Logical Methods in Computer Science
Vol. 2 (5:5) 2006, pp. 1–64
www.lmcs-online.org

Submitted Feb. 16, 2006
Published Nov. 15, 2006

LINEAR ENCODINGS OF BOUNDED LTL MODEL CHECKING

ARMIN BIERE [a], KEIJO HELJANKO [b], TOMMI JUNTTILA [c], TIMO LATVALA [d], AND VIKTOR SCHUPPAN [e]

[a] Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstrasse 69, A-4040 Linz, Austria
e-mail address: biere@jku.at

[b,c] Laboratory for Theoretical Computer Science, Helsinki University of Technology, P.O. Box 5400, FI-02015 TKK, Finland
e-mail address: {Keijo.Heljanko,Tommi.Junttila}@tkk.fi

[d] Department of Computer Science, University of Illinois at Urbana-Champaign, 201 Goodwin Ave., Urbana, IL 61801-2302, USA
e-mail address: tlatvala@uiuc.edu

[e] Computer Systems Institute, ETH Zentrum, CH-8092 Zürich, Switzerland
e-mail address: vschuppan@acm.org

Abstract. We consider the problem of bounded model checking (BMC) for linear temporal logic (LTL). We present several efficient encodings that have size linear in the bound. Furthermore, we show how the encodings can be extended to LTL with past operators (PLTL). The generalised encoding is still of linear size, but cannot detect minimal length counterexamples. By using the virtual unrolling technique minimal length counterexamples can be captured, however, the size of the encoding is quadratic in the specification. We also extend virtual unrolling to Büchi automata, enabling them to accept minimal length counterexamples.

Our BMC encodings can be made incremental in order to benefit from incremental SAT technology. With fairly small modifications the incremental encoding can be further enhanced with a termination check, allowing us to prove properties with BMC.

An analysis of the liveness-to-safety transformation reveals many similarities to the BMC encodings in this paper. We conduct experiments to determine the advantage of employing dedicated BMC encodings for PLTL over combining more general but potentially less efficient approaches with BMC: the liveness-to-safety transformation with invariant checking and Büchi automata with fair cycle detection.

Experiments clearly show that our new encodings improve performance of BMC considerably, particularly in the case of the incremental encoding, and that they are very competitive for finding bugs. Dedicated encodings seem to have an advantage over using more general methods with BMC. Using the liveness-to-safety translation with BDD-based invariant checking results in an efficient method to find shortest counterexamples that complements the BMC-based approach. For proving complex properties BDD-based methods still tend to perform better.

2000 ACM Subject Classification: F.3.1, B.6.3, D.2.4, F.4.1.
Key words and phrases: Bounded Model Checking – LTL – PLTL – Liveness to Safety – NuSMV.

Introduction

Bounded model checking [BCCZ99] was introduced as an alternative to binary decisions diagrams (BDDs) to implement symbolic model checking. This paper describes some of the key results of [Lat05, Sch06] on bounded model checking, and some extensions. The main results have been published in [LBHJ04, LBHJ05, HJL05, SB04, SB05].

The basic idea behind bounded model checking (BMC) is to restrict the general model checking problem to a *bounded* problem. Instead of asking whether the system M violates the property ψ, we ask whether the system M has any counterexample of length k to ψ. This bounded problem is *encoded* into SAT, the propositional satisfiability problem, in order to obtain the benefits of symbolic representations of states. In other words, a Boolean formula $|[M, \neg\psi, k]|$ is generated which is satisfiable iff M has a counterexample to ψ of length k. The satisfiability of this formula can then be checked with a SAT solver.

The key insight behind BMC for linear-time formalisms such as linear temporal logic (LTL) is that a witness for LTL given as an *infinite* execution path of the system can be captured by a *finite* path in two ways: either the finite path represents all its infinite extensions or the finite path loops and in fact captures the behaviour of an infinite path. Let $\pi = s_0 s_1 s_2 \ldots$ be an infinite path of a system. We say that π is a (k, l)-loop if $\pi = (s_0 s_1 \ldots s_{l-1})(s_l \ldots s_k)^\omega$ such that $0 < l \leq k$ and $s_{l-1} = s_k$.

In BMC the transition relation $T(s, s')$ of a system M is represented symbolically as a Boolean formula, where the states s, s' are modelled as bit vectors. To capture the finite paths of length k, we unroll the transition relation k times and obtain the following Boolean formula:

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

Here $I(s)$ is the initial state predicate and $T(s, s')$ a total transition relation predicate. Since only counterexamples to the given LTL formula ψ should be accepted, additional constraints must be generated to restrict the models of the Boolean formula. If we denote the formula constraints by $|[\neg\psi]|_k$, the Boolean formula $|[M, \neg\psi, k]| \Leftrightarrow |[M]|_k \wedge |[\neg\psi]|_k$ is satisfiable iff M has a counterexample of length k to ψ.

Compared with using BDDs to implement symbolic model checking, BMC has a few advantages. BMC can leverage the impressive gains that have been achieved in SAT solver technology in recent years [BS05]. The increase in efficiency of the solvers can directly be translated to more effective BMC. The use of SAT procedures as a practical implementation technique to search for bounded length executions of systems has also been used in the context of SAT-based artificial intelligence (AI) planning [KS92, KS96] and in sequential ATPG [KL93]. In practice, SAT solvers seem to be able to solve certain problems that are not feasible for BDDs.

An important advantage of BMC is that the counterexamples produced by most BMC encodings are minimal and that the counterexample is immediately available. Producing short counterexamples using BDDs is a fairly involved process [CGMZ95] and minimality is seldom guaranteed. In many cases producing the counterexample consumes more resources than answering the model checking query [CGMZ95]. However, recently a BDD model checking procedure [SB05] based on the BMC encoding of [LBHJ05] was presented that provably produces minimal counterexamples. The method appears to consume more memory than standard BDD model checkers, but can in some cases be faster.

Boolean formulas, or more specifically circuits, are a more compact encoding than BDDs for many Boolean functions: there are Boolean functions whose BDDs are exponential in the number of propositional variables [Bry86] that still have polynomial circuits. However, since BMC represents the length of the paths explicitly it is not always more space efficient than using BDDs [CKOS04]. For instance, for a simple binary counter system an exponential number of unrollings of the transition relation is required before the system loops and we can be sure that the whole behaviour of the system has been covered.

Although BMC has been very successful in practice [BCRZ99, CFF$^+$01, Str04], improving BMC remains a high priority. Increasing the efficiency of BMC can be done in several ways. Two important approaches are developing smarter encodings of the problem to SAT and utilising improvements in solver technology. Better encodings of the problem boil down to finding new representations of the formula $|[M, \neg\psi, k]|$, which are easier for the SAT solver. As a rule of thumb, good BMC encodings are compact but still propagate information efficiently, thus minimising the non-deterministic choices the solver has to make.

LTL with temporal operators that can reference the past is exponentially more succinct than LTL [LMS02]. In many cases the future fragment of LTL, which is the only fragment usually supported, is not expressive enough in practice. The main argument for adding support for past operators is motivated by practice: LTL with past operators (PLTL) allows more succinct and natural specifications. Especially compositional reasoning benefits from the added succinctness [LPZ85]. Efficient encodings for LTL with past operators is therefore one way to increase the usability, efficiency, and the scope of BMC. Utilising new solver technology such as incremental SAT solvers can result in huge benefits for BMC [WKS01, Str01]. When solving a sequence of similar SAT problems, as is the case in BMC, an incremental solver can retain much of the learned clauses obtained while solving earlier related instances. This can result in large time savings for solving the whole sequence of problems. The benefits of incremental SAT technology can be maximised by adapting BMC encodings to suit the incremental framework.

In this paper we will introduce several efficient BMC encodings for LTL that all have linear size encodings in the bound k. Efficient encodings can make a big difference when the specification is complex [LBHJ04]. We will present several encodings that take a slightly different view of the problem. In particular we highlight the relation of BMC encodings to the automata-theoretic approach to model checking [Kur94, VW86]. We also show how our encodings can be efficiently generalised to PLTL. The generalised encoding is still of linear size in the bound and in the size of the PLTL formula but does not detect minimal length counterexamples. By increasing the size of the encoding to quadratic in the size of the PLTL formula, minimal length counterexamples can be guaranteed. Our technique is based on *virtual unrolling* [BC03]. We also show how virtual unrolling enables symbolic Büchi automata to detect minimal length counterexamples.

Furthermore, with some modifications, our new more efficient encoding for PLTL can be adapted to utilise incremental SAT technology. We try to maximise the number of learnt clauses which can be kept when the solver moves from one problem instance to the next (i.e., when the bound k is increased). Experiments show that the increase in efficiency can be quite dramatic.

Model checking ω-regular properties depends on finding fair loops in the system. Using the *liveness-to-safety* model transformation [SB04], fair loop detection can be integrated in the system model. This effectively reduces the general unbounded model checking problem to reachability of bad states. We present the technique and discuss similarities with the BMC approaches introduced in this paper. In experiments we compare the performance of invariant checking, after translating liveness to safety, with dedicated BMC encodings for PLTL.

From its inception BMC has been predominantly seen as an efficient method for finding bugs. BDD-based methods have had the advantage of being complete and thus being able to prove that no counterexample exists. However, several methods have been developed in the recent years which can be used to achieve completeness with BMC (see for instance the recent survey [PBG05]). Our incremental encoding can also be extended with a termination check. The approach naturally integrates with our incremental approach and can prove properties for full PLTL.

We implemented the BMC encodings and the liveness-to-safety transformation on top of the NuSMV system [CCG$^+$02], version 2.2.3. Starting with version 2.4.0, the BMC encoding variant published in [HJL05] and discussed in more detail in this work has recently become a part of the standard distribution of NuSMV [NuS]. Based on the former, we have experimentally evaluated the encodings using a large set of models with complex specifications. Compared to the original encoding [BCCZ99] and its newer versions [CPRS02, BC03], our new linear encodings are clearly superior. We observed additional impressive performance gains for the incremental versions. Alternative linear sized encodings to do BMC based either on the liveness-to-safety transformation and invariant checking or on Büchi automata and fair loop detection did not prove quite as effective as the dedicated BMC encodings, although they were clearly more efficient than the original encoding and its relatives. With the termination check activated our linear BMC encoding did not perform quite as well as without it, but still better than old encodings. For proving properties BDD-based methods perform better. It is clear that the termination check must developed further in order for BMC to be competitive also for proving properties. Combining the liveness-to-safety transformation with BDD-based invariant checking results in an efficient BDD-based method to find shortest counterexamples. It significantly reduces the length of counterexamples in comparison to the standard BDD-based algorithm. It performs competitively with SAT-based methods for this purpose and complements them with respect to solved examples. Using virtual unrolling for Büchi automata with the standard BDD-based algorithm significantly increases running time and gives mixed results at best in terms of counterexample length.

In the next section we will introduce basic notation and recall fundamental definitions that will be used throughout the paper. In Sect. 2 the basics of bounded model checking are described and the results of the original BMC-paper [BCCZ99] are discussed. Section 3 presents our efficient BMC encoding for LTL published in [LBHJ04]. The section also considers alternative encodings of the BMC problem and contrasts the encodings to model checking based on symbolic Büchi automata. Section 4 presents the liveness-to-safety transformation and discusses its connection to the presented BMC encodings. To extend BMC to full PLTL, we use the technique of "virtual unrolling". We present our generalised BMC encoding that encompasses full PLTL in Sect. 5. We also show that virtual unrolling also can be applied to symbolic Büchi automata. Section 6 shows how our encodings can be adapted to the incremental setting [HJL05]. The adapted encodings are developed to maximise the information learnt between the SAT solver invocations. In Sect. 7 we discuss how BMC can be made complete. Specifically we show how our encodings can be extended with a termination check to achieve completeness. Section 8 experimentally compares the different encodings presented in the paper. We discuss conclusions and directions of future work in Sect. 9.

1. Preliminaries

1.1. **Linear Temporal Logic with Past.** Linear temporal logic with past (PLTL) is a commonly used specification logic. Although all PLTL properties are definable using only two basic temporal operators ($\mathbf{U}$ and $\mathbf{X}$), it has been argued that especially compositional reasoning benefits from the use of past operators [LPZ85]. Using only the basic operators results in a logic that is exponentially less succinct than PLTL [LMS02].

The *syntax* of PLTL is defined over a set of atomic propositions AP. Boolean operators we use are negation, disjunction and conjunction. The temporal operators we will use are "next time" ($\mathbf{X}$) and its two past-time counterparts, the "previous time" past temporal operators ($\mathbf{Y}$, $\mathbf{Z}$); the future temporal connectives "until" ($\mathbf{U}$) and "release" ($\mathbf{R}$) and their past-time counterparts "since" ($\mathbf{S}$) and "trigger" ($\mathbf{T}$). We will call the commonly used subset of PLTL that does not contain any past temporal operators linear temporal logic (LTL).

The *semantics* of a PLTL formula is defined along infinite paths $\pi = s_0 s_1 \ldots$[1] of states s_i where we assume a mapping L from each state to the set of atomic propositions true in that state. Let π^i denote the path π with a designated formula evaluation position i. The semantics can then be defined inductively as follows:

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

Commonly used *abbreviations* for PLTL formulas 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 $\mathbf{F}\,\psi_1 \equiv \top\,\mathbf{U}\,\psi_1$ ('finally'), $\mathbf{G}\,\psi_1 \equiv \neg\mathbf{F}\,\neg\psi_1$ ('globally'), $\mathbf{O}\,\psi_1 \equiv \top\,\mathbf{S}\,\psi_1$ ('once'), and $\mathbf{H}\,\psi_1 \equiv \bot\,\mathbf{T}\,\psi_1$ ('historically').

It is always possible to rewrite any formula to *positive normal form*, where all negations only appear in front of atomic propositions. This can be accomplished by using the dualities $\neg(\psi_1\,\mathbf{U}\,\psi_2) \equiv \neg\psi_1\,\mathbf{R}\,\neg\psi_2$, $\neg(\psi_1\,\mathbf{R}\,\psi_2) \equiv \neg\psi_1\,\mathbf{U}\,\neg\psi_2$, $\neg\mathbf{X}\,\psi_1 \equiv \mathbf{X}\,\neg\psi_1$, $\neg\mathbf{Y}\,\psi_1 \equiv \mathbf{Z}\,\neg\psi_1$, $\neg\mathbf{Z}\,\psi_1 \equiv \mathbf{Y}\,\neg\psi_1$, $\neg(\psi_1\,\mathbf{S}\,\psi_2) \equiv \neg\psi_1\,\mathbf{T}\,\neg\psi_2$, $\neg(\psi_1\,\mathbf{T}\,\psi_2) \equiv \neg\psi_1\,\mathbf{S}\,\neg\psi_2$, and DeMorgan's rules for propositional logic. In this paper we assume all formulas are in positive normal form unless otherwise explicitly stated.

[1]We use commas between elements of a tuple (such as a state consisting of the valuations of several state variables) and no separator between elements of a sequence (such as a path). While we generally follow the latter convention also for composition of sequences, we sometimes prefer to emphasise composition using $\circ$, e.g., if the entire sequence spans multiple lines.

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

Definition 1.1. *The past operator depth* [LMS02, BC03] *for a PLTL formula ψ is denoted by $\delta(\psi)$ and is inductively defined as:*

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

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:

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

1.2. **Kripke Structures.** The states of a path are members of the finite set of states S of a model (a Kripke structure) $M = (S, T, I, L)$ with a total transition relation T, a set of initial states I, and a mapping $L : S \mapsto 2^{AP}$ indicating the set of atomic propositions that are true in a state. L is extended to sequences of states (paths) in the natural way. A path is initialised iff its first state belongs to I. The set of initialised infinite paths is denoted Π. The language of a Kripke structure can then be defined as $\text{Lang}(M) = \{\alpha \mid \exists \pi \in \Pi \,.\, L(\pi) = \alpha\}$.

Sometimes we equip a Kripke structure with a number of *acceptance sets* (or fairness constraints) $F_0, \ldots, F_f$, where each F_m, $0 \le m \le f$ is a subset of S. $M = (S, T, I, L, F = \{F_0, \ldots, F_f\})$ is then called a fair Kripke structure. A path in M is fair iff it contains infinitely many occurrences of states from each acceptance set. Π and $\text{Lang}(M)$ are then restricted to fair paths.

We usually construct a Kripke structure *symbolically* over a set of variables V. In that case the set of states S is given by the set of valuations of V, possibly constrained by a set of state invariants. Similarly, I, T, and $F_0, \ldots, F_f$ are the largest subsets of S or $S \times S$ fulfilling certain constraints. The valuation of a variable v in a state s is denoted $v(s)$.

For a Kripke structure M we say that a *formula ψ_1 holds* in M if for every infinite initialised path π of M we have that $\pi \models \psi_1$. This is denoted $M \models \psi_1$. For a formula to hold in a fair Kripke structure it is required to hold only along all fair paths.

1.3. **Büchi Automata.** Büchi automata are frequently used as an operational model of the more descriptive PLTL formulae [VW86]. In this paper a Büchi automaton is simply a fair Kripke structure. However, if we speak of a "model" we refer to a Kripke structure that is to be verified (it is used as a language generator). When we say "Büchi automaton" we intend a Kripke structure to serve as a specification (it is used as a language acceptor).

A Büchi automaton B has a *run* π on an infinite sequence α over 2^{AP} iff π is an initialised path in B with $L(\pi) = \alpha$. The run is accepting iff it is fair. Hence, B has an accepting run on α iff $\alpha \in \text{Lang}(B)$.

Typically a Büchi automaton B specifies undesirable behaviour. The question whether a model M conforms to the specification then reduces to the question whether there is an initialised fair path in the product $M \times B$ [VW86]. As both M and B are finite state the

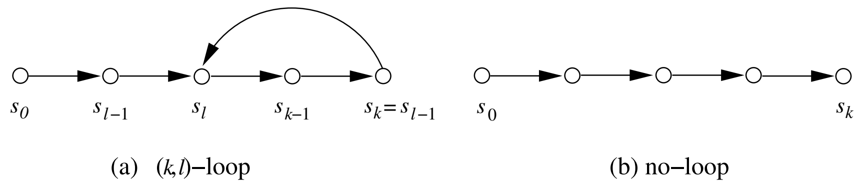


Figure 1: The two possible cases for a bounded path

search for such a path can be restricted to lasso-shaped paths, i.e., paths which are of form $\beta\gamma^\omega$, where β and γ are finite paths.

If a witness to the violation of the specification is to be extracted from an initialised fair path in the product of M and B for debugging, it is desirable that this path is short. A Büchi automaton is *tight* iff for every $\alpha = \beta\gamma^\omega \in \mathrm{Lang}(B)$ it has an accepting run $\rho = \sigma\tau^\omega$ such that α and ρ have the same shape: $|\beta| = |\sigma|$ and $|\gamma| = |\tau|$ [SB05, KV01]. Hence, the Büchi automaton can adapt as a chameleon to the shape of any potential lasso-shaped witness.

2. Bounded Model Checking

The main idea of bounded model checking [BCCZ99] is to search for *bounded witnesses* for a temporal property. A bounded witness is an initialised infinite path in 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. In (a) the (k, l)-loop case the finite path forms a *loop* and contains all infinite behaviour, or (b) the no-loop case when the finite path represents all its infinite extensions. More formally, an infinite path $\pi = s_0 s_1 s_2 \dots$ of states contains a (k, l)-loop, or just a k-loop, if $\pi = (s_0 s_1 \dots s_{l-1})(s_l \dots s_k)^\omega$ such that $0 < l \leq k$ and $s_{l-1} = s_k$. The two cases we consider are depicted in Fig. 1.

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 [FSW02, BC03, LBHJ04, LBHJ05, HJL05] the shape of the loop and accordingly the meaning of the bound k is slightly different from [BCCZ99]. In this paper a finite path of length k always has k transitions, and an infinite path with a loop contains the *looping state* twice, at position $l - 1$ and at position k.

Bounded model checking uses a *bounded semantics of PLTL* which safely under-approximates the normal semantics. It allows us to use a bounded prefix $\pi_k = s_0 s_1 \dots s_k$ of an initialised infinite path π to check the formula. The semantics is split into two cases. If the infinite path π is a k-loop a different semantics is used than in the case where it is not a k-loop. The definition below assumes the formula is in positive normal form.

Definition 2.1. *(See also* [BCCZ99, FSW02]*.) Given an initialised infinite path* π *and bound* $k \in \mathbb{N}$*,* $\pi \models_k \psi$ *iff (a)* π *is a* (k, l)*-loop for some* $0 < l \leq k$ *and* $\pi^0 \models \psi$*, or (b)*

$\pi^0 \models_{\mathrm{nl}} \psi$, *where:*

$$\begin{array}{lcl}
\pi^i \models_{\mathrm{nl}} p & \Leftrightarrow & \pi^i \models p. \\
\pi^i \models_{\mathrm{nl}} \neg p & \Leftrightarrow & \pi^i \models \neg p. \\
\pi^i \models_{\mathrm{nl}} \psi_1 \wedge \psi_2 & \Leftrightarrow & \pi^i \models_{\mathrm{nl}} \psi_1 \textit{ and } \pi^i \models_{\mathrm{nl}} \psi_2. \\
\pi^i \models_{\mathrm{nl}} \psi_1 \vee \psi_2 & \Leftrightarrow & \pi^i \models_{\mathrm{nl}} \psi_1 \textit{ or } \pi^i \models_{\mathrm{nl}} \psi_2. \\
\pi^i \models_{\mathrm{nl}} \mathbf{X}\,\psi_1 & \Leftrightarrow & i < k \textit{ and } \pi^{i+1} \models_{\mathrm{nl}} \psi_1. \\
\pi^i \models_{\mathrm{nl}} \psi_1 \,\mathbf{U}\, \psi_2 & \Leftrightarrow & \exists i \leq j \leq k \textit{ such that } \pi^j \models_{\mathrm{nl}} \psi_2 \textit{ and } \pi^n \models_{\mathrm{nl}} \psi_1 \textit{ for all } i \leq n < j. \\
\pi^i \models_{\mathrm{nl}} \psi_1 \,\mathbf{R}\, \psi_2 & \Leftrightarrow & \exists i \leq j \leq k \textit{ such that } \pi^j \models_{\mathrm{nl}} \psi_1 \textit{ and } \pi^n \models_{\mathrm{nl}} \psi_2 \textit{ for all } i \leq n \leq j. \\
\pi^i \models_{\mathrm{nl}} \mathbf{Y}\,\psi_1 & \Leftrightarrow & i > 0 \textit{ and } \pi^{i-1} \models_{\mathrm{nl}} \psi_1. \\
\pi^i \models_{\mathrm{nl}} \mathbf{Z}\,\psi_1 & \Leftrightarrow & i = 0 \textit{ or } \pi^{i-1} \models_{\mathrm{nl}} \psi_1. \\
\pi^i \models_{\mathrm{nl}} \psi_1 \,\mathbf{S}\, \psi_2 & \Leftrightarrow & \exists 0 \leq j \leq i \textit{ such that } \pi^j \models_{\mathrm{nl}} \psi_2 \textit{ and } \pi^n \models_{\mathrm{nl}} \psi_1 \textit{ for all } j < n \leq i. \\
\pi^i \models_{\mathrm{nl}} \psi_1 \,\mathbf{T}\, \psi_2 & \Leftrightarrow & \textit{for all } 0 \leq j \leq i : \pi^j \models_{\mathrm{nl}} \psi_2 \textit{ or } \pi^n \models_{\mathrm{nl}} \psi_1 \textit{ for some } j < n \leq i.
\end{array}$$

Because the language defined by the models of a PLTL formula belong to the ω-regular languages, we can restrict ourselves to searching for ultimately periodic witnesses in our models. Notice that for every ultimately periodic infinite path π, the bounded semantics becomes equivalent to the exact semantics when the k grows large enough to represent π as a (k, l)-loop. Thus for a model M and a PLTL property ψ there always exists some $k \in \mathbb{N}$ such that the bounded semantics becomes exact, i.e., $M \models \psi$ iff $M \models_k \psi$.

2.1. **Original BMC Encoding for LTL.**

The original encoding [BCCZ99] is defined recursively over the structure of the LTL formula ψ and the current position i. It is parameterised by the bound k, the start of the loop l and closely follows the bounded semantics of Def. 2.1. Therefore, for fixed i, k, and l, each subformula $\mathbf{F}\,\psi_1$ resp. $\mathbf{G}\,\psi_1$ of ψ requires constraints of size $O(k)$ using the encoding of ψ_1 at various positions. The binary operators $\mathbf{U}$ and $\mathbf{R}$ need constraints of size $O(k^2)$. Since the encoding of a subformula ψ_2 is only dependent on i, l, and k, and, in particular, multiple occurrences of the encoding of ψ_2 under the same set of parameters can be shared, the overall size can be bounded by $O(|\psi| \cdot k^4)$.

Parts of the constraints can be shared for different i. This reduces the overall complexity of the original encoding to $O(|\psi| \cdot k^3)$. It can be reduced even further to $O(|\psi| \cdot k)$, if only unary future temporal operators occur in ψ. As example consider the formula $\psi \equiv \mathbf{F}\,\mathbf{G}\,p$. As shown in [CPRS02, LBHJ04] a linear encoding of ψ can be obtained by optimising the original encoding using associativity and sharing. The encoding of, for instance, $\mathbf{G}\,(r \rightarrow (p\mathbf{U}\; q))$ is at least quadratic no matter what simplifications based on sharing and associativity are used [LBHJ04].

Even if more sophisticated circuit optimisations would allow to reduce the cubic original encoding to linear size, it is much more natural to start with a linear encoding in the first place. Finally, the original encoding translates *looping* and *non-looping* witnesses separately, while more advanced encodings, as discussed in this article, combine both.

It is tempting to use the recursive one step identities of the (unbounded) semantics of temporal operators $\psi_1\,\mathbf{U}\,\psi_2 \equiv \psi_2 \vee (\psi_1 \wedge \mathbf{X}\,(\psi_1\,\mathbf{U}\,\psi_2))$ and $\psi_1\,\mathbf{R}\,\psi_2 \equiv \psi_2 \wedge (\psi_1 \vee \mathbf{X}\,(\psi_1\,\mathbf{R}\,\psi_2))$ without any notion of fairness to encode LTL in a straightforward way, as for instance suggested in [BCC$^+$03]. In order to represent all witnesses for $\mathbf{G}\,p$ in a Kripke structure consisting of a single state with a self loop the following propositional formula can be used:

$$I(s_0) \;\wedge\; T(s_0, s_0) \;\wedge\; |[\mathbf{G}\,p]|_0 \;\wedge\; (|[\mathbf{G}\,p]|_0 \Leftrightarrow p_0 \wedge |[\mathbf{G}\,p]|_0).$$

Note the direct translation of the one step identity of the semantics of $\mathbf{G}$ on the right side. In this case, and in general for temporal operators with greatest fix-point semantics, this construction is sound, because the existence of an arbitrary fix-point implies the existence of the greatest fix-point and the recursively defined variable, denoted $|[\mathbf{G}\, p]|_0$, has only positive occurrences.

If applied in a naïve way, the same construction is incorrect for temporal operators with *least* fix-point semantics, as the following example shows. Again we are interested in all witnesses consisting of a single state with a self loop, but now for the LTL formula $\mathbf{F}\, p$. Using the same construction as above, simply following the one step LTL identities of $\mathbf{F}$ without any notion of fairness the following propositional encoding is obtained:

$$I(s_0) \;\wedge\; T(s_0, s_0) \;\wedge\; |[\mathbf{F}\, p]|_0 \;\wedge\; (|[\mathbf{F}\, p]|_0 \Leftrightarrow p_0 \vee |[\mathbf{F}\, p]|_0).$$

This formula can always be satisfied as long the transition relation has a self loop in an initial state by setting the boolean variable $|[\mathbf{F}\, p]|_0$ to $\top$. Therefore it will hold even if p is false in the initial state, and the encoding is therefore incorrect.

3. Improved Encodings of Bounded Model Checking for LTL

In this section several alternative bounded model checking encodings for LTL (i.e., PLTL without past temporal operators) are presented. How to extend the approaches to full PLTL containing also past temporal formulas is the topic of Sect. 5.

3.1. BMC for LTL with Fixpoint Evaluation.

One of the key factors affecting the efficiency of BMC is the size of the resulting SAT encoding. If the encoding produces unnecessarily large formulas the solver can quickly be overwhelmed, and we may not be able to proceed deep enough to find all violations to the specification in the design under model checking.

In [LBHJ04] we presented a BMC encoding to SAT for LTL which is linear in k that outperformed previous encodings. It consists of three types of constraints on the state variables representing the possible paths of length k: model constraints, loop constraints and LTL constraints. *Model constraints* $|[M]|_k$ encode legal initialised finite paths of the model M of length k:

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

where $I(s)$ is the initial state predicate and $T(s, s')$ is a total transition relation predicate. The *loop constraints* are used to non-deterministically select loops of paths encoded by the model constraints. We introduce $k+1$ fresh *loop selector variables* $l_0, \ldots, l_k$ which determine where the path loops. At most one loop selector variable is allowed to be true. If l_j is true then $s_{j-1} = s_k$, i.e., the bit vectors representing the state s_{j-1} and state s_k have bitwise identical values. In this case the LTL constraints treat the bounded path as a (k, j)-loop. If no loop selector variable is true then the LTL constraints treat the path as not having a loop (the *no-loop* case). Some counterexamples can be detected at lower bounds with the no-loop case (informative safety counterexamples of [KV01] to be exact). The loop constraints are encoded by conjuncting the constraints below. Only the loop selector variables l_i require fresh unconstrained variables; everything else can be implemented as constraints, i.e., variables that are constrained to be functionally dependent on the other variables of the formula. We denote the constraints by $|[\mathit{LoopConstraints}]|_k$:

Base	l_0	$\Leftrightarrow$	$\bot$
	InLoop_0	$\Leftrightarrow$	$\bot$
$1 \le i \le k$	l_i	$\Rightarrow$	$(s_{i-1} = s_k)$
	InLoop_i	$\Leftrightarrow$	$\text{InLoop}_{i-1} \lor l_i,$
	InLoop_{i-1}	$\Rightarrow$	$\neg l_i$
	LoopExists	$\Leftrightarrow$	InLoop_k

InLoop_i is true if the position i is in the loop part of the path. The loop selector variables indicate where the bounded path loops and select either a (k, j)-loop when l_j holds[2] or the no-loop case when no l_j holds. In the (k, j)-loop case the variable LoopExists will be true and in the no-loop case it will be false. Finally, the *LTL constraints* check if the bounded path defined by the model constraints and loop constraints is a model of the LTL formula. The LTL encoding utilises the fact that for (k, l)-loops the semantics of CTL and LTL coincide, see e.g., [KV01, TH02]. The intuitive reason is that if each state has exactly one successor (i.e., the path is lasso-shaped) then the semantics of the path quantifiers **A** and **E** of CTL agree. An LTL formula can therefore be evaluated in a lasso-shaped Kripke structure by a CTL model checker in linear time by prefixing each temporal operator by an **E** path quantifier [TH02], which results in a CTL formula.[3] 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 **U** and **R**. In CTL the until operator $\mathbf{E}(\psi_1 \,\mathbf{U}\, \psi_2)$ can be evaluated by computing the least fixed point $\mathbf{E}(\psi_1 \,\mathbf{U}\, \psi_2) = \mu Z.\psi_2 \lor (\psi_1 \land \mathbf{EX}\, Z)$ while the release operator $\mathbf{E}(\psi_1 \,\mathbf{R}\, \psi_2)$ can be evaluated by computing the greatest fixpoint $\mathbf{E}(\psi_1 \,\mathbf{R}\, \psi_2) = \nu Z.\psi_2 \land (\psi_1 \lor \mathbf{EX}\, Z)$, see e.g., [CGP99]. The encoding model checks lasso-shaped Kripke structures by computing the least and greatest fixpoints for **U** and **R**.

Given a formula φ we denote by $|[\varphi]|_i$ the Boolean formula for computing the truth value of φ at position i. To evaluate whether a formula φ holds in the initial state we must generate the formula for $|[\varphi]|_0$. The computation of the fixpoints for **U** and **R** is done in two parts. The *auxiliary translation* $\langle\langle\cdot\rangle\rangle$ computes an over-approximation for greatest fixpoints and an under-approximation for least fixpoints. The approximations are refined to exact values by $|[\cdot]|$. The auxiliary translation $\langle\langle\cdot\rangle\rangle$ under-approximates $\psi_1 \,\mathbf{U}\, \psi_2$-formulas by assuming that $\psi_1 \,\mathbf{U}\, \psi_2$ does *not hold* in the successor of the end state s_k. Conversely, $\psi_1 \,\mathbf{R}\, \psi_2$ is over-approximated by assuming that $\psi_1 \,\mathbf{R}\, \psi_2$ *holds* in the successor of the end state s_k. Both of these approximations are exact at the loop point j where l_j holds, because of the simple looping structure of the models.

The encoding can be understood as a recursively defined function where there is a case for each logical or temporal connective. For propositional LTL formulas the encoding is as follows:

$\lvert[\varphi]\rvert_i$	$0 \le i \le k$
$\lvert[p]\rvert_i$	$p \in L(s_i)$
$\lvert[\neg p]\rvert_i$	$p \notin L(s_i)$
$\lvert[\psi_1 \land \psi_2]\rvert_i$	$\lvert[\psi_1]\rvert_i \land \lvert[\psi_2]\rvert_i$
$\lvert[\psi_1 \lor \psi_2]\rvert_i$	$\lvert[\psi_1]\rvert_i \lor \lvert[\psi_2]\rvert_i$

[2]There is at most one index j where l_j holds, as otherwise $|[\mathit{LoopConstraints}]|_k$ would be unsatisfiable.

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

The encoding for temporal LTL formulas is as follows:

$\lvert[\varphi]\rvert_i$	$0 \le i < k$	$i = k$
$\lvert[\mathbf{X}\,\psi_1]\rvert_i$	$\lvert[\psi_1]\rvert_{i+1}$	$\bigvee_{j=1}^{k}\left(l_j \wedge \lvert[\psi_1]\rvert_j\right)$
$\lvert[\psi_1\,\mathbf{U}\,\psi_2]\rvert_i$	$\lvert[\psi_2]\rvert_i \vee \left(\lvert[\psi_1]\rvert_i \wedge \lvert[\psi_1\,\mathbf{U}\,\psi_2]\rvert_{i+1}\right)$	$\lvert[\psi_2]\rvert_i \vee \left(\lvert[\psi_1]\rvert_i \wedge \left(\bigvee_{j=1}^{k}\left(l_j \wedge \langle\langle\psi_1\,\mathbf{U}\,\psi_2\rangle\rangle_j\right)\right)\right)$
$\lvert[\psi_1\,\mathbf{R}\,\psi_2]\rvert_i$	$\lvert[\psi_2]\rvert_i \wedge \left(\lvert[\psi_1]\rvert_i \vee \lvert[\psi_1\,\mathbf{R}\,\psi_2]\rvert_{i+1}\right)$	$\lvert[\psi_2]\rvert_i \wedge \left(\lvert[\psi_1]\rvert_i \vee \left(\bigvee_{j=1}^{k}\left(l_j \wedge \langle\langle\psi_1\,\mathbf{R}\,\psi_2\rangle\rangle_j\right)\right)\right)$

The until (release) formulas at k refer to an auxiliary translation $\langle\langle\psi_1\,\mathbf{U}\,\psi_2\rangle\rangle_j$ ($\langle\langle\psi_1\,\mathbf{R}\,\psi_2\rangle\rangle_j$) at the loop point j where l_j holds. It computes an approximation of the semantics of until (release). If a loop exists this approximation is, in fact, exact for $\psi_1\,\mathbf{U}\,\psi_2$ ($\psi_1\,\mathbf{R}\,\psi_2$) at the loop index j corresponding to the time point $k+1$ in the (k,j)-loop path. In the no-loop case the effect of the encoding at index k is the same as if all subformulas at the index $k+1$ would be evaluated to $\bot$.

The auxiliary encoding for temporal LTL formulas is as follows:

$\lvert[\varphi]\rvert_i$	$1 \le i < k$	$i = k$
$\langle\langle\psi_1\,\mathbf{U}\,\psi_2\rangle\rangle_i$	$\lvert[\psi_2]\rvert_i \vee \left(\lvert[\psi_1]\rvert_i \wedge \langle\langle\psi_1\,\mathbf{U}\,\psi_2\rangle\rangle_{i+1}\right)$	$\lvert[\psi_2]\rvert_k$
$\langle\langle\psi_1\,\mathbf{R}\,\psi_2\rangle\rangle_i$	$\lvert[\psi_2]\rvert_i \wedge \left(\lvert[\psi_1]\rvert_i \vee \langle\langle\psi_1\,\mathbf{R}\,\psi_2\rangle\rangle_{i+1}\right)$	$\lvert[\psi_2]\rvert_k$

Because the semantics of until is a least fixpoint, the encoding of $\langle\langle\psi_1\,\mathbf{U}\,\psi_2\rangle\rangle_k$ is just the simplified form of the expression $\lvert[\psi_2]\rvert_k \vee (\lvert[\psi_1]\rvert_k \wedge \bot)$, where $\langle\langle\psi_1\,\mathbf{U}\,\psi_2\rangle\rangle_{k+1}$ has been replaced by $\bot$. Similarly, because the semantics of release is a greatest fixpoint, we have $\lvert[\psi_2]\rvert_k \wedge (\lvert[\psi_1]\rvert_k \vee \top)$ for $\langle\langle\psi_1\,\mathbf{R}\,\psi_2\rangle\rangle_k$.

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

$$\lvert[M,\psi,k]\rvert \Leftrightarrow \lvert[M]\rvert_k \wedge \lvert[\mathit{LoopConstraints}]\rvert_k \wedge \lvert[\psi]\rvert_0.$$

We have the following result:

Theorem 3.1. *Given a Kripke structure M and an LTL formula ψ, M has an initialised path π such that $\pi \models \psi$ iff there exists a $k \in \mathbb{N}$ such that the fixpoint evaluation encoding $\lvert[M,\psi,k]\rvert$ is satisfiable. In particular, if $\pi \models_k \psi$ then the fixpoint evaluation encoding $\lvert[M,\psi,k]\rvert$ is satisfiable.* [4]

Proof. We first prove a stronger result than the second part of the theorem: M has an initialised path π such that $\pi \models_k \psi$ iff the fixpoint evaluation encoding $\lvert[M,\psi,k]\rvert$ is satisfiable. The first part of the theorem follows from this together with the fact that when the bound k is increased large enough $M \models \psi$ iff $M \models_k \psi$.

It is easy to see that the model constraints $\lvert[M]\rvert_k$ encode all legal initialised finite paths π' of the model M of length k. Now consider the loop constraints $\lvert[\mathit{LoopConstraints}]\rvert_k$. As in the definition of the semantics of $\models_k$, we have two cases: (a) π' is a (k,j)-loop for some j inducing an infinite path π: In this case by setting l_j to true and all other l_i to false the truth values of all other variables in $\lvert[\mathit{LoopConstraints}]\rvert_k$ are uniquely determined, in particular LoopExists will be true. Because $s_{j-1} = s_k$ we can satisfy all constraints in $\lvert[\mathit{LoopConstraints}]\rvert_k$. It is also easy to check that if more than one l_i variable is true, these constraints are unsatisfiable. The second case is: (b) We are in the no-loop case: π' is

[4]As immediate corollary minimal length (k,l)-loop counterexamples for LTL can be detected. The encoding also detects minimal length informative safety counterexamples for LTL.

a finite prefix of some initialised infinite path π through the system. The only remaining option is that all l_i variables are false. Now the truth values of all other variables in $|[\mathit{LoopConstraints}]|_k$ are again uniquely determined, in particular LoopExists will be false. Thus all constraints in $|[\mathit{LoopConstraints}]|_k$ are satisfied.

Consider a satisfying truth assignment of $|[M]|_k \wedge |[\mathit{LoopConstraints}]|_k$ inducing an initialised infinite path π. We want to check that it can be extended to a model of the full encoding $|[M, \psi, k]|$ iff $\pi \models_k \psi$. Because the encoding $|[\psi]|_0$ is just a Boolean circuit, the truth value of each of its nodes are uniquely determined by the other variables of the encoding, and we will evaluate these values in what follows.

We will prove by induction on the structure of the LTL formula ψ that for all $\varphi \in cl(\psi), 0 \leq i \leq k$: $\pi^i \models_k \varphi$ iff $|[\varphi]|_i$ is true. In particular, $\pi \models_k \psi$ iff $|[\psi]|_0$ is true.

The cases where φ is an atomic proposition or its negation are trivial in both cases (a) and (b). The same holds for all propositional cases, where the claim holds for the subformulas by the induction hypothesis.

What remains to be proven are the cases where φ is a temporal operator. Because the encoding of $|[\varphi]|_i$ for all indices $0 \leq i < k$ just uses the one-step identities for LTL formulas, the claim holds for all of them provided that for the last index k it holds that $\pi^k \models_k \varphi$ iff $|[\varphi]|_k$ is true.

First consider the easier no-loop case (b): By the above we have that none of the l_i variables is true. In this case we can simplify the encoding by substituting $\perp$ for every l_i variable and simplifying the result. After doing this it is easy to check that the encoding of $|[\varphi]|_k$ behaves as if $\pi^{k+1} \not\models \psi_1$ for all subformulas $\psi_1 \in cl(\psi)$. It is now easy to check that at index k this matches the definition of the no-loop semantics $\models_{\mathrm{nl}}$ for all temporal operators, and thus the semantics matches $\models_{\mathrm{nl}}$ also for all indexes $0 \leq i < k$.

Now consider the (k, j)-loop case (a): Recall that an LTL formula can be evaluated in a lasso-shaped Kripke structure by a CTL model checker by prefixing each temporal operator by an $\mathbf{E}$ path quantifier [TH02], which results in a CTL state formula. Thus in a (k, j)-loop we need to only consider the truth value of LTL formulas at indexes $0 \leq i \leq k$, as the truth values for any larger index, for example $i = k + 1$, can be reduced to evaluating the LTL formula at the corresponding state of the model, in this case the loop state $i = j$.

By the above we know that l_j is the only loop selector variable which is true, and that the subformulas are correctly evaluated for all indices by the induction hypothesis. If $\varphi = \mathbf{X}\, \psi_1$, the encoding of $|[\varphi]|_k$ picks the truth value of ψ_1 from $|[\psi_1]|_j$ corresponding to the index $k + 1$ in the (k, j)-loop (recall that l_j is the only loop selector variable which holds), and we are done.

In the case $\varphi = \psi_1 \,\mathbf{U}\, \psi_2$ we have to do a case analysis. First consider case (i): $\pi^i \models \psi_2$ for some $j \leq i \leq k$, and therefore $\pi^i \models \psi_1 \mathbf{U} \psi_2$. Without loss of generality, pick the smallest such i. Now clearly at index i the auxiliary translation $\langle\langle \psi_1 \,\mathbf{U}\, \psi_2 \rangle\rangle_i$ is true. Because the auxiliary translation $\langle\langle \psi_1 \,\mathbf{U}\, \psi_2 \rangle\rangle_n$ for all indices $j \leq n \leq i$ is just the one-step identity of until, $\langle\langle \psi_1 \,\mathbf{U}\, \psi_2 \rangle\rangle_n$ is true iff $\pi^n \models \psi_1 \,\mathbf{U}\, \psi_2$. In particular, at the loop point j we have: $\langle\langle \psi_1 \,\mathbf{U}\, \psi_2 \rangle\rangle_j$ is true iff $\pi^j \models \psi_1 \,\mathbf{U}\, \psi_2$. Now consider case (ii): $\pi^i \not\models \psi_2$ for all $j \leq i \leq k$. In this case clearly $\pi^j \not\models \psi_1 \,\mathbf{U}\, \psi_2$. It is now easy to check from the definition of the auxiliary encoding that $\langle\langle \psi_1 \,\mathbf{U}\, \psi_2 \rangle\rangle_n$ is false for all indices $j \leq n \leq k$. In both cases we have $\langle\langle \psi_1 \,\mathbf{U}\, \psi_2 \rangle\rangle_j$ is true iff $\pi^j \models \psi_1 \,\mathbf{U}\, \psi_2$, and because the encoding of $|[\psi_1 \,\mathbf{U}\, \psi_2]|_k$ uses $\langle\langle \psi_1 \,\mathbf{U}\, \psi_2 \rangle\rangle_j$ to obtain the value of $\pi^{k+1} \models \psi_1 \,\mathbf{U}\, \psi_2$, we have $\pi^k \models \psi_1 \,\mathbf{U}\, \psi_2$ iff $|[\psi_1 \,\mathbf{U}\, \psi_2]|_k$ is true.

In the case $\varphi = \psi_1 \mathbf{R} \psi_2$ we have to do a very similar (dual) case analysis. First consider case (i): $\pi^i \not\models \psi_2$ for some $j \leq i \leq k$, and therefore $\pi \not\models \psi_1 \mathbf{R} \psi_2$. Without loss of generality, pick the smallest such i. Now clearly at index i the auxiliary translation $\langle\langle \psi_1 \mathbf{R}\, \psi_2 \rangle\rangle_i$ is false. Because the auxiliary translation $\langle\langle \psi_1 \mathbf{R}\, \psi_2 \rangle\rangle_n$ for all indices $j \leq n \leq i$ is just the one-step identity for release, $\langle\langle \psi_1 \mathbf{R}\, \psi_2 \rangle\rangle_n$ is true iff $\pi^n \models \psi_1 \mathbf{R} \psi_2$. In particular, at the loop point j we have: $\langle\langle \psi_1 \mathbf{R}\, \psi_2 \rangle\rangle_j$ is true iff $\pi^j \models \psi_1 \mathbf{R}\, \psi_2$. Now consider case (ii): $\pi^i \models \psi_2$ for all $j \leq i \leq k$. In this case clearly $\pi^j \models \psi_1 \mathbf{R}\, \psi_2$. It is now easy to check from the definition of the auxiliary encoding that $\langle\langle \psi_1 \mathbf{R}\, \psi_2 \rangle\rangle_n$ is true for all indices $j \leq n \leq k$. In both cases we have $\langle\langle \psi_1 \mathbf{R}\, \psi_2 \rangle\rangle_j$ is true iff $\pi^j \models \psi_1 \mathbf{R}\, \psi_2$, and because the encoding $|[\psi_1 \mathbf{R}\, \psi_2]|_k$ uses $\langle\langle \psi_1 \mathbf{R}\, \psi_2 \rangle\rangle_j$ to obtain the value of $\pi^{k+1} \models \psi_1 \mathbf{R}\, \psi_2$, we have $\pi^k \models \psi_1 \mathbf{R}\, \psi_2$ iff $|[\psi_1 \mathbf{R}\, \psi_2]|_k$ is true.

Thus by forcing the top level formula $|[\psi]|_0$ to be true we get that M has an initialised path π such that $\pi \models_k \psi$ iff $|[M, \psi, k]|$ is satisfiable, from which the full theorem follows. $\square$

The encoding has a few desirable properties of which the most important one is that when the encoding is seen as a Boolean circuit where the loop selector variables and the atomic propositions of the model are input variables, the size of the generated formula is $O(|I|+k\cdot|T|+k\cdot|\psi|)$. The encoding also has a *unique model property* in the following sense: if the (k, l)-loop is given (i.e., the computation π together with the l_j variables are fixed), the Boolean circuit representing the LTL encoding has no free variables. Consequently, there is no nondeterminism in evaluating the circuit that evaluates the LTL formula, and if the encoding is satisfiable the given (k, l)-loop defines a unique model of the Boolean circuit.

If the loop selector variables, atomic propositions and their negations are seen as inputs to the circuit, the circuit for the LTL encoding $|[\psi]|_0$ is monotonic. This can be exploited to devise an improved encoding of the Boolean circuit to conjunctive normal form (CNF) formulas. A similar optimisation has been presented in the encoding of [FSW02].

The original encoding [BCCZ99] and its improved version [CPRS02] both result in formulas that are at least quadratic w.r.t. k. Frisch et al. [FSW02] have presented an alternative encoding based on normal forms for LTL. This so-called fixpoint encoding is more efficient than previous attempts, but it produces formulas that are non-linear w.r.t. k [LBHJ04]. An improved version of the fixpoint encoding, which includes a generalisation to PLTL, is linear w.r.t. k but does not provide minimal length counterexamples for PLTL formulas [CRS04]. Note, that [CRS04] contains an ambiguity in its description that may lead an implementation choice that results in wrong handling of formulas containing past temporal operators. For details see Sect. 5. The normal form used in the fixpoint encoding [FSW02] is similar to tableau methods for constructing a symbolic Büchi automaton $\mathcal{A}_\psi$ representing an LTL formula ψ. It is also possible to do BMC by applying the automata theoretic approach and symbolically encode a product system $M \times \mathcal{A}_{\neg\psi}$ [dMRS02, CKOS05]. BMC is performed by searching for fair loops in the product system. This approach produces a linear size encoding if the search for fair loops is encoded with an encoding such as [CPRS02, LBHJ04] that can encode $\mathbf{G}\,\mathbf{F}\,p$ in linear size. Since this method only searches for looping counterexamples, it must sometimes go deeper than other methods also accepting no-loop safety counterexamples.

See [HN03] for earlier work on linear size bounded model checking encodings for LTL employing logic programs with the stable model semantics instead of using SAT. This work does not directly give us a linear size SAT encoding because the best known automatic translation from logic programs with the stable model semantics into SAT are super-linear (roughly $O(n \log_2 n)$), see [Jan04].

3.2. **BMC for LTL with Eventualities.** An alternative approach to encoding semantics of LTL formulas is to use an *eventuality encoding*, which in the loop case requires that for each until formula $\psi_1 \mathbf{U} \psi_2$ the right hand side formula ψ_2 holds at some point in the loop (and dually for release). The main idea for until formulas is to first evaluate whether the *eventuality formula* $\mathbf{F} \psi_2$ holds in the last state k, and use this knowledge to evaluate the value of the main encoding. If we know that $\mathbf{F} \psi_2$ does not hold at k, then surely $\psi_1 \mathbf{U} \psi_2$ cannot hold at k either. In all other cases the one-step LTL identities actually evaluate the bounded LTL semantics correctly. A dual construction is applied for release formulas. This idea above enables one to replace the auxiliary encodings of until and release with simpler ones, but at the same time the encoding becomes a set of Boolean equations with cyclic dependencies between variables instead of a Boolean circuit where no such cyclic dependencies exist. Having cyclic dependencies allows for a slightly smaller encoding but in our opinion makes the approach a bit harder to understand.

The eventuality encoding is quite similar to the fixpoint evaluation encoding, so only the LTL part of the new encoding will presented. The encoding is no longer defined as a recursive function over the LTL formula but as Boolean constraints over the so called *formula variables* $|[\varphi]|_i$, which are fresh unconstrained propositional variables. There is a variable $|[\varphi]|_i$ for every subformula $\varphi \in cl(\psi)$ and for all $0 \leq i \leq k+1$. The interpretation of $|[\varphi]|_i$ is still that it is true iff φ holds at position i in the model. For propositional LTL formulas the encoding is as follows:

φ	$0 \leq i \leq k+1$
p_i	$\lvert[p]\rvert_i \Leftrightarrow p \in L(s_i)$
$\neg p_i$	$\lvert[\neg p]\rvert_i \Leftrightarrow p \notin L(s_i)$
$\psi_1 \wedge \psi_2$	$\lvert[\psi_1 \wedge \psi_2]\rvert_i \Leftrightarrow \lvert[\psi_1]\rvert_i \wedge \lvert[\psi_2]\rvert_i$
$\psi_1 \vee \psi_2$	$\lvert[\psi_1 \vee \psi_2]\rvert_i \Leftrightarrow \lvert[\psi_1]\rvert_i \vee \lvert[\psi_2]\rvert_i$

The encoding for the temporal subformulas is changed to the following (the only thing that changes is the encoding at index k):

φ	$0 \leq i \leq k$
$\mathbf{X} \psi_1$	$\lvert[\mathbf{X} \psi_1]\rvert_i \Leftrightarrow \lvert[\psi_1]\rvert_{i+1}$
$\psi_1 \mathbf{U} \psi_2$	$\lvert[\psi_1 \mathbf{U} \psi_2]\rvert_i \Leftrightarrow \lvert[\psi_2]\rvert_i \vee \big(\lvert[\psi_1]\rvert_i \wedge \lvert[\psi_1 \mathbf{U} \psi_2]\rvert_{i+1}\big)$
$\psi_1 \mathbf{R} \psi_2$	$\lvert[\psi_1 \mathbf{R} \psi_2]\rvert_i \Leftrightarrow \lvert[\psi_2]\rvert_i \wedge \big(\lvert[\psi_1]\rvert_i \vee \lvert[\psi_1 \mathbf{R} \psi_2]\rvert_{i+1}\big)$

To compensate for the change at index k we will for each subformula $\varphi \in cl(\psi)$ add the following constraints $|[\mathit{LastStateFormula}]|_k$:

Base	$\neg \mathit{LoopExists} \Rightarrow \big(\lvert[\varphi]\rvert_{k+1} \Leftrightarrow \bot\big)$
$1 \leq i \leq k$	$l_i \Rightarrow \big(\lvert[\varphi]\rvert_{k+1} \Leftrightarrow \lvert[\varphi]\rvert_i\big)$

The constraints state that if there is no loop, all formula variables at index $k+1$ should evaluate to $\bot$. This is the same as in the fixpoint evaluation encoding and results in the no-loop case in the the bounded LTL semantics. For the case when a loop exists, the added constraints force all formula variables at index $k+1$ to get their values from the loop point j, where l_j holds. Note that this can create a cyclic dependency between variables in the Boolean equation system as $|[\varphi]|_j$ can depend indirectly through the states $j+1, j+2, \ldots, k-1, k$ on the value of $|[\varphi]|_{k+1}$ which is constrained to be equal to $|[\varphi]|_j$ itself.

The reader might be puzzled why the $|[\mathit{LoopConstraints}]|_k$ contains constraints of the form: $l_i \Rightarrow (s_{i-1} = s_k)$ while $|[\mathit{LastStateFormula}]|_k$ contains analogous constraint with off-by-one indices: $l_i \Rightarrow \big(|[\varphi]|_{k+1} \Leftrightarrow |[\varphi]|_i\big)$. This is an optimisation which allows detection of no-loop safety counterexamples one unrolling of the system transition relation earlier. This optimisation (used also in [FSW02, BC03, LBHJ04, LBHJ05, HJL05]) could easily be undone changing the loop shape of (k,l)-loops to match that of [BCRZ99] and requiring: $|[M]|_k \Leftrightarrow I(s_0) \wedge \bigwedge_{i=1}^{\boxed{k+1}} T(s_{i-1}, s_i)$ and $l_i \Rightarrow (s_{\boxed{i}} = s_{\boxed{k+1}})$, thus bringing the system and formula indices back to synch.

There is still one final piece missing because the encoding as it stands so far has models which do not agree with the semantics of LTL. The constraints introduced so far allow the case where $|[\psi_1 \, \mathbf{U} \, \psi_2]|$ is true at all indices of the loop even if $|[\psi_2]|$ is true at no index of the loop (this can happen when $|[\psi_1]|$ is true at all indices of the loop). This clearly violates the semantics of until and needs to be taken care of. In such a case the SAT solver has found a solution for the evaluation of the cyclic dependencies between until variables mentioned above, but this solution is not the required least fixpoint solution (see also discussion on this topic in Sect. 2.1). For release formulas the situation is less severe. It can be the case that $|[\psi_2]|$ holds at all indices of the loop but $|[\psi_1 \, \mathbf{R} \, \psi_2]|$ holds in no index. This is not fatal in the sense that in this case the semantics of release have been under-approximated (as is also done by the no-loop safety case). In addition, the encoding has a satisfying truth assignment where the semantics of release is, in fact, evaluated correctly.

To disallow assignments as described above, where the eventualities of until and release are not fulfilled, we use a set of auxiliary constraints for until and release subformulas. The constraints perform a similar function to the auxiliary encoding of until and release in the fixpoint encoding. In the table below $\langle\langle\varphi\rangle\rangle_i$ are new auxiliary formula variables used by the constraints.

	φ	
Base	$\psi_1 \, \mathbf{U} \, \psi_2$	$\mathit{LoopExists} \Rightarrow (|[\psi_1 \, \mathbf{U} \, \psi_2]|_k \Rightarrow \langle\langle \mathbf{F} \, \psi_2\rangle\rangle_k)$
	$\psi_1 \, \mathbf{R} \, \psi_2$	$\mathit{LoopExists} \Rightarrow (|[\psi_1 \, \mathbf{R} \, \psi_2]|_k \Leftarrow \langle\langle \mathbf{G} \, \psi_2\rangle\rangle_k)$
	$\psi_1 \, \mathbf{U} \, \psi_2$	$\langle\langle \mathbf{F} \, \psi_2\rangle\rangle_0 \Leftrightarrow \bot$
	$\psi_1 \, \mathbf{R} \, \psi_2$	$\langle\langle \mathbf{G} \, \psi_2\rangle\rangle_0 \Leftrightarrow \top$
$1 \leq i \leq k$	$\psi_1 \, \mathbf{U} \, \psi_2$	$\langle\langle \mathbf{F} \, \psi_2\rangle\rangle_i \Leftrightarrow \langle\langle \mathbf{F} \, \psi_2\rangle\rangle_{i-1} \vee (\mathrm{InLoop}_i \wedge |[\psi_2]|_i)$
	$\psi_1 \, \mathbf{R} \, \psi_2$	$\langle\langle \mathbf{G} \, \psi_2\rangle\rangle_i \Leftrightarrow \langle\langle \mathbf{G} \, \psi_2\rangle\rangle_{i-1} \wedge (\neg\mathrm{InLoop}_i \vee |[\psi_2]|_i)$

We use the names $\langle\langle \mathbf{F} \, \psi_2\rangle\rangle_i$ and $\langle\langle \mathbf{G} \, \psi_2\rangle\rangle_i$ for the auxiliary variables because it describes the function of the constraints well. The constraint $\mathit{LoopExists} \Rightarrow (|[\psi_1 \, \mathbf{U} \, \psi_2]|_k \Rightarrow \langle\langle \mathbf{F} \, \psi_2\rangle\rangle_k)$ intuitively ensures that in the loop case if $\psi_1 \, \mathbf{U} \, \psi_2$ holds at k, then there is some index in the loop where ψ_2 holds. This is quite similar to, but not technically identical to, the use of Büchi acceptance sets for ensuring the correct semantics for until, as will be shown later. The encoding for release is only required to get the exact LTL semantics for release formulas. The constraint $\mathit{LoopExists} \Rightarrow (|[\psi_1 \,\mathbf{R}\, \psi_2]|_k \Leftarrow \langle\langle \mathbf{G}\, \psi_2 \rangle\rangle_k)$ could be safely dropped if we allow the satisfying models of the encoding to safely under-approximate the bounded semantics instead of exactly capturing it.[5] Dropping the auxiliary constraints could also be done for the fixpoint encoding of Sect. 3.1 by adding $|[\mathit{LastStateFormula}]|_k$ constraints for release subformulas. The intuitive idea of the auxiliary encoding is that if a loop exists, $\langle\langle \mathbf{F}\, \psi_2 \rangle\rangle_k$ $(\langle\langle \mathbf{G}\, \psi_2 \rangle\rangle_k)$ is the evaluation of the formula $\mathbf{F}\, \psi_2$ $(\mathbf{G}\, \psi_2)$ at π^k.

We denote the constraints on the formula variables and the auxiliary variables above with $|[\mathit{EventuallyLTL}]|_k$. The conjunction of these four sets of constraints and requiring that the formula holds in the initial state forms the full *eventuality encoding* of the bounded model checking problem into SAT:

$$|[M, \psi, k]| \Leftrightarrow |[M]|_k \wedge |[\mathit{LoopConstraints}]|_k \wedge |[\mathit{LastStateFormula}]|_k \wedge |[\mathit{EventuallyLTL}]|_k \wedge |[\varphi]|_0.$$

Theorem 3.2. *Given a Kripke structure M and an LTL formula ψ, M has an initialised path π such that $\pi \models \psi$ iff there exists a $k \in \mathbb{N}$ such that the eventuality encoding $|[M, \psi, k]|$ is satisfiable. In particular, if $\pi \models_k \psi$ then the eventuality encoding $|[M, \psi, k]|$ is satisfiable.*

Proof. We proceed similarly to the proof of Thm. 3.1, and will only give the changes to the proof needed to reflect changes in the encoding. The only changes are the encoding of temporal subformulas at index k, the use of proxy variables $|[\varphi]|_{k+1}$, the new auxiliary encoding $|[\mathit{EventuallyLTL}]|_k$, and the new $|[\mathit{LastStateFormula}]|_k$ constraints.

We will now prove by induction on the structure of the LTL formula ψ that the eventuality encoding is satisfiable and for all $\varphi \in cl(\psi), 0 \leq i \leq k$: $\pi^i \models_k \varphi$ iff in the unique satisfying truth assignment of the eventuality encoding $|[\varphi]|_i$ is true.

First consider the no-loop case (b): In this case, because LoopExists is false, it is easy to see that the new $|[\mathit{LastStateFormula}]|_k$ constraints will force the proxy variables $|[\varphi]|_{k+1}$ to $\perp$, and the encoding becomes exactly the same as in the fixpoint encoding case and thus has a unique satisfying truth assignment. Also the new auxiliary encoding constraints will lead to a unique satisfying truth assignment as as *LoopExists* is false.

Now consider the (k, j)-loop case (a): Recall from the proof of Thm. 3.1 that in a (k, j)-loop we need to only consider the truth value of LTL formulas at indexes $0 \leq i \leq k$, as the truth values for any larger index, for example $i = k+1$, can be reduced to evaluating the LTL formula at the corresponding state of the model, in this case the loop state $i = j$.

By earlier analysis we know that l_j is the only loop selector variable which is true, and that the the encoding for all subformulas are correctly evaluated for all indices by the induction hypothesis. In this case the $|[\mathit{LastStateFormula}]|_k$ constraints are satisfiable and uniquely set the value of the proxy variable $|[\varphi]|_{k+1}$ for every subformula $\varphi \in cl(\psi)$ to be equivalent to the value of the subformula at the loop point j, namely $|[\varphi]|_j$. Therefore we do not need to consider the index $i = k+1$ in our proofs provided that the index $i = j$ is evaluated correctly.

If $\varphi = \mathbf{X}\, \psi_1$, the encoding differs from the fixpoint evaluation encoding only at the index k. The encoding of $|[\varphi]|_k$ together with $|[\mathit{LastStateFormula}]|_k$ picks the truth value of ψ_1 from $|[\psi_1]|_j$ corresponding to the index $k+1$ in the (k, j)-loop (recall that l_j is the only

[5]This is similar to the fact that most LTL to Büchi automata translations do not employ acceptance sets for release.

loop selector variable which holds), all constraints are satisfiable in a unique way, and we are done.

For until and release formulas our proof strategy is the following. We first prove that if there is a satisfying truth assignment then it must for the end point $n = k$ have the property that $\pi^n \models_k \varphi$ iff $|[\varphi]|_n$ is true. After this we observe that for both until and release formulas the following holds: the truth assignment that matches the bounded semantics of LTL for all indexes is satisfiable. We will simultaneously prove uniqueness by noting that any truth assignment which matches the bounded semantics of LTL at the index $n = k$ will force all other variables of the truth assignment to a unique value that matches the semantics of LTL for all indexes, and also satisfies the auxiliary encoding in a unique way. This is the case because when the truth value of $|[\varphi]|_k$ is fixed, for all other $0 \leq n < k$ the formula $|[\varphi]|_n$ will obtain a truth value in a functional way based on the value of $|[\varphi]|_{n+1}$ matching the bounded semantics of LTL. Also the encoding of $|[\varphi]|_k$ is satisfiable, as it matches the bounded semantics of LTL, and $|[\varphi]|_{k+1}$ matches the value of $|[\varphi]|_j$ at the loop point j. The auxiliary encoding also has a unique satisfying truth assignment as the auxiliary encoding contains no cyclic dependencies.

In the case $\varphi = \psi_1 \mathbf{U} \psi_2$ we have to do a case analysis. First consider case (i): $\pi^i \models \psi_2$ for some $j \leq i \leq k$. Without loss of generality, pick the smallest such i (intuition: the cyclic dependency over until subformulas is broken at index i). Clearly at index i the auxiliary translation $\langle\langle \mathbf{F}\, \psi_2 \rangle\rangle_i$ is true. Because of this, the auxiliary translation $\langle\langle \mathbf{F}\, \psi_2 \rangle\rangle_k$ is true, and the corresponding auxiliary translation Base constraint is satisfied. Therefore, $\pi^i \models \psi_1 \mathbf{U} \psi_2$ and $|[\psi_1 \mathbf{U}\, \psi_2]|_i$ is also true. Because the encoding follows the one-step identity of until we also get for all $j \leq n \leq i$: $|[\psi_1 \mathbf{U}\, \psi_2]|_n$ iff $\pi^n \models \psi_1 \mathbf{U}\, \psi_2$, and from encoding at k together with $|[\mathit{LastStateFormula}]|_k$ that $|[\psi_1 \mathbf{U}\, \psi_2]|_k$ iff $\pi^k \models \psi_1 \mathbf{U}\, \psi_2$. Thus we have established that for all indexes $j \leq n \leq i$ and $n = k$ the encoding matches the semantics of LTL, and because of this and our proof strategy, the encoding has a unique satisfying truth assignment that matches the semantics of LTL for all $0 \leq n \leq k$. Now consider case (ii): $\pi^i \not\models \psi_2$ for all $j \leq i \leq k$. In this case the auxiliary translation $\langle\langle \mathbf{F}\, \psi_2 \rangle\rangle_k$ is false. We have that $\pi^k \not\models \psi_1 \mathbf{U} \psi_2$, and if we set $|[\psi_1 \mathbf{U}\, \psi_2]|_k$ to be true then the auxiliary translation Base constraint is not satisfied. Therefore we must set $|[\psi_1 \mathbf{U}\, \psi_2]|_k$ to false (intuition: the cyclic dependency over until subformulas is broken at index k) which matches the bounded LTL semantics of until at $n = k$ and also satisfies the auxiliary constraints. By our proof strategy all other indices $0 \leq i < k$ have a unique satisfying truth assignment obtained from the one-step identity of until based on $|[\psi_1 \mathbf{U}\, \psi_2]|_k$ matching the semantics of LTL, and also leading to the satisfaction of the constraints $|[\mathit{LastStateFormula}]|_k$. In both cases (i) and (ii) we have for all $0 \leq n \leq k$ that $\pi^n \models \psi_1 \mathbf{U}\, \psi_2$ iff in the unique satisfying truth assignment $|[\psi_1 \mathbf{U}\, \psi_2]|_n$ is true.

In the case $\varphi = \psi_1 \mathbf{R}\, \psi_2$ we have to do a very similar (dual) case analysis. First consider case (i): $\pi^i \not\models \psi_2$ for some $j \leq i \leq k$. Without loss of generality, pick the smallest such i. Now clearly at index i the auxiliary translation $\langle\langle \mathbf{G}\, \psi_2 \rangle\rangle_i$ is false. Because of this, the auxiliary translation $\langle\langle \mathbf{G}\, \psi_2 \rangle\rangle_k$ is false, and the corresponding auxiliary translation Base constraint is satisfied. Hence $\pi^i \not\models \psi_1 \mathbf{R}\, \psi_2$, and $|[\psi_1 \mathbf{R}\, \psi_2]|_i$ is also false. Because the encoding follows the bounded LTL semantics of release we also get for all $j \leq n \leq i$: $|[\psi_1 \mathbf{R}\, \psi_2]|_n$ iff $\pi^n \models \psi_1 \mathbf{R} \psi_2$, and from the encoding at k together with $|[\mathit{LastStateFormula}]|_k$ that $|[\psi_1 \mathbf{R}\, \psi_2]|_k$ iff $\pi^k \models \psi_1 \mathbf{R}\, \psi_2$, and we can proceed similarly to the until case. Now consider case (ii): $\pi^i \models \psi_2$ for all $j \leq i \leq k$. In this case the auxiliary translation $\langle\langle \mathbf{G}\, \psi_2 \rangle\rangle_k$

is true. We have that $\pi^k \models \psi_1 \, \mathbf{R} \, \psi_2$, and if we set $|[\psi_1 \, \mathbf{R} \, \psi_2]|_k$ to be false then the auxiliary translation Base constraint is not satisfied. Therefore we must set $|[\psi_1 \, \mathbf{R} \, \psi_2]|_k$ to true, satisfying the auxiliary constraints, and we can proceed similarly to the until case. In both cases (i) and (ii) we have for all $0 \leq n \leq k$ that $\pi^n \models \psi_1 \, \mathbf{R} \, \psi_2$ iff in the unique satisfying truth assignment $|[\psi_1 \, \mathbf{R} \, \psi_2]|_n$ is true.

Now proceed similarly to the proof of Thm. 3.1 to complete the proof. $\square$

The eventuality encoding has the unique model property in a very similar sense as the fixpoint evaluation encoding: after fixing the loop point and the valuation of atomic propositions at all time points there can only be a unique valuation of the variables of the encoding that satisfies all the constraints. However, this cannot be explained by the fact that the encoding is a Boolean circuit (it is not, as it contains cyclic dependencies between variables); it follows from Thm. 3.2 that all the formula variables of the encoding are uniquely determined by the bounded semantics of LTL.

There is some similarity with the separated normal form (SNF) encodings of [FSW02, CRS04] for BMC and the eventuality encoding presented here in the sense that the SNF encodings first split a (strong) until to a conjunction of a weak until and an eventuality formula, and use this to devise the BMC encoding for all time steps. We instead use the eventuality formula to evaluate the correct value for the (strong) until formula at the last state k only.

3.3. **BMC for LTL with Büchi Automata.**

The knowledgeable reader has certainly noticed the close correspondence between our eventuality encoding and the use of Büchi automata symbolically implementing the tableau construction [LP85] for LTL model checking, such as [BCM$^+$92, CGH97, KPR98, Sch01]. Wolper, Vardi and Sistla were the first to show how to compile LTL directly into Büchi automata [WVS83, VW94]. Gerth et al. [GPVW95] suggested an algorithm that produces smaller automata. It has subsequently been improved by a number of authors [Cou99, DGV99, SB00, EH00, GO01, GO03, ST03]. These improved versions are used today mainly in explicit-state (e.g., SPIN [Hol03]) but also in some symbolic model checkers (e.g., VIS [VIS96]).

In symbolic treatment of LTL, a compact symbolic representation of the automaton has mostly been preferred to a small number of states. Büchi automata for that purpose are usually symbolic implementations of the tableau construction in [LP85]. A first application of the tableau in symbolic context is given by Burch et al. [BCM$^+$92]; for proofs and an experimental evaluation see [CGH97]. A self-contained presentation of symbolic model checking of LTL with past can be found in [KPR98]. Schneider exploits the temporal hierarchy for further optimisations [Sch01].

Another consideration is the depth at which the verification procedure stops. A tight Büchi automaton is required to accept shortest witnesses [SB05, Sch06, KV01]. Büchi automata constructed with an algorithm based on [GPVW95] typically fail this criterion; methods based on [LP85] such as [BCM$^+$92, CGH97] fulfil it for the future fragment only [SB05, Sch06]. In Sect. 5.2 we apply the idea of virtual unrolling (see Sect. 5.1) to Büchi automata to obtain a Büchi automaton with a small symbolic representation that is tight for PLTL. On the other hand, Awedh and Somenzi [AS06] showed experimentally that bounded model checking with constructions based on [LP85] often lead to larger termination depths than with those based on [GPVW95] if the property holds [AS06].

Following the automata-theoretic approach [VW86], Büchi automata are employed for bounded model checking of infinite state systems in de Moura et al. [dMRS02], instead of using a dedicated encoding. Clarke et al. employ Büchi automata to obtain completeness bounds for arbitrary ω-regular properties [CKOS05]. Awedh and Somenzi present a complete bounded model checking procedure based on such an encoding [AS04, AS06]. In Sect. 8 we report on experiments comparing the performance of a dedicated encoding with the automata-theoretic approach in bounded model checking. Below we first slightly modify the eventuality encoding to obtain an encoding along the lines of [BCM$^+$92, CGH97]. This approach is then generalised to show how to encode emptiness checking of the product of a model with an arbitrary Büchi automaton.

3.3.1. *Modifying the Eventuality Encoding.* Only minor changes are needed to obtain a Büchi automata-based LTL encoding from the eventuality encoding. For every until and release subformula we introduce new auxiliary variables $\langle\langle Acc(\cdot)\rangle\rangle_i$. The auxiliary eventuality encoding needs to be replaced by the auxiliary Büchi encoding defined as follows:

	φ	$1 \le i \le k$
Base	$\psi_1\,\mathbf{U}\,\psi_2$	$\mathit{LoopExists} \Rightarrow \langle\langle Acc(\psi_1\,\mathbf{U}\,\psi_2)\rangle\rangle_k, \langle\langle Acc(\psi_1\,\mathbf{U}\,\psi_2)\rangle\rangle_0 \Leftrightarrow \bot$
	$\psi_1\,\mathbf{R}\,\psi_2$	$\mathit{LoopExists} \Rightarrow \langle\langle Acc(\psi_1\,\mathbf{R}\,\psi_2)\rangle\rangle_k, \langle\langle Acc(\psi_1\,\mathbf{R}\,\psi_2)\rangle\rangle_0 \Leftrightarrow \bot$
	$\psi_1\,\mathbf{U}\,\psi_2$	$\langle\langle Acc(\psi_1\,\mathbf{U}\,\psi_2)\rangle\rangle_i \Leftrightarrow \langle\langle Acc(\psi_1\,\mathbf{U}\,\psi_2)\rangle\rangle_{i-1} \vee \big(\mathrm{InLoop}_i \wedge (\lvert[\psi_2]\rvert_i \vee \neg\lvert[\psi_1\,\mathbf{U}\,\psi_2]\rvert_i)\big)$
	$\psi_1\,\mathbf{R}\,\psi_2$	$\langle\langle Acc(\psi_1\,\mathbf{R}\,\psi_2)\rangle\rangle_i \Leftrightarrow \langle\langle Acc(\psi_1\,\mathbf{R}\,\psi_2)\rangle\rangle_{i-1} \vee \big(\mathrm{InLoop}_i \wedge (\neg\lvert[\psi_2]\rvert_i \vee \lvert[\psi_1\,\mathbf{R}\,\psi_2]\rvert_i)\big)$

We denote the full set of modified LTL constraints with $\lvert[B\ddot{u}chiLTL]\rvert_k$ The conjunction of the five sets of constraints forms the full *Büchi encoding* of the bounded model checking problem into SAT:

$$\lvert[M,\psi,k]\rvert \Leftrightarrow \lvert[M]\rvert_k \wedge \lvert[LoopConstraints]\rvert_k \wedge \lvert[LastStateFormula]\rvert_k \wedge \lvert[B\ddot{u}chiLTL]\rvert_k \wedge \lvert[\varphi]\rvert_0.$$

By comparing to the general Büchi encoding on (k,l)-loops below, it is easy to see that our Büchi encoding is nothing else than an emptiness checker for a symbolic Büchi automaton following [BCM$^+$92, CGH97]. The initial state predicate $\lvert[\psi]\rvert_0$ requires the top level formula to hold at the initial state, the symbolic transition relation is given by the encoding rules for propositional and temporal operators, and acceptance sets are defined by the auxiliary translation as follows. For each until formula $\psi_1\,\mathbf{U}\,\psi_2$ we add an acceptance set $F_{\psi_1\mathbf{U}\psi_2}$ into which the states satisfying $\lvert[\psi_2]\rvert_i \vee \neg\lvert[\psi_1\,\mathbf{U}\,\psi_2]\rvert_i$ belong to, and for each release formula $\psi_1\,\mathbf{R}\,\psi_2$ we add an acceptance set $F_{\psi_1\mathbf{R}\psi_2}$ into which the states satisfying $\neg\lvert[\psi_2]\rvert_i \vee \lvert[\psi_1\,\mathbf{R}\,\psi_2]\rvert_i$ belong to.

Theorem 3.3. *Given a Kripke structure M and an LTL formula ψ, M has an initialised path π such that $\pi \models \psi$ iff there exists a $k \in \mathbb{N}$ such that the Büchi encoding $\lvert[M,\psi,k]\rvert$ is satisfiable. In particular, if $\pi \models_k \psi$ then the Büchi encoding $\lvert[M,\psi,k]\rvert$ is satisfiable.*

Proof. We prove that the auxiliary eventuality encoding is satisfiable iff the auxiliary Büchi encoding is. The claim then follows from Thm. 3.2.

We only show that LoopExists $\Rightarrow (\lvert[\psi_1\,\mathbf{U}\,\psi_2]\rvert_k \Rightarrow \langle\langle\mathbf{F}\,\psi_2\rangle\rangle_k)$ is satisfiable if and only if LoopExists $\Rightarrow \langle\langle Acc(\psi_1\,\mathbf{U}\,\psi_2)\rangle\rangle_k$ is satisfiable. The proof for $\mathbf{R}$ is similar.

The case $\neg$LoopExists is clear. Hence, assume LoopExists is true. We start with the direction from left to right. First, let $\langle\langle\mathbf{F}\,\psi_2\rangle\rangle_k$ be true. There must be $0 \le i \le k$

such that $\text{InLoop}_i \wedge |[\psi_2]|_i$ is true. This immediately gives that $\langle\langle \mathit{Acc}(\psi_1 \mathbf{U}\, \psi_2)\rangle\rangle_i$ and also $\langle\langle \mathit{Acc}(\psi_1 \mathbf{U}\, \psi_2)\rangle\rangle_k$ is true. Now, let $\neg|[\psi_1 \mathbf{U}\, \psi_2]|_k$ be true. With $\text{LoopExists} \Leftrightarrow \text{InLoop}_k$ we have $\langle\langle \mathit{Acc}(\psi_1 \mathbf{U}\, \psi_2)\rangle\rangle_k$.

For the other direction assume $\langle\langle \mathit{Acc}(\psi_1 \mathbf{U}\, \psi_2)\rangle\rangle_k$ is true. Hence, there exists $0 \leq i \leq k$ such that $\text{InLoop}_i \wedge (|[\psi_2]|_i \vee \neg|[\psi_1 \mathbf{U}\, \psi_2]|_i)$ is true. If $\text{InLoop}_{i'} \wedge |[\psi_2]|_{i'}$ for some i' we have $\langle\langle \mathbf{F}\, \psi_2\rangle\rangle_{i'}$ and, therefore, $\langle\langle \mathbf{F}\, \psi_2\rangle\rangle_k$. Otherwise, there is $0 \leq i' \leq k$ such that $\text{InLoop}_{i'} \wedge \neg|[\psi_1 \mathbf{U}\, \psi_2]|_{i'}$. By definition of the encoding for $\mathbf{U}$ we obtain $\text{InLoop}_j \Rightarrow \neg|[\psi_1 \mathbf{U}\, \psi_2]|_j$ for all $0 \leq j < i'$ and, via $\neg|[\psi_1 \mathbf{U}\, \psi_2]|_{k+1}$, $\neg|[\psi_1 \mathbf{U}\, \psi_2]|_k$. □

Notice that the Büchi encoding above also generates no-loop safety counterexamples. It also has the unique model property unlike in most other Büchi automata constructions which do not employ acceptance sets for release formulas. The unique model property allows us to read the exact bounded semantics for all LTL subformulas and all time indexes considered directly from the truth assignment given by the SAT engine. As in the other encodings, if the unique model property is not of interest to us, we can do what most other Büchi automata constructions do and drop the constraint $\mathit{LoopExists} \Rightarrow \langle\langle \mathit{Acc}(\psi_1 \mathbf{R}\, \psi_2)\rangle\rangle_k$ and the auxiliary translation for release to obtain a slightly smaller encoding.

3.3.2. *General Approach.* The above approach can easily be generalised to obtain an encoding to check existence of an initialised fair path in a fair Kripke structure. If $M = (S, T, I, L, F = \{F_0, \ldots, F_f\})$ is a fair Kripke structure, it is sufficient to extend the loop constraints with the following Büchi loop constraints:

	$0 \leq m \leq f$
Base	$\text{LoopExists} \Leftrightarrow \top$ $\text{LoopExists} \Rightarrow \langle\langle \mathit{Acc}_m\rangle\rangle_k$ $\langle\langle \mathit{Acc}_m\rangle\rangle_0 \Leftrightarrow \bot$
$1 \leq i \leq k$	$\langle\langle \mathit{Acc}_m\rangle\rangle_i \Leftrightarrow \langle\langle \mathit{Acc}_m\rangle\rangle_{i-1} \vee (\text{InLoop}_i \wedge s_i \in F_m)$

For each acceptance set F_m an additional constraint $\langle\langle \mathit{Acc}_m\rangle\rangle$ is introduced to check satisfaction of F_m in the loop. Hence, the following conjunction forms the *general Büchi encoding* of the bounded model checking problem into SAT:

$$|[M,k]| \Leftrightarrow |[M]|_k \wedge |[\mathit{LoopConstraints}]|_k \wedge |[\mathit{B\ddot{u}chiLoopConstraints}]|_k.$$

Theorem 3.4. *Given a Kripke structure M, M has a fair (k,l)-loop π for some $0 < l \leq k$ iff there exists a $k \in \mathbb{N}$ such that the general Büchi encoding $|[M,k]|$ is satisfiable.*

Proof. First we show that if $|[M,k]|$ is satisfiable then M has a fair loop. Assume $|[M,k]|$ is satisfiable for some k. Fix an arbitrary satisfying assignment. As LoopExists is true, there is a unique $0 < l \leq k$ such that l_l is true. It follows that $s_{l-1} = s_k$. Hence, $s_0 \ldots s_{l-1}(s_l \ldots s_k)^\omega$ is an initialised (k,l)-loop in M. Further, the loop is fair, as for each acceptance set F_m, $0 \leq m \leq f$ there is some $0 \leq j \leq k$ such that InLoop_j is true and $s_j \in F_m$.

In the second case let M have a fair loop. We need to prove that $|[M,k]|$ is satisfiable for some $k \in \mathbb{N}$. Assume $\pi = s_0 \ldots s_{l-1}(s_l \ldots s_k)^\omega$ with $s_{l-1} = s_k$ is a fair loop in M. For each $0 \leq m \leq f$ there is $l \leq i_m \leq k$ such that $s_{i_m} \in F_m$. With $s_0 \ldots s_k$, $\mathit{LoopExists} \Leftrightarrow \top$,

$l_i \Leftrightarrow i = l$, $InLoop_i \Leftrightarrow i \geq l$, and $\langle\langle Acc_m \rangle\rangle_i \Leftrightarrow i \geq i_m$ for all $0 \leq m \leq f$ we obtain a satisfying assignment for $|[M, k]|$. □

Note that the above variant only considers looping witnesses, as is often done in the automata-theoretic approach to model checking of LTL. Finite (no-loop) witnesses to safety properties help, as they do not need to close a loop, focus attention on that part of an infinite path that is most relevant for violation of a safety property. In addition, minimising a no-loop witness to a safety property minimises the distance between an initial state and the actual point of violation. In contrast, minimising a looping witness just minimises the total length of the looping path, regardless of where the property fails. To also obtain finite witnesses, M can be given as the product of the model, a Büchi automaton accepting looping witnesses, and an automaton on finite words accepting finite witnesses to the property.

4. Liveness Checking as Safety Checking

While verification of safety properties can be handled using (simple) reachability checking, verification of liveness or, more generally, ω-regular properties requires detection of fair loops. Traditionally, loop detection is an integral part of the search algorithm [LP85, VW86, EL87]. Bounded model checking has to pull the algorithm out of the search procedure, i.e., the SAT solver, by making it part of the propositional formula submitted to the SAT solver [BCCZ99]. Building on that, we below present an approach that fully integrates loop detection into the model.

The *liveness-to-safety transformation* takes a fair Kripke structure M and transforms it into another Kripke structure $M^{\mathbf{S}}$ such that there is an initialised fair path in M iff a certain set of states is reachable in $M^{\mathbf{S}}$. This method makes techniques available for arbitrary ω-regular properties that have only been applicable to safety properties so far. It has already proven to be useful as a method to find shortest looping counterexamples with a BDD-based model checker [SB05], and to extend SAT-based interpolation [McM03] and large-scale directed model checking [EJ06] to ω-regular properties. On selected examples, an exponential speedup can be observed compared to traditional BDD-based model checking [SB04]. Still, because of its impact on the size of the state space (see below), this approach may in many cases not be able to replace dedicated methods for verifying ω-regular properties. In Sect. 8 we evaluate experimentally how invariant checking of a transformed model performs in comparison to dedicated encodings for PLTL properties. The liveness-to-safety transformation was originally proposed in [BAS02] and has been further developed in [SB04, SB06, Sch06]. Bouajjani et al. independently applied the same technique in the context of regular model checking [BHV04]. The presentation below contains no new results, but deviates from previous work to emphasise similarities with the bounded model checking approach at the core of this paper.

4.1. **Transformation.** A typical modelling language of a model checker allows only access to the current and next states of a path. It is not directly possible to ask whether the current state has been seen before, thus preventing a loop check in the model. On the other hand, a bounded model checker has all states of the current path available on the propositional formula level. Hence, in the latter situation the loop check is easy. The key idea of the transformation is now to augment the model M with a second instance of the state variables to hold a copy of one previously seen state of M. This avoids storing every

line no	constraint	applies to
1	S	$S^{\mathbf{S}}$
2	T	$T^{\mathbf{S}}$
3	I	$I^{\mathbf{S}}$
4	$l_s \Leftrightarrow \bot$	$I^{\mathbf{S}}$
5	$\text{InLoop} \Leftrightarrow \bot$	$I^{\mathbf{S}}$
6	$\text{InLoop}' \Leftrightarrow \text{InLoop} \vee {l_s}'$	$T^{\mathbf{S}}$
7	$\text{InLoop} \Rightarrow \neg {l_s}'$	$T^{\mathbf{S}}$
8	$({l_s}' \Rightarrow \hat{v}' = v) \wedge (\neg {l_s}' \Rightarrow \hat{v}' = \hat{v})$	$T^{\mathbf{S}}$
9	$\forall 0 \leq m \leq f : (\langle\langle Acc_m \rangle\rangle \Leftrightarrow \bot)$	$I^{\mathbf{S}}$
10	$\forall 0 \leq m \leq f : (\langle\langle Acc_m \rangle\rangle' \Leftrightarrow \langle\langle Acc_m \rangle\rangle \vee (\text{InLoop}' \wedge v' \in F_m))$	$T^{\mathbf{S}}$
11	$\text{LoopClosed} \Rightarrow \text{InLoop}$	$S^{\mathbf{S}}$
12	$\text{LoopClosed} \Rightarrow v = \hat{v}$	$S^{\mathbf{S}}$
13	$\forall 0 \leq m \leq f : (\text{LoopClosed} \Rightarrow \langle\langle Acc_m \rangle\rangle)$	$S^{\mathbf{S}}$

Figure 2: Formal definition of the liveness-to-safety transformation

state of a (then necessarily bounded) path. Triggered by an oracle, the augmented model $M^{\mathbf{S}}$ then at some point of a forward exploration guesses the loop start and records that guess in the second instance of the state variables of M. Once the guess has been made, the forward search proceeds as if moving forward from time point l to k in a witness for a bounded model checker: record which acceptance sets have been visited ($M^{\mathbf{S}}$ contains a corresponding set of flags), and, once all of them have been visited, try to close the loop by comparing the current state with the recorded guess.

Formally, the transformation is defined in Fig. 2. Let $M = (S, T, I, L, F = \{F_0, \ldots, F_f\})$ be a fair Kripke structure. Assume, its state space S is made up of a single state variable v with range S. We construct $M^{\mathbf{S}} = (S^{\mathbf{S}}, T^{\mathbf{S}}, I^{\mathbf{S}}, L^{\mathbf{S}})$ as follows. The set of state variables in $M^{\mathbf{S}}$ consists of v, $\hat{v}$, l_s, InLoop, LoopClosed, and, for each acceptance set F_m, $\langle\langle Acc_m \rangle\rangle$. v and $\hat{v}$ have range S, all other variables are Booleans. $S^{\mathbf{S}}$, $T^{\mathbf{S}}$ and $I^{\mathbf{S}}$ are the maximal subsets of $S \times S \times \mathbb{B}^3 \times \mathbb{B}^{f+1}$, $S^{\mathbf{S}} \times S^{\mathbf{S}}$, and $S^{\mathbf{S}}$, respectively, which fulfil the constraints in the following table. $L^{\mathbf{S}}$ is L extended with LoopClosed: $L^{\mathbf{S}}(s^{\mathbf{S}}) = L(s)$ if $\neg\text{LoopClosed}(s^{\mathbf{S}})$, $L(s) \cup \{\text{LoopClosed}\}$ otherwise.

The original instance of the state variables, v, is subject to the same constraints in $M^{\mathbf{S}}$ as in M (lines 1–3). For example, if $s^{\mathbf{S}} \in S^{\mathbf{S}}$, then it must also be the case that $v(s^{\mathbf{S}}) \in S$. Similarly, $(s^{\mathbf{S}}, s^{\mathbf{S}\prime}) \in T^{\mathbf{S}}$ only if $(v(s^{\mathbf{S}}), v(s^{\mathbf{S}\prime})) \in T$. $\hat{v}$ is the second instance of the state variables. When the oracle l_s becomes true, the loop start is guessed by recording the previous value of v in $\hat{v}$ (line 8). InLoop then becomes and remains true to signal the fact that the loop has been started (line 6). It prevents l_s from becoming true for a second time (line 7), which, in turn, ensures that the recorded value in $\hat{v}$ will not be overwritten (line 8). When InLoop is true, visiting an accepting set F_m is recorded in $\langle\langle Acc_m \rangle\rangle$ (line 10). LoopClosed can finally become true to signal that a fair loop has been found when $M^{\mathbf{S}}$ is in the loop, all acceptance sets have been seen, and the valuation of the original instance of the state variables, v, is equal to the guess kept in $\hat{v}$ (lines 11–13).

Note the similarity with the encodings for bounded model checking presented in Sect. 3. Lines 4–7 and line 11 correspond to the loop constraints. Lines 9, 10, and 13 are equivalent to the part of the general Büchi encoding that handles acceptance sets. LoopExists has been renamed to LoopClosed to emphasise that there is no implication from InLoop to LoopClosed and LoopClosed is present in each state while there is only a single instance of LoopExists in BMC. l_i has been turned into oracle l_s, i.e., rather than indicating that a loop exists between states with index $l-1$ and k, it triggers saving the previous value of v in $\hat{v}$. The corresponding check for equality of v and $\hat{v}$ has been shifted to LoopClosed.[6]

Theorem 4.1 states correctness of the construction.

Theorem 4.1. *Let $M = (S, T, I, L, F = \{F_0, \ldots, F_f\})$ be a fair Kripke structure, let $M^{\mathbf{S}}$ be defined as above. M has an initialised fair path π iff some state $s^{\mathbf{S}}$ is reachable in $M^{\mathbf{S}}$ such that* LoopClosed$(s^{\mathbf{S}})$ *is true.*

Proof. For simplicity, we restrict the proof to a single acceptance set F_0. Generalisation to multiple acceptance sets is straightforward. States in $M^{\mathbf{S}}$ are written as tuples $(v, \hat{v}, l_s, \mathrm{InLoop}, \mathrm{LoopClosed}, \langle\langle Acc_0\rangle\rangle)$. Further, it is sufficient to prove the following bi-implication [VW94]:

$$\begin{gathered}\exists\pi = (s_0 \ldots s_{l-1})(s_l \ldots s_m \ldots s_k)^\omega \text{ initialised fair path in } M\\ \text{with } k \geq m \geq l > 0 \wedge s_{l-1} = s_k \wedge s_l, \ldots, s_{m-1} \notin F_0 \wedge s_m \in F_0\\ \Leftrightarrow\\ \exists s^{\mathbf{S}} \text{ reachable in } M^{\mathbf{S}} \text{ such that } \mathrm{LoopClosed}(s^{\mathbf{S}}) \Leftrightarrow \top\end{gathered}$$

"$\Rightarrow$" Let $\pi = (s_0 \ldots s_{l-1})(s_l \ldots s_m \ldots s_k)^\omega$ be an initialised fair path in M with $k \geq m \geq l > 0$, $s_{l-1} = s_k$, $s_l, \ldots, s_{m-1} \notin F_0$, and $s_m \in F_0$. Clearly, for arbitrary $\hat{s}_0 \in S$, $(s_0, \hat{s}_0, \bot, \bot, \bot, \bot) \ldots (s_{l-1}, \hat{s}_0, \bot, \bot, \bot, \bot)$ is an initialised finite path in $M^{\mathbf{S}}$. We extend that prefix to reach a state $s_k^{\mathbf{S}}$ with LoopClosed$(s_k^{\mathbf{S}}) \Leftrightarrow \top$ by distinguishing four cases:

(1) $k = m = l$: Set $s_k^{\mathbf{S}} = s_m^{\mathbf{S}} = s_l^{\mathbf{S}}$ to $(s_k, s_{l-1}, \top, \top, \top, \top)$.
(2) $k = m > l$: Proceed from $s_l^{\mathbf{S}} = (s_l, s_{l-1}, \top, \top, \bot, \bot)$ via $(s_{l+1}, s_{l-1}, \bot, \top, \bot, \bot) \ldots (s_{k-1}, s_{l-1}, \bot, \top, \bot, \bot)$ to $s_m^{\mathbf{S}} = s_k^{\mathbf{S}} = (s_k, s_{l-1}, \bot, \top, \top, \top)$.
(3) $k > m = l$: Continue from $s_m^{\mathbf{S}} = s_l^{\mathbf{S}} = (s_l, s_{l-1}, \top, \top, \bot, \top)$ via $(s_{l+1}, s_{l-1}, \bot, \top, \bot, \top) \ldots (s_{k-1}, s_{l-1}, \bot, \top, \bot, \top)$ to $s_k^{\mathbf{S}} = (s_k, s_{l-1}, \bot, \top, \top, \top)$.
(4) $k > m > l$: Combine cases (2) and (3) to obtain

$$\begin{aligned}&(s_l, s_{l-1}, \top, \top, \bot, \bot)(s_{l+1}, s_{l-1}, \bot, \top, \bot, \bot) \ldots (s_{m-1}, s_{l-1}, \bot, \top, \bot, \bot) \circ\\ &\quad \circ (s_m, s_{l-1}, \bot, \top, \bot, \top) \ldots (s_{k-1}, s_{l-1}, \bot, \top, \bot, \top)(s_k, s_{l-1}, \bot, \top, \top, \top)\end{aligned}$$

"$\Leftarrow$" Let $\widetilde{s^{\mathbf{S}}}$ be a reachable state in $M^{\mathbf{S}}$ with LoopClosed$(\widetilde{s^{\mathbf{S}}}) \Leftrightarrow \top$. Hence, there is an initialised finite path $\widetilde{\pi^{\mathbf{S}}}$ that ends in $\widetilde{s^{\mathbf{S}}}$. Let $\pi^{\mathbf{S}} = s_0^{\mathbf{S}} \ldots s_k^{\mathbf{S}}$ be the prefix of $\widetilde{\pi^{\mathbf{S}}}$ such that $s_k^{\mathbf{S}}$ is the first (and only) state in $\pi^{\mathbf{S}}$ with LoopClosed$(s_k^{\mathbf{S}}) \Leftrightarrow \top$. By definition of $M^{\mathbf{S}}$, InLoop$(s_k^{\mathbf{S}}) \Leftrightarrow \top$, $\hat{v}(s_k^{\mathbf{S}}) = v(s_k^{\mathbf{S}})$, and $\langle\langle Acc_0\rangle\rangle(s_k^{\mathbf{S}}) \Leftrightarrow \top$. Further, InLoop

[6]We state without proof that for a fair (k, l)-loop π there is an initialised path in the transformed model and a satisfying assignment of the general Büchi encoding such that the valuations of l_s, InLoop, and $\langle\langle Acc_m\rangle\rangle$ coincide on corresponding indices of the path.

starts off false at $s_0^{\mathbf{S}}$, switches to true when l_s becomes true at some index $l > 0$, and remains true up to $s_k^{\mathbf{S}}$. Note, that l_s is true only at index $l > 0$. This ensures, that $\hat{v}$ contains an arbitrary $\hat{v}(s_0^{\mathbf{S}})$ from index 0 to $l-1$ and $v(s_{l-1}^{\mathbf{S}})$ from index l onward. Thus, we have $v(s_k^{\mathbf{S}}) = \hat{v}(s_k^{\mathbf{S}}) = \hat{v}(s_l^{\mathbf{S}}) = v(s_{l-1}^{\mathbf{S}})$. $\langle\langle Acc_0\rangle\rangle$ also is false initially and changes at some index $l \leq m \leq k$ to true to remain there up to index k. From the definition of $\langle\langle Acc_0\rangle\rangle$ we have $v(s_m^{\mathbf{S}}) \in F_0$ and $\forall l \leq i < m : v(s_i^{\mathbf{S}}) \notin F_0$. It follows that, depending on the values of k, m, and l, $\pi^{\mathbf{S}}$ corresponds to one of the shapes (1) – (4) outlined in the first part of the proof. By the construction of $M^{\mathbf{S}}$, in all cases $\pi' = s_0 \dots s_l \dots s_m \dots s_k = v(s_0^{\mathbf{S}}) \dots v(s_l^{\mathbf{S}}) \dots v(s_m^{\mathbf{S}}) \dots v(s_k^{\mathbf{S}})$ is an initialised finite path in M with $s_{l-1} = s_k$, $s_l, \dots, s_{m-1} \notin F_0$, and $s_m \in F_0$. Hence, $\pi = (s_0 \dots s_{l-1})(s_l \dots s_m \dots s_k)^\omega$ is an initialised fair path in K as desired. $\square$

The following immediate corollary enables using methods such as [SSS00, ES03, McM03, AFF$^+$05] to obtain a complete bounded model checking procedure for PLTL:

Corollary 4.2. *Given a fair Kripke structure M, M has an initialised fair path π iff there exists a $k \in \mathbb{N}$ such that $|[M^{\mathbf{S}}]|_k \wedge \mathrm{LoopClosed}(s_k^{\mathbf{S}})$ is satisfiable.*

The liveness-to-safety transformation roughly doubles the number of state variables in the model. It can be shown that, with a small modification of the way acceptance sets are handled, radius and diameter of $M^{\mathbf{S}}$ increase only by a small, constant factor [Sch06]. If forward breadth-first search is used for reachability analysis of $M^{\mathbf{S}}$, the proof of Thm. 4.1 implies that a shortest fair looping path in M is found. If M is the product of a model $\tilde{M}$ and a tight Büchi automaton B for some property ψ, that implies that the path is a shortest witness with respect to ψ in $\tilde{M}$.

4.2. **Optimising the Transformation.**

BDD Variable Order. If a BDD-based model checker is used to determine reachability in a transformed model it is important to use a variable order that interleaves the Boolean variables making up s and $s^{\mathbf{S}}$. Otherwise the sizes of the BDDs representing $M^{\mathbf{S}}$ may explode [SB04].

Variable Optimisation. The overhead induced by the transformation of M into $M^{\mathbf{S}}$ mostly stems from the additional instance of the state variables of M present in $M^{\mathbf{S}}$. Hence, leaving some of M's state variables out of loop detection might reduce that overhead. Kroening and Strichman proved in the context of bounded model checking that input variables can be ignored when computing the recurrence diameter for simple liveness properties of the form $\mathbf{F}\, p$ [KS03]. Eén and Sörensson [ES03] use the same idea in temporal induction for safety properties in incremental BMC. We show below that this idea can be extended to the liveness-to-safety transformation.

We call a state variable v_i a *transition input variable* iff its value in the next state, x_i', is not constrained by its value in the current state, x_i, and the values of other variables in the current and next state: if $((x_0, x_1, \dots, x_i, \dots), (x_0', x_1', \dots, x_i', \dots))$ is a transition in T, then, for all $\widetilde{x_i'}$ in the range of v_i, $((x_0, x_1, \dots, x_i, \dots)(x_0', x_1', \dots, \widetilde{x_i'}, \dots))$ is also in T.

A state variable v_i is *irrelevant for fairness* iff its value x_i does not influence whether a state is in an acceptance set or not: for all acceptance sets F_m, for all v_i in V_i, we have that $(x_0, x_1, \dots, x_i, \dots)$ is in F_m iff for all $\widetilde{x}_i$ in the range of v_i, $(x_0, x_1, \dots, \widetilde{x}_i, \dots)$ is also in F_m.

Let V_i be the set of transition input variables that are irrelevant for fairness. Elements of V_i can be left out of loop detection:

Proposition 4.3. *Let M be a fair Kripke structure with set of state variables V and set of transition input variables that are irrelevant for fairness $V_i \subseteq V$. Let $M^{\mathbf{S}}$ be defined as above, let $\widetilde{M^{\mathbf{S}}}$ be the variant of $M^{\mathbf{S}}$ that restricts loop detection (i.e., lines 8 and 12 in the definition of $M^{\mathbf{S}}$) to the variables in $V \setminus V_i$. There is a reachable state $s^{\mathbf{S}}$ such that* $\text{LoopClosed}(s^{\mathbf{S}})$ *is true in $M^{\mathbf{S}}$ iff there is one in $\widetilde{M^{\mathbf{S}}}$.*

Proof. The "$\Rightarrow$"-direction is trivial. For "$\Leftarrow$" it is sufficient to prove the following implication: if $\widetilde{\pi} = s_0 \dots s_{l-1} \dots s_m \dots \widetilde{s_k}$ is an initialised finite path in M with $k \geq m \geq l > 0$, $v(\widetilde{s_k}) = v(s_{l-1})$ for all variables $v \in V \setminus V_i$, and $s_m \in F_0$, then $\widetilde{\pi}$ with its last state replaced by s_{l-1} is an initialised finite path in M with $k \geq m \geq l > 0$, $s_k = s_{l-1}$, and $s_m \in F_0$.

(1) By assumption, $(s_{k-1}, \widetilde{s_k}) \in T$. Construct a sequence of states $\widetilde{s_k} = t_0, t_1, \dots, t_{|V_i|} = s_{l-1}$ such that all t_j, t_{j+1} differ at most by the value of one variable in V_i. By definition, for each t_j, t_{j+1}, $(s_{k-1}, t_j) \in T$ iff $(s_{k-1}, t_{j+1}) \in T$. Hence, $(s_{k-1}, s_{l-1}) \in T$.
(2) If $k > m$, $s_m \in F_0$. Otherwise, use the same sequence of states $\widetilde{s_k} = t_0, t_1, \dots, t_{|V_i|} = s_{l-1}$ to show that $s_m = \widetilde{s_k} \in F_0$ iff $s_{l-1} \in F_0$.

$\square$

Note that the restriction w.r.t. acceptance sets can be dropped if visiting an acceptance set is detected from index $l-1$ to $k-1$ rather than from l to k.

We remark that if the Kripke structure being transformed is the product of a model and a Büchi automaton generated from a PLTL formula, the set of input variables must be determined with respect to both. Hence, input variables of the model that appear in the PLTL property to be verified may need to be included in the loop detection. Clearly, variables that remain constant after initialisation need not be considered for loop detection either. Leaving constant and input variables out of loop detection as described above is referred to as *variable optimisation*.[7] For more aggressive optimisations, which, however, may not preserve length of counterexamples or even lead to false positives, see [Sch06].

Kroening and Strichman assume that input variables are a separate syntactic entity. While a corresponding `IVAR` declaration is available in the NuSMV input language [CCJ+06], many benchmarks were written before NuSMV was available or don't make use of this feature to retain compatibility to the original version of SMV [McM93, CMU]. Therefore, Kroening and Strichman also use an approach based on the transition relation of the system. Eén and Sörensson [ES03] additionally remove output variables. As ignoring these may lead to shorter counterexamples on the reduced set of variables in our approach (though only by one state), they are handled by the more aggressive optimisations in [Sch06].

5. BMC for PLTL

PLTL has features which impact the way model checking can be done. We illustrate these features through a running example, taken from [BC03] and 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

[7]Note that variable optimisation could also be applied in specialised algorithms for bounded model checking such as the one presented in Sect. 6 but this is not currently implemented.

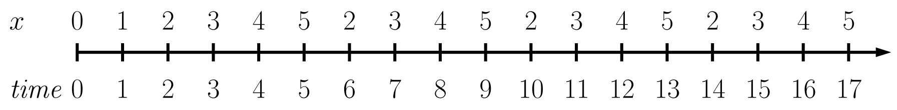


Figure 3: Execution of the counter system

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. 3. 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. 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 5.1. *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 $i < l + ((d+1) \cdot p(\pi))$.*

The formula $\mathbf{Y}\,\mathbf{Y}\,\mathbf{Y}\,(x = 0)$ holds at time point 3, which belongs to the 0-unrolling of the loop. However, at 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 [BC03] 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. [BCCZ99] to include past operators, as they have simple semantics. In the no loop case our encoding reduces to essentially the same as [BC03]. When loops are allowed the matter is more complicated, and therefore we will focus on this part in the rest of this section. 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 [LMS02] and later independently by [BC03]: 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 5.2. *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 Thm. 1 and Lemma 2 of [BC03]. □

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

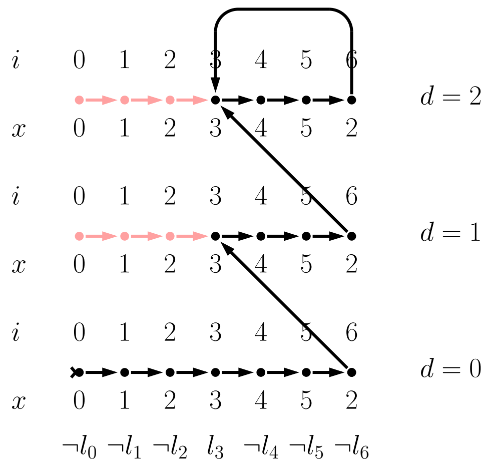
$$\mathbf{F}\ ((x = 3) \wedge \mathbf{O}\ ((x = 4) \wedge \mathbf{O}\ (x = 5))) \tag{5.1}$$

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 the formula in question, which is two. 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 $8 = 16 - (3 - 1) \cdot 4$.

The previous discussion suggests the following extension of the encodings presented in Sect. 3. Intuitively, past temporal operators can be encoded in a similar way as the future operators by using their characterisation in terms of previous and current state values. The issue of stabilisation needs to be dealt with though. Otherwise a subformula can have different truth values at equivalent positions in the path, which can lead to other subformulas being incorrectly evaluated. One way to ensure stabilisation is to extend the loop check $l_i \Rightarrow (s_{i-1} = s_k)$ to also include the truth values of all formula variables (see [KPR98]). While being intuitive and straightforward to implement, the approach just sketched requires that the model is unrolled deep enough so that loop in the model is unrolled to guarantee the stabilisation of all temporal formulas.

Benedetti and Cimatti [BC03] suggested an alternative. The transition relation of the model is only unrolled virtually. Rather than having one variable representing the truth of a subformula at a given index in the loop, several variables $|[\varphi]|_i^d$ are used per subformula φ, which represent the truth of φ at the same relative position i to the underlying finite path but at different unrollings d, see Fig. 4. The number of such variables required for each subformula can be limited by Proposition 5.2.

The bound k at which a particular witness is reported may be different for both variants. The first variant cannot guarantee that the minimal length witnesses are found. However, if the bound required by the first variant is not much larger than that of the alternative, even with a higher bound the first encoding may be more compact as only one variable per subformula and index is introduced. On the other hand, if several unrollings of the loop are required for stabilisation, the second variant may be more compact: in that case, savings due to having fewer instances of the transition relation of the model will more than compensate for the overhead introduced by the virtual unrolling of the formula variables.



i 0 1 2 3 4 5 6
x 0 1 2 3 4 5 2
$d = 2$
$d = 1$
$d = 0$
$\neg l_0$ $\neg l_1$ $\neg l_2$ l_3 $\neg l_4$ $\neg l_5$ $\neg l_6$

Figure 4: 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

Below we develop a propositional encoding of the BMC problem for PLTL that integrates both variants. We first use the idea of Benedetti and Cimatti [BC03] to extend the eventuality encoding for LTL with past formulas as it is our encoding of choice for an incremental SAT encoding to be presented in Sect. 6. Based on that we briefly discuss how the other LTL encodings can also be extended to PLTL along similar lines. In fact, the encoding presented in this section is essentially a non-incremental version of the incremental PLTL encoding presented in [HJL05]. We then show that by adding a check for stabilisation of all temporal subformulas, the level of virtual unrolling can be chosen freely between full and no unrolling. Finally, we extend the idea of virtual unrolling to Büchi automata.

5.1. **BMC for PLTL with Eventualities.** The basic idea of the encoding is to virtually unroll the path by making several copies of the original finite 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 subformula φ is dictated by its 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. Figure 4 shows the running example unrolled to depth $d = 2$, for evaluating the formula (5.1).

First of all the *PLTL eventuality encoding* contains the model constraints $|[M]|_k$ and the loop constraints $|[\mathit{LoopConstraints}]|_k$ which are both encoded exactly as in the LTL case.

To represent the original path and its copies, the PLTL formula variables $|[\varphi]|_i^d$ have 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 Proposition 5.2 be mapped to the depth corresponding to the past operator depth. From this we get our first rule for each subformula $\varphi \in cl(\psi)$:

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

The rest of the encoding is split into cases based on the values of i and d. The encoding for propositional formulas is the same as in the LTL case except that each subformula has constraints for several different d-unrollings. Constraints for atomic propositions and their negation are straightforward. We simply project the atomic propositions onto the original path. The Boolean operators $\vee$ and $\wedge$ are encoded to stay in the current d-unrolling.

φ	$0 \leq i \leq k, 0 \leq d \leq \delta(\varphi)$
p	$\lvert[p]\rvert_i^d \Leftrightarrow p \in L(s_i)$
$\neg p$	$\lvert[\neg p]\rvert_i^d \Leftrightarrow p \notin L(s_i)$
$\psi_1 \wedge \psi_2$	$\lvert[\psi_1 \wedge \psi_2]\rvert_i^d \Leftrightarrow \lvert[\psi_1]\rvert_i^d \wedge \lvert[\psi_2]\rvert_i^d$
$\psi_1 \vee \psi_2$	$\lvert[\psi_1 \vee \psi_2]\rvert_i^d \Leftrightarrow \lvert[\psi_1]\rvert_i^d \vee \lvert[\psi_2]\rvert_i^d$

The translation of the future operators is also a very straightforward generalisation of the pure future LTL encoding of Sect. 3.2; we just have introduce constraints for all d-unrollings.

φ	$0 \leq i \leq k, 0 \leq d \leq \delta(\varphi)$
$\mathbf{X}\,\psi_1$	$\lvert[\mathbf{X}\,\psi_1]\rvert_i^d \Leftrightarrow \lvert[\psi_1]\rvert_{i+1}^d$
$\psi_1\,\mathbf{U}\,\psi_2$	$\lvert[\psi_1\,\mathbf{U}\,\psi_2]\rvert_i^d \Leftrightarrow \lvert[\psi_2]\rvert_i^d \vee \left(\lvert[\psi_1]\rvert_i^d \wedge \lvert[\psi_1\,\mathbf{U}\,\psi_2]\rvert_{i+1}^d\right)$
$\psi_1\,\mathbf{R}\,\psi_2$	$\lvert[\psi_1\,\mathbf{R}\,\psi_2]\rvert_i^d \Leftrightarrow \lvert[\psi_2]\rvert_i^d \wedge \left(\lvert[\psi_1]\rvert_i^d \vee \lvert[\psi_1\,\mathbf{R}\,\psi_2]\rvert_{i+1}^d\right)$

The $\lvert[\mathit{LastStateFormula}]\rvert_k$ constraints of the LTL case have to be changed in the PLTL case to take care of binding the different unrollings of the encoding together in the way shown by following the black arcs of Fig. 4 in the forward direction. The truth values of $\lvert[\varphi]\rvert_{k+1}^d$ are picked from the loop point i of the next unrolling level $\lvert[\varphi]\rvert_i^{d+1}$, or if we are at the last level $d = \delta(\varphi)$ then from the loop point at the last level $\lvert[\varphi]\rvert_i^{\delta(\varphi)}$. This is achieved by the expression $\lvert[\varphi]\rvert_i^{min(d+1,\delta(\varphi))}$. For all $\varphi \in cl(\psi)$ the following constraints are created:

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

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 all subformulas has stabilised (see Proposition 5.2). Therefore at the maximum unrolling depth we add the auxiliary translation constraints which, similarly to the LTL case, are needed to correctly evaluate the until and release formulas along the loop.

	φ	
Base	$\psi_1 \mathbf{U} \psi_2$	$\mathit{LoopExists} \Rightarrow \left(\vert[\psi_1 \mathbf{U} \psi_2]\vert_k^{\delta(\varphi)} \Rightarrow \langle\langle \mathbf{F} \psi_2 \rangle\rangle_k^{\delta(\psi_2)}\right)$
	$\psi_1 \mathbf{R} \psi_2$	$\mathit{LoopExists} \Rightarrow \left(\vert[\psi_1 \mathbf{R} \psi_2]\vert_k^{\delta(\varphi)} \Leftarrow \langle\langle \mathbf{G} \psi_2 \rangle\rangle_k^{\delta(\psi_2)}\right)$
	$\psi_1 \mathbf{U} \psi_2$	$\langle\langle \mathbf{F} \psi_2 \rangle\rangle_0^{\delta(\psi_2)} \Leftrightarrow \bot$
	$\psi_1 \mathbf{R} \psi_2$	$\langle\langle \mathbf{G} \psi_2 \rangle\rangle_0^{\delta(\psi_2)} \Leftrightarrow \top$
$1 \le i \le k$	$\psi_1 \mathbf{U} \psi_2$	$\langle\langle \mathbf{F} \psi_2 \rangle\rangle_i^{\delta(\psi_2)} \Leftrightarrow \langle\langle \mathbf{F} \psi_2 \rangle\rangle_{i-1}^{\delta(\psi_2)} \vee \left(\mathrm{InLoop}_i \wedge \vert[\psi_2]\vert_i^{\delta(\psi_2)}\right)$
	$\psi_1 \mathbf{R} \psi_2$	$\langle\langle \mathbf{G} \psi_2 \rangle\rangle_i^{\delta(\psi_2)} \Leftrightarrow \langle\langle \mathbf{G} \psi_2 \rangle\rangle_{i-1}^{\delta(\psi_2)} \wedge \left(\neg\mathrm{InLoop}_i \vee \vert[\psi_2]\vert_i^{\delta(\psi_2)}\right)$

The starting point for the encoding for the past operators is using their characterisation in terms of the current and the previous state. This enables the encoding of the past operators to fit in nicely 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.

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 not need to take into account the loop edge, so the encoding follows from the characterisation $\psi_1 \mathbf{S} \psi_2$ and $\psi_1 \mathbf{T} \psi_2$ in terms of the current and the previous state. Encoding $\mathbf{Y} \psi_1$ and $\mathbf{Z} \psi_1$ is trivial.[8]

φ	$i = 0, 0 \le d \le \delta(\varphi)$	$1 \le i \le k, d = 0$
$\psi_1 \mathbf{S} \psi_2$	$\vert[\psi_1 \mathbf{S} \psi_2]\vert_i^d \Leftrightarrow \vert[\psi_2]\vert_i^d$	$\vert[\psi_1 \mathbf{S} \psi_2]\vert_i^d \Leftrightarrow \vert[\psi_2]\vert_i^d \vee \left(\vert[\psi_1]\vert_i^d \wedge \vert[\psi_1 \mathbf{S} \psi_2]\vert_{i-1}^d\right)$
$\psi_1 \mathbf{T} \psi_2$	$\vert[\psi_1 \mathbf{T} \psi_2]\vert_i^d \Leftrightarrow \vert[\psi_2]\vert_i^d$	$\vert[\psi_1 \mathbf{T} \psi_2]\vert_i^d \Leftrightarrow \vert[\psi_2]\vert_i^d \wedge \left(\vert[\psi_1]\vert_i^d \vee \vert[\psi_1 \mathbf{T} \psi_2]\vert_{i-1}^d\right)$
$\mathbf{Y} \psi_1$	$\vert[\mathbf{Y} \psi_1]\vert_i^d \Leftrightarrow \bot$	$\vert[\mathbf{Y} \psi_1]\vert_i^d \Leftrightarrow \vert[\psi_1]\vert_{i-1}^d$
$\mathbf{Z} \psi_1$	$\vert[\mathbf{Z} \psi_1]\vert_i^d \Leftrightarrow \top$	$\vert[\mathbf{Z} \psi_1]\vert_i^d \Leftrightarrow \vert[\psi_1]\vert_{i-1}^d$

When $d > 0$ the key challenge 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. 4 in the reverse direction. We implement the choice with an if-then-else construct $(l_i \wedge \varphi_1) \vee (\neg l_i \wedge \varphi_2)$. The expression encodes the choice if l_i is true then the truth value of the expression is decided by φ_1, otherwise φ_2 decides the truth value of the expression.

φ	$1 \le i \le k, 1 \le d \le \delta(\varphi)$
$\psi_1 \mathbf{S} \psi_2$	$\vert[\psi_1 \mathbf{S} \psi_2]\vert_i^d \Leftrightarrow \vert[\psi_2]\vert_i^d \vee \left(\vert[\psi_1]\vert_i^d \wedge \left(\left(l_i \wedge \vert[\varphi]\vert_k^{d-1}\right) \vee \left(\neg l_i \wedge \vert[\varphi]\vert_{i-1}^d\right)\right)\right)$
$\psi_1 \mathbf{T} \psi_2$	$\vert[\psi_1 \mathbf{T} \psi_2]\vert_i^d \Leftrightarrow \vert[\psi_2]\vert_i^d \wedge \left(\vert[\psi_1]\vert_i^d \vee \left(\left(l_i \wedge \vert[\varphi]\vert_k^{d-1}\right) \vee \left(\neg l_i \wedge \vert[\varphi]\vert_{i-1}^d\right)\right)\right)$
$\mathbf{Y} \psi_1$	$\vert[\mathbf{Y} \psi_1]\vert_i^d \Leftrightarrow \left(l_i \wedge \vert[\psi_1]\vert_k^{d-1}\right) \vee \left(\neg l_i \wedge \vert[\psi_1]\vert_{i-1}^d\right)$
$\mathbf{Z} \psi_1$	$\vert[\mathbf{Z} \psi_1]\vert_i^d \Leftrightarrow \left(l_i \wedge \vert[\psi_1]\vert_k^{d-1}\right) \vee \left(\neg l_i \wedge \vert[\psi_1]\vert_{i-1}^d\right)$

[8]The column $i = 0$ has been included to make all unrollings evaluate exactly the same truth values in the no-loop case, which has a slight advantage if the encoding is used in a complete model checking procedure as described in Section 7.

Combining the tables above we get the full PLTL encoding $|[\mathit{EventualityPLTL}]|_k$ for ψ. Given a Kripke structure M, a PLTL formula ψ, and a bound k, the *PLTL eventuality encoding* as a propositional formula is given by:

$$|[M, \psi, k]| = |[M]|_k \wedge |[\mathit{LoopConstraints}]|_k \wedge |[\mathit{LastStateFormula}]|_k \wedge |[\mathit{EventualityPLTL}]|_k \wedge |[\psi]|_0^0.$$

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

Theorem 5.3. *Given a Kripke structure M and a PLTL formula ψ, M has an initialised path π such that $\pi \models \psi$ iff there exists a $k \in \mathbb{N}$ such that the PLTL eventuality encoding $|[M, \psi, k]|$ is satisfiable. In particular, if $\pi \models_k \psi$ then the PLTL eventuality encoding $|[M, \psi, k]|$ is satisfiable.* [9]

Proof. We proceed similarly to the proof of Thm. 3.2, only changes are given below. The main change to the future only LTL encoding is that all the subformulas $\varphi \in cl(\psi)$ are virtually unrolled to their past operator depth $\delta(\varphi)$. In addition the new past formula encodings have been introduced.

First consider the (k, j)-loop case (a): We have the same induction scheme as in the proof of Thm. 3.2. The main change is that we have to take the virtual unrolling into account. We will prove by induction on the structure of the PLTL formula ψ that the PLTL eventuality encoding is satisfiable with a unique satisfying truth assignment. Moreover, for all pairs of indices i, d in $0 \leq i \leq k, 0 \leq d \leq \delta(\varphi)$ such that $d = 0$ or $i \geq j$ (we are in the black nodes of Fig. 4) it holds that $\pi^{i+(d \cdot p(\pi))} \models_k \varphi$ iff in the unique satisfying truth assignment of the PLTL eventuality encoding $|[\varphi]|_i^d$ is true.

For a future subformula $\varphi \in cl(\psi)$ (the top-level subformula of φ is a future time formula) we do this by first proving that if the encoding is satisfiable, the variable $|[\varphi]|_k^{\delta(\varphi)}$ for the last state of the top unrolling of Fig. 4 is true iff $\pi^{k+(\delta(\varphi) \cdot p(\pi))} \models_k \varphi$. This is done similarly to the proof of Thm. 3.2; only small indexing changes are needed in order to always refer to states in the unrolling $\delta(\varphi)$ both for the encoding and for the PLTL semantics. All formulas referred to in the proof have in the unrolling $\delta(\varphi)$ stabilised by Proposition 5.2 and thus we get that if the encoding is satisfiable, $|[\varphi]|_k^{\delta(\varphi)}$ is true iff $\pi^{k+(\delta(\varphi) \cdot p(\pi))} \models_k \varphi$. Now it is also easy to check that the encoding for all other pairs of indices i, d in $0 \leq i \leq k, 0 \leq d \leq \delta(\varphi)$ such that $d = 0$ or $i \geq j$ follows the one-step identities of the bounded PLTL semantics for φ in a functional manner (proof by induction following the straight black arcs of Fig. 4 in the reverse direction jumping from one unrolling to the previous as shown by the arcs) and thus the truth assignment matching the bounded PLTL semantics leads to the only truth assignment satisfying all constraints of the encoding. The new part in this proof compared to the future case is that we also have to prove for all pairs of indexes $0 \leq i \leq k, 0 \leq d \leq \delta(\varphi)$ such that $d > 0$ and $i < j$ the corresponding constraints are satisfiable in a unique way. This is the case because these constraints can be seen to form Boolean circuits where all inputs are fixed and the output is not constrained in any way. We thus obtain a unique satisfying truth assignment for the full PLTL eventuality encoding in a similar manner as in the proof of Thm. 3.2.

For a past formula $\varphi \in cl(\psi)$ the proof starts by showing that if the encoding is satisfiable, then $|[\varphi]|_0^0$ corresponding to the first state of the bottom unrolling of Fig. 4 is true

[9] As immediate corollary minimal length (k, l)-loop counterexamples for PLTL can be detected. The encoding also detects minimal length informative safety counterexamples for PLTL.

iff $\pi^0 \models_k \varphi$. This can be easily checked by comparing the encoding of $|[\varphi]|_0^0$ with the PLTL semantics of past formulas combined with our induction hypothesis that the subformulas are correctly evaluated. Now it is also easy to check that the rest of the encoding for all other pairs of indices i, d in $0 \le i \le k, 0 \le d \le \delta(\varphi)$ such that $d = 0$ or $i \ge j$ follows the one-step identities of the bounded PLTL semantics for φ in a functional manner (proof by induction following the straight black arcs of Fig. 4 in the forward direction jumping from one unrolling to the next as shown by the arcs) and thus the truth assignment matching the bounded PLTL semantics for φ leads to the only truth assignment satisfying all constraints of the encoding. The new part in this proof compared to the future case is that we also have to prove for all pairs of indexes $0 \le i \le k, 0 \le d \le \delta(\varphi)$ such that $d > 0$ and $i < j$ the corresponding constraints are satisfiable in a unique way. This is the case because these constraints can be seen to form Boolean circuits where all the inputs are fixed and the output is not constrained in any way. We thus obtain a unique satisfying truth assignment for the full PLTL eventuality encoding in a similar manner as in the proof of Thm. 3.2.

Next consider the no-loop case (b): We first note that in the no-loop case LoopExists is false and in this case the encoding for all indexes $d > 0$ can be seen to form Boolean circuits where all the inputs are fixed and the output is not constrained in any way. Thus all of these constraints are satisfiable in a unique way.

Therefore we need to only consider the case $d = 0, 0 \le i \le k$. We proceed similarly to the proof of Thm. 3.2 for future PLTL formulas, but due to the simplicity of the proof we reproduce it here. Because LoopExists is false, it is easy to see that the $|[\mathit{LastStateFormula}]|_k$ constraints will force the proxy variables $|[\varphi]|_{k+1}^0$ to $\bot$, and the encoding becomes exactly the same as in the fixpoint encoding case and thus has a unique satisfying truth assignment. Also the auxiliary encoding constraints will lead to a unique satisfying truth assignment as as *LoopExists* is false.

For a past PLTL formula φ we first find that if the encoding is satisfiable, $|[\varphi]|_0^0$ is true iff $\pi^0 \models_{\text{nl}} \varphi$. This can be easily checked by comparing the encoding of $|[\varphi]|_0^0$ with the no-loop case PLTL semantics of past formulas combined with our induction hypothesis that the subformulas are correctly evaluated. It is also easy to check that the rest of the encoding for all other indices $0 < i \le k$ follows the one-step identities of the no-loop case PLTL semantics for φ in a functional manner, and thus the truth assignment matching the no-loop case PLTL semantics for φ leads to the only truth assignment satisfying all constraints of the encoding. □

The size of the encoding is $O(|I| + k \cdot |T| + k \cdot |\psi| \cdot \delta(\psi))$. The encoding for PLTL above also has the unique model property in the same sense as in the LTL case. The unique model property allows us to read the exact bounded semantics for all PLTL subformulas and all time indexes considered directly from the truth assignment given by the SAT engine. In fact, it also evaluates some value for the formula variables in the light nodes of Fig. 4. These nodes could be easily detected and forced to some fixed value (e.g., $\bot$) but that would make the encoding slightly larger. For the BMC encoding we preferred not to do that, as the truth values of these nodes do not matter because they cannot be referenced from $|[\psi]|_0^0$ by either forward or backward arcs.

Similarly to the LTL case, the PLTL eventuality encoding of this section (see Sect. 3.2 for the LTL version) can alternatively be replaced with either a PLTL fixpoint evaluation encoding [LBHJ05] (see Sect. 3.1 for the LTL version) or the Büchi encoding (see Sect. 3.3 for the LTL case). Intuitively the main difference to the LTL case is evaluating the required auxiliary encoding at such an unrolling depth $d = \delta(\varphi)$ that the evaluated formula φ has stabilised according to Proposition 5.2.

Partial Unrolling. An interesting feature of the PLTL encoding is that a simple modification makes it sound even if 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 $|\psi|$, and a Büchi encoding variant of the PLTL encoding becomes essentially a BMC encoding of [KPR98], see also [SB05]. In fact, we can limit the maximum virtual unrolling depth of any subformula to any value d_{max} between zero (minimal size encoding, potentially longer counterexamples) and $\delta(\varphi)$ (minimal length counterexamples, larger encoding). Counterexamples will still be detected but the bound required to do so will depend on the amount of unrolling done.

For the last unrolling we have to add the *stabilisation forcing constraints* shown below which constrain the past formulas to also consider that the predecessor can be the last state of the last unrolling. Such constraints are also required for the encoding of [CRS04] to work correctly for formulas containing past operators; [CRS04] does not state this explicitly. The intuition for the stabilisation forcing constraints is that they ensure that past formulas in the loop state of the last unrolling evaluate to the same truth value, no matter whether it is seen as the successor of the end state at current or the previous unrolling. In other words, all subformulas have stabilised. Proposition 5.2 will guarantee that when we have unrolled to the maximum depth $\delta(\varphi)$, these constraints will not remove any satisfying models of the encoding as the truth values of all formulas, in particular all the past formulas themselves, have stabilised when the last unrolling has been reached.

φ	$1 \le i \le k, d = \delta(\varphi)$
$\psi_1 \mathbf{S} \psi_2$	$\lvert[\psi_1 \mathbf{S} \psi_2]\rvert_i^d \Leftrightarrow \lvert[\psi_2]\rvert_i^d \vee \Big(\lvert[\psi_1]\rvert_i^d \wedge \Big(\big(l_i \wedge \lvert[\varphi]\rvert_k^d\big) \vee \big(\neg l_i \wedge \lvert[\varphi]\rvert_{i-1}^d\big)\Big)\Big)$
$\psi_1 \mathbf{T} \psi_2$	$\lvert[\psi_1 \mathbf{T} \psi_2]\rvert_i^d \Leftrightarrow \lvert[\psi_2]\rvert_i^d \wedge \Big(\lvert[\psi_1]\rvert_i^d \vee \Big(\big(l_i \wedge \lvert[\varphi]\rvert_k^d\big) \vee \big(\neg l_i \wedge \lvert[\varphi]\rvert_{i-1}^d\big)\Big)\Big)$
$\mathbf{Y} \psi_1$	$\lvert[\mathbf{Y} \psi_1]\rvert_i^d \Leftrightarrow \big(l_i \wedge \lvert[\psi_1]\rvert_k^d\big) \vee \big(\neg l_i \wedge \lvert[\psi_1]\rvert_{i-1}^d\big)$
$\mathbf{Z} \psi_1$	$\lvert[\mathbf{Z} \psi_1]\rvert_i^d \Leftrightarrow \big(l_i \wedge \lvert[\psi_1]\rvert_k^d\big) \vee \big(\neg l_i \wedge \lvert[\psi_1]\rvert_{i-1}^d\big)$

The correctness of the stabilisation constraints is not difficult to see. If we assume that for every past subformula $\varphi \in cl(\psi)$ it holds that $\pi^{j+(d_{max}\cdot p(\pi))} \models_k \varphi$ iff $\pi^{k+1+(d_{max}\cdot p(\pi))} \models_k \varphi$ then we can easily prove that the evaluated formula has stabilised for all subformulas at all indices in the unrolling d_{max}.

To prove soundness of the modified encoding we proceed as follows. If the assumption of stabilisation at d_{max} does not hold, we can find a past time subformula φ such that all its subformulas have stabilised at d_{max} but $\pi^{j+(d_{max}\cdot p(\pi))} \models_k \varphi$ iff $\pi^{k+1+(d_{max}\cdot p(\pi))} \models_k \varphi$ does not hold. In this case it is easy to see that the original constraints force $|[\varphi]|_j^{d_{max}}$ to true iff $\pi^{j+(d_{max}\cdot p(\pi))} \models_k \varphi$, and it is easy to prove that the stabilisation forcing constraints force $|[\varphi]|_j^{d_{max}}$ to true iff $\pi^{k+1+(d_{max}\cdot p(\pi))} \models_k \varphi$. Therefore the whole encoding becomes unsatisfiable.

For completeness we note that Proposition 5.2 ensures that eventually all PLTL formulas become periodic. This ensures that eventually all past subformulas will satisfy the stabilisation assumption above with any value $0 \le d_{max} \le \delta(\psi)$ when k is increased large enough, for some value of j. If the assumption about stabilisation holds, then by using $min(d_{max}, \delta(\varphi))$ in the encoding and in the Proof of Thm. 5.3 instead of $\delta(\varphi)$, we can prove the encoding to be satisfiable and matching the bounded semantics of PLTL using our assumption about the stabilisation at unrolling d_{max} instead of Proposition 5.2. The only new thing that needs to be proven is that the stabilisation enforcing constraints are satisfiable, and this is immediate by the new constraints and our assumption of the stabilisation of all past subformulas at the unrolling level d_{max}.

As a historical note, at the point of writing [LBHJ05] we were unfortunately not aware of the symbolic PLTL Büchi automata translation of [KPR98]. Quite late in writing [HJL05] we became aware of it by stumbling on a bug — stemming from the ambiguity mentioned above — in an unpublished prototype implementation of [CRS04] kindly provided to us by its authors. After that we discovered that [KPR98] did not have that problem, we quickly figured out how to use a similar optimisation in our context.

5.2. **Virtual Unrolling for Büchi Automata.** In this subsection we extend the idea of virtual unrolling to Büchi automata. Starting from a Büchi automaton $\tilde{B}^{\psi}$ based on [KPR98], which is tight only if ψ is a future time formula [SB05], we obtain a Büchi automaton B^{ψ} accepting the same language that is tight for all PLTL formulae ψ.

The situation is very similar to the BMC case: on a shortest witness, the original Büchi automaton $\tilde{B}^{\psi}$ needs some additional unrollings of the transition relation of the model M till both have a loop of the same length. Note, that the intuition provided by the example below does not rely on the fact that $\tilde{B}^{\psi}$ is derived from [KPR98]. It only requires $\tilde{B}^{\psi}$ to have an accepting loop of the same length as the witness. Further generalisation to arbitrary Büchi automata is possible but so far of mostly theoretical interest [Sch06].

For technical reasons we have to deviate from the convention that l_i is true at index l of a (k, l)-loop and InLoop is true from index l through index k. Rather, both are shifted one state towards the initial state, i.e., l_i is true at index $l - 1$ (which could be regarded as being the loop start as well) and, correspondingly, InLoop is true from $l - 1$ through k.

The construction of the tight Büchi automaton is by and large the same as in [SB05]. The presentation is changed to highlight similarities with the encoding of PLTL for BMC in the previous subsection.

Example. We first walk through the steps of the construction using our running example in Fig. 3. Figure 5a shows a run of a [KPR98]-like Büchi automaton $\tilde{B}^{\psi}$ on the path $(01)(2345)^{\omega}$ — remember, that we start the loop one state earlier in this subsection. The model M enters a loop of length 4 at time point 2 while $\tilde{B}^{\psi}$ needs 6 more states until it enters a loop of the same length. An accepting loop in the product $M \times \tilde{B}^{\psi}$ can be closed only at time point 12 (the last occurrence of $x = 4$ in Fig. 5a).

By virtually unrolling the transition relation of M (or, in other words, by folding in the transition relation of $\tilde{B}^{\psi}$) some parts of the run of $\tilde{B}^{\psi}$ can take place in parallel to reduce some or all of the excess length (Fig. 5b,c). So far, there is no difference to the BMC case. We now have to decide how to define states, transition relation, and acceptance sets of the new automaton B^{ψ}.

The states of B^{ψ} consist of tuples of states of $\tilde{B}^{\psi}$ (Fig. 5d). Before the loop starts the tuples need only have size 1, i.e., they are identical to the states of $\tilde{B}^{\psi}$. After the loop start the tuples must be able to accommodate the maximal excess length of an accepting run of $\tilde{B}^{\psi}$. If $\tilde{B}^{\psi}$ is derived from [KPR98] we can obtain a similar result as in Prop. 5.2 on the excess length of accepting runs of $\tilde{B}^{\psi}$ [SB05]. Hence, the maximum required size of the

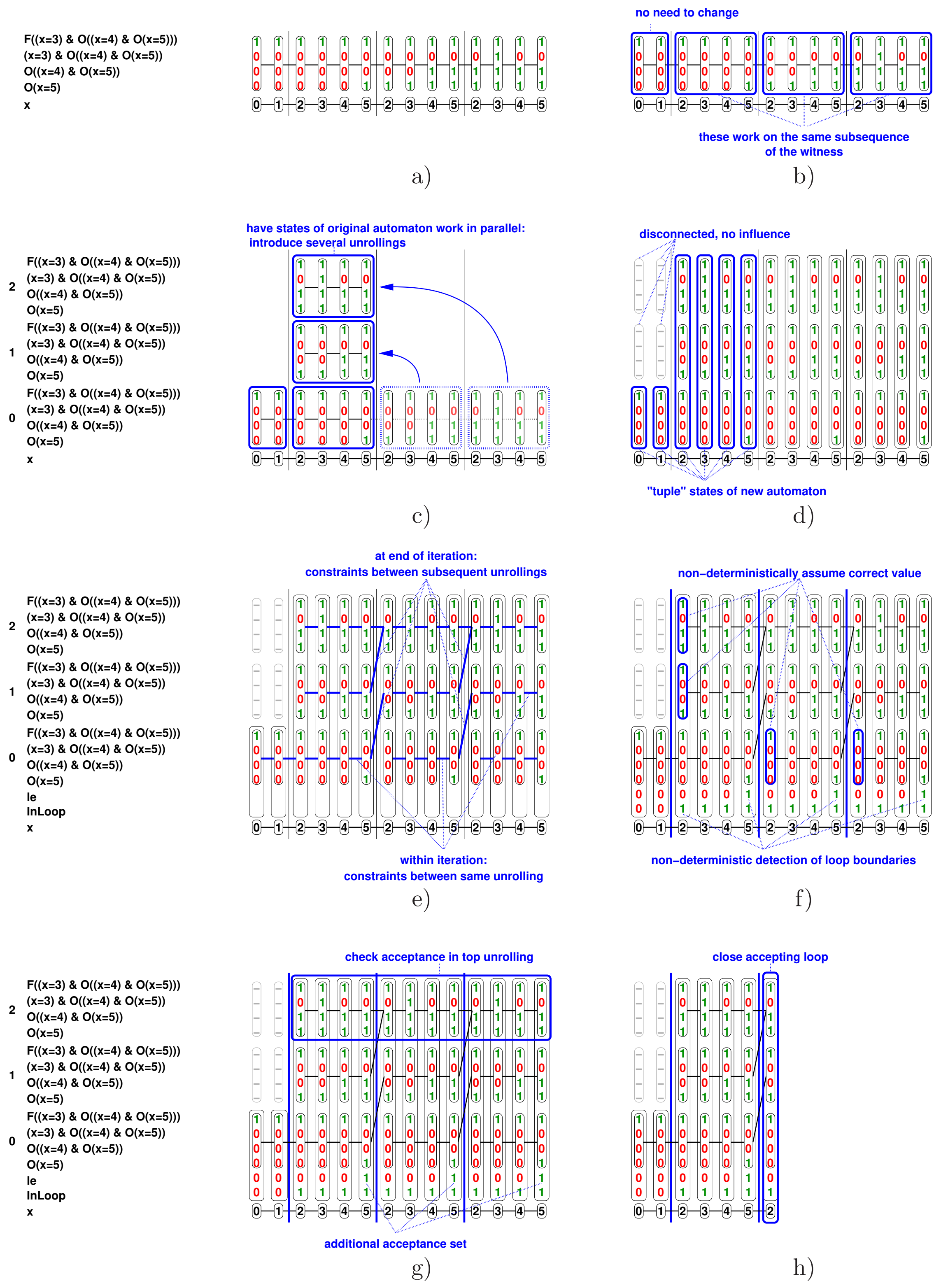

F((x=3) & O((x=4) & O(x=5)))
(x=3) & O((x=4) & O(x=5))
O((x=4) & O(x=5))
O(x=5)
x
a)
no need to change
these work on the same subsequence of the witness
b)
have states of original automaton work in parallel: introduce several unrollings
c)
disconnected, no influence
"tuple" states of new automaton
d)
le
InLoop
at end of iteration: constraints between subsequent unrollings
within iteration: constraints between same unrolling
e)
non−deterministically assume correct value
non−deterministic detection of loop boundaries
f)
check acceptance in top unrolling
additional acceptance set
g)
close accepting loop
h)

Figure 5: Tightening [KPR98] by example

tuples is given by the past operator depth of ψ plus one. In practice, the tuples before the loop start also have that size but its constituent states at unrollings > 0 are disconnected from the rest of the automaton. Note that the states of B^{ψ} at time points 6–9 and 10–13 are the same as at time points 2–5.

Defining transitions not crossing a loop boundary is easy: there is a transition from one tuple state to another in B^{ψ} iff each pair of constituent states at the same unrolling has a transition in $\tilde{B}^{\psi}$ (Fig. 5e). When crossing a loop boundary, a constituent state at unrolling $d-1$ in the pre-state is connected to a constituent state at unrolling d in the post-state. In addition, there must be a transition in $\tilde{B}^{\psi}$ between the constituent states at the highest unrolling of the pre- and post-state to ensure that a loop exists in $\tilde{B}^{\psi}$.

Clearly, we cannot know in which state $\tilde{B}^{\psi}$ would be in an unrolling > 0 when first entering the loop in B^{ψ} (time point 2 in Fig. 5f). Hence, the corresponding constituent states in B^{ψ} are not constrained to the past.[10] The constituent states at unrolling 0 at time points 6 and 10 could in principle be forced to be identical to their predecessor at time point 2; however, it turns out that this is not required for correctness of the construction. The loop boundaries are "detected" non-deterministically using oracle variables InLoop with the same meaning as before and *le* indicating the last state of a loop iteration.

As acceptance of a run in $\tilde{B}^{\psi}$ is determined in its looping part, each tuple state in B^{ψ} is in the acceptance set $\tilde{F}_m$ of B^{ψ} iff its constituent state in the top unrolling belongs to the corresponding acceptance set F_m of $\tilde{B}^{\psi}$ (Fig. 5g). One additional acceptance set is needed in B^{ψ} to guarantee that infinitely often a loop boundary is guessed. Otherwise, there might not be a connection between the bottom and top unrollings and, therefore, acceptance might not be determined correctly. Finally, an accepting loop can be closed (Fig. 5h).

Construction. We symbolically construct a Büchi automaton $B^{\psi} = (S, T, I, L, F)$ for a PLTL formula ψ as follows. For each $\varphi \in cl(\psi)$, V contains state variables $|[\varphi]|^0, \ldots, |[\varphi]|^{\delta(\varphi)}$ meant to represent the truth of φ at unrollings $0 \leq i \leq \delta(\varphi)$. Two oracles InLoop and *le* signal the presumed start of the loop and the end of each loop iteration. The rest of the encoding is developed step by step below.

line	constraint	applies to
1	$\text{InLoop} \Rightarrow \text{InLoop}'$	T
2	$le \Rightarrow \text{InLoop}$	S

As in the BMC case we set $|[\varphi]|^d \Leftrightarrow |[\varphi]|^{\delta(\varphi)}$ if $d > \delta(\varphi)$. The state variables for atomic propositions are unconstrained; their valuations are linked to the corresponding atomic propositions via L, though.[11] The valuation of the state variables for Boolean operators is again the same as in the previous subsection:

[10]If $\tilde{B}^{\psi}$ is derived from [KPR98] some constraints similar to those in Sect. 5 of [HJL05] could be applied for monotonic operators.

[11]Here we assume that the product of Kripke structures is formed by demanding that product states match on shared atomic propositions, see, e.g., [Sch06]. In a symbolic setting atomic propositions often correspond directly to valuations of state variables and, hence, the product can be formed more directly by sharing these state variables.

line	φ	$0 \le d \le \delta(\varphi)$	applies to
3	p	$\top$	S
4	$\neg p$	$\top$	S
5	$\psi_1 \wedge \psi_2$	$\vert[\psi_1 \wedge \psi_2]\vert^d \Leftrightarrow \vert[\psi_1]\vert^d \wedge \vert[\psi_2]\vert^d$	S
6	$\psi_1 \vee \psi_2$	$\vert[\psi_1 \vee \psi_2]\vert^d \Leftrightarrow \vert[\psi_1]\vert^d \vee \vert[\psi_2]\vert^d$	S

Within a loop iteration and on the stem the valuation of the variables for temporal operators directly follows their characterisation in terms of current and next state values. Note that stem and loop are disconnected at unrollings > 0. In the following tables we sometimes use parentheses to disambiguate the scope of the next state operator $'$ and 'applies to' abbreviated with a.t.

line	φ	$0 \le d \le \delta(\varphi)$	a.t.
7	$\mathbf{X}\,\psi_1$	$\neg le \wedge \neg(\neg \text{InLoop} \wedge \text{InLoop}' \wedge d > 0) \Rightarrow (\vert[\mathbf{X}\,\psi_1]\vert^d \Leftrightarrow (\vert[\psi_1]\vert^d)')$	T
8	$\psi_1\,\mathbf{U}\,\psi_2$	$\neg le \wedge \neg(\neg \text{InLoop} \wedge \text{InLoop}' \wedge d > 0) \Rightarrow (\vert[\psi_1\,\mathbf{U}\,\psi_2]\vert^d \Leftrightarrow \vert[\psi_2]\vert^d \vee (\vert[\psi_1]\vert^d \wedge (\vert[\varphi]\vert^d)'))$	T
9	$\psi_1\,\mathbf{R}\,\psi_2$	$\neg le \wedge \neg(\neg \text{InLoop} \wedge \text{InLoop}' \wedge d > 0) \Rightarrow (\vert[\psi_1\,\mathbf{R}\,\psi_2]\vert^d \Leftrightarrow \vert[\psi_2]\vert^d \wedge (\vert[\psi_1]\vert^d \vee (\vert[\varphi]\vert^d)'))$	T
10	$\mathbf{Y}\,\psi_1$	$\neg le \wedge \neg(\neg \text{InLoop} \wedge \text{InLoop}' \wedge d > 0) \Rightarrow ((\vert[\mathbf{Y}\,\psi_1]\vert^d)' \Leftrightarrow \vert[\psi_1]\vert^d)$	T
11	$\mathbf{Z}\,\psi_1$	$\neg le \wedge \neg(\neg \text{InLoop} \wedge \text{InLoop}' \wedge d > 0) \Rightarrow ((\vert[\mathbf{Z}\,\psi_1]\vert^d)' \Leftrightarrow \vert[\psi_1]\vert^d)$	T
12	$\psi_1\,\mathbf{S}\,\psi_2$	$\neg le \wedge \neg(\neg \text{InLoop} \wedge \text{InLoop}' \wedge d > 0) \Rightarrow ((\vert[\psi_1\,\mathbf{S}\,\psi_2]\vert^d)' \Leftrightarrow (\vert[\psi_2]\vert^d)' \vee ((\vert[\psi_1]\vert^d)' \wedge \vert[\varphi]\vert^d))$	T
13	$\psi_1\,\mathbf{T}\,\psi_2$	$\neg le \wedge \neg(\neg \text{InLoop} \wedge \text{InLoop}' \wedge d > 0) \Rightarrow ((\vert[\psi_1\,\mathbf{T}\,\psi_2]\vert^d)' \Leftrightarrow (\vert[\psi_2]\vert^d)' \wedge ((\vert[\psi_1]\vert^d)' \vee \vert[\varphi]\vert^d))$	T

When the end of a loop iteration is reached, subsequent unrollings (other than the topmost) are linked by taking current state values from unrolling d and next state values from unrolling $d+1$. In the topmost unrolling current and next state values are taken from the same unrolling to ensure stabilisation of all variables. This case corresponds to the loop-back case in BMC.

line	φ	$0 \le d \le \delta(\varphi)$	a.t.
14	$\mathbf{X}\,\psi_1$	$le \Rightarrow (\vert[\mathbf{X}\,\psi_1]\vert^d \Leftrightarrow (\vert[\psi_1]\vert^{\min(d+1,\delta(\varphi))})')$	T
15	$\psi_1\,\mathbf{U}\,\psi_2$	$le \Rightarrow (\vert[\psi_1\,\mathbf{U}\,\psi_2]\vert^d \Leftrightarrow \vert[\psi_2]\vert^d \vee (\vert[\psi_1]\vert^d \wedge (\vert[\varphi]\vert^{\min(d+1,\delta(\varphi))})'))$	T
16	$\psi_1\,\mathbf{R}\,\psi_2$	$le \Rightarrow (\vert[\psi_1\,\mathbf{R}\,\psi_2]\vert^d \Leftrightarrow \vert[\psi_2]\vert^d \wedge (\vert[\psi_1]\vert^d \vee (\vert[\varphi]\vert^{\min(d+1,\delta(\varphi))})'))$	T
17	$\mathbf{Y}\,\psi_1$	$le \Rightarrow ((\vert[\mathbf{Y}\,\psi_1]\vert^{\min(d+1,\delta(\varphi))})' \Leftrightarrow \vert[\psi_1]\vert^d)$	T
18	$\mathbf{Z}\,\psi_1$	$le \Rightarrow ((\vert[\mathbf{Z}\,\psi_1]\vert^{\min(d+1,\delta(\varphi))})' \Leftrightarrow \vert[\psi_1]\vert^d)$	T
19	$\psi_1\,\mathbf{S}\,\psi_2$	$le \Rightarrow ((\vert[\psi_1\,\mathbf{S}\,\psi_2]\vert^{\min(d+1,\delta(\varphi))})' \Leftrightarrow (\vert[\psi_2]\vert^{\min(d+1,\delta(\varphi))})' \vee ((\vert[\psi_1]\vert^{\min(d+1,\delta(\varphi))})' \wedge \vert[\varphi]\vert^d))$	T
20	$\psi_1\,\mathbf{T}\,\psi_2$	$le \Rightarrow ((\vert[\psi_1\,\mathbf{T}\,\psi_2]\vert^{\min(d+1,\delta(\varphi))})' \Leftrightarrow (\vert[\psi_2]\vert^{\min(d+1,\delta(\varphi))})' \wedge ((\vert[\psi_1]\vert^{\min(d+1,\delta(\varphi))})' \vee \vert[\varphi]\vert^d))$	T

Variables representing past operators are initialised in unrolling 0 as usual:

line	φ		applies to
21	$\mathbf{Y}\,\psi_1$	$\vert[\mathbf{Y}\,\psi_1]\vert^0 \Leftrightarrow \bot$	I
22	$\mathbf{Z}\,\psi_1$	$\vert[\mathbf{Z}\,\psi_1]\vert^0 \Leftrightarrow \top$	I
23	$\psi_1\,\mathbf{S}\,\psi_2$	$\vert[\psi_1\,\mathbf{S}\,\psi_2]\vert^0 \Leftrightarrow \vert[\psi_2]\vert^0$	I
24	$\psi_1\,\mathbf{T}\,\psi_2$	$\vert[\psi_1\,\mathbf{T}\,\psi_2]\vert^0 \Leftrightarrow \vert[\psi_2]\vert^0$	I

Acceptance for **U**- and **R**-formulae is defined in their topmost unrolling but is otherwise standard:

line	φ		applies to
25	$\psi_1\,\mathbf{U}\,\psi_2$	$\neg\vert[\psi_1\,\mathbf{U}\,\psi_2]\vert^{\delta(\varphi)} \vee \vert[\psi_2]\vert^{\delta(\varphi)}$	F
26	$\psi_1\,\mathbf{R}\,\psi_2$	$\vert[\psi_1\,\mathbf{R}\,\psi_2]\vert^{\delta(\varphi)} \vee \neg\vert[\psi_2]\vert^{\delta(\varphi)}$	F

Finally, we add $|[\psi]|^0$ as an initial state constraint to ensure the desired semantics and $\{le\}$ as acceptance set to guarantee that ultimately all unrollings are linked. The labelling is defined as $L(s) = \{p \in AP(\psi) \mid |[p]|^0 \in V \wedge |[p]|^0(s) = \top\}$ where $AP(\psi)$ is the set of atomic propositions occurring in ψ.

In the following we prove that B^ψ accepts the desired language and is tight.

Lemma 5.4. $\mathrm{Lang}(B^\psi) = \{\alpha \mid \alpha \models \psi\}$

Proof. Let $\breve{B}^\psi$ be defined as B^ψ without the initial state constraint $|[\psi]|^0$.

(Correctness) We show that on every initialised fair path ρ in $\breve{B}^\psi$ the values of $|[\varphi]|^{d_i}(\rho_i)$ represent the validity of the subformula φ at time point i, where d_i is either the number of *le*'s seen up to time point $i-1$ or $\delta(\varphi)$, whichever is smaller. Formally, let ρ be an initialised fair path with $L(\rho) = \alpha$ in $\breve{B}^\psi$. For each time point i in ρ, let $d_i = \min(|\{j \mid (j \leq i-1) \wedge (le(\rho_j) \Leftrightarrow \top)\}|, \delta(\varphi))$.[12] The initial, invariant, transition, and fairness constraints on $|[\varphi]|^{d_i}$ are identical to the constraints that a Büchi automaton based on [KPR98] imposes on its state variables representing the corresponding subformula. Hence, $\alpha^i \models \varphi \Leftrightarrow |[\varphi]|^{d_i}(\rho_i)$.

(Completeness) We show that there is an initialised fair path ρ in $\breve{B}^\psi$ with $L(\rho) = \alpha$ for each word α. Choose a set of indices $U = \{i_0, i_1, \ldots\}$ (for "up") such that $le(\rho_i) \Leftrightarrow i \in U$. Further, choose $l \leq i_0$ and set $\mathrm{InLoop}(\rho_j) \Leftrightarrow j \geq l$. We inductively construct a valuation for $|[\varphi]|^d(\rho_i)$ for each subformula φ of ψ, $d \leq \delta(\varphi)$, and $i \geq 0$.

- If φ is an atomic proposition p, set $|[p]|^0(\rho_i) \Leftrightarrow (\alpha^i \models p)$.
- If the top level operator of φ is Boolean, the valuation follows directly from the semantics of the operator.
- For $\mathbf{X}$, each $|[\mathbf{X}\,\psi_1]|^d(\rho_i)$ appears at most once in $\mathbf{X}$'s defining constraint (line 7).
- $\varphi = \mathbf{Y}\,\psi_1$ is similar. Note that $\delta(\psi_1) = \delta(\varphi)-1$. Therefore, $|[\mathbf{Y}\,\psi_1]|^{\delta(\varphi)'} \Leftrightarrow |[\psi_1]|^{\delta(\varphi)}$ and $|[\mathbf{Y}\,\psi_1]|^{\delta(\varphi)'} \Leftrightarrow |[\psi_1]|^{\delta(\varphi)-1}$ are equivalent. $|[\mathbf{Y}\,\psi_1]|^d(\rho_i)$ is unconstrained if $d = 0$ and $i-1 \in U$ as well as if $d \geq 1$ and $i = l$.
- For $\varphi = \psi_1 \mathbf{U}\, \psi_2$, start with the topmost unrolling $\delta(\varphi)$. If $|[\psi_2]|^{\delta(\psi_2)}$ remains false from some i_d on, assign $\forall i \geq i_d\ .\ |[\varphi]|^{\delta(\varphi)}(\rho_i) \Leftrightarrow \bot$. Now work towards decreasing i from each i_n with $|[\psi_2]|^{\delta(\psi_2)}(i_n) \Leftrightarrow \top$, using line 8 in the definition of T for $\mathbf{U}$. Continue with unrolling $\delta(\varphi)-1$. Start at each $i \in U$ by obtaining $|[\varphi]|^{\delta(\varphi)-1}(\rho_i)$ from the previously assigned $|[\varphi]|^{\delta(\varphi)}(\rho_{i+1})$ via line 15. Then work towards decreasing i again using line 8 in the definition of T until $|[\varphi]|^{\delta(\varphi)-1}$ is assigned for all ρ_i. This is repeated in decreasing order for each unrolling $0 \leq d < \delta(\varphi)-1$.
- For $\mathbf{S}$, start with $|[\varphi]|^0(\rho_0)$ and proceed towards increasing i, also increasing d when $i \in U$ (lines 12, 19 in the definition of T for $\mathbf{S}$). When $d = \delta(\varphi)$ is reached, assign $|[\varphi]|^{\delta(\varphi)}(\rho_i)$ for all i using line 12 in the definition of T. Then, similar to $\mathbf{U}$, work towards decreasing d and i from each $i \in U$.
- $\mathbf{Z}$, $\mathbf{R}$, and $\mathbf{T}$ are as their duals.

For state variables on the stem with $d > 0$ any assignment satisfying the constraints in the definition of B^ψ can be chosen. It is easy to verify that such assignment always exists. Fairness follows from the definition of U, l, and the valuation chosen for $\mathbf{U}$ and $\mathbf{R}$.

The claim is now immediate by the definition of I. □

[12] In Fig. 5e this corresponds to the thick sequence of transitions starting in unrolling 0 at time point 0, jumping to unrolling 1 between time points 5 and 6, and finally reaching unrolling 2 at time point 10.

Lemma 5.5. *B^ψ is tight.*

Proof. We show inductively that the valuations of the variables $|[\varphi]|^d(\rho_i)$ can be chosen such that the valuation at a given relative index in a loop iteration is the same for each iteration in an unrolling d. Formally, let $\alpha = \beta\gamma^\omega$ with $\alpha \models \psi$. There exists a run ρ on α such that for all subformulas φ of ψ

$$\forall d \leq \delta(\varphi) \;.\; \forall i_1, i_2 \geq |\beta| \;.\; ((\exists k \geq 0 \;.\; i_2 - i_1 = k|\gamma|) \Rightarrow (|[\varphi]|^d(\rho_{i_1}) \Leftrightarrow |[\varphi]|^d(\rho_{i_2})))$$

Atomic propositions, Boolean connectives, and $\mathbf{X}$ are clear. $\mathbf{Y}$ is also easy, we only have to assign the appropriate value from other iterations when $|[\varphi]|^d(i)$ is unconstrained. For $\varphi = \psi_1 \,\mathbf{U}\, \psi_2$, by the induction hypothesis, $|[\psi_2]|^{\delta(\psi_2)}$ is either always false (in which case we assign $|[\varphi]|^{\delta(\varphi)}(\rho_i)$ to false according to the proof of Lemma 5.4) or becomes true at the same time in each loop iteration. Hence, the claim holds for unrolling $\delta(\varphi)$. From there we can proceed to lower unrollings in the same manner as in the proof of Lemma 5.4. For $\mathbf{S}$ we follow the order of assignments from the proof of Lemma 5.4. By induction, the claim holds for unrolling $\delta(\varphi)$. From there, we proceed towards decreasing i and d. We use, by induction, the same valuations of subformulas and the same equations (though in reverse direction) as we used to get from $|[\varphi]|^0(\rho_0)$ to unrolling $\delta(\varphi)$. $\mathbf{Z}$, $\mathbf{R}$, and $\mathbf{T}$ are as their duals. □

Theorem 5.6. *Let ψ be a PLTL formula, let B^ψ be defined as above. Then,* $\text{Lang}(B^\psi) = \{\alpha \mid \alpha \models \psi\}$ *and B^ψ is tight.*

Proof. By Lemma 5.4 and 5.5. □

As an optimisation, state variables representing atomic propositions, Boolean operators, and values of subformulas φ at unrollings $d > \delta(\varphi)$ can be replaced with macros. If the automaton is used with the liveness-to-safety transformation (with appropriate changes to shift l_s and InLoop back one state), InLoop can be taken directly from the transformation and le can be defined as LoopClosed$'$.

6. Incremental SAT and BMC

We now present an incremental eventuality encoding for PLTL (see Sect. 5.1 for the non-incremental version). The encoding has been first published in [HJL05] and is based on an earlier PLTL fixpoint evaluation encoding published in [LBHJ05].

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, learned clauses (see e.g., [ZMMM01]) 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 [Str01] and Whittemore et al. [WKS01] 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 [ES03] consider incremental BMC combined with the inductive scheme presented in [SSS00]. 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 for learned clauses between two sequential problem instances. Jin and Somenzi [JS05] present efficient ways of filtering learned clauses when creating the next problem instance. In [BB04] 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 (for past formulas) encoding of [BC03].

The incremental 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 [LBHJ05].

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 values of k.

Keeping the number of k-dependent constraints small is achieved largely by the introduction of *proxy states*, which serve as placeholders for the endpoint of a path. The disentanglement of the constraints at index k from the fixpoint encoding to the eventuality encoding by introducing formula variables also for index $k+1$ can be seen as a first step in that direction. This is the reason we chose the eventuality encoding of Sect. 5.1 as the base of our incremental encoding. Below only the differences needed to obtain incrementality are given. All of the non-modified parts of the encoding are k-invariant.

The loop constraints $|[LoopConstraints]|_k$ are modified to (changes are shown in blue boxes):

Base	l_0	$\Leftrightarrow$	$\bot$
	InLoop_0	$\Leftrightarrow$	$\bot$
$k-$invariant	l_i	$\Rightarrow$	$(s_{i-1} = s_{\boxed{E}})$
$1 \leq i \leq k$	InLoop_i	$\Leftrightarrow$	$\mathrm{InLoop}_{i-1} \vee l_i,$
	InLoop_{i-1}	$\Rightarrow$	$\neg l_i$
$k-$dependent	LoopExists	$\Leftrightarrow$	InLoop_k
	$\boxed{s_E \quad = \quad s_k}$		

Many k-dependent constraints of the non-incremental encoding of Sect. 5.1 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 k of the path. In the k-dependent part the proxy state s_E is constrained to be equivalent to s_k. The constraint defining the variable LoopExists is k-dependent as it is defined in terms of InLoop_k.

The $|[LastStateFormula]|_k$ constraints are modified to (changes are shown in blue boxes):

	$0 \le d \le \delta(\varphi)$
Base	$\neg \mathit{LoopExists} \Rightarrow \left(\vert[\varphi]\vert_{\boxed{L}}^{d} \Leftrightarrow \bot \right)$
$k-$invariant, $1 \le i \le k$	$l_i \Rightarrow \left(\vert[\varphi]\vert_{\boxed{L}}^{d} \Leftrightarrow \vert[\varphi]\vert_{i}^{d} \right)$
$k-$dependent	$\boxed{\vert[\varphi]\vert_{E}^{d} \Leftrightarrow \vert[\varphi]\vert_{k}^{d}}$
	$\boxed{\vert[\varphi]\vert_{k+1}^{d} \Leftrightarrow \vert[\varphi]\vert_{L}^{min(d+1,\delta(\varphi))}}$

The proxy state s_E has the corresponding new formula variables $|[\varphi]|_E^d$ which have been introduced to make the encodings for the past formulas k-invariant. For the future formulas another proxy state with index L has been introduced. This loop proxy state introduces new formula variables $|[\varphi]|_L^d$. All the formulas at the proxy states are bound to their corresponding states at the same time point, implementing jumping from one unrolling to another as shown in Fig. 4.

We need to extend the first rule of the PLTL encoding also to indices E and L, for each subformula $\varphi \in cl(\psi)$:

$$|[\varphi]|_E^d = |[\varphi]|_E^{\delta(\varphi)}, \text{ when } d > \delta(\varphi); \text{ and}$$
$$|[\varphi]|_L^d = |[\varphi]|_L^{\delta(\varphi)}, \text{ when } d > \delta(\varphi).$$

The auxiliary formula encoding is modified to (as before, changes are shown in blue boxes):

	φ	
Base	$\psi_1 \mathbf{U} \psi_2$	$\mathit{LoopExists} \Rightarrow \left(\vert[\psi_1 \mathbf{U} \psi_2]\vert_{\boxed{E}}^{\delta(\varphi)} \Rightarrow \langle\langle \mathbf{F} \psi_2 \rangle\rangle_{\boxed{E}}^{\delta(\psi_2)} \right)$
	$\psi_1 \mathbf{R} \psi_2$	$\mathit{LoopExists} \Rightarrow \left(\vert[\psi_1 \mathbf{R} \psi_2]\vert_{\boxed{E}}^{\delta(\varphi)} \Leftarrow \langle\langle \mathbf{G} \psi_2 \rangle\rangle_{\boxed{E}}^{\delta(\psi_2)} \right)$
	$\psi_1 \mathbf{U} \psi_2$	$\langle\langle \mathbf{F} \psi_2 \rangle\rangle_{0}^{\delta(\psi_2)} \Leftrightarrow \bot$
	$\psi_1 \mathbf{R} \psi_2$	$\langle\langle \mathbf{G} \psi_2 \rangle\rangle_{0}^{\delta(\psi_2)} \Leftrightarrow \top$
$k-$invariant	$\psi_1 \mathbf{U} \psi_2$	$\langle\langle \mathbf{F} \psi_2 \rangle\rangle_{i}^{\delta(\psi_2)} \Leftrightarrow \langle\langle \mathbf{F} \psi_2 \rangle\rangle_{i-1}^{\delta(\psi_2)} \vee \left(\mathrm{InLoop}_i \wedge \vert[\psi_2]\vert_{i}^{\delta(\psi_2)} \right)$
$1 \le i \le k$	$\psi_1 \mathbf{R} \psi_2$	$\langle\langle \mathbf{G} \psi_2 \rangle\rangle_{i}^{\delta(\psi_2)} \Leftrightarrow \langle\langle \mathbf{G} \psi_2 \rangle\rangle_{i-1}^{\delta(\psi_2)} \wedge \left(\neg\mathrm{InLoop}_i \vee \vert[\psi_2]\vert_{i}^{\delta(\psi_2)} \right)$
$k-$dependent	$\psi_1 \mathbf{U} \psi_2$	$\boxed{\langle\langle \mathbf{F} \psi_2 \rangle\rangle_{E}^{\delta(\psi_2)} \Leftrightarrow \langle\langle \mathbf{F} \psi_2 \rangle\rangle_{k}^{\delta(\psi_2)}}$
	$\psi_1 \mathbf{R} \psi_2$	$\boxed{\langle\langle \mathbf{G} \psi_2 \rangle\rangle_{E}^{\delta(\psi_2)} \Leftrightarrow \langle\langle \mathbf{G} \psi_2 \rangle\rangle_{k}^{\delta(\psi_2)}}$

Basically all references to the index k have been removed in the k-invariant parts by references to E. The new k-dependent constraints constrain the auxiliary encodings at E to get their values from the state at the current bound k.

We also have to modify the encoding of past formulas slightly, as they explicitly mention the bound k used. The change is to replace the index k with the proxy end index E, and after this the encoding becomes k-invariant. The case $d = 0$ does not have to be changed and is therefore omitted. The indexing changes required are again shown in blue boxes. The table below includes also the (optional) stabilisation forcing constraints.

φ	$1 \leq i \leq k, 1 \leq d \leq \delta(\varphi)$
$\psi_1 \mathbf{S}\, \psi_2$	$\vert[\psi_1 \mathbf{S}\, \psi_2]\vert_i^d \Leftrightarrow \vert[\psi_2]\vert_i^d \vee \left(\vert[\psi_1]\vert_i^d \wedge \left(\left(l_i \wedge \vert[\varphi]\vert_{\boxed{E}}^{d-1}\right) \vee \left(\neg l_i \wedge \vert[\varphi]\vert_{i-1}^d\right)\right)\right)$
$\psi_1 \mathbf{T}\, \psi_2$	$\vert[\psi_1 \mathbf{T}\, \psi_2]\vert_i^d \Leftrightarrow \vert[\psi_2]\vert_i^d \wedge \left(\vert[\psi_1]\vert_i^d \vee \left(\left(l_i \wedge \vert[\varphi]\vert_{\boxed{E}}^{d-1}\right) \vee \left(\neg l_i \wedge \vert[\varphi]\vert_{i-1}^d\right)\right)\right)$
$\mathbf{Y}\, \psi_1$	$\vert[\mathbf{Y}\, \psi_1]\vert_i^d \Leftrightarrow \left(l_i \wedge \vert[\psi_1]\vert_{\boxed{E}}^{d-1}\right) \vee \left(\neg l_i \wedge \vert[\psi_1]\vert_{i-1}^d\right)$
$\mathbf{Z}\, \psi_1$	$\vert[\mathbf{Z}\, \psi_1]\vert_i^d \Leftrightarrow \left(l_i \wedge \vert[\psi_1]\vert_{\boxed{E}}^{d-1}\right) \vee \left(\neg l_i \wedge \vert[\psi_1]\vert_{i-1}^d\right)$
$\psi_1 \mathbf{S}\, \psi_2$	$\vert[\psi_1 \mathbf{S}\, \psi_2]\vert_i^{\delta(\varphi)} \Leftrightarrow \vert[\psi_2]\vert_i^{\delta(\varphi)} \vee \left(\vert[\psi_1]\vert_i^{\delta(\varphi)} \wedge \left(\left(l_i \wedge \vert[\varphi]\vert_{\boxed{E}}^{\delta(\varphi)}\right) \vee \left(\neg l_i \wedge \vert[\varphi]\vert_{i-1}^{\delta(\varphi)}\right)\right)\right)$
$\psi_1 \mathbf{T}\, \psi_2$	$\vert[\psi_1 \mathbf{T}\, \psi_2]\vert_i^{\delta(\varphi)} \Leftrightarrow \vert[\psi_2]\vert_i^{\delta(\varphi)} \wedge \left(\vert[\psi_1]\vert_i^{\delta(\varphi)} \vee \left(\left(l_i \wedge \vert[\varphi]\vert_{\boxed{E}}^{\delta(\varphi)}\right) \vee \left(\neg l_i \wedge \vert[\varphi]\vert_{i-1}^{\delta(\varphi)}\right)\right)\right)$
$\mathbf{Y}\, \psi_1$	$\vert[\mathbf{Y}\, \psi_1]\vert_i^{\delta(\varphi)} \Leftrightarrow \left(l_i \wedge \vert[\psi_1]\vert_{\boxed{E}}^{\delta(\varphi)}\right) \vee \left(\neg l_i \wedge \vert[\psi_1]\vert_{i-1}^{\delta(\varphi)}\right)$
$\mathbf{Z}\, \psi_1$	$\vert[\mathbf{Z}\, \psi_1]\vert_i^{\delta(\varphi)} \Leftrightarrow \left(l_i \wedge \vert[\psi_1]\vert_{\boxed{E}}^{\delta(\varphi)}\right) \vee \left(\neg l_i \wedge \vert[\psi_1]\vert_{i-1}^{\delta(\varphi)}\right)$

Combining the tables above we get the full incremental PLTL encoding $|[\mathit{IncPLTL}]|_k$ for ψ. Given a Kripke structure M, a PLTL formula ψ, and a bound k, the *incremental PLTL eventuality encoding* as a propositional formula is given by:

$$|[M, \psi, k]| = |[M]|_k \wedge |[\mathit{LoopConstraints}]|_k \wedge |[\mathit{LastStateFormula}]|_k \wedge |[\mathit{IncPLTL}]|_k \wedge |[\psi]|_0^0.$$

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

Theorem 6.1. *Given a Kripke structure M and a PLTL formula ψ, M has an initialised path π such that $\pi \models \psi$ iff there exists a $k \in \mathbb{N}$ such that the incremental PLTL eventuality encoding $|[M, \psi, k]|$ is satisfiable. In particular, if $\pi \models_k \psi$ then the incremental PLTL eventuality encoding $|[M, \psi, k]|$ is satisfiable.*

Proof. Note, that the fact that the encoding is used incrementally does not influence correctness of the claim. Hence, we show that the incremental PLTL eventuality encoding is satisfiable iff the (non-incremental) PLTL eventuality encoding presented in Sect. 5 is satisfiable. Correctness then follows from Thm. 5.3.

It's not hard to verify that, essentially by applying substitutions to the proxy variables, the incremental encoding can be transformed into the non-incremental version plus the following set of constraints

$$
\begin{aligned}
& s_E = s_k \\
& \neg\text{LoopExists} \Rightarrow (|[\varphi]|_L^0 \Leftrightarrow \bot) \\
& \forall 1 \leq i \leq k : l_i \Rightarrow (|[\varphi]|_L^0 \Leftrightarrow |[\varphi]|_i^0) \\
& \forall 0 \leq d \leq \delta(\varphi) : |[\varphi]|_E^d \Leftrightarrow |[\varphi]|_k^d \\
& \forall 0 \leq d \leq \delta(\varphi) : |[\varphi]|_{k+1}^d \Leftrightarrow |[\varphi]|_L^{\min(d+1,\delta(\varphi))} \\
& |[\varphi]|_E^d \Leftrightarrow |[\varphi]|_E^{\delta(\varphi)} \text{ if } d > \delta(\varphi) \\
& |[\varphi]|_L^d \Leftrightarrow |[\varphi]|_L^{\delta(\varphi)} \text{ if } d > \delta(\varphi) \\
& \langle\langle \mathbf{F}\, \psi_2 \rangle\rangle_E^{\delta(\psi_2)} \Leftrightarrow \langle\langle \mathbf{F}\, \psi_2 \rangle\rangle_k^{\delta(\psi_2)} \\
& \langle\langle \mathbf{G}\, \psi_2 \rangle\rangle_E^{\delta(\psi_2)} \Leftrightarrow \langle\langle \mathbf{G}\, \psi_2 \rangle\rangle_k^{\delta(\psi_2)}
\end{aligned}
$$

without changing the set of satisfying truth assignments. It is easy to see that with $l_i \Rightarrow (|[\varphi]|_{k+1}^{\delta(\varphi)-1} \Leftrightarrow |[\varphi]|_i^{\delta(\varphi)} \Leftrightarrow |[\varphi]|_{k+1}^{\delta(\varphi)})$ the set of constraints is a conflict-free assignment of the proxy variables. □

The incrementality of the encoding works as follows. The encoding $|[M, \psi, k+1]|$ for bound $k+1$ is obtained from the encoding $|[M, \psi, k]|$ for bound k. First, all the k-dependent rules, and everything learned from them by the SAT solver have to be dropped. After this the encoding must be extended by all the constraints needed for encoding the new time step $k+1$.

We have taken care to keep most of the encoding rules k-independent, and to make all of the k-dependent constraints as simple as possible (they are all just equivalences between two variables) in order to make the size of the k-dependent part as small as possible. This was made in order to make the overhead to a non-incremental version as small as possible. The experimental results of Sect. 8 and [HJL05] confirm that the incremental approach does lead to performance benefits.

7. Completeness: Proving Properties

In its basic form bounded model checking only finds counterexamples and does not prove systems to be correct. To prove that a system has no counterexamples for a given property with BMC, we must prove that no counterexample can be longer than a certain bound, the *completeness threshold*, and prove that there are no shorter counterexamples. The obvious upper bound for the completeness threshold is exponential in the number of state bits in the system. We could thus obtain a complete BMC procedure by always doing BMC until reaching this upper bound, but clearly such an approach is unacceptable and we actually want a procedure that will in many practical cases terminate with a much smaller bound. There are several approaches to making BMC complete in a more practical sense, i.e., which are able to prove properties by more precisely approximating the required completeness threshold.

A complete method for proving invariant properties is k-induction originally developed by Sheeran et al. [SSS00]. They give several different variants for proving invariant properties. The variant closest to our approach is the following: If the invariant holds in every state in each initialised path of length k, and there is no initialised loop-free path, which does not visit an initial state, of length $k+1$; then we can conclude that the invariant holds for the system. The longest initialised loop-free path in the state graph is called the *recurrence diameter*, which can be used as an upper bound for the completeness threshold when proving invariants. Clearly the number of reachable states of the system gives a worst case upper bound for the recurrence diameter. For a bound k a straightforward encoding of this loop-free path predicate is of the size $O(k^2)$. Kroening and Strichman [KS03] show that the size of this loop-free predicate can be optimised to $O(k \log^2 k)$ using sorting networks. They also suggest ways to leave out state bits from the loop-free predicate to improve efficiency while maintaining completeness. The benefits of having a smaller predicate are two-fold: a smaller predicate is easier to manage for the SAT solver and with fewer state variables we can prove properties at shallower depths because the system loops earlier.

It is now easy to see that by combining the Büchi automata-based BMC encoding of Sect. 5.2 for PLTL and the liveness to safety reduction of Sect. 4 with k-induction we get a complete BMC method. The method can also be made incremental as shown in [ES03].

In this section we show a more refined approach to completeness based on the incremental BMC encoding presented in Sect. 6. The approach has been first published in [HJL05]. It is based on similar ideas as [ES03] but due to the increased flexibility of the BMC encoding, like the ability to refer to arbitrary states in the run, it is able to avoid the doubling of the number of state bits as required by the liveness-to-safety transformation. This doubling would increase the size of the already large loop-free predicate needed by the approach. Our method also works in the forward direction only (we always have the initial state predicate present), unlike some other approaches to obtaining completeness such as [ES03, AS06].

Practical experience seems to indicate that already model checking general safety properties using induction is challenging [AFF$^+$05]. Simply synchronising a finite state automaton (FSA) representing a safety property with the system to model check safety properties from does not scale well, and forces model checkers to go deeper than the current capacity of SAT solvers. One reason is the non-determinism in the FSA representing the property [AFF$^+$05]. It seems that specifications using deterministic FSAs can be treated more efficiently [AEF$^+$05, Lat03]. Our BMC encodings follow this line of reasoning by trying to be as deterministic as possible.

Two papers that consider strengthening of induction without always doing deeper BMC queries, which is expensive, are [dMRS03, AFF$^+$05]. In [dMRS03] the inductive method of [SSS00] is generalised to an induction scheme based on simulations. Inductive invariants are automatically strengthened from failed induction proofs using a procedure based on existential quantification. Since existential quantification is resource intensive, a method for quantifying on demand is developed. Another approach is presented in [AFF$^+$05]. They develop a methodology for flexible manual strengthening of induction. The key idea is to make the induction scheme part of the specification to allow a high degree of control of the induction process. Counterexamples produced by the model checker aid the designer in choosing new invariants.

Finding a completeness threshold for general LTL properties has proven fairly challenging. Clarke et al. [CKOS05] 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 representing the negation of the property. Awedh and Somenzi [AS06] 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. Thus, they potentially use an exponential number of state bits in the size of the formula to represent the Büchi automaton. Additionally, our encoding is able to find counterexamples for full PLTL with smaller bounds than previous methods for LTL [CKOS04, AS04], as these papers employ a method for translating generalised Büchi automata to standard (non-generalised) Büchi automata in a way (called the counter method in [AS06]) which does not preserve the minimal length of counterexamples. Recently, the authors of [AS04] have refined their approach in [AS06] to also in effect use generalised Büchi automata directly (called the flag method in [AS06]).

A different approach to proving completeness is taken by McMillan [McM03]. He uses interpolants derived from unsatisfiability proofs of BMC counterexample queries to over-approximate symbolic reachability. The deeper the BMC query is, the more exact the over-approximation is. The method is complete and can be extended to LTL model checking through the liveness-to-safety transformation discussed in Sect. 4. Although the method can in many cases converge more quickly than the recurrence diameter, which is the relevant completeness threshold for most other methods, the unsatisfiability proofs can be of exponential size and cause a blow-up.

Suggested BMC Procedure for Completeness. The incremental encoding of Sect. 6 can easily be extended to also prove properties. The basic ideas used are similar to the variant of k-induction of [SSS00] discussed above. However, the approach of [SSS00] is restricted to proving invariants, while our approach can handle proving of all PLTL properties.

The procedure starts with bound $k = 0$. First we create a *completeness formula*, denoted by $\langle\langle M, \psi, k\rangle\rangle$, 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). The completeness formula $\langle\langle M, \psi, k\rangle\rangle$ we use consists of exactly the incremental translation $|[M, \psi, k]|$ of Sect. 6 with all k-dependent constraints removed. Because these constraints are a subset of the constraints $|[M, \psi, k']|$ for every $k' \geq k$, if $\langle\langle M, \psi, k\rangle\rangle$ is unsatisfiable, so will also $|[M, \psi, k']|$ be.

Now, similarly to the k-induction method, we want to conjunct the completeness formula $\langle\langle M, \psi, k\rangle\rangle$ with a *simple path* formula which is satisfied for only initialised loop-free paths. This formula is needed in order to guarantee termination of the procedure. However, we use a certain product automaton instead of the Kripke structure itself. The states of this product automaton at time point i consist of tuples of: (a) system state s_i, (b) a bit vector of values of all formula variables $|[\varphi]|_i^d$, denoted $|[s_\varphi]|_i$, (c) a bit vector of values of all auxiliary formula variables $\langle\langle\varphi\rangle\rangle_i^d$, denoted $\langle\langle s_\varphi\rangle\rangle_i$, and (d) value of the InLoop_i predicate. As an optimisation we disregard any differences in unrollings $d > 0$ between two indices where InLoop_i is false, as these bits are not constrained by the top-level formula, and thus are always satisfiable (these are the light nodes of Fig. 4). To do so, we use $|[s_\varphi]|_i^0$ to denote $|[s_\varphi]|_i$ restricted to the bits $|[\varphi]|_i^0$. The simple path formula we use is the following:

$$|[\mathit{SimplePath}]|_k \Leftrightarrow \bigwedge_{0\leq i<j\leq k} \begin{aligned}[t]&\Big(s_i \neq s_j \vee \mathrm{InLoop}_i \neq \mathrm{InLoop}_j \vee |[s_\varphi]|_i^0 \neq |[s_\varphi]|_j^0 \vee \\ &\Big(\mathrm{InLoop}_i \wedge \mathrm{InLoop}_j \wedge \Big(|[s_\varphi]|_i \neq |[s_\varphi]|_j \vee \langle\langle s_\varphi\rangle\rangle_i \neq \langle\langle s_\varphi\rangle\rangle_j\Big)\Big)\Big) .\end{aligned}$$

If at bound k the conjunction of the completeness $\langle\langle M, \psi, k\rangle\rangle$ and the simple path formula is unsatisfiable the model checked formula $\neg\psi$ holds in the system and the procedure can be terminated. Otherwise the *witness formula* $|[M, \psi, k]|$ is created (and optionally conjuncted with the simple path formula) and the result is satisfiable for bounded witnesses of length k to the formula ψ (see Thm. 6.1). If the witness formula is satisfiable, the model checked formula $\neg\psi$ does not hold, and the procedure can terminate. Otherwise, the procedure is repeated after incrementing k by one.

The $|[\mathit{SimplePath}]|_k$ constraint above is obviously quadratic in k. We could use the standard simple path constraint used in other works employing k-induction by slight modifications to the encoding, e.g., forcing the light nodes of Fig. 4 to $\perp$ in the encoding. This would enable, e.g., using the optimisations of [KS03].

The procedure above has been designed to be easily implemented using one incremental SAT solver only, and this is what our implementation does. The only place where constraints have to be dropped is moving from a witness formula $|[M, \psi, k]|$ for bound k to the completeness formula $\langle\langle M, \psi, k+1\rangle\rangle$ for bound $k+1$, at which point all k-dependent constraints of $|[M, \psi, k]|$ and everything learned from them by the SAT solver have to be dropped. We use implementation techniques similar to those of [ES03] to implement this.

We have the following result:

Theorem 7.1. *Given a Kripke structure M and a PLTL formula ψ, $M \models \psi$ iff for some $k \geq 0$: $\langle\langle M, \neg\psi, k\rangle\rangle \wedge |[SimplePath]|_k$ is unsatisfiable and $|[M, \neg\psi, i]| \wedge |[SimplePath]|_i$ is unsatisfiable for all $0 \leq i < k$.*

The proof requires the following Lemma:

Lemma 7.2. *Given a Kripke structure M and a PLTL formula ψ, if $|[M, \neg\psi, k]|$ is satisfiable for some k, there is $\tilde{k} \leq k$ such that $|[M, \neg\psi, \tilde{k}]| \wedge |[SimplePath]|_{\tilde{k}}$ is satisfiable.*

Proof. We are given that $|[M, \neg\psi, k]|$ is satisfiable. If $|[M, \neg\psi, k]| \wedge |[SimplePath]|_k$ is already satisfiable for k we are done. Otherwise, the proof strategy is to show that for some $\tilde{k} < k$ the encoding $|[M, \neg\psi, \tilde{k}]|$ is satisfiable, and repeating the process. By the finiteness of k, this process can only be repeated a limited number of times. The base case is proved by the fact that $|[SimplePath]|_0$ is an empty set of constraints, thus proving termination at some $\tilde{k}$ where $|[M, \neg\psi, \tilde{k}]| \wedge |[SimplePath]|_{\tilde{k}}$ is satisfiable.

Consider the induction step where $|[M, \neg\psi, k]|$ is satisfiable but $|[SimplePath]|_k$ is not satisfiable. Hence, there are $0 \leq i < j \leq k$ such that $s_i = s_j$, $\text{InLoop}_i \Leftrightarrow \text{InLoop}_j$ and either (a): $\text{InLoop}_i \wedge \text{InLoop}_j \wedge |[s_\varphi]|_i = |[s_\varphi]|_j \wedge \langle\langle s_\varphi\rangle\rangle_i = \langle\langle s_\varphi\rangle\rangle_j$, or (b): $\neg\text{InLoop}_i \wedge \neg\text{InLoop}_j \wedge |[s_\varphi]|_i^0 = |[s_\varphi]|_j^0$. In the following we show that also $|[M, \neg\psi, \tilde{k}]|$ is satisfiable for $\tilde{k} = k - j + i$, i.e., $\tilde{k} < k$. Intuitively, we construct a satisfying truth assignment by "cutting out" the part of the encoding between indices $i + 1$ and j (both inclusive) of the satisfying truth assignment of $|[M, \neg\psi, k]|$ and "pasting together" the remaining parts by reducing all variable indices to the right of the cut point by $j - i$, obtaining a satisfying truth assignment for $|[M, \neg\psi, \tilde{k}]|$.

An exception to the above rule are the formula variables with unrolling index $d > 0$ such that InLoop_i is false, i.e., the light nodes of Fig. 4. By similar reasoning as used in the proof of Thm. 5.3 their constraints can never lead to the unsatisfiability of the encoding, and they can therefore be ignored in constructing the (now actually partial) truth assignment below. In other words a satisfying truth assignment for them always exists, and will be fully determined by the partial truth assignment for all the other variables to be constructed below.

Note, that in the case a loop exists: either $i < j < l$ or $l \leq i < j$ as $\text{InLoop}_i \Leftrightarrow \text{InLoop}_j$. Hence, the loop start at index l is never cut out. For ease of notation we define a function f mapping indices from the new to the old assignment:

$$f(n) = \text{if } n \leq i \text{ then } n \text{ else } n + j - i$$

With that we define:

$$\begin{array}{lrcl}
\forall 0 \leq n \leq \tilde{k}: & \tilde{s}_n & = & s_{f(n)} \\
\forall 0 \leq n \leq \tilde{k}: & \tilde{l}_n & \Leftrightarrow & l_{f(n)} \\
\forall 0 \leq n \leq \tilde{k}: & \widetilde{\text{InLoop}}_n & \Leftrightarrow & \text{InLoop}_{f(n)} \\
& \widetilde{\text{LoopExists}} & \Leftrightarrow & \text{LoopExists} \\
& \tilde{s}_E & = & s_E
\end{array}$$

We start with the model constraints. Let π be an initialised path in M induced by a satisfying truth assignment of $|[M, \neg\psi, k]|$. Because $s_i = s_j$ the path $\tilde{\pi}$ constructed from π by cutting out indices $i + 1$ and j (both inclusive) is still an initialised path of M. Hence, the model constraints are satisfied.

For the loop constraints note first that $\tilde{s}_{\tilde{k}} = s_k$ both in the case $j < k$ and in the case $j = k$. Furthermore, a loop point is never cut out. Hence, if some l_l was true in the original assignment, there is $\tilde{l}$ such that $l_{\tilde{l}}$ is true in the new assignment. In this case we also have $\tilde{s}_{\tilde{l}-1} = s_{l-1}$. Thus by simple case analysis of the loop constraints we get that they are satisfiable also in $|[M, \neg\psi, \tilde{k}]|$.

What remains to be done is to prove that the formula encoding $\widetilde{|[\varphi]|_0^0}$ is still satisfiable in $|[M, \neg\psi, \tilde{k}]|$. We do this by analysing the structure of the encoding rules for temporal formulas. Below, in each case we consider the mapped pairs of indices i, d such that $d = 0$ or InLoop_i is true. For simplicity all indices below refer to the original encoding $|[M, \neg\psi, k]|$.

For all future formulas in $|[M, \neg\psi, k]|$ at index i the references to formula values at $i + 1$ have been replaced in the encoding $|[M, \neg\psi, \tilde{k}]|$ with references to formula variables at index $j + 1$ (note that potentially $j + 1 = k + 1$). Now because both the formula values at i and j are identical and the future formula constraints at i and j are identical modulo index changes, the constraints at i will still be satisfiable with the same truth assignment when all references to $i + 1$ have been replaced with references to $j + 1$.

For all past formulas in $|[M, \neg\psi, k]|$ at index $j + 1$ (at the loop index l, when $j = k$) the references to formula values at j have been replaced in the encoding $|[M, \neg\psi, \tilde{k}]|$ with references to formula variables at index i. Now because the formula values at i and j are identical, the constraints at $j + 1$ (at the loop index l, when $j = k$) will still be satisfiable with the same truth assignment when all references to j have been replaced with references to i.

For the auxiliary encoding all the constraints are also satisfied by replacing all references to index j in $|[M, \neg\psi, k]|$ with references to index i in $|[M, \neg\psi, \tilde{k}]|$. This is the case because $\langle\langle s_\varphi \rangle\rangle_i = \langle\langle s_\varphi \rangle\rangle_j$ holds in case (a) due to the simple path constraint $\langle\langle s_\varphi \rangle\rangle_i = \langle\langle s_\varphi \rangle\rangle_j$, and in case (b) because the encoding for auxiliary variables keeps them constant for all indices $0 \le i < j < l$.

Now combining all the cases above we were able to "cut out" a part of the encoding $|[M, \neg\psi, k]|$ while still retaining its satisfiability. Thus $|[M, \neg\psi, \tilde{k}]|$ will also be satisfiable. □

We can now continue with the proof of Thm. 7.1:

Proof. "⇒" We only deal with finite models M and finite formulas $\neg\psi$. $|[\mathit{SimplePath}]|_k$ must therefore become and remain unsatisfiable from some k onward. From correctness of the incremental PLTL eventuality encoding (Thm. 6.1) we have that $|[M, \neg\psi, i]|$ is unsatisfiable for all $i \ge 0$ if $M \models \psi$.

"⇐" Assume that $\langle\langle M, \neg\psi, k \rangle\rangle \wedge |[\mathit{SimplePath}]|_k$ is unsatisfiable for some $k \ge 0$ and $|[M, \neg\psi, i]| \wedge |[\mathit{SimplePath}]|_i$ is unsatisfiable for all $0 \le i < k$. As noted above, unsatisfiability of $\langle\langle M, \neg\psi, k \rangle\rangle$ implies unsatisfiability of $|[M, \neg\psi, k']|$ for all $k' \ge k$. Similarly, if $|[\mathit{SimplePath}]|_k$ is unsatisfiable, so is $|[\mathit{SimplePath}]|_{k'}$ for all $k' \ge k$. Hence, we have that $|[M, \neg\psi, i]| \wedge |[\mathit{SimplePath}]|_i$ is unsatisfiable for all $i \ge 0$. Using Thm. 6.1 together with Lemma 7.2 in the reverse direction we can conclude $M \models \psi$.

□

We could also increase the bound k by more than one at a time if the witness formula is not conjuncted with the simple path formula. The proof requires the fact that if $|[M, \neg\psi, k]|$ is satisfiable for some k, it is satisfiable for all $k' \ge k$.

Lemma 7.3. *Given a Kripke structure M and a PLTL formula ψ, if $|[M, \neg\psi, k]|$ is satisfiable for some k, then $|[M, \neg\psi, k']|$ is satisfiable for all $k' \geq k$.*

Proof. Assume $\pi = s_0 \dots s_k$ is a bounded witness for $\neg\psi$. We show below that π can be extended by one state so that the result is again a bounded witness for $\neg\psi$. By Thm. 6.1, $|[M, \neg\psi, k+1]|$ is then also satisfiable. Repeated application gives satisfiability of $|[M, \neg\psi, k']|$ for any $k' \geq k$.

Consider the no-loop case first. By definition of $\models_{\mathrm{nl}}$, π extended with an arbitrary successor of s_k, s_{k+1}, is also a bounded no-loop witness for $\neg\psi$. If π is a (k,l)-loop, we rewrite π into a $(k+1, l+1)$-loop by delaying the loop start by one state: $\pi' = s_0 \dots s_l s_{l+1} \dots s_k s_{k+1} = s_l$. Clearly, π' interpreted as $(k+1, l+1)$-loop represents the same infinite path as π interpreted as (k,l)-loop and, hence, also satisfies $\neg\psi$. ☐

Theorem 7.4. *Given a Kripke structure M and a PLTL formula ψ, $M \models \psi$ iff for some $k \geq 0$: $\langle\langle M, \neg\psi, k\rangle\rangle \wedge |[SimplePath]|_k$ is unsatisfiable and either $k = 0$ or $|[M, \neg\psi, k-1]|$ is unsatisfiable.*

Proof. The "$\Rightarrow$" direction is exactly as in the proof of Thm. 7.1. For "$\Leftarrow$" assume that for some $k \geq 0$ we have that $\langle\langle M, \neg\psi, k\rangle\rangle \wedge |[SimplePath]|_k$ is unsatisfiable and either $k = 0$ or $|[M, \neg\psi, k-1]|$ is unsatisfiable. In the case $k = 0$ the result follows directly from Thm. 7.1. Now consider the case $k > 0$. By Lemma 7.3, we have that $|[M, \neg\psi, k']|$ is unsatisfiable for all $0 \leq k' < k$. Therefore also obviously $|[M, \neg\psi, k']| \wedge |[SimplePath]|_{k'}$ is unsatisfiable for all $0 \leq k' < k$ and the result follows from Thm. 7.1. ☐

8. Experiments and Comparisons

In this section we experimentally evaluate and compare the approaches presented in this paper. The benchmarks, implementations, and scripts are available at

`http://www.tcs.hut.fi/Software/benchmarks/LMCS-2006`

8.1. Benchmark Instances.

We mostly use examples of nontrivial complexity. The majority are taken from the NuSMV distribution [CCG$^+$02], one is from the examples of the Rebeca tool [SMSdB04], and two are from previous work of the authors [SB03, LBHJ05]. Table 1 provides a brief description of the models. For "1394" and "dme" we use instances of different sizes (indicated by the numerical parameters). For "1394" a buggy variant is used as well (denoted "1394b").

Table 2 gives templates of the properties used. The first column states the name of the model. Columns 2–4 indicate names and truth of the properties. To save space we combine a property and its negated version in a single line. The negation of property "p" is later referred to as "¬p". Truth is indicated by "t" for true, "f" for false, "?" for unknown (if none of our approaches terminated successfully), and "–" for not used. Sometimes we make the resulting witnesses more interesting by requiring that the request of a request-response property holds infinitely often (marked "nv"). We also prefix a property with "**F**" to turn a safety property into a liveness property. For "1394" the first entry in column 3 refers to the correct, the second to the buggy version. The last two columns give past operator depth and the template of the property.

model	state-bits	description	source
1394{b}-[345]-[23]	97–197	IEEE 1394 FireWire tree identify protocol with 3–5 nodes and 2 or 3 ports per node	[SB03]
abp4	30	alternating bit protocol for 4 bits	[CCG$^+$02]
brp	45	bounded retransmission protocol	[CCG$^+$02]
counter	3	3-bit counter	[CCG$^+$02]
csmacd	126	MAC sublayer of CSMA/CD protocol	[SMSdB04]
dme[35]	54, 90	asynchronous distributed mutual exclusion circuit with 3 or 5 nodes	[CCG$^+$02]
mutex	5	mutual exclusion with 2 participants	[CCG$^+$02]
pci	64	PCI Bus protocol	[CCG$^+$02]
prod-cons	26	producer consumer	[CCG$^+$02]
production-cell	54	production cell control model	[CCG$^+$02]
bc57-sensors	78	reactor system model	[CCG$^+$02]
ring	3	3 inverters forming a cycle	[CCG$^+$02]
short	2	simple request handler	[CCG$^+$02]
srg5	8	5 bit shift register	[LBHJ05]

Table 1: Models used in the experiments

8.2. **Implementations.** Following the automata-theoretic approach to LTL [VW86], a model checking procedure consists of encoding the property and subsequent fair cycle detection. As a special case, the second step can be performed by applying the liveness-to-safety translation and doing invariant checking. Where available we use off-the-shelf model checking procedures that include all steps to evaluate a particular approach. We make the following exceptions to that rule. Our implementation of the liveness-to-safety translation has the encoding of the property included but needs to be complemented with an algorithm to check invariants. To determine whether the effort of a dedicated implementation of a BMC encoding with the corresponding opportunities for optimisation is worthwhile we also combine our BMC encodings with the liveness-to-safety translation and with separately generated Büchi automata (Fig. 7(e), (f)). Finally, when comparing a tight with a non-tight Büchi automaton in BDD-based symbolic model checking we invoke the conversion from LTL to a Büchi automaton externally for both variants to minimise the influence of different variable orders (Fig. 7(j)).

Encoding of PLTL properties for model checking has been widely researched (for references see Sect. 3.3). However, in symbolic model checking, the dominating encodings are still more or less close to a symbolic implementation of the tableau construction [LP85] in [BCM$^+$92, CGH97]. This shifts a potential exponential blow-up from generation of the Büchi automaton to the search for a fair cycle. All encodings presented in this paper fall in this category. The question whether optimised Büchi automata constructions actually yield better overall performance in symbolic model checking algorithms is still open: while actual search for a fair cycle seems to benefit from optimised Büchi automata, there are cases where those benefits are more than offset by generating the Büchi automaton [STV05, CRST06]. Note, finally, that we currently don't have a construction that yields an explicit Büchi automaton that is both, small and tight. In Fig. 7(f) below we evaluate whether there is any overhead in forming the product of the model and the Büchi automaton for the property

model	property	truth		$\delta(\psi)$	template
		p	$\neg p$		
1394{b} -[345]-[23]	1	t	f	0	$\mathbf{F}((p) \vee ((q \vee (r)))$
	2	t/f	–	4	$\mathbf{G}((\mathbf{O}((p) \wedge (\mathbf{O}((\neg(p)) \wedge (\mathbf{O}((p) \wedge (\mathbf{O}(\neg(p))))))))) \rightarrow (\mathbf{F}(\mathbf{G}(\mathbf{X}(\neg(p))))))$
	3	t/f	–	6	$\mathbf{G}((\mathbf{O}((p) \wedge (\mathbf{O}((\neg(p)) \wedge (\mathbf{O}((p) \wedge (\mathbf{O}((\neg(p)) \wedge (\mathbf{O}((p) \wedge (\mathbf{O}(\neg(p))))))))))))) \rightarrow (\mathbf{F}(\mathbf{G}(\mathbf{X}(\neg(p))))))$
	4	t/f	–	8	$\mathbf{G}((\mathbf{O}((p) \wedge (\mathbf{O}((\neg(p)) \wedge (\mathbf{O}((p) \wedge (\mathbf{O}((\neg(p)) \wedge (\mathbf{O}((p) \wedge (\mathbf{O}((\neg(p)) \wedge (\mathbf{O}((p) \wedge (\mathbf{O}(\neg(p))))))))))))))))) \rightarrow (\mathbf{F}(\mathbf{G}(\mathbf{X}(\neg(p))))))$
	5	t	f	0	$(\mathbf{G}(p)) \vee ((q)\ \mathbf{U}\ (\mathbf{G}((r) \vee (s))))$
abp4	0	f	t	2	$\mathbf{G}((p) \rightarrow (\mathbf{Y}(\mathbf{H}(q))))$
	1	t	–	0	$\mathbf{G}(\mathbf{F}(p))$
	2	f	–	0	$\mathbf{G}((p) \rightarrow (\mathbf{X}((\neg(p))\ \mathbf{U}\ ((q) \wedge (((\neg(r)) \wedge (s)) \vee ((r) \wedge (t)))))))$
	3	t	–	0	$\mathbf{G}((p) \rightarrow (\mathbf{X}(((p)\ \mathbf{U}\ (\neg(p)))\ \mathbf{U}\ ((q) \wedge (((\neg(r)) \wedge (s)) \vee ((r) \wedge (t)))))))$
brp	0	t	f	2	$\mathbf{F}(\mathbf{G}((p) \rightarrow (\mathbf{O}((q) \rightarrow (\mathbf{O}(r))))))$
	¬ 0, nv	f	–	2	$\neg((\mathbf{F}(\mathbf{G}((p) \rightarrow (\mathbf{O}((q) \rightarrow (\mathbf{O}(r))))))) \wedge ((\mathbf{G}(\mathbf{F}(p))) \wedge (\mathbf{G}(\mathbf{F}(q)))))$
	1	t	f	0	$(\mathbf{G}((p) \rightarrow ((\mathbf{X}((q) \vee ((r) \vee (s))))\ \mathbf{R}\ (p)))) \wedge ((\mathbf{G}((q) \rightarrow ((\mathbf{X}((t) \vee (s)))\ \mathbf{R}\ (q)))) \wedge ((\mathbf{G}((t) \rightarrow ((\mathbf{X}((p) \vee (s)))\ \mathbf{R}\ (t)))) \wedge ((\mathbf{G}((q) \rightarrow ((\mathbf{X}((p) \vee (s)))\ \mathbf{R}\ (q)))) \wedge (\mathbf{G}((s) \rightarrow ((\mathbf{X}(p))\ \mathbf{R}\ (s)))))))$
counter	0	t	f	0	$\mathbf{F}(\mathbf{G}(p))$
csmacd	0	f	f	0	$\mathbf{G}((p) \rightarrow (\mathbf{F}(q)))$
	1	?	f	0	$(p) \wedge ((\mathbf{F}(q)) \rightarrow ((((((r)\ \mathbf{U}\ (s))\ \mathbf{U}\ (t))\ \mathbf{U}\ (u))\ \mathbf{U}\ (v))\ \mathbf{U}\ (q)))$
dme[35]	0	f	f	2	$\mathbf{G}((p) \rightarrow ((p)\ \mathbf{T}\ ((\neg(p))\ \mathbf{T}\ (\neg(q)))))$
	¬ 0, nv	f	–	2	$\neg((\mathbf{G}((p) \rightarrow ((p)\ \mathbf{T}\ ((\neg(p))\ \mathbf{T}\ (\neg(q)))))) \wedge (\mathbf{G}(\mathbf{F}(p))))$
	1	t	f	0	$\mathbf{G}(((p) \wedge (\mathbf{X}(\neg(p)))) \rightarrow (\mathbf{X}((\mathbf{G}(\neg(p))) \vee (((\neg(p))\ \mathbf{U}\ (q))\ \mathbf{U}\ (r)))))$
mutex	0	t	f	0	$\mathbf{G}((p) \rightarrow (\mathbf{F}(q)))$
pci	0	f	f	4	$\mathbf{G}((p) \rightarrow (\mathbf{G}(((q) \wedge (\mathbf{Y}((r) \wedge (\mathbf{O}((s) \wedge (\mathbf{O}((t) \wedge (\mathbf{O}(u))))))))) \rightarrow (\mathbf{O}((v) \wedge (\mathbf{O}((w) \wedge (\neg(\mathbf{O}(x))))))))))$
	F 0	f	–	4	$\mathbf{F}(\mathbf{G}((p) \rightarrow (\mathbf{G}(((q) \wedge (\mathbf{Y}((r) \wedge (\mathbf{O}((s) \wedge (\mathbf{O}((t) \wedge (\mathbf{O}(u))))))))) \rightarrow (\mathbf{O}((v) \wedge (\mathbf{O}((w) \wedge (\neg(\mathbf{O}(x)))))))))))$
	1	?	f	0	$(((((\mathbf{G}((o) \rightarrow ((o)\ \mathbf{U}\ (p)))) \wedge (\mathbf{G}((q) \rightarrow ((q)\ \mathbf{U}\ ((r) \vee (o)))))) \wedge (\mathbf{G}((s) \rightarrow ((s)\ \mathbf{U}\ ((t) \vee ((q) \vee (o))))))) \wedge (\mathbf{G}((u) \rightarrow ((u)\ \mathbf{U}\ ((v) \vee ((s) \vee ((q) \vee (o)))))))) \wedge (\mathbf{G}((w) \rightarrow ((w)\ \mathbf{U}\ ((x) \vee ((w) \vee ((u) \vee ((q) \vee (o))))))))) \wedge (\mathbf{G}((y) \rightarrow ((y)\ \mathbf{U}\ ((z) \vee ((y) \vee ((w) \vee ((u) \vee ((q) \vee (o))))))))))$
prod-cons	0	f	f	1	$((\mathbf{G}(\neg(p))) \wedge (\mathbf{G}(\mathbf{F}((q) \wedge ((q)\ \mathbf{S}\ (r)))))) \wedge (\mathbf{G}(\mathbf{F}(((q) \wedge ((q)\ \mathbf{S}\ (r))) \rightarrow ((s)\ \mathbf{S}\ (t)))))$
	1	t	–	4	$\mathbf{G}((p) \rightarrow ((p)\ \mathbf{S}\ ((q)\ \mathbf{S}\ ((r)\ \mathbf{S}\ ((s)\ \mathbf{S}\ (t))))))$
	¬ 1, nv	f	–	4	$\neg((\mathbf{G}((p) \rightarrow ((p)\ \mathbf{S}\ ((q)\ \mathbf{S}\ ((r)\ \mathbf{S}\ ((s)\ \mathbf{S}\ (t))))))) \wedge (\mathbf{G}(\mathbf{F}(p))))$
	2	f	–	0	$\mathbf{G}((p) \rightarrow (\mathbf{F}(((q) \wedge (r)) \wedge (s))))$
	3	f	–	0	$\mathbf{G}((p) \rightarrow (\mathbf{F}(q)))$
	4	t	–	0	$\mathbf{G}((p) \rightarrow (\mathbf{F}(q)))$
	5	t	f	0	$(\mathbf{X}(((\mathbf{X}(((\mathbf{X}(p))\ \mathbf{R}\ (q)) \wedge (r)))\ \mathbf{R}\ (s)) \wedge (t)))\ \mathbf{R}\ (u)$
production-cell	0	t	f	6	$\mathbf{G}(\mathbf{F}(((p) \vee (q)) \wedge (\mathbf{O}((r) \wedge (\mathbf{O}(((s) \vee (t)) \wedge (\mathbf{O}((u) \wedge (\mathbf{O}(((s) \vee (t)) \wedge (\mathbf{O}(((v) \vee (w)) \wedge (\mathbf{O}(x))))))))))))))$
	1	t	f	12	$\mathbf{G}(\mathbf{F}(((p) \vee (q)) \wedge (\mathbf{Y}(\mathbf{O}((r) \wedge (\mathbf{Y}(\mathbf{O}(((s) \vee (t)) \wedge (\mathbf{Y}(\mathbf{O}((u) \wedge (\mathbf{Y}(\mathbf{O}(((s) \vee (t)) \wedge (\mathbf{Y}(\mathbf{O}(((v) \vee (w)) \wedge (\mathbf{Y}(\mathbf{O}(x))))))))))))))))))))$
	2	t	f	10	$\mathbf{G}(\mathbf{F}(((\neg(p)) \vee (\neg(q))) \wedge (\mathbf{O}((\neg(r)) \wedge (\mathbf{Y}(\mathbf{O}(((\neg(s)) \vee (\neg(t))) \wedge (\mathbf{O}((\neg(u)) \wedge (\mathbf{Y}(\mathbf{O}(((\neg(s)) \vee (\neg(t))) \wedge (\mathbf{Y}(\mathbf{O}(((\neg(v)) \vee (\neg(w))) \wedge (\mathbf{Y}(\mathbf{O}(x))))))))))))))))))$
	3	t	f	0	$((((((((((((1)\ \mathbf{U}\ ((e) \wedge (\neg(f))))\ \mathbf{U}\ ((e) \wedge (f)))\ \mathbf{U}\ ((g) \wedge (h \wedge ((i) \wedge (j)))))\ \mathbf{U}\ ((k) \wedge ((l) \wedge ((i) \wedge (j)))))\ \mathbf{U}\ ((k) \wedge ((l) \wedge (m))))\ \mathbf{U}\ ((n) \wedge ((l) \wedge (o))))\ \mathbf{U}\ ((p) \wedge (q)))\ \mathbf{U}\ ((r) \wedge (q)))\ \mathbf{U}\ ((s) \wedge (q)))\ \mathbf{U}\ ((t) \wedge ((u) \wedge (v))))\ \mathbf{U}\ ((w) \wedge ((u) \wedge (x))))\ \mathbf{U}\ ((y) \wedge (\neg(z))))\ \mathbf{U}\ ((y) \wedge (z))$
	4	t	f	0	$((((((((((((((((1)\ \mathbf{U}\ ((a) \wedge (\neg(b))))\ \mathbf{U}\ ((a) \wedge (b)))\ \mathbf{U}\ ((c) \wedge ((d) \wedge ((e) \wedge (f)))))\ \mathbf{U}\ ((g) \wedge ((h) \wedge ((e) \wedge (f)))))\ \mathbf{U}\ ((i) \wedge ((j) \wedge (k))))\ \mathbf{U}\ ((l) \wedge ((j) \wedge (m))))\ \mathbf{U}\ ((n) \wedge (o)))\ \mathbf{U}\ ((p) \wedge (o)))\ \mathbf{U}\ ((q) \wedge (o)))\ \mathbf{U}\ ((r) \wedge ((s) \wedge (t))))\ \mathbf{U}\ ((u) \wedge ((s) \wedge (v))))\ \mathbf{U}\ ((w) \wedge (\neg(x))))\ \mathbf{U}\ ((w) \wedge (x)))\ \mathbf{U}\ ((y) \wedge ((z) \wedge ((aa) \wedge ((ab) \wedge ((ac) \wedge (ad)))))))\ \mathbf{U}\ ((ae) \wedge ((af) \wedge (ag))))\ \mathbf{U}\ ((a) \wedge (\neg(b)))$
bc57	0	t	f	2	$\mathbf{G}(\mathbf{F}((p) \wedge (\mathbf{O}((q) \wedge (\mathbf{F}((r) \wedge (\mathbf{O}(s))))))))$
	1	t	f	0	$(((((\mathbf{G}((a) \rightarrow (((b) \wedge ((c) \wedge (d)))\ \mathbf{R}\ ((e) \wedge (f))))) \wedge (\mathbf{G}((g) \rightarrow (((b) \wedge ((h) \wedge (i)))\ \mathbf{R}\ ((j) \wedge (k)))))) \wedge (\mathbf{G}((l) \rightarrow (((b) \wedge ((m) \wedge (n)))\ \mathbf{R}\ ((o) \wedge (p)))))) \wedge (\mathbf{G}((q) \rightarrow (((b) \wedge ((r) \wedge (s)))\ \mathbf{R}\ ((t) \wedge (u)))))) \wedge (\mathbf{G}((v) \rightarrow (((b) \wedge ((w) \wedge (\neg(x))))\ \mathbf{R}\ ((e) \wedge (y)))))) \wedge (\mathbf{G}((z) \rightarrow (((b) \wedge ((aa) \wedge (\neg(ab))))\ \mathbf{R}\ ((ac) \wedge (ad)))))$
	2	t	f	0	$(((((\mathbf{G}((a) \rightarrow (((b) \wedge ((c) \wedge (d)))\ \mathbf{U}\ ((e) \wedge (f))))) \wedge (\mathbf{G}((g) \rightarrow (((b) \wedge ((h) \wedge (i)))\ \mathbf{U}\ ((j) \wedge (k)))))) \wedge (\mathbf{G}((l) \rightarrow (((b) \wedge ((m) \wedge (n)))\ \mathbf{U}\ ((o) \wedge (p)))))) \wedge (\mathbf{G}((q) \rightarrow (((b) \wedge ((r) \wedge (s)))\ \mathbf{U}\ ((t) \wedge (u)))))) \wedge (\mathbf{G}((v) \rightarrow (((b) \wedge ((w) \wedge (\neg(x))))\ \mathbf{U}\ ((e) \wedge (y)))))) \wedge (\mathbf{G}((z) \rightarrow (((b) \wedge ((aa) \wedge (\neg(ab))))\ \mathbf{U}\ ((ac) \wedge (ad)))))$
	3	f	–	0	$(((((\mathbf{G}(\mathbf{F}(p))) \vee (\mathbf{G}(\mathbf{F}(q)))) \vee (\mathbf{G}(\mathbf{F}(r)))) \vee (\mathbf{G}(\mathbf{F}(s)))) \vee (\mathbf{G}(\mathbf{F}(t)))) \vee (\mathbf{G}(\mathbf{F}(u)))$
ring	0	t	f	0	$(\mathbf{G}(\mathbf{F}(p))) \wedge (\mathbf{G}(\mathbf{F}(\neg(p))))$
short	0	t	f	0	$\mathbf{G}((p) \rightarrow (\mathbf{F}(q)))$
srg5	0	t	f	4	$(((\mathbf{F}(\mathbf{G}(\neg(p)))) \wedge (\mathbf{G}(\mathbf{F}(q)))) \wedge (\mathbf{G}(\mathbf{F}(r)))) \rightarrow (\mathbf{F}((s)\ \mathbf{S}\ ((t)\ \mathbf{S}\ ((u)\ \mathbf{S}\ ((v)\ \mathbf{S}\ (w))))))$
	¬ 0, nv	f	–	4	$\neg(((((\mathbf{F}(\mathbf{G}(\neg(p)))) \wedge (\mathbf{G}(\mathbf{F}(q)))) \wedge (\mathbf{G}(\mathbf{F}(r)))) \rightarrow (\mathbf{F}((s)\ \mathbf{S}\ ((t)\ \mathbf{S}\ ((u)\ \mathbf{S}\ ((v)\ \mathbf{S}\ (w))))))) \wedge (((\mathbf{F}(\mathbf{G}(\neg(p)))) \wedge (\mathbf{G}(\mathbf{F}(q)))) \wedge (\mathbf{G}(\mathbf{F}(r)))))$

Table 2: Templates of the properties used in the experiments

first and have the conversion to SAT only encode the search for a fair cycle compared to encoding the property in a way very similar to such Büchi automaton as part of the conversion to SAT (which our encodings do). Therefore, the translation of a PLTL formula into a Büchi automaton in Fig. 7(f) is chosen to be similar to our BMC encoding. Comparing the performance of optimised translations from PLTL into Büchi automata with SAT-based approaches is out of the scope of this work. Similar reservations apply to our other experiments.

The details of the approaches and implementations used in the experiments are listed below. The first four approaches include all steps while the latter three are partial.

CAV2005(c,u,o)**:** means an implementation of a linear, incremental BMC procedure for PLTL on top of NuSMV 2.2.3. The exact BMC encoding is described in Sect. 6 and is essentially the encoding given in [HJL05]. The parameters describe

- whether the completeness check of Sect. 7 is enabled ($c =$ compl) or not ($c =$ nocompl),
- whether full virtual unrolling is applied ($u =$ unroll) or not ($u =$ nounroll), and
- whether the optimisations described in Sect. 5 of [HJL05] are active ($o =$ opt) or not ($o =$ noopt).

VMCAI2005: stands for an implementation of a linear, non-incremental BMC procedure for PLTL on top of NuSMV version 2.2.3. The exact BMC encoding is described in [LBHJ05]; it is very similar to the non-incremental PLTL encoding given in Sect. 5.1 except that it uses the fixed point encoding similar to that in Sect. 3.1 instead of the eventuality encoding.

NuSMV(BMCLTL)**:** is an example of a non-linear, non-incremental BMC encoding. It is the standard way to perform SAT-based bounded model checking for PLTL in NuSMV [CCG+02], version 2.2.3. For a description see [BC03].

NuSMV(BDDLTL)**:** is the standard method for BDD-based PLTL model checking in NuSMV [CCG+02], version 2.2.3. The property is translated into a symbolic Büchi automaton with [KPR98]. Cycle detection is performed with the backward version of the Emerson-Lei algorithm [EL86]; we always enabled the restriction to the set of reachable states. Neither dynamic reordering nor model-specific variable orders are used.

L2S(t,o)**:** is the liveness-to-safety transformation. The implementation of the transformation is based on previous work [SB04, Sch06] rather than on the formulation in Sect. 4. The encoding of the automaton representing the property is based on the construction outlined in Sect. 5.2 but is slightly modified for a tighter integration with the liveness-to-safety transformation. As an example, the signals indicating the start of the looping part and the end of a loop iteration are provided directly by the reduction rather than being separate input variables. The result is close to [LBHJ05] — in fact, [LBHJ05] was the starting point of our construction of a tight Büchi automaton.

The first parameter states which degree of virtual unrolling is used in the encoding of the property:

- $t =$ tight means full virtual unrolling up to the past operator depth of the property, and
- $t =$ notight performs no virtual unrolling at all.

The second parameter, o, indicates whether variable optimisation (see Sect. 4.2) is

- enabled ($o =$ ic), or
- not ($o =$ none).

Identification of input and constant variables is largely based on (conservative) syntactic criteria: a variable v in an SMV model clearly is an input variable if v appears only on the right-hand side of `next` assignments such that v is not itself in the scope of a `next` operator. Sometimes knowledge of a model was used to conclude that a variable is either a constant or an input variable.

A full model checking procedure is obtained in combination with any of the previous four or with **NuSMV**(BDDINVAR) below. We only use **CAV2005** and **NuSMV**(BDDINVAR).

B(t)**:** stands for a symbolic implementation of a Büchi automaton. The parameter t indicates whether a tight ($t =$ tight) version is used or not ($t =$ notight). The former corresponds to Sect. 5.2 while the latter is produced by NuSMV's ltl2smv tool, which implements [KPR98].[13] To obtain a model checking procedure we combine **B**(t) either with **NuSMV**(BDDLTL) or with **CAV2005**.

NuSMV(BDDINVAR)**:** is BDD-based forward invariant checking with version 2.2.3 of NuSMV [CCG+02]. We use this to perform BDD-based model checking with the liveness-to-safety transformation. State variables of the original model and their second instances are interleaved, but neither dynamic reordering nor a model-specific variable order are employed.

8.3. **Results and Comparisons.**

8.3.1. *Setting and notation.* We ran the benchmarks on Linux PC machines with a AMD Athlon(tm) 64 3200+ processor and 2 GB of memory. The memory limit for each run was set to 1.5 GB and the time limit to 1 hour by using the Linux `ulimit` command. For all SAT-based BMC procedures we used zChaff [MMZ+01], version 2004.11.15, as the SAT solver.

Tables 3 and 4 show the results for selected approaches. The a columns tell whether the property was found to be true (t) or false (f) in the instance (model,property) by the approach in question. The running times in t-columns are given in seconds except that TO (MO) means that the instance was not solved because of a timeout (running out of memory). For **L2S**(t,o)+X and **B**(t)+X approaches the running time does not include the liveness-to-safety transformation or Büchi automaton generation time, but only the solving time of X.[14]

The BMC-based approaches were run in the usual way: starting with the bound 0 and increasing it by one until (i) a counterexample or a proof was found, or (ii) the time or memory limit was reached. In the incremental approaches (**CAV2005**) there is only one SAT instance that is updated and solved again when the bound increases, while in the non-incremental approaches (**VMCAI2005**,**NuSMV**(BMCLTL)), the SAT instance for each bound is independently generated and checked. The k-columns give the bound that was

[13]Note, that the notight version still accepts shortest counterexamples for the future fragment of LTL. This is in contrast to the notation used in [AS06] where "tight" refers to an automaton based on [CGH97] (i.e., [KPR98] restricted to the future fragment of PLTL) and "non-tight" to one based on [SB00].

[14]Note that both the liveness-to-safety transformation and the Büchi automaton generation are performed symbolically and, therefore, can be done in polynomial time in the length of the description of the model and the formula.

reached. In particular, if the problem was solved (no TO or MO in the t-column), the k column gives the length of the counterexample or the bound required to prove the property.

For BDD-based approaches, the |cex|-column gives the length of the produced counterexample.

Fig. 6 shows scatter plots comparing the running times of different approaches to solve the benchmark instances. Red squares denote benchmark instances where the property is false (i.e., they have a counterexample), and black diamonds denote instances where the property holds. As mentioned earlier, the time limit was set to 3600 seconds (1 hour): timeouts are denoted by the time “value” 7200 and running out of memory by the time “value” 14400 in the scatter plots.

8.3.2. *Evaluation of different approaches.* Comparing the columns **NuSMV**(BMCLTL) and **VMCAI2005** of Table 3, plotted against each other also in Fig. 6(a), we can see the positive effect of having a more compact BMC encoding for PLTL formulae. Most of the properties we use involve past operators and for such formulae the encoding of **VMCAI2005** [LBHJ05] is linear in k while the encoding of **NuSMV**(BMCLTL) [BC03] is not.

From the columns **VMCAI2005** and **CAV2005**(nocompl,unroll,opt) of Table 3 and from Fig. 6(b) we see that by adapting the compact encoding of **VMCAI2005** to exploit modern *incremental* SAT solvers gives an additional major performance boost. Note that although **VMCAI2005** uses a fixed point encoding while **CAV2005**(nocompl,unroll,opt) uses eventuality encoding, we can claim that the major part of the observed performance boost is due to incrementality because of the results in Table 1 of [HJL05]: **VMCAI2005** and **CAV2005**(nocompl,unroll,opt) with no incrementality seem to behave very similarly.

As we can see from Fig. 6(c), the effect of doing virtual unrolling on the running times is not clear. However, there are slightly more cases in which unrolling helped than where it made things slower. This is due to the fact that unrolling can shorten counterexamples for formulas with past operators. Figure 7(a) illustrates that removing virtual unrolling may increase the counterexample length not only in theory but in practice, too. We also experimented with the option of not applying the optimisations of [HJL05, Sect. 5] in **CAV2005** and found that the optimisations don’t seem to have noticeable effect in practice, except that they sometimes reduce the bound required to prove a property when virtual unrolling is applied.

Figure 6(d) shows the effect of adding the completeness check described in Sect. 7 to the incremental PLTL BMC procedure **CAV2005**(nocompl,unroll,opt). The results demonstrate that the completeness check (i) enables one to sometimes also prove properties and not only find counterexamples, and (ii) generally slows down the BMC procedure by a factor of 2 or 3. However, if we compare the incremental and complete **CAV2005**(compl,unroll,opt) to the non-incremental and incomplete **VMCAI2005**, we see that incremental SAT solving techniques allows us to have a complete BMC procedure that almost always outperforms a non-incremental and incomplete state-of-the-art BMC procedure on the benchmarks we ran.

In Fig. 6(e) and (f) we compare bounded model checking with the specialised BMC encoding **CAV2005**(nocompl,unroll,opt) with an encoding based on the liveness-to-safety transformation (requiring only invariant checking in the BMC procedure) and based on using a tight Büchi automaton (requiring only fair loop detection in the BMC procedure). There is a noticeable overhead when using the liveness-to-safety transformation while, based on our set of experiments, we cannot conclude that a specialised encoding improves performance over the Büchi automaton. A main benefit of the specialised BMC encoding is, however, that it can also capture no-loop counterexamples.

Figure 6(g) contrasts finding shortest counterexamples using a BMC-based and a BDD-based method. While the former solves slightly more instances (that have a counterexample) within the given resource bounds, there are also some instances that it can not solve but the latter can. With respect to running time there is no clear winner either.

If we compare standard BDD-based PLTL model checking (**NuSMV**(BDDLTL)) and a state-of-the-art complete BMC procedure (**CAV2005**(compl,unroll,opt)), refer to Fig. 6(h), we can see that they are quite incomparable. Although the BDD-based approach seems to be better in proving properties as it produces less timeouts and memouts, there are instances that BMC proves much faster. And vice versa for properties having counterexamples: **NuSMV**(BDDLTL) solves (brp,¬0,nv) much faster but it (or any of the BDD-based methods we experimented) cannot solve the properties appearing in Tables 3 and 4 on the 1394-5-2 and 1394b-6-4 models.

Using a tight Büchi automaton with standard BDD-based model checking incurs a severe performance penalty (Fig. 6(i)). The lengths of the counterexamples produced by the tight and non-tight variants are very different with neither being consistently better (Fig. 7(b)). Note that **B**(tight)+**NuSMV**(BDDLTL) does not necessarily produce shortest possible counterexamples although it uses tight automata: the fair path finding algorithm employed in **NuSMV**(BDDLTL) does not produce shortest fair (k, l)-loops.

Tightness tends to come with a price in the liveness-to-safety and BDD-based approach as seen in Fig. 6(j), though less noticeable than in the standard BDD-based approach. While there is a price that grows with increasing past operator depth for some examples (1394-4-2,p2–4 and production-cell,p0/2/1), there is also the opposite case (1394b-4-2,p2–4). For the production-cell examples the partial unrolling optimisation proved valuable (not shown here): one level of unrolling (i.e., treating the specification as having past operator depth 1) gives shortest counterexamples as with a tight encoding but takes time only as with a non-tight encoding.

BDD-based model checking using the liveness-to-safety transformation is often faster than the standard approach of using BDDs (Fig. 6(k)) when the property is false, while it is typically slower for true properties. Further analysis indicates that early termination might play a role in this behaviour. Another, yet unexplored factor could be that **L2S**(t, o)+**NuSMV**(BDDINVAR) uses a forward invariant checking algorithm while **NuSMV**(BDDLTL) uses the backward version of the Emerson-Lei algorithm. While not shown, **L2S**(t, o) + **NuSMV**(BDDINVAR) tends to use more memory for both, false and true properties [Sch06]. However, it produces significantly shorter counterexamples (Fig. 7(c)) and is able to solve some examples where the standard approach reaches the time or memory limit. Note that the gain in counterexample length in (Fig. 7(c)) is the same when using **CAV2005** with unrolling.

The plot in Fig. 6(l) illustrates that the variable optimisation presented in Sect. 4.2 helps in BDD-based model checking with the liveness-to-safety transformation as expected, and it does not seem to have any adverse side effects.

We also experimented with a combination of the liveness-to-safety transformation and the temporal induction of [ES03]. That is, we use **L2S**(t,ic) to transform the PLTL problem to an invariant problem and then apply the temporal induction algorithm implemented in NuSMV (the command `check_invar_bmc_inc -a zigzag`). We were surprised that the resulting approach could not prove any of the true properties among the benchmarks we ran. We have no explanation for this behaviour at the moment, but we suspect that the liveness-to-safety transformation and the backwards working completeness checking of [ES03] might not fit together well.

9. Discussion and Conclusions

When comparing BMC approaches, the linear sized dedicated BMC encodings for PLTL offer better performance than alternative approaches based either on symbolic Büchi automata using the liveness-to-safety transformation or the original BMC encodings. The main advantage of the dedicated encodings over approaches using symbolic Büchi automata with fair loop detection is the ability of the dedicated encodings to also detect no-loop counterexamples. Adapted to incremental SAT solving techniques, BMC based on our encodings offers an efficient method for finding bugs. Virtual unrolling proved a useful technique to obtain both BMC encodings and Büchi automata that accept shortest counterexamples. The BMC experiments also show that the shorter counterexamples often lead to shorter times needed to find counterexamples to PLTL properties.

Using the liveness-to-safety translation with BDD-based invariant checking represents a competitive way to produce shortest counterexamples. For both SAT- and BDD-based approaches that find minimal length counterexamples there are problem instances that are solved by one approach but not by the other. Thus neither approach dominates the other.

When it comes to proving complex properties, the BMC approach presented here cannot yet compete with BDD-based methods. However, there are cases where our BMC approach is faster than the BDD-based approaches. Improving the capability to prove properties with BMC is therefore an important research direction.

There are at least two complementary research directions on proving properties of larger systems with BMC. One direction is based on generating stronger invariants than the current completeness formula. This can be done by adding invariants to formula states such as $|[\varphi]|_{k+1}^{d}$ to bind variables that are free. The invariants can be deduced from PLTL semantics. Another approach for generating invariants is formulating invariants based on the system's behaviour [dMRS03, AFF^{+}05]. The capability to prove properties can also be greatly improved if the $|[SimplePath]|_k$-predicate would have to include fewer state bits. A cone-of-influence reduction [CGP99] tailored for full PLTL or implementing the variable optimisations mentioned in Sect. 4.2 also for BMC could make this possible. Some insights might be gained by understanding why the combination of k-induction and the liveness-to-safety transformation performs so poorly for proving properties. We would also like to investigate methods based on Craig interpolants [McM03] to better understand the implementation techniques needed and performance obtainable from that method.

In this work we have concentrated on BMC encodings of PLTL properties. There are also other places where BMC can be improved. For example, [She04] discusses methods to improve CNF generation employed inside a prototype NuSMV variant used in [CRS04], an area we have not covered in our BMC implementations. Using SAT preprocessors, such as [EB05], to simplify the CNF after generation, is an alternative. Usually bounded model checking papers take the system transition relation $T(s, s')$ as given and do not try to exploit any special properties it might have. By more careful encoding of $T(s, s')$ significant performance gains can be obtained, at least for special classes of systems such as asynchronous systems [Hel01, HN03, JHN05, Jus05].

Table 3: Results of the experiments 1

		NuSMV (BMCLTL)			**VMCAI2005**			**CAV2005 (nocompl,unroll,opt)**			**CAV2005 (compl,unroll,opt)**		
model	prop.	a	k	t	a	k	t	a	k	t	a	k	t
1394-3-2	1		45	TO		40	TO		1237	MO	t	16	45
1394-3-2	¬1	f	11	17	f	11	31	f	11	9	f	11	12
1394-3-2	5		47	TO		49	TO		946	TO		65	TO
1394-3-2	¬5	f	11	19	f	11	17	f	11	8	f	11	11
1394-4-2	1		17	TO		18	TO		24	TO		24	TO
1394-4-2	¬1	f	16	1316	f	16	1676	f	16	424	f	16	679
1394-4-2	2		17	TO		22	TO		33	TO		29	TO
1394-4-2	3		8	TO		21	TO		31	TO		29	TO
1394-4-2	4		5	TO		22	TO		30	TO		29	TO
1394-4-2	5		26	TO		26	TO		43	TO		38	TO
1394-4-2	¬5	f	16	1287	f	16	1371	f	16	338	f	16	551
1394-5-2	1		14	TO		14	TO		15	TO		16	TO
1394-5-2	¬1	f	14	2937	f	14	2749	f	14	976	f	14	1295
1394-5-2	5		16	TO		16	TO		21	TO		20	TO
1394-5-2	¬5	f	14	3360	f	14	3425	f	14	938	f	14	1200
1394b-4-2	2	f	11	172	f	11	44	f	11	12	f	11	24
1394b-4-2	3		8	TO	f	11	46	f	11	15	f	11	23
1394b-4-2	4		5	TO	f	11	43	f	11	17	f	11	25
1394b-5-3	2	f	11	832	f	11	654	f	11	223	f	11	505
1394b-5-3	3		8	TO	f	11	665	f	11	484	f	11	426
1394b-5-3	4		5	TO	f	11	775	f	11	385	f	11	259
1394b-6-4	2		10	TO		10	TO	f	11	1875	f	11	2241
1394b-6-4	3		8	TO		10	TO	f	11	1930	f	11	2070
1394b-6-4	4		5	TO		9	TO	f	11	2125	f	11	2738
abp4	0	f	16	62	f	16	46	f	16	27	f	16	20
abp4	¬0		47	TO		52	TO		354	TO		46	TO
abp4	1		30	TO		29	TO		45	TO		38	TO
abp4	2	f	17	70	f	17	36	f	17	39	f	17	59
abp4	3		29	TO		30	TO		37	TO		36	TO
brp	0		31	TO		241	TO		3040	TO		86	TO
brp	¬0	f	1	0	f	1	0	f	1	0	f	1	0
brp	¬0, nv		22	TO		21	TO	f	24	600	f	24	573
brp	1		25	TO		38	TO		196	TO		78	TO
brp	¬1	f	1	0	f	1	0	f	1	0	f	1	0
counter	0		202	TO		1263	MO		11849	TO	t	23	0
counter	¬0	f	8	0	f	8	0	f	8	0	f	8	0
csmacd	0		18	TO		18	TO		19	TO		19	TO
csmacd	¬0	f	6	3	f	7	5	f	6	2	f	6	5
csmacd	1		22	TO		24	TO		33	TO		29	TO
csmacd	¬1	f	6	3	f	7	5	f	6	2	f	6	4
dme3	0		27	MO		49	TO		48	TO	f	62	2547
dme3	¬0	f	1	0	f	1	0	f	1	0	f	1	0
dme3	¬0, nv		27	MO	f	59	2330	f	59	641	f	59	1136
dme3	1		42	TO		53	TO		58	TO		65	TO
dme3	¬1	f	1	0	f	1	0	f	1	0	f	1	0
dme5	0		27	MO		57	TO		74	TO		67	TO
dme5	¬0	f	1	0	f	1	0	f	1	0	f	1	0
dme5	¬0, nv		27	MO		58	TO		75	TO		69	TO
dme5	1		42	TO		44	TO		48	TO		68	TO
dme5	¬1	f	1	0	f	1	0	f	1	0	f	1	0
mutex	0		226	TO		950	MO		10624	TO	t	18	0
mutex	¬0	f	6	0	f	6	0	f	6	0	f	6	0
pci	0		17	TO	f	18	3092	f	18	1339	f	18	1631
pci	¬0	f	0	0	f	0	0	f	0	0	f	0	0
pci	**F**0		14	TO	f	18	1121	f	18	514	f	18	610
pci	1		16	TO		18	TO		20	TO		20	TO
pci	¬1	f	1	0	f	1	0	f	1	0	f	1	0
prod-cons	0	f	21	972	f	21	63	f	21	14	f	21	35
prod-cons	¬0		25	TO	f	26	390	f	26	96	f	26	233
prod-cons	1		16	TO		68	TO		180	TO		72	TO
prod-cons	¬1, nv		16	TO	f	21	49	f	21	16	f	21	29
prod-cons	2	f	24	114	f	24	101	f	24	2	f	24	10
prod-cons	3	f	24	145	f	24	140	f	24	15	f	24	47
prod-cons	4		65	TO		63	TO		259	TO		91	TO
prod-cons	5		44	TO		49	TO		233	TO		82	TO
prod-cons	¬5	f	21	79	f	21	58	f	21	13	f	21	32
production-cell	0		6	TO		87	TO		1255	TO		149	MO
production-cell	¬0		6	TO		66	TO		73	TO	f	81	53
production-cell	1		4	TO		82	TO		619	TO		105	MO
production-cell	¬1		4	TO		63	TO	f	81	301	f	81	104
production-cell	2		4	TO		85	TO		1115	MO		115	MO
production-cell	¬2		4	TO		66	TO	f	81	104	f	81	85
production-cell	3		5	TO		19	TO		2391	MO	t	110	103
production-cell	¬3		6	TO		19	TO	f	81	234	f	81	42
production-cell	4		4	TO		15	TO		2295	MO	t	110	111
production-cell	¬4		5	TO		16	TO	f	81	73	f	81	46
bc57-sensors	0		20	TO		300	MO		1845	MO		130	MO
bc57-sensors	¬0		20	TO		99	TO	f	103	863		101	TO
bc57-sensors	1		57	TO		101	TO		101	TO		111	TO
bc57-sensors	¬1		56	TO		96	TO	f	103	1078		94	TO
bc57-sensors	2		56	TO		130	TO		1709	MO		132	MO
bc57-sensors	¬2		57	TO		97	TO	f	103	1215		92	TO
bc57-sensors	3		89	TO		99	TO	f	103	1173	f	103	3158
ring	0		184	TO		312	TO		756	MO	t	65	1012
ring	¬0	f	7	0	f	7	0	f	7	0	f	7	0
short	0		213	TO		1132	MO		3414	TO	t	10	0
short	¬0	f	1	0	f	1	0	f	1	0	f	1	0
srg5	0		13	TO		312	TO		805	MO		56	TO
srg5	¬0	f	1	0	f	1	0	f	1	0	f	1	0
srg5	¬0, nv	f	6	8	f	6	0	f	6	0	f	6	0

Table 4: Results of the experiments 2

		NuSMV (BDDLTL)			L2S(notight,ic)+ NuSMV (BDDINVAR)			L2S(tight,ic)+ NuSMV (BDDINVAR)			L2S(tight,ic)+ CAV2005 (nocompl,unroll,opt)			B(tight)+ CAV2005 (nocompl,unroll,opt)		
model	prop.	a	|cex|	t	a	|cex|	t	a	|cex|	t	a	k	t	a	k	t
1394-3-2	1	t		5	t		7	t		7		696	MO		1190	MO
1394-3-2	¬1	f	12	15	f	11	5	f	11	5	f	11	18	f	11	8
1394-3-2	5	t		3	t		13	t		13		177	MO		1099	TO
1394-3-2	¬5	f	11	25	f	11	7	f	11	7	f	11	25	f	11	8
1394-4-2	1	t		505	t		445	t		445		20	TO		26	TO
1394-4-2	¬1	f	20	721	f	16	336	f	16	336	f	16	689	f	16	462
1394-4-2	2	t		271	t		470	t		615		24	TO		27	TO
1394-4-2	3	t		288	t		610	t		825		24	TO		28	TO
1394-4-2	4	t		300	t		726	t		1276		23	TO		28	TO
1394-4-2	5	t		318	t		811	t		811		27	TO		40	TO
1394-4-2	¬5	f	19	1231	f	16	536	f	16	535	f	16	902	f	16	368
1394-5-2	1			MO			MO			MO		15	TO		15	TO
1394-5-2	¬1			MO			MO			MO	f	14	1097	f	14	896
1394-5-2	5			MO			MO			MO		18	TO		20	TO
1394-5-2	¬5			MO			MO			MO	f	14	1634	f	14	908
1394b-4-2	2	f	20	131	f	15	110	f	11	90	f	11	41	f	11	21
1394b-4-2	3	f	23	140	f	18	132	f	11	97	f	11	48	f	11	25
1394b-4-2	4	f	26	152	f	21	183	f	11	100	f	11	32	f	11	20
1394b-5-3	2			MO			MO			MO	f	11	1601	f	11	577
1394b-5-3	3			MO			MO			MO	f	11	850	f	11	202
1394b-5-3	4			MO			MO			MO	f	11	524	f	11	198
1394b-6-4	2			TO			TO			TO		10	TO	f	11	2839
1394b-6-4	3			TO			TO			TO	f	11	2857	f	11	1521
1394b-6-4	4			TO			TO			TO	f	11	2505	f	11	3415
abp4	0	f	37	1	f	19	11	f	16	6	f	16	33	f	16	17
abp4	¬0	t		0	t		0	t		0		58	TO		51	TO
abp4	1	t		0	t		58	t		58		33	TO		38	TO
abp4	2	f	40	2	f	17	11	f	17	11	f	17	153	f	17	38
abp4	3	t		0	t		62	t		62		24	TO		38	TO
brp	0	t		0			TO			TO		312	TO		106	TO
brp	¬0	f	6	4	f	1	0	f	1	0	f	1	0	f	1	0
brp	¬0, nv	f	68	14	f	24	81	f	24	85	f	24	1485	f	24	579
brp	1	t		4			TO			TO		26	TO		29	TO
brp	¬1	f	23	3	f	1	0	f	1	0	f	1	0	f	1	0
counter	0	t		0	t		0	t		0		2277	TO		1598	TO
counter	¬0	f	8	0	f	8	0	f	8	0	f	8	0	f	8	0
csmacd	0			MO	f	27	439	f	27	438		18	TO		18	TO
csmacd	¬0			MO	f	6	2	f	6	2	f	6	7	f	6	3
csmacd	1			TO			TO			TO		30	TO		21	TO
csmacd	¬1			MO	f	6	2	f	6	2	f	6	10	f	6	3
dme3	0	f	215	42	f	63	8	f	63	8	f	63	1488	f	63	670
dme3	¬0	f	1	0	f	1	0	f	1	0	f	1	0	f	1	0
dme3	¬0, nv	f	217	39	f	59	5	f	59	7	f	59	477	f	59	513
dme3	1	t		17	t		246	t		246		62	TO		64	TO
dme3	¬1	f	68	56	f	1	0	f	1	0	f	1	0	f	1	0
dme5	0	f	343	1289	f	103	303	f	103	299		71	TO		76	TO
dme5	¬0	f	1	10	f	1	1	f	1	1	f	1	0	f	1	0
dme5	¬0, nv	f	344	1371	f	99	271	f	99	309		74	TO		76	TO
dme5	1	t		483			TO			TO		46	TO		57	TO
dme5	¬1	f	108	1465	f	1	1	f	1	1	f	1	0	f	1	0
mutex	0	t		0	t		0	t		0		392	MO		1958	TO
mutex	¬0	f	7	0	f	6	0	f	6	0	f	6	0	f	6	0
pci	0	f	23	214			MO			MO	f	18	2971		13	TO
pci	¬0	f		3593	f	1	0	f	1	1	f	1	0	f	1	0
pci	**F**0	f	40	212			MO			MO	f	18	2450	f	18	893
pci	1			TO			MO			MO		15	TO		20	TO
pci	¬1	f	6	374	f	1	0	f	1	0	f	1	0	f	1	0
prod-cons	0	f	36	1200	f	21	78	f	21	85	f	21	49	f	21	14
prod-cons	¬0	f	69	42	f	26	188	f	26	204	f	26	210	f	26	122
prod-cons	1	t		1			MO			TO		48	TO		30	TO
prod-cons	¬1, nv	f	53	43	f	21	37	f	21	51	f	21	25	f	21	14
prod-cons	2	f	57	308	f	24	50	f	24	50	f	24	7	f	24	16
prod-cons	3	f	42	119	f	24	50	f	24	50	f	24	44	f	24	24
prod-cons	4	t		15			MO			MO		72	TO		196	TO
prod-cons	5	t		3			TO			TO		140	TO		169	TO
prod-cons	¬5	f	54	191	f	21	65	f	21	65	f	21	61	f	21	16
production-cell	0	t		694	t		3	t		12		192	MO		498	TO
production-cell	¬0	f	85	1094	f	83	6	f	81	10	f	81	219	f	81	13
production-cell	1	t		95	t		6			TO		168	MO		358	TO
production-cell	¬1	f	146	37	f	126	8			TO		66	TO	f	81	95
production-cell	2	t		81	t		6	t		517		171	MO		390	TO
production-cell	¬2	f	126	34	f	125	11	f	81	256		70	TO	f	81	17
production-cell	3	t		1657	t		5	t		5		1188	MO		1726	MO
production-cell	¬3	f	271	3050	f	81	11	f	81	11	f	81	457	f	81	19
production-cell	4			MO	t		8	t		8		1130	MO		1584	MO
production-cell	¬4			MO	f	81	26	f	81	26	f	81	551	f	81	35
bc57-sensors	0	t		65	t		2430	t		2192		101	TO		116	TO
bc57-sensors	¬0	f	112	206	f	103	109	f	103	102	f	103	2793	f	103	1198
bc57-sensors	1	t		152	t		3388	t		3391		93	TO		111	TO
bc57-sensors	¬1	f	104	909	f	103	99	f	103	99	f	103	2819	f	103	1084
bc57-sensors	2	t		39	t		2347	t		2352		82	TO		120	TO
bc57-sensors	¬2	f	104	866	f	103	101	f	103	101	f	103	3501	f	103	1317
bc57-sensors	3			TO	f	103	96	f	103	95	f	103	2973	f	103	1109
ring	0	t		0	t		0	t		0		90	TO		639	MO
ring	¬0	f	13	0	f	7	0	f	7	0	f	7	0	f	7	0
short	0	t		0	t		0	t		0		336	MO		3288	TO
short	¬0	f	3	0	f	1	0	f	1	0	f	1	0	f	1	0
srg5	0	t		0	t		0	t		0		117	TO		241	MO
srg5	¬0	f	16	0	f	1	0	f	1	0	f	1	0	f	1	0
srg5	¬0, nv	f	15	0	f	6	0	f	6	1	f	6	0	f	6	0

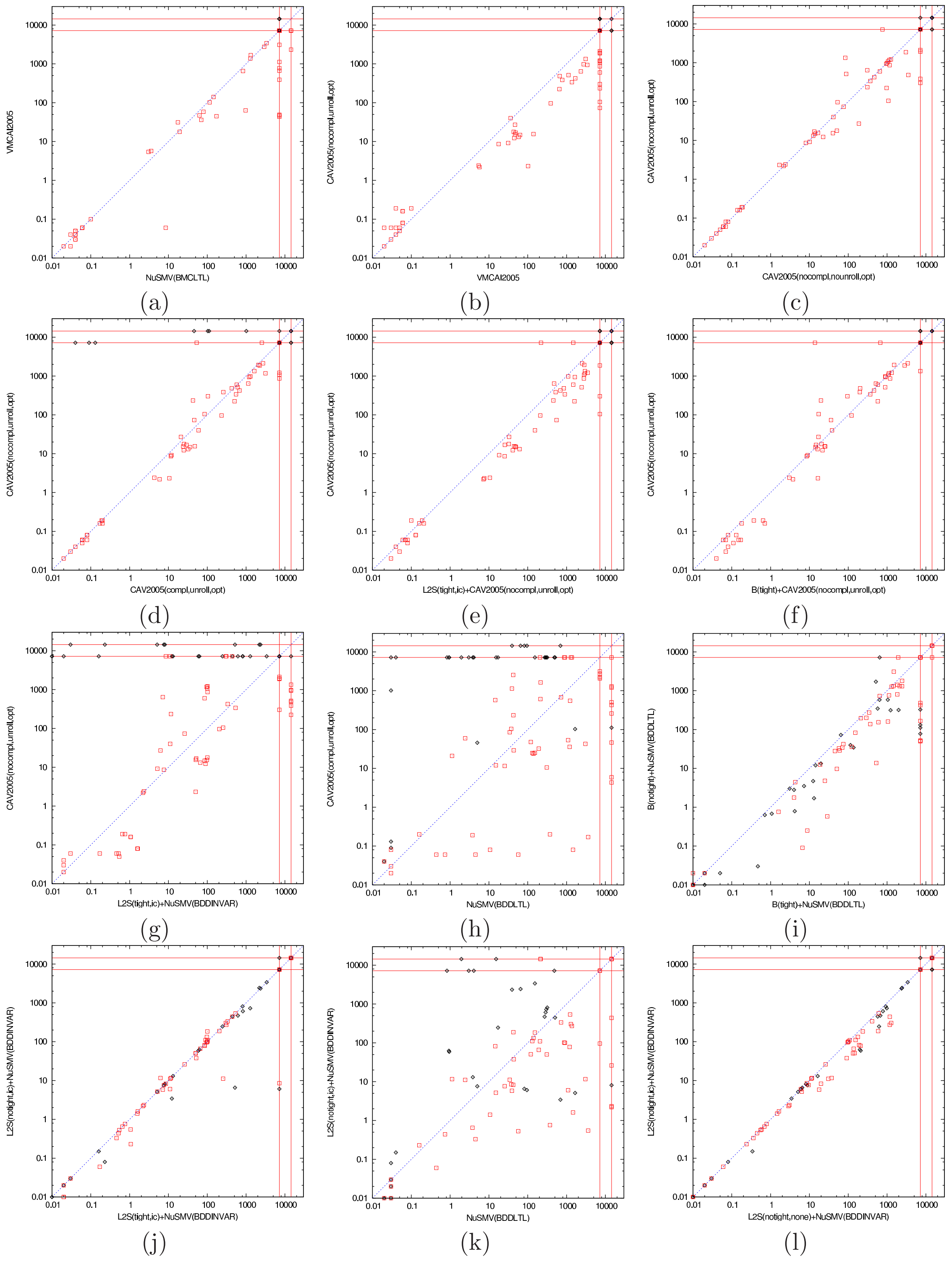
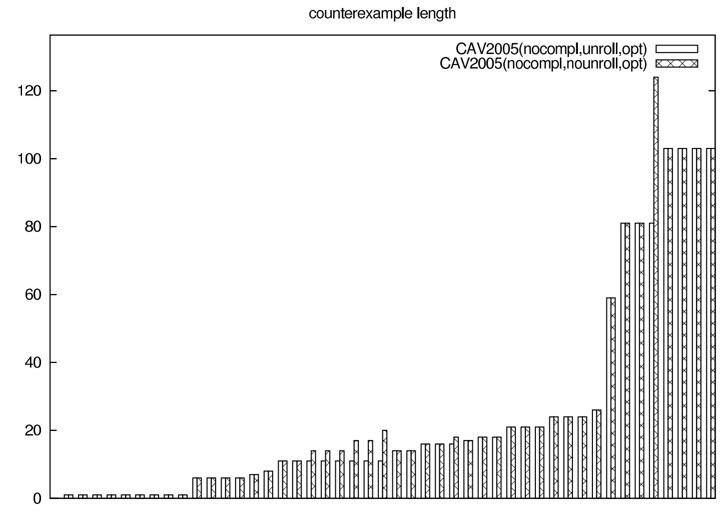
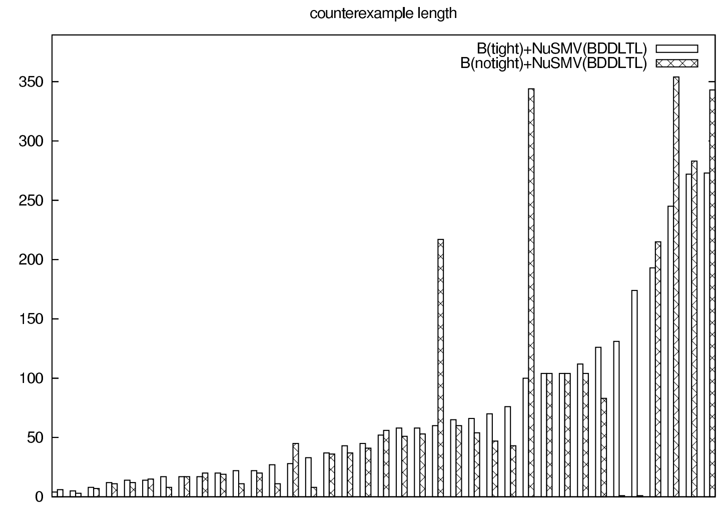


Figure 6: Scatter plots comparing the running times (in seconds) of different approaches

(a)



(b)



(c)

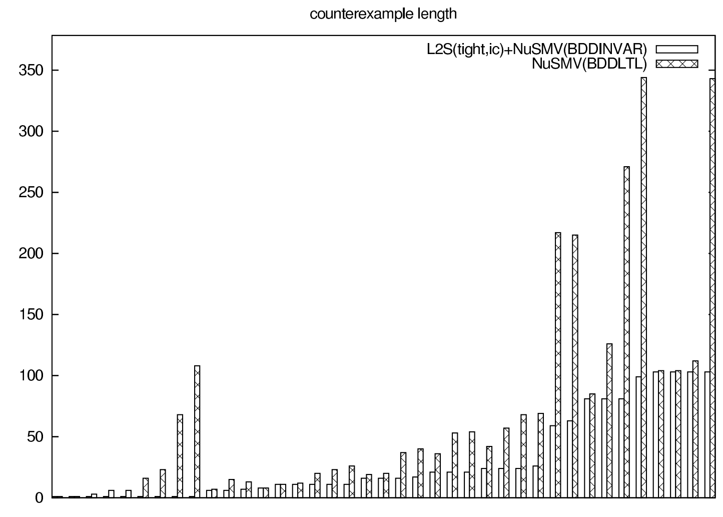


Figure 7: Counterexample length comparisons

Although PLTL is exponentially more succinct than LTL, it cannot express all ω-regular properties unlike some industry standard specification languages such as Accellera's PSL [Acc04, IEE05]. There are some encouraging initial results on bounded model checking of ω-regular properties very recently published [HJK+06], building on top of the work presented here. For an alternative approach to handling ω-regular properties, see [BCP+06].

Acknowledgements. The authors would like to thank the anonymous referees of Logical Methods in Computer Science for valuable comments that helped us to improve this paper. This work has been financially supported by the Academy of Finland (projects 109539, 112016, 211025, 213113) and the Emil Aaltonen Foundation.

References

[Acc04] Accellera. Property specification language: Reference manual – version 1.1, 2004. `http://www.eda.org/vfv/docs/PSL-v1.1.pdf`. 60

[AEF+05] R. Armoni, S. Egorov, R. Fraer, D. Korchemny, and M. Y. Vardi. Efficient LTL compilation for SAT-based model checking. In *ICCAD'05*, pages 877–884. IEEE, 2005. 44

[AFF+05] R. Armoni, L. Fix, R. Fraer, S. Huddleston, N. Piterman, and M. Y. Vardi. SAT-based induction for temporal safety properties. *ENTCS*, 119(2):3–16, 2005. 24, 44, 55

[AS04] M. Awedh and F. Somenzi. Proving more properties with bounded model checking. In *CAV'04*, volume 3114 of *LNCS*, pages 96–108, 2004. 19, 44

[AS06] M. Awedh and F. Somenzi. Termination criteria for bounded model checking: Extensions and comparison. *ENTCS*, 144(1):51–66, 2006. 18, 19, 44, 52

[BAS02] A. Biere, C. Artho, and V. Schuppan. Liveness checking as safety checking. In *FMICS'02*, ENTCS, 66(2). Elsevier, 2002. 21

[BB04] M. Benedetti and S. Bernardini. Incremental compilation-to-SAT procedures. In SAT [SAT04]. 40

[BC03] M. Benedetti and A. Cimatti. Bounded model checking for past LTL. In *TACAS'03*, volume 2619 of *LNCS*, pages 18–33. Springer, 2003. 3, 4, 6, 7, 15, 25, 26, 27, 28, 40, 51, 53

[BCC+03] A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. In *Advances in Computers*, volume 58. Academic Press, 2003. 8

[BCCZ99] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *TACAS'99*, volume 1579 of *LNCS*, pages 193–207. Springer, 1999. 2, 4, 7, 8, 13, 21, 26

[BCM+92] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98:142–170, 1992. 18, 19, 49

[BCP+06] R. Bloem, A. Cimatti, I. Pill, M. Roveri, and S. Semprini. Symbolic implementation of alternating automata. In *CIAA'06*, volume 4094 of *LNCS*, pages 208–218, August 2006. 60

[BCRZ99] A. Biere, E. Clarke, R. Raimi, and Y. Zhu. Verifying safety properties of a Power PC microprocessor using symbolic model checking without BDDs. In *CAV'99*, volume 1633 of *LNCS*, pages 60–71. Springer, 1999. 3, 15

[BHV04] A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract regular model checking. In *CAV'04*, volume 3114 of *LNCS*, pages 372–386. Springer, 2004. 21

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 38(8):677–691, 1986. 3

[BS05] D. Le Berre and L. Simon. Fifty-five solvers in Vancouver: The SAT 2004 competition. In *SAT*, volume 3542 of *LNCS*, pages 321–344. Springer, 2005. 2

[CCG+02] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, Marco Roveri, R. Sebastiani, and Armando Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *CAV'02*, volume 2404 of *LNCS*, pages 359–364. Springer, 2002. 4, 48, 49, 51, 52

[CCJ+06] R. Cavada, A. Cimatti, C. Jochim, G. Keighren, E. Olivetti, M. Pistore, M. Roveri, and A. Tchaltsev. *NuSMV 2.4 User Manual.* 2006. `http://nusmv.irst.itc.it/NuSMV/userman/v24/nusmv.pdf`. 25

[CFF$^+$01] F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M. Y. Vardi. Benefits of bounded model checking at an industrial setting. In *CAV'01*, volume 2102 of *LNCS*, pages 436–453. Springer, 2001. 3

[CGH97] E. M. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. *Formal Methods in System Design.*, 10(1):47–71, 1997. 18, 19, 49, 52

[CGMZ95] E. M. Clarke, O. Grumberg, K.L. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *DAC'95*, pages 427–432, 1995. 2

[CGP99] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, 1999. 10, 55

[CKOS04] E. Clarke, D. Kroening, J. Ouaknine, and O. Strichman. Completeness and complexity of bounded model checking. In *VMCAI'04*, volume 2937 of *LNCS*, pages 85–96. Springer, 2004. 3, 44

[CKOS05] E. Clarke, D. Kroening, J. Ouaknine, and O. Strichman. Computational challenges in bounded model checking. *STTT*, 7(2):174–183, 2005. 13, 19, 44

[CMU] The SMV system. `http://www.cs.cmu.edu/~modelcheck/smv.html`. 25

[Cou99] J-M. Couvreur. On-the-fly verification of linear temporal logic. In *Proceeding of the World Congress on Formal Methods in the Development of Computing Systems (FM'99)*, volume 1708 of *LNCS*, pages 253–271, Berlin, 1999. Springer. 18

[CPRS02] A. Cimatti, M. Pistore, M. Roveri, and R. Sebastiani. Improving the encoding of LTL model checking into SAT. In *VMCAI'02*, volume 2294 of *LNCS*, pages 196–207. Springer, 2002. 4, 8, 13

[CRS04] A. Cimatti, M. Roveri, and D. Sheridan. Bounded verification of past LTL. In *FMCAD'04*, volume 3312 of *LNCS*, pages 245–259. Springer, 2004. 13, 18, 33, 34, 55

[CRST06] A. Cimatti, M. Roveri, S. Semprini, and S. Tonetta. From PSL to NBA: a modular symbolic encoding. In *FMCAD'06*, 2006. To appear. 49

[DGV99] M. Daniele, F. Giunchiglia, and M.Y Vardi. Improved automata generation for linear temporal logic. In *Proceedings of the International Conference on Computer Aided Verification (CAV'99)*, volume 1633 of *LNCS*, pages 249–260, Berlin, 1999. Springer. 18

[dMRS02] L. de Moura, H. Rueß, and M. Sorea. Lazy theorem proving for bounded model checking. In *CADE'02*, volume 2392 of *LNCS*, pages 438–455. Springer, 2002. 13, 19

[dMRS03] L. de Moura, H. Rueß, and M. Sorea. Bounded model checking and induction: From refutation to verification. In *CAV'03*, volume 2725 of *LNCS*, pages 14–26. Springer, 2003. 44, 55

[EB05] N. Eén and A. Biere. Effective preprocessing in sat through variable and clause elimination. In *SAT*, volume 3569 of *LNCS*. Springer, 2005. 55

[EH00] K. Etessami and G. Holzmann. Optimizing Büchi automata. In *Concurrency Theory (CONCUR'2000)*, volume 1877 of *LNCS*, pages 153–167. Springer, 2000. 18

[EJ06] S. Edelkamp and S. Jabbar. Large-scale directed model checking LTL. In *SPIN'06*, volume 3925 of *LNCS*, pages 1–18. Springer, 2006. 21

[EL86] E. Emerson and C. Lei. Efficient model checking in fragments of the propositional mu-calculus (extended abstract). In *LICS'86*, pages 267–278. IEEE Computer Society, 1986. 51

[EL87] E. A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8(3):275–306, 1987. 21

[ES03] N. Eén and N. Sörensson. Temporal induction by incremental SAT solving. In *BMC'03*, volume 89 of *ENTCS*. Elsevier, 2003. 24, 25, 39, 43, 44, 45, 54, 55

[FSW02] A. Frisch, D. Sheridan, and T. Walsh. A fixpoint encoding for bounded model checking. In *FMCAD'02*, volume 2517 of *LNCS*, pages 238–255. Springer, 2002. 7, 13, 15, 18

[GO01] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *Computer Aided Verification (CAV'2001)*, volume 2102 of *LNCS*, pages 53–65. Springer, 2001. 18

[GO03] P. Gastin and D. Oddoux. LTL with past and two-way very-weak alternating automata. In *Mathematical Foundations of Computer Science 2003 (MFCS 2003)*, volume 2747 of *LNCS*, pages 439–448. Springer, 2003. 18

[GPVW95] R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification, Testing, and Verification*, pages 3–18. Chapman & Hall, 1995. 18

[Hel01] K. Heljanko. Bounded reachability checking with process semantics. In *CONCUR'01*, volume 2154 of *LNCS*, pages 218–232. Springer-Verlag, 2001. 55

[HJK$^+$06] K. Heljanko, T. Junttila, M. Keinänen, M. Lange, and T. Latvala. Bounded model checking for weak alternating Büchi automata. In *CAV'06*, volume 4144 of *LNCS*, pages 95–108. Springer-Verlag, August 2006. 60

[HJL05] K. Heljanko, T. Junttila, and T. Latvala. Incremental and complete bounded model checking for full PLTL. In *CAV'05*, volume 3576 of *LNCS*, pages 98–111. Springer-Verlag, July 2005. 2, 4, 7, 15, 28, 34, 36, 39, 43, 44, 51, 53

[HN03] K. Heljanko and I. Niemelä. Bounded LTL model checking with stable models. *Theory and Practice of Logic Programming*, 3(4–5):519–550, 2003. 13, 55

[Hol03] G. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003. 18

[IEE05] IEEE. IEEE Standard 1850 - Property Specification Language (PSL), 2005. 60

[Jan04] T. Janhunen. Representing normal programs with clauses. In *European Conference on Artificial Intelligence*, pages 358–362. IOS Press, 2004. 14

[JHN05] T. Jussila, K. Heljanko, and I. Niemelä. BMC via on-the-fly determinization. *STTT*, 7(2):89–101, 2005. 55

[JS05] H. Jin and F. Somenzi. An incremental algorithm to check satisfiability for bounded model checking. *ENTCS*, 119(2):51–65, 2005. 39

[Jus05] T. Jussila. On bounded model checking of asynchronous systems. Research Report A97, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, October 2005. Doctoral dissertation. 55

[KL93] H. Konuk and T. Larrabee. Explorations of sequential ATPG using boolean satisfiability. In *IEEE VLSI Test Symposium*. IEEE, 1993. 2

[KPR98] Y. Kesten, A. Pnueli, and L. Raviv. Algorithmic verification of linear temporal properties. In *ICALP'98*, volume 1443 of *LNCS*, pages 1–16. Springer, 1998. 18, 27, 33, 34, 35, 36, 38, 51, 52

[KS92] Henry A. Kautz and Bart Selman. Planning as satisfiability. In *10th European Conference on Artificial Intelligence (ECAI'92)*, pages 359–363, 1992. 2

[KS96] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In *National Conference on Artificial Intelligence and the Innovative Applications of Artificial Intelligence Conference*, pages 1194–1201. AAAI Press / MIT Press, 1996. 2

[KS03] D. Kroening and O. Strichman. Efficient computation of recurrence diameters. In *VMCAI'03*, volume 2575 of *LNCS*, pages 298–309. Springer, 2003. 24, 43, 45

[Kur94] R.P. Kurshan. *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach*. Princeton University Press, 1994. 3

[KV01] O. Kupferman and M.Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001. 7, 9, 10, 18

[Lat03] T. Latvala. Efficient model checking of safety properties. In *SPIN'03*, pages 74–88. Springer, 2003. 44

[Lat05] T. Latvala. Automata-theoretic and bounded model checking for linear temporal logic. Research Report A95, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, August 2005. Doctoral dissertation. 2

[LBHJ04] T. Latvala, A. Biere, K. Heljanko, and T. Junttila. Simple bounded LTL model checking. In *FMCAD'04*, volume 3312 of *LNCS*, pages 186–200. Springer, 2004. 2, 3, 4, 7, 8, 9, 13, 15

[LBHJ05] T. Latvala, A. Biere, K. Heljanko, and T. Junttila. Simple is better: Efficient bounded model checking for past LTL. In *VMCAI'05*, volume 3385 of *LNCS*, pages 380–395. Springer, jan 2005. 2, 7, 15, 32, 34, 39, 40, 48, 49, 51, 53

[LMS02] F. Laroussinie, N. Markey, and Ph. Schnoebelen. Temporal logic with forgettable past. In *LICS'02*, pages 383–392. IEEE Computer Society Press, 2002. 3, 5, 6, 26

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state programs satisfy their linear specification. In *ACM Symposium on Principles of Programming Languages*, pages 97–107, 1985. 18, 21, 49

[LPZ85] O. Lichtenstein, A. Pnueli, and L. D. Zuck. The glory of the past. In *Logic of Programs*, volume 193 of *LNCS*, pages 196–218. Springer, 1985. 3, 5

[McM93] K. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. Kluwer Academic Publishers, 1993. 25

[McM03] K.L. McMillan. Interpolation and SAT-based model checking. In *CAV'03*, volume 2725 of *LNCS*, pages 1–13. Springer, 2003. 21, 24, 44, 55

[MMZ$^+$01] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *DAC'01*. IEEE, 2001. 52

[NuS] NuSMV home page. `http://nusmv.irst.itc.it/`. 4

[PBG05] M.R. Prasad, A. Biere, and A. Gupta. A survey of recent advances in SAT-based formal verification. *STTT*, 7(2):156–173, 2005. 4

[SAT04] *SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing*, 2004. 60, 63

[SB00] F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *CAV'00*, volume 1855 of *LNCS*, pages 248–263. Springer, 2000. 18, 52

[SB03] V. Schuppan and A. Biere. Verifying the IEEE 1394 FireWire Tree Identify Protocol with SMV. *Formal Asp. Comput.*, 14(3):267–280, 2003. 48, 49

[SB04] V. Schuppan and A. Biere. Efficient reduction of finite state model checking to reachability analysis. *STTT*, 5(2-3):185–204, March 2004. 2, 3, 21, 24, 51

[SB05] V. Schuppan and A. Biere. Shortest counterexamples for symbolic model checking of LTL with past. In *TACAS'05*, volume 3440 of *LNCS*, pages 493–509. Springer, 2005. 2, 7, 18, 21, 33, 34

[SB06] V. Schuppan and A. Biere. Liveness checking as safety checking for infinite state spaces. In *INFINITY '05*, ENTCS, 149(1), pages 79–96. Elsevier, 2006. 21

[Sch01] K. Schneider. Improving automata generation for linear temporal logic by considering the automaton hierarchy. In *LPAR'01*, volume 2250 of *LNCS*, pages 39–54. Springer, 2001. 18

[Sch06] V. Schuppan. *Liveness Checking as Safety Checking to Find Shortest Counterexamples to Linear Time Properties*. PhD thesis, ETH Zürich, 2006. 2, 18, 21, 24, 25, 34, 36, 51, 54

[She04] D. Sheridan. The optimality of a fast CNF conversion and its use with SAT. In SAT [SAT04]. 55

[SMSdB04] M. Sirjani, A. Movaghar, A. Shali, and F. de Boer. Modeling and verification of reactive systems using Rebeca. *Fundamenta Informaticae*, 63(4):385–410, 2004. 48, 49

[SSS00] M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a SAT-solver. In *FMCAD'00*, volume 1954 of *LNCS*, pages 108–125. Springer, 2000. 24, 39, 43, 44, 45

[ST03] R. Sebastiani and S. Tonetta. "More deterministic" vs. "smaller" Büchi automata for efficient LTL model checking. In *CHARME'03*, volume 2860 of *LNCS*, pages 126–140. Springer, 2003. 18

[Str01] O. Strichman. Pruning techniques for the SAT-based bounded model checking problem. In *CHARME'01*, volume 2144 of *LNCS*, pages 58–70. Springer, 2001. 3, 39

[Str04] O. Strichman. Accelerating bounded model checking of safety properties. *Formal Methods in System Design*, 24(1):5–24, 2004. 3

[STV05] R. Sebastiani, S. Tonetta, and M. Vardi. Symbolic systems, explicit properties: On hybrid approaches for LTL symbolic model checking. In *CAV'05*, volume 3576 of *LNCS*, pages 350–363. Springer, 2005. 49

[TH02] H. Tauriainen and K. Heljanko. Testing LTL formula translation into Büchi automata. *STTT*, 4(1):57–70, 2002. 10, 12

[VIS96] The VIS Group. VIS: A system for verification and synthesis. In *CAV'96*, volume 1102 of *LNCS*, pages 428–432. Springer, 1996. 18

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *LICS'86*, pages 322–331, Cambridge, 1986. 3, 6, 19, 21, 49

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994. 18, 23

[WKS01] J. Whittemore, J. Kim, and K. A. Sakallah. SATIRE: A new incremental satisfiability engine. In *DAC'01*, pages 542–545, 2001. 3, 39

[WVS83] P. Wolper, M. Vardi, and A. Sistla. Reasoning about infinite computation paths. In *FOCS'83*, pages 185–194. IEEE Computer Society, 1983. 18

[ZMMM01] L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient conflict driven learning in boolean satisfiability solver. In *ICCAD'01*, pages 279–285. IEEE, 2001. 39