




Past Matters: Supporting LTL+Past in the BLACK Satisfiability Checker

Luca Geatti   

University of Udine, Italy
Fondazione Bruno Kessler, Trento, Italy

Nicola Gigante   

Free University of Bozen-Bolzano, Italy

Angelo Montanari   

University of Udine, Italy

Gabriele Venturato   

University of Udine, Italy
KU Leuven, Belgium

Abstract

LTL+Past is the extension of Linear Temporal Logic (LTL) supporting *past* temporal operators. The addition of the past does not add expressive power, but does increase the usability of the language both in formal verification and in artificial intelligence, e.g., in the context of multi-agent systems. In this paper, we add the support of past operators to BLACK, a satisfiability checker for LTL based on a SAT encoding of a tree-shaped tableau system. We implement two ways of supporting the past in the tool. The first one is an equisatisfiable translation that removes the past operators, obtaining a future-only formula that can be solved with the original LTL engine. The second one extends the SAT encoding of the underlying tableau to directly support the tableau rules that deal with past operators. We describe both approaches and experimentally compare the two between themselves and with the nuXmv model checker, obtaining promising results.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics

Keywords and phrases SAT, LTL, LTL+Past, Tableaux

Digital Object Identifier 10.4230/LIPIcs.TIME.2021.8

Acknowledgements The open access publication of this article was supported by the Alpen-Adria-Universität Klagenfurt, Austria. Nicola Gigante acknowledges the partial support of the *TOTA* project “*Temporal Ontologies and Tableaux Algorithms*”. Gabriele Venturato acknowledges the partial support of the KU Leuven Research Fund (C14/18/062).

1 Introduction

Linear Temporal Logic (LTL) [20] is the de-facto standard temporal specification language in many areas including *formal verification* [8] and *artificial intelligence* [11]. Satisfiability checking, that is, deciding whether a given formula admits a model, is a particularly important problem because of its wide range of applications, and one of the first that have been studied [23, 26]. Many techniques and tools have been developed to solve it, ranging from tableau systems [1, 17, 22, 26] to reduction to model checking [5], from temporal resolution [9, 10, 14] to automata-theoretic techniques [16].

The *Bounded LTL sAtisfiability ChecKer*, BLACK for short, is a recently developed tool [12] that solves the satisfiability checking problem for LTL by providing a SAT encoding of the one-pass and tree-shaped tableau method for LTL proposed by Reynolds [22]. In an iterative procedure, the tree-shaped tableau is symbolically explored in a breadth-first way through a SAT encoding of its branches of depth at most k , for increasing values of k . The tool proved to be competitive with other state-of-the-art approaches [12].



© Luca Geatti, Nicola Gigante, Angelo Montanari, and Gabriele Venturato;
licensed under Creative Commons License CC-BY 4.0

28th International Symposium on Temporal Representation and Reasoning (TIME 2021).

Editors: Carlo Combi, Johann Eder, and Mark Reynolds; Article No. 8; pp. 8:1–8:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we extend BLACK to support LTL+Past, which enriches LTL with *past operators*, i.e. temporal modalities that talk about the past of the current time point. Although past operators do not add expressive power to the language, interestingly LTL+Past is exponentially more succinct than LTL, and it is thus able to express some useful properties in a more compact and natural way [19]. LTL+Past has been investigated from both theoretical and algorithmic viewpoints [13, 17, 18, 21], in the areas of formal verification [7] and artificial intelligence, e.g., in the context of multi-agent systems (see [3] and references therein).

We present and compare two different methods to support LTL+Past in BLACK.

The first one is a Tseitin-style [25] translation procedure that, given an LTL+Past formula, generates an *equisatisfiable* LTL formula that can be solved by the original LTL engine of BLACK, or, in principle, by any other tool for LTL satisfiability checking. Since the resulting formula is equisatisfiable, and not equivalent, to the original one, it can avoid the exponential blowup (it causes only a linear size increase). The core idea behind the translation is folklore, but, to the best of our knowledge, this is the first time it is explicitly worked out for LTL, implemented in a tool, and experimentally compared with other approaches.

The second method extends the SAT encoding [12] of the tableau system for LTL described in [13, 22] with the ability of directly handling *past* temporal operators. The resulting encoding successfully supports the claim given in [12, 13] that the one-pass and tree-shaped tableau system for LTL, together with its SAT encoding, can be easily extended to other temporal logics. Last but not least, our encoding for the support of past operators is much simpler than similar methods, like, for instance, the one based on *virtual unrollings* by Latvala et al. [15].

We make a comparison of the two methods, showing that the direct encoding outperforms the translation, although both methods show comparable performance. This makes the direct encoding the preferred way of handling the past in BLACK, but shows that the translation can be a useful and effective preprocessing step to deal with past operators in tools that do not support them natively. Since there is not a standardized set of benchmarks involving past operators in the literature, we introduce some novel sets of formulas for the sake of this comparison.

Finally, we compare both solutions with the nuXmv model checker, which is the only widely available tool, as far as we know, that directly supports past operators. The results are promising, showing BLACK to be competitive.

The paper is organized as follows. Section 2 introduces LTL+Past and Reynolds' tableau [22]. Then, Section 3 and Section 4 describe the translation and the direct encoding, respectively. Finally, Section 5 describes the results of the experimental comparison, and Section 6 concludes the paper.

2 Preliminaries

Let Σ be an alphabet of proposition letters. The syntax of an LTL+Past formula ϕ over Σ can be defined as follows:

$$\begin{array}{ll}
 \phi := p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid & \text{Boolean connectives} \\
 X\phi \mid \phi_1 \mathcal{U} \phi_2 \mid \phi_1 \mathcal{R} \phi_2 \mid & \text{future temporal operators} \\
 Y\phi \mid Z\phi \mid \phi_1 \mathcal{S} \phi_2 \mid \phi_1 \mathcal{T} \phi_2 & \text{past temporal operators}
 \end{array}$$

where $p \in \Sigma$ and ϕ , ϕ_1 , and ϕ_2 are LTL+Past formulas. LTL is the fragment that only uses Boolean connectives and future operators. We denote by $\text{LTL}[\Sigma]$ and $\text{LTL+Past}[\Sigma]$, respectively, the sets of LTL and LTL+Past formulas built over the alphabet Σ . Standard shorthands and derived operators are also available, such as $\top \equiv p \vee \neg p$, for some $p \in \Sigma$, $\perp \equiv \neg\top$, $F\phi \equiv \top \mathcal{U} \phi$, $G\phi \equiv \neg F\neg\phi$, $O\phi \equiv \top \mathcal{S} \phi$, $H\phi \equiv \neg O\neg\phi$.

LTL+Past is interpreted over infinite *state sequences* $\bar{\sigma} \in (2^\Sigma)^\omega$. Given a state sequence $\bar{\sigma} \in (2^\Sigma)^\omega$, the *satisfaction* of a formula ϕ by $\bar{\sigma}$ at a time point $i \geq 0$, denoted as $\bar{\sigma}, i \models \phi$, is defined as follows:

1. $\bar{\sigma}, i \models p$ iff $p \in \sigma_i$;
2. $\bar{\sigma}, i \models \neg\phi$ iff $\bar{\sigma}, i \not\models \phi$;
3. $\bar{\sigma}, i \models \phi_1 \vee \phi_2$ iff $\bar{\sigma}, i \models \phi_1$ or $\bar{\sigma}, i \models \phi_2$;
4. $\bar{\sigma}, i \models \phi_1 \wedge \phi_2$ iff $\bar{\sigma}, i \models \phi_1$ and $\bar{\sigma}, i \models \phi_2$;
5. $\bar{\sigma}, i \models X\phi$ iff $\bar{\sigma}, i + 1 \models \phi$;
6. $\bar{\sigma}, i \models Y\phi$ iff $i > 0$ and $\bar{\sigma}, i - 1 \models \phi$;
7. $\bar{\sigma}, i \models Z\phi$ iff either $i = 0$ or $\bar{\sigma}, i - 1 \models \phi$;
8. $\bar{\sigma}, i \models \phi_1 U \phi_2$ iff there exists $j \geq i$ such that $\bar{\sigma}, j \models \phi_2$,
and $\bar{\sigma}, k \models \phi_1$ for all k , with $i \leq k < j$;
9. $\bar{\sigma}, i \models \phi_1 S \phi_2$ iff there exists $j \leq i$ such that $\bar{\sigma}, j \models \phi_2$,
and $\bar{\sigma}, k \models \phi_1$ for all k , with $j < k \leq i$;
10. $\bar{\sigma}, i \models \phi_1 R \phi_2$ iff either $\bar{\sigma}, j \models \phi_2$ for all $j \geq i$, or there exists
 $k \geq i$ such that $\bar{\sigma}, k \models \phi_1$ and
 $\bar{\sigma}, j \models \phi_2$ for all $i \leq j \leq k$;
11. $\bar{\sigma}, i \models \phi_1 T \phi_2$ iff either $\bar{\sigma}, j \models \phi_2$ for all $0 \leq j \leq i$, or there exists
 $k \leq i$ such that $\bar{\sigma}, k \models \phi_1$ and
 $\bar{\sigma}, j \models \phi_2$ for all $i \geq j \geq k$

We say that a state sequence $\bar{\sigma}$ satisfies ϕ , written $\bar{\sigma} \models \phi$, if $\bar{\sigma}, 0 \models \phi$. Observe that the \wedge connective, the *release* operator ($\phi_1 R \phi_2$), the *triggered* operator ($\phi_1 T \phi_2$), and the *weak yesterday* operator ($Z\phi$) can be defined in terms of the \vee connective, the *until* operator ($\phi_1 U \phi_2$), the *since* operator ($\phi_1 S \phi_2$), and the *yesterday* operator ($Y\phi$), respectively, but here we consider them as primitive operators since this allows us to put any formula into *negation normal form* (NNF), which will be useful later. Moreover, note that state sequences have a definite starting point, hence *the past is bounded*, and we need to distinguish between the *yesterday* operator ($Y\phi$, ϕ holds at the previous state) and the *weak yesterday* operator ($Z\phi$, ϕ holds at the previous state, if it exists) as opposed to a single *tomorrow* operator ($X\phi$, ϕ holds at the next state).

The notion of *closure* of a formula will be useful later.

► **Definition 1** (Closure of an LTL+Past formula). *Let ψ be an LTL+Past formula built over Σ . The closure of ψ is the smallest set of formulas $\mathcal{C}(\psi)$ satisfying the following properties:*

1. $\psi \in \mathcal{C}(\psi)$;
2. for each sub-formula ψ' of ψ , $\psi' \in \mathcal{C}(\psi)$;
3. for each $p \in \Sigma$, $p \in \mathcal{C}(\psi)$ if and only if $\neg p \in \mathcal{C}(\psi)$;
4. if $\psi_1 U \psi_2 \in \mathcal{C}(\psi)$, then $X(\psi_1 U \psi_2) \in \mathcal{C}(\psi)$;
5. if $\phi_1 R \psi_2 \in \mathcal{C}(\psi)$, then $X(\psi_1 R \phi_2) \in \mathcal{C}(\psi)$;
6. if $\psi_1 S \psi_2 \in \mathcal{C}(\psi)$, then $Y(\psi_1 S \psi_2) \in \mathcal{C}(\psi)$;
7. if $\psi_1 T \psi_2 \in \mathcal{C}(\psi)$, then $Z(\psi_1 T \psi_2) \in \mathcal{C}(\psi)$.

It is worth pointing out that item 3 of Definition 1 only applies to proposition letters because formulas are assumed to be in NNF.

The one-pass and tree-shaped tableau for LTL+Past

Let us now briefly describe the tableau system for LTL+Past introduced by Geatti et al. [13], which will be used as the basis for the direct encoding discussed in Section 4. It extends the tableau system for LTL by Reynolds [22]. The latter has the distinctive features of being *tree-shaped*, as opposed to standard graph-shaped LTL tableaux, e.g., [17], and *one-pass*, since a single pass is sufficient to either accept or reject a given branch.

■ **Table 1** Tableau expansion rules. When a formula ϕ of one of the types shown in the table is found in the label Γ of a node u , one or two children u' and u'' are created with the same label as u , but replacing ϕ by the formulas from $\Gamma_1(\phi)$ and $\Gamma_2(\phi)$, respectively.

| Rule | $\phi \in \Gamma$ | $\Gamma_1(\phi)$ | $\Gamma_2(\phi)$ |
|-------------|----------------------------|---------------------|--|
| DISJUNCTION | $\alpha \vee \beta$ | $\{\alpha\}$ | $\{\beta\}$ |
| CONJUNCTION | $\alpha \wedge \beta$ | $\{\alpha, \beta\}$ | |
| UNTIL | $\alpha \mathcal{U} \beta$ | $\{\beta\}$ | $\{\alpha, \mathbf{X}(\alpha \mathcal{U} \beta)\}$ |
| SINCE | $\alpha \mathcal{S} \beta$ | $\{\beta\}$ | $\{\alpha, \mathbf{Y}(\alpha \mathcal{S} \beta)\}$ |
| RELEASE | $\alpha \mathcal{R} \beta$ | $\{\alpha, \beta\}$ | $\{\beta, \mathbf{X}(\alpha \mathcal{R} \beta)\}$ |
| TRIGGERED | $\alpha \mathcal{T} \beta$ | $\{\alpha, \beta\}$ | $\{\beta, \mathbf{Z}(\alpha \mathcal{T} \beta)\}$ |

For ease of exposition, *w.l.o.g.* we assume formulas to be in NNF. A tableau for a formula ϕ is a tree where each node u is labeled by a set of formulas $\Gamma(u)$, with the root u_0 labeled with $\Gamma(u_0) = \{\phi\}$. At each step, a set of rules is applied to a leaf, until all branches have been either *accepted* or *rejected*. Each rule either adds one or more children to the current leaf or either accept or reject the current branch. Given a branch $\bar{u} = \langle u_0, \dots, u_n \rangle$, the sequence of nodes $\langle u_i, \dots, u_j \rangle$, for some $0 \leq i \leq j \leq n$, is denoted by $\bar{u}_{[i,j]}$.

At each step, the selected node is subject to a number of *expansion rules*, that select a formula of the label and expand it according to its semantics, as reported in Table 1. Each expansion rule creates one or two children depending on the selected formula. After repeated applications of the expansion rules, a node that only contains *elementary* formulas, that is, propositions, *tomorrow*, *yesterday*, or *weak yesterday* formulas, is obtained (*poised* node). Elementary formulas of the form $\mathbf{X}(\phi_1 \mathcal{U} \phi_2)$ are called *X-eventualities*. An X-eventuality is a formula that, intuitively, requests something to be fulfilled later. Given an X-eventuality $\phi \equiv \mathbf{X}(\phi_1 \mathcal{U} \phi_2)$, ϕ is said to be *fulfilled* in a node u if $\phi_2 \in \Gamma(u)$.

The tableau advances through time by making *temporal steps*. To do that, the following rules are applied to poised nodes.

STEP A child u_{n+1} is added to u_n , with:

$$\Gamma(u_{n+1}) = \{\alpha \mid \mathbf{X}\alpha \in \Gamma(u_n)\}$$

FORECAST Let

$$G_n = \left\{ \alpha \in \mathcal{C}(\phi) \mid \begin{array}{l} \mathbf{Y}\alpha \in \mathcal{C}(\psi) \text{ or} \\ \mathbf{Z}\alpha \in \mathcal{C}(\psi) \text{ for some } \psi \in \Gamma(u_n) \end{array} \right\}$$

For each subset $G'_n \subseteq G_n$ (including \emptyset), a child u'_n is added to u_n such that $\Gamma(u'_n) = \Gamma(u_n) \cup G'_n$. This is done once and only once before every application of the STEP rule.

The STEP rule advances the construction of the current branch to the subsequent temporal state. The FORECAST is essential to the well-functioning of the rule dealing with *past*, as it adds a number of branches that nondeterministically guess formulas that may be needed to fulfill past requests coming from future states. For details on the FORECAST rule, we refer the reader to Geatti et al. [13].

Since the STEP rule is not applied to all the poised nodes (to some of which the FORECAST rule is applied instead), we need the following definition.

► **Definition 2** (Step node). *In a complete tableau for an LTL+Past formula, a poised node u_n is a step node if it is either a poised leaf or a poised node to which the STEP rule was applied.*

Given a node u , we define u^* as the closest ancestor of u that is child of a step node, if any. $\Gamma^*(u)$ is the union of the labels of the nodes from u to u^* or to the root, if u^* does not exist.

Before applying the STEP rule though, poised nodes are subject to the application of a few *termination rules*, that is, rules that decide whether the construction has to continue or the current branch has to be either rejected or accepted. Given a branch $\bar{u} = \langle u_0, \dots, u_n \rangle$, with u_n a step node, the termination rules are the following.

CONTRADICTION If $\{p, \neg p\} \subseteq \Gamma(u_n)$, for some $p \in \Sigma$, then \bar{u} is *rejected*.

EMPTY If $\Gamma(u_n) = \emptyset$, then \bar{u} is *accepted*.

YESTERDAY If $\Upsilon\alpha \in \Gamma(u_n)$, then the branch \bar{u} is *rejected* if either u_n^* does not exist or $Y_n \not\subseteq \Gamma^*(u_n^*)$, where $Y_n = \{\psi \mid \Upsilon\psi \in \Gamma(u_n)\}$.

W-YESTERDAY If $Z\alpha \in \Gamma(u_n)$, then \bar{u} is *rejected* if u_n^* exists and $Z_n \not\subseteq \Gamma^*(u_n^*)$, where $Z_n = \{\psi \mid Z\psi \in \Gamma(u_n)\}$.

LOOP If there exists a position $i < n$ such that $\Gamma(u_i) = \Gamma(u_n)$ and all the X- eventualities requested in u_i are fulfilled in $\bar{u}_{[i+1..n]}$, then \bar{u} is *accepted*.

PRUNE If there exist two positions i and j such that $i < j \leq n$, $\Gamma(u_i) = \Gamma(u_j) = \Gamma(u_n)$, and all the X- eventualities requested in these nodes which are fulfilled in $\bar{u}_{[j+1..n]}$ are also fulfilled in $\bar{u}_{[i+1..j]}$, then \bar{u} is *rejected*.

Intuitively, the **CONTRADICTION**, **YESTERDAY**, and **W-YESTERDAY** rules reject branches that contain some contradiction, either a propositional one or because of some unfulfilled past request. The **EMPTY** rule accepts a branch devoid of contradictions where there is nothing left to do, while the **LOOP** one accepts a looping branch where all the X- eventualities are proposed again and fulfilled at every repetition of the loop. Finally, the **PRUNE** rule, which was the main novelty of the system when introduced by Reynolds [22], rejects a branch that, otherwise, is going to be infinitely unrolled because of an X- eventuality impossible to fulfill.

3 The translation

In this section, we define a procedure which takes as input an LTL+Past formula, and returns as output an *equisatisfiable* LTL formula. The increase in size is only linear, thus avoiding the exponential blowup of the worst-case complexity of the translation of an LTL+Past formula into an equivalent (and not simply an equisatisfiable) LTL one [19]. The idea behind the translation is simple, but this is the first time, as far as we know, that it has been actually implemented and experimentally compared with other approaches to support past operators.

The key idea is to replace past subformulas by fresh proposition letters that are forced to replicate the semantics of past operators with ad-hoc axioms. Even though the produced formula is not equivalent to the original one, the proposed translation procedure allows us to easily recover a model of the original formula, if it is satisfiable in the first place, by simply discarding the additional proposition letters. Note that we will define the translation *without* assuming formulas to be in NNF.

To begin, we define two functions, τ and θ , which are the building blocks of the translation. The function τ replaces past formulas with the corresponding placeholder proposition letters, while θ enriches the formula with the axioms to force those letters to behave correctly.

Let Σ be the alphabet of proposition letters of the input formula ϕ . The alphabet of the output formula is $\Sigma_+ = \Sigma \cup \Sigma_{past}$, where Σ_{past} is the set of fresh proposition letters introduced by τ . We recursively define the function $\tau : \text{LTL+Past}[\Sigma] \rightarrow \text{LTL}[\Sigma_+]$ as follows:

$$\begin{aligned}
\tau(p) &= p && \text{where } p \in \Sigma \\
\tau(\neg\phi) &= \neg\tau(\phi) \\
\tau(\mathbf{X}\phi) &= \mathbf{X}\tau(\phi) \\
\tau(\phi_1 \otimes \phi_2) &= \tau(\phi_1) \otimes \tau(\phi_2) && \text{where } \otimes \in \{\wedge, \vee, \mathcal{U}, \mathcal{R}\} \\
\tau(\mathbf{Y}\phi) &= p_{\mathbf{Y}\tau(\phi)} \\
\tau(\mathbf{Z}\phi) &= p_{\mathbf{Z}\tau(\phi)} \\
\tau(\phi_1 \mathcal{S} \phi_2) &= p_{\tau(\phi_1)\mathcal{S}\tau(\phi_2)} \\
\tau(\phi_1 \mathcal{T} \phi_2) &= \tau(\neg(\neg\phi_1 \mathcal{S} \neg\phi_2))
\end{aligned}$$

The function $\Theta : \text{LTL}[\Sigma_+] \rightarrow 2^{\text{LTL}[\Sigma_{++}]}$, where $\Sigma_{++} = \Sigma_+ \cup \{p_{\mathbf{Y}p_\psi} \mid p_\psi \in \Sigma_{past}, \psi \equiv \phi_1 \mathcal{S} \phi_2\}$, produces a set of formulas which gives the appropriate semantics to the proposition letters in Σ_{past} . It is defined as follows:

$$\begin{aligned}
\Theta(p) &= \emptyset && \text{where } p \in \Sigma \\
\Theta(\otimes \phi) &= \Theta(\phi) && \text{where } \otimes \in \{\neg, \mathbf{X}\} \\
\Theta(\phi_1 \otimes \phi_2) &= \Theta(\phi_1) \cup \Theta(\phi_2) && \text{where } \otimes \in \{\wedge, \vee, \mathcal{U}, \mathcal{R}\} \\
\Theta(p_{\mathbf{Y}\phi}) &= \{Y_{p_{\mathbf{Y}\phi}}\} \cup \Theta(\phi) && \text{where } Y_{p_{\mathbf{Y}\phi}} \equiv \neg p_{\mathbf{Y}\phi} \wedge \mathbf{G}(\mathbf{X}p_{\mathbf{Y}\phi} \leftrightarrow \phi) \\
\Theta(p_{\mathbf{Z}\phi}) &= \{Z_{p_{\mathbf{Z}\phi}}\} \cup \Theta(\phi) && \text{where } Z_{p_{\mathbf{Z}\phi}} \equiv p_{\mathbf{Z}\phi} \wedge \mathbf{G}(\mathbf{X}p_{\mathbf{Z}\phi} \leftrightarrow \phi) \\
\Theta(p_\psi) &= \{S_{p_\psi}, Y_{p_{\mathbf{Y}p_\psi}}\} \cup \Theta(\phi_1) \cup \Theta(\phi_2)
\end{aligned}$$

where $\psi \equiv \phi_1 \mathcal{S} \phi_2$ and

$$S_{p(\phi_1 \mathcal{S} \phi_2)} \equiv \mathbf{G}(p(\phi_1 \mathcal{S} \phi_2) \leftrightarrow (\phi_2 \vee (\phi_1 \wedge p_{\mathbf{Y}p(\phi_1 \mathcal{S} \phi_2)})))$$

The first three cases are pretty straightforward. The last three, which are the core of the whole translation, are, instead, more involved.

As for the *yesterday* operator, we state that if the argument ϕ of the yesterday operator is true in a certain state, then in the next state $\mathbf{Y}\phi$ is true. Note that we force the proposition letter $p_{\mathbf{Y}\phi}$ to be *false* in the initial state, because $\mathbf{Y}\phi$ cannot be true in that state. The *weak yesterday* operator behaves almost the same. However, according to its semantics, $\mathbf{Z}\phi$ is always true at the initial state, and thus we constrain the proposition letter $p_{\mathbf{Z}\phi}$ to hold at the initial state.

Let us consider now the *since* operator. By exploiting its semantics and the corresponding *expansion rule* defined for the tableau in Table 1, we say – in the scope of the *always* operator – that $\phi_1 \mathcal{S} \phi_2$ is true at a certain state if and only if, at that state, either ϕ_2 is true, or ϕ_1 is true and $\phi_1 \mathcal{S} \phi_2$ was true at the previous state. However, this is not enough to capture the intended semantics, because we have introduced a new symbol, that is, $p_{\mathbf{Y}p_\psi}$, and we need to force the (yesterday) semantics also for it. This can be done exactly as before.

The function $\theta : \text{LTL}[\Sigma_+] \rightarrow \text{LTL}[\Sigma_{++}]$ wraps a formula with the semantics of the additional proposition letters. It is formally defined as follows:

$$\theta(\phi) = \phi \wedge \bigwedge_{\psi \in \Theta(\phi)} \psi$$

The proposed translation procedure is simply the *function composition* of θ and τ .

► **Definition 3** (REMOVEPAST). *The function $\text{REMOVEPAST} : \text{LTL+Past}[\Sigma] \rightarrow \text{LTL}[\Sigma_{++}]$ is defined as follows: $\text{REMOVEPAST} = \theta \circ \tau$.*

It is possible to prove that the above translation results in an *equisatisfiable* formula.

► **Theorem 4.** *Let ϕ be an LTL+Past formula. Then, $\text{REMOVEPAST}(\phi)$ is an LTL formula equisatisfiable with ϕ .*

Proof. Since the input formula ϕ is finite, the procedure $\text{REMOVEPAST}(\phi)$ always terminates. We have to show that $\text{REMOVEPAST}(\phi)$ is satisfiable if and only if ϕ is satisfiable.

(\leftarrow) Let $\bar{\sigma}$ be the model which satisfies ϕ , i.e. $\bar{\sigma} \models \phi$. Let us show that there exists $\bar{\sigma}'$ such that $\bar{\sigma}' \models \text{REMOVEPAST}(\phi)$, where, for each $i \geq 0$ and each sub-formula ψ of ϕ , $\bar{\sigma}'$ is defined as follows.

1. $p \in \sigma'_i$ iff $\bar{\sigma} \models_i p$ for $p \in \Sigma$
2. $\tau(Y\psi) \in \sigma'_i$ iff $i > 0$ and $\bar{\sigma}' \models_{i-1} \tau(\psi)$
3. $\tau(Z\psi) \in \sigma'_i$ iff either $i = 0$ or $\bar{\sigma}' \models_{i-1} \tau(\psi)$
4. $\tau(\psi_1 \mathcal{S} \psi_2) \in \sigma'_i$ iff there exists $0 \leq j \leq i$ such that
 $\bar{\sigma}' \models_j \tau(\psi_2)$, and $\bar{\sigma}' \models_k \tau(\psi_1)$
for all k , with $j < k \leq i$

By the definition above, we can prove by induction on the structure of ϕ , that $\bar{\sigma}' \models \tau(\phi)$. Moreover, by Item 2 we have that, for each *yesterday* sub-formula ψ of ϕ , $\bar{\sigma}' \models Y_{\tau(\psi)}$, because with Table 2 we force a step-wise consistency between a *yesterday* formula and its request at the previous state, in $\bar{\sigma}'$, which is exactly what is stated by $Y_{\tau(\psi)}$. Similarly, by Item 2 and Item 3, we also have that $\bar{\sigma}' \models Z_{\tau(\psi)}$ and $\bar{\sigma}' \models S_{\tau(\psi)}$ for, respectively, each *weak yesterday* and each *since* sub-formula ψ of ϕ . This means that $\bar{\sigma}'$ satisfies the conjunction of the three formulas, hence $\bar{\sigma}' \models \bigwedge_{\psi \in \Theta(\tau(\phi))} \psi$. Thus, $\bar{\sigma}' \models \theta(\tau(\phi))$. This allows us to conclude that $\bar{\sigma}' \models \text{REMOVEPAST}(\phi)$.

(\rightarrow) Given a model $\bar{\sigma}'$ such that $\bar{\sigma}' \models \text{REMOVEPAST}(\phi)$, we can easily build $\bar{\sigma}$ for ϕ by setting that $p \in \sigma_i$ iff $\bar{\sigma}' \models_i p$ for all $p \in \Sigma$. By induction on the structure of ϕ , using the semantics of past operators stated by $Y_{\tau(\psi)}$, $Z_{\tau(\psi)}$, $S_{\tau(\psi)}$, we can prove that $\bar{\sigma} \models \phi$. ◀

4 The direct encoding

The BLACK satisfiability checker is based on an iterative procedure that symbolically explores the tableau tree breadth-first by means of a SAT encoding of the tableau branches up to a given depth k , for increasing values of k . The satisfiability checking procedure employed by BLACK is reported in Algorithm 1 [12]. The three formulas $\llbracket \phi \rrbracket^k$, $|\phi|^k$, and $|\phi|_T^k$ encode different rules of the tableau. This section shows how to extend them to support past operators by encoding the tableau rules recalled in Section 2.

Let us start with some notation. Let ϕ be an LTL+Past formula in NNF over the alphabet Σ . We define the following sets of formulas:

$$\begin{aligned} \text{XR} &= \{\psi \in \mathcal{C}(\phi) \mid \psi \text{ is a } \textit{tomorrow} \text{ formula}\} \\ \text{YR} &= \{\psi \in \mathcal{C}(\phi) \mid \psi \text{ is a } \textit{yesterday} \text{ formula}\} \\ \text{ZR} &= \{\psi \in \mathcal{C}(\phi) \mid \psi \text{ is a } \textit{weak yesterday} \text{ formula}\} \\ \text{XEV} &= \{\psi \in \mathcal{C}(\phi) \mid \psi \text{ is an } \textit{X-eventuality}\} \end{aligned}$$

The three encoding formulas are defined over an extended alphabet $\bar{\Sigma}$, which includes:

1. any proposition letter from the original alphabet Σ ;
2. the set $\{p_\psi \mid \psi \in \text{XR}, \text{YR}, \text{ZR}\}$, that is, the set of all the *grounded X-, Y-, and Z-requests*;
3. a *stepped* version p^k of all the proposition letters defined in items 1 and 2, with $k \in \mathbb{N}$ and p^0 identified as p .

■ **Algorithm 1** BLACK's main procedure [12].

```

1: procedure BLACK( $\phi$ )
2:    $k \leftarrow 0$ 
3:   while True do
4:     if  $\llbracket \phi \rrbracket^k$  is UNSAT then
5:       return  $\phi$  is UNSAT
6:     end if
7:     if  $|\phi|^k$  is SAT then
8:       return  $\phi$  is SAT
9:     end if
10:    if  $|\phi|_T^k$  is UNSAT then
11:      return  $\phi$  is UNSAT
12:    end if
13:     $k \leftarrow k + 1$ 
14:  end while
15: end procedure

```

Intuitively, different stepped versions of the same proposition letter p are used to represent the value of p at different states. Thus, when p^i holds, it means that p holds at i -th step node of the branch, i.e. the i -th state of the model.

Moreover, given $\psi \in \mathcal{C}(\phi)$, we denote by ψ_G the formula in which all the X-, Y-, and Z-requests are replaced by their grounded version. Similarly, given $\psi \in \mathcal{C}(\phi)$, we denote by ψ^k the formula in which all proposition letters are replaced by their k stepped version. We write ψ_G^k to denote $(\psi_G)^k$.

The formula $\llbracket \phi \rrbracket^k$ is called the k -unraveling of ϕ , and encodes the expansion of the tableau tree. To define it, we need to encode the expansion rules of Table 1.

► **Definition 5 (Stepped Normal Form).** *Given an LTL+Past formula ϕ in NNF, its stepped normal form, denoted by $\text{snf}(\phi)$, is defined as follows:*

$$\begin{aligned}
\text{snf}(\ell) &= \ell && \text{where } \ell \in \{p, \neg p\}, \text{ for } p \in \Sigma \\
\text{snf}(\otimes \phi_1) &= \otimes \phi_1 && \text{where } \otimes \in \{X, Y, Z\} \\
\text{snf}(\phi_1 \otimes \phi_2) &= \text{snf}(\phi_1) \otimes \text{snf}(\phi_2) && \text{where } \otimes \in \{\wedge, \vee\} \\
\text{snf}(\phi_1 \mathcal{U} \phi_2) &= \text{snf}(\phi_2) \vee (\text{snf}(\phi_1) \wedge X(\phi_1 \mathcal{U} \phi_2)) \\
\text{snf}(\phi_1 \mathcal{R} \phi_2) &= \text{snf}(\phi_2) \wedge (\text{snf}(\phi_1) \vee X(\phi_1 \mathcal{R} \phi_2)) \\
\text{snf}(\phi_1 \mathcal{S} \phi_2) &= \text{snf}(\phi_2) \vee (\text{snf}(\phi_1) \wedge Y(\phi_1 \mathcal{S} \phi_2)) \\
\text{snf}(\phi_1 \mathcal{T} \phi_2) &= \text{snf}(\phi_2) \wedge (\text{snf}(\phi_1) \vee Z(\phi_1 \mathcal{T} \phi_2))
\end{aligned}$$

The stepped normal form is the extension to past operators of the *next normal form* used by Geatti et al. [13]. It can be noted how it follows the expansion rules of each operator in Table 1. We can now define the k -unraveling of ϕ recursively as follows:

$$\begin{aligned}
\llbracket \phi \rrbracket^0 &= \text{snf}(\phi)_G \wedge \bigwedge_{\psi \in \text{YR}} \neg \psi_G \wedge \bigwedge_{\psi \in \text{ZR}} \psi_G \\
\llbracket \phi \rrbracket^{k+1} &= \llbracket \phi \rrbracket^k \wedge S_k \wedge Y_k \wedge Z_k \\
S_k &\equiv \bigwedge_{X\alpha \in \text{XR}} \left((X\alpha)_G^k \leftrightarrow \text{snf}(\alpha)_G^{k+1} \right),
\end{aligned}$$

$$Y_k \equiv \bigwedge_{Y\alpha \in YR} \left((Y\alpha)_G^{k+1} \leftrightarrow \text{snf}(\alpha)_G^k \right), \quad Z_k \equiv \bigwedge_{Z\alpha \in ZR} \left((Z\alpha)_G^{k+1} \leftrightarrow \text{snf}(\alpha)_G^k \right)$$

The S_k , Y_k and Z_k formulas encode, respectively, the STEP, YESTERDAY, and W-YESTERDAY rules of the tableau, while the base case of the 0-unraveling ensures that *yesterday* formulas are false and *weak yesterday* formulas are true at the first state. The CONTRADICTION rule of the tableau is implicitly encoded in the fact that only satisfying assignments of the formula are considered. Note that the FORECAST rule as well does not need to be explicitly encoded: the intrinsic nondeterminism of the SAT solving process accounts for the nondeterministic choices implemented by the rule.

Intuitively, if $\llbracket \phi \rrbracket^k$ is unsatisfiable, all the branches of the tableau for ϕ are rejected before $k + 1$ steps.

► **Lemma 6.** *Let ϕ be an LTL+Past formula. Then, $\llbracket \phi \rrbracket^k$ is unsatisfiable if and only if all the branches of the complete tableau for ϕ are crossed by the CONTRADICTION or (W-)YESTERDAY rules and contain at most $k + 1$ step nodes.*

Proof. We prove the contrapositive, i.e. that $\llbracket \phi \rrbracket^k$ is satisfiable if and only if the complete tableau for ϕ has at least a branch that is either accepted, crossed by PRUNE, or longer than $k + 1$ step nodes. To do that we establish a connection between truth assignments of $\llbracket \phi \rrbracket^k$ and suitable branches of the tableau.

From branches to assignments. Let $\bar{u} = \langle u_0, \dots, u_n \rangle$ be a branch that is either accepted, crossed by PRUNE, or longer than $k + 1$ step nodes. Let $\bar{\pi} = \langle \pi_0, \dots, \pi_m \rangle$ be the sequence of its step nodes. We define a truth assignment ν for $\llbracket \phi \rrbracket^k$ as follows. Note that $\llbracket \phi \rrbracket^k$ contains stepped propositions from p^0 until p^k for any given p , so we need at most $k + 1$ step nodes from \bar{u} , which however can be shorter if it is accepted or crossed by the PRUNE rule. Hence, let us define $\ell = \min\{m, k\}$. Moreover, let us define p_U to be p if $p \in \Sigma$, and to be ψ if $p = \psi_G$ for some X-, Y-, or Z-request ψ , i.e. $(\cdot)_U$ is the inverse of the $(\cdot)_G$ operation. Then, for $0 \leq i \leq \ell$, we set $\nu(p^i) = \top$ if and only if $p_U \in \Gamma(\pi_i)$. Then, we complete the assignments for positions $m < j \leq k + 1$ (if any) as follows:

1. if the branch has been accepted by the EMPTY rule, all the other positions $j > m$ can be filled arbitrarily;
2. if the branch has been accepted by the LOOP or crossed by the PRUNE rule, then there is a position w such that $\Gamma(\pi_w) = \Gamma(\pi_m)$. Then we continue filling the truth assignment considering the successor of π_w as a successor of π_m .

It can be verified that the truth assignment so constructed satisfies $\llbracket \phi \rrbracket^k$.

From assignments to branches. Let ν be a truth assignment for $\llbracket \phi \rrbracket^k$. We use ν as a guide to navigate the tableau tree to find a suitable branch which is either accepted, crossed by PRUNE, or has more than $k + 1$ step nodes. To do that we build a sequence of branch prefixes $\bar{u}_i = \langle u_0, \dots, u_i \rangle$ where at each step we obtain \bar{u}_{i+1} by choosing u_{i+1} among the children of u_i , until we find a leaf or we reach $k + 1$ step nodes. During the descent, we build a partial function $J : \mathbb{N} \rightarrow \mathbb{N}$ that maps positions j in \bar{u}_i to indexes $J(j)$ such that for all ψ it holds that $\psi \in \Gamma(u_j)$ if and only if $\nu \models \text{snf}(\psi)_G^{J(j)}$, i.e. we build a relationship between positions in the branch and steps in ν . As the base case, we put $\bar{u}_0 = \langle u_0 \rangle$ and $J(0) = 0$ so that the invariant holds since $\Gamma(u_0) = \{\phi\}$ and $\nu \models \text{snf}(\phi)_G^0$ by the definition of $\llbracket \phi \rrbracket^k$. Then, depending on the rule that was applied to u_i , we choose u_{i+1} among its children as follows:

1. if the STEP rule has been applied to u_i , then there is a unique child that we choose as u_{i+1} , and we define $J(i + 1) = J(i) + 1$. Now, for all $X\alpha \in \Gamma(u_i)$, we have $\alpha \in \Gamma(u_{i+1})$ by construction of the tableau. Note that $\text{snf}(X\alpha) = X\alpha$, hence we know by construction that $\nu \models (X\alpha)_G^{J(j)}$. Then, by definition of $\llbracket \phi \rrbracket^k$, we know that $\nu \models \text{snf}(\alpha)_G^{J(j)+1}$, i.e.

- $\nu \models \text{snf}(\alpha)_G^{J(i+1)}$. On the other direction, if $\nu \models \text{snf}(\alpha)_G^{J(i+1)}$, then by definition of $\llbracket \phi \rrbracket^k$ we have $\nu \models (\mathbf{X}\alpha)_G^{J(i)}$, hence $\nu \models \text{snf}(\mathbf{X}\alpha)_G^{J(i)}$, hence $\mathbf{X}\alpha \in \Gamma(u_i)$, so by construction of the tableau we have $\alpha \in \Gamma(u_{i+1})$. Hence the invariant holds.
2. if the FORECAST rule has been applied to u_i , then there are n children $\{u_i^1, \dots, u_i^n\}$ such that $\Gamma(u_i) \subseteq \Gamma(u_i^m)$ for all $1 \leq m \leq n$. Now, we set $J(i+1) = J(i)$ and we choose u_{i+1} as a child u_i^m with a label $\Gamma(u_i^m)$ such that for any ψ we have $\psi \in \Gamma(u_{i+1})$ if and only if $\nu \models \text{snf}(\psi)_G^{J(i+1)}$. Note that at least one such child exists, because at least one child has the same label as u_i . Thus the invariant holds by construction.
 3. if an *expansion rule* has been applied to u_i , then there are one or two children. In both cases, we set $J(i+1) = J(i)$. Then:
 - a. if there is one child, then it is chosen as u_{i+1} . In this case, the rule is the CONJUNCTION rule and has been applied to a formula $\psi \equiv \psi_1 \wedge \psi_2$, hence $\psi_1, \psi_2 \in \Gamma(u_{i+1})$. By construction we know that $\nu \models \text{snf}(\psi)_G^{J(i)}$, hence $\nu \models \text{snf}(\psi)_G^{J(i+1)}$. Now, note that $\text{snf}(\psi_1 \wedge \psi_2) = \text{snf}(\psi_1) \wedge \text{snf}(\psi_2)$, so it holds that $\nu \models \text{snf}(\psi_1)_G^{J(i+1)}$ and $\nu \models \text{snf}(\psi_2)_G^{J(i+1)}$. On the other direction, if $\nu \models \text{snf}(\psi_1)_G^{J(i+1)}$ and $\nu \models \text{snf}(\psi_2)_G^{J(i+1)}$ we know that $\nu \models \text{snf}(\psi_1 \wedge \psi_2)_G^{J(i+1)}$ hence $\nu \models \text{snf}(\psi_1 \wedge \psi_2)_G^{J(i)}$, hence by construction we have $\psi_1 \wedge \psi_2 \in \Gamma(u_i)$ and so we have $\psi_1, \psi_2 \in \Gamma(u_i)$, hence the invariant holds.
 - b. if there are two children u_i' and u_i'' , then let us suppose the rule applied is the DISJUNCTION rule. Similar arguments will hold for the other rules. In this case, the rule has been applied to a formula $\psi \equiv \psi_1 \vee \psi_2$, hence $\psi_1 \in \Gamma(u_i')$ and $\psi_2 \in \Gamma(u_i'')$. We know $\nu \models \text{snf}(\psi)_G^{J(i)}$, hence $\nu \models \text{snf}(\psi)_G^{J(i+1)}$. Since $\text{snf}(\psi_1 \vee \psi_2) = \text{snf}(\psi_1) \vee \text{snf}(\psi_2)$, it holds that either $\nu \models \text{snf}(\psi_1)_G^{J(i+1)}$ or $\nu \models \text{snf}(\psi_2)_G^{J(i+1)}$. Now, we choose u_{i+1} accordingly, so to respect the invariant. Note that if both nodes are eligible, which one is chosen does not matter. The other direction of the invariant also holds, since if either $\nu \models \text{snf}(\psi_1)_G^{J(i+1)}$ or $\nu \models \text{snf}(\psi_2)_G^{J(i+1)}$, then $\nu \models \text{snf}(\psi_1)_G^{J(i)}$ or $\nu \models \text{snf}(\psi_2)_G^{J(i)}$, so $\nu \models \text{snf}(\psi_1 \vee \psi_2)_G^{J(i)}$, hence $\psi_1 \vee \psi_2 \in \Gamma(u_i)$, hence either $\psi_1 \in \Gamma(u_{i+1})$ or $\psi_2 \in \Gamma(u_{i+1})$.
- Let $\bar{u} = \langle u_0, \dots, u_i \rangle$ be the branch prefix constructed as above, and let $\bar{\pi} = \langle \pi_0, \dots, \pi_n \rangle$ be the sequence of its step nodes. As mentioned, the descent stops when π_n is a leaf or when $n = k + 1$. Note in any case that $u_i = \pi_n$. In case we find a leaf, note that it cannot have been crossed by the CONTRADICTION rule. Otherwise, we would have $\{p, \neg p\} \subseteq \Gamma(u_i)$, which would mean $\nu \models p^{J(i)}$ and $\nu \models \neg p^{J(i)}$, which is not possible. Moreover, it cannot have been crossed by the YESTERDAY rule, since that would mean there is some $\mathbf{Y}\alpha \in \Gamma(\pi_n)$ with $\alpha \notin \Gamma^*(\pi_{n-1})$. But, we know that $\nu \models \text{snf}(\mathbf{Y}\alpha)_G^{J(i)}$, hence $\nu \models (\mathbf{Y}\alpha)_G^{J(i)}$ since $\text{snf}(\mathbf{Y}\alpha) = \mathbf{Y}\alpha$. Then, by definition of $\llbracket \phi \rrbracket^k$, we know that $\nu \models \text{snf}(\alpha)_G^{J(i)-1}$. Since $u_i = \pi_n$ is a step node, $J(i) - 1 = J(j)$ for some j such that $u_j = \pi_{n-1}$, hence $\nu \models \text{snf}(\alpha)_G^{J(j)}$, and by construction we know that $\alpha \in \Gamma(u_j)$, which conflicts with the hypothesis that the YESTERDAY rule crossed the branch. With a similar argument, we can see that it cannot have been crossed by the W-YESTERDAY rule neither. Hence we found a branch which is either longer than $k + 1$ step nodes, or have been accepted, or have been crossed by the PRUNE rule. ◀

The formula $|\phi|^k$ is called the *base encoding* of ϕ and, in addition to the k -unraveling, includes the encoding of the EMPTY and LOOP rules, i.e. the rules that can accept branches. The formula is defined as:

$$|\phi|^k \equiv \llbracket \phi \rrbracket^k \wedge (E_k \vee L_k)$$

where the E_k formula encodes the EMPTY rule and is defined as follows:

$$E_k \equiv \bigwedge_{\psi \in \mathbf{X}\mathbf{R}} \neg \psi_G^k$$

and the L_k formula encodes the LOOP rule and is defined as:

$$L_k \equiv \bigvee_{l=0}^{k-1} ({}_lR_k \wedge {}_lF_k)$$

where

$${}_lR_k \equiv \bigwedge_{\psi \in \text{XRUYR} \cup \text{ZR}} \left(\psi_G^l \leftrightarrow \psi_G^k \right), \text{ and}$$

$${}_lF_k \equiv \bigwedge_{\substack{\psi \in \text{XEV} \\ \psi \equiv \text{X}(\psi_1 \mathcal{U} \psi_2)}} \left(\psi_G^k \rightarrow \bigvee_{i=l+1}^k \text{snf}(\psi_2)_G^i \right).$$

Intuitively, ${}_lR_k$ encodes the presence of two nodes whose labels contain the same requests for the next and the previous nodes, while ${}_lF_k$ checks that all the X -eventualities are fulfilled between those nodes. It can be proved that $|\phi|^k$ correctly encodes tableau trees where at least one branch is accepted in $k + 1$ steps.

► **Lemma 7.** *Let ϕ be an LTL+Past formula. If the complete tableau for ϕ contains an accepted branch of $k + 1$ step nodes, then $|\phi|^k$ is satisfiable.*

Proof. Suppose that the complete tableau for ϕ contains an accepted branch of $k + 1$ step nodes, so let $\bar{u} = \langle u_0, \dots, u_n \rangle$ be such a branch, and let $\bar{\pi} = \langle \pi_0, \dots, \pi_k \rangle$ be the sequence of its step nodes. Then, by Lemma 6, $\llbracket \phi \rrbracket^k$ is satisfiable. We can then build a truth assignment ν in the same way as in the proof of Lemma 6, such that $\nu \models \llbracket \phi \rrbracket^k$. Remember that this means we set $\nu(p^i) = \top$ if and only if $p_U \in \Gamma(\pi_i)$ for all $0 \leq i \leq k$. So now we have to prove that ν satisfies either E_k or L_k . We will need an auxiliary fact, that is, that $\psi \in \Gamma^*(\pi_i)$ if and only if $\nu \models \text{snf}(\psi)_G^i$. That can be done by induction on the structure of ψ , exploiting the definition of the expansion rules of the tableau.

Now, we distinguish two cases depending on which rule accepted the branch:

1. if the branch was accepted by the EMPTY rule, then $\Gamma(\pi_k) = \emptyset$, hence, in particular $\Gamma(\pi_k)$ does not contain any X -request. Hence by definition of ν we have that $\nu \models \neg \psi_G^k$ for any $\psi \in \text{XR}$, so E_k is satisfied;
2. if the branch was accepted by the LOOP rule, then we have a node π_l such that $\Gamma(\pi_l) = \Gamma(\pi_k)$, hence by definition of ν we have $\nu \models \psi_G^l$ if and only if $\nu \models \psi_G^k$ for any $\psi \in \text{XR} \cup \text{YR} \cup \text{ZR}$, so ${}_lR_k$ is satisfied. Moreover, we know that for any X -eventuality $\psi \equiv \text{X}(\psi_1 \mathcal{U} \psi_2)$ requested in $\Gamma(\pi_k)$, ψ has been fulfilled between π_l and π_k , i.e. there is a $l < j \leq k$ such that $\psi_2 \in \Gamma^*(\pi_j)$. Hence we know that $\nu \models \text{snf}(\psi_2)_G^j$, hence ${}_lF_k$ is satisfied. Then, ${}_lR_k \wedge {}_lF_k$ is satisfied for at least one l , so L_k is satisfied. ◀

► **Lemma 8.** *Let ϕ be an LTL+Past formula. If $|\phi|^k$ is satisfiable then the complete tableau for ϕ contains an accepted branch.*

Proof. Suppose that $|\phi|^k$ is satisfiable, hence we have a truth assignment ν such that $\nu \models |\phi|^k$. Then, $\llbracket \phi \rrbracket^k$ is satisfiable, and we know from Lemma 6 that the complete tableau for ϕ has a branch that is either accepted, crossed by PRUNE, or longer than $k + 1$ step nodes. Let $\bar{u} = \langle u_0, \dots, u_n \rangle$ be the branch prefix found as shown in the proof of Lemma 6, and let $\bar{\pi} = \langle \pi_0, \dots, \pi_m \rangle$ be the sequence of its step nodes. By construction we have a function $J : \mathbb{N} \rightarrow \mathbb{N}$ fulfilling the invariant that $\psi \in \Gamma(u_i)$ if and only if $\nu \models \text{snf}(\psi)_G^{J(i)}$. We now show that indeed \bar{u} is accepted or is the prefix of an accepted branch. Since $|\phi|^k$ is satisfiable, either E_k or L_k are satisfiable as well:

1. if E_k is satisfiable, we know that $\nu \models \neg\psi_G^k$ for each $\psi \in \text{XR}$. Since ψ is an X-request, $\text{snf}(\psi) \equiv \psi$, so $\nu \not\models \text{snf}(\psi)_G^k$. Here, $k = J(j)$ for some j , and from the invariant we know that $\psi \notin \Gamma(u_j)$. Hence, u_j does not contain any X-request, so its successor u_{j+1} has an empty label, triggering the EMPTY rule that accepts the branch.
2. if L_k is satisfiable, so are ${}_lR_k$ and ${}_lF_k$ for some $0 \leq l < k$. Hence from ${}_lR_k$ we know that $\nu \models \psi_G^l$ if and only if $\nu \models \psi_G^k$ for all $\psi \in \text{XR} \cup \text{YR} \cup \text{ZR}$, that is $\nu \models \text{snf}(\psi)_G^l$ if and only if $\nu \models \text{snf}(\psi)_G^k$ because ψ is an X-, Y-, or Z-request. Here, $l = J(i)$ and $k = J(j)$ for some i and some j . Since the value of the function J increments at each step node, we can assume *w.l.o.g.* that u_i and u_j are step nodes, and by the invariant we know $\psi \in \Gamma(u_i)$ if and only if $\psi \in \Gamma(u_j)$, i.e. u_i and u_j have the same X-, Y-, and Z-request. Similarly, the fact that $\nu \models {}_lF_k$ tells us that all the X- eventualities requested in u_i are fulfilled between u_{i+1} and u_j . The LOOP rule requires two identical labels in order to trigger, but u_i and u_j only have the same requests. However, since they have the same X-requests, we know that $\Gamma(u_{i+1}) = \Gamma(u_{j+1})$. Then, there is a step node $u_{j'}$, grandchild of u_j , such that $\Gamma(u_j) = \Gamma(u_{j'})$ and the segment of the branch between u_{i+1} and u_j is equal to the segment between u_{j+1} and $u_{j'}$, hence all the X- eventualities requested in u_i and u_j , fulfilled between u_{i+1} and u_j , are fulfilled between u_{j+1} and $u_{j'}$ as well, and the LOOP rule can apply to $u_{j'}$, accepting the branch. \blacktriangleleft

Lastly, the formula $|\phi|_T^k$, called the *termination encoding*, encodes the PRUNE rule. The formula is defined as follows:

$$|\phi|_T^k \equiv \llbracket \phi \rrbracket^k \wedge \bigwedge_{i=0}^k \neg P^i$$

where

$$P^k \equiv \bigvee_{l=0}^{k-2} \bigvee_{j=l+1}^{k-1} ({}_lR_j \wedge {}_jR_k \wedge {}_lP_j^k)$$

$${}_lP_j^k \equiv \bigwedge_{\substack{\psi \in \text{XEV} \\ \psi \equiv \text{X}(\psi_1 \mathcal{U} \psi_2)}} (\psi_G^k \wedge \bigvee_{i=j+1}^k \text{snf}(\psi_2)_G^i \rightarrow \bigvee_{i=l+1}^j \text{snf}(\psi_2)_G^i)$$

It can be proved that $|\phi|_T^k$ is unsatisfiable if the tableau for ϕ contains only rejected branches.

► **Lemma 9.** *Let ϕ be an LTL+Past formula. If $|\phi|_T^k$ is unsatisfiable, then the complete tableau for ϕ contains only rejected branches.*

Proof. We prove the contrapositive, i.e. that if the complete tableau for ϕ contains an accepted branch, then $|\phi|_T^k$ is satisfiable. Let $\bar{u} = \langle u_0, \dots, u_n \rangle$ be such a branch, and let $\bar{\pi} = \langle \pi_0, \dots, \pi_m \rangle$ be the sequence of its step nodes. By Lemma 6, we know $\llbracket \phi \rrbracket^k$ is satisfiable, thus we can obtain a truth assignment ν such that $\nu \models \llbracket \phi \rrbracket^k$. We can build ν as in the proof of Lemma 6, i.e. such that $\nu(p^i) = \top$ if and only if $p_U \in \Gamma(\pi_i)$ for all $0 \leq i \leq k$. Similarly to the proof of Lemma 7, we highlight the fact that $\psi \in \Gamma^*(\pi_i)$ if and only if $\nu \models \text{snf}(\psi)_G^i$. Now, since the branch is accepted, the PRUNE rule cannot be applied to it. This means that either a) there are no three nodes π_u, π_v, π_w such that $\Gamma(\pi_u) = \Gamma(\pi_v) = \Gamma(\pi_w)$, or b) these three nodes exist but there is an X-eventuality ψ requested in $\Gamma(\pi_w)$ that is fulfilled between π_u and π_v and not between π_v and π_w . In case a) this means ${}_uR_v \wedge {}_vR_w$ does not hold for any u and v . In case b), ${}_uR_v \wedge {}_vR_w$ holds but ${}_uP_v^w$ does not. In any case, it follows that $\neg P^i$ holds for any $0 \leq i \leq k$, hence $|\phi|_T^k$ is satisfied. \blacktriangleleft

Together with the soundness and completeness results for the underlying tableau [13], the above Lemmata allow us to prove the soundness and completeness of the procedure of Algorithm 1.

► **Theorem 10** (Soundness and completeness). *Let ϕ be an LTL+Past formula. The BLACK algorithm answers satisfiable on ϕ if and only if ϕ is satisfiable.*

Proof. (\rightarrow) Suppose the BLACK algorithm answers *satisfiable* on the formula ϕ . Then, it means there is a $k \geq 0$ such that $|\phi|^k$ is satisfiable. By Lemma 8, the complete tableau for ϕ has an accepting branch. By the soundness of the tableau, then ϕ is satisfiable.

(\leftarrow) Now suppose the formula ϕ is satisfiable. By the completeness of the tableau, the complete tableau for ϕ has an accepting branch. Let us suppose such a branch has $k + 1$ step nodes for some $k \geq 0$. Then, we want to show that the BLACK algorithm eventually answers *satisfiable*. Let $i < k$ be any earlier iteration of the main loop of the algorithm. We have that by Lemma 6, $\llbracket \phi \rrbracket^i$ is satisfiable because there is a branch longer than $i + 1$ step nodes. Similarly, by Lemma 9, $|\phi|_T^i$ is satisfiable because not all the branches of the tableau are rejected. Hence, the algorithm does not answer *unsatisfiable* at step i . Arrived at step k , $|\phi|^k$ is satisfiable by Lemma 7 because the tableau has an accepted branch of $k + 1$ step nodes, hence the algorithm answers *satisfiable*. ◀

Despite the final encoding may seem trivially simple, this simplicity makes the approach interesting for two reasons. First of all, it was not *a priori* clear that such a simple encoding would be possible, given the presence of the FORECAST rule: a rule that kills the performance of the explicit construction of the tableau turns out to be a non-problem in the symbolic exploration of the tree. Secondly, this simplicity is what drives the experimental success of the encoding. Comparing it for example with Biere et al. [2] which is, as far as we know, the approach implemented by nuXmv: all the *virtual unrollings* machinery that they need to support past operators is much more complex than our seemingly trivial encoding which, however, performs better even though it does not generate CNF formulas of linear size. Hence, showing that the approach by Geatti et al. [12] can be successfully extended so easily can be considered one of the main contributions of this work.

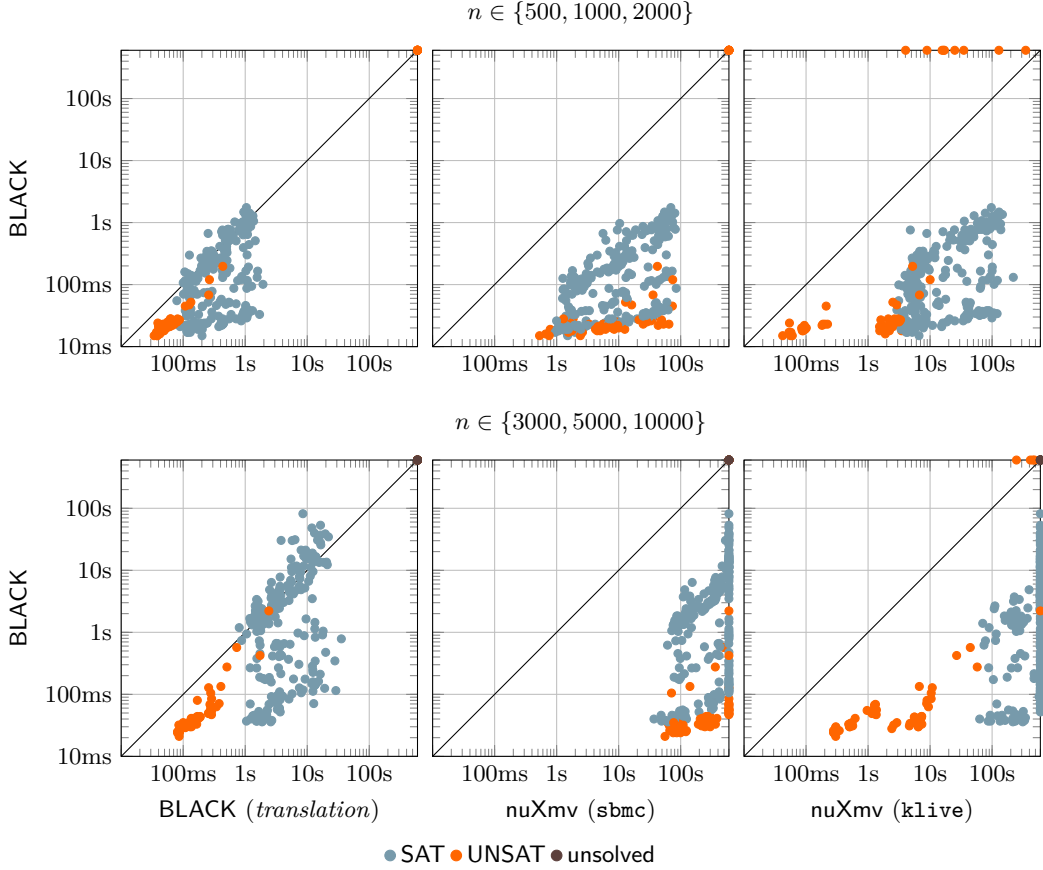
5 Experimental results

We have implemented the above two approaches in version 0.3.0 of the BLACK tool¹: the translation as an optional module that can be activated upon user request, and the direct encoding as an expansion of the core procedure.

Since the literature lacks significant LTL+Past family of formulas for benchmarks, we have devised two novel sets of formulas. As for the first family, we chose a set of *random formulas*, generated with an algorithm adapted from [24], in order to verify how the tool scales in general. The second family, that we called **crscounter**, is inspired by and adapted from Cimatti et al. [7], where a Kripke structure called *Counter(N)*, where N is a power of two, is introduced. *Counter(N)* works as follows: it starts at $c = 0$, counts up to $c = N$, jumps back to $c = N/2$, and then loops, counting up to $c = N$ and jumping back to $c = N/2$, forever. Afterwards, they evaluated, on top of that Kripke structure, some parametric properties of the form:

$$P(i) \equiv \neg F(O((c = \frac{N}{2}) \wedge O((c = \frac{N}{2} + 1) \wedge \dots \wedge O(c = \frac{N}{2} + i) \dots))).$$

¹ The tool can be found at <https://github.com/black-sat/black>. Packages for macOS and common Linux distributions are provided, together with all the necessary scripts to reproduce the tests performed.

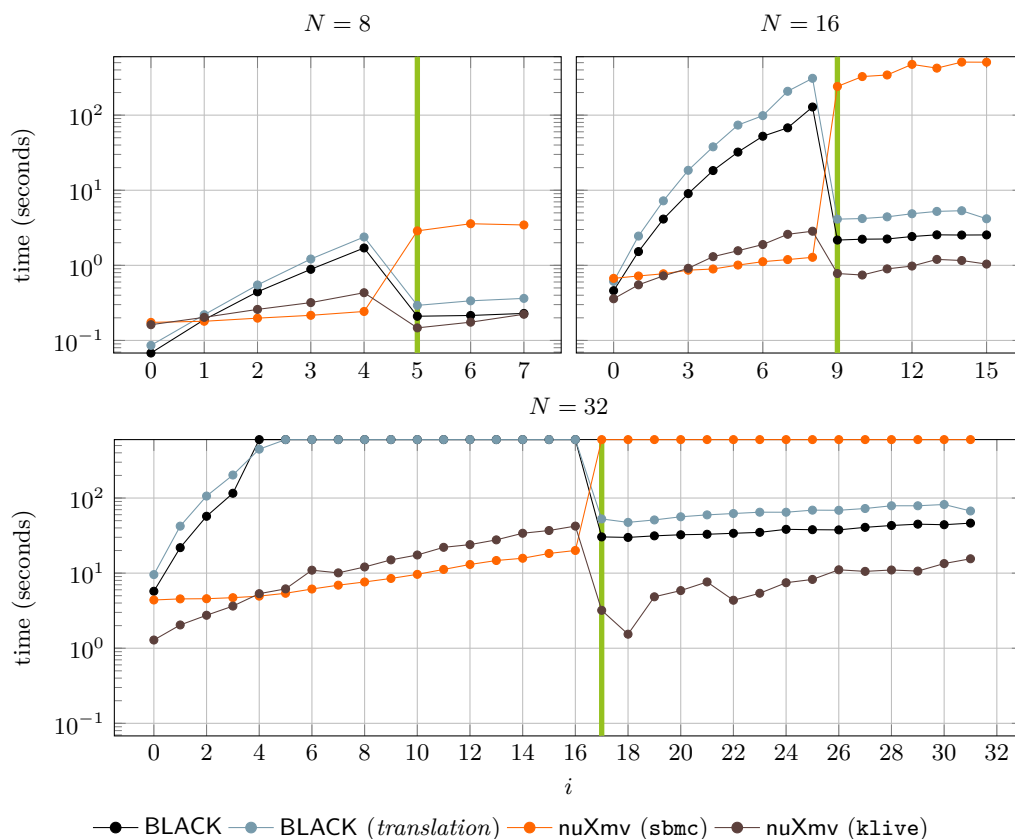


■ **Figure 1** Experimental results of BLACK against the other tested tools and modalities, for random formulas of size $n \in \{500, 1000, 2000, 3000, 5000, 10000\}$. Times are in *seconds*. Note that only instances that could not be solved by *any* tool are marked as *unsolved*.

The value i identifies the number of nested *once* operators, while the structure of such properties requires that the loop of length $N/2$ in the model is traversed backwards several times in order to reach a counterexample.

The `crscounter` benchmarks were introduced in the context of model checking. Thus, we made a reduction from the model checking problem to the satisfiability checking one for LTL+Past: we built the LTL+Past formulas $\phi_{Counter(N)}$ and $\phi_{P(i)}$ encoding the above elements. In this way, $\neg(\phi_{Counter(N)} \rightarrow \phi_{P(i)})$ is UNSAT if and only if $Counter(N) \models P(i)$. With this framework, we were able to obtain both SAT ($i \leq \frac{N}{2}$) and UNSAT ($i > \frac{N}{2}$) instances. Moreover, this family of formulas stresses the ability to process past operators and find short counterexamples, and thus, it specifically challenges our contribution.

For each formula, we executed both an *internal comparison* between the two proposed techniques – with BLACK over MathSAT [4], which is the best performing solver among those supported by BLACK so far –, and an *external comparison* with nuXmv [5], which, as far as we know, is the only state-of-the-art tool directly supporting past operators. Specifically, we tested BLACK against nuXmv in both `sbmc` and `klive` modalities. The former stands for Simple Bounded Model Checking [2], and it is the closest to our approach between the two. The latter has been proposed more recently, and it is based on K-Liveness [6].



■ **Figure 2** Experimental results for `crscounter` formulas of size $N \in \{8, 16, 32\}$. Green vertical bars indicate where formulas start to be UNSAT.

All the experiments have been performed on a Dell PowerEdge rack server equipped with a 16-cores AMD EPYC™ 7281 CPU (2.7GHz) and 64GB of RAM (DDR4 2400 MT/s). Moreover, experiments have been run in parallel, each on a single core, with a *memory limit* of 3GB of RAM per core, and a 10 minutes *timeout*. Experimental results plots can be found in Figure 1 and Figure 2.

Regarding the random formulas, what immediately catches the eye is that **BLACK** – in the direct approach – performs overall significantly better, in particular if compared with **nuXmv**. Indeed, the latter starts to be in difficulty already at size $n = 3000$, and gets almost always stuck at size $n = 10000$. It can also be noticed that **BLACK** solves UNSAT instances quicker than the SAT ones. This could be a bit surprising, considering that the former is a universal property, while the latter is an existential one. However, this has a twofold explanation. Firstly, the Boolean encoding acts like a breadth-first search, which is in some sense an exhaustive search, up to depth k . Secondly, in all UNSAT random formulas the tableaux are closed by contradiction, and never by the PRUNE rule. This is because of the nature of the random formulas generator: it is less likely to generate a formula with such a peculiar structure as to trigger the PRUNE rule. Also, it is more likely to produce contradictions that can be spotted in the first depth levels. Nevertheless, it is interesting to note that **BLACK** is able to manage those UNSAT formulas better than **nuXmv**, overall.

Looking instead at the `crscounter` plots, there are two interesting aspects to point out. First, **BLACK** performs slightly better with the direct past encoding than with the translation approach. Second, while **BLACK** performs worse with SAT formulas, the situation overturns

when the formulas become UNSAT. This can be explained by the PRUNE rule: it may cause an overhead in the encoding of SAT formulas, but it has the advantage of guaranteeing a quicker termination, i.e. closure of the tableau, in case of UNSAT ones. Indeed, in this family of formulas, all UNSAT ones have a tableau which is closed by the PRUNE rule.

6 Conclusions

This work contributes to the state of the art of the satisfiability problem for LTL+Past in different directions. First of all, it provides a translation procedure for LTL+Past, that has been implemented into the BLACK tool, but, in principle, can be used also as a preprocessing step for any LTL satisfiability checker in order to let them support the past. Then, it extends the SAT-based encoding of a one-pass and tree-shaped tableau for LTL to LTL+Past, and shows that the resulting tool is very effective and efficient compared to the state-of-the-art ones. It is also worth noting that the encoding is quite simple compared to the *virtual unrollings* techniques used to support past operators by Latvala et al. [15], and it offers an exponential advantage over the explicit construction of the tableau since the FORECAST rule is not required to be encoded. Finally, it introduces two new sets of formulas which aim at starting to build a larger set of standardized formulas for benchmarks in the field.

Results showed that the direct encoding is the preferred way to manage the past, but also that the translation is a viable alternative, as it adds only a linear overhead.

Future work should head at reducing the overhead introduced by the PRUNE rule, possibly by triggering it not at each step, but with some predetermined heuristics, in order to reduce the gap in performance between SAT and UNSAT properties. Moreover, since the framework has been shown to be efficient and modular, it could be interesting to investigate its extension to other logics. Orthogonally, some in-depth theoretical comparison with other state-of-the-art techniques could suggest new ways of improvement.

References

- 1 Matteo Bertello, Nicola Gigante, Angelo Montanari, and Mark Reynolds. Leviathan: A new LTL satisfiability checking tool based on a one-pass tree-shaped tableau. In *Proc. of the 25th International Joint Conference on Artificial Intelligence*, pages 950–956. IJCAI/AAAI Press, 2016. URL: <http://www.ijcai.org/Abstract/16/139>.
- 2 Armin Biere, Keijo Heljanko, Tommi A. Junttila, Timo Latvala, and Viktor Schuppan. Linear encodings of bounded LTL model checking. *Log. Methods Comput. Sci.*, 2(5), 2006. doi:10.2168/LMCS-2(5:5)2006.
- 3 Laura Bozzelli, Aniello Murano, and Loredana Sorrentino. Alternating-time temporal logics with linear past. *Theor. Comput. Sci.*, 813:199–217, 2020. doi:10.1016/j.tcs.2019.11.028.
- 4 Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani. The MathSAT 4 SMT Solver. In *Computer Aided Verification*, LNCS, pages 299–303. Springer, 2008. doi:10.1007/978-3-540-70545-1_28.
- 5 Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuXmv Symbolic Model Checker. In *Computer Aided Verification*, pages 334–342. Springer, 2014. doi:10.1007/978-3-319-08867-9_22.
- 6 Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. Verifying LTL Properties of Hybrid Systems with K-Liveness. In *Computer Aided Verification*, LNCS, pages 424–440. Springer, 2014. doi:10.1007/978-3-319-08867-9_28.
- 7 Alessandro Cimatti, Marco Roveri, and Daniel Sheridan. Bounded Verification of Past LTL. In *Formal Methods in Computer-Aided Design*, LNCS, pages 245–259. Springer, 2004. doi:10.1007/978-3-540-30494-4_18.
- 8 Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, 2018. doi:10.1007/978-3-319-10575-8.

- 9 Michael Fisher. A resolution method for temporal logic. In John Mylopoulos and Raymond Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 99–104. Morgan Kaufmann, 1991.
- 10 Michael Fisher. A normal form for temporal logics and its applications in theorem-proving and execution. *J. Log. Comput.*, 7(4):429–456, 1997. doi:10.1093/logcom/7.4.429.
- 11 Michael Fisher, Dov M. Gabbay, and Lluís Vila, editors. *Handbook of Temporal Reasoning in Artificial Intelligence*, volume 1 of *Foundations of Artificial Intelligence*. Elsevier, 2005. URL: <http://cgi.csc.liv.ac.uk/%7Emichael/handbook.html>.
- 12 Luca Geatti, Nicola Gigante, and Angelo Montanari. A SAT-Based Encoding of the One-Pass and Tree-Shaped Tableau System for LTL. In *Proc. of the 28th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, volume 11714 of *LNCS*, pages 3–20. Springer, 2019. doi:10.1007/978-3-030-29026-9_1.
- 13 Luca Geatti, Nicola Gigante, Angelo Montanari, and Mark Reynolds. One-pass and tree-shaped tableau systems for TPTL and TPTL_b+Past. *Information and Computation*, 2021. in press. doi:10.1016/j.ic.2020.104599.
- 14 Ullrich Hustadt and Boris Konev. TRP++2.0: A Temporal Resolution Prover. In *Proc. of the 19th International Conference on Automated Deduction*, pages 274–278, 2003.
- 15 Timo Latvala, Armin Biere, Keijo Heljanko, and Tommi A. Junttila. Simple is better: Efficient bounded model checking for past LTL. In *Proc. of the 6th International Conference on Verification, Model Checking, and Abstract Interpretation*, volume 3385 of *LNCS*, pages 380–395. Springer, 2005. doi:10.1007/978-3-540-30579-8_25.
- 16 Jianwen Li, Yinbo Yao, Geguang Pu, Lijun Zhang, and Jifeng He. Aalta: an LTL satisfiability checker over infinite/finite traces. In *Proc. of the 22nd ACM International Symposium on Foundations of Software Engineering*, pages 731–734. ACM, 2014. doi:10.1145/2635868.2661669.
- 17 Orna Lichtenstein and Amir Pnueli. Propositional Temporal Logics: Decidability and Completeness. *Logic Journal of the IGPL*, 8(1):55–85, 2000. doi:10.1093/jigpal/8.1.55.
- 18 Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. The Glory of the Past. In *Proc. of the 1st Conference on Logics of Programs*, volume 193 of *LNCS*, pages 196–218. Springer, 1985. doi:10.1007/3-540-15648-8_16.
- 19 Nicolas Markey. Temporal logic with past is exponentially more succinct. *Bulletin of the EATCS*, 79:122–128, 2003.
- 20 Amir Pnueli. The Temporal Logic of Programs. In *Proc. of the 18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.
- 21 Mark Reynolds. More Past Glories. In *Proc. of the 15th Annual IEEE Symposium on Logic in Computer Science*, pages 229–240. IEEE Computer Society, 2000. doi:10.1109/LICS.2000.855772.
- 22 Mark Reynolds. A New Rule for LTL Tableaux. In *Proc. of the 7th International Symposium on Games, Automata, Logics and Formal Verification*, volume 226 of *EPTCS*, pages 287–301, 2016. doi:10.4204/EPTCS.226.20.
- 23 A. Prasad Sistla and Edmund M. Clarke. The Complexity of Propositional Linear Temporal Logics. *Journal of the ACM*, 32(3):733–749, 1985. doi:10.1145/3828.3837.
- 24 Heikki Tauriainen and Keijo Heljanko. Testing LTL formula translation into Büchi automata. *International Journal on Software Tools for Technology Transfer*, 4(1):57–70, 2002. doi:10.1007/s100090200070.
- 25 G. S. Tseitin. On the Complexity of Derivation in Propositional Calculus. In Jörg H. Siekmann and Graham Wrightson, editors, *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*, Symbolic Computation, pages 466–483. Springer, Berlin, Heidelberg, 1983. doi:10.1007/978-3-642-81955-1_28.
- 26 Pierre Wolper. Temporal Logic Can Be More Expressive. *Information and Control*, 56(1/2):72–99, 1983. doi:10.1016/S0019-9958(83)80051-5.