: LTL with past and two-way very-weak alternating automata. In: Mathematical Foundations of Computer Science 2003 (MFCS 2003). Volume 2747 of LNCS., Springer (2003) 439–448
7. 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
8. Lichtenstein, O., Pnueli, A., Zuck, L.D.: The glory of the past. In: Logic of Programs. Volume 193 of LNCS., Springer (1985) 196–218
9. Kamp, J.: Tense Logic and the Theory of Linear Order. PhD thesis, University of California, Los Angeles, California (1968)
10. Gabbay, D.M., Pnueli, A., Shelah, S., Stavi, J.: On the temporal basis of fairness. In: Conference Record of the Seventh Annual ACM Symposium on Principles of Programming Languages, Las Vegas, Nevada, ACM (1980) 163–173
11. Laroussinie, F., Markey, N., Schnoebelen, P.: Temporal logic with forgettable past. In: 17th IEEE Symp. Logic in Computer Science (LICS 2002), IEEE Computer Society Press (2002) 383–392
12. Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logics. Journal of the ACM **32** (1985) 733–749
13. Latvala, T., Biere, A., Heljanko, K., Junttila, T.: Simple bounded LTL model checking. In: Formal Methods in Computer-Aided Design (FMCAD 2004). Volume 3312 of LNCS., Springer (2004) 186–200
14. 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
15. 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

16. Kupferman, O., Vardi, M.: Model checking of safety properties. Formal Methods in System Design **19** (2001) 291–314
17. Tauriainen, H., Heljanko, K.: Testing LTL formula translation into Büchi automata. STTT - International Journal on Software Tools for Technology Transfer **4** (2002) 57–70
18. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the 38th Design Automation Conference, IEEE (2001)
19. Cimatti, A., Roveri, M., Sheridan, D.: Bounded verification of past LTL. In: Formal Methods in Computer-Aided Design (FMCAD 2004). Volume 3312 of LNCS., Springer (2004) 245–259
20. Eén, N., Sörensson, N.: Temporal induction by incremental SAT solving. In: First International Workshop on Bounded Model Checking. Volume 89 of ENTCS., Elsevier (2003)