Incremental and Complete Bounded Model Checking for Full PLTL

Keijo Heljanko*, Tommi Junttila**, and Timo Latvala***

Laboratory for Theoretical Computer Science,
Helsinki University of Technology,
P.O. Box 5400, FI-02015 TKK, Finland
{Keijo.Heljanko, Tommi.Junttila, Timo.Latvala}@tkk.fi

Abstract. Bounded model checking is an efficient method for finding bugs in system designs. The major drawback of the basic method is that it cannot prove properties, only disprove them. Recently, some progress has been made towards proving properties of LTL. We present an *incremental* and *complete* bounded model checking method for the full linear temporal logic with past (PLTL). Compared to previous works, our method both improves and extends current results in many ways: (i) our encoding is incremental, resulting in improvements in performance, (ii) we can prove non-existence of a counterexample at shallower depths in many cases, and (iii) we support full PLTL. We have implemented our method in the NuSMV2 model checker and report encouraging experimental results.

Keywords: Bounded Model Checking, Incremental, Complete, PLTL, NuSMV.

1 Introduction

Bounded model checking (BMC) [1] has established itself as an efficient method of finding bugs to LTL specifications from system designs. The method works by searching for witnesses of length k to the negation of the specification. This bounded search problem is translated to the propositional satisfiability problem (SAT) and a SAT solver is used to get an answer.

A problem with BMC is knowing how large the bound k should be, before we can be sure that no counterexample exists. This bound, referred to as the *completeness threshold* [2], depends on the system, the property, and how the problem is mapped to SAT. Computing a tight bound on the completeness threshold is a challenging problem.

One method of finding small completeness thresholds for invariant properties is using induction. Sheeran et al. [3] present an inductive scheme for invariants. They show that invariants can be proven by automatically strengthening induction to show that no path of length k breaks the invariant and that there is no initialised loop free path of length $k+1$. The longest initialised loop free path in the state graph is called the *recurrence diameter* [1]. The inductive method can easily be generalised to safety proper-

* Supported by the Academy of Finland (projects 53695, 211025, 213397, research fellow post).
** Supported by the Academy of Finland (projects 53695, 211025).
*** Supported by the Helsinki Graduate School in Computer Science, the Academy of Finland (projects 53695, 211025), and the Nokia Foundation.

K. Etessami and S.K. Rajamani (Eds.): CAV 2005, LNCS 3576, pp. 98–111, 2005.

ties and also to LTL properties by using the liveness-to-safety transformation presented in [4], generalised to BDD based model checking of PLTL in [5]. However, the liveness-to-safety transformation doubles the number of state variables in the model [4,5], which is unnecessary for BMC. This increases the size of the already large loop free predicate that in many cases already is the biggest bottleneck. The method of Sheeran et al. has been generalised in various ways. Kroening and Strichman [2] show that the size of loop free predicate can be optimised to $O(k \log^2 k)$ (vs. $O(k^2)$) using sorting networks. They also suggest ways to leave out state variables from the loop free predicate to improve efficiency while maintaining completeness. Two papers that consider strengthening of induction without always doing deeper BMC queries are [6,7].

Recently, some work has focused on computing a completeness threshold for general LTL properties. Clarke et al. [8] show how the completeness threshold can be computed for general LTL properties by computing the recurrence diameter of the product of the system and a Büchi automaton. Awedh and Somenzi [9] apply the same approach, but they use a refined method for calculating the completeness threshold. Both papers have the problem that they use an explicit representation of Büchi automata in their implementations and thus potentially using an exponential number of state bits in the size of the formula to represent the Büchi automaton. Furthermore, they do not use generalised Büchi automata to represent LTL properties and might therefore have to proceed deeper to prove properties than the method proposed in this paper or methods based on generalised Büchi automata.

McMillan [10] uses interpolants derived from unsatisfiability proofs of BMC counterexample queries to overapproximate reachability. The deeper the BMC query is, the more exact the overapproximation is. The method is complete and can be extended to LTL through the liveness to safety transformation [4]. A weakness of the method is that the unsatisfiability proofs can be of exponential size and cause a blow up.

A promising technique for improving the performance of BMC is using *incremental* SAT solving. When a solver is faced with a sequence of related problems, learning clauses (see e.g., [11]) from the previous problems can drastically improve the solution time for the next problem and thus for the whole sequence. BMC is a natural candidate for incremental solving as two BMC instances for bounds k and $k+1$ are very similar. Strichman [12] and Whittemore et al. [13] were among the first to consider incremental BMC. Both papers presented frameworks for transforming a SAT problem to the next in the sequence by adding and removing clauses from the current problem instance. Eén and Sörensson [14] consider incremental BMC combined with the inductive scheme presented in [3]. Their approach is based on using the special syntactic structure of the BMC encoding for invariants to forward all learned clauses, and therefore they do not need to perform any potentially expensive conflict analysis between two sequential problem instances. Jin and Somenzi [15] present efficient ways of filtering conflict clauses when creating the next problem instance. In [16] a framework for incremental SAT solving based on incremental compilation of the encoding to SAT is presented, however, their PLTL encoding is based on the original and inefficient encoding of [17].

Our contribution is a BMC encoding specifically adapted to an incremental setting based on the PLTL encoding presented in [18]. The encoding has been designed to allow easy separation of constraints that remain active over all instances and constraints that should be removed when the bound is increased. In addition, we have tried to minimise

the number of constraints that must be removed in order to allow maximal learning in a solver independent fashion. Both of these are achieved while maintaining the efficiency of the original encoding [18]. Additionally, our encoding is able to prove properties of full PLTL with smaller bounds than previous methods for LTL [8,9], as these papers employ a method for translating generalised Büchi automata to standard (non-generalised) Büchi automata in a way which does not preserve the minimal length of counterexamples. We have implemented our method in the NuSMV model checker [19] and present promising experimental results.

2 Bounded Model Checking for LTL

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*. Given $l > 0$, an infinite path $\pi = s_0 s_1 s_2 \ldots$ of states is a (k,l)-loop, if $\pi = (s_0 s_1 \ldots s_{l-1})(s_l \ldots s_k)^\omega$. In a finite state system we can restrict ourselves to searching for counterexamples to an LTL (and also PLTL) property representable as a (k,l)-loop. 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.

2.1 PLTL

PLTL is a commonly used specification logic with both past and future temporal operators. We refer to the sublogic consisting of only the future temporal operators as LTL. The semantics of an PLTL formula is defined along infinite paths $\pi = s_0 s_1 \ldots$ of states. Each state s_i is labelled by a labelling function L such that $L(s_i) \in 2^{AP}$, where AP is a set of atomic propositions. The states are part of a model M with a total transition relation T and initial state constraint I. Let π^i denote the suffix of π starting from the i:th state. The semantics is as follows:

$$
\begin{array}{ll}
\pi^i \models \psi & \Leftrightarrow \psi \in L(s_i) \text{ for } \psi \in AP. \\
\pi^i \models \neg\psi & \Leftrightarrow \pi^i \not\models \psi. \\
\pi^i \models \psi_1 \vee \psi_2 & \Leftrightarrow \pi^i \models \psi_1 \text{ or } \pi^i \models \psi_2. \\
\pi^i \models \psi_1 \wedge \psi_2 & \Leftrightarrow \pi^i \models \psi_1 \text{ and } \pi^i \models \psi_2. \\
\pi^i \models \mathbf{X}\psi & \Leftrightarrow \pi^{i+1} \models \psi. \\
\pi^i \models \psi_1 \mathbf{U} \psi_2 & \Leftrightarrow \exists n \geq i \text{ such that } \pi^n \models \psi_2 \text{ and } \pi^j \models \psi_1 \text{ for all } i \leq j < n. \\
\pi^i \models \psi_1 \mathbf{R} \psi_2 & \Leftrightarrow \forall n \geq i, \pi^n \models \psi_2 \text{ or } \pi^j \models \psi_1 \text{ for some } i \leq j < n. \\
\pi^i \models \mathbf{Y}\psi & \Leftrightarrow i > 0 \text{ and } \pi^{i-1} \models \psi. \\
\pi^i \models \mathbf{Z}\psi & \Leftrightarrow i = 0 \text{ or } \pi^{i-1} \models \psi. \\
\pi^i \models \mathbf{O}\psi & \Leftrightarrow \pi^j \models \psi \text{ for some } 0 \leq j \leq i. \\
\pi^i \models \mathbf{H}\psi & \Leftrightarrow \pi^j \models \psi \text{ for all } 0 \leq j \leq i. \\
\pi^i \models \psi_1 \mathbf{S} \psi_2 & \Leftrightarrow \pi^j \models \psi_2 \text{ for some } 0 \leq j \leq i \text{ and } \pi^n \models \psi_1 \text{ for all } j < n \leq i. \\
\pi^i \models \psi_1 \mathbf{T} \psi_2 & \Leftrightarrow \text{ for all } 0 \leq j \leq i : \pi^j \models \psi_2 \text{ or } \pi^n \models \psi_1 \text{ for some } j < n \leq i.
\end{array}
$$

When $\pi^0 \models \psi$ we simply write $\pi \models \psi$. With $M \models \psi$ we denote that $\pi \models \psi$ for all infinite initialised paths π of M. 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 appear only in front of atomic propositions. This can be accomplished by using dualities of the form $\neg(\psi_1\,\mathbf{U}\,\psi_2) \equiv \neg\psi_1\,\mathbf{R}\,\neg\psi_2$, which are available for all operators. In the rest of the paper 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:*

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

The set of subformulas of a PLTL formula ψ is denoted by $cl(\psi)$ and is defined as the smallest set satisfying the following conditions:

$$\begin{array}{llll}
\psi \in cl(\psi), & & & \\
\text{if } \circ\,\phi \in cl(\psi) & \text{for } \circ \in \{\neg, \mathbf{X}, \mathbf{F}, \mathbf{G}, \mathbf{Y}, \mathbf{Z}, \mathbf{O}, \mathbf{H}\} & \text{then } \phi \in cl(\psi), \text{ and} \\
\text{if } \psi_1 \circ \psi_2 \in cl(\psi) & \text{for } \circ \in \{\vee, \wedge, \mathbf{U}, \mathbf{R}, \mathbf{S}, \mathbf{T}\} & \text{then } \psi_1, \psi_2 \in cl(\psi).
\end{array}$$

2.2 Incremental Bounded Model Checking for LTL

We start by presenting an incremental encoding for LTL based on our simple BMC encoding [20,18]. There are a few considerations that need to be taken into account for a good incremental encoding. First of all, the encoding needs to be formulated so that it is easy to derive the case $k = i+1$ from $k = i$. This is done by separating the encoding to a *k-invariant* part and a *k-dependent* part. The information learned from the k-invariant constraints can be reused when the bound is increased while the information learned from the k-dependent constraints needs to be discarded. Thus we try to minimise the use of k-dependent constraints in our encoding. The so called *Base constraints* are also k-invariant, but they are conditions that are constant for all $0 \leq i \leq k$.

Given an LTL property ψ, in our new encoding the state variables of the system are split at each time i to the actual state variables s_i of the system, to the set of variables for all subformulas $|[s_\psi]|_i$ (one for each subformula $\varphi \in cl(\psi)$), and to the set of variables for the so called auxiliary translation $\langle\langle s_\psi \rangle\rangle_i$ (one for each $\mathbf{F}, \mathbf{G}, \mathbf{U}, \mathbf{R}$ subformula $\varphi \in cl(\psi)$). The encoding also contains a few additional variables which will be referred to explicitly. The rules of the encoding are given as a set of Boolean constraints.

Paths of length k are encoded using *model constraints*. They encode 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* employ $k+2$ fresh *loop selector variables* $l_0, \ldots, l_{k+1}$. They constrain the finite path of the system to be: (a) a finite path, in which case none of the $l_0, \ldots, l_{k+1}$ variables is true, or (b) a (k,i)-loop, in which case the variable l_i is true and all other l_j variables are false. Many k-dependent constraints of our original encoding [20] have been eliminated by introducing a new special system state s_E with fresh (unconstrained) state variables acting as a proxy state for the endpoint of the path. In the k-dependent part the proxy state s_E is constrained to be equivalent to s_k. The variable InLoop_i is true iff the state s_i belongs to the loop part of a (k,l)-loop. The variable LoopExists is k-dependent, and true when π is a (k,l)-loop and false otherwise. This is encoded by conjuncting the constraints below and denoted by $|[\mathit{LoopConstraints}]|_k$:

Base	$l_0 \Leftrightarrow \bot$
	$\text{InLoop}_0 \Leftrightarrow \bot$
$k-$invariant	$l_i \Rightarrow (s_{i-1} = s_E)$
$1 \leq i \leq k$	$\text{InLoop}_i \Leftrightarrow \text{InLoop}_{i-1} \vee l_i,$
	$\text{InLoop}_{i-1} \Rightarrow \neg l_i$
$k-$dependent	$l_{k+1} \Leftrightarrow \bot$
	$s_E \Leftrightarrow s_k$
	$\text{LoopExists} \Leftrightarrow \text{InLoop}_k$

The *LTL constraints* restrict the bounded path defined by the model constraints and loop constraints to witnesses of the given LTL formula ψ. One intuition for understanding the encoding is given by the fact that for (k,l)-loops the semantics of CTL and LTL coincide [21,22,20]. Thus for (k,l)-loops the encoding can be seen as a CTL model checker for the single (k,l)-loop selected by the loop constraints.

As mentioned above, the translation for LTL uses for each time point i and each subformula $\varphi \in cl(\psi)$ one state variable denoted by $|[\varphi]|_i^d$. The superscript d is needed in order to extend the encoding to the PLTL case but $d = 0$ holds for all LTL properties. Note that although the transition relation is unrolled up to $i = k$, there are formula state variables up to $i = k+1$. We will now start introducing constraints on these free subformula variables.

We use a proxy state s_L with associated (free) formula variables $|[\varphi]|_L^d$ to simplify the notation a bit. In the no-loop case they will all be forced to false in order to safely underapproximate the semantics of LTL. In the loop case they will pick up the truth value for each subformula in the loop state s_L, i.e. the successor state of s_E. The k-dependent rules bind the truth values of $|[\varphi]|_E^d$ to $|[\varphi]|_k^d$ and the truth values of $|[\varphi]|_{k+1}^d$ to $|[\varphi]|_L^{min(d+1,\delta(\varphi))}$ (jump to the next unrolling level $d+1$, needed for the PLTL case).

For all $\varphi \in cl(\psi)$ the following constraints are created:

	$0 \leq d \leq \delta(\varphi)$
Base	$\neg \mathit{LoopExists} \Rightarrow \left(\lvert[\varphi]\rvert_L^d \Leftrightarrow \bot \right)$
$k-$invariant, $1 \leq i \leq k$	$l_i \Rightarrow \left(\lvert[\varphi]\rvert_L^d \Leftrightarrow \lvert[\varphi]\rvert_i^d \right)$
$k-$dependent	$\lvert[\varphi]\rvert_E^d \Leftrightarrow \lvert[\varphi]\rvert_k^d$ $\lvert[\varphi]\rvert_{k+1}^d \Leftrightarrow \lvert[\varphi]\rvert_L^{min(d+1,\delta(\varphi))}$

Atomic propositions, their negations and the basic Boolean connectives can be dealt with straightforwardly in a k-invariant fashion. The encoding for the temporal subformulas follows the recursive semantic definition of LTL temporal subformulas. The Base encoding guarantees in the (k,l)-loop case the following. If an until holds at s_E then the ψ_2 subformula will hold in some state in the loop. In the release case if ψ_2 is true on all states along the loop, then also the release formula holds at s_E. In the case of a non-looping path we do not have to care about eventualities and thus the Base encoding is disabled. Rules for finally and globally are just special case optimisations of these two.

	φ	
Base	$\mathbf{F}\phi$	$\mathit{LoopExists} \Rightarrow \left(\lvert[\mathbf{F}\phi]\rvert_E^{\delta(\phi)} \Rightarrow \langle\langle \mathbf{F}\phi \rangle\rangle_E^{\delta(\phi)} \right)$
	$\mathbf{G}\phi$	$\mathit{LoopExists} \Rightarrow \left(\lvert[\mathbf{G}\phi]\rvert_E^{\delta(\phi)} \Leftarrow \langle\langle \mathbf{G}\phi \rangle\rangle_E^{\delta(\phi)} \right)$
	$\psi_1 \mathbf{U} \psi_2$	$\mathit{LoopExists} \Rightarrow \left(\lvert[\psi_1 \mathbf{U} \psi_2]\rvert_E^{\delta(\varphi)} \Rightarrow \langle\langle \mathbf{F}\psi_2 \rangle\rangle_E^{\delta(\psi_2)} \right)$
	$\psi_1 \mathbf{R} \psi_2$	$\mathit{LoopExists} \Rightarrow \left(\lvert[\psi_1 \mathbf{R} \psi_2]\rvert_E^{\delta(\varphi)} \Leftarrow \langle\langle \mathbf{G}\psi_2 \rangle\rangle_E^{\delta(\psi_2)} \right)$
$k-$invariant	p	$\lvert[p]\rvert_i^d \Leftrightarrow p_i$
	$\neg p$	$\lvert[\neg p]\rvert_i^d \Leftrightarrow \neg p_i$
$0 \leq i \leq k$,	$\psi_1 \wedge \psi_2$	$\lvert[\psi_1 \wedge \psi_2]\rvert_i^d \Leftrightarrow \lvert[\psi_1]\rvert_i^d \wedge \lvert[\psi_2]\rvert_i^d$
$0 \leq d \leq \delta(\varphi)$	$\psi_1 \vee \psi_2$	$\lvert[\psi_1 \vee \psi_2]\rvert_i^d \Leftrightarrow \lvert[\psi_1]\rvert_i^d \vee \lvert[\psi_2]\rvert_i^d$
	$\mathbf{X}\phi$	$\lvert[\mathbf{X}\phi]\rvert_i^d \Leftrightarrow \lvert[\phi]\rvert_{i+1}^d$
	$\mathbf{F}\phi$	$\lvert[\mathbf{F}\phi]\rvert_i^d \Leftrightarrow \lvert[\phi]\rvert_i^d \vee \lvert[\mathbf{F}\phi]\rvert_{i+1}^d$
	$\mathbf{G}\phi$	$\lvert[\mathbf{G}\phi]\rvert_i^d \Leftrightarrow \lvert[\phi]\rvert_i^d \wedge \lvert[\mathbf{G}\phi]\rvert_{i+1}^d$
	$\psi_1 \mathbf{U} \psi_2$	$\lvert[\psi_1 \mathbf{U} \psi_2]\rvert_i^d \Leftrightarrow \lvert[\psi_2]\rvert_i^d \vee \left(\lvert[\psi_1]\rvert_i^d \wedge \lvert[\psi_1 \mathbf{U} \psi_2]\rvert_{i+1}^d \right)$
	$\psi_1 \mathbf{R} \psi_2$	$\lvert[\psi_1 \mathbf{R} \psi_2]\rvert_i^d \Leftrightarrow \lvert[\psi_2]\rvert_i^d \wedge \left(\lvert[\psi_1]\rvert_i^d \vee \lvert[\psi_1 \mathbf{R} \psi_2]\rvert_{i+1}^d \right)$

The *auxiliary encoding* $\langle\langle\varphi\rangle\rangle_i^d$ is used to enforce eventualities. As it is not referenced in the no-loop case, we consider only the (k,l)-loop case. In that case $\langle\langle \mathbf{F}\phi \rangle\rangle_E^d$ ($\langle\langle \mathbf{G}\phi \rangle\rangle_E^d$) is true iff $\mathbf{F}\phi$ ($\mathbf{G}\phi$) holds in the end state s_E (in unrolling d for the PLTL case). For $\langle\langle \mathbf{F}\phi \rangle\rangle_E^d$, this is implemented by requiring that $\lvert[\phi]\rvert_i^d$ holds in at least one state in the loop, while $\langle\langle \mathbf{G}\phi \rangle\rangle_E^d$ requires that $\lvert[\psi]\rvert_i^d$ holds in all states of the loop. The formulation

of the auxiliary encoding is one of the main differences to the encoding of [18] and allows several new optimisations in Sect. 5.

	φ	
Base	$\mathbf{F}\phi$	$\langle\langle \mathbf{F}\phi \rangle\rangle_0^{\delta(\phi)} \Leftrightarrow \bot$
	$\mathbf{G}\phi$	$\langle\langle \mathbf{G}\phi \rangle\rangle_0^{\delta(\phi)} \Leftrightarrow \top$
$k-$invariant	$\mathbf{F}\phi$	$\langle\langle \mathbf{F}\phi \rangle\rangle_i^{\delta(\phi)} \Leftrightarrow \langle\langle \mathbf{F}\phi \rangle\rangle_{i-1}^{\delta(\phi)} \vee \left(\mathrm{InLoop}_i \wedge \lvert[\phi]\rvert_i^{\delta(\phi)}\right)$
$1 \leq i \leq k$	$\mathbf{G}\phi$	$\langle\langle \mathbf{G}\phi \rangle\rangle_i^{\delta(\phi)} \Leftrightarrow \langle\langle \mathbf{G}\phi \rangle\rangle_{i-1}^{\delta(\phi)} \wedge \left(\neg\mathrm{InLoop}_i \vee \lvert[\phi]\rvert_i^{\delta(\phi)}\right)$
$k-$dependent	$\mathbf{F}\phi$	$\langle\langle \mathbf{F}\phi \rangle\rangle_E^{\delta(\phi)} \Leftrightarrow \langle\langle \mathbf{F}\phi \rangle\rangle_k^{\delta(\phi)}$
	$\mathbf{G}\phi$	$\langle\langle \mathbf{G}\phi \rangle\rangle_E^{\delta(\phi)} \Leftrightarrow \langle\langle \mathbf{G}\phi \rangle\rangle_k^{\delta(\phi)}$

Our incremental encoding is close to the symbolic Büchi automata construction for LTL formulae presented in [23], were it to be adapted to an incremental BMC setting (see also [5] for more on this connection). By restricting the encoding to looping counterexamples, and replacing our auxiliary encoding with an encoding that would track the Büchi acceptance conditions for until and finally formulas, the encoding for LTL would essentially correspond to an incremental BMC version of [23].

For LTL we have now generated the full encoding which will be extended to PLTL in the next section, where we will also present the correctness claims. Let $\lvert[M,\psi]\rvert_k$ denote the encoding above with the bound set to k. Intuitively, the LTL formula ψ has a bounded witness of length k in M iff the encoding is satisfiable. Moreover, when moving from an instance with bound $k = i$ to an instance with bound $k = i+1$ only the k-dependent constraints (and of course things learned from them by the SAT solver) need to be discarded.

3 Generalising to PLTL

The generalisation to full PLTL is based on our BMC encoding for PLTL [18]. With past operators, encoding the BMC problem is slightly more complicated. The main source of complexity is the fact that when a (k,l)-loop is traversed forward, each time we reach the loop point, the future looks the same but the past is different. Fortunately, the ability of a PLTL formula ψ to distinguish between different loop points is bounded by the past formula nesting depth $\delta(\psi)$ [24,17]. Therefore the evaluations of past operators inside the loop will eventually stabilise. This can be exploited in BMC by *virtually unrolling* the (k,l)-loop $\delta(\psi)$ times, to ensure that the evaluations of the past operators have stabilised. Consider a simple counter that increments a variable x at each step. When x reaches five the value of x is reset to two. The counter has the single execution $\pi = 012(3452)^\omega$ that corresponds to a $(6,3)$-loop. Let $\varphi = (x = 3 \wedge \mathbf{O}\,(x = 4 \wedge \mathbf{O}\,(x = 5)))$ which has $\delta(\varphi) = 2$. At the loop state $i = 3$ we have that $\pi^3 \not\models \varphi$. At the next corresponding time step $i = 7$ the formula φ still does not hold. However, when we reach $i = 11$ the formula φ has stabilised and $\pi^{11} \models \varphi$. In Fig. 1 the $(6,3)$-loop of the counter

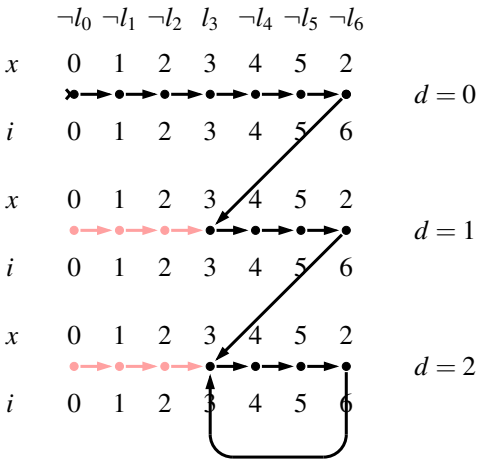

Fig. 1. Virtual unrolling for $\delta(\psi) = 2$ with a $(6,3)$-loop

system has been virtually unrolled to depth $d = 2$.. The corresponding state to $i = 11$ is at depth $d = 2$ at $i = 3$. The encoding is modified by introducing for each time point i and each subformula $\varphi \in cl(\psi)$ the formula variables $|[\varphi]|_i^d$, where $0 \leq d \leq \delta(\varphi)$. Please refer to our paper on BMC for PLTL for details [18]. The first rule of the encoding for PLTL is based on the property mentioned above: PLTL can only distinguish between different unrollings of the loop up to the past depth of the formula.

	$0 \leq d \leq \delta(\varphi)$
$k-$invariant	$\|[\varphi]\|_i^d := \|[\varphi]\|_i^{\delta(\varphi)}$, when $d > \delta(\varphi)$

The encoding for the past operators is very similar to our encoding presented in [18]. The basic idea is to use the recursive definitions of the past temporal operators. The history past operators should evaluate corresponds to the straight black arrows of Fig. 1. An if-then-else construct is used to determine if the previous time point in the past is s_{i-1} at the current depth or s_E at the previous depth $d-1$ (see Fig. 1). We define $ite(a,b,c) \equiv (a \wedge b) \vee (\neg a \wedge c)$ to obtain a more compact representation of the encoding.

The largest change to our previous encoding [18] is the translation at $i = 0$ when $d > 0$, which is now identical to $i = 0$ of $d = 0$. This is done on purpose to make all the virtual unrollings identical in the no-loop case (this allows some of the optimisations given in Sect. 5). Moreover, references to subformulas at state s_k have been replaced with references to subformulas at the proxy state s_E making all the constraints k-invariant.

Combining all the components the encoding $|[M,\psi]|_k$ for PLTL is defined to be:

$$|[M,\psi]|_k := |[M]|_k \wedge |[LoopConstraints]|_k \wedge |[\psi]|_0^0.$$

	φ	
$k-$invariant	$\mathbf{Y}\phi$	$\lvert[\mathbf{Y}\phi]\rvert_0^0 \Leftrightarrow \bot, \lvert[\mathbf{Y}\phi]\rvert_i^0 \Leftrightarrow \lvert[\phi]\rvert_{i-1}^0$
	$\mathbf{Z}\phi$	$\lvert[\mathbf{Z}\phi]\rvert_0^0 \Leftrightarrow \top, \lvert[\mathbf{Z}\phi]\rvert_i^0 \Leftrightarrow \lvert[\phi]\rvert_{i-1}^0$
$1 \leq i \leq k+1$,	$\mathbf{O}\phi$	$\lvert[\mathbf{O}\phi]\rvert_0^0 \Leftrightarrow \lvert[\phi]\rvert_0^0, \lvert[\mathbf{O}\phi]\rvert_i^0 \Leftrightarrow \lvert[\phi]\rvert_i^0 \vee \lvert[\mathbf{O}\phi]\rvert_{i-1}^0$
$d = 0$	$\mathbf{H}\phi$	$\lvert[\mathbf{H}\phi]\rvert_0^0 \Leftrightarrow \lvert[\phi]\rvert_0^0, \lvert[\mathbf{H}\phi]\rvert_i^0 \Leftrightarrow \lvert[\phi]\rvert_i^0 \wedge \lvert[\mathbf{H}\phi]\rvert_{i-1}^0$
	$\psi_1 \mathbf{S} \psi_2$	$\lvert[\psi_1 \mathbf{S} \psi_2]\rvert_0^0 \Leftrightarrow \lvert[\psi_2]\rvert_0^0, \lvert[\psi_1 \mathbf{S} \psi_2]\rvert_i^0 \Leftrightarrow \lvert[\psi_2]\rvert_i^0 \vee \left(\lvert[\psi_1]\rvert_i^0 \wedge \lvert[\psi_1 \mathbf{S} \psi_2]\rvert_{i-1}^0\right)$
	$\psi_1 \mathbf{T} \psi_2$	$\lvert[\psi_1 \mathbf{T} \psi_2]\rvert_0^0 \Leftrightarrow \lvert[\psi_2]\rvert_0^0, \lvert[\psi_1 \mathbf{T} \psi_2]\rvert_i^0 \Leftrightarrow \lvert[\psi_2]\rvert_i^0 \wedge \left(\lvert[\psi_1]\rvert_i^0 \vee \lvert[\psi_1 \mathbf{T} \psi_2]\rvert_{i-1}^0\right)$
$k-$invariant	$\mathbf{Y}\phi$	$\lvert[\mathbf{Y}\phi]\rvert_0^d \Leftrightarrow \bot, \lvert[\mathbf{Y}\phi]\rvert_i^d \Leftrightarrow ite\left(l_i, \lvert[\phi]\rvert_E^{d-1}, \lvert[\phi]\rvert_{i-1}^d\right)$
	$\mathbf{Z}\phi$	$\lvert[\mathbf{Z}\phi]\rvert_0^d \Leftrightarrow \top, \lvert[\mathbf{Z}\phi]\rvert_i^d \Leftrightarrow ite\left(l_i, \lvert[\phi]\rvert_E^{d-1}, \lvert[\phi]\rvert_{i-1}^d\right)$
$1 \leq i \leq k+1$,	$\mathbf{O}\phi$	$\lvert[\mathbf{O}\phi]\rvert_0^d \Leftrightarrow \lvert[\phi]\rvert_0^0, \lvert[\mathbf{O}\phi]\rvert_i^d \Leftrightarrow \lvert[\phi]\rvert_i^d \vee ite\left(l_i, \lvert[\varphi]\rvert_E^{d-1}, \lvert[\varphi]\rvert_{i-1}^d\right)$
$1 \leq d \leq \delta(\varphi)$	$\mathbf{H}\phi$	$\lvert[\mathbf{H}\phi]\rvert_0^d \Leftrightarrow \lvert[\phi]\rvert_0^0, \lvert[\mathbf{H}\phi]\rvert_i^d \Leftrightarrow \lvert[\phi]\rvert_i^d \wedge ite\left(l_i, \lvert[\varphi]\rvert_E^{d-1}, \lvert[\varphi]\rvert_{i-1}^d\right)$
	$\psi_1 \mathbf{S} \psi_2$	$\lvert[\psi_1 \mathbf{S} \psi_2]\rvert_0^d \Leftrightarrow \lvert[\psi_2]\rvert_0^0, \lvert[\psi_1 \mathbf{S} \psi_2]\rvert_i^d \Leftrightarrow \lvert[\psi_2]\rvert_i^d \vee \left(\lvert[\psi_1]\rvert_i^d \wedge ite\left(l_i, \lvert[\varphi]\rvert_E^{d-1}, \lvert[\varphi]\rvert_{i-1}^d\right)\right)$
	$\psi_2 \mathbf{T} \psi_1$	$\lvert[\psi_1 \mathbf{T} \psi_2]\rvert_0^d \Leftrightarrow \lvert[\psi_2]\rvert_0^0, \lvert[\psi_1 \mathbf{T} \psi_2]\rvert_i^d \Leftrightarrow \lvert[\psi_2]\rvert_i^d \wedge \left(\lvert[\psi_1]\rvert_i^d \vee ite\left(l_i, \lvert[\varphi]\rvert_E^{d-1}, \lvert[\varphi]\rvert_{i-1}^d\right)\right)$

The correctness claims are stated below but proofs have been omitted due to space considerations. Moreover, similarly to the LTL case, when moving from a PLTL encoding instance $\lvert[M,\psi]\rvert_k$ to the instance $\lvert[M,\psi]\rvert_{k+1}$ only the k-dependent constraints (and of course things learned from them by the SAT solver) need to be discarded.

Theorem 1. *Given a finite Kripke structure M and a PLTL formula ψ, M has a path π such that $\pi \models \psi$ iff there exists a $k \in \mathbb{N}$ such that $\lvert[M,\psi]\rvert_k$ is satisfiable. Specifically, if $\pi = s_0 s_1 s_2 \ldots$ is a (k,l)-loop such that $\pi \models \psi$ then $\lvert[M,\psi]\rvert_k$ is satisfiable.* [1]

4 Completeness for PLTL

The incremental encoding above can easily be extended to prove properties. The basic intuition is similar to the induction strengthening of [3] for invariants, restricted to the forward direction but extended for PLTL.

The procedure starts with bound $k = 0$. First we create a *completeness formula*, denoted by $\lvert[M,\psi,k]\rvert$, which is satisfied only for the initialised finite paths of length k which one might be able to extend to a bounded witness of formula ψ (of length k or longer). Furthermore, the completeness formula should be conjuncted with a *simple path* formula which is satisfied for exactly those paths which do not contain two "equivalent" states. If the conjunction of the completeness and simple path formulas is unsatisfiable the model checked formula $\neg\psi$ holds in the system and the procedure can be terminated. Otherwise the *witness formula* $\lvert[M,\psi]\rvert_k$ is created which is satisfied for bounded witnesses of length k to the formula ψ (see Theorem 1). If the witness formula is satisfiable, a witness is found, and the procedure can terminate and the model checked formula $\neg\psi$ does not hold. Otherwise, the procedure is repeated after incrementing k by one.

The termination of the procedure above is guaranteed if there are only finitely many equivalence classes of states considered by the simple path formula. The soundness

[1] A direct corollary of this is that minimal length (k,l)-loop counterexamples can be detected.

and completeness of the procedure is more involved and basically requires the witness, completeness, and simple path formulas to be compatible with each other.

In our case we will use as the completeness formula $|[M,\psi,k]|$ the encoding $|[M,\psi]|_k$ where all the k-dependent constraints have been discarded. Clearly, any witness formula for bounds $> k$ will contain all the constraints in $|[M,\psi,k]|$ and thus if $|[M,\psi,k]|$ is unsatisfiable all more constrained formulas are going to be unsatisfiable as well.

The definition of which states are equivalent w.r.t. the simple path formula is a bit more involved and the following definition is used: two states s_i and s_j are equivalent if either: (i) they both do not belong to the loop and agree on the system state $s_i = s_j$ and the formula state restricted to the first virtual unrolling $|[s_\psi]|_i^0 = |[s_\psi]|_j^0$, or (ii) they both belong to the loop and agree on the system state $s_i = s_j$, on the formula state on all unrollings $|[s_\psi]|_i = |[s_\psi]|_j$ and on the auxiliary formula state $\langle\langle s_\psi\rangle\rangle_i = \langle\langle s_\psi\rangle\rangle_j$. The simple path formula can thus be expressed by:

$$|[SimplePath]|_k := \bigwedge_{0\le i<j\le k} \Big(s_i \neq s_j \vee \mathrm{InLoop}_i \neq \mathrm{InLoop}_j \vee |[s_\psi]|_i^0 \neq |[s_\psi]|_j^0 \vee \Big(\mathrm{InLoop}_i \wedge \mathrm{InLoop}_j \wedge \Big(|[s_\psi]|_i \neq |[s_\psi]|_j \vee \langle\langle s_\psi\rangle\rangle_i \neq \langle\langle s_\psi\rangle\rangle_j\Big)\Big)\Big).$$

The intuition of case (i) is that if the states s_i and s_j agree on both the system state and the formula state restricted to the first virtual unrolling, then the witness is of non-minimal length and would have been already detected with bound $k-(j-i)$. Similar reasoning also applies to case (ii), where in addition to the system state, all unrollings have to agree on the formula state (please refer to Fig. 1) and the auxiliary formula state has to be identical in s_i and s_j in order not to erroneously remove any states from the witness which are needed to fulfil the temporal eventualities.

The proof why this notion of state equivalence is a sound formulation in the context of our procedure is slightly more involved but here we will give a sketch. Assume we have a bounded witness of length k which contains two equivalent states s_i and s_j. In the case of two equivalent states of type either (i) or (ii) we can show by (a slightly tedious but straightforward analysis of) the structure of the formulas $|[M,\psi]|_k$ and $|[M,\psi]|_{k-(j-i)}$ that there exists a bounded witness of length $k-(j-i)$ to ψ where all system states s_m such that $i < m \le j$ have been removed and all system states with indexes $n > j$ have their indexes decreased by $j-i$. Thus if a witness contains two equivalent states then also a shorter witness exists. Because repeating the procedure will terminate in a situation where no two equivalent states exist, the simple path formula does not remove any minimal length witnesses. We have the following result.

Theorem 2. *Given a model M and a PLTL formula ψ, $M \models \psi$ iff for some $k \in \mathbb{N}$ $|[M,\neg\psi,k]| \wedge |[SimplePath]|_k$ is unsatisfiable and $|[M,\neg\psi]|_i$ is unsatisfiable for all $0 \le i < k$.*

5 Optimising the Encoding

There are several straightforward ways of optimising the encoding. We present some of them below. For example, many subformulas in $cl(\psi)$ do not need a formula state variable bit as only subformulas to which other time points can refer to with temporal subformulas need to be included in $|[s_\psi]|$. In the following table we have included

several optimisations. The first class consists of binding formula variables in $|[s_\psi]|_E$ to those in s_E and $|[s_\psi]|_L$ in a k-invariant manner. There are several optimisations which exploit the monotonic nature of the unary LTL operators to infer things even before a loop on the system side has been closed. For example, if $\mathbf{G}\,\phi$ holds in a state s whose future is a superset of the future of state s' then clearly $\mathbf{G}\,\phi$ has to also hold in s'. (Some of these optimisations have to be enabled by InLoop_i because the required subset relations only hold for the black nodes of Fig. 1.) Also the over (under) approximation of the main encoding and auxiliary encoding for unary future formulas is exploited, and so on.

Base, $0 \leq d \leq \delta(\varphi)$	$k-$invariant, $0 \leq i \leq k, 0 \leq d \leq \delta(\varphi)$
$\|[p]\|_E^d \Leftrightarrow p_E$	$l_i \Rightarrow \mathit{LoopExists}$
$\|[\neg p]\|_E^d \Leftrightarrow \neg p_E$	$\|[\mathbf{G}\,\phi]\|_i^d \Rightarrow \|[\mathbf{G}\,\phi]\|_E^d$
$\|[\psi_1 \wedge \psi_2]\|_E^d \Leftrightarrow \|[\psi_1]\|_E^d \wedge \|[\psi_2]\|_E^d$	$\text{InLoop}_i \Rightarrow \left(\|[\mathbf{O}\,\phi]\|_i^d \Rightarrow \|[\mathbf{O}\,\phi]\|_E^d\right)$
$\|[\psi_1 \vee \psi_2]\|_E^d \Leftrightarrow \|[\psi_1]\|_E^d \vee \|[\psi_2]\|_E^d$	$\|[\mathbf{F}\,\phi]\|_E^d \Rightarrow \|[\mathbf{F}\,\phi]\|_i^d$
$\|[\mathbf{X}\,\phi]\|_E^d \Leftrightarrow \|[\phi]\|_L^{min(d+1,\delta(\phi))}$	$\text{InLoop}_i \Rightarrow \left(\|[\mathbf{H}\,\phi]\|_E^d \Rightarrow \|[\mathbf{H}\,\phi]\|_i^d\right)$
$\|[\psi_1\,\mathbf{U}\,\psi_2]\|_E^d \Leftrightarrow \|[\psi_2]\|_E^d \vee \left(\|[\psi_1]\|_E^d \wedge \|[\psi_1\,\mathbf{U}\,\psi_2]\|_L^{min(d+1,\delta(\varphi))}\right)$	
$\|[\psi_1\,\mathbf{R}\,\psi_2]\|_E^d \Leftrightarrow \|[\psi_2]\|_E^d \wedge \left(\|[\psi_1]\|_E^d \vee \|[\psi_1\,\mathbf{R}\,\psi_2]\|_L^{min(d+1,\delta(\varphi))}\right)$	
$\|[\mathbf{F}\,\phi]\|_E^d \Leftrightarrow \|[\phi]\|_E^d \vee \|[\mathbf{F}\,\phi]\|_L^{min(d+1,\delta(\varphi))}$	$\langle\langle \mathbf{F}\,\phi\rangle\rangle_i^{\delta(\phi)} \Rightarrow \langle\langle \mathbf{F}\,\phi\rangle\rangle_E^{\delta(\phi)}$
$\|[\mathbf{G}\,\phi]\|_E^d \Leftrightarrow \|[\phi]\|_E^d \wedge \|[\mathbf{G}\,\phi]\|_L^{min(d+1,\delta(\varphi))}$	$\langle\langle \mathbf{G}\,\phi\rangle\rangle_E^{\delta(\phi)} \Rightarrow \langle\langle \mathbf{G}\,\phi\rangle\rangle_i^{\delta(\phi)}$
$\|[\mathbf{G}\,\phi]\|_E^{\delta(\phi)} \Rightarrow \langle\langle \mathbf{G}\,\phi\rangle\rangle_E^{\delta(\phi)}$	$\|[\mathbf{G}\,\phi]\|_i^{\delta(\phi)} \Rightarrow \langle\langle \mathbf{G}\,\phi\rangle\rangle_i^{\delta(\phi)}$
$\langle\langle \mathbf{F}\,\phi\rangle\rangle_E^{\delta(\phi)} \Rightarrow \|[\mathbf{F}\,\phi]\|_E^{\delta(\phi)}$	$\langle\langle \mathbf{F}\,\phi\rangle\rangle_i^{\delta(\phi)} \Rightarrow \|[\mathbf{F}\,\phi]\|_i^{\delta(\phi)}$
$\|[\mathbf{G}\,\phi]\|_E^d \Rightarrow \|[\mathbf{G}\,\phi]\|_E^{d+1}$, when $d < \delta(\phi)$	$\text{InLoop}_i \Rightarrow \left(\|[\mathbf{G}\,\phi]\|_i^d \Rightarrow \|[\mathbf{G}\,\phi]\|_i^{d+1}\right)$, when $d < \delta(\phi)$
$\|[\mathbf{O}\,\phi]\|_E^d \Rightarrow \|[\mathbf{O}\,\phi]\|_E^{d+1}$, when $d < \delta(\phi)$	$\text{InLoop}_i \Rightarrow \left(\|[\mathbf{O}\,\phi]\|_i^d \Rightarrow \|[\mathbf{O}\,\phi]\|_i^{d+1}\right)$, when $d < \delta(\phi)$
$\|[\mathbf{F}\,\phi]\|_E^{d+1} \Rightarrow \|[\mathbf{F}\,\phi]\|_E^d$, when $d < \delta(\phi)$	$\text{InLoop}_i \Rightarrow \left(\|[\mathbf{F}\,\phi]\|_i^{d+1} \Rightarrow \|[\mathbf{F}\,\phi]\|_i^d\right)$, when $d < \delta(\phi)$
$\|[\mathbf{H}\,\phi]\|_E^{d+1} \Rightarrow \|[\mathbf{H}\,\phi]\|_E^d$, when $d < \delta(\phi)$	$\text{InLoop}_i \Rightarrow \left(\|[\mathbf{H}\,\phi]\|_i^{d+1} \Rightarrow \|[\mathbf{H}\,\phi]\|_i^d\right)$, when $d < \delta(\phi)$

6 Experiments

We have implemented our work in version 2.2.3 of the NuSMV model checker [19]. We have built an incremental SAT solver interface to NuSMV allowing one to add constraints to the solver either in a permanent or a temporary manner. The temporary constraints can be removed from the solver, automatically also removing all the constraints that the solver has learned based on those. The incremental SAT solvers currently supported by the interface are MiniSat [25] and ZChaff [26]. In the experiments we use the latest version 2004.11.15 of ZChaff as the SAT solver. The memory was limited to 900MiB and the cumulative time to one hour. No cone of influence reductions are used during benchmarking and our implementation does *not* contain any invariant (or any other property class) specific optimisations (left for further work). As the benchmark problems we use: (i) the systems involving PLTL specifications that we have used earlier in [18] (the VMCAI/* problems), (ii) IBM benchmarks from [27], and (iii) some systems in the standard NuSMV distribution involving LTL specifications which we could prove to be true.

Table 1 reports some of the results. The NuSMV 2.2.3 column refers to the non-incremental PLTL BMC model checker of NuSMV 2.2.3, the VMCAI column refers to

Table 1. Experimental results

	NuSMV 2.2.3			VMCAI			new inc. counter-ex.			new non-inc. counter-ex.			new inc. completeness			new non-inc. completeness		
problem	t/f	k	time	t/f	k	time	t/f	k	time	t/f	k	time	t/f	k	time	t/f	k	time
VMCAI2005/abp4	f	16	70	f	16	47	f	16	56	f	16	55	f	16	26	f	16	68
VMCAI2005/brp		28			152			1771			166			89			39	
VMCAI2005/dme4		23			49			56			51			57			39	
VMCAI2005/pci		15			18		f	18	2388		17			18			17	
VMCAI2005/srg5		12			242			736			210			54			44	
IBM/IBM_FV_2002_01	f	14	90	f	14	113	f	14	44	f	14	87	f	14	54	f	14	113
IBM/IBM_FV_2002_03	f	32	134	f	32	206	f	32	32	f	32	200	f	32	74	f	32	727
IBM/IBM_FV_2002_04	f	24	38	f	24	91	f	24	12	f	24	90	f	24	38	f	24	156
IBM/IBM_FV_2002_05	f	31	258	f	31	421	f	31	17	f	31	251	f	31	52	f	31	617
IBM/IBM_FV_2002_06	f	31	573	f	31	732	f	31	77	f	31	723	f	31	270	f	31	2032
IBM/IBM_FV_2002_09		232			84			787			81			81			76	
IBM/IBM_FV_2002_15	f	9	38	f	9	40	f	9	3	f	9	4	f	9	3	f	9	9
IBM/IBM_FV_2002_18		26			27		f	29	2362		26		f	29	1789		24	
IBM/IBM_FV_2002_19	f	29	3057		29		f	29	86		28		f	29	300		23	
IBM/IBM_FV_2002_20		27			27			35			26			35			24	
IBM/IBM_FV_2002_21	f	29	2276	f	29	3442	f	29	144	f	29	2741	f	29	239		24	
IBM/IBM_FV_2002_22		25			26			49			25			42			20	
IBM/IBM_FV_2002_23		25			25			31			24			31			21	
IBM/IBM_FV_2002_27	f	25	298	f	25	291	f	25	15	f	25	322	f	25	44	f	25	406
IBM/IBM_FV_2002_28	f	14	1046	f	14	973	f	14	245	f	14	1023	f	14	278	f	14	1160
IBM/IBM_FV_2002_29		14			15			17			14			20			14	
bmc/barrel5		28			28			67			26		t	11	63	t	11	314
ctl-ltl/counter_1		181			970			8025			3019		t	24	0	t	24	0
ctl-ltl/mutex		196			728			6855			2578		t	19	0	t	19	0
ctl-ltl/periodic		561			331			2063			781		t	201	593	t	201	2596
ctl-ltl/ring		159			264			713			165		t	66	3203		44	3598
ctl-ltl/short		182			870			2496			800		t	11	0	t	11	0

the implementation of our earlier non-incremental PLTL translation [18] ported on top of NuSMV 2.2.3, while "new inc. completeness" ("new non-inc. completeness", resp.) is an incremental (non-incremental, resp.) implementation of the new translation with completeness. The "new inc. counter-ex." ("new non-inc. counter-ex.") column refers to a "counter-example only" version of the new (non-incremental) implementation in which the completeness check is disabled (and thus the simple path constraint is not used). The "t/f" column shows whether the implementation was able to show that the property is either true or false, the k column shows the maximum k that was reached within the memory and time limits, and time is the cumulative time in seconds used to solve the problem.

The results show many interesting things. First of all, it can be seen that incrementality usually significantly improves the running times and thus also enables higher bounds to be reached faster. Furthermore, it never degraded the performance considerably. Second, compared to the counter-example only version, the completeness check is sometimes essentially free but sometimes it significantly slows down the search. Third, with the incremental SAT solver technology and the new translation it was possible to build a BMC procedure with completeness that usually performs much better than the standard NuSMV BMC procedure (without completeness) even when the specifications are simple invariants (the IBM problems). Last, our new incremental implementation is significantly better than the previously suggested compact encoding for PLTL (the VMCAI column). The implementation and experiments are available from: `http://www.tcs.hut.fi/~tjunttil/experiments/CAV05/`

7 Discussion and Conclusions

We have created an efficient incremental and complete BMC procedure for full PLTL detecting counterexamples at minimal bounds, an improvement over complete BMC procedures for LTL based on non-generalised Büchi automata [9,8]. The encoding most similar to ours is [5], which is unsurprising as it is also based on the joint work [18]. However, the main aim of [5] was to create a BDD based symbolic model checker, without incremental BMC in mind.

An interesting feature of our new PLTL encoding (unlike the earlier version [18]) is that it is sound also in the case we replace the function $\delta(\cdot)$ with a constant function that always returns 0. In this case the size of the encoding will be linear in $|\psi|$ (similar to [28], see [5]). In fact, we can limit the maximum virtual unrolling depth of formulas to any value between zero (minimal size encoding, potentially longer counterexamples) and $\delta(\psi)$ (minimal length counterexamples, larger encoding) and Theorem 2 still holds. Limiting the number of virtual unrollings to zero can be beneficial in order to sometimes prove completeness with a smaller bound.

In order to enhance the performance of the completeness part of the translation, more efficient ways of handling simple path constraints are needed. Thus an interesting future improvement in the SAT solver technology would be to enhance the constraint language of solvers so that they could handle mutual disequality of sets of Boolean vectors natively. An alternative approach would be to develop an incremental version of the more compact loop free predicate presented in [2]. In addition, we haven't tried adding the simple path constraints lazily on-demand as is done in [14]. Our current implementation of the new translation can be improved in many ways. Firstly, we suspect that the simple path formula can be refined, and several new k-independent formula invariants can be developed. Several interesting subsets of PLTL could possibly benefit from special case treatment, especially w.r.t. completeness. Also incremental and complete BMC for PLTL using backward traversal is left for further work.

Acknowledgements. The authors would like to thank Viktor Schuppan for interesting discussions on the topic.

References

1. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In: TACAS. Volume 1579 of LNCS., Springer (1999) 193–207
2. Kroening, D., Strichman, O.: Efficient computation of recurrence diameters. In: VMCAI. Volume 2575 of LNCS., Springer (2003) 298–309
3. Sheeran, M., Singh, S., Stålmarck, G.: Checking safety properties using induction and a SAT-solver. In: FMCAD. Volume 1954 of LNCS., Springer (2000) 108–125
4. Schuppan, V., Biere, A.: Efficient reduction of finite state model checking to reachability analysis. International Journal on Software Tools for Technology Transfer **5** (2004) 185–204
5. Schuppan, V., Biere, A.: Shortest counterexamples for symbolic model checking of LTL with past. In: TACAS. Volume 3440 of LNCS., Springer (2005) 493–509
6. de Moura, L.M., Rueß, H., Sorea, M.: Bounded model checking and induction: From refutation to verification. In: CAV. Volume 2725 of LNCS., Springer (2003) 14–26

7. Armoni, R., Fix, L., Fraer, R., Huddleston, S., Piterman, N., Vardi, M.Y.: SAT-based induction for temporal safety properties. In: Preliminary proceedings of BMC'04. (2004)
8. Clarke, E., Kroening, D., Oukanine, J., Strichman, O.: Completeness and complexity of bounded model checking. In: VMCAI. Volume 2937 of LNCS., Springer (2004) 85–96
9. Awedh, M., Somenzi, F.: Proving more properties with bounded model checking. In: CAV. Volume 3114 of LNCS. (2004) 96–108
10. McMillan, K.L.: Interpolation and SAT-based model checking. In: CAV. Volume 2725 of LNCS., Springer (2003) 1–13
11. Zhang, L., Madigan, C.F., Moskewicz, M.W., Malik, S.: Efficient conflict driven learning in boolean satisfiability solver. In: ICCAD, IEEE (2001) 279–285
12. Strichman, O.: Pruning techniques for the SAT-based bounded model checking problem. In: CHARME. Volume 2144 of LNCS., Springer (2001) 58–70
13. Whittemore, J., Kim, J., Sakallah, K.A.: Satire: A new incremental satisfiability engine. In: DAC, IEEE (2001) 542–545
14. Eén, N., Sörensson, N.: Temporal induction by incremental SAT solving. In: First International Workshop on Bounded Model Checking. Volume 89 of ENTCS., Elsevier (2003)
15. Jin, H., Somenzi, F.: An incremental algorithm to check satisfiability for bounded model checking. In: Preliminary proceedings of BMC'04. (2004)
16. Benedetti, M., Bernardini, S.: Incremental compilation-to-SAT procedures. In: Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT04). (2004)
17. Benedetti, M., Cimatti, A.: Bounded model checking for past LTL. In: Tools and Algorithms for Construction and Analysis of Systems. Volume 2619 of LNCS., Springer (2003) 18–33
18. Latvala, T., Biere, A., Heljanko, K., Junttila, T.: Simple is better: Efficient bounded model checking for past LTL. In: VMCAI. Volume 3385 of LNCS., Springer (2005) 380–395
19. 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: CAV. Volume 2404 of LNCS., Springer (2002) 359–364
20. Latvala, T., Biere, A., Heljanko, K., Junttila, T.: Simple bounded LTL model checking. In: FMCAD. Volume 3312 of LNCS., Springer (2004) 186–200
21. Kupferman, O., Vardi, M.: Model checking of safety properties. Formal Methods in System Design **19** (2001) 291–314
22. Tauriainen, H., Heljanko, K.: Testing LTL formula translation into Büchi automata. STTT - International Journal on Software Tools for Technology Transfer **4** (2002) 57–70
23. Clarke, E.M., Grumberg, O., Hamaguchi, K.: Another look at LTL model checking. Formal Methods in System Design. **10** (1997) 47–71
24. Laroussinie, F., Markey, N., Schnoebelen, P.: Temporal logic with forgettable past. In: LICS, IEEE Computer Society Press (2002) 383–392
25. Eén, N., Sörensson, N.: An extensible SAT-solver. In: SAT 2003. Volume 2919 of LNCS., Springer (2004) 502–518
26. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Design Automation Conference, IEEE (2001)
27. Zarpas, E.: Simple yet efficient improvements of SAT based bounded model checking. In: FMCAD. Volume 3312 of LNCS., Springer (2004) 174–185
28. Kesten, Y., Pnueli, A., Raviv, L.: Algorithmic verification of linear temporal properties. In: ICALP 1998. Volume 1443 of LNCS., Springer (1998) 1–16