Bounded Model Checking for Past LTL

Marco Benedetti and Alessandro Cimatti

Istituto per la Ricerca Scientifica e Tecnologica (IRST)
Via Sommarive 18, 38055 Povo, Trento, Italy
{benedetti,cimatti}@irst.itc.it

Abstract. The introduction of Past Operators enables to produce more natural formulation of a wide class of properties of reactive systems, compared to traditional pure future temporal logics. For this reason, past temporal logics are gaining increasing interest in several application areas, ranging from Requirement Engineering to Formal Verification and Model Checking. We show how SAT-based Bounded Model Checking techniques can be extended to deal with Linear Temporal Logics with Past Operators (PLTL). Though apparently simple, this task turns out to be absolutely non-trivial when tackled in its full generality. We discuss a bounded semantics for PLTL, we show that it is correct (and complete), and propose an encoding scheme able to cope with PLTL formulas. Finally, we implement the encoding in NuSMV, and present a first experimental evaluation of the approach.

1 Introduction

Temporal logics [14] are traditionally used in formal verification to predicate about the future evolutions of dynamic systems, both with a linear model or a branching model of time. The most typical application is the representation of the properties of dynamic systems within model checking tools. However, many interesting properties of dynamic systems are naturally formulated in a way that is not limited to the future evolution, but may refer to events in the past. For instance, properties such as "if a problem is diagnosed, then a failure must have occurred in the past" or "a grant is always issued as a consequence of a previous request" are not straightforward to express with future temporal operators. For this reason, temporal logics with operators that allow for direct reference of *past* events are being devoted increasing interest in formal verification for requirement engineering [8, 15, 18, 23], and planning [2].

We are interested in extending state-of-the-art verification techniques developed for (future) temporal logics, to encompass the case of past operators. In particular, we are interested in extending to SAT-based Bounded Model Checking (BMC) techniques, originally introduced in [5], that are being widely accepted as an effective alternative to BDD-based symbolic methods [3, 13, 21, 22]. In BMC, an existential model checking problem for Linear Temporal Logic is reduced to a problem of propositional satisfiability, and efficient SAT solvers are then used to tackle this problem. The idea behind BMC is to look for finitely represented paths. Two cases arise: Counterexamples are either finite prefixes of paths (in the case of safety properties), or exhibit an infinite lasso-shaped structure (in the case of liveness properties) based on the existence of a loopback.

We tackle the BMC problem for Linear Temporal Logic with Past Operators (PLTL). From a theoretical point of view, it is well known that past operators do not add expressive power w.r.t. pure-future LTL (as opposite to other temporal logics [19]). In fact, a result from [16] states that any PLTL formula can be re-written by only using future-time operators, even though a non-elementary blow-up (w.r.t. the size of the formula) stems from every known translation procedure. Even if the expressive power of the underlying logic is left unchanged, past operators are still very useful in practice, in that they bring additional expressivity from the perspective of end users. In fact, it is of paramount importance to provide formalisms that allow for an easy-to-understand and compact characterization of the desired behaviours of the system. Past operators help keeping specifications short and simple.

The problem of BMC for PLTL is rather simple in the case of finite prefixes, since the extent of the past is clear from any point. The construction becomes non-trivial when loops are taken into account, because infinite paths are presented by means of a loopback, and there is potentially more than one past for each point in the loop (at the loopback point we have to chose whether a "back to the future" step is to be taken).

Here we provide a full characterization of the problem of BMC for PLTL, define a bounded semantics, and show how to encode PLTL problems into propositional satisfiability instances. We implement the encoding into the NuSMV model checker, and provide some experimental evidence on the advantages of the approach.

This paper is structured as follows. Section 2 introduces the syntax and semantics of PLTL. In Section 3 we recall the basics ideas underlying Bounded Model Checking. Section 4 discusses the encoding of PLTL for bounded paths. In Section 5 we highlight the problems with loop paths, and present our solution. In Section 6 we discuss the implementation of these ideas within NuSMV and present a preliminary experimental comparison. Section 7 closes the paper with a few concluding remarks.

2 Linear Temporal Logic with Past Operators

In this paper we consider PLTL, i.e. the Linear Temporal Logic (LTL) augmented with past operators. The starting point is standard LTL, the formulas of which are constructed from propositional symbols by applying the future temporal operators $\mathbf{X}$ (next), $\mathbf{F}$ (future), $\mathbf{G}$ (globally), $\mathbf{U}$ (until), and $\mathbf{R}$ (releases), in addition to the usual boolean connectives. PLTL extends LTL by introducing the past operators $\mathbf{Y}$, $\mathbf{Z}$, $\mathbf{O}$, $\mathbf{H}$, and $\mathbf{S}$, which are the temporal duals of the future operators and allow us to express statements on the past time instants. The $\mathbf{Y}$ (for "$\mathbf{Y}esterday$") operator is the dual of $\mathbf{X}$ and refers to the *previous* time instant. At any non-initial time, $\mathbf{Y}f$ is true if and only f holds at the previous time instant. The $\mathbf{Z}$ (the name is just a mnemonic choice) operator is very similar to the $\mathbf{Y}$ operator, and it differs in the way the initial time instant is dealt with. At time zero, $\mathbf{Y}f$ is false, while $\mathbf{Z}f$ is true. The $\mathbf{O}$ (for "$\mathbf{O}nce$") operator is the dual of $\mathbf{F}$ (sometimes in the future), so that $\mathbf{O}f$ is true iff f is true at some past time instant (including the present time[1]). Likewise, $\mathbf{H}$ (for "$\mathbf{H}istorically$") is the past-time version of $\mathbf{G}$ (always in the future), so that $\mathbf{H}f$ is true iff f is always true

[1] We adopt a non-strict semantics for time operators, so that all temporal operators other than $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$ take into account the present time instant.

in the past. The $\mathbf{S}$ (for "$\mathbf{S}ince$") operator is the dual of $\mathbf{U}$ (until), so that $f\mathbf{S}g$ is true iff g holds somewhere in the past and f is true from then up to now. Finally, we have $f\mathbf{T}g = \neg(\neg f\mathbf{S}\neg g)$ ($\mathbf{T}$ is called the "$\mathbf{T}rigger$" operator), exactly as in the future case we have $f\mathbf{R}g = \neg(\neg f\mathbf{U}\neg g)$. The syntax of PLTL is formally defined as follows.

Definition 1 (Syntax of PLTL). *The grammar for PLTL formulas is*

$$PLTL \ni f, g \doteq q \mid \neg f \mid f \circ^{\mathbf{B}} g \mid \circ^{\mathbf{F}}_{\mathbf{1}} f \mid f \circ^{\mathbf{F}}_{\mathbf{2}} g \mid \circ^{\mathbf{P}}_{\mathbf{1}} f \mid f \circ^{\mathbf{P}}_{\mathbf{2}} g$$

where $q \in \mathcal{A}$ and $\mathcal{A}$ is a set of atomic propositions, $\circ^{\mathbf{B}} \in \{\wedge, \vee\}$ stands for a boolean connective, $\circ^{\mathbf{F}}_{\mathbf{1}} \in \{\mathbf{X}, \mathbf{F}, \mathbf{G}\}$ and $\circ^{\mathbf{F}}_{\mathbf{2}} \in \{\mathbf{R}, \mathbf{U}\}$ are future temporal operators (unary and binary, respectively) , and $\circ^{\mathbf{P}}_{\mathbf{1}} \in \{\mathbf{Y}, \mathbf{Z}, \mathbf{O}, \mathbf{H}\}$ and $\circ^{\mathbf{P}}_{\mathbf{2}} \in \{\mathbf{T}, \mathbf{S}\}$ are past temporal operators (unary and binary).

We write $f \rightarrow g$ for $\neg f \vee g$, and $f \leftrightarrow g$ for $(f \rightarrow g) \wedge (g \rightarrow f)$. As usual, Kripke structures are used to give the semantics of PLTL formulas.

Definition 2. *A* Kripke structure *is a tuple $M = \langle S, I, T, \ell\rangle$, where S is a finite set of states, $I \subseteq S$ is the set of initial states, $T \subseteq S \times S$ is a transition relation between states and $\ell : S \rightarrow 2^{\mathcal{A}}$ is a function which labels each state with a subset of the set $\mathcal{A}$ of atomic propositions.*

For an infinite sequence of states $\pi = (s_0, s_1, ...)$, we define $\pi(i) = s_i$, $\pi^i = (s_i, s_{i+1}, ...)$ and $\pi_{|i} = (0, 1, ..., s_i)$ for $i \in \mathbb{N}$, and we say that π is a *path* in M if $\pi(i) \rightarrow \pi(i+1)$ for all $i \in \mathbb{N}$, where $s \rightarrow t$ means that $\langle s, t\rangle \in T$. We also assume, without loss of generality, that the transition relation is *total*, i.e. that for every state $s \in S$, there exists at least one state $t \in S$ such that $\langle s, t\rangle \in T$. Infinite paths which are made up of a finite prefix u followed by a portion v repeated infinitely many times are called *loop paths*.

Definition 3. *A path π is a (k, l)-*loop*, with $l < k$, if $\pi(l) = \pi(k)$ and $\pi = u \cdot v^{\omega}$, where $u = (\pi(0), \ldots, \pi(l-1))$ and $v = (\pi(l), \ldots, \pi(k-1))$. We define the* period *of a (k, l)-loop as $k - l$. The successor of the i instant in a (k, l)-loop, $succ(i)$, is defined as $k + 1$ if $i < k - 1$, and l otherwise.*

In the following, unless specified otherwise, we assume that a given Kripke structure $M = \langle S, I, T, \ell\rangle$ is given. We also assume that different states in S have different labelings, i.e. for all $s, s' \in S$, $\ell(s) \neq \ell(s')$ iff $s \neq s'$. We use π to denote a (k, l)-loop (with $l < k$), p to denote the period $k - l$ of the loop, f and g to denote PLTL formulae, and q to denote propositions in $\mathcal{A}$. We write $[a, b)$ and $(a, b]$ to denote right-open and left-open intervals of integers, and (ab)use this notation by writing $[a, \infty)$ to denote the infinite set $\{i \in \mathbb{N}, i \geq a\}$.

Definition 4 (Semantics of PLTL). *Let M be a boolean Kripke structure, π be a path in M and f be a PLTL formula. Then $(\pi, i) \models f$ (f holds in π at time i) is inductively* *defined as follows.*

$$
\begin{array}{lll}
(\pi, i) \models q & \mathit{iff} & q \in \ell(\pi(i)) \\
(\pi, i) \models \neg f & \mathit{iff} & (\pi, i) \not\models f \\
(\pi, i) \models f \vee g & \mathit{iff} & (\pi, i) \models f \mathit{\ or\ } (\pi, i) \models g \\
(\pi, i) \models f \wedge g & \mathit{iff} & (\pi, i) \models f \mathit{\ and\ } (\pi, i) \models g \\
 & & \\
(\pi, i) \models \mathbf{X} f & \mathit{iff} & (\pi, i+1) \models f \\
(\pi, i) \models \mathbf{F} f & \mathit{iff} & \exists j \in [i, \infty) \,.\, (\pi, j) \models f \\
(\pi, i) \models \mathbf{G} f & \mathit{iff} & \forall j \in [i, \infty) \,.\, (\pi, j) \models f \\
(\pi, i) \models f \mathbf{U} g & \mathit{iff} & \exists j \in [i, \infty) \,.\, ((\pi, j) \models g \mathit{\ and\ } \forall k \in [i, j) \,.\, (\pi, k) \models f) \\
(\pi, i) \models f \mathbf{R} g & \mathit{iff} & \forall j \in [i, \infty) \,.\, ((\pi, j) \models g \mathit{\ or\ } \exists k \in [i, j) \,.\, (\pi, k) \models f) \\
 & & \\
(\pi, i) \models \mathbf{Y} f & \mathit{iff} & i > 0 \mathit{\ and\ } (\pi, i-1) \models f \\
(\pi, i) \models \mathbf{Z} f & \mathit{iff} & i = 0 \mathit{\ or\ } (\pi, i-1) \models f \\
(\pi, i) \models \mathbf{O} f & \mathit{iff} & \exists j \in [0, i] \,.\, (\pi, j) \models f \\
(\pi, i) \models \mathbf{H} f & \mathit{iff} & \forall j \in [0, i] \,.\, (\pi, j) \models f \\
(\pi, i) \models f \mathbf{S} g & \mathit{iff} & \exists j \in [0, i] \,.\, ((\pi, j) \models g \mathit{\ and\ } \forall k \in (j, i] \,.\, (\pi, k) \models f) \\
(\pi, i) \models f \mathbf{T} g & \mathit{iff} & \forall j \in [0, i] \,.\, ((\pi, j) \models g \mathit{\ or\ } \exists k \in (j, i] \,.\, (\pi, k) \models f)\,.
\end{array}
$$

A formula f is valid on a path π in M (written $\pi \models f$) iff $(\pi, 0) \models f$. A formula f is existentially *valid in M ($M \models \mathbf{E} f$) iff $\pi \models f$ for some path π in M. Conversely, f is* universally *valid in M ($M \models \mathbf{A} f$) iff $\pi \models f$ for every path π in M.*

Although the use of past operators in LTL does not introduce expressive power, it allows us to formalize properties more naturally. For instance, "*if a problem is diagnosed, then a failure must have previously occurred*" can be represented in PLTL as

$$\mathbf{G}(\mathit{problem} \rightarrow \mathbf{O}\ \mathit{failure}) \tag{1}$$

that is more natural than its pure-future counterpart $\neg(\neg \mathit{failure} \mathbf{U} \mathit{problem})$. Similarly, the property "*grants are issued only upon requests*" can be easily specified as

$$\mathbf{G}(\mathit{grant} \rightarrow \mathbf{Y}(\neg \mathit{grant}\ \mathbf{S}\ \mathit{request})) \tag{2}$$

compared to the corresponding pure-future translation

$$(\mathit{request}\, \mathbf{R} \neg \mathit{grant}) \ \wedge\ \mathbf{G}(\mathit{grant} \rightarrow (\mathit{request} \vee (\mathbf{X}(\mathit{request}\, \mathbf{R} \neg \mathit{grant})))).$$

As for the pure future case, any formula in PLTL can be reduced to *Negation Normal Form* (NNF), where negation only occurs in front of atomic propositions. This linear time transformation is obtained by pushing the negation towards the leaves of the syntactic tree of the formula and exploiting the dualities between $\mathbf{F}$ and $\mathbf{G}$, $\mathbf{U}$ and $\mathbf{R}$, $\mathbf{O}$ and $\mathbf{H}$, and $\mathbf{S}$ and $\mathbf{T}$. The case of previous time is a bit tricky, since we have to rely on the two properties $\neg \mathbf{Y} f \equiv \mathbf{Z} \neg f$ and $\neg \mathbf{Z} f \equiv \mathbf{Y} \neg f$ which extend the single future-case rule $\neg \mathbf{X} f \equiv \mathbf{X} \neg f$ (we have both $\neg \mathbf{Y} f \not\equiv \mathbf{Y} \neg f$ and $\neg \mathbf{Z} f \not\equiv \mathbf{Z} \neg f$, because of their semantics at the initial time point). Notice that whenever we limit our attention to NNF formulas, the semantic rule $(\pi, i) \models \neg q$ iff $q \notin \ell(\pi(i))$ can be substituted for $(\pi, i) \models \neg f$ iff $(\pi, i) \not\models f$ in Definition 4, with no loss of completeness.

3 Bounded Model Checking

LTL Model Checking is interpreted universally, as the problem of checking whether a certain ϕ holds on all the paths of a Kripke structure M. The problem can be tackled by refutation, by checking the existential problem $M \models \mathbf{E}\neg\phi$. BMC tackles the *bounded* version of the existential problem $M \models_k \mathbf{E}\neg\phi$, by looking for witnesses of the violation within a certain bound k. When the k-bounded version of the problem is considered, only paths with at most k distinct transitions are taken into account. Such limited paths can be either finite (in which case they are finite prefixes of a path) or infinite (in which case paths exhibit a looping behaviour). Whichever the case, if a witness is found with bound k, then the property ϕ is violated in the general sense ($M \models \mathbf{E}\neg\phi$, so $M \not\models \mathbf{A}\phi$). Otherwise, if no violation is found, the bound can be increased until either a witness with a higher bound is found, or a limit bound is reached that enables to conclude that no violation exists, and thus $M \models \mathbf{A}\phi$. In the following, we focus on the existential model checking problem $M \models \mathbf{E}\neg\phi$, in particular on its bounded version $M \models_k \mathbf{E}\neg\phi$. This problem can be effectively reduced to a propositional formula [5] that is satisfiable if and only if there exists a violation of ϕ within bound k. The satisfiability of the propositional formula can then be effectively tackled by exploiting the impressive power of state-of-the-art propositional solvers (e.g. Chaff [20]). Since the seminal work in [5], the approach has been thoroughly investigated and extended [6, 7, 25, 22, 1], and its practical applicability has been widely recognized [13, 3]. Given the finiteness of M, it is possible to define a k beyond which it is impossible to find a violation. The simple limits given in [6] are too large to be reached in practice, and therefore, BMC was initially proposed as a technique oriented to debugging. Improvements are proposed in [21] and in [3], where inductive reasoning and structural techniques allow the overapproximation of the bound for safety properties.

The encoding into propositional logic is based on the standard representation of Kripke structures used in symbolic model checking, where two sets of state variables — the current set V and the next set V' — are used to represent sets of states and transitions. In the following, we write $I(V)$ for a formula in the V variables representing the set of initial states of M, and $T(V, V')$ for the formula representing the transition relation of M. Given a bound k, the vector of state variables V is replicated $k+1$ times, thus obtaining the vectors $V^0, \ldots, V^k$. Intuitively, an assignment to V_i represents a value of the state vector after i transitions. We write q^i for the variable representing proposition q at time i. The propositional encoding $[\![M \models_k \mathbf{E}\neg\phi]\!]$ of the problem $M \models_k \mathbf{E}\neg\phi$, is a formula in the variables $V^0, \ldots, V^k$, structured as a binary conjunction. The first conjunct is the formula $[\![M]\!]_k \doteq I(V^0) \wedge T(V^0, V^1) \wedge \ldots \wedge T(V^{k-1}, V^k)$, where $T(V^i, V^{i+1})$ stands for the formula obtained by substituting the variables in V with the (corresponding) variables in V^i, and the variables in V' with the (corresponding) variables in V^{i+1}. A similar argument holds for $I(V^0)$. The formula $[\![M]\!]_k$ constrains the values of the state vectors at the different time instants in such a way that a satisfying assignment represents a path in the Kripke structure.

The second conjunct, in the following referred to as encoding of the formula with bound k, constrains the $k+1$ state vectors in such a way that their assignments characterize a path satisfying $\neg\phi$, so that a satisfying assignment to $[\![M \models_k \mathbf{E}\neg\phi]\!]$ represents a path in the Kripke structure that violates ϕ. The encoding of $\neg\phi$ with bound k has the form

$$[\![\neg\phi]\!]_k \vee \bigvee_{0\leq l<k} ({}_l L_k \wedge {}_l[\![\neg\phi]\!]_k).$$

The formula $[\![\neg\phi]\!]_k$ represents a violation of ϕ on a finite prefix of a path with k transitions, without assuming the existence of a loop. So, every finite sequence of states satisfying the conjunction of this formula with $[\![M]\!]_k$ can be extended to at least one infinite behaviour violating ϕ, thanks to the totality of the transition relation.

The formula ${}_l[\![\neg\phi]\!]_k$ relates to the construction of a particular counterexample of infinite length. In fact — depending on the structure of the formula being analyzed — there are cases where the production of a particular infinite behaviour is required to show that a property is violated. Although only a finite number of transitions and states are available in the encoding, this representation is possibly enough to represent an infinite path as well. In fact, the formula ${}_l[\![\neg\phi]\!]_k$ encodes — for each value of l — the existence of a counterexample for ϕ on a path structured as a (k, l)-loop. We produce such encoding with bound k assuming that there exists a loopback at a certain previous time instant $l < k$ and enforcing this loop condition by constraining the variable of the state vectors at l and k to be pairwise equivalent, by means of the condition ${}_l L_k \doteq \bigwedge_{q\in\mathcal{A}}(q^l \leftrightarrow q^k)$. (This definition of the loop condition slightly differs from the one given in [5], in the way the loopback point is identified. The new definition also allows us to interpret the bound k as the number of transitions uniformly for the cases with and without loop.)

4 Bounded Model Checking for PLTL without Loopbacks

We now consider the encoding for PLTL formulae in NNF. We first build $[\![\neg\phi]\!]_k$, under the hypothesis that the existence of a loopback is not enforced.

Definition 5 (Translation of a PLTL formula on a bounded path). *The translation of a PLTL formula on a path π with bound k at time point i (with $k, l, i \in \mathbb{N}$ and $l < k$, $i \leq k$) is a propositional formula inductively defined as follows.*

$$\begin{array}{rcl rcl}
[\![q]\!]_k^i & \doteq & q^i & [\![f \wedge g]\!]_k^i & \doteq & [\![f]\!]_k^i \wedge [\![g]\!]_k^i \\
[\![\neg q]\!]_k^i & \doteq & \neg q^i & [\![f \vee g]\!]_k^i & \doteq & [\![f]\!]_k^i \vee [\![g]\!]_k^i \\
[\![\mathbf{X}f]\!]_k^i & \doteq & \begin{cases} \bot & i = k \\ [\![f]\!]_k^{i+1} & i < k \end{cases} & & & \\
[\![\mathbf{F}f]\!]_k^i & \doteq & \bigvee_{j\in[i,k]} [\![f]\!]_k^j & [\![f\mathbf{U}g]\!]_k^i & \doteq & \bigvee_{j\in[i,k]} \left([\![g]\!]_k^j \wedge \bigwedge_{h\in[i,j)} [\![f]\!]_k^h\right) \\
[\![\mathbf{G}f]\!]_k^i & \doteq & \bot & [\![f\mathbf{R}g]\!]_k^i & \doteq & \bigwedge_{j\in[i,k]} \left([\![g]\!]_k^j \vee \bigvee_{h\in[i,j)} [\![f]\!]_k^h\right) \\
[\![\mathbf{Y}f]\!]_k^i & \doteq & \begin{cases} \bot & i = 0 \\ [\![f]\!]_k^{i-1} & i > 0 \end{cases} & [\![\mathbf{Z}f]\!]_k^i & \doteq & \begin{cases} \top & i = 0 \\ [\![f]\!]_k^{i-1} & i > 0 \end{cases} \\
[\![\mathbf{O}f]\!]_k^i & \doteq & \bigvee_{j\in[0,i]} [\![f]\!]_k^j & [\![f\mathbf{S}g]\!]_k^i & \doteq & \bigvee_{j\in[0,i]} \left([\![g]\!]_k^j \wedge \bigwedge_{h\in(j,i]} [\![f]\!]_k^h\right) \\
[\![\mathbf{H}f]\!]_k^i & \doteq & \bigwedge_{j\in[0,i]} [\![f]\!]_k^j & [\![f\mathbf{T}g]\!]_k^i & \doteq & \bigwedge_{j\in[0,i]} \left([\![g]\!]_k^j \vee \bigvee_{h\in(j,i]} [\![f]\!]_k^h\right)
\end{array}$$

The translation $[\![f]\!]_k$ of a PLTL formula f on a path π with bound k is defined as $[\![f]\!]_k^0$.

The index i in $[\![f]\!]_k^i$ represent the time instant at which the formula is being evaluated. The structural rules reflect quite closely the compositional semantics presented in Definition 4. At each time point – recursively traversing the structure of the formula – the quantifications over time points can be unwound into boolean connectives, over the finite set of time points of interest. For instance, in the case of the $\mathbf{F}f$, the encoding at point i results in a disjunction over the time points from i to k of the encoding of f: in fact, $\mathbf{F}f$ holds at i iff we can produce a point in the (bounded) future such that f can be shown to hold in it. Likewise, we can show that $\mathbf{X}f$ holds at i if we can produce a future point where f holds. For this reason, $\mathbf{X}f$ is always false at k, since there is no "visible" future. The case of $\mathbf{G}f$ always reduces to $\perp$, because the above encoding can not show an infinite sequence of f. The cases for $\mathbf{U}$ and $\mathbf{R}$ follow Definition 4 as well.

Let us consider the case of past temporal operators. The formula $\mathbf{Y}f$ is encoded at i as the encoding of f at the previous time step $i-1$, if i is not initial, otherwise it reduces to $\perp$. The encoding for $\mathbf{Z}f$ only differs at the initial time point. The case of $\mathbf{O}f$ behaves similarly to $\mathbf{F}f$: we need to show that there is a point between 0 and i where f holds. The case for $\mathbf{H}f$ differs from the future case $\mathbf{G}f$ in that the past is finite. Therefore, it is enough to show that f holds in all the time points from i down to 0 to conclude that $\mathbf{H}f$ holds at time point i. Similar arguments apply to the case of $\mathbf{S}$ and $\mathbf{T}$.

5 Bounded Model Checking for PLTL with Loopbacks

We now tackle the problem of BMC for PLTL in its generality, by widening the scope of the encoding presented in the previous section to the case when the existence of a loopback at time l is assumed. We aim at finitely encoding into a formula ${}_l[\![f]\!]_k$ the semantics of a PLTL formula f on an infinite path with a cyclic structure.

5.1 The Problem

Consider the following simple example, depicted in Figure 1 (above), where a deterministic counter starts at 0, then increases its value until 5, and then restarts over from 2. The path $01 \cdot (2345)^\omega$ can be seen as a (6,2)-loop. In the future case, the encoding of a specification is based on the idea that, for every time in the encoding, exactly one successor time exists. To reach the successor of the time instant 5, we loop back to time 2. The encoding is formed structurally by analyzing the subformulae of the specification in the loop between k and l. The future LTL formulae enjoy the following properties: first, the evaluation of every pure-future LTL formula f at time i *only depends on time instants not preceding* i, i.e. only depends on the suffix π^i; second, $\pi^i = \pi^j$ for any two indexes i, j at the same position in the loop (i.e., in the same set $T_m \doteq \{m + np, n \in \mathbb{N}\}$, with $m \in [l, k)$). This is the reason the pure-future encoding works fine: the evaluation of a formula f on a (k, l)-loop π at each time point $i \geq k$ can be traced back to the evaluation of the same formula at a particular time point $i' < k$. In particular, the infinitely many time points in each set T_m on a (k, l)-loop are *equivalent*, in the sense that for every $m \in [l, k)$ and every $i, j \in T_m$, it is $(\pi, i) \models f$ iff $(\pi, j) \models f$.

Unfortunately, the idea of a simple lifting of the pure future construction of [5] to the past case breaks down immediately, as past formulas do not enjoy the above properties.

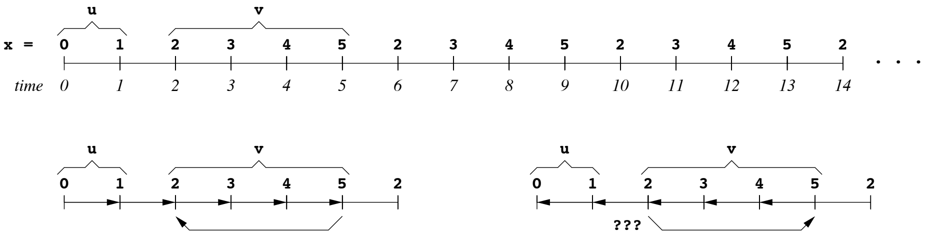


Fig. 1. An example of (6,2)-loop

First, when we progress backward in the past, at the point of loop back we have two possible predecessor points. In the case of the example, in order to encode $\mathbf{Y}(q)$ at point 2, we have to choose between progressing back through the loopback from point 2 to point 5 or moving to time 1 (see figure 1). Furthermore, we notice that this choice comes into play since the evaluation of a formula may depend on the past sequence $\pi_{|i}$, and $\pi_{|i} \neq \pi_{|j}$ whenever $i \neq j$. Consider for instance the formula

$$\mathbf{F}(x = 2 \wedge \mathbf{O}(x = 3 \wedge \mathbf{O}(x = 4 \wedge \mathbf{O}(x = 5))))$$

expressing that it is possible to reach a point where the values $\{2, 3, 4, 5\}$ of the counter occur in increasing order *in the past*. In the unbounded case, we need to get to the fourth occurrence of 2 in the path, i.e. at time 14, in order to show that $(x = 2 \wedge \mathbf{O}(x = 3 \wedge \mathbf{O}(x = 4 \wedge \mathbf{O}(x = 5))))$ holds. If we only look at the third occurrence (time 10), we can find previous points where the counter has values 3 and 4, but not 5. These issues are clearly relevant, since we are working on a bounded path representation. In order for the argument of $\mathbf{F}$ to evaluate to true, we will have to assume that we are far enough from the initial state, in order to progress in the past through the loop back a sufficient number of times. On the other hand, always choosing to progress in the past to the k-th step is not a viable option, since otherwise the encoding procedure might not terminate.

5.2 The Solution: Intuition

In order to propose a solution to this problem, we note that the evaluation of the formula $(x = 2 \wedge \mathbf{O}(x = 3 \wedge \mathbf{O}(x = 4 \wedge \mathbf{O}(x = 5))))$ is true in all the occurrences of $x = 2$ after the fourth, i.e. all the time points of the form $14 + 4i$. This is an example of the fact that a formula with past operators is able to discriminate its past, (i.e. among the number of times a loop has been traversed forwards), but only to a limited extent. Therefore, from a certain point on, it is useless to take into account more unrolls of the loop (i.e. to progress in the past by jumping from l to $k - 1$). The idea underlying our solution is to identify sets of points in which the evaluation stabilizes, and to deal with them at once. This is viable since the ability to refer to the past of a PLTL formula f is somehow predictable, once its syntactic structure is known. The key idea here is that every formula has a finite discriminating power for events in the past. So, when evaluated sufficiently far from the origin of time, a formula becomes unable to distinguish its past sequence from infinitely many other past sequences with a “similar” behaviour. The idea is then to collapse the undistinguishable versions of the past together into the same equivalence class. As we will see, only a finite number of such equivalence classes exists.

5.3 The Solution: Formalization

The intuition is captured by the notion of Past Temporal Horizon (PTH). Given a specific path, the PTH of a formula is the *minimal* number of loop unrolls after which the behaviour of the formula with respect to its truth value on π *stabilizes*, i.e. starts repeating in a cyclic way, according to the loop in the path. The PTH of a formula also provides a measure of the *maximal* amount of past a formula is able to take into account in a significant way along cyclic paths.

Definition 6. *The* past temporal horizon *(PTH) $\tau_{{}_l\pi_k}(f)$ of a PLTL formula f with respect to a (k,l)-loop π (with period $p = k - l$) is the smallest value $n \in \mathbb{N}$ such that*

$$\forall i \in [l,k) \quad ((\pi, i+np) \models f \text{ iff } (\forall n' > n \ (\pi, i+n'p) \models f)) \,.$$

We can abstract away the dependence of the PTH on a specific path, and give a notion of PTH which is *inherent* to the behaviour of a PLTL formula on a cyclic path, no matter which particular path is considered, nor even the structure of the loop.

Definition 7 (Past temporal horizon of a PLTL formula). *The* past temporal horizon *$\tau(f)$ of a PLTL formula f is defined as $\tau(f) \doteq \max_{\pi \in \Pi} \tau_{{}_l\pi_k}(f)$ where Π is the set of all the paths which are (k,l)-loops for some $k > l \geq 0$.*

The following theorem shows that a PLTL formula is guaranteed to have a finite PTH, and that an upper bound can be found based on its syntactic structure.

Theorem 1. *Let f and g be PLTL formulas. Then, it holds that:*

- $\tau(q) = 0$, *when* $q \in \mathcal{A}$ *and* $\tau(f) = \tau(\neg f)$*;*
- $\tau(\circ f) \leq \tau(f)$, *when* $\circ \in \{\mathbf{X}, \mathbf{F}, \mathbf{G}\}$
- $\tau(\circ f) \leq \tau(f) + 1$, *when* $\circ \in \{\mathbf{Z}, \mathbf{Y}, \mathbf{O}, \mathbf{H}\}$*;*
- $\tau(f \circ g) \leq \max(\tau(f), \tau(g))$, *when* $\circ \in \{\wedge, \vee, \mathbf{U}, \mathbf{R}\}$*;*
- $\tau(f \circ g) \leq \max(\tau(f), \tau(g)) + 1$, *when* $\circ \in \{\mathbf{S}, \mathbf{T}\}$*;*

This result (proved in [4] together with all the others) makes precise an unsurprising property of PLTL formulas: regardless of the particular path π, the ability of a formula in referring to the past along looping paths is bounded by its structural complexity. Put another way: when a formula is evaluated along a cyclic path, its truth value eventually starts looping as well. A delay in general exists between the starting points of these looping behaviours (the formula starts looping later than the path). An upper bound to this delay can be computed as a function of the syntactic structure of the formula itself.

The intuition behind the PTH is that it specifies the least number of times it is necessary to traverse the loop backwards before safely progressing towards the origin of time. In the following we provide the formal notions to deal with the idea of repeated unrolling of the (k,l)-loop. The intuition underlying the concept of *projection* is that each formula can be "safely" encoded on a finite representation (such as a (k,l)-loop) by suitably projecting the possibly infinite time interval the formula refers to onto its finite counterpart. The key difference with respect to the pure-future case, is that the projection also depends on the formula, not only on the shape of the path.

Definition 8. *We call* $\textsc{Lb}(n) \doteq l + np$ *the n-th left border of* π*,* $\textsc{Rb}(n) \doteq k + np$ *the n-th right border of* π*, and the interval* $\mathcal{M}(n) \doteq [0, \textsc{Rb}(n))$ *the* n-th main domain *of a* (k, l)*-loop. Let* $i \in \mathbb{N}$*. The projection of the point* i *in the n-th main domain of a* (k, l)*-loop is* $\rho_n(i)$*, defined as*

$$\rho_n(i) \doteq \begin{cases} i & i < \textsc{Rb}(n) \\ \rho_n(i-p) & \textit{otherwise} \end{cases}$$

Let $a, b \in \mathbb{N}$*, with* $a < b$*. The projection of the interval* $[a, b)$ *on the n-th main domain of a* (k, l)*-loop is*

$$\rho_n([a, b)) \doteq \{\rho_n(i) : i \in [a, b)\}.$$

We call $\textsc{Lb}(f) \doteq \textsc{Lb}(\tau(f))$ *and* $\textsc{Rb}(f) \doteq \textsc{Rb}(\tau(f))$ *the left and right borders of* f*, respectively, and* $\mathcal{M}(f) \doteq \mathcal{M}(\tau(f))$ *the main domain of* f*. The projections of the point* i *and of the interval* $[a, b)$ *onto the main domain of* f *are defined as* $\rho_f(i) \doteq \rho_{\tau(f)}(i)$ *and* $\rho_f([a, b)) \doteq \rho_{\tau(f)}([a, b))$ *respectively.*

While it is clear that the projection of a point onto the main domain of a function is still a point, it is not immediately evident what an interval is projected onto, as the projection of intervals is implicitly defined in terms of the projection function for time points. It is possible to explicitly characterize the resulting set of points as follows.

Lemma 1. *For an open interval* $[a, b)$*, it is*

$$\rho_n([a, b)) \doteq \begin{cases} \emptyset & \text{if } a = b, \text{ else} \\ [a, b) & \text{if } b < \textsc{Rb}(n), \text{ else} \\ [\min(a, \textsc{Lb}(n)), \textsc{Rb}(n)) & \text{if } b - a \geq p, \text{ else} \\ [\rho_n(a), \rho_n(b)) & \text{if } \rho_n(a) < \rho_n(b), \text{ else} \\ [\rho_n(a), \textsc{Rb}(n)) \cup [\textsc{Lb}(n), \rho_n(b)) & \end{cases}$$

This lemma shows that the projection of an interval is an interval in all but one case. It could seem that the conjunction of intervals in the last row of this lemma gives rise to a fragmentation of the interval-based representation. However, this apparent fragmentation disappears if we admit *extended intervals* of the form $[a, b)$ where b is possibly less than a (or even it is equal to ∞). With this position, we can re-write the last two rows of Lemma 1 in a single rule $[\rho_n(a), \rho_n(b))$ and generalize the notion of projection in such a way that the projection of an extended interval is always an extended interval.

Definition 9 (Extended projection). *Let* $[a, b)$ *be an extended interval. We define the* extended projection *of* $[a, b)$ *onto the n-th main domain of a* (k, l)*-loop as follows*

$$\rho^*_n([a, b)) \doteq \begin{cases} \rho^*_n([a, \max(a, \textsc{Rb}(n)) + p)) & b = \infty \\ \rho^*_n([a, b + p)) & b < a \\ \rho_n([a, b)) & \text{otherwise} \end{cases}$$

As before, we pose $\rho^*_f([a, b)) \doteq \rho^*_{\tau(f)}([a, b))$. The intuitive meaning of the projection $\rho^*_f([a, b))$ of the interval $[a, b)$ w.r.t. f, is that the finite set of time instants $\rho^*_f([a, b)) \subseteq \mathcal{M}(f)$ is the equivalent counterpart along cyclic paths of the (possibly infinite) interval $[a, b)$ for f. For example, one might wonder whether f is true at some time point in the (possibly infinite) interval $[a, b)$ of a loop path π. This happens if and only if f is true at some time point in the (always finite) interval $\rho^*_f([a, b))$.

Theorem 2. *For any PLTL formula f, any (k,l)-loop π, and any extended interval $[a,b)$, a point $i \in [a,b)$ such that $(\pi,i) \models f$ exists iff a point $i' \in \rho_f^*([a,b))$ exists such that $(\pi,i') \models f$.*

This argument can be specialized to the case when the interval contains only one point, by saying that on every (k,l)-loop path π and for every $i \geq 0$, $(\pi,i) \models f$ iff $(\pi,\rho_f(i)) \models f$. We now define the translation of a PLTL formula on a (k,l)-loop.

Definition 10 (Translation of a PLTL formula on a (k,l)-loop). *The translation of a PLTL formula on a (k,l)-loop π at time point i (with $k,l,i \in \mathbb{N}$ and $0 \leq l < k$) is a propositional formula inductively defined as follows.*

$$
\begin{aligned}
{}_{l}[\![q]\!]_k^i &\doteq q^{\rho_0(i)} & {}_{l}[\![f \wedge g]\!]_k^i &\doteq {}_{l}[\![f]\!]_k^{\rho_f(i)} \wedge {}_{l}[\![g]\!]_k^{\rho_g(i)} \\
{}_{l}[\![\neg q]\!]_k^i &\doteq \neg q^{\rho_0(i)} & {}_{l}[\![f \vee g]\!]_k^i &\doteq {}_{l}[\![f]\!]_k^{\rho_f(i)} \vee {}_{l}[\![g]\!]_k^{\rho_g(i)} \\
{}_{l}[\![\mathbf{X}f]\!]_k^i &\doteq {}_{l}[\![f]\!]_k^{\rho_f(i+1)}
\end{aligned}
$$

$$
\begin{aligned}
{}_{l}[\![\mathbf{F}f]\!]_k^i &\doteq \bigvee\nolimits_{j\in\rho_f^*([i,\infty))} {}_{l}[\![f]\!]_k^j \qquad {}_{l}[\![\mathbf{G}f]\!]_k^i \doteq \bigwedge\nolimits_{j\in\rho_f^*([i,\infty))} {}_{l}[\![f]\!]_k^j \\
{}_{l}[\![f\mathbf{U}g]\!]_k^i &\doteq \bigvee\nolimits_{j\in\rho_g^*([i,\infty))} \left({}_{l}[\![g]\!]_k^j \wedge \bigwedge\nolimits_{h\in\rho_f^*([i,j))} {}_{l}[\![f]\!]_k^h \right) \\
{}_{l}[\![f\mathbf{R}g]\!]_k^i &\doteq \bigwedge\nolimits_{j\in\rho_g^*([i,\infty))} \left({}_{l}[\![g]\!]_k^j \vee \bigvee\nolimits_{h\in\rho_f^*([i,j))} {}_{l}[\![f]\!]_k^h \right) \\
{}_{l}[\![\mathbf{Y}f]\!]_k^i &\doteq \begin{cases} \bot & i=0 \\ {}_{l}[\![f]\!]_k^{\rho_f(i-1)} & i>0 \end{cases} \qquad {}_{l}[\![\mathbf{Z}f]\!]_k^i \doteq \begin{cases} \top & i=0 \\ {}_{l}[\![f]\!]_k^{\rho_f(i-1)} & i>0 \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
{}_{l}[\![\mathbf{O}f]\!]_k^i &\doteq \bigvee\nolimits_{j\in\rho_f^*([0,i])} {}_{l}[\![f]\!]_k^j \qquad {}_{l}[\![\mathbf{H}f]\!]_k^i \doteq \bigwedge\nolimits_{j\in\rho_f^*([0,i])} {}_{l}[\![f]\!]_k^j \\
{}_{l}[\![f\mathbf{S}g]\!]_k^i &\doteq \bigvee\nolimits_{j\in\rho_g^*([0,i])} \left({}_{l}[\![g]\!]_k^j \wedge \bigwedge\nolimits_{h\in\rho_f^*((j,i])} {}_{l}[\![f]\!]_k^h \right) \\
{}_{l}[\![f\mathbf{T}g]\!]_k^i &\doteq \bigwedge\nolimits_{j\in\rho_g^*([0,i])} \left({}_{l}[\![g]\!]_k^j \vee \bigvee\nolimits_{h\in\rho_f^*((j,i])} {}_{l}[\![f]\!]_k^h \right)
\end{aligned}
$$

The translation of a PLTL formula f on a (k,l)-loop is defined as ${}_{l}[\![f]\!]_k \doteq {}_{l}[\![f]\!]_k^0$.

Notice how the encoding of each operator closely resembles the semantics of that operator (Definition 4). For example, the encoding rule ${}_{l}[\![\mathbf{F}f]\!]_k^i \doteq \bigvee_{j\in\rho_f^*([i,\infty))} {}_{l}[\![f]\!]_k^j$ is a quite straightforward interpretation of the semantic rule $(\pi,i) \models \mathbf{F}f$ iff $\exists j \in [i,\infty) \,.\, (\pi,j) \models f$, thanks to the introduction of the projection operator, which maps infinite sets of time points into equivalent but finite ones and shrinks finite intervals as much as possible, according to the upper bound given in Lemma 2.

Differently from the encoding for pure-future LTL given in [5], the above construction allows to evaluate the encoding of subformulas at time points greater than k. However, it is easy to see that no sub-formula f is encoded outside its main domain $\mathcal{M}(f)$. Furthermore, the encoding of any PLTL formula always results in a propositional formula with variables in $\{q^i .\, q \in \mathcal{A}, i \in [0,k)\}$, like in the pure-future case. While the encoding goes on from the root of the syntactic tree of the formula towards its leaves, the main domain of subformulas encountered along the way shrinks (the nesting depth of past operators cannot increase moving from a formula to its subformulas). When pure-future subformulas are reached the main domain is just $[0,k)$, and this is guaranteed to happen, since propositional leaves are pure-future formulas.

For example, in the case of the formula $\mathbf{F}(x = 2 \wedge \mathbf{O}(x = 3 \wedge \mathbf{O}(x = 4 \wedge \mathbf{O}(x = 5))))$ presented in Section 5.1, the encoding is able to perform a "virtual" unrolling of the (6,2)-path up to time 14 with no necessity of introducing more than 7 different states in the propositional encoding. The loop is virtually unrolled three times w.r.t. the subformula $x = 2 \wedge \mathbf{O}(x = 3 \wedge \mathbf{O}(x = 4 \wedge \mathbf{O}(x = 5))$, because this subformula has PTH equal to 3. The example confirms that this sufficies (and is necessary) to reach time 14 where the formula first evaluates to true. Inner subformulas have smaller and smaller virtual unrolling, as the PTH decreases. For example, though the sub-formula $(x = 4 \wedge \mathbf{O}(x = 5))$ needs to be evaluated up to time 13, it is explicitly evaluated only up to time 9 (PTH=1), and this suffices to catch all the variety of its behaviour, also comprising time 8 when the formula is true for the first of infinitely many subsequent times. The encoding is guaranteed to be correct by the following result.

Theorem 3. *For any PLTL formula f, a (k, l)-loop path π in M such that $\pi \models f$ exists iff $[\![M]\!]_k \wedge {}_lL_k \wedge {}_l[\![f]\!]_k$ is satisfiable.*

The computation of the PTH of a formula is not trivial in general. Therefore, we over-approximate it by means of the nesting depth of past operators in the formula.

Definition 11 (Past operator depth). *The* past operator depth $\delta(f)$ *of a PLTL formula f is defined as follows*

- $\delta(q) = 0$, *when* $q \in \mathcal{A}$, *and* $\delta(\circ f) = \delta(f)$, *when* $\circ \in \{\neg, \mathbf{X}, \mathbf{F}, \mathbf{G}\}$
- $\delta(f \circ g) = \max(\delta(f), \delta(g))$, *when* $\circ \in \{\wedge, \vee, \mathbf{U}, \mathbf{R}\}$*;*
- $\delta(\circ f) = \delta(f) + 1$, *when* $\circ \in \{\mathbf{Z}, \mathbf{Y}, \mathbf{O}, \mathbf{H}\}$*;*
- $\delta(f \circ g) = \max(\tau(f), \tau(g)) + 1$, *when* $\circ \in \{\mathbf{S}, \mathbf{T}\}$*;*

By comparing Theorem 1 and Definition 11, we obtain the following result, that guarantees the correctness of the resulting construction.

Lemma 2. *For any PLTL formula f, it is $\tau(f) \leq \delta(f)$.*

6 Implementation and evaluation

We implemented our PLTL bounded model checking algorithms within NuSMV [10, 11, 9], a state-of-the-art symbolic model checker designed as an open architecture integrating BDD-based and SAT-based model checking on the whole input language, and such that as many functionalities as possible are independent of the particular model

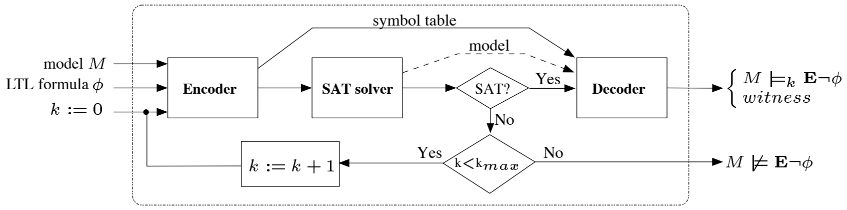

symbol table
model
model M
LTL formula ϕ
$k := 0$
Encoder
SAT solver
SAT?
Yes
Decoder
$\left\{ \begin{array}{l} M \models_k \mathbf{E}\neg\phi \\ witness \end{array} \right.$
No
$k := k + 1$
Yes
$k < k_{max}$
No
$M \not\models \mathbf{E}\neg\phi$

Fig. 2. An high-level view of the Bounded Model Checker module in NuSMV

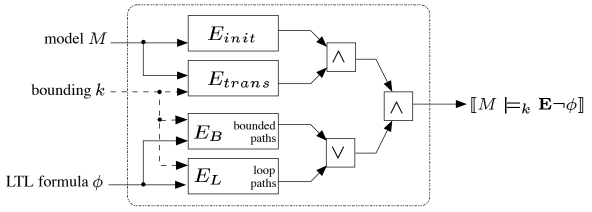


Fig. 3. The Encoder block, expanded from Figure 2

checking engine. NuSMV has been used for the verification of industrial designs, as a core for custom verification tools, and as a testbed for formal verification techniques. We benefit from its pre-processing abilities, that include parsing, flattening, boolean encoding, predicate encoding and cone of influence reduction (see [11] for details).

The BMC module was extended (see Figure 2) by enlarging its input language and implementing the new encoding. Only the encoder needs changes (see Figure 3), in particular within the sub-encoders E_B (for bounded paths, see Section 4) and E_L (for loop paths, see Section 5). Formulas represented as RBCs are produced, then converted into CNF and passed to the SAT solver. The optimizing techniques used within the RBC package, the CNF-ization procedure, the interface to the solver, the trace reconstruction sub-system, and the control system are inherited from the existing architecture.

We cannot take into account other systems to evaluate the effectiveness of our approach with an experimental comparison, as NuSMV appears to be the first system featuring past LTL operators: None of the available generic model checkers encompasses past operators[2], neither in a direct way (e.g.: like we do) nor in an indirect way (e.g.: by somehow pre-processing PLTL specifications). So, we push our analysis of past operators beyond the presented results, by preliminary investigating two alternative strategies for handling LTL and past operators within a BMC framework.

LTL model checking can be implemented via reduction to CTL model checking with fairness constraints, along the lines suggested in [12] for a BDD-based framework. The approach composes the model with the observer automaton [24] $M_{\neg\phi}$ corresponding to the negation of the specification, thus looking for unwanted behaviours of the model. If a fair path in the model-automaton composition is found, then the property is violated, and diagnostic information can be returned. This construction was recently extended to allow full-fledged BDD-based PLTL reasoning within NuSMV. We modified this construction by exploiting a SAT solver to look for fair paths. Two resulting scenarios can then be compared: in one case, the encoding block in Figure 3 is presented with the original model M and PLTL specification ϕ (encoded as explained in the previous sections). In the other case, the encoding block is presented with the composition of $M_{\neg\phi}$ with M, and the BMC machinery just searches for a fair path.

We experimented with these two alternatives. Figure 4 shows a sample comparison on the models “queue” and “syncarb5”, taken from the standard NuSMV distribution. Safety properties of the form $\mathbf{G}\phi$ (known to be true) are considered. None of the two approaches is dominant, even in case of pure-future specifications only: A tradeoff seems

[2] It is worth mentioning that past operators are sometimes used by specialized model checkers (such as BRUTUS[17], which only works in the verification of security protocols).

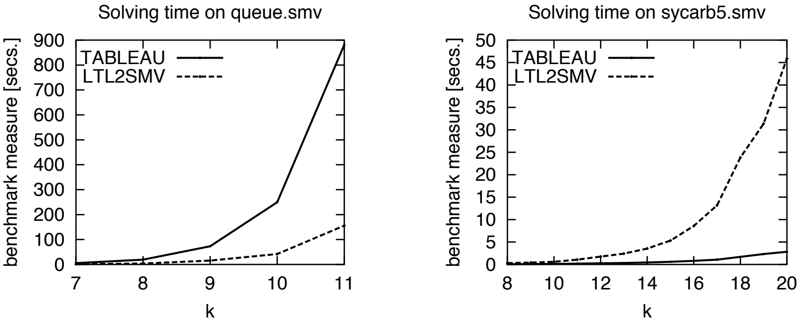


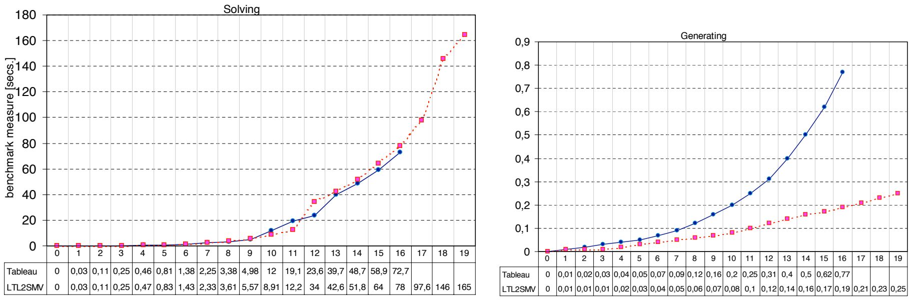
Fig. 4. Two instances showing that none of the approaches is dominating

to exist between the additional variables introduced with the model-automaton based approach to take into account the status of the observer, and the additional number of clauses produced by the implicit unwinding of loops in the other case. However, preliminary results suggest that the tableau-based construction often outperforms the automaton-based one, despite some cases where the opposite happens. Even though such tradeoff deserves further investigation, interesting features of the encoding for past operators can still be significantly evaluated. At least two advantages come from our approach w.r.t. the automaton-based one: first, the search for a fair path can lead to a needlessly long counterexample. Second, the virtual unrolling of loop paths is potentially able to discover counterexample (far) before the actual time the property fails to be true. In both cases, the time spent solving unnecessary instances is saved. As a very simple example of this advantage, we present the sender/receiver model "abp4"from the NuSMV distribution, checked against the false property:

$$\mathbf{G}(\texttt{sender.state=waitForAck} \rightarrow \mathbf{YH}\ \texttt{sender.state}{\neq}\texttt{waitForAck}).$$

Our encoding is able to produce a counterexample as soon as a wait state appears in the middle of the cyclic portion of a loop path, by unrolling in a virtual manner such a cyclic portion. Conversely, the observer automaton is forced to explicitly reach the second occurrence of the wait state. Figure 5 shows how this difference can be very significant also for not so shorter counterexamples: The automaton based approach finds a counterexample at length 19, while 16 is sufficient for the tableau. This leads to a clear advantage in terms of time, as the growth of the solving time is usually dominant.

The number of virtual unrolls necessary to exhibit a counterexample increase as the PLTL formula gets more complex, and the automaton-based approach is forced to reach further and further length to find a solution. Our encoding always "foresees" the consequences of a looping behaviour up the necessary point and never needs to explicitly produce and solve additional instances. The duty paid for this advantage is that more time is spent both on producing and on solving an instance of size k w.r.t. the analogous k-sized instance of the model-based approach. The additional solving time is usually very small (see Figure 5). The additional time for generation is eventually overcome by the solving time, even if for very small models it may be sensitive. Figure 5 shows that the additional generation time is completely negligible in our example.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Tableau	0	0,03	0,11	0,25	0,46	0,81	1,38	2,25	3,38	4,98	12	19,1	23,6	39,7	48,7	58,9	72,7			
LTL2SMV	0	0,03	0,11	0,25	0,47	0,83	1,43	2,33	3,61	5,57	8,91	12,2	34	42,6	51,8	64	78	97,6	146	165

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Tableau	0	0,01	0,02	0,03	0,04	0,05	0,07	0,09	0,12	0,16	0,2	0,25	0,31	0,4	0,5	0,62	0,77			
LTL2SMV	0	0,01	0,01	0,01	0,02	0,03	0,04	0,05	0,06	0,07	0,08	0,1	0,12	0,14	0,16	0,17	0,19	0,21	0,23	0,25

Fig. 5. The same instance dealt with in two ways

Though very preliminary, this experimental evaluation suggests that in addition to the increased complexity of the model, the unbounded approach may also require longer counterexamples, which in turn makes it necessary to solve harder SAT problems.

7 Conclusions

We tackle the problem of extending BMC to the case of LTL with Past Operators. We have shown that the task is not trivial in the case of loops: when traversing a path backward, we have to choose whether to proceed towards the origin or to jump “back to the future”. We have provided a formal account that allows us to solve the problem by projecting infinite sets of points into equivalent finite ones. Then, we have provided an effective tableau construction that encodes full PLTL into propositional formulae, and we showed that it is correct and complete. The formal treatment is the basis for the implementation of the technique in the NuSMV symbolic model checker. A preliminary experimental evaluation was discussed. In the future, we plan to extend and optimize the construction, and encompass verification problems in Requirement Engineering.

Acknowledgements

We thank Marco Pistore for his invaluable help in the definition of the framework and in the integration with NuSMV, and Dan Sheridan for reading early versions of the paper.

References

1. P. A. Abdullah, P. Bjesse, and N. Een. Symbolic Reachability Analysis based on SAT-Solvers. In *Sixth Int.nl Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS’00)*, 2000.
2. F. Bacchus and F. Kabanza. Control Strategies in Planning. In *Proc. of the AAAI Spring Symposium Series on Extending Theories of Action: Formal Theory and Practical Applications*, pages 5–10, Stanford University, CA, USA, March 1995.
3. J. Baumgartner, A. Kuehlmann, and J. Abraham. Property Checking via Structural Analysis. In *Proc. CAV’02*, volume 2404, pages 151–165, 2002.

4. M. Benedetti and A. Cimatti. Bounded Model Checking for Past LTL. Technical Report 0301-05, ITC-Irst, Trento, Italy, 2003.
5. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. *LNCS*, 1579:193–207, 1999.
6. A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic Model Checking Using SAT Procedures instead of BDDs. In *Proc. DAC'99*, pages 317–320, 1999.
7. A. Biere, E. Clarke, R. Raimi, and Y. Zhu. Verifying Safety Properties of a Power PC Microprocessor Using Symbolic Model Checking without BDDs. In *Proc CAV99*, volume 1633 of *LNCS*. Springer, 1999.
8. J. Castro, M. Kolp, and J. Mylopoulos. A Requirements-Driven Development Methodology. In *Proc. of the 13th Int.nl Conf. on Advanced Information Systems Engineering*, 2001.
9. A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. of Int.nl Conf. on Computer-Aided Verification (CAV 2002)*, 2002.
10. A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new Symbolic Model Verifier. In N. Halbwachs and D. Peled, editors, *Proceedings Eleventh Conference on Computer-Aided Verification (CAV'99)*, number 1633 in LNCS, pages 495–499, 1999.
11. A. Cimatti, E. Giunchiglia, M. Roveri, M. Pistore, R. Sebastiani, and A. Tacchella. Integrating BDD-based and SAT-based Symbolic Model Checking. In *Proceeding of 4th International Workshop on Frontiers of Combining Systems (FroCoS'2002)*, 2002.
12. E. Clarke, O. Grumberg, and K. Hamaguchi. Another Look at LTL Model Checking. *Formal Methods in System Design*, 10:47–71, 1997.
13. F. Copty, L. Fix, E. Giunchiglia, G. Kamhi, A. Tacchella, and M. Vardi. Benefits of Bounded Model Checking at an Industrial Setting. In *Proceedings of CAV 2001*, pages 436–453, 2001.
14. E.A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier Science Publisher B.V., 1990.
15. A. Fuxman. *Formal Analysis of Early Requirements Specifications*. PhD thesis, University of Toronto, Toronto, Canada, 2001.
16. Dov Gabbay. The Declarative Past and Imperative Future. In *Proccedings of the Colloquium on Temporal Logic and Specifications*, volume 398, pages 409–448. Springer-Verlag, 1987.
17. S. Gnesi, D. Latella, and G. Lenzini. Formal Verification of Cryptographic Protocols using History Dependent Automata. In *Proc. of the 4th Workshop on Sistemi Distribuiti: Algoritmi, Architetture e Linguaggi*, 1999.
18. O. Kupferman, N. Piterman, and M. Vardi. Extended Temporal Logic Revisited. In *Proc. 12th Int.nl Conf. on Concurrency Theory*, number 2154 in LNCS, pages 519–534, 2001.
19. F. Laroussinie and Ph. Schnoebelen. A Hierarchy of Temporal Logics with Past. *Theoretical Computer Science*, 148:303–324, 1995.
20. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proc. of the 38th Design Automation Conference*, 2001.
21. M. Sheeran, S. Singh, and G. Stalmarck. Checking safety properties using induction and a SAT-solver. In *Proc. Int.nl Conf. on Formal Methods in Computer-Aided Design*, 2000.
22. O. Shtrichmann. Tuning SAT Checkers for Bounded Model Checking. In *Proc. CAV'2000*, volume 1855 of *LNCS*. Springer, 2000.
23. A. van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *Proc. 5th IEEE International Symposium on Requirements Engineering*, pages 249–263, 2001.
24. M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Annual Symposium on Logic in Computer Science*, 1986.
25. P. F. Williams, A. Biere, E. M. Clarke, and A. Gupta. Combining Decision Diagrams and SAT Procedures for Efficient Symbolic Model Checking. In *Proc. CAV'2000*, volume 1855 of *LNCS*, pages 124–138. Springer, 2000.