Journal of Automated Reasoning (2024) 68:6
https://doi.org/10.1007/s10817-023-09691-1

SAT Meets Tableaux for Linear Temporal Logic Satisfiability

Luca Geatti[1] · Nicola Gigante[2] · Angelo Montanari[1] · Gabriele Venturato[3]

Received: 11 March 2022 / Accepted: 15 December 2023

Abstract
Linear temporal logic (LTL) and its variant interpreted on *finite traces* ($\mathsf{LTL_f}$) are among the most popular specification languages in the fields of formal verification, artificial intelligence, and others. In this paper, we focus on the satisfiability problem for LTL and $\mathsf{LTL_f}$ formulas, for which many techniques have been devised during the last decades. Among these are *tableau systems*, of which the most recent is Reynolds' tree-shaped tableau. We provide a SAT-based algorithm for LTL and $\mathsf{LTL_f}$ satisfiability checking based on Reynolds' tableau, proving its correctness and discussing experimental results obtained through its implementation in the BLACK satisfiability checker.

Keywords Linear temporal logic · SAT · Tableaux

Mathematics Subject Classification 68N30 · 68T15

1 Introduction

Linear temporal logic (LTL) [44] is the de-facto standard language for the specification of system properties in the fields of *formal verification* and *artificial intelligence*. LTL is a modal logic usually interpreted over infinite and discrete linear orders, but recently its variant interpreted over *finite traces* ($\mathsf{LTL_f}$) has gained traction, especially in the artificial intelligence community [16, 18]. Satisfiability checking, that is, the problem of deciding whether a given

This work is an extended and improved version of [27] and [29].

✉ Nicola Gigante
nicola.gigante@unibz.it

Luca Geatti
luca.geatti@uniud.it

Angelo Montanari
angelo.montanari@uniud.it

Gabriele Venturato
gabriele.venturato@kuleuven.be

[1] University of Udine, Udine, Italy
[2] Free University of Bozen-Bolzano, Bolzano, Italy
[3] KU Leuven, Leuven, Belgium

formula admits a satisfying model, is one of the most important computational tasks associated with the logic, and one of the first that have been carefully studied [48]. Tableau systems are among the first methods that have been proposed to solve the satisfiability checking problem for LTL [55]. Classic tableau systems for LTL [39, 55] are *graph-shaped* and *two-passes*, that is, they build a graph structure, that represents all tentative paths of the transition system described by the input formula, and then traverse it, in a second pass, to find a suitable model for the formula. In contrast, the alternative tableau system recently devised by Reynolds [45] is *tree-shaped* and *one-pass*, since it builds a tree structure where a single pass is required to build a given branch and simultaneously decide whether it corresponds to a model for the formula. Reports on early implementations of Reynolds tableau [4, 42] highlights how *propositional reasoning* is the bottleneck for this kind of procedures, since the search for solutions of the propositional part of temporal formulas is basically done by brute force.

To overcome this limitation, in this paper we join Reynolds' tableau with Boolean satisfiability (SAT) solvers, by providing a SAT-based satisfiability checking procedure for LTL and $\mathsf{LTL_f}$ based on Reynolds' tableau. In our procedure, the tableau is never built explicitly. Instead, suitable SAT formulas representing all the branches of the tableau tree up to a given depth k are solved, for increasing values of k. If a successful branch (i.e. a model for the formula) is found within the given bound, the formula is satisfiable, otherwise k is incremented and the procedure continues. In contrast to similar approaches *à la* bounded model checking, our procedure guarantees completeness and termination both for satisfiable and unsatisfiable formulas, without the need to precompute unsatisfiability thresholds, thanks to the pruning rule of Reynolds' tableau.

The modularity of Reynolds' tableau and of its encoding allows our procedure to support both *future* and *past* temporal modalities, interpreted on both *finite* and *infinite* traces. On the one hand, *past* temporal modalities do not add expressive power to the logic, but do increase its succinctness [41] and allow many properties to be expressed in a more natural way [40]. Independently of the set of temporal modalities and the considered class of models, our procedure is also able to output a model for satisfiable formulas.

We describe the procedure, prove its correctness, and evaluate its performance. To this aim, the procedure has been implemented in the BLACK[1] satisfiabilty checker, whose architecture we describe here as well. To assess the performance of the tool, we compare it with several state-of-the-art tools that support satisfiability checking for LTL over a benchmark set of thousands of formulas gathered from various sources. The results show that our technique is competitive for many classes of benchmark formulas.

The paper is structured as follows. After reviewing the relevant literature in Sect. 2, we recall the needed background in Sect. 3, including Reynolds' one-pass and tree-shaped tableau. Then, in Sect. 4, we describe in detail our SAT-based procedure, including full proofs of soundness, completeness, and termination. Section 5 discusses the implementation of BLACK, with a particular attention to design choices. Finally, Sect. 6 experimentally compares the tool with other state-of-the-art solvers. Section 7 concludes with some final remarks and a discussion of future developments.

2 Related Work

Shortly after its introduction [44], Linear Temporal Logic has become the de-facto standard language for specification of temporal properties both in formal verification [15] and in

[1] BLACK can be downloaded from https://github.com/black-sat/black.

artificial intelligence [2]. The satisfiability problem for LTL has been proved to be PSPACE-complete [48]. Despite such a theoretically high computational complexity, many techniques and tools have been developed to solve it, ranging from tableau systems [1, 4, 39, 45, 47, 54] to reduction to model checking [10], from temporal resolution [24, 25, 33], to labelled superposition [51], to automata-theoretic techniques [36].

Tableau methods were among the first techniques to be proposed. Born in the context of propositional and first-order logic [5], tableaux are quite easy to extend to the non-classical setting. Most tableaux for LTL are *graph-shaped* [39, 55], as they build a graph structure which is then traversed in a *second pass* to find a suitable model for the formula. Since in many cases the built structure is really huge, various techniques have been proposed to improve the efficiency of the procedure. As an example, in *incremental* tableaux [35], only those parts of the graph that are actually involved in the search for the model are built. In contrast, *tree-shaped* tableaux try to avoid building the entire structure altogether, focusing instead on its paths. The first such tableau was proposed by Schwendimann [47], followed by Reynolds' tableau [45]. In this paper, we focus on the latter. Even if both tableaux are tree-shaped and can be regarded as being one-pass, Reynolds' one has the advantage of expanding each branch of the tree completely independently from the others, as each branch corresponds to a distinct tentative model for the formula. In contrast, Schwendimann's tableau needs to keep track of multiple branches in order to accept or reject a given subtree. The possibility of such an independent exploration of branches has been heavily exploited by an early implementation of Reynolds' tableau [4] and its later parallelization [42]. As a matter of fact, the SAT-based procedure shown in this paper would not be possible for Schwendimann's tableau exactly because of this difference. In addition to that, the modular, rule-based structure of Reynolds' tableau system allowed it to be extended to various logics beyond standard LTL. In particular, support for *past* modalities has been added, as well as for more expressive logics like the real-time Timed Propositional Temporal Logic (TPTL) [28].

An important property of LTL is that satisfiability checking can be easily reduced to *model checking*: to establish whether a formula is satisfiable, its negation is model checked against the complete transition system, with any counterexample being a model of the original formula. Hence, any model checking technique can be seen as an alternative satisfiability checking technique. In this perspective, the procedure presented here is similar in spirit to *bounded model checking* techniques [6, 12]: a counterexample (here, a tableau branch) of length (tree depth) up to k, for increasing values of k, is found by encoding the paths of the structure (the branches of the tree) up to length (depth) k into a SAT formula. However, bounded model checking techniques are usually incomplete, since the computation of the diameter of the graph, which witnesses the exploration of *all paths*, is usually a very hard task (requiring, e.g., to solve the satisfiability problem of a quantified Boolean formula [6]). Here, instead, our algorithm is complete thanks to the encoding of the PRUNE rule of Reynolds' tableau (see Sect. 3). In addition to that, the encoding of past modalities, coming from the tableau rules, is much simpler than the *virtual unrollings* technique used to support past modalities in bounded model checking approaches [7]. In our encoding, support for past modalities comes almost for free. This was surprising at first, since support for past modalities (sketched by Gigante et al. [31] and finalized by Geatti et al. [28]) is a bit more involved in the explicit construction of Reynolds' tableau.

Although LTL has been historically defined over infinite traces, the finite-trace semantics has recently gained popularity in the artificial intelligence [16] and business process modelling fields [17]. Although the computational complexity of all the main problems remain the same, the manipulation of finite state automata on finite words, instead of Büchi automata on infinite words, guarantees a notable speed-up in practice. This led to much work revisiting,

for example, model checking and synthesis [18], and the use of LTL on finite traces as specification language for non-Markovian rewards in Markov Decision Processes [9], for restraining specifications in reinforcement learning applications [19], and for specifications of temporally extended goals in fully observable nondeterministic planning [8].

The approach presented here would not be sensible without the use of efficient SAT solvers as the backend. The satisfiability problem for propositional logic is the canonical NP-complete problem and one of the most studied problems in computer science. For this reason, the efficiency of modern SAT solvers has grown beyond the best expectations. BLACK, the tool we developed to evaluate our procedure, supports different solvers as backend in order to be able to exploit the advantages of each. In addition to the classic, but now outdated, MiniSAT [21], we support CryptoMiniSAT [49], a modern, parallelized and very flexible SAT solver. In addition to that, we support two Satisfiability Modulo Theories (SMT) solvers, Z3 [20], cvc5 [3], and MathSAT [14]. For LTL satisfiability, we do not make use of any SMT feature, but the two SMT solvers proved to be very competitive backends also for purely propositional problems.

The present paper is an extension of previous work [27, 29], which presented the SAT-based procedure and later extended it to past modalities. Support for finite-trace semantics has never been presented before.

3 Reynolds' One-Pass and Tree-Shaped Tableau System

In this section, we recall the syntax and semantics of LTL and of its extension with past operators, LTL+Past. Then, we describe the rules of Reynolds' tableau, that is the subject of the encoding presented in Sect. 4.

3.1 Syntax and Semantics of LTL+Past

Let us consider an alphabet Σ of *proposition letters* (or *propositions*). Then, the syntax of an LTL+Past formula ϕ over Σ can be defined as follows:

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

where $p \in \Sigma$ and ϕ, ϕ_1, and ϕ_2 are LTL+Past formulas. The (future-only) fragment LTL only uses Boolean connectives and future operators. One can define the standard shorthands and derived operators as usual, e.g., $\top \equiv p \vee \neg p$, for some $p \in \Sigma$, $\bot \equiv \neg\top$, $\mathsf{F}\phi \equiv \top \,\mathcal{U}\, \phi$, $\mathsf{G}\phi \equiv \neg\mathsf{F}\neg\phi$, $\mathsf{O}\phi \equiv \top \,\mathcal{S}\, \phi$, $\mathsf{H}\phi \equiv \neg\mathsf{O}\neg\phi$. Given a temporal operator *op*, a formula is an *op formula* if the top-level operator of the formula is *op* (e.g. $\mathsf{X}p$ is a *tomorrow* formula).

Given a set of symbols A, we denote as A^* the set of finite words over A, and as A^ω the set of infinite words over A. Given a word $w \in A^*$, we denote as $|w|$ the length of w, while we set $|w| = \omega$ for $w \in A^\omega$. LTL+Past is interpreted over finite or infinite *state sequences*, i.e. words $\overline{\sigma} \in (2^\Sigma)^*$ or $\overline{\sigma} \in (2^\Sigma)^\omega$. Given a (finite or infinite) state sequence $\overline{\sigma} = \langle \sigma_0, \sigma_1, \ldots \rangle$, the *satisfaction* of a formula ϕ by $\overline{\sigma}$ at a time point $i \geq 0$, denoted as $\overline{\sigma}, i \models \phi$, is defined as follows:

1. $\overline{\sigma}, i \models p$ iff $p \in \sigma_i$;
2. $\overline{\sigma}, i \models \neg\phi$ iff $\overline{\sigma}, i \not\models \phi$;

3. $\overline{\sigma}, i \models \phi_1 \vee \phi_2$ iff $\overline{\sigma}, i \models \phi_1$ or $\overline{\sigma}, i \models \phi_2$;
4. $\overline{\sigma}, i \models \phi_1 \wedge \phi_2$ iff $\overline{\sigma}, i \models \phi_1$ and $\overline{\sigma}, i \models \phi_2$;
5. $\overline{\sigma}, i \models \mathsf{X}\phi$ iff $i + 1 < |\sigma|$ and $\overline{\sigma}, i + 1 \models \phi$;
6. $\overline{\sigma}, i \models \widetilde{\mathsf{X}}\phi$ iff either $i + 1 = |\sigma|$ or $\overline{\sigma}, i + 1 \models \phi$;
7. $\overline{\sigma}, i \models \mathsf{Y}\phi$ iff $i > 0$ and $\overline{\sigma}, i - 1 \models \phi$;
8. $\overline{\sigma}, i \models \mathsf{Z}\phi$ iff either $i = 0$ or $\overline{\sigma}, i - 1 \models \phi$;
9. $\overline{\sigma}, i \models \phi_1 \,\mathcal{U}\, \phi_2$ iff there exists $i \leq j < |\sigma|$ such that $\overline{\sigma}, j \models \phi_2$, and $\overline{\sigma}, k \models \phi_1$ for all k, with $i \leq k < j$;
10. $\overline{\sigma}, i \models \phi_1 \,\mathcal{S}\, \phi_2$ iff there exists $j \leq i$ such that $\overline{\sigma}, j \models \phi_2$, and $\overline{\sigma}, k \models \phi_1$ for all k, with $j < k \leq i$;
11. $\overline{\sigma}, i \models \phi_1 \,\mathcal{R}\, \phi_2$ iff either $\overline{\sigma}, j \models \phi_2$ for all $i \leq j < |\sigma|$, or there exists $k \geq i$ such that $\overline{\sigma}, k \models \phi_1$ and $\overline{\sigma}, j \models \phi_2$ for all $i \leq j \leq k$;
12. $\overline{\sigma}, i \models \phi_1 \,\mathcal{T}\, \phi_2$ iff either $\overline{\sigma}, j \models \phi_2$ for all $0 \leq j \leq i$, or there exists $k \leq i$ such that $\overline{\sigma}, k \models \phi_1$ and $\overline{\sigma}, j \models \phi_2$ for all $i \geq j \geq k$

A state sequence $\overline{\sigma}$ satisfies ϕ, written $\overline{\sigma} \models \phi$, if $\overline{\sigma}, 0 \models \phi$. Observe that some operators can be derived from a smaller set of so-called "primitive" ones. In particular, the $\wedge$ connective, the *release* operator ($\phi_1 \,\mathcal{R}\, \phi_2$), the *triggered* operator ($\phi_1 \,\mathcal{T}\, \phi_2$), and the *weak yesterday* operator ($\mathsf{Z}\phi$) can be defined in terms of the $\vee$ connective, the *until* operator ($\phi_1 \,\mathcal{U}\, \phi_2$), the *since* operator ($\phi_1 \,\mathcal{S}\, \phi_2$), and the *yesterday* operator ($\mathsf{Y}\phi$), respectively. However, here we consider them as primitive operators as well, since this allows us to put any formula into *negation normal form* (NNF), i.e. with the negations applied only to propositions, 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 $\mathsf{Y}\phi$ (ϕ holds at the previous state) and the *weak yesterday* operator $\mathsf{Z}\phi$ (ϕ holds at the previous state, if it exists). Similarly, when the logic is interpreted over finite state sequences, we have to distinguish between the *tomorrow* operator $\mathsf{X}\phi$ (ϕ holds at the next state) and the *weak tomorrow* operator $\widetilde{\mathsf{X}}\phi$ (ϕ holds at the next state, if it exists). When the logic is interpreted over infinite state sequences, the semantics of the *tomorrow* and *weak tomorrow* operators coincide. We nevertheless keep them separated for uniformity. When interpreted over finite state sequences, the logic is often referred to as $\mathsf{LTL_f}$.

The notion of *closure* of a formula will be useful later. It is defined as follows.

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 $\psi_1 \,\mathcal{U}\, \psi_2 \in \mathcal{C}(\psi)$, then $\mathsf{X}(\psi_1 \,\mathcal{U}\, \psi_2) \in \mathcal{C}(\psi)$;
5. if $\phi_1 \,\mathcal{R}\, \psi_2 \in \mathcal{C}(\psi)$, then $\widetilde{\mathsf{X}}(\psi_1 \,\mathcal{R}\, \psi_2) \in \mathcal{C}(\psi)$;
6. if $\psi_1 \,\mathcal{S}\, \psi_2 \in \mathcal{C}(\psi)$, then $\mathsf{Y}(\psi_1 \,\mathcal{S}\, \psi_2) \in \mathcal{C}(\psi)$;
7. if $\psi_1 \,\mathcal{T}\, \psi_2 \in \mathcal{C}(\psi)$, then $\mathsf{Z}(\psi_1 \,\mathcal{T}\, \psi_2) \in \mathcal{C}(\psi)$.

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

3.2 The One-Pass and Tree-Shaped Tableau for LTL+Past

In this subsection we will describe the tableau system for LTL+Past introduced by Geatti et al. [28]. This extends the tableau system for LTL by Reynolds [45], and will be used as the basis for the direct encoding discussed in Sect. 4. As presented by Geatti et al. [28], the tableau rules only handle LTL+Past interpreted over *infinite* state sequences. However, the modifications to support finite state sequences are straightforward, as will be shown in Sect. 4.

Every LTL+Past formula can be trivially transformed in NNF, therefore, for ease of exposition, we will 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 by $\Gamma(u_0) = \{\phi\}$. This tree is built step-wise: at each step, a set of rules is applied to a leaf, until all branches have been either *accepted* or *rejected*. A tableau where this is the case is called *complete*.[2] Each rule either (i) adds one or more children to the current leaf or (ii) either accepts or rejects the current branch. Given a branch $\overline{u} = \langle u_0, \ldots, u_n \rangle$, the sequence of nodes $\langle u_i, \ldots, u_j \rangle$, for some $0 \leq i \leq j \leq n$, is denoted by $\overline{u}_{[i,j]}$.

One node is selected at each time step and it is subject to a number of *expansion rules*. These rules select a formula of the node label and expand the tableau according to its semantics, as shown in Table 1. Each expansion rule creates one or two children depending on the selected formula. 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. Only one child is created if $\Gamma_2(\phi)$ is empty. An *elementary* formula is either a proposition, a negated proposition, or a *tomorrow*, *weak tomorrow*, *yesterday*, or *weak yesterday* formula. The expansion rules in Table 1 always decompose formulas into their subformulas or elementary formulas. Hence, after repeated applications of the expansion rules, sooner or later a node that only contains *elementary* formulas is obtained. We call such a node a *poised* node. Elementary formulas of the form $\mathsf{X}(\phi_1 \,\mathcal{U}\, \phi_2)$ are called X-*eventualities*. An X-eventuality is a formula that, intuitively, requests something to be fulfilled later. Given an X-eventuality $\phi \equiv \mathsf{X}(\phi_1 \,\mathcal{U}\, \phi_2)$, ϕ is said to be *requested* in a node u if $\phi \in \Gamma(u)$, and it is said to be *fulfilled* in a node u if $\phi_2 \in \Gamma(u)$. An X-eventuality is *fulfilled* in a subsequence of a branch is it is fulfilled in at least one node of the subsequence.

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

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

$$\Gamma(u_{n+1}) = \{\alpha \mid \mathsf{X}\alpha \in \Gamma(u_n) \text{ or } \widetilde{\mathsf{X}}\alpha \in \Gamma(u_n)\}$$

FORECAST Let

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

For each subset $G'_n \subseteq G_n$ (including $\varnothing$), 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

[2] Of course, when checking the satisfiability of a formula, the tableau construction can stop as soon as a single accepted branch is found.

Table 1 Tableau expansion rules

Rule	$\phi \in \Gamma$	$\Gamma_1(\phi)$	$\Gamma_2(\phi)$
Disjunction	$\alpha \vee \beta$	$\{\alpha\}$	$\{\beta\}$
Conjunction	$\alpha \wedge \beta$	$\{\alpha, \beta\}$	
Until	$\alpha U \beta$	$\{\beta\}$	$\{\alpha, X(\alpha U \beta)\}$
Since	$\alpha S \beta$	$\{\beta\}$	$\{\alpha, Y(\alpha S \beta)\}$
Release	$\alpha R \beta$	$\{\alpha, \beta\}$	$\{\beta, wX(\alpha R \beta)\}$
Triggered	$\alpha T \beta$	$\{\alpha, \beta\}$	$\{\beta, Z(\alpha T \beta)\}$

adds a number of branches that nondeterministically guess formulas that may be needed to fulfil past requests coming from future states. For details on the FORECAST rule, we refer the reader to Geatti et al. [28].

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.

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.

Let u be a node. We define u^* as the closest ancestor of u that is a 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. Given a branch $\overline{u} = \langle u_0, \ldots, u_n \rangle$, with u_n a step node, the termination rules are the following ones, which are *checked in the following order*.

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

EMPTY If $\Gamma(u_n)$ does *not* contain *tomorrow* or *weak tomorrow* formulas, then $\overline{u}$ is accepted.

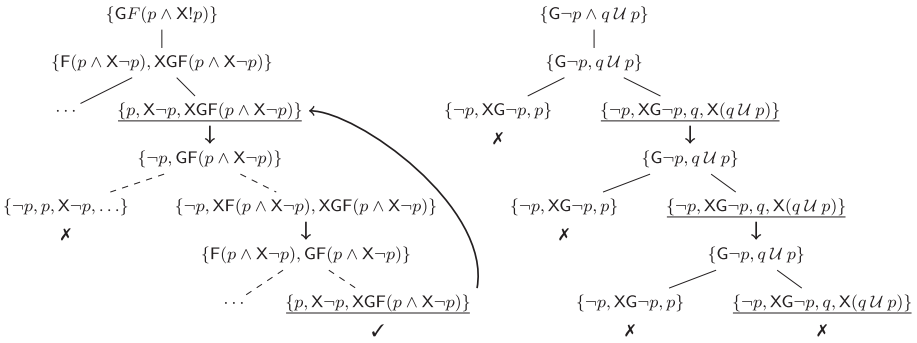
YESTERDAY If $\mathsf{Y}\alpha \in \Gamma(u_n)$, then the branch $\overline{u}$ is *rejected* if either u_n^* does not exist or $Y_n \nsubseteq \Gamma^*(u_n^*)$, where $Y_n = \{\psi \mid \mathsf{Y}\psi \in \Gamma(u_n)\}$.

W-YESTERDAY If $\mathsf{Z}\alpha \in \Gamma(u_n)$, then $\overline{u}$ is *rejected* if u_n^* exists and $Z_n \nsubseteq \Gamma^*(u_n^*)$, where $Z_n = \{\psi \mid \mathsf{Z}\psi \in \Gamma(u_n)\}$.

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 $\overline{u}_{[i+1,n]}$, then $\overline{u}$ is *accepted*.

PRUNE If there exist two positions i and j such that $i < j < n$, $\Gamma(u_i) = \Gamma(u_j) = \Gamma(u_n)$, and all the X-eventualities requested in these nodes which are fulfilled in $\overline{u}_{[j+1,n]}$ are also fulfilled in $\overline{u}_{[i+1,j]}$, then $\overline{u}$ 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. This scenario can be seen in Fig. 1a, that depicts the tableau for the satisfiable formula $\mathsf{GF}(p \wedge \mathsf{X}\neg p)$. Finally, the PRUNE rule, which was the main novelty of the system when introduced by Reynolds [45], rejects a branch that, otherwise, is going to be infinitely unrolled because of an X-eventuality impossible to fulfil. This happens in the tableau of the formula $\mathsf{G}\neg p \wedge q\,\mathcal{U}\,p$ depicted in Fig. 1b, where the X-eventuality $q\,\mathcal{U}\,p$ is never going to be fulfilled, hence the rightmost branch is closed by the PRUNE rule.



(a) Tableau for $\mathsf{GF}(p \wedge \mathsf{X}\neg p)$, closed by the LOOP rule.

(b) Tableau for $\mathsf{G}\neg p \wedge q\,\mathcal{U}\,p$, closed by the PRUNE rule.

Fig. 1 Example tableaux for two formulae, involving the LOOP and PRUNE rules. Dashed edges represent subtrees collapsed to save space, bold arrows represent the application of a STEP rule to a poised label

The following has been proved to hold.

Proposition 1 *(*Termination, soundness and completeness of Reynolds' tableau [28]*) Let ϕ be an* LTL+Past *formula. The complete tableau for ϕ is* finite*. Moreover, it contains an accepted branch if and only if ϕ is satisfiable.*

Besides what Proposition 1 states, it is useful to build an intuition of the correspondence between accepted branches of the tableau and models of the formulas. In particular, the *proof* of Proposition 1 [26] shows how to effectively build a model of the formula from an accepted tableau. This is accomplished by simply stating that any proposition appearing in the label of the i-th node of the branch holds at the i-th state of the model.

3.3 Adapting the LTL+Past Tableau to Finite Traces

We conclude the section by briefly discussing the finite-trace case. First of all, we observe that Reynolds' tableau, as presented above, assumes an infinite-trace semantics, and treats *tomorrow* and *weak tomorrow* formulas exactly in the same way (as they have exactly the same semantics in the infinite-trace case).

Adapting Reynolds' tableau to the finite-trace semantics is done as follows:

1. Remove the LOOP rule;
2. Change the EMPTY rule as follows:

   $\mathsf{EMPTY}_{\mathsf{FIN}}$ If $\Gamma(u_n)$ does *not* contain *tomorrow* formulas, then $\overline{u}$ is accepted.

3. Optionally, change the PRUNE rule as follows:

   $\mathsf{PRUNE}_{\mathsf{FIN}}$ If there is a $j < n$ such that $\Gamma(u_j) = \Gamma(u_n)$, then $\overline{u}$ is *rejected*.

Intuitively, by removing the LOOP rule, we ignore infinite periodic models, which end up being rejected by the PRUNE rule instead. In this way, we focus only on finite models accepted by the EMPTY rule, which is changed to accept branches with pending *weak tomorrow* requests, thus respecting the semantics of the *weak tomorrow* operator on finite traces. The $\mathsf{PRUNE}_{\mathsf{FIN}}$ rule can optionally be used instead of the PRUNE one. Completeness is ensured

in either case, but $\mathsf{PRUNE}_{\mathsf{FIN}}$ can amount to a considerable speedup, because it can prune branches much earlier.

The proofs of *termination*, *soundness* and *completeness* of the tableau given in [28] (Proposition 1) work pretty much unchanged in the finite-trace case with the above changes. In particular:

1. termination is unaffected, since the same argument (see [28], Theorem 1) applies to both the PRUNE and the $\mathsf{PRUNE}_{\mathsf{FIN}}$ rules: the number of labels is finite, and therefore, sooner or later, a label will repeat in any long-enough branch. The PRUNE rule has to wait this to happen twice, while the $\mathsf{PRUNE}_{\mathsf{FIN}}$ can reject the branch straightaway.
2. soundness is unaffected, since the same arguments that worked for the EMPTY and the LOOP rule work now for the $\mathsf{EMPTY}_{\mathsf{FIN}}$ rule (see [28], Theorem 2);
3. the arguments required to show completeness are unaffected if the PRUNE rule is used (see [28], Theorem 3). If the $\mathsf{PRUNE}_{\mathsf{FIN}}$ is used, completeness is much easier to show. If a branch $\overline{u} = \langle u_0, \ldots, u_n \rangle$ has a position $j < n$ such that $\Gamma(u_j) = \Gamma(u_n)$, then the whole subtree obtained by expanding u_n will also be found as the subtree of u_j. Hence, any successful branch obtained by expanding u_n can also be obtained by expanding u_j, hence the search can stop at u_n and $\overline{u}$ can be rejected.

Note that the argument at Item 3 above does not work for infinite traces, i.e. $\mathsf{PRUNE}_{\mathsf{FIN}}$ would break completeness in that setting, because infinite traces may have to periodically fulfill multiple eventualities (e.g. $\mathsf{F}\alpha \wedge \mathsf{F}\beta$), and partially fulfilling loops can still be valuable. See the counterexample shown by Geatti et al. [26] in their Fig. 2.

In the following, we will assume to work with the $\mathsf{PRUNE}_{\mathsf{FIN}}$ rule when looking for finite traces.

4 A SAT-Based Procedure Based on Reynolds' Tableau

This section describes our SAT-based LTL satisfiability checking procedure based on Reynolds' tableau. The procedure exploits SAT solvers to find suitable branches of the tableau tree without expanding the tableau nodes explicitly. This exploration is performed up to a given depth k, for increasing values of k (bounded). The procedure is reported in Algorithm 1. The five formulas $[\![\phi]\!]^k$, $|\phi|^k$, $|\phi|^k_{\text{fin}}$, $|\phi|^k_T$, and $|\phi|^k_{T,fin}$ encode different rules of the tableau. Note that, at Line 7 of Algorithm 1, $|\phi|^k$ has to be used to solve the formula for the infinite-trace semantics, while $|\phi|^k_{\text{fin}}$ has to be used for the finite-trace semantics. A similar distinction applies to Line 10.

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:

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

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

Algorithm 1 Procedure for infinite (resp. finite) trace semantics

```
1: procedure IS-SAT(φ)
2:     k ← 0
3:     while True do
4:         if [[φ]]^k is UNSAT then
5:             return φ is UNSAT
6:         end if
7:         if |φ|^k (resp. |φ|^k_fin) is SAT then
8:             return φ is SAT
9:         end if
10:        if |φ|^k_T (resp. |φ|^k_{T,fin}) is UNSAT then
11:            return φ is UNSAT
12:        end if
13:        k ← k + 1
14:    end while
15: end procedure
```

1. any proposition letter from the original alphabet Σ;
2. the set $\{p_\psi \mid \psi \in \mathsf{XR}, \widetilde{\mathsf{X}}\mathsf{R}, \mathsf{YR}, \mathsf{ZR}\}$ of propositions that are the *surrogate* version of the corresponding X-, Y-, and Z-*formulas*;
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.

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 belongs to the label of the 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_S the formula where all the X-, $\widetilde{\mathsf{X}}$-, Y- and Z-*formulas* are replaced by their surrogate 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_S^k to denote $(\psi_S)^k$.

The formula $[\![\phi]\!]^k$ is called the *k-unraveling* of ϕ, and it encodes the expansion of the tableau tree. To define it, we need an encoding of the expansion rules of Table 1.

Definition 3 (*Stepped normal form*) Let ϕ be an LTL+Past formula in NNF. Its *stepped normal form*, denoted by $\mathrm{snf}(\phi)$, is defined as follows:

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

The stepped normal form is the extension to past operators of the *next normal form* used by Geatti et al. [28]. It easily follows from the expansion rules of each operator in Table 1. We can now define the k-unraveling of ϕ recursively as follows:

$$
[\![\phi]\!]^0 = \mathrm{snf}(\phi)_S \wedge \bigwedge_{\psi \in \mathsf{YR}} \neg \psi_S \wedge \bigwedge_{\psi \in \mathsf{ZR}} \psi_S;
$$

$$[\![\phi]\!]^{k+1} = [\![\phi]\!]^k \wedge T_k \wedge Y_k \wedge Z_k,$$

where

$$T_k \equiv \bigwedge_{\mathsf{X}\alpha \in \mathsf{XR}} \left((\mathsf{X}\alpha)_S^k \leftrightarrow \mathrm{snf}(\alpha)_S^{k+1} \right) \wedge \bigwedge_{\widetilde{\mathsf{X}}\alpha \in \widetilde{\mathsf{X}}\mathsf{R}} \left((\widetilde{\mathsf{X}}\alpha)_S^k \leftrightarrow \mathrm{snf}(\alpha)_S^{k+1} \right),$$

$$Y_k \equiv \bigwedge_{\mathsf{Y}\alpha \in \mathsf{YR}} \left((\mathsf{Y}\alpha)_S^{k+1} \leftrightarrow \mathrm{snf}(\alpha)_S^k \right)$$

$$Z_k \equiv \bigwedge_{\mathsf{Z}\alpha \in \mathsf{ZR}} \left((\mathsf{Z}\alpha)_S^{k+1} \leftrightarrow \mathrm{snf}(\alpha)_S^k \right).$$

The T_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 by the fact that only satisfying assignments of the formula are considered. Similarly, the FORECAST rule 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 $[\![\phi]\!]^k$ is unsatisfiable, all the branches of the tableau for ϕ (either the one for finite or infinite traces) are rejected before $k+1$ steps, as formally stated by the next lemma.

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

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

From branches to assignments. Let $\overline{u} = \langle u_0, \ldots, u_n \rangle$ be a branch that is either accepted, rejected by PRUNE (or $\mathsf{PRUNE_{FIN}}$), or longer than $k+1$ step nodes. Let $\overline{\pi} = \langle \pi_0, \ldots, \pi_m \rangle$ be the sequence of its step nodes. We define a truth assignment ν for $[\![\phi]\!]^k$ as follows. Since $[\![\phi]\!]^k$ contains stepped propositions from p^0 until p^k, for any given p, we need at most $k+1$ step nodes from $\overline{u}$, which, however, can be shorter if it is accepted or rejected by the PRUNE or $\mathsf{PRUNE_{FIN}}$ rules. Hence, let $\ell = \min\{m, k\}$. Moreover, let p_U be p, if $p \in \Sigma$, and ψ, if $p = \psi_S$, for some X-, $\widetilde{\mathsf{X}}$-, Y-, or Z-request ψ, i.e. $(\cdot)_U$ is the inverse of the $(\cdot)_S$ 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 or the $\mathsf{EMPTY_{FIN}}$ rule, the evaluation of any proposition p^j with $j > m$ can be chosen arbitrarily;
2. if the branch has been accepted by the LOOP rule or rejected by the PRUNE or $\mathsf{PRUNE_{FIN}}$ rules, then there is a position w such that $\Gamma(\pi_w) = \Gamma(\pi_m)$, and we continue by filling the truth assignment considering the successor of π_w as a successor of π_m.

It can be easily checked that the truth assignment built in this way satisfies $[\![\phi]\!]^k$.

From assignments to branches. Let ν be a truth assignment for $[\![\phi]\!]^k$. We use ν as a guide to navigate the tableau tree to find a suitable branch which is either accepted, rejected by PRUNE or $\mathsf{PRUNE_{FIN}}$, or has more than $k+1$ step nodes. To do that, we build a sequence of branch prefixes $\overline{u}_i = \langle u_0, \ldots, u_i \rangle$ where at each step we obtain $\overline{u}_{i+1}$ 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 $J : \mathbb{N} \to \mathbb{N}$ that maps positions j in $\overline{u}_i$ to indexes $J(j)$ such that, for all ψ, it holds that $\psi \in \Gamma(u_j)$ if and only if $\nu \models \mathrm{snf}(\psi)_S^{J(j)}$, i.e. we build a relationship between positions in the branch and steps in ν. As the base case, we put $\overline{u}_0 = \langle u_0 \rangle$ and $J(0) = 0$ so that the invariant holds since $\Gamma(u_0) = \{\phi\}$ and $\nu \models \mathrm{snf}(\phi)_S^0$ by the definition of $[\![\phi]\!]^k$. 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 $\mathsf{X}\alpha \in \Gamma(u_i)$ or $\widetilde{\mathsf{X}}\alpha \in \Gamma(u_j)$, we have $\alpha \in \Gamma(u_{i+1})$ by construction of the tableau. Note that $\mathrm{snf}(\mathsf{X}\alpha) = \mathsf{X}\alpha$ and $\mathrm{snf}(\widetilde{\mathsf{X}}\alpha) = \widetilde{\mathsf{X}}\alpha$, hence we know by construction that $\nu \models (\mathsf{X}\alpha)_S^{J(j)}$ (or $\nu \models (\widetilde{\mathsf{X}}\alpha)_S^{J(j)}$). Then, by definition of $[\![\phi]\!]^k$, we know that $\nu \models \mathrm{snf}(\alpha)_S^{J(j)+1}$, i.e. $\nu \models \mathrm{snf}(\alpha)_S^{J(i+1)}$. For the other direction, if $\nu \models \mathrm{snf}(\alpha)_S^{J(i+1)}$, then by definition of $[\![\phi]\!]^k$ we have both $\nu \models (\mathsf{X}\alpha)_S^{J(i)}$ and $\nu \models (\widetilde{\mathsf{X}}\alpha)_S^{J(i)}$, hence $\nu \models \mathrm{snf}(\mathsf{X}\alpha)_S^{J(i)}$ and $\nu \models \mathrm{snf}(\widetilde{\mathsf{X}}\alpha)_S^{J(i)}$, and thus $\mathsf{X}\alpha \in \Gamma(u_i)$ and $\nu \models \mathrm{snf}(\widetilde{\mathsf{X}}\alpha)_S^{J(i)}$, so by construction of the tableau it holds that $\alpha \in \Gamma(u_{i+1})$. Hence the invariant holds.
2. If the FORECAST rule has been applied to u_i, then there are n children $\{u_i^1, \ldots, u_i^n\}$ 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, $\psi \in \Gamma(u_{i+1})$ if and only if $\nu \models \mathrm{snf}(\psi)_S^{J(i+1)}$. 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, we proceed as follows.
   (a) If there is only one child, then it is chosen as u_{i+1}. In such a case, the applied rule is necessarily the CONJUNCTION one, applied to a formula $\psi \equiv \psi_1 \wedge \psi_2$, and thus $\psi_1, \psi_2 \in \Gamma(u_{i+1})$. By construction, $\nu \models \mathrm{snf}(\psi)_S^{J(i)}$, and thus $\nu \models \mathrm{snf}(\psi)_S^{J(i+1)}$. Since $\mathrm{snf}(\psi_1 \wedge \psi_2) = \mathrm{snf}(\psi_1) \wedge \mathrm{snf}(\psi_2)$, it holds that $\nu \models \mathrm{snf}(\psi_1)_S^{J(i+1)}$ and $\nu \models \mathrm{snf}(\psi_2)_S^{J(i+1)}$. As for the other direction, if $\nu \models \mathrm{snf}(\psi_1)_S^{J(i+1)}$ and $\nu \models \mathrm{snf}(\psi_2)_S^{J(i+1)}$, then $\nu \models \mathrm{snf}(\psi_1 \wedge \psi_2)_S^{J(i+1)}$, and thus $\nu \models \mathrm{snf}(\psi_1 \wedge \psi_2)_S^{J(i)}$. Then, by construction, it holds that $\psi_1 \wedge \psi_2 \in \Gamma(u_i)$, and thus $\psi_1, \psi_2 \in \Gamma(u_i)$. Hence, the invariant holds.
   (b) If there are two children u_i' and u_i'', then let us suppose the applied rule is the DISJUNCTION rule (similar arguments hold for the other rules). In this case, the rule has been applied to a formula $\psi \equiv \psi_1 \vee \psi_2$, and thus $\psi_1 \in \Gamma(u_i')$ and $\psi_2 \in \Gamma(u_i'')$. We know that $\nu \models \mathrm{snf}(\psi)_S^{J(i)}$, and hence $\nu \models \mathrm{snf}(\psi)_S^{J(i+1)}$. Since $\mathrm{snf}(\psi_1 \vee \psi_2) = \mathrm{snf}(\psi_1) \vee \mathrm{snf}(\psi_2)$, it holds that either $\nu \models \mathrm{snf}(\psi_1)_S^{J(i+1)}$ or $\nu \models \mathrm{snf}(\psi_2)_S^{J(i+1)}$. 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 holds as well, since if either $\nu \models \mathrm{snf}(\psi_1)_S^{J(i+1)}$ or $\nu \models \mathrm{snf}(\psi_2)_S^{J(i+1)}$, then $\nu \models \mathrm{snf}(\psi_1)_S^{J(i)}$ or $\nu \models \mathrm{snf}(\psi_2)_S^{J(i)}$, and thus $\nu \models \mathrm{snf}(\psi_1 \vee \psi_2)_S^{J(i)}$. Hence, $\psi_1 \vee \psi_2 \in \Gamma(u_i)$, and then either $\psi_1 \in \Gamma(u_{i+1})$ or $\psi_2 \in \Gamma(u_{i+1})$.

Let $\overline{u} = \langle u_0, \ldots, u_i \rangle$ be the branch prefix built as above explained, and let $\overline{\pi} = \langle \pi_0, \ldots, \pi_n \rangle$ be the sequence of its step nodes. As already pointed out, the descent stops when π_n is a leaf or when $n = k+1$. Note that any leaf is a step node, so $u_i = \pi_n$. In case we find a leaf, it is not possible that it has been rejected 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 is not possible that it has been rejected by the YESTERDAY rule, as that would mean there is some $\mathsf{Y}\alpha \in \Gamma(\pi_n)$, with $\alpha \notin \Gamma^*(\pi_{n-1})$, and we know that $\nu \models \mathrm{snf}(\mathsf{Y}\alpha)_S^{J(i)}$, and then $\nu \models (\mathsf{Y}\alpha)_S^{J(i)}$, since $\mathrm{snf}(\mathsf{Y}\alpha) = \mathsf{Y}\alpha$. Then, by definition of $[\![\phi]\!]^k$, we know that $\nu \models \mathrm{snf}(\alpha)_S^{J(i)-1}$. Since $u_i = \pi_n$ is a step node, $J(i) - 1 = J(j)$, for some j such that $u_j = \pi_{n-1}$, and thus $\nu \models \mathrm{snf}(\alpha)_S^{J(j)}$, and by construction we know that $\alpha \in \Gamma(u_j)$, which conflicts with the hypothesis that the YESTERDAY rule rejected the branch. By a similar argument, we can conclude that it is not possible that it has been rejected by the W-YESTERDAY rule. Hence, we found a branch that either is longer than $k+1$ step nodes, or has been accepted, or has been rejected by the PRUNE or $\mathsf{PRUNE_{FIN}}$ rules. □

The formulas $|\phi|^k$ and $|\phi|^k_{\mathrm{fin}}$ are called respectively the *base encoding* and the *finite base encoding* of ϕ and, in addition to the k-unraveling, include the encoding of the EMPTY and LOOP rules (in $|\phi|^k$), and of the $\mathsf{EMPTY_{FIN}}$ rule (in $|\phi|^k_{\mathrm{fin}}$). These are the rules that accept the branches. The two formulas are defined as follows:

$$|\phi|^k \equiv [\![\phi]\!]^k \wedge ((E_k \wedge \widetilde{E}_k) \vee L_k)$$
$$|\phi|^k_{\mathrm{fin}} \equiv [\![\phi]\!]^k \wedge E_k$$

where the formulas E_k and $\widetilde{E}_k$, that together encode the EMPTY and $\mathsf{EMPTY_{FIN}}$ rules, are defined as follows:

$$E_k \equiv \bigwedge_{\psi \in \mathsf{XR}} \neg \psi_S^k \qquad \widetilde{E}_k \equiv \bigwedge_{\psi \in \widetilde{\mathsf{X}}\mathsf{R}} \neg \psi_S^k$$

and the formula L_k, that encodes the LOOP rule, is defined as follows:

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

where

$${}_lR_k \equiv \bigwedge_{\psi \in \mathsf{XR} \cup \widetilde{\mathsf{X}}\mathsf{R} \cup \mathsf{YR} \cup \mathsf{ZR}} (\psi_S^l \leftrightarrow \psi_S^k)$$

$${}_lG_k \equiv \bigwedge_{\mathsf{Y}\alpha \in \mathsf{YR}} (\mathsf{Y}\alpha)_S^{l+1} \leftrightarrow \mathrm{snf}(\alpha)_S^k \wedge \bigwedge_{\mathsf{Z}\alpha \in \mathsf{ZR}} (\mathsf{Z}\alpha)_S^{l+1} \leftrightarrow \mathrm{snf}(\alpha)_S^k$$

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

Intuitively, ${}_lR_k$ encodes the presence of two nodes whose labels contain the same requests for the next and the previous nodes. Note that the LOOP rule demands the two nodes to have *the same labels*, while here we are checking something looser. Hence, we need ${}_lG_k$ to be sure that the two nodes can be used to loop, and in particular, that the past requests at the step $l+1$ are fulfilled at step k. This turns out to be sufficient (see Lemma 3 below). Then, ${}_lF_k$ checks that all the X-eventualities are fulfilled between those nodes. The following result shows that $|\phi|^k$ encodes tableau trees where at least one branch is accepted in $k+1$ steps.

Lemma 2 *Let ϕ be an* LTL+Past *formula. If the complete tableau for infinite-trace (resp. for finite-trace) semantics for ϕ contains an accepted branch of $k+1$ step nodes, then $|\phi|^k$ (resp. $|\phi|^k_{fin}$) is satisfiable.*

Proof Suppose that the complete tableau (either for finite- or infinite-trace semantics) for ϕ contains an accepted branch of $k+1$ step nodes, so let $\overline{u} = \langle u_0, \ldots, u_n \rangle$ be such a branch, and let $\overline{\pi} = \langle \pi_0, \ldots, \pi_k \rangle$ be the sequence of its step nodes. By Lemma 1, $[\![\phi]\!]^k$ is satisfiable. We can then build a truth assignment ν, in the same way as in the proof of Lemma 1, such that $\nu \models [\![\phi]\!]^k$. Remember that this means that we set $\nu(p^i) = \top$ if and only if $p_U \in \Gamma(\pi_i)$, for all $0 \leq i \leq k$. Thus, depending on the semantics we have to prove that ν satisfies either $E_k \wedge \widetilde{E}_k$ or L_k (for the infinite-trace semantics) or just E_k (for the finite-trace semantics). To this end, we need to preliminarily show that $\psi \in \Gamma^*(\pi_i)$ if and only if $\nu \models \mathrm{snf}(\psi)^i_S$. Such a statement can be proved by induction on the structure of ψ, by exploiting the definition of the expansion rules of the tableau.

Now, we distinguish three cases, depending on which rule accepted the branch.

1. If, in the tableau for infinite-trace semantics, the branch was accepted by the EMPTY rule, then $\Gamma(\pi_k)$ does not contain *tomorrow* or *weak tomorrow* formulas. By definition of ν, it follows that $\nu \models \neg\psi^k_S$, for any $\psi \in \mathsf{XR} \cup \widetilde{\mathsf{X}}\mathsf{R}$, and thus E_k and $\widetilde{E}_k$ are satisfied.
2. If, in the tableau for finite-trace semantics, the branch was accepted by the $\mathsf{EMPTY}_{\mathsf{FIN}}$ rule, a similar reasoning implies that E_k is satisfied.
3. If the branch was accepted by the LOOP rule, then there exists a node π_l such that $\Gamma(\pi_l) = \Gamma(\pi_k)$. By definition of ν, it holds that $\nu \models \psi^l_S$ if and only if $\nu \models \psi^k_S$, for any $\psi \in \mathsf{XR} \cup \widetilde{\mathsf{X}}\mathsf{R} \cup \mathsf{YR} \cup \mathsf{ZR}$, and thus ${}_lR_k$ is satisfied. Moreover, since $\Gamma(\pi_l) = \Gamma(\pi_k)$, any past request $\mathsf{Y}\alpha$ or $\mathsf{Z}\alpha$ contained in $\Gamma(\pi_{l+1})$, that by construction is fulfilled in π_l, is also fulfilled in π_k, hence ${}_lG_k$ is satisfied as well. Then, we know that for any X-eventuality $\psi \equiv \mathsf{X}(\psi_1 \,\mathcal{U}\, \psi_2)$ requested in $\Gamma(\pi_k)$, ψ has been fulfilled between π_l and π_k, i.e. there exists $l < j \leq k$ such that $\psi_2 \in \Gamma^*(\pi_j)$. Hence, it holds that $\nu \models \mathrm{snf}(\psi_2)^j_S$, and thus ${}_lF_k$ is satisfied. Then, ${}_lR_k \wedge {}_lG_k \wedge {}_lF_k$ is satisfied for at least one l, so L_k is satisfied. □

Lemma 3 *Let ϕ be an* LTL+Past *formula. If $|\phi|^k$ (resp. $|\phi|^k_{fin}$) is satisfiable, then the complete tableau for ϕ for infinite-trace (resp. finite-trace) semantics contains an accepted branch.*

Proof Suppose that $|\phi|^k$ (resp. $|\phi|^k_{\mathrm{fin}}$) is satisfiable. Hence, there exists a truth assignment ν such that $\nu \models |\phi|^k$ (resp. $\nu \models |\phi|^k_{\mathrm{fin}}$). Then, $[\![\phi]\!]^k$ is satisfiable, and we know from Lemma 1 that the complete tableau for ϕ has a branch that is either accepted, rejected by PRUNE (or $\mathsf{PRUNE}_{\mathsf{FIN}}$), or longer than $k+1$ step nodes. Let $\overline{u} = \langle u_0, \ldots, u_n \rangle$ be the branch prefix found as shown in the proof of Lemma 1, and let $\overline{\pi} = \langle \pi_0, \ldots, \pi_m \rangle$ be the sequence of its step nodes. By construction, there exists a function $J : \mathbb{N} \to \mathbb{N}$ fulfilling the invariant: $\psi \in \Gamma(u_i)$ if and only if $\nu \models \mathrm{snf}(\psi)^{J(i)}_S$. We now show that indeed $\overline{u}$ is accepted or is the prefix of an accepted branch. Now we distinguish whether we are talking about $|\phi|^k$ or $|\phi|^k_{\mathrm{fin}}$:

1. If $|\phi|^k$ is satisfiable, either $E_k \wedge \widetilde{E}_k$ or L_k are satisfiable as well. We distinguish the two cases:
   (a) If $E_k \wedge \widetilde{E}_k$ is satisfiable, then $\nu \models \neg\psi^k_S$ for each $\psi \in \mathsf{XR} \cup \widetilde{\mathsf{X}}\mathsf{R}$. Since ψ is an X- or $\widetilde{\mathsf{X}}$-request, $\mathrm{snf}(\psi) \equiv \psi$, and thus $\nu \not\models \mathrm{snf}(\psi)^k_S$. Here, $k = J(j)$, for some j, and, from the invariant, it follows that $\psi \notin \Gamma(u_j)$. Hence, u_j does not contain any X- or $\widetilde{\mathsf{X}}$-request, triggering the EMPTY rule that accepts the branch.
   (b) If L_k is satisfiable, so are ${}_lR_k$, ${}_lG_k$ and ${}_lF_k$, for some $0 \leq l < k$. From ${}_lR_k$, we have that $\nu \models \psi^l_S$ if and only if $\nu \models \psi^k_S$ for all $\psi \in \mathsf{XR} \cup \widetilde{\mathsf{X}}\mathsf{R} \cup \mathsf{YR} \cup \mathsf{ZR}$, that is, $\nu \models \mathrm{snf}(\psi)^l_S$ if and only if $\nu \models \mathrm{snf}(\psi)^k_S$, because ψ is an X-, $\widetilde{\mathsf{X}}$-, 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, *w.l.o.g.* we can assume that u_i and u_j are step nodes, and by the invariant it holds that $\psi \in \Gamma(u_i)$ if and only if $\psi \in \Gamma(u_j)$, that is, u_i

and u_j have the same X-, $\widetilde{\mathsf{X}}$-, Y-, and Z-requests. Similarly, the fact that $\nu \models {}_l F_k$ 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.

2. If $|\phi|^k_{\text{fin}}$ is satisfiable, then E_k is satisfiable, and the same reasoning applied above for $E_k \wedge E_k$ applies, to conclude that the $\mathsf{EMPTY}_{\mathsf{FIN}}$ has accepted the branch. □

Finally, the formula $|\phi|^k_T$ and $|\phi|^k_{T,fin}$, called the *termination encoding* and the *finite termination encoding*, and encode the PRUNE and the $\mathsf{PRUNE}_{\mathsf{FIN}}$ rule, respectively. The formulas are defined as follows:

$$|\phi|^k_T \equiv [\![\phi]\!]^k \wedge \bigwedge_{i=0}^{k} \neg P^i$$

$$|\phi|^k_{T,fin} \equiv [\![\phi]\!]^k \wedge \bigwedge_{i=0}^{k} \neg P^i_{fin}$$

where

$$P^k_{fin} \equiv \bigvee_{j=0}^{k-1} {}_j R_k$$

$$P^k \equiv \bigvee_{l=0}^{k-2} \bigvee_{j=l+1}^{k-1} \left({}_l R_j \wedge {}_j R_k \wedge {}_l P^k_j \right),$$

$${}_l P^k_j \equiv \bigwedge_{\substack{\psi \in \mathsf{XEV} \\ \psi \equiv \mathsf{X}(\psi_1 \mathcal{U} \psi_2)}} \left(\psi^k_S \wedge \bigvee_{i=j+1}^{k} \mathrm{snf}(\psi_2)^i_S \rightarrow \bigvee_{i=l+1}^{j} \mathrm{snf}(\psi_2)^i_S \right).$$

It can be shown that $|\phi|^k_T$ or $|\phi|^k_{T,fin}$ are unsatisfiable if the tableau for ϕ, for infinite or finite traces respectively, contains only rejected branches.

Lemma 4 *Let ϕ be an* LTL+Past *formula. If $|\phi|^k_T$ (resp. $|\phi|^k_{T,fin}$) is unsatisfiable, then the complete tableau for ϕ for infinite traces (resp. finite traces) contains only rejected branches.*

Proof We prove the contrapositive, that is, if the complete tableau for ϕ contains an accepted branch, then $|\phi|^k_T$ (or $|\phi|^k_{T,fin}$) is satisfiable. Let $\overline{u} = \langle u_0, \ldots, u_n \rangle$ be such a branch, and let $\overline{\pi} = \langle \pi_0, \ldots, \pi_m \rangle$ be the sequence of its step nodes. By Lemma 1, we know $[\![\phi]\!]^k$ is satisfiable, thus we can obtain a truth assignment ν such that $\nu \models [\![\phi]\!]^k$. We can build ν as in the proof of Lemma 1, that is, 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 2, it holds that $\psi \in \Gamma^*(\pi_i)$ if and only if $\nu \models \mathrm{snf}(\psi)^i_S$. Now, since the branch is accepted, neither the PRUNE nor the $\mathsf{PRUNE}_{\mathsf{FIN}}$ rule can be applied to it. This has the following consequences, depending on whether we are dealing with infinite or finite traces:

1. for infinite traces, it means that either (i) there are no three nodes π_u, π_v, π_w such that $\Gamma(\pi_u) = \Gamma(\pi_v) = \Gamma(\pi_w)$, or (ii) these three nodes exist, but there is an X-eventuality ψ, requested in $\Gamma(\pi_w)$, which is fulfilled between π_u and π_v and not between π_v and π_w. In case (i), this means that ${}_uR_v \wedge {}_vR_w$ does not hold for any u and v. In case (ii), ${}_uR_v \wedge {}_vR_w$ holds, but ${}_uP_v^w$ does not. In both cases, it follows that $\neg P^i$ holds for any $0 \leq i \leq k$, and thus $|\phi|_T^k$ is satisfied;
2. for finite traces, it means that there are no two nodes π_u, π_w such that $\Gamma(\pi_v) = \Gamma(\pi_w)$. A similar reasoning as above concludes that $|\phi|_{T,fin}^k$ is satisfied. □

Together with the soundness and completeness results for the underlying tableau (Proposition 1), the above Lemmata allow us to prove the soundness and completeness of Algorithm 1.

Theorem 1 (Soundness and completeness) *Let ϕ be an* LTL+Past *formula. Algorithm 1 always terminates and answers* SAT *on ϕ if and only if ϕ is satisfiable.*

Proof As for the left-to-right direction, suppose that the procedure at Algorithm 1 answers SAT on the formula ϕ. Then, it means there is a $k \geq 0$ such that $|\phi|^k$ is satisfiable. By Lemma 3, the complete tableau for ϕ has an accepting branch. By the soundness of the tableau, ϕ is satisfiable.

As for the right-to-left direction, suppose that the formula ϕ is satisfiable. By the completeness of the tableau, the complete tableau for ϕ has an accepting branch. Let us suppose that such a branch has $k+1$ step nodes, for some $k \geq 0$. Then, we have to show that the procedure eventually answers SAT. Let $i < k$ be any earlier iteration of the main loop of the algorithm. By Lemma 1, $[\![\phi]\!]^i$ is satisfiable because there is a branch longer than $i+1$ step nodes. Similarly, by Lemma 4, $|\phi|_T^i$ (or $|\phi|_{T,fin}^i$) is satisfiable because not all the branches of the tableau are rejected. Hence, the algorithm does not answer UNSAT at step i. At step k, $|\phi|^k$ is satisfiable by Lemma 2, because the tableau has an accepted branch of $k+1$ step nodes, and thus the algorithm answers SAT.

To see the guarantee of termination, consider that if the formula is satisfiable, then by the above soundness argument we get a SAT answer, hence the algorithm terminates. Otherwise, the termination argument for the tableau ensures us that at some depth k all the branches will be closed by PRUNE (or $\text{PRUNE}_{\text{FIN}}$), hence $|\phi|_T^k$ (or $|\phi|_{T,fin}^k$) will be unsatisfiable, and the algorithm will return UNSAT. □

4.1 Incremental Algorithm

Algorithm 1 can be improved to better exploit the incrementality features of modern SAT solvers. We describe here how to do that in the case of infinite traces. The finite-traces case is very similar. Most solvers give the possibility to ask multiple related satisfiability queries without discarding the learnt clauses between subsequent calls. Depending on the cases, performance can dramatically increase in this way. Algorithm 2 is a variant of Algorithm 1 that shows how to exploit the incrementality of the underlying solver, making use of an *assumption-based* interface. With such an interface, supported by all major SAT and SMT solvers (see e.g. [21]), the satisfiability of multiple assertions can be asked, each time maintaining the clauses learnt from older assertions. Moreover, the satisfiability of the current set of assertions can be asked multiple times with a different set of *assumptions*, where learnt clauses that follow from the current assumption are discarded at the following satisfiability query. In this way, the procedure maintains learnt clauses in the SAT solvers between multiple iteration steps, allowing the solver to build a rich understanding of the tableau tree that is maintained step after step.

Algorithm 2 LTL satisfiability procedure, incremental version

```
1: procedure IS-SAT(φ)
2:     k ← 0
3:     s ← new Solver
4:     s.assert([[φ]]^0)
5:     while True do
6:         if s.solve() = UNSAT then
7:             return φ is UNSAT
8:         end if
9:         if s.solve-assuming((E_k ∧ Ẽ_k) ∨ L_k) = SAT then
10:            return φ is SAT
11:        end if
12:        s.assert(¬P^k)
13:        if s.solve() = UNSAT then
14:            return φ is UNSAT
15:        end if
16:        k ← k + 1
17:        s.assert(T_k ∧ Y_k ∧ Z_k)
18:    end while
19: end procedure
```

We will now define the assumption-based interface that we assume a SAT solver to support in order to make Algorithm 2 work correctly. Many solvers support this style of interfacing. An incremental solver supports these four operations:

1. **Initialization** The expression ***new*** *Solver* (used at Line 3 of Algorithm 2) returns a new object, say s, that keeps track of the *state* of the solver, that includes $A(s)$, a *Boolean formula* called the *current assertion* of the solver. After the initialization, $A(s) = \top$, initially.
2. **Assertion** Given a solver object s and a Boolean formula ϕ, a call to s.assert(ϕ) conjuncts ϕ to the current assertion, i.e. $A(s) \leftarrow A(s) \wedge \phi$.
3. **Solving** Given a solver object s, a call to s.solve() asks the solver to solve the satisfiability problem for the current assertion. That is, the call returns SAT if $A(s)$ is satisfiable, or UNSAT otherwise.
4. **Solving with assumptions** Given a solver object s and a Boolean formula ϕ, a call to s. solve-assuming (ϕ) asks the solver to solve the satisfiability problem for the current assertion set conjuncted with ϕ. That is, the call returns SAT if $A(s) \wedge \phi$ is satisfiable, or UNSAT otherwise. Note that the current assertion is not modified by the call.

The semantics of the above operations do not talk about what happens under the hood. In practice, a solver implementing these operations will keep as much information as possible between subsequent calls to solve() and solve-assuming(), to waste as little work as possible. This can dramatically speed up SAT based algorithms in many cases [22, 43].

Assuming the underlying SAT solver correctly implements the above interface, we can prove the correctness of Algorithm 2.

Theorem 2 (Incremental algorithm) *Let ϕ be an* LTL+Past *formula. Algorithm 2 always terminates and answers* SAT *if and only if ϕ is satisfiable.*

Proof Let ϕ be an LTL formula. At first, let us track the evolution of $A(s)$ through the execution of the algorithm. We track the calls to s.assert(), which is called at Line 4 (i.e. before the loop), at Line 12, and at Line 17 (at the end of the loop). It is easy to see that at each iteration

k, at the beginning of the loop (Line 6), the current assertion is as follows:

$$A(s) \equiv [\![\phi]\!]^0 \wedge \bigwedge_{i=1}^{k} (T_i \wedge Y_i \wedge Z_i) \wedge \bigwedge_{i=0}^{k-1} \neg P^i$$
$$\equiv [\![\phi]\!]^k \wedge \bigwedge_{i=0}^{k-1} \neg P^i$$

Note, instead, that after Line 12, at each iteration k, the assertion is as follows:

$$A(s) \equiv [\![\phi]\!]^k \wedge \bigwedge_{i=0}^{k-1} \neg P^i \equiv |\phi|_T^k$$

Now, suppose Algorithm 2 returns SAT at some iteration $k \geq 0$. This can only happen at Line 9. Then, the call to s.solve-assuming$((E_k \wedge \widetilde{E}_k) \vee L_k)$ returned SAT. This means $A(s) \wedge ((E_k \wedge \widetilde{E}_k) \vee L_k)$ is satisfiable. Notice that this implies $|\phi|^k$ is satisfiable, hence by Lemma 3, ϕ is satisfiable.

Conversely, suppose ϕ is satisfiable. By the completeness of the tableau, the complete tableau for ϕ has an accepting branch. Let us suppose that such a branch has $k+1$ step nodes, for some $k \geq 0$. For all iterations $i \leq k$, by Lemma 1, $[\![\phi]\!]^i$ is satisfiable, and by Lemma 4, $|\phi|_T^k$ is satisfiable. Hence, neither Line 7 nor Line 14 returns UNSAT at any iterations $i < k$. Instead, by Lemma 2, we know $|\phi|^k \equiv [\![\phi]\!]^k \wedge (E_k \wedge \widetilde{E}_k) \vee L_k$ is satisfiable, hence Line 10 correctly returns SAT.

Termination is proved along the lines of Theorem 1, considering that Line 13 effectively tests the satisfiability of $|\phi|_T^k$ at each iteration. □

4.2 Extraction of Models

We conclude the section by briefly describing an additional feature of the proposed encoding and algorithm: they can be used not only to decide the satisfiability of LTL+Past formulas, but also to extract a model for satisfiable ones.

Such an extraction consists of two steps. First of all, one has to ask the SAT solver to return the values of the proposition letters p^t from the assignment to $|\phi|^k$. These proposition letters tell the truth value of the proposition p at each time step t.

Next, in the case of infinite-state semantics, one has to extract the starting state of the loop of the periodic model identified by the LOOP rule, if any. To this end, a few propositions $\ell_{l,k}$ are introduced, for some k and some $l < k$, and the formula L_k, which is used in the base encoding $|\phi|^k$, is modified as follows:

$$L_k \equiv \bigwedge_{l=0}^{k-1} (\ell_{l,k} \leftrightarrow ({}_l R_k \wedge {}_l G_k \wedge {}_l F_k)) \wedge \bigvee_{l=0}^{k-1} \ell_{l,k}$$

It is clear that this formulation of L_k is equisatisfiable to the formulation of L_k that has been presented above. However, in this way, for a satisfiable formula φ, the first $\ell_{l,k}$ that turns out to be true tells us that the loop starts at time $t = l$. If no one of them holds, the branch was closed by the EMPTY rule and the model can be regarded as looping through its last state (if we are looking for an infinite model).

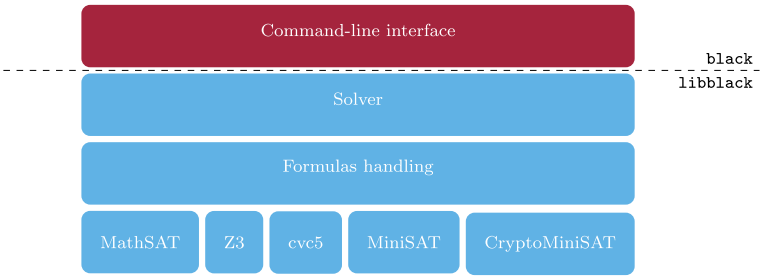


Fig. 2 High-level architecture of BLACK

5 Implementation

In order to perform the experimental evaluation described in the next section, the proposed algorithm has been implemented in a tool that has been called BLACK (Bounded Ltl sAtisfiability ChecKer). To ease reproducibility of the results, the tool is portable across all major platforms and runs on Windows, Linux, and macOS, and easy-to-install binary packages are provided for all the supported platforms. Developed over the course of a few years, BLACK evolved beyond being a simple prototype to test our algorithm. For this reason, this section describes BLACK's internal architecture and its main features.

5.1 BLACK's Library

An important feature of the tool is its embeddability into third-party applications: BLACK is implemented as a shared library, written in C++20, that can be linked by any client application, with a well-defined API that provides access to BLACK's solver and more. The tool itself is then just a thin command-line wrapper over the library. We will call BLACK's library libblack, and the BLACK tool black, to distinguish them from BLACK the project as a whole. A high-level account of the architecture of BLACK is given in Fig. 2.

The libblack library provides several facilities besides the solver itself:

1. basic supporting data structures and metaprogramming utilities;
2. mechanisms to create, parse, and manipulate LTL and propositional formulas;
3. an abstraction layer over the supported SAT solver backends.

The mechanisms to handle LTL formulas inside the library merit their own discussion. To ease memory management (one of the primary source of bugs in C++ programs), formulas are represented by objects whose lifetime is exclusively handled by a central `alphabet` object whose purpose is to create and keep track of created formulas. Formulas are created by asking the `alphabet` object to create new (objects representing) propositional symbols, and then combining such symbols with logical and temporal operators to obtain more complex formulas. All formulas are destroyed when the lifetime of the parent `alphabet` object ends. In order to save memory, equal formulas are reused. This means that creating twice the same formula (say, $p \wedge q$), will not create two different objects, but the same object is returned. This deduplicating mechanism, implemented through the use of an efficient hash table, ensures that checking equality of formulas (for example to implement memoization of recursive procedures) can be done in $\mathcal{O}(1)$ time with a simple and very fast pointer comparison. On top of that, formulas can of course also be created by parsing strings or text files.

Fig. 3 How to compute the depth of a formula with the `libblack` API

```
int depth(formula f) {
  return f.match(
    [](boolean) { return 1; },
    [](atom)    { return 1; },
    [](unary, formula op) {
      return depth(op) + 1;
    },
    [](binary, formula left, formula right) {
      return depth(left) + depth(right) + 1;
    }
  );
}
```

The above structure is similar to the API of many SMT solvers (such as Z3 [20] or MathSAT [14]), but there are two key differences. The first regards how propositional letters are labelled. In similar APIs, the *name* of a propositional symbol or variable is usually a string of some kind (`"p"`, `"q"`, etc.). Instead, libblack allows objects of any type to be used as the label of propositional letters. This means that labels of such symbols can carry complex information, and, in particular, propositional letters can be labelled by other formulas. Thanks to this particular feature, the *surrogating* transformation of a formula ϕ into its corresponding grounded symbol ϕ_S, used in the encoding described in Sect. 4, is essentially a no-op: the object representing ϕ_S is just a propositional symbol whose label is ϕ.

The second key ingredient is how formulas can be handled once they are created. In order to ease the implementation of several parts of the solver (the NNF, the stepped normal form, the encoding, etc.) the libblack library provides facilities to manipulate formulas using a functional-style pattern matching approach, instead of complex *visitor* patterns usually required for such scenarios in classical object-oriented approaches. As an example, the snippet of code shown in Fig. 3 computes the depth of a formula.

Besides the unavoidable noise introduced by the C++ syntax, the above snippet shows a clear connection with the formal definition of depth of a formula (which we do not need to recall here). Note that the burden of implementing this mechanism of formula manipulation is on the library, since the language does not natively support functional-style pattern matching on objects.

Thanks to these particular features, the whole encoding and solver algorithm are implemented in just over 400 lines of readable source code. Note that this formula manipulation layer works for both LTL and propositional formulas (which are just a subset of LTL formulas) and is useful beyond its use of feeding formulas to the solver, making libblack a flexible basic block for any application that needs to create, parse, and manipulate LTL formulas.

The second major purpose of the library is to provide access to the SAT solver backends. As already mentioned, libblack supports MiniSAT [21], CryptoMiniSAT [49], Z3 [20], cvc5 [3], and MathSAT [14] as backends. Supporting additional backends is a matter of implementing a few functions to plug into the abstraction layer. Support for further backends is planned for the future. The major difference in the API of MiniSAT and CryptoMiniSAT on one side, and Z3 and MathSAT on the other, is that the former two solvers accept input in *conjunctive normal form* (CNF), while the latter two accept free-form Boolean formulas. In order to support the CNF-based solvers, libblack implements a CNF conversion utility based on the classic Tseitin encoding. This is all hidden behind an abstraction layer that allows the client application to just create a generic `sat::solver` object, feed it with propositional formulas, and get the result of the SAT solving process.

It is also worth mentioning that BLACK supports a first-order extension of $\mathsf{LTL_f}$ called $\mathsf{LTL_f}$ *modulo theories* ($\mathsf{LTL_f^{MT}}$). The description of this logic is beyond the scope of this paper, but readers are redirected to the relevant literature [30].

6 Experimental Evaluation

In this section, we describe the experimental evaluation of BLACK against other state-of-the-art tools for the satisfiability of LTL, LTL+Past, and $\mathsf{LTL_f}$ formulas. Benchmarks consist of a set of input formulas (see later for a detailed account) over which the different tools have been run to measure their solving speed. All the benchmarks have been run on a 16-core AMD EPYC 7281 processor with 64GB of RAM, with a timeout of five minutes and a memory limit of 3GB for each formula. All the benchmark formulas and the supporting scripts are available in BLACK's source code repository.[3] In all the tests described here, BLACK has been run using the Z3 backend [20], which is the default backend included in BLACK's binary distributions.

6.1 LTL Over Infinite Traces

To evaluate BLACK's performance on LTL over infinite traces, we compared it with the following tools:

1. the nuXmv [11] model checker, both in SBMC and K-Liveness modes[4];
2. Aalta 2 [37], a tool based on an explicit graph-shaped tableau built with the help of a SAT solver[5];
3. ls4 [51], version 1.1,[6] a solver based on labelled superposition; we used the ltl2snf conversion tool to put the formulas in the required *separated normal form* [34][7];
4. pltl, a tableau-based tool that implements both a graph-shaped tableau [1] and a tree-shaped tableau *à la* Schwendimann [47];
5. Leviathan [4], a tool that directly implements the construction of Reynolds' tableau [45].[8]

Of these, nuXmv in SBMC mode and Leviathan are the most similar to BLACK, in different ways. The former implements an iterative model checking procedure that looks for counter-examples to the specification of length at most k, for increasing values of k, with a completeness check that ensures termination for unsatisfiable formulas [32]. Note that this means we do not need to specify a maximum value for k in SBMC mode. The latter is the first implementation of Reynolds' tableau, which is the tableau underlying BLACK's SAT encoding and algorithm, that Leviathan constructs explicitly. All the tools are run with their default options. In particular, they do not offer tuning options nor anything that can affect the result of the evaluation except for the operating mode of nuXmv (SBMC *vs.* K-Liveness) and pltl (graph- *vs.* tree-shaped tableau).

These tools have been tested on a total of 4181 formulas, which have been obtained from two different sources:

[3] https://github.com/black-sat/black.

[4] https://nuxmv.fbk.eu.

[5] https://github.com/lijwen2748/aalta.

[6] https://github.com/quickbeam123/ls4.

[7] https://nalon.org.

[8] https://github.com/Corralx/leviathan.

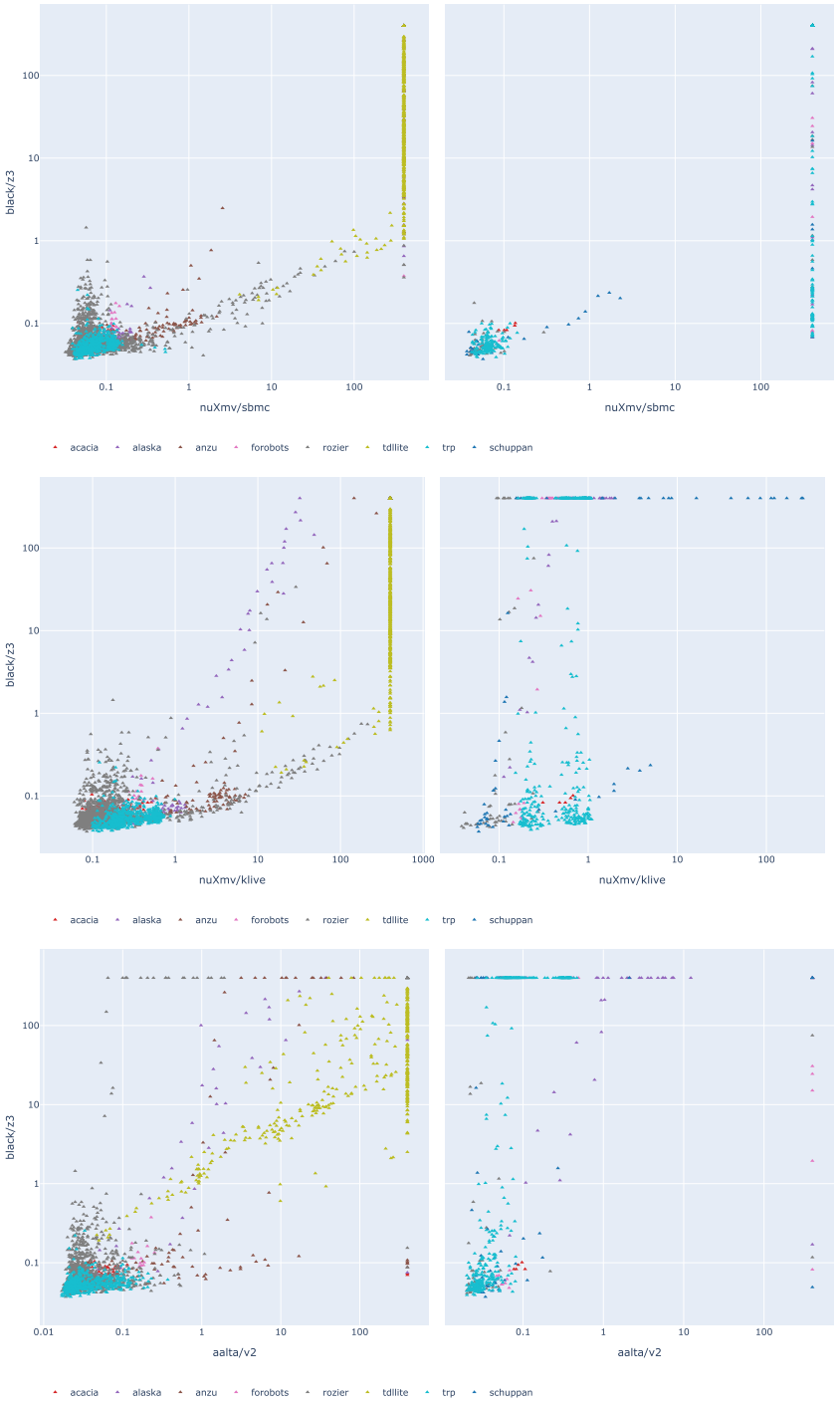


Fig. 4 Scatter plots of running time of BLACK versus nuXmv in SBMC (top row), nuXmv in K-live (middle row), and Aalta (bottom row) over LTL on infinite traces. The scale is in s. The colors distinguish different formulas families

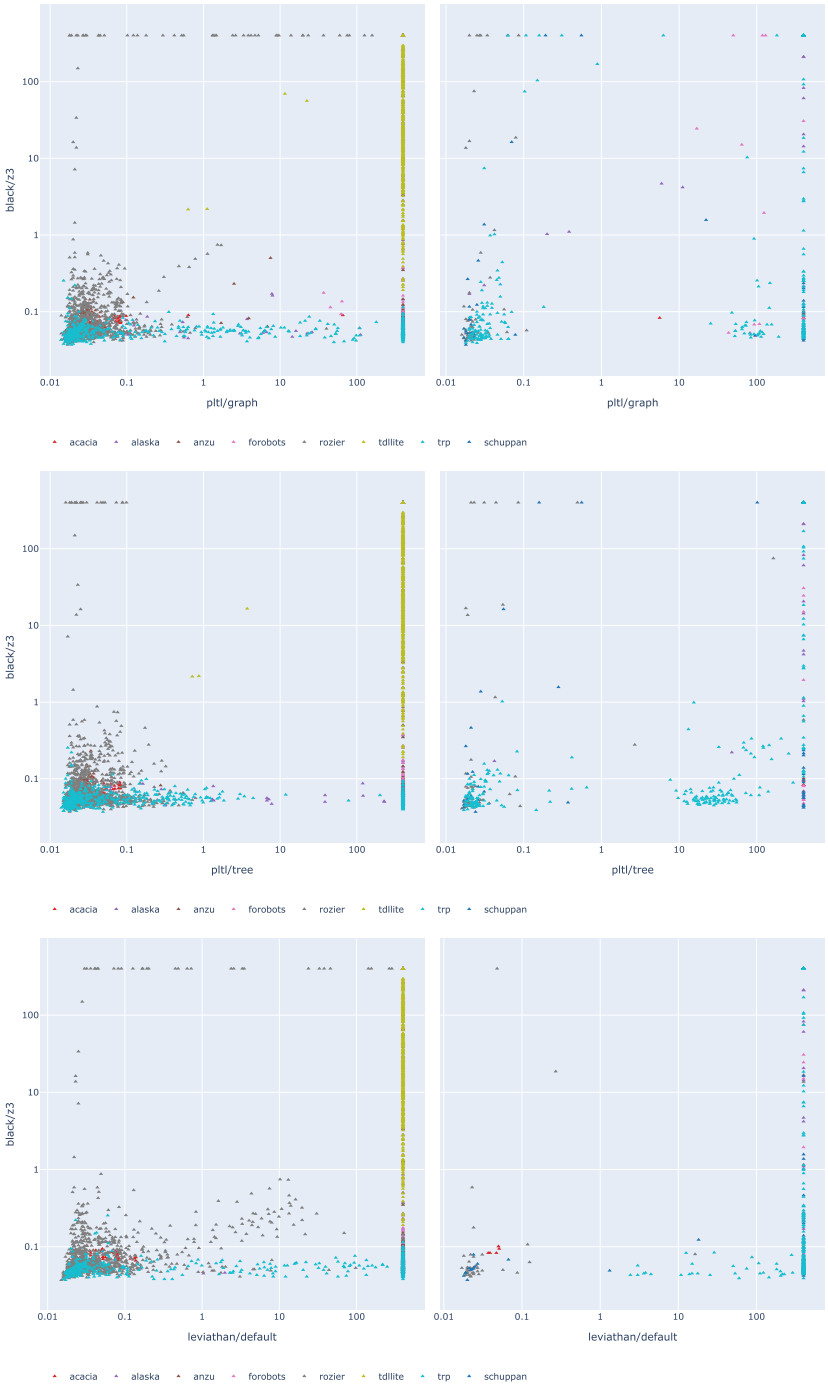


Fig. 5 Scatter plots of the running time of BLACK versus pltl in graph (top row), pltl in tree (middle row), and Leviathan (bottom row) over LTL on infinite traces. The scale is in s. The colors distinguish different formulas families

Fig. 6 Scatter plots of the running time of BLACK versus ls4 over LTL on infinite traces. The scale is in s. The colors distinguish different formulas families

1. the formulas collected by Schuppan and Darmawan [46], which include the `acacia`, `alaska`, `anzu`, `forobots`, `rozier`, `trp`, and `schuppan` sets[9];
2. a set of formulas coming from the LTL encoding of the temporal description logic TDL-lite, as described by Tahrat et al. [52], which are referred to in the plots as the `tdllite` set.

The results are shown in the scatter plots of Figs. 4, 5, 6, in the cactus plot of Fig. 9, and in the bar charts of Fig. 10. In the comparison with nuXmv, it can be seen that BLACK outperforms it in SBMC mode, which is the most similar approach to BLACK. With respect to nuXmv in K-Liveness mode, BLACK performs better over satisfiable formulas and suffers over unsatisfiable formulas. This is a pattern that repeats also in the comparison with Aalta. This suffering in unsatisfiable formulas can be explained with the cubic growth in size of the termination encoding $|\phi|_T^k$ at increasing values of k. On the other hand, in the comparison with the tableau-based tools Leviathan and pltl, it can be seen that BLACK performs considerably better both on satisfiable and unsatisfiable formulas. It is worthwhile to note that BLACK performs particularly well on the `tdllite` set, followed by Aalta and ls4.

6.2 LTL+Past

To evaluate BLACK over LTL+Past formulas, we compared it with nuXmv [11] which, as far as we know, is the only other tool available that directly supports past operators. As there are no readily available benchmark sets for LTL+Past in the literature, we came up with our own. The benchmark formulas consist of two sets:

1. the `random` set, which consists of 1300 randomly generated formulas of varying size, generated in a straightforward way, similarly to what was shown by Tauriainen and Heljanko [53] (269 UNSAT, 1011 SAT, 20 unknown);

[9] The original set contained also the negations of all the formulas but the benchmark set we used was downloaded from the StarExec platform [50], where the negations are not included.

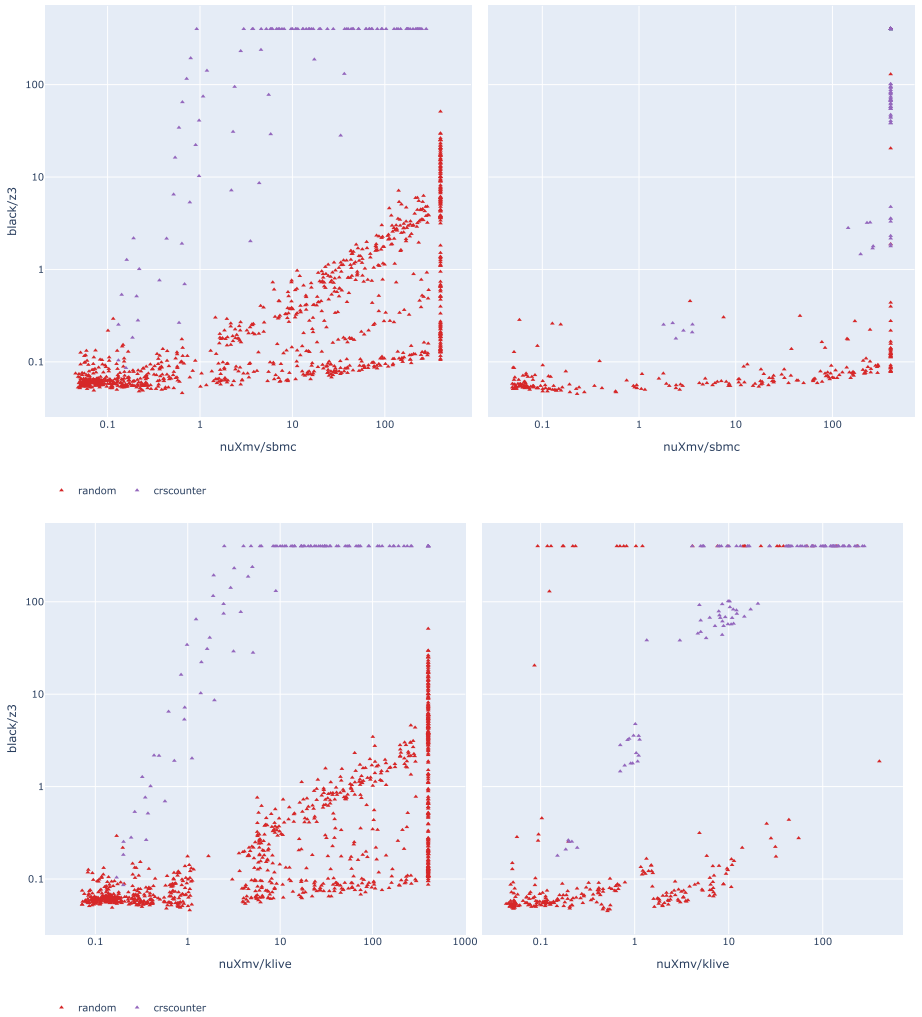


Fig. 7 Scatter plots of BLACK versus nuXmv in SBMC (top) and K-Live (bottom) modes, over LTL+Past formulas

2. the `crscounter` set, which consists of 240 formulas resulted from the adaptation to the satisfiability problem of a benchmark set for model checking provided by Cimatti et al. [13] (112 UNSAT, 101 SAT, 27 unknown).

The second family needs some explanations. In the original benchmark set for model checking, 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:

$$P(i) \equiv \neg \mathsf{F}\big(\mathsf{O}((c = \tfrac{N}{2}) \wedge \mathsf{O}((c = \tfrac{N}{2} + 1) \wedge \ldots \wedge \mathsf{O}(c = \tfrac{N}{2} + i)\ldots))\big).$$

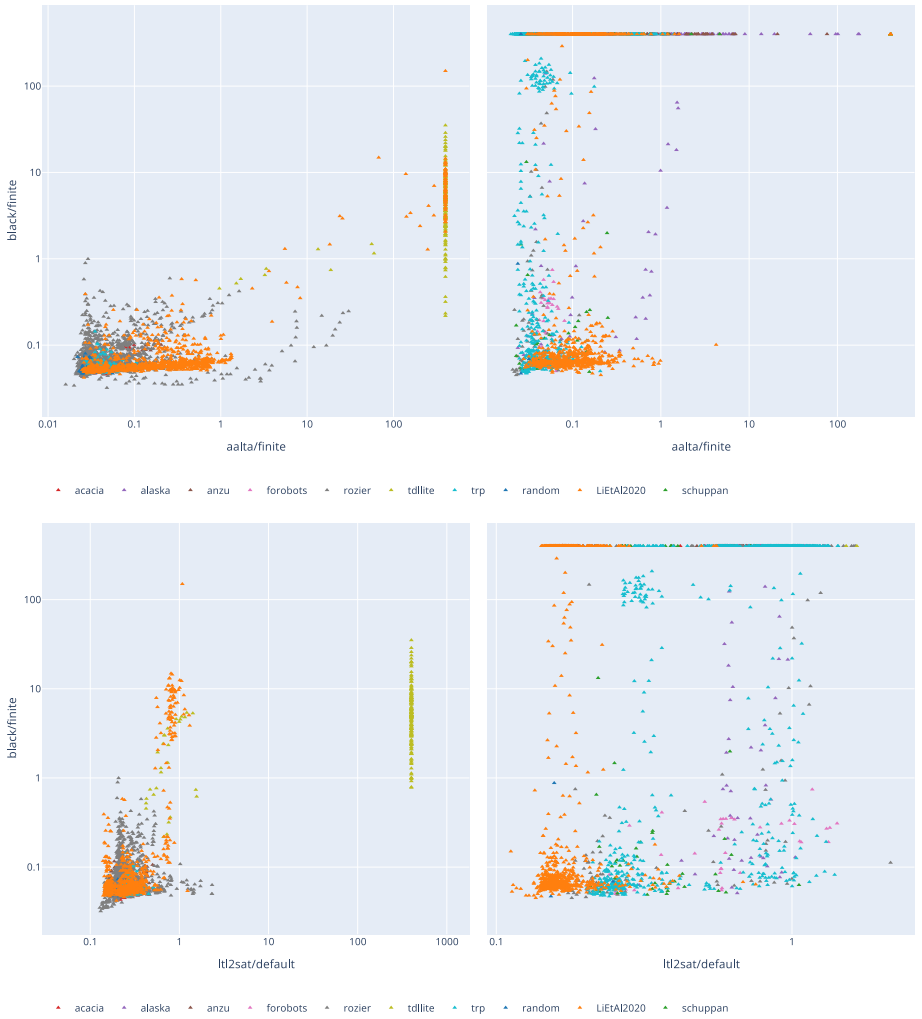


Fig. 8 Scatter plots of BLACK versus Aaltaf (top) and LTL2SAT (bottom) over LTL on finite traces

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.

Since these benchmarks were introduced in the context of model checking, 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 ($i > \frac{N}{2}$) formulas. Moreover, this family of formulas stresses the ability to process past operators and find short counterexamples, and thus it specifically challenges BLACK's performance.

The results are shown in the scatter plots of Fig. 7, in the cactus plot of Fig. 9, and in the bar charts of Fig. 10. As can be seen, again BLACK outperforms nuXmv in SBMC mode,

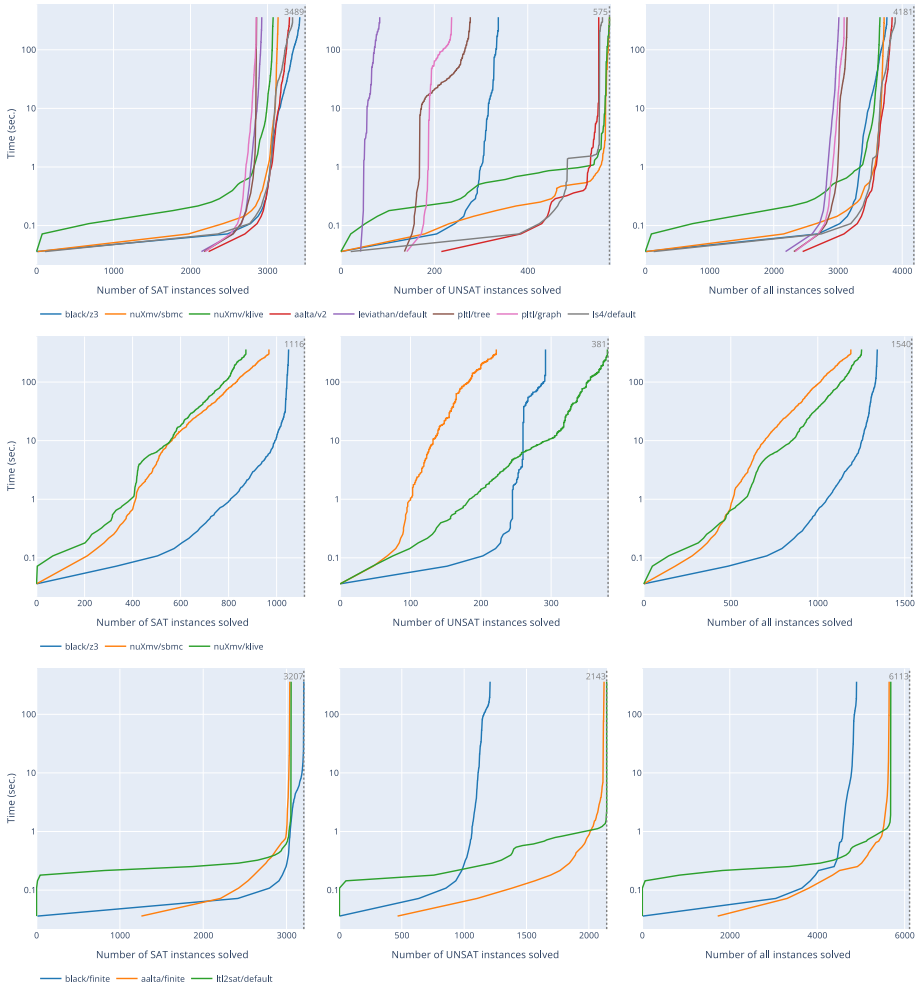


Fig. 9 Cactus plots for LTL over infinite traces (top row), LTL+Past (middle row), and LTL over finite traces (bottom row). The total number of formulas in the third column is higher than the sum of SAT and UNSAT formulas. This is because some formulas are not solved by any of the tools, i.e. we don't know if they are SAT or not. Therefore they are counted only when aggregating all the results

and is competitive with nuXmv in K-Liveness mode. The performance gain is more visible in satisfiable formulas, as in the future-only formulas of the previous sections. In general, BLACK's performance is better on random formulas. Note that, while for LTL (finite or infinite traces), most of the solvers solve most of the formulas in short time, general times needed to solve LTL+Past formulas are higher. The reason for this behaviour is that for the LTL+Past we have formulas (both the random and the crscounter sets) of increasing size. This translates into a slightly more uniform distribution of formulas over time with respect to the other cases where formulas tend to be fixed to a specific size.

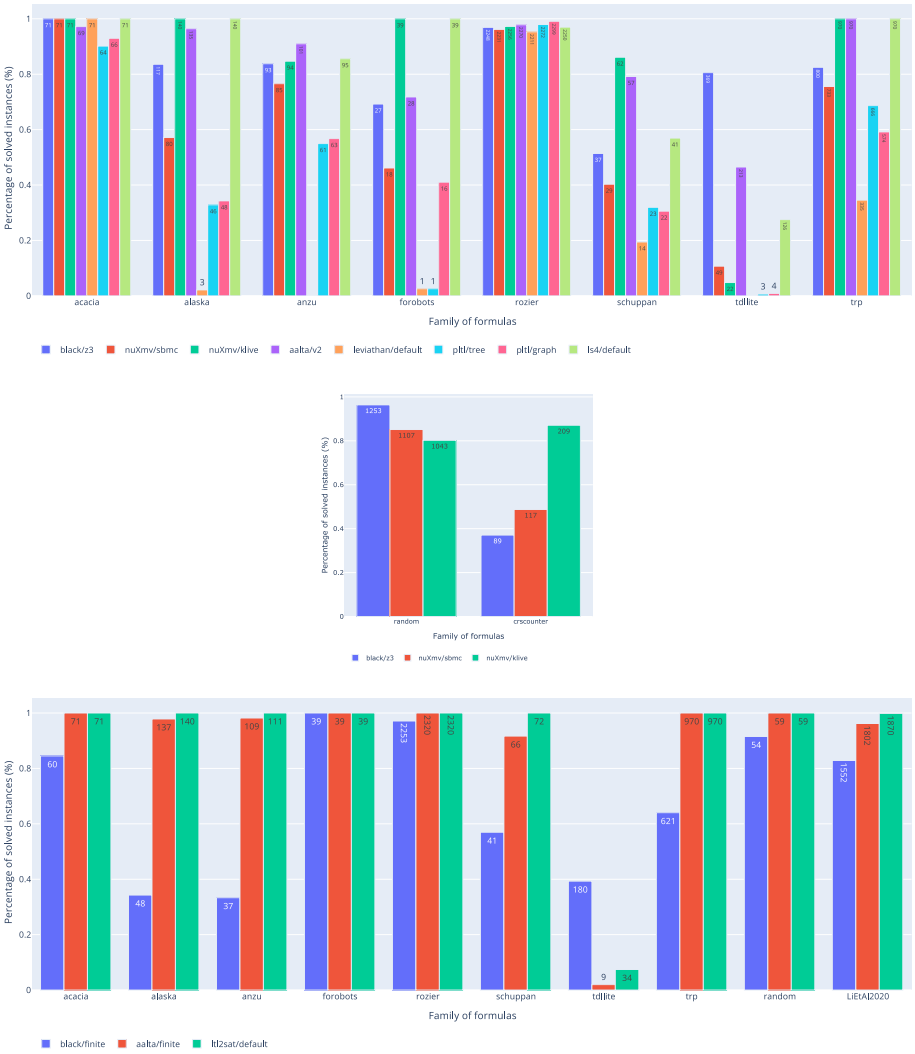


Fig. 10 Bar charts for LTL over infinite traces (top row), LTL+Past (middle row), and LTL over finite traces (bottom row)

6.3 LTL Over Finite Traces

Finally, we compared BLACK against two state-of-the-art tools for LTL over finite traces, namely Aaltaf [38],[10] which implements an algorithm, called *Conflict-Driven* $\mathsf{LTL_f}$ *Satisfiability Checking*, where a SAT-aided explicit tableau construction is paired with the extraction of unsatisfiable cores to prune the search space, and LTL2SAT [23],[11] a tool which employs a SAT-based reduction, but with specific heuristics for particular classes of formulas.

We compared the tools against the following sets of formulas:

[10] https://github.com/lijwen2748/aaltaf.

[11] https://ltl2sat.wordpress.com.

1. all the formula sets from the infinite traces case as described above, but interpreted over finite traces;
2. a set of 1875 formulas particularly crafted for finite traces, referred to as `LiEtAl2020` in the plots, taken from Li et al. [38] (1301 UNSAT, 567 SAT, 7 unknown).

The results are shown in the scatter plots of Fig. 8, in the cactus plot of Fig. 9, and in the bar charts of Fig. 10. The finite traces setting is the one where BLACK suffers the most. As usual, the performance is better on satisfiable formulas. As in the infinite trace setting, it is worthwhile to note that BLACK performs comparably particularly well on the `tdllite` set. It is worthwhile to note that, even though LTL2SAT generally has very good performance, it seems to suffer from some correctness problems since, in a non-negligible number of cases,[12] it reports results different from those reported by BLACK and Aaltaf.

7 Conclusions

In this paper, we presented a new SAT-based satisfiability checking technique for LTL, LTL+Past, and LTL_f, inspired by the one-pass and tree-shaped tableau by Reynolds [45]. We provided full correctness proofs and an extensive experimental evaluation of the performance of its implementation, included in the BLACK satisfiability checker, whose architecture has also been described. Performance-wise, the evaluation shows that our implementation of the technique is competitive with other state-of-the-art tools in most circumstances, especially on satisfiable formulas.

Many future developments are possible. From the point of view of the performance, it would be interesting to find heuristics to optimize the application of the PRUNE rule of the tableau in order to speed up the execution on unsatisfiable formulas. In particular, a linear-size encoding (similar to the ones for Bounded Model Checking introduced by Biere et al. [7]) would arguably provide a great speed-up in such cases.

Reynolds' tableau has been extended to other logics beyond LTL, such as TPTL. A similar SAT (or SMT) encoding for the TPTL tableau would allow BLACK to support this real-time logic as well. Similar extensions are being investigated for first-order extensions of LTL, e.g. LTL *modulo theories* [30].

As far as the tool itself is concerned, many improvements are possible, such as the support of more SAT backends, the improvement of the efficiency of the current CNF translation, the stabilization and documentation of the library API, the integration of more input and/or output formats. Moreover, the modular structure of the tool and of the library is not at all tied to the specific algorithm and encoding presented here, hence the implementation of different satisfiability checking approaches is possible and would provide a nice portfolio-based solver able to cope with even more application scenarios.

Author Contributions All authors contributed equally to the realisation of this manuscript.

Funding Open access funding provided by Libera Università di Bolzano within the CRUI-CARE Agreement. Nicola Gigante acknowledges the support of the TOTA project (Temporal Ontologies and Tableau Algorithms) funded by the Free University of Bozen-Bolzano. Luca Geatti, Nicola Gigante, and Angelo Montanari would also like to acknowledge the support from the GNCS project: "Strategic reasoning and automatic synthesis of multi-agent systems". Gabriele Venturato acknowledges the partial support of the KU Leuven Research Fund (C14/18/062).

[12] Precisely, 468 formulas out of 6113.

Declarations

Competing Interests The authors declare no competing interests affecting this manuscript.

References

1. Abate, P., Goré, R., Widmann, F.: An on-the-fly tableau-based decision procedure for PDL-satisfiability. Electron. Notes Theor. Comput. Sci. **231**, 191–209 (2009). https://doi.org/10.1016/j.entcs.2009.02.036
2. Bacchus, F., Kabanza, F.: Planning for temporally extended goals. Ann. Math. Artif. Intell. **22**(1–2), 5–27 (1998)
3. Barbosa, H., Barrett, C.W., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.: cvc5: a versatile and industrial-strength SMT solver. In: Fisman, D., Rosu, G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems-28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I, *Lecture Notes in Computer Science*, vol. 13243, pp. 415–442. Springer (2022). https://doi.org/10.1007/978-3-030-99524-9_24
4. Bertello, M., Gigante, N., Montanari, A., Reynolds, M.: Leviathan: a new LTL satisfiability checking tool based on a one-pass tree-shaped tableau. In: Proceedings of the 25th International Joint Conference on Artificial Intelligence, pp. 950–956. IJCAI/AAAI Press (2016)
5. Beth, E.W.: Semantic entailment and formal derivability. Sapientia **14**(54), 311 (1959)
6. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Adv. Comput. **58**, 117–148 (2003). https://doi.org/10.1016/S0065-2458(03)58003-2
7. Biere, A., Heljanko, K., Junttila, T.A., Latvala, T., Schuppan, V.: Linear encodings of bounded LTL model checking. Log. Methods Comput. Sci. (2006). https://doi.org/10.2168/LMCS-2(5:5)2006
8. Brafman, R.I., De Giacomo, G.: Planning for LTLf/LDLf goals in non-markovian fully observable nondeterministic domains. In: Kraus, S. (ed.) Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, pp. 1602–1608 (2019). https://doi.org/10.24963/ijcai.2019/222
9. Brafman, R.I., De Giacomo, G., Patrizi, F.: LTLf/LDLf non-markovian rewards. In: S.A. McIlraith, K.Q. Weinberger (eds.) Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, pp. 1771–1778. AAAI Press (2018)
10. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuXmv symbolic model checker. In: Computer Aided Verification, pp. 334–342. Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_22
11. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuXmv symbolic model checker. In: Proceedings of the 26th International Conference on Computer Aided Verification, pp. 334–342. Springer (2014)
12. Cimatti, A., Roveri, M., Sheridan, D.: Bounded verification of past LTL. In: Proceedings of the 5th International Conference on Formal Methods in Computer-Aided Design, pp. 245–259. Springer (2004)
13. Cimatti, A., Roveri, M., Sheridan, D.: Bounded Verification of Past LTL. In: Formal Methods in Computer-Aided Design, LNCS, pp. 245–259. Springer (2004). https://doi.org/10.1007/978-3-540-30494-4_18
14. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The MathSAT5 SMT solver. In: Piterman, N., Smolka, S.A. (eds.) Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, *Lecture Notes in Computer Science*, vol. 7795, pp. 93–107. Springer (2013). https://doi.org/10.1007/978-3-642-36742-7_7
15. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (2001)
16. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: Rossi, F. (ed.) Proceedings of the 23rd International Joint Conference on Artificial Intelligence, pp. 854–860. IJCAI/AAAI (2013)

17. De Giacomo, G., De Masellis, R., Grasso, M., Maggi, F.M., Montali, M.: Monitoring business metaconstraints based on LTL and LDL for finite traces. In: Sadiq, S.W., Soffer, P., Völzer, H. (eds.) Proceedings of the 12th International Conference on Business Process Management, *Lecture Notes in Computer Science*, vol. 8659, pp. 1–17. Springer (2014). https://doi.org/10.1007/978-3-319-10172-9_1
18. De Giacomo, G., Vardi, M.Y.: Synthesis for LTL and LDL on finite traces. In: Yang, Q., Wooldridge, M.J. (eds.) Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, pp. 1558–1564. AAAI Press (2015)
19. De Giacomo, G., Iocchi, L., Favorito, M., Patrizi, F.: Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications. In: Benton, J., Lipovetzky, N., Onaindia, E., Smith, D.E., Srivastava, S. (eds.) Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, pp. 128–136. AAAI Press (2019)
20. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, *Lecture Notes in Computer Science*, vol. 4963, pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24
21. Eén, N., Sörensson, N.: An extensible sat-solver. In: Selected Revised Papers of the 6th International Conference on Theory and Applications of Satisfiability Testing, pp. 502–518 (2003). https://doi.org/10.1007/978-3-540-24605-3_37
22. Eén, N., Sörensson, N.: An extensible sat-solver. In: International conference on theory and applications of satisfiability testing, pp. 502–518. Springer (2003)
23. Fionda, V., Greco, G.: LTL on finite and process traces: complexity results and a practical reasoner. J. Artif. Intell. Res. **63**, 557–623 (2018). https://doi.org/10.1613/jair.1.11256
24. Fisher, M.: A resolution method for temporal logic. In: Mylopoulos, J., Reiter, R. (eds.) Proceedings of the 12th International Joint Conference on Artificial Intelligence, pp. 99–104. Morgan Kaufmann (1991)
25. Fisher, M.: A normal form for temporal logics and its applications in theorem-proving and execution. J. Logic Comput. **7**(4), 429–456 (1997). https://doi.org/10.1093/logcom/7.4.429
26. Geatti, L., Gigante, N., Montanari, A., Reynolds, M.: One-pass and tree-shaped tableau systems for TPTL and TPTLb+Past. In: Orlandini, A., Zimmermann, M. (eds.) Proceedings 9th International Symposium on Games, Automata, Logics, and Formal Verification, *EPTCS*, vol. 277, pp. 176–190 (2018). https://doi.org/10.4204/EPTCS.277.13
27. Geatti, L., Gigante, N., Montanari, A.: A SAT-Based encoding of the one-pass and tree-shaped tableau system for LTL. In: Cerrito, S., Popescu, A. (eds.) Proceedings of the 28th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, *Lecture Notes in Computer Science*, vol. 11714, pp. 3–20. Springer (2019). https://doi.org/10.1007/978-3-030-29026-9_1
28. Geatti, L., Gigante, N., Montanari, A., Reynolds, M.: One-pass and tree-shaped tableau systems for TPTL and TPTL_b Past. Inf. Comput. **278**, 104599 (2021). https://doi.org/10.1016/j.ic.2020.104599
29. Geatti, L., Gigante, N., Montanari, A., Venturato, G.: Past matters: Supporting LTL+Past in the BLACK satisfiability checker. In: Proceedings of the 28th International Symposium on Temporal Representation and Reasoning (2021)
30. Geatti, L., Gianola, A., Gigante, N.: Linear temporal logic modulo theories over finite traces. In: L.D. Raedt (ed.) Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23–29 July 2022, pp. 2641–2647. ijcai.org (2022). https://doi.org/10.24963/ijcai.2022/366
31. Gigante, N., Montanari, A., Reynolds, M.: A one-pass tree-shaped tableau for LTL+Past. In: Proc. of 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, *EPiC Series in Computing*, vol. 46, pp. 456–473 (2017)
32. Heljanko, K., Junttila, T., Latvala, T.: Incremental and complete bounded model checking for full PLTL. In: Proceedings of the 17th International Conference on Computer Aided Verification, pp. 98–111. Springer (2005)
33. Hustadt, U., Konev, B.: TRP++2.0: A temporal resolution prover. In: Proceedings of the 19th International Conference on Automated Deduction, *LNCS*, vol. 2741, pp. 274–278. Springer (2003). https://doi.org/10.1007/978-3-540-45085-6_21
34. Hustadt, U., Nalon, C., Dixon, C.: Evaluating pre-processing techniques for the separated normal form for temporal logics. In: B. Konev, J. Urban, P. Rümmer (eds.) Proceedings of the 6th Workshop on Practical Aspects of Automated Reasoning co-located with Federated Logic Conference 2018 (FLoC 2018), Oxford, UK, July 19th, 2018, *CEUR Workshop Proceedings*, vol. 2162, pp. 34–48. CEUR-WS.org (2018)
35. Kesten, Y., Manna, Z., McGuire, H., Pnueli, A.: A decision algorithm for full propositional temporal logic. In: Proc. of the 5th International Conference on Computer Aided Verification, *LNCS*, vol. 697, pp. 97–109. Springer (1993). https://doi.org/10.1007/3-540-56922-7_9

36. Li, J., Yao, Y., Pu, G., Zhang, L., He, J.: Aalta: an LTL satisfiability checker over infinite/finite traces. In: Cheung, S., Orso, A., Storey, M.D. (eds.) Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 731–734. ACM (2014). https://doi.org/10.1145/2635868.2661669
37. Li, J., Zhu, S., Pu, G., Zhang, L., Vardi, M.Y.: Sat-based explicit LTL reasoning and its application to satisfiability checking. Formal Methods Syst. Des. **54**(2), 164–190 (2019). https://doi.org/10.1007/s10703-018-00326-5
38. Li, J., Pu, G., Zhang, Y., Vardi, M.Y., Rozier, K.Y.: Sat-based explicit LTLF satisfiability checking. Artif. Intell. **289**, 103369 (2020). https://doi.org/10.1016/j.artint.2020.103369
39. Lichtenstein, O., Pnueli, A.: Propositional temporal logics: decidability and completeness. Logic J. IGPL **8**(1), 55–85 (2000). https://doi.org/10.1093/jigpal/8.1.55
40. Lichtenstein, O., Pnueli, A., Zuck, L.D.: The Glory of the Past. In: Procedings of the Logics of Programs Conference, *LNCS*, vol. 193, pp. 196–218. Springer (1985). https://doi.org/10.1007/3-540-15648-8_16
41. Markey, N.: Temporal logic with past is exponentially more succinct. Bull. EATCS **79**, 122–128 (2003)
42. McCabe-Dansted, J.C., Reynolds, M.: A parallel linear temporal logic tableau. In: Bouyer, P., Orlandini, A., Pietro, P.S. (eds.) Proceedings of the 8th International Symposium on Games, Automata, Logics and Formal Verification, *EPTCS*, vol. 256, pp. 166–179 (2017)
43. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient sat solver. In: Proceedings of the 38th annual Design Automation Conference, pp. 530–535 (2001)
44. Pnueli, A.: The Temporal Logic of Programs. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science, pp. 46–57. IEEE Computer Society (1977). https://doi.org/10.1109/SFCS.1977.32
45. Reynolds, M.: A New Rule for LTL Tableaux. In: Proceedings of the 7th International Symposium on Games, Automata, Logics and Formal Verification, *EPTCS*, vol. 26, pp. 287–301 (2016). https://doi.org/10.4204/EPTCS.226.20
46. Schuppan, V., Darmawan, L.: Evaluating LTL satisfiability solvers. In: Proceedings of the 9th International Symposium on Automated Technology for Verification and Analysis, pp. 397–413 (2011)
47. Schwendimann, S.: A new one-pass tableau calculus for PLTL. In: Proceedings of the 7th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, *LNCS*, vol. 1397, pp. 277–292. Springer (1998). https://doi.org/10.1007/3-540-69778-0_28
48. Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logics. J. ACM **32**(3), 733–749 (1985). https://doi.org/10.1145/3828.3837
49. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: O. Kullmann (ed.) Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing, *Lecture Notes in Computer Science*, vol. 5584, pp. 244–257. Springer (2009). https://doi.org/10.1007/978-3-642-02777-2_24
50. Stump, A., Sutcliffe, G., Tinelli, C.: Starexec: A cross-community infrastructure for logic solving. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) Automated Reasoning—7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19–22, 2014. Proceedings, *Lecture Notes in Computer Science*, vol. 8562, pp. 367–373. Springer (2014). https://doi.org/10.1007/978-3-319-08587-6_28
51. Suda, M., Weidenbach, C.: A PLTL-prover based on labelled superposition with partial model guidance. In: Proceedings of the 6th International Joint Conference on Automated Reasoning, *LNCS*, vol. 7364, pp. 537–543. Springer (2012). https://doi.org/10.1007/978-3-642-31365-3_42
52. Tahrat, S., Braun, G., Artale, A., Ozaki, A.: Abstracting temporal aboxes in TDL-Lite. In: Proceedings of the 34th International Workshop on Description Logics (2021)
53. Tauriainen, H., Heljanko, K.: Testing LTL formula translation into Büchi automata. Int. J. Softw. Tools Technol. Transf. **4**(1), 57–70 (2002). https://doi.org/10.1007/s100090200070
54. Wolper, P.: Temporal logic can be more expressive. Inf. Control **56**(1/2), 72–99 (1983). https://doi.org/10.1016/S0019-9958(83)80051-5
55. Wolper, P.: The tableau method for temporal logic: an overview. Logique et Analyse **28**, 119–136 (1985)