EPiC Series in Computing

Volume 46, 2017, Pages 456–473

LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning

A One-Pass Tree-Shaped Tableau for LTL+Past

Nicola Gigante[1], Angelo Montanari[1], and Mark Reynolds[2]

[1] University of Udine, Italy
gigante.nicola@spes.uniud.it
angelo.montanari@uniud.it
[2] University of Western Australia,
Perth, Australia
mark.reynolds@uwa.edu.au

Abstract

Linear Temporal Logic (LTL) is a de-facto standard formalism for expressing properties of systems and temporal constraints in formal verification, artificial intelligence, and other areas of computer science. The problem of LTL satisfiability is thus prominently important to check the consistency of these temporal specifications. Although adding *past* operators to LTL does not increase its expressive power, recently the interest for explicitly handling the past in temporal logics has increased because of the clarity and succinctness that those operators provide. In this work, a recently proposed one-pass tree-shaped tableau system for LTL is extended to support past operators. The modularity of the required changes provides evidence for the claimed ease of extensibility of this tableau system.

1 Introduction

In this paper, we propose a new one-pass and tree-shaped tableau system for LTL extended with past modalities, that generalizes the one developed and implemented in [3, 23].

The problem of checking the consistency of specifications, or parts of them, is receiving renewed interest as computing power increases and clever algorithms are brought to bear. Since specifications are commonly expressed as formulas of a suitable temporal logic, satisfiability checkers can be used for their sanity check. Linear Temporal Logic (LTL) is commonly used as the specification language for concurrent and reactive systems, as it features a good balance between expressiveness and computational complexity (the satisfiability problem for LTL is PSPACE-complete [28], while, e.g., that for CTL is EXPTIME-complete [7]). Different tools for LTL satisfiability checking, based on a variety of techniques, have been proposed over the years (see [11,24,26]), but there is no one overall winner. Common techniques include automata-based approaches [30], resolution methods [4, 8, 18, 29, 31], and tableau systems [3, 12, 14, 23, 27, 32]. Parallel implementations of some of the above solutions are available as well, *e.g.* [25].

In this paper, we focus on the extension of LTL with past operators (LTL+P for short). It is well known that if we restrict the temporal domain to executions of systems that have a definite starting point in time, then past modalities do not add any expressive power, that is, by exploiting the expressiveness results given in [9, 10], a formula with past modalities has

an initially equivalent formula without them. Nevertheless, there are various reasons for their addition. First, as noted in [17], many relevant properties are easier to express using past modalities, allowing specifications to match more closely the way they are expressed in natural language. Some authors have also highlighted some theoretical motivations for the introduction of past-time operators [21, 22]. For instance, if we consider succinctness, past-time modalities do add expressive power, as LTL+P is (at least) exponentially more succinct than LTL [20].

How can we check an LTL+P formula for satisfiability? In view of [9,10], there is, in principle, the possibility of reducing the satisfiability question for LTL+P to that of just LTL. However, as shown in [13], the translation given in [10] may involve a non-elementary blow up in the size of the formula. A more efficient algorithm is provided in [20]. It consists of three steps: it first turns the LTL+P formula into a Büchi automaton, then it transforms such an automaton into a deterministic Muller automaton, which can be assumed to be counter free, and, finally, it translates the Muller automaton into an LTL formula. Since each step may involve an exponential blowup, the size of the resulting LTL formula is at most triply exponential in the size of the initial LTL+P formula. As we will show in a subsequent section, there is a simpler satisfiability-preserving mapping from LTL+P into LTL, which is enough to decide satisfiability of LTL+P formulas. Unfortunately, we show that using this mapping produces formulae that force tableaux to make decisions involving many extra disjunctions at each temporal step.

There are not many (practical) satisfiability checkers that directly deal with LTL+P. One of them is the symbolic model checker NuSMV [5], that does handle the past; another one is the tableau system developed by Lichtenstein *et al.* [17]. Tableau systems have some important advantages over other satisfiability checkers. On the one hand, they allow one to have a clear understanding of the models (behaviors) that satisfy the formula (specification); on the other hand, in many cases, they turn out to be the most efficient solution for some significant classes of formulas. This is the case, for instance, with tableau systems for LTL [11]. Unfortunately, as far as we know, the tableau system for LTL+P given in [17] lacks an implementation. Moreover, it is a two-pass tableau system, which is not as fast as one-pass ones [3, 11].

In this paper, we show that the one-pass and tree-shaped tableau system for LTL, developed in [3, 23], can be extended to past modalities retaining its simplicity. Its ease of understanding was a driving force in the design of the fast implementation studied in [3]. In contrast to other tableaux for LTL, this is a pure rule-based tree search, where each branch can be explored independently from any other. Our work shows that the past can be handled in this framework without destroying these features.

The paper contribution is twofold. In the first place, a new one-pass, tree-shaped tableau system for LTL+P is shown, which extends the one for LTL recently introduced in [23]. The addition was possible without changing the fundamental structure of the underlying tableau system, thus providing some evidence for the claim made in [3] that the clean rule-based tree search structure of this tableau system for LTL can be easily extended to variants and generalizations of the language. In the second place, the new tableau system is provided with complete correctness proofs: (i) the soundness proof, never appeared in a formal publication for the future-only version, and (ii) the completeness proof, adapted to the new rules for the past operators but also reworked to highlight the important backing concept of *trace* of a model, that eases the understanding of the underlying structure.

The rest of the paper is organized as follows. In Section 2, we introduce syntax and semantics of LTL+P. In Section 3, we describe the tableau system for LTL+P and we show its termination. Then, in Section 4, we prove its soundness and completeness. In Section 5, we highlight its distinctive features. Finally, in Section 6, we provide a short assessment of the work done and we outline future research directions.

2 Linear Temporal Logic with Past

Linear Temporal Logic (LTL) is a propositional temporal logic interpreted over infinite, discrete linear orders. Syntactically, LTL can be viewed as an extension of propositional logic with the *tomorrow* ($\mathsf{X}\,\phi$, at the *next* state ϕ holds) and *until* ($\phi_1\,\mathcal{U}\,\phi_2$, ϕ_2 will eventually hold and ϕ_1 will hold *until* then) temporal operators. LTL+P is obtained from LTL by adding the *yesterday* ($\mathsf{Y}\,\phi$, at the *previous* state ϕ holds) and *since* ($\phi_1\,\mathcal{S}\,\phi_2$, there was a past state where ϕ_2 held, and ϕ_1 has held *since* then) *past* temporal operators. Formally, given a set $\mathcal{AP}$ of proposition letters, LTL+P formulae are generated by the following syntax:

$$\begin{aligned}
\phi \coloneqq\ & p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 && \text{propositional connectives}\\
& \mid \mathsf{X}\,\phi_1 \mid \phi_1\,\mathcal{U}\,\phi_2 \mid \phi_1\,\mathcal{R}\,\phi_2 \mid \mathsf{F}\,\phi_1 \mid \mathsf{G}\,\phi_1 && \textit{future} \text{ temporal operators}\\
& \mid \mathsf{Y}\,\phi_1 \mid \phi_1\,\mathcal{S}\,\phi_2 \mid \phi_1\,\mathcal{T}\,\phi_2 \mid \mathsf{P}\,\phi_1 \mid \mathsf{H}\,\phi_1 && \textit{past} \text{ temporal operators}
\end{aligned}$$

where $p \in \mathcal{AP}$ and ϕ_1 and ϕ_2 are LTL+P formulae. Note that most of the temporal operators of the language can be defined in terms of a small number of basic ones. In particular, the *release* ($\phi_1 \mathcal{R} \phi_2 \equiv \neg(\neg\phi_1 \mathcal{U} \neg\phi_2)$), *eventually* ($\mathsf{F}\,\phi \equiv \top\,\mathcal{U}\,\phi$), and *always* ($\mathsf{G}\,\phi \equiv \neg\,\mathsf{F}\,\neg\phi$) future operators can all be defined in terms of the *until* operator, while the *triggered* ($\phi_1\,\mathcal{T}\,\phi_2 \equiv \neg(\neg\phi_1\,\mathcal{S}\,\neg\phi_2)$), *once* ($\mathsf{P}\,\phi \equiv \top\,\mathcal{S}\,\phi$), and *historically* ($\mathsf{H}\,\phi \equiv \neg\,\mathsf{P}\,\neg\phi$) past operators can all be defined in terms of the *since* operator. However, in our setting, it is useful to consider them part of the language.

A *model* for an LTL+P formula ϕ is a pair (σ, λ), where $\sigma = \langle \sigma_0, \sigma_1, \ldots \rangle$ is an ω-sequence of *states* and $\lambda : \{\sigma_0, \sigma_1, \ldots\} \to 2^{\mathcal{AP}}$ is a labeling function mapping each state σ_i to the set of proposition letters that hold in it. We will simply write σ for (σ, λ) when the labeling function is clear from the context. Given a model (σ, λ), a position $i \geq 0$, and an LTL+P formula ϕ, we inductively define the *satisfaction* of ϕ by σ at position i, written as $\sigma, i \models \phi$, as follows:

- $\sigma, i \models p$ iff $p \in \lambda(\sigma_i)$;
- $\sigma, i \models \neg\phi$ iff $\sigma, i \not\models \phi$;
- $\sigma, i \models \phi_1 \vee \phi_2$ iff $\sigma, i \models \phi_1$ or $\sigma, i \models \phi_2$;
- $\sigma, i \models \phi_1 \wedge \phi_2$ iff $\sigma, i \models \phi_1$ and $\sigma, i \models \phi_2$;
- $\sigma, i \models \mathsf{X}\,\phi$ iff $\sigma, i+1 \models \phi$;
- $\sigma, i \models \mathsf{Y}\,\phi$ iff $i > 0$ and $\sigma, i-1 \models \phi$;
- $\sigma, i \models \phi_1\,\mathcal{U}\,\phi_2$ iff there exists $j \geq i$ such that $\sigma, j \models \phi_2$, and $\sigma, k \models \phi_1$ for all k, with $i \leq k < j$;
- $\sigma, i \models \phi_1\,\mathcal{S}\,\phi_2$ iff there exists $j \leq i$ such that $\sigma, j \models \phi_2$, and $\sigma, k \models \phi_1$ for all k, with $j < k \leq i$.
- $\sigma, i \models \phi_1\,\mathcal{R}\,\phi_2$ iff either $\sigma, j \models \phi_2$ for all $j \geq i$, or there exists $k \geq i$ such that $\sigma, k \models \phi_1$ and $\sigma, j \models \phi_2$ for all $i \leq j \leq k$
- $\sigma, i \models \phi_1\,\mathcal{T}\,\phi_2$ iff either $\sigma, j \models \phi_2$ for all $0 \leq j \leq i$, or there exists $k \leq i$ such that $\sigma, k \models \phi_1$ and $\sigma, j \models \phi_2$ for all $i \geq j \geq k$

We say that σ *satisfies* ϕ, $\sigma \models \phi$, if it satisfies the formula at the first state, *i.e.*, if $\sigma, 0 \models \phi$. This is often called *initial* satisfiability, as opposed to the notion of (*global*) satisfiability, where a formula is said to be satisfied by a model if it holds at some position of the model. Without loss of generality, we restrict our attention to initial satisfiability. It can be indeed easily shown that ϕ is (globally) satisfiable if $\mathsf{F}\,\phi$ is initially satisfiable. We will also make use of the notion of *initial* equivalence between formulae, *i.e.*, we say that two formulae ϕ and ψ are *equivalent* ($\phi \equiv \psi$) if and only if they are satisfied by the same set of models at the initial state.

3 A New Tableau System for LTL + Past

We now show how to extend the one-pass and tree-shaped tableau system for LTL developed in [23] in order to handle past operators. From now on, we will assume all formulae to be in *Negated Normal Form*, *i.e.*, with all the negations pushed down to literals. This is the reason why it was convenient in Section 2 to consider all possible temporal operators as part of the core language (*e.g.*, the NNF of an *until* formula needs the *release* operator).

Definition 1 (Closure of a formula)**.** Given an LTL+P formula ϕ, the (positive) *closure* of ϕ is the set of formulae $\mathcal{C}(\phi)$ built as follows:

- $\phi \in \mathcal{C}(\phi)$;
- for every subformula ϕ' of ϕ, $\phi' \in \mathcal{C}(\phi)$;
- for every $p \in \mathcal{AP}$, $p \in \mathcal{C}(\phi)$ if and only if $\neg p \in \mathcal{C}(\phi)$;
- for every $\phi\,\mathcal{U}\,\psi \in \mathcal{C}(\phi)$ (resp., $\phi\,\mathcal{R}\,\psi \in \mathcal{C}(\phi)$), $\mathsf{X}(\phi\,\mathcal{U}\,\psi) \in \mathcal{C}(\phi)$ (resp., $\mathsf{X}(\phi\,\mathcal{R}\,\psi) \in \mathcal{C}(\phi)$);
- for every $\mathsf{F}\,\phi \in \mathcal{C}(\phi)$ (resp., $\mathsf{G}\,\phi \in \mathcal{C}(\phi)$), $\mathsf{X}\,\mathsf{F}\,\phi \in \mathcal{C}(\phi)$ (resp., $\mathsf{X}\,\mathsf{G}\,\phi \in \mathcal{C}(\phi)$);
- for every $\phi\,\mathcal{S}\,\psi \in \mathcal{C}(\phi)$ (resp., $\phi\,\mathcal{T}\,\psi \in \mathcal{C}(\phi)$), $\mathsf{Y}(\phi\,\mathcal{S}\,\psi) \in \mathcal{C}(\phi)$ (resp., $\mathsf{Y}(\phi\,\mathcal{T}\,\psi) \in \mathcal{C}(\phi)$);
- for every $\mathsf{P}\,\phi \in \mathcal{C}(\phi)$ (resp., $\mathsf{H}\,\phi \in \mathcal{C}(\phi)$), $\mathsf{Y}\,\mathsf{P}\,\phi \in \mathcal{C}(\phi)$ (resp., $\mathsf{Y}\,\mathsf{H}\,\phi \in \mathcal{C}(\phi)$).

A tableau for ϕ is a tree T where each node u is labeled by a subset $\Gamma(x)$ of the closure $C(\phi)$ and the label of the root node u_0 contains only ϕ, *i.e.*, $\Gamma(u_0) = \{\phi\}$. The tableau is built recursively from the root, at each step applying one from a set of *rules* to a leaf of the tree. Each rule can add one or two children to the current node, advancing the construction of the tree, or close the current branch by *accepting* (✓) or *rejecting* (✗) the current node. Once a *complete* tableau has been obtained, *i.e.*, when all the leaves are either ticked or crossed (at the end of the section, we will prove that this is always the case), then the formula is recognized as satisfiable if there is at least one accepted branch. In the following, given two nodes u and v, we will write $u \geq v$ to mean that u is an ancestor of v, and $u > v$ to mean that u is a *proper* ancestor of v, *i.e.*, $u \geq v$ and $u \neq v$.

The construction of a branch of the tree can be seen as the search for a model of the formula in a state-by-state way. At each step, *expansion rules* are applied first, building a possible assignment of proposition letters for the current state, which is then verified by the two *soundness rules*. Next, the *termination rules* are checked to possibly detect the truth of the termination conditions for the construction. At the end, information about the current state is used to determine the next one, and the construction proceeds by executing a temporal *step*.

CONJUNCTION	$\alpha \wedge \beta$	$\rightarrow$	$\{\alpha, \beta\}$
DISJUNCTION	$\alpha \vee \beta$	$\rightarrow$	$\{\alpha\} \mid \{\beta\}$
UNTIL	$\alpha\,\mathcal{U}\,\beta$	$\rightarrow$	$\{\beta\} \mid \{\alpha, \mathsf{X}(\alpha\,\mathcal{U}\,\beta)\}$
RELEASE	$\alpha\,\mathcal{R}\,\beta$	$\rightarrow$	$\{\alpha, \beta\} \mid \{\beta, \mathsf{X}(\alpha\,\mathcal{R}\,\beta)\}$
EVENTUALLY	$\mathsf{F}\,\alpha$	$\rightarrow$	$\{\alpha\} \mid \{\mathsf{X}\,\mathsf{F}\,\alpha\}$
ALWAYS	$\mathsf{G}\,\alpha$	$\rightarrow$	$\{\alpha, \mathsf{X}\,\mathsf{G}\,\alpha\}$
SINCE	$\alpha\,\mathcal{S}\,\beta$	$\rightarrow$	$\{\beta\} \mid \{\alpha, \mathsf{Y}(\alpha\,\mathcal{S}\,\beta)\}$
TRIGGERED	$\alpha\,\mathcal{T}\,\beta$	$\rightarrow$	$\{\alpha, \beta\} \mid \{\beta, \mathsf{Y}(\alpha\,\mathcal{T}\,\beta)\}$
ONCE	$\mathsf{P}\,\alpha$	$\rightarrow$	$\{\alpha\} \mid \{\mathsf{Y}\,\mathsf{P}\,\alpha\}$
HISTORICALLY	$\mathsf{H}\,\alpha$	$\rightarrow$	$\{\alpha, \mathsf{Y}\,\mathsf{H}\,\alpha\}$

Table 1: Expansion rules

The *expansion rules* look for a specific formula into the label, creating one or two children whose labels are obtained by replacing the target formula with some others. Table 1 shows the expansion rules with the following notation: a rule of the form $\phi \rightarrow \Delta$, where Δ is a set of formulae, means that whenever the rule is applied to a node u with $\phi \in \Gamma(u)$, a child u' of u is created with label $\Gamma(u') = \Gamma(x) \setminus \{\phi\} \cup \Delta$. A rule of the form $\phi \rightarrow \Delta_1 \mid \Delta_2$ creates two different children in the same way. Note that the order in which formulae are considered for the application of expansion rules does not matter.

After a finite number of applications of expansion rules, the construction will eventually reach a node whose label contains only *elementary* formulae, *i.e.*, only formulae of the forms p or $\neg p$, for some $p \in \Sigma$, $\mathsf{X}\,\alpha$, and $\mathsf{Y}\,\alpha$, for some $\alpha \in \mathcal{C}(\phi)$ (while p and $\neg p$ cannot both belong to the same label, it can be the case that both $\mathsf{X}\,\alpha$ and $\mathsf{X}\neg\alpha$ belong to it). Such a label is called a *poised label* and the node is called a *poised node*. Intuitively, a poised node represents a guess for the truth of elementary formulae at the current state. Once a poised node has been obtained, the search can proceed to the next state, by applying the STEP rule:

STEP Let v be a poised node. A child v' is added to v with label $\Gamma(v') = \{\alpha \mid \mathsf{X}\,\alpha \in \Gamma(v)\}$.

Since the STEP rule represents an advancement in time, (some of the) poised nodes will be used later to extract a model of ϕ from a successful tableau branch. The STEP rule only adds a single child to a poised node, but other rules may need to insert additional children to any given poised node to handle subsequent unfulfilled past requests, as it will be explained below. Hence, it will be useful to recognize when, in a branch of the tree, a poised node is traversed going through the child added directly by the STEP rule, or through some other of its children.

Definition 2 (Step nodes)**.** Consider a branch $\overline{x} = \langle x_0, \ldots, x_i, \ldots, x_n \rangle$ of a complete tableau T. If x_i is a poised node, and either $i = n$ or x_{i+1} is the child of x_i added directly by a STEP rule, then x_i is said to be a *step node* for the branch $\overline{x}$.

In any case, the label of a poised node has to be analyzed to check the *soundness* of the current state with regards to propositional contradictions (CONTRADICTION rule) and the transition from the previous state (YESTERDAY rule). If the outcome is positive, then the system advances the search to the next temporal state; otherwise, the construction has to be stopped.

Formally, given a branch $\overline{x} = \langle x_0, \ldots, x_i, \ldots, x_n \rangle$ of a complete tableau T and a step node x_i for it, let $x_j \geq x_i$ be either the root node, if x_i is the first step node in $\overline{x}$, or the child of the closest step node among the proper ancestors of x_i. We define $\Delta(x_i)$ as the set $\Gamma(x_j) \cup \ldots \cup \Gamma(x_i)$. The *soundness rules* are the following ones:

CONTRADICTION Let v be a poised node. If $\{p, \neg p\} \subseteq \Gamma(v)$, for some $p \in \mathcal{AP}$, the node is crossed and the branch rejected.

YESTERDAY Let v be a poised node such that $\mathsf{Y}\,\alpha \in \Gamma(v)$, for some $\alpha \in \mathcal{C}(\phi)$. If v is the first *step node* of its branch, then v is crossed and the branch rejected. Otherwise, let Ω be the set of all the formulae α such that $\mathsf{Y}\,\alpha \in \Gamma(v)$. The poised node v is crossed if there is some $\alpha \in \Omega$ such that $\alpha \notin \Delta(x_i)$. In such a case, a new child node u' is added to the closest poised node u among the proper ancestors of v, with label $\Gamma(u') = \Gamma(u) \cup \Omega$, unless a child u'' of u already existed such that $\Gamma(u') \subseteq \Gamma(u'')$.

The YESTERDAY rule ensures that all the past requests made by the elementary formulae that were chosen for the current state are already satisfied by the current branch. If this is not the case, the construction restarts from the previous state, assuming the truth of the requested formulae. Besides the expansion rules for the other past operators of Table 1, this is the only significant change needed to the original structure of the tableau to support past operators.

The rules introduced so far allow us to build a tentative model for the formula step-by-step, but we introduced only rules that can reject wrong branches, so we still need to specify how to recognize good branches corresponding to actual models. The first obvious case in which we have to stop the construction is when there is nothing left to do:

EMPTY Let u be a poised node. If $\Gamma(u) = \emptyset$, then the node is ticked and the branch accepted.

Since LTL+P models are infinite, only simple formulae will have models satisfying the EMPTY rule. Others, *e.g.*, $\mathsf{G}\,\mathsf{F}\,p$, at any state require something to happen in the future. Thus, some criteria are needed to ensure that the construction of every branch will eventually terminate.

To this end, we introduce a pair of *termination rules* which are checked at each poised node (label). As a matter of fact, the potentially infinite expansion of the branches is caused by the recursive nature of most of the expansion rules. The UNTIL and EVENTUALLY rules are the most critical ones. In the expansion of a formula of the form $\alpha\,\mathcal{U}\,\beta$ or $\mathsf{F}\,\beta$, these rules try to either fulfill the request for β immediately or to postpone its fulfillment to a later step by adding $\mathsf{X}(\alpha\,\mathcal{U}\,\beta)$ or $\mathsf{X}\,\mathsf{F}\,\beta$ to the label. A formula of these kinds is called *X-eventuality*. If an X-eventuality appears in a label, it means that some pending request still needs to be fulfilled in the future, and some criterion has to be used to avoid postponing its fulfillment indefinitely. An X-eventuality of the form $\mathsf{X}(\alpha\,\mathcal{U}\,\beta)$ or $\mathsf{X}\,\mathsf{F}\,\beta$ appearing in $\Gamma(u)$ is said to be *requested* in u; moreover, we say that it is *fulfilled* in v, with $u > v$, if $\beta \in \Gamma(v)$, and that it is *fulfilled* between u and v, with $u > v$, if there exists w, with $u > w \geq v$, such that $\beta \in \Gamma(w)$. The LOOP and PRUNE rules, checked in this order, allow us to respectively handle the case of a branch that is repeating by successfully fulfilling recurrent requests and the case of a branch that is indefinitely postponing the fulfillment of an eventuality that is impossible to fulfill.[1]

LOOP If two step nodes $u > v$ are found such that $\Gamma(v) = \Gamma(u)$, and all the X-eventualities requested in u are fulfilled between u and v, then v is ticked and the branch accepted.

PRUNE If three step nodes $u > v > w$ are found such that $\Gamma(u) = \Gamma(v) = \Gamma(w)$, and among the X-eventualities requested in these nodes, all those fulfilled between v and w were also fulfilled between u and v, then w is crossed and the branch rejected.

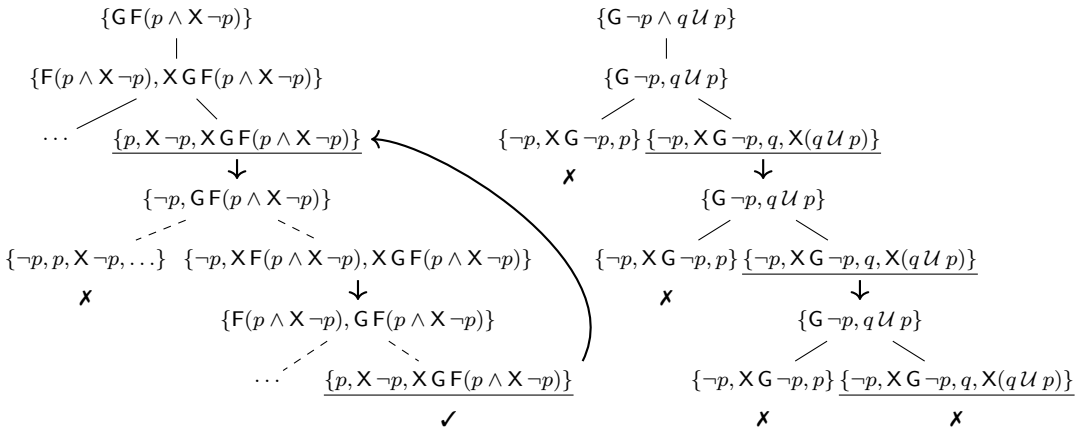
Thus, the LOOP rule is responsible for recognizing when a model for the formula has been found, while the PRUNE rule rejects branches that were doing redundant work without managing to fulfill any new eventuality. The PRUNE rule was the main novelty of this tableau system when introduced in [23]. At this point, if none of these two rules have closed the branch, the current state is ready and the construction can advance in time by applying the STEP rule.

The rules described above are repeatedly applied to the leaves of any non-closed branch unless all branches have been either ticked or crossed. This process is guaranteed to terminate.

Theorem 1 (Termination)**.** *Given a formula ϕ, the construction of a (complete) tableau T for ϕ will always terminate in a finite number of steps.*

Proof. To start with, we observe that the tree has a finite branching degree: each rule adds at most two children to every node, with the exception of the YESTERDAY rule that can add more than two nodes; however, by definition, such a rule will never result into two siblings with the same label, and the number of possible labels is finite. Thus, by König's lemma, for the construction to go on forever the tree should contain at least one infinite branch. This, however, cannot be the case, again because of the finite number of possible labels and of possible X-eventualities: after a finite number of steps, a branch will contain either two occurrences of the same label triggering the LOOP rule, or three occurrences of the same label triggering the PRUNE rule. Note that the LOOP rule may be never triggered, *i.e.*, when the formula is unsatisfiable. However, if the LOOP rule is never triggered, the PRUNE rule will always eventually be because the set of X-eventualities is finite and the number of X-eventualities encountered along a branch from a given node on is obviously non-decreasing. □

[1] It must be noticed that the PRUNE rule may also reject some branches where the fulfillment of some requests is simply delayed. However, this is not a problem because the completeness result of Theorem 3 (see Section 4) guarantees us that we cannot lose in this way all the useful branches.



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

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

Figure 1: Examples of tableaux for two formulae, with the use of the LOOP and PRUNE rules.

As for the complexity of the procedure, it can be seen that the whole decision procedure runs using exponential space, as only a single branch at the time is needed to be kept in memory. The procedure is thus not optimal with regards to the complexity of LTL satisfiability, which is PSPACE-complete. However, this is in line with other tableau-based decision procedures, *e.g.*, the one from Lichtenstein *et al.* [17] and the one from Schwendimann [27].

To get an intuitive understanding of how the tableau works for typical formulae, take a look at Figure 1, where two example tableaux are shown. Here and in the following pictures, dashed edges mean that a part of the subtree is hidden to save space, while bold arrows represent the application of a STEP rule to a poised label.

Figure 1a shows part of the tableau for the liveness formula $\mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\,\neg p)$. This formula requires something, *i.e.*, $p \wedge \mathsf{X}\,\neg p$, to happen infinitely often. As we go down any branch, we can see that the request $\mathsf{X}\,\mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\,\neg p)$ is present at any poised label, propagated by the corresponding expansion rule. Then, any time $\mathsf{F}(p \wedge \mathsf{X}\,\neg p)$ is added to a label, the branch forks to choose between adding $p \wedge \mathsf{X}\,\neg p$ immediately or postponing it. To satisfy the formula, p has to be set true once every two steps, and this is handled by the CONTRADICTION rule, that fires every time a wrong choice is made. Then, the LOOP rule is triggered by the rightmost branch, which is repeating the same label for the second time. The looping arrow does not represent a real edge, since otherwise it would not be a tree, but it is just a way to highlight to which label the loop is jumping to.

Figure 1b shows an example of application of the PRUNE rule in the tableau for the formula $\mathsf{G}\,\neg p \wedge q\,\mathcal{U}\,p$. The formula is unsatisfiable, not because of a (direct) propositional contradiction, but rather because the eventuality requested by $q\,\mathcal{U}\,p$ cannot be realized for the presence of the $\mathsf{G}\,\neg p$ component. The expansion of the *until* operator will then try to realize p at each step, each time resulting into a propositional contradiction. The rightmost branch would then continue postponing the X-eventuality forever, if not for the PRUNE rule which crosses the branch after the third repetition of the same label (with no X-eventuality fulfilled in between).

One may wonder why the PRUNE rule needs to look for the third occurrence of a label before triggering. An enlightening example is given in Figure 2. The formula ϕ shown in the figure

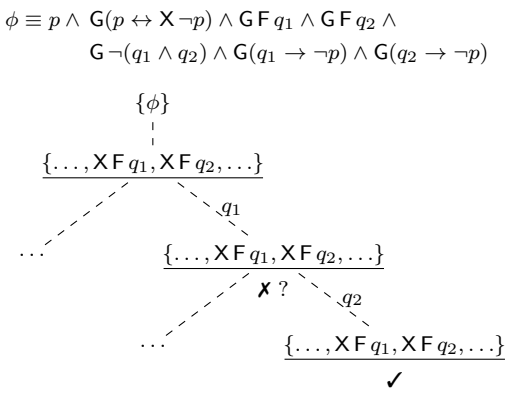


Figure 2: Example of why the PRUNE rule waits for *three* repetitions of the same label.

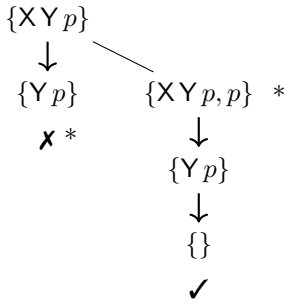


Figure 3: Use of the YESTERDAY rule

requires q_1 and q_2 to appear infinitely often, but never at the same time, thus forcing a kind of (not necessarily strict) alternation between the two. Developing the tableau for ϕ, one can see that the requests $\mathsf{X}\,\mathsf{F}\,q_1$ and $\mathsf{X}\,\mathsf{F}\,q_2$ will be permanently present in the labels, in particular, after realizing one of the two. Thus, crossing the branch after the second occurrence of the label would be wrong, since the repetition of the label alone does not imply that the branch is doing wasteful. Indeed, after the first repetition, the branch can continue making different choices, realizing the other request, and can be closed by the LOOP rule after having realized both.

Finally, Figure 3 shows an example of the application of the YESTERDAY rule to the very simple formula $\mathsf{X}\,\mathsf{Y}\,p$. After the first application of the STEP rule, the poised label $\{\mathsf{Y}\,p\}$ triggers the YESTERDAY rule, which however cannot find p in the label of the previous step node, which in this case is the root. Then, a child of the node is added (marked with $*$ in the picture), adding p to the label of the original node. Proceeding as usual, the following triggering of the YESTERDAY will now find p in the label of its preceding step node, thus allowing the construction of the tree to proceed. A third STEP rule will then produce an empty label, as no occurrence of *tomorrow* is present, hence ticking the branch.

4 Soundness and Completeness

In the following, we prove the soundness and the completeness of the tableau system described in Section 3. The proofs differ from those given in [23] for the future-only version of the tableau in (at least) two respects: (i) various lemmata have been considerably revised to cope with the new rules for past operators, and (ii) in order to increase the readability of the completeness proof, we made explicit mention of the backing concept of *trace* of a model.

4.1 Soundness

The argument to show the soundness of the tableau is similar to the one used in more classic tableaux such as those shown in [17, 19]. However, the notion of *atom* associated with a node in the proposed tableau is different from that of classic tableaux. While most (graph-shaped) classic tableaux keep track, in each node, of each subformula from $\mathcal{C}(\phi)$ that is true in a specific

state, here some formulae which are not relevant to the current state can be missing from the labels, even if they can be true. The following definition of *atom* reflects such a difference.

Definition 3. Let ϕ be an LTL+P formula. An *atom* of ϕ is a subset Δ of $\mathcal{C}(\phi)$ such that, for each *non elementary* formula $\psi \in \Delta$:

- $\psi \equiv \alpha \wedge \beta \in \Delta$ iff $\{\alpha, \beta\} \subseteq \Delta$
- $\psi \equiv \alpha \vee \beta \in \Delta$ iff either $\alpha \in \Delta$ or $\beta \in \Delta$
- $\psi \equiv \alpha \,\mathcal{U}\, \beta \in \Delta$ iff either $\beta \in \Delta$ or $\{\alpha, \mathsf{X}(\alpha \,\mathcal{U}\, \beta)\} \subseteq \Delta$
- $\psi \equiv \alpha \,\mathcal{R}\, \beta \in \Delta$ iff either $\{\alpha, \beta\} \subseteq \Delta$ or $\{\beta, \mathsf{X}(\alpha \,\mathcal{R}\, \beta)\} \subseteq \Delta$
- $\psi \equiv \alpha \,\mathcal{S}\, \beta \in \Delta$ iff either $\beta \in \Delta$ or $\{\alpha, \mathsf{Y}(\alpha \,\mathcal{S}\, \beta)\} \subseteq \Delta$
- $\psi \equiv \alpha \,\mathcal{T}\, \beta \in \Delta$ iff either $\{\alpha, \beta\} \subseteq \Delta$ or $\{\beta, \mathsf{Y}(\alpha \,\mathcal{T}\, \beta)\} \subseteq \Delta$

Note that, as in the rest of the section, for brevity we have left out the parts of Definition 3 corresponding to the F, G, P, and H operators that can be easily derived from the given ones.

Intuitively, our atoms are sets of formulae such that the presence of each non elementary one is *justified*, that is, implied, by the elementary formulae in the set, and each non elementary formula that can be justified by elements in the set is present. This concept has a strong similarity with the concept of *atom* employed in classic graph-shaped tableaux because we are looking at consistent sets of formulae true at a specific state, but it differs from it as the sets may be not maximal. As an example, given the formula $(p \vee q) \wedge p \,\mathcal{U}\, q$, the set $\{q, p \,\mathcal{U}\, q\}$ is a valid (part of an) atom, because $p \,\mathcal{U}\, q$ is implied by q, but neither p nor $\neg p$ needs to be part of it, even if the set of formulae obtained by adding one of them would still be consistent. To summarize, according to the proposed definition, it is not the case that for each formula $\psi \in \mathcal{C}(\phi)$, either ψ or $\neg\psi$ belong to the atom.

The atoms of a formula are *indirectly* considered in the construction of the tableau. *Poised labels* of nodes, if not crossed by the CONTRADICTION rule, are trivially atoms, since they only contain elementary formulae, but other labels might lack justification for some elements. However, the collection of poised and non-poised labels between step nodes forms an atom.

Definition 4. Let $\overline{x} = \langle x_0, \ldots, x_n \rangle$ be a branch of a complete tableau for ϕ and let $\pi = \langle \pi_0, \ldots, \pi_m \rangle$ be the subsequence of step nodes for $\overline{x}$ occurring in it. Given a π_j, let $x > \pi_j$ be either x_0, if $j = 0$, or the successor of π_{j-1} in $\overline{x}$ otherwise. Then, *the* atom $\Delta(\pi_j)$ of π_j is the minimal atom including all formulae in $\Gamma(x')$, for all the $x \geq x' \geq \pi_j$.

Proposition 1. *Let $\overline{x} = \langle x_0, \ldots, x_n \rangle$ be a branch of a complete tableau for ϕ, and let x_j be a* step node *for $\overline{x}$. Then, the atom $\Delta(x_j)$ does not contain propositional contradictions.*

Proof. By structural induction on $\alpha \in \Delta(x_j)$, using the expansion rules from Table 1. □

Now, given a successful branch $\overline{x} = \langle x_0, \ldots, x_n \rangle$ of the tableau, we can use the atoms of the step nodes of $\overline{x}$ to extract a model for the formula, thus proving the soundness of the tableau system. The model will be extracted from the sequence of atoms of the step nodes of the branch, called a *pre-model*. We now formally define pre-models, then prove some basic facts about them, and finally provide the soundness proof.

Let $\pi = \langle \pi_0, \ldots, \pi_m \rangle$ be the subsequence of *step nodes* of $\overline{x}$. If $x_n = \pi_m$ was ticked by the LOOP rule, then there is a step node $\pi_k > \pi_m$ such that $\Gamma(\pi_k) = \Gamma(\pi_m)$ and all the X-eventualities requested in π_k are fulfilled between π_k and π_m. The corresponding model is *periodic*, with period $T = m - k$, and it repeats forever the atoms of the step nodes belonging to the suffix of π between π_{k+1} and π_m. If π_m was instead ticked by the EMPTY rule, we consider $k = m - 2$, so that the period is $T = 2$ and the model repeats forever the states corresponding to π_{m-1} and π_m.

In the case of a branch closed by the EMPTY rule instead of the LOOP one, the model can be closed in any arbitrary way after the last state. The choice made here allows us to treat uniformly the branches closed by any of the two rules, as in the following definition.

Definition 5. Let $\overline{x} = \langle x_0, \ldots, x_n \rangle$ be a successful branch of a complete tableau for the formula ϕ, let $\pi = \langle \pi_0, \ldots, \pi_m \rangle$ be the subsequence of its *step nodes*, and let k and T be defined as above. Then, the *pre-model* of $\overline{x}$ is an *infinite* sequence $\Delta(\overline{x}) = \langle \Delta_0, \Delta_1, \ldots \rangle$ such that for all i, $\Delta_i = \Delta(\pi_{K(i)})$, where $K(i) = i$, if $i \leq m$, and $K(i) = k + ((i-k) \bmod T)$ otherwise.

The next lemma states some relevant properties of pre-models.

Lemma 1. *Let $\overline{x} = \langle x_0, \ldots, x_n \rangle$ be a successful branch of a complete tableau for the formula ϕ and let $\Delta(\overline{x}) = \langle \Delta_0, \Delta_1, \ldots \rangle$ be its pre-model. Then, the following facts hold:*

1. *for all $i \geq 0$, if $\mathsf{X}\alpha \in \Delta_i$ then $\alpha \in \Delta_{i+1}$;*
2. *for all $i \geq 0$, if $\mathsf{Y}\alpha \in \Delta_i$, then $i > 0$ and $\alpha \in \Delta_{i-1}$;*
3. *for all $i \geq 0$, if $\alpha \,\mathcal{U}\, \beta \in \Delta_i$, then there exists $h \geq i$ such that $\beta \in \Delta_h$, and for all j such that $i \leq j < h$, $\alpha \in \Delta_j$;*
4. *for all $i \geq 0$, if $\alpha \,\mathcal{S}\, \beta \in \Delta_i$, then there exists $h \leq i$ such that $\beta \in \Delta_h$, and for all j such that $i \geq j > h$, $\alpha \in \Delta_j$.*
5. *for all $i \geq 0$, if $\alpha \,\mathcal{R}\, \beta \in \Delta_i$, then either $\beta \in \Delta_j$ for all $j \geq i$, or there exists $j \geq i$ such that $\alpha \in \Delta_j$ and $\beta \in \Delta_h$ for all $i \leq h \leq j$.*
6. *for all $i \geq 0$, if $\alpha \,\mathcal{T}\, \beta \in \Delta_i$, then either $\beta \in \Delta_j$ for all $j \leq i$, or there exists $j \leq i$ such that $\alpha \in \Delta_j$ and $\beta \in \Delta_h$ for all $i \geq h \geq j$.*

Proof. By definition of pre-model, $\Delta_i = \Delta(\pi_{K(i)})$, where $\pi = \langle \pi_0, \ldots, \pi_m \rangle$ is the subsequence of step nodes of $\overline{x}$ and $K(i) = k + ((i-k) \bmod T)$. It holds that $\pi_{K(i+1)}$ is either $\pi_{K(i)+1}$ if $i-k$ is not a multiple of T, or π_{k+1} otherwise. This helps us proving the items above as follows.

1. If $\mathsf{X}\alpha \in \Delta_i$, then either Δ_{i+1} is the atom of the next step node and, by the STEP rule, $\alpha \in \Delta_{i+1}$, or $\pi_{K(i)} = \pi_m$ and Δ_{i+1} is $\Delta(\pi_{k+1})$. In this case, π_m was ticked by the LOOP rule, because $\mathsf{X}\alpha$ cannot belong to an empty label. Thus, $\Gamma(\pi_m) = \Gamma(\pi_k)$ and, by the STEP rule, for all the $\mathsf{X}\alpha \in \Gamma(\pi_m)$, we have that $\alpha \in \Delta(\pi_{k+1}) = \Delta_{i+1}$.
2. Suppose $\mathsf{Y}\alpha \in \Delta_i$. By definition of the YESTERDAY rule, the first step node of a branch cannot contain a formula of the form $\mathsf{Y}\alpha$ without being crossed, and thus $i > 0$. Then, similarly to the argument of the previous item, either Δ_{i-1} is the atom of the previous step node in $\overline{x}$, and thus $\alpha \in \Delta_{i-1}$ by the YESTERDAY rule, or $\Delta_i = \Delta(\pi_{k+1})$ and, by the YESTERDAY rule, $\alpha \in \Delta(\pi_k)$ and we know that π_m was ticked by the LOOP rule. Thus α is implied by the elementary formulae in $\Gamma(\pi_k)$, and since $\Gamma(\pi_k) = \Gamma(\pi_m)$ by the LOOP rule, we can conclude that $\alpha \in \Delta(\pi_m) = \Delta_{i-1}$, by definition of atom.
3. If $\alpha \,\mathcal{U}\, \beta \in \Delta_i$, then, by the UNTIL rule, either $\beta \in \Delta_i$ or both $\alpha \in \Delta_i$ and $\mathsf{X}(\alpha \,\mathcal{U}\, \beta) \in \Delta_i$. In the first case, we are done; otherwise, by item 1. above, $\alpha \,\mathcal{U}\, \beta \in \Delta_{i+1}$. By iterating the same argument, we can conclude that either there exists $h \geq i$ such at $\beta \in \Delta_h$ and for all $i \leq j \leq h$, both $\alpha \in \Delta_j$ and $\alpha \,\mathcal{U}\, \beta \in \Delta_j$, or there is no such h, and $\{\alpha, \alpha \,\mathcal{U}\, \beta\} \in \Delta_j$ for all $j \geq i$. To show that the latter case is not possible, for all $j \geq i$, by the definition of atom, $\mathsf{X}(\alpha \,\mathcal{U}\, \beta) \in \Delta_j$, including $\Delta(\pi_m)$. Then, since π_m is ticked, but it has not an empty label, it has been ticked by the LOOP rule. It immediately follows that $\Gamma(\pi_m) = \Gamma(\pi_k)$ and all the X-eventualities in $\Gamma(\pi_m)$ should have been fulfilled at least once between π_k and π_m. Hence, β should belong to at least one Δ_j (contradiction).

For the remaining items, the argument is similar to that for the third one, and thus their proofs are omitted. □

By exploiting Lemma 1, we can now prove the soundness of the proposed tableau method.

Theorem 2 (Soundness)**.** *Let ϕ be a* LTL+P *formula. If a complete tableau for ϕ contains at least a ticked branch, then ϕ has a model.*

Proof. Let $\overline{x} = \langle x_0, \ldots, x_n \rangle$ be a successful branch of a complete tableau for ϕ and let $\Delta(\overline{x}) = \langle \Delta_0, \Delta_1, \ldots \rangle$ be its pre-model. We first build a (tentative) model (σ, λ) for ϕ in the following way: for all $p \in \Sigma$ and all $i \geq 0$, we let $p \in \lambda(\sigma_i)$ iff $p \in \Delta_i$.[2] Then, we show that the model σ, built in this way from $\overline{x}$, is indeed a model satisfying ϕ. To this end, we prove that for all $i \geq 0$ and any $\psi \in \mathcal{C}(\phi)$, if $\psi \in \Delta_i$, then $\sigma, i \models \psi$. This is done by structural induction on ψ:

- if $p \in \Delta_i$ or $\neg p \in \Delta_i$ for some $p \in \Sigma$, then trivially $\sigma, i \models \psi$ by definition of σ;
- if $\alpha \vee \beta \in \Delta_i$, then either $\alpha \in \Delta_i$ or $\beta \in \Delta_i$ since Δ_i is an atom; hence, by the inductive hypothesis, either $\sigma, i \models \alpha$ or $\sigma, i \models \beta$, which implies that $\sigma, i \models \alpha \vee \beta$;
- if $\alpha \wedge \beta \in \Delta_i$, the argument is similar to that for the previous item;
- if $\mathsf{X}\,\alpha \in \Delta_i$, then, by Lemma 1, we have that $\alpha \in \Delta_{i+1}$, and, by the inductive hypothesis, $\sigma, i+1 \models \alpha$, which implies that $\sigma, i \models \mathsf{X}\,\alpha$;
- if $\mathsf{Y}\,\alpha \in \Delta_i$, then, by Lemma 1, we know that $i > 0$ and $\alpha \in \Delta_{i-1}$, and thus $\sigma, i-1 \models \alpha$, which means that $\sigma, i \models \mathsf{Y}\,\alpha$;
- if $\alpha\,\mathcal{U}\,\beta \in \Delta_i$, then, by Lemma 1, there exists $k \geq i$ such that $\beta \in \Delta_i$ and $\alpha \in \Delta_j$ for all $i \leq j \leq k$. Thus, for such k and j, $\sigma, k \models \beta$ and $\sigma, j \models \alpha$, which implies that $\sigma, i \models \alpha\,\mathcal{U}\,\beta$;
- if $\alpha\,\mathcal{S}\,\beta \in \Delta_i$, $\alpha\,\mathcal{R}\,\beta \in \Delta_i$, or $\alpha\,\mathcal{T}\,\beta \in \Delta_i$ the argument is similar to that for the previous item.

Since $\phi \in \Delta_0$, it holds that $\sigma, 0 \models \phi$, that is, σ is a model of ϕ. □

4.2 Completeness

This section proves the completeness of the tableau, showing that, given a model satisfying a formula ϕ, a ticked leaf can be found in any complete tableau for ϕ. Moreover, we will show that this holds regardless of which order was chosen to apply the expansion rules during the construction of the tableau.

The proof is based on the concept of *trace* of a model, which is a particular sequence of nodes built by using the model as a guide to suitably traverse the tableau.

Definition 6 (Trace of a model)**.** Let ϕ be a formula of LTL+P, σ be one of its models ($\sigma \models \phi$), and T be a complete tableau for ϕ. The *trace* of σ for ϕ in T is a pair $\tau_{\phi,T}(\sigma) = (\overline{x}, J)$, where $\overline{x} = \langle x_0, x_1, \ldots \rangle$ is a sequence of nodes from T, with x_0 being the root node, and $J : \mathbb{N} \to \mathbb{N}$ is a mapping between positions in $\overline{x}$ and positions in σ such that $\sigma, J(i) \models \Gamma(x_i)$ for all $i \geq 0$.

Note that, *a priori*, nodes in the trace of a model do not need to be related in any way, *e.g.*, they might even not belong to the same branch, although, as we will see, successful branches are natural examples of traces. We say that a trace is *complete* if it ends with a leaf node. Moreover, given two complete traces τ and τ', we say that τ' *extend* τ if τ is a prefix of τ'.

The next lemma proves that, given a satisfiable formula ϕ of LTL+P, a complete trace can be generated for any model σ of ϕ and any complete tableau T for it.

[2] As we already pointed out, some proposition may be missing from the atoms of a pre-model, being neither asserted nor negated. The model produced by this definition puts to false those propositions, but this is an arbitrary choice, because, by following the soundness proof of Theorem 2, it is clear that any assignment to those propositions results into a sound model.

Lemma 2. *Let ϕ be a formula, σ be a model of ϕ ($\sigma \models \phi$), and T be a complete tableau for ϕ. Then, a complete trace $\tau_{\phi,T}(\sigma)$ can be effectively built.*

Proof. The trace $\tau_{\phi,T}(\sigma) = (\overline{x}, J)$, hereafter simply τ, can be built recursively by letting the model guide us through a traversal of the tree. First of all, observe that x_0 is the root node. Thus, $\Gamma(x_0) = \{\phi\}$ and a sequence of a single node $\langle x_0 \rangle$, with $J(0) = 0$, is a trace, as $\sigma, 0 \models \phi$. Then, given a trace $\tau = (\langle x_0, \ldots, x_n \rangle, J)$ where x_n is not a leaf, we can extend it to a longer trace $\overline{x}' = \langle x_0, \ldots, x_n, x_{n+1} \rangle$ by choosing a suitable node x_{n+1} among the children of x_n, and by properly setting the corresponding $J(n+1)$, depending on the fact that x_n was a poised node or not.

- *If x_n is not a poised node*, then an expansion rule was applied to it, and it has at most two children. If it had only one child x'_n, like in the case of the application of the ALWAYS rule, then the next element in the trace will be $x_{n+1} = x'_n$, and we set $J(n+1) = J(n)$. Then $\overline{x}'$ is a trace, because $\Gamma(x_n) \models \Gamma(x'_n)$: $\Delta \cup \{\alpha \wedge \beta\} \models \Delta \cup \{\alpha, \beta\}$, $\Delta \cup \{\mathsf{G}\,\alpha\} \models \Delta \cup \{\alpha, \mathsf{X}\,\mathsf{G}\,\alpha\}$, and $\Delta \cup \{\mathsf{H}\,\alpha\} \models \Delta \cup \{\alpha, \mathsf{Y}\,\mathsf{H}\,\alpha\}$. Otherwise, x_n has two children, say, x'_n and x''_n. Let us consider the application of the UNTIL rule. It generates two children x'_n and x''_n such that $\beta \in \Gamma(x'_n)$ and $\mathsf{X}(\alpha\,\mathcal{U}\,\beta) \in \Gamma(x''_n)$. Since $\overline{x}$ is a trace, then $\sigma_{J(n)} \models \Delta \cup \{\alpha\,\mathcal{U}\,\beta\}$, and thus either $\sigma_{J(n)} \models \Delta \cup \{\beta\}$ or $\sigma_{J(n)} \models \Delta \cup \{\alpha, \mathsf{X}(\alpha\,\mathcal{U}\,\beta)\}$ (or both), which amounts to say that either $\sigma_{J(n)} \models \Gamma(x'_n)$ or $\sigma_{J(n)} \models \Gamma(x''_n)$ (or both). Hence, we set $J(n+1) = J(n)$, and chose $x_{n+1} = x'_n$, if $\sigma_{J(n)} \models \Gamma(x'_n)$, and $x_{n+1} = x''_n$ otherwise. In both cases, $\overline{x}'$ is still a trace. The other temporal operators are treated similarly. Note that we explicitly chose x'_n over x''_n when the labels of both are satisfied, in order to make the eventuality appear in the trace as soon as possible.
- *If x_n is a poised node*, but it is not a leaf, then the STEP rule was applied to it. In this case, x_n has a child x'_n created by the STEP rule, and a set of additional children $Y = \{x_{n,1}, \ldots, x_{n,m}\}$ added by failed instances of the YESTERDAY rule on some leaves of the subtree rooted at x_n. Then, if there is a $x_{n,j}$ such that $\sigma, J(n) \models \Gamma(x_{n,j})$, then we choose $x_{n+1} = x_{n,j}$ and we set $J(n+1) = J(n)$, making $\overline{x}'$ still a trace. *Only* if no such a node exists, we choose $x_{n+1} = x'_n$ and we let $J(n+1) = i+1$, *i.e.*, we make one step ahead in the model. In this case, $\overline{x}'$ is still a trace since, by definition of the STEP rule, for each $\alpha \in \Gamma(x_{n+1})$, we have $\mathsf{X}\,\alpha \in \Gamma(x_n)$, and since $\sigma, J(n) \models \mathsf{X}\,\alpha$, then $\sigma, J(n+1) \models \alpha$.

Since in any case we extend the trace by choosing a child of x_n as its successor and the tree is finite, the construction will always terminate at a leaf, leading to a *complete* trace. □

The next lemma shows how to extend a complete trace that ends in a leaf crossed by the PRUNE rule. A complete trace can thus be built using the construction of Lemma 2 and, if it ends with a node crossed by the PRUNE rule, extended as shown by Lemma 3. Note that, in general, the traces built in this way are not simple branches of the tree. Rather, they are obtained by suitably composing pieces of branches, which generally go down towards the leaves but occasionally jump back from a leaf up to an ancestor node, and continue from there.

Lemma 3. *Let ϕ be a formula, σ be a model of ϕ, and T be a complete tableau for ϕ. Any* complete *trace that ends in a leaf crossed by the* PRUNE *rule can be effectively* extended.

Proof. Let $\tau = (\langle x_0, \ldots, x_n \rangle, J)$ be a complete trace such that x_n is a leaf that was crossed by the PRUNE rule. This means that the tableau contains some nodes y and z, with $z > y > x_n$, such that, in particular, $\Gamma(x_n) = \Gamma(y) = \Gamma(z)$. Since τ is a trace, then it holds that $\sigma, J(n) \models \Gamma(y)$, and thus the trace can be extended by choosing x_{n+1} and $J(n+1)$ in the same way as in the proof of Lemma 2, but as if x_n was y. Following the construction of Lemma 2, we can reach another leaf, obtaining a new complete trace τ' of which τ is a prefix. □

The next lemma proves that if a complete trace built in this way ends with a crossed leaf, then the leaf can only have been crossed by the PRUNE rule.

Lemma 4. *Let ϕ be a formula, σ be a model of ϕ, T be a complete tableau for ϕ, and $\tau_{\phi,T}(\sigma) = (\langle x_0, \ldots, x_n \rangle, J)$ be a complete trace built as shown in Lemma 2 and then extended any number of times as explained in Lemma 3. If x_n is a crossed node, then it can only have been crossed because of the application of the* PRUNE *rule.*

Proof. We show that the construction outlined in Lemmata 2 and 3 cannot produce a trace $\tau = (\langle x_0, \ldots, x_n \rangle, J)$ such that x_n was crossed because of the CONTRADICTION rule or the YESTERDAY rule.

Suppose by contradiction that x_n was crossed by the CONTRADICTION rule. This cannot be the case in any trace, independently on how it was built, because that would mean $\{p, \neg p\} \subseteq \Gamma(x_n)$, and since τ is a trace, it would contradict the fact that $\sigma, J(n) \models \Gamma(x_n)$.

Then, suppose that x_n was crossed by the YESTERDAY rule, so we know that $\mathsf{Y}\,\alpha \in \Gamma(x_n)$ for some α. The node may have been crossed by the rule for two reasons.

One possibility is that the node has been crossed because there were no previous step nodes in its branch. Then, this is also the first step node in the trace, since the construction in Lemma 2 increments $J(n)$ only in correspondence of the application of the STEP rule on a poised node, and also the extension of the trace of Lemma 3 cannot make the trace jump to the first step node in a branch. Thus, $J(n) = 0$, and since τ is a trace we would have $\sigma_0 \models \mathsf{Y}\,\alpha$, which cannot be the case.

Otherwise, let x_m be the last previous step node of the trace before x_n, so that $J(m) = J(n) - 1$. We know that x_m is also an ancestor of x_n in the tree, *i.e.*, $x_m > x_n$, since the construction of Lemma 2 always goes down a branch and the extension of the trace in Lemma 3 only jumps to non-crossed step nodes. Then, the YESTERDAY rule that crossed x_n would also have added some child x'_m to x_m with label $\Gamma(x'_m) = \Gamma(x_m) \cup \{\alpha\}$. Since x_m is the last previous step node before x_n, it means that the construction chose x_{m+1} to be the child of x_m created by the STEP rule; otherwise, x_m would not be a step node for the branch leading to x_n. If this is the case, by the definition of the YESTERDAY rule, we know that $\sigma_{J(m)} \not\models \Gamma(x'_m)$; otherwise, the construction of the trace would not have chosen x_{m+1}, instead of x'_m, as the successor of x_m. Now, since τ is a trace, we know that $\sigma, J(n) \models \mathsf{Y}\,\alpha$, which amounts to say that $\sigma, J(n)-1 \models \alpha$, *i.e.*, $\sigma, J(m) \models \alpha$. Then, it follows that $\sigma, J(m) \models \Gamma(x_m) \cup \{\alpha\}$, which is the label of x'_m, thus contradicting the previously established fact that $\sigma, J(m) \not\models \Gamma(x'_m)$. □

One may wonder whether the construction outlined in Lemmata 2 and 3 can be iterated forever, leading at each step to a complete trace ending in a leaf crossed by the PRUNE rule, from which the trace can be further extended. The following lemma shows that this cannot be the case, *i.e.*, that a ticked leaf must eventually be found.

Lemma 5. *Let ϕ be a formula, σ be a model of ϕ, T be a complete tableau for ϕ, and $\tau_{\phi,T}(\sigma)$ be a complete trace built as shown in Lemma 2 such that its last node was crossed by the* PRUNE *rule. Then, the extension described in Lemma 3 can be applied only a finite number of times.*

Proof. Suppose by contradiction that the extension process can proceed forever, *i.e.*, that one can repeatedly extend the trace, obtaining an infinite trace $\tau = (\langle x_0, \ldots \rangle, J)$, by executing an infinite number of extension steps, as described in Lemma 3, without ever reaching a ticked leaf. By Lemma 3, at each step i that involves an extension, the trace continues from a node $x_i = w$ that was crossed by the PRUNE rule because of other two step nodes u and v, with $u > v > w$ and $\Gamma = \Gamma(u) = \Gamma(v) = \Gamma(w)$, such that any X-eventuality requested in Γ and fulfilled along

the branch between v and w was also fulfilled between u and v. Thus, each extension of τ can be identified by a triple of nodes $\langle u, v, w\rangle$ that was involved in such a way in the PRUNE rule. If more than one triple apply, choose any one of them.

If an infinite number of extension steps have been applied, then there are some triples that are involved in an infinite number of extensions, since the total number of nodes in the tree is finite. Let us call these the *recurring triples*. Then let $\langle u^*, v^*, w^*\rangle$ be any recurring triple such that no other recurring triple $\langle u, v, w\rangle$ exists with $u > u^*$ (choose any if multiple exist).

Now, observe that, in building τ, there is a finite number of extension steps after which all the non-recurring triples will never be involved in any subsequent extension step, and only extensions involving recurring ones are used in τ from then onwards. Since $\langle u^*, v^*, w^*\rangle$ is used infinitely often in τ, there will be a subsequent extension step N, with $x_N = w^*$, in which we extend the trace using $\langle u^*, v^*, w^*\rangle$ again. From that point on, in the extension of the trace $(\langle x_0, \ldots, x_n\rangle, J)$, with $n \geq N$, the node chosen as x_{n+1} cannot ever be again neither u^* nor an ancestor of u^*, since that would mean that u^* was the descendant of a step node v' from a triple $\langle u', v', w'\rangle$ with $u' > v' > u^*$, which cannot be the case by the maximality of u^*. Thus, we have established that there is a last position k in the trace such that $x_k = u^*$ and, after that, all the x_i are descendants of u^*.

Now consider any X-eventuality $\mathsf{X}\,\mathsf{F}\,\beta$ in $\Gamma(x_k)$ (or, similarly, any X-eventuality $\mathsf{X}(\alpha\,\mathcal{U}\,\beta)$ in $\Gamma(x_k)$). Since at any use of the step rule in getting from x_n to x_{n+1}, the X-eventuality is either fulfilled or postponed to the next state, by a simple induction one can show that $\mathsf{X}\,\mathsf{F}\,\beta$ will be in the label of all the following step nodes x_j of the trace from x_k until the *first* index $b > k$ such that $\beta \in \Gamma(x_b)$ and thus $\sigma, J(b) \models \beta$.

Note that such a b exists, since we do not keep postponing fulfilling the eventuality forever, especially after any index c such that $\sigma, J(c) \models \beta$. Such index c exists, since σ is a model and in the construction in Lemma 2, when $\sigma, J(c) \models \beta$ and we are trying to fulfill $\mathsf{X}\,\mathsf{F}\,\beta$, the child which immediately realizes β is preferred, and the construction realizes $\beta \in \Gamma(x_c)$ even if the model also satisfied $\mathsf{X}\,\mathsf{F}\,\beta$ at the same time.

Now, consider any subsequent extension in τ using the triple $\langle u, v, w\rangle$. If $v > x_b$, it would mean that β would have been fulfilled between v and w, *i.e.*, at x_b, but not between u and v, because x_b was its *first* appearance on the trace and on the branch since $x_k = u^*$. But that contradicts the triggering condition of the PRUNE rule since all the X-eventualities fulfilled between v and w should have been fulfilled between u and v as well. Thus, any extension after the one involving x_k must involve only triples $\langle u, v, w\rangle$ such that $x_k > v$. In other words, after encountering β for the first time, any extension cannot jump back above that point ever again.

This applies to any X-eventuality in $\Gamma(x_k)$, so after a finite number of extensions we will not be able to find again a repetition of $\Gamma(x_k)$, because that would have triggered the LOOP rule instead of the PRUNE, and this contradicts the assumption that $\langle u^*, v^*, w^*\rangle$ is involved in an infinite number of extensions, showing that we cannot infinitely extend the trace. □

Thanks to Lemmata 2 to 5 we are now ready to show the full completeness of the tableau.

Theorem 3 (Completeness)**.** *Let ϕ be a formula, σ be a model of ϕ, and T be a complete tableau for ϕ. Then, T contains at least one ticked leaf.*

Proof. The ticked leaf in the tableau T can be effectively found by using the model σ to build a complete trace $\tau_{\sigma,T}$ as in Lemma 2 and then extending it any number of times as in Lemma 3, until a complete trace ending in a ticked leaf is found. By Lemma 4, at each step the ending leaf can only be either ticked, in which case we found what we were looking for, or crossed by the PRUNE rule, so the extension by Lemma 3 is applicable. By Lemma 5, the trace cannot be extended indefinitely, and thus a ticked leaf has to be eventually found. □

5 Discussion and Related Work

The tableau system described in this paper is an extension of the one described in [23] that supports past operators. This was the first completely tree-shaped tableau for LTL proposed in the literature, and proved to be efficiently usable in practice for some important classes of formulae, especially when compared to other tableaux [3]. Classic graph-shaped tableau systems [17,19,32] are very simple to understand and provide a useful theoretical tool, but they are difficult to implement efficiently, essentially because of their two-pass nature. First, a graph of maximally consistent atoms is built, whose infinite paths include all the possible pre-models of the formula. Then, the graph is traversed to look for a path satisfying all the eventualities, which is usually found by analyzing the strongly connected components of the graph. Since the construction phase is clearly the bottleneck of the procedure, various authors worked to reduce its cost. To this end, an *incremental* construction technique for the tableau has been proposed [14], which only builds the part of the graph reachable from the initial atoms, thus pruning a large number of nodes in common cases. However, the construction phase can still end up in an exponentially sized graph, which is then traversed exactly as in classic procedures.

The first solution that avoids the construction of the graph was the one by Schwendimann [27], who proposed the first one-pass tableaux-based procedure for LTL satisfiability checking. Schwendimann's calculus for LTL is very similar to the one presented here, *i.e.*, a tree-shaped structure is built by expanding nodes labeled by set of formulae, looking for a branch where a loop can be formed which satisfies all the eventualities requested in its nodes. However, the two procedures differ significantly in the way *bad* branches are detected. In Schwendimann's tableau, branches are closed with a loop as soon as a repetition of the label is found. If the loop satisfies all the eventualities, a positive answer is returned, otherwise the branch is discarded. Translated in our terms, it is roughly as if our PRUNE rule was made to cross branches at the first repetition of a label (which would cause incompleteness in our tableau, as we saw in Section 3). However, every non-fulfilling loop is used to update, for each node v, the set of eventualities that are not satisfied by *any* of the branches rooted at v. If each eventuality requested in v is fulfilled in at least *some* of its branches, then the tableau is accepting. This construction requires the tableau structure to maintain back edges to represent the loops, and makes it rely on pieces of information that have to be computed bottom-up. Hence, parallel developments of different branches cannot run in a completely independent way, since their results have to be combined at the end. This also implies that, while the search itself can be implemented keeping only the current branch in memory, extracting models of the formula requires keeping track of all the involved branches. In contrast, in the tableau presented here, each accepting branch is sufficient to directly extract a satisfying model for the formula, and the search procedure on each branch is completely independent from the others.

For the above reasons, our tableau system was recognized as being easy to describe and understand while also being reasonably fast in practice [3]. The simple rule-based structure of the algorithm also suggested that extending it to other logics and additional operators should be reasonably easy. Providing some evidence for this claim, the extension of the tableau system outlined in this paper is non-trivial, but definitely exploits the modularity of the original design. In particular, a new rule is introduced to deal with the *yesterday* temporal operator, without changing neither the one-pass, tree-shaped nature of the tableau nor the shape of any existing rule. Removing the YESTERDAY rule (and the past-specific expansion rules) or executing the algorithm on a future-only formula yields exactly what shown in [23]. The addition of past operators is very easy also in classic graph-shaped tableaux, as can be seen, for instance, in [14,16]. However, it was not obvious *a priori* how to achieve this goal in a modular way in a

one pass tableau. In a graph-shaped tableau, all the possible future and past states of a given state are available in the graph, and it is easy to restrict valid edges to those that respect the semantics of the *yesterday* operator. In a one pass design, however, at any given time we are committed to exploring one possible set of futures (*e.g.*, those that come from having chosen to fulfill an eventuality now instead of tomorrow). Our YESTERDAY rule shows that handling the past in a one-pass system is possible, at the cost of some additional backtracking.

We know from [20] that given an LTL+P formula, an *equivalent* LTL formula may see at least an exponential increase in size, but the result does not apply if *equisatisfiable* formulae are acceptable. In this case, a simple widely known encoding is available which avoids the exponential blowup. For this reason, one may wonder whether handling the past explicitly in the tableau is at all needed, as encoding past operators with the use of future ones is also possible. One may claim, however, that handling the past explicitly is a convenient choice. For example, consider the formula $\phi \equiv \mathsf{F}(\mathsf{P}\, s \vee \mathsf{P}\, t)$. The aforementioned encoding consists of replacing past formulae by proposition letters that are forced to replicate the semantics of past operators with ad-hoc axioms. Thus, an LTL formula equisatisfiable to ϕ is the following one, that adds two proposition letters p_s and p_t which respectively take the place of $\mathsf{P}\, s$ and $\mathsf{P}\, t$:

$$\phi' \equiv \underbrace{\neg p_s \wedge \neg p_t}_{\sigma,0 \not\models \mathsf{Y}\,\alpha} \wedge \underbrace{\mathsf{G}(\mathsf{X}\, p_s \leftrightarrow (p_s \vee s)) \wedge \mathsf{G}(\mathsf{X}\, p_t \leftrightarrow (p_t \vee t))}_{\text{semantics of } \mathsf{P}\, p \text{ and } \mathsf{P}\, q} \wedge \underbrace{\mathsf{F}(p_s \vee p_t)}_{\text{translated } \phi}$$

It can be seen that, even if not exponential, the size increase of ϕ' is quite relevant. The backtracking involved in our YESTERDAY rule is still there, but worsened: the future-only tableau has no knowledge of the semantics of p_s and p_t. An implementation of our YESTERDAY rule in the existing tool described in [3] is underway, and a careful qualitative and quantitative experimental comparison of both approaches is an interesting future goal.

6 Conclusions

This work extended the recently proposed one-pass tree-shaped tableau system for LTL [3, 23] to support *past* temporal operators, providing full proofs of its soundness and completeness. While the soundness proof was not previously published even for the future-only version, the completeness proof was also reworked with respect to [23] in order to cover the new rules for past operators, but also to better highlight the supporting concept of *trace* of a model.

As claimed in [3], the tableau system was found to be easy to extend in a modular way: the changes required, even if non-trivial, can be added without changing the overall structure of the algorithm. An implementation is underway, trying to preserve the speed and low memory consumption that characterizes the original future-only tool. This will be an interesting task by itself, since the new YESTERDAY rule can cause additional backtracking that one may want to limit with suitable heuristics. A thorough experimental comparison with other tools supporting LTL+P is planned for the future, in the line of what done in [3].

Continuing on this front, future work is planned to show that the same system can be adapted in a modular way to support a number of different extensions of LTL. Examples may include metric extensions of LTL [1], LTL interpreted over finite words [6], LTL with *forgettable past* [15], and LTL extended with *freeze quantifiers* [2]. Given the successful implementation available [3], the prospect of having a single unified tool supporting different kinds of linear time temporal logics seems quite promising.

Acknowledgements

This work has been partially supported by the ¡ℕδA⋈ GNCS project *Logic and Automata for Interval Model Checking* (Nicola Gigante and Angelo Montanari), and by the Australian Research Council Discovery Project DP140103365 (Mark Reynolds).

References

[1] R. Alur, T. Feder, and T. A. Henzinger. The Benefits of Relaxing Punctuality. *Journal of the ACM*, 43(1):116–146, 1996.

[2] Rajeev Alur and Thomas A. Henzinger. A Really Temporal Logic. *Journal of the ACM*, 41(1):181–204, 1994.

[3] M. Bertello, N. Gigante, A. Montanari, and M. Reynolds. Leviathan: A new LTL satisfiability checking tool based on a one-pass tree-shaped tableau. In *Proc. of the 25th International Joint Conference on Artificial Intelligence*, pages 950–956. IJCAI/AAAI Press, 2016.

[4] A. R. Cavalli and L. Fariñas del Cerro. A Decision Method for Linear Temporal Logic. In *Proc. of the 7th International Conference on Automated Deduction*, volume 170 of *LNCS*, pages 113–127. Springer, 1984.

[5] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. of the 14th International Conference on Computer Aided Verification*, volume 2404 of *LNCS*, pages 359–364. Springer, 2002.

[6] G. De Giacomo and M. Y. Vardi. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *Proc. of the 23rd International Joint Conference on Artificial Intelligence*, pages 854–860. IJCAI/AAAI Press, 2013.

[7] E. Emerson and J. Halpern. Decision Procedures and Expressiveness in the Temporal Logic of Branching Time. *Journal of Computer and System Sciences*, 30(1):1–24, 1985.

[8] M. Fisher, C. Dixon, and M. Peim. Clausal Temporal Resolution. *ACM Transactions on Computational Logic*, 2(1):12–56, 2001.

[9] D. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic: Mathematical Foundations and Computational Aspects, Volume 1.* Oxford University Press, 1994.

[10] D. M. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the Temporal Basis of Fairness. In *Proc. of the 7th Annual ACM Symposium on Principles of Programming Languages*, pages 163–173, 1980.

[11] V. Goranko, A. Kyrilov, and D. Shkatov. Tableau Tool for Testing Satisfiability in LTL: Implementation and Experimental Analysis. *Electronic Notes in Theoretical Computer Science*, 262:113–125, 2010.

[12] G. Gough. Decision Procedures for Temporal Logics. Technical Report UMCS-89-10-1, Department of Computer Science, University of Manchester, 1989.

[13] I. Hodkinson and M. Reynolds. Separation—Past, Present and Future. In *We Will Show Them: Essays in Honour of Dov Gabbay, Vol 2.*, pages 117–142. College Publications, 2005.

[14] Y. Kesten, Z. Manna, H. McGuire, and A. Pnueli. A Decision Algorithm for Full Propositional Temporal Logic. In *Proc. of the 5th International Conference on Computer Aided Verification*, volume 697 of *LNCS*, pages 97–109. Springer, 1993.

[15] F. Laroussinie, N. Markey, and P. Schnoebelen. Temporal Logic with Forgettable Past. In *Proc. of the 17th IEEE Symposium on Logic in Computer Science*, pages 383–392. IEEE Computer Society, 2002.

[16] O. Lichtenstein and A. Pnueli. Propositional Temporal Logics: Decidability and Completeness. *Logic Journal of the IGPL*, 8(1):55–85, 2000.

[17] Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. The Glory of the Past. In *Proc. of the Logics of Programs Conference*, volume 193 of *LNCS*, pages 196–218. Springer, 1985.

[18] M. Ludwig and U. Hustadt. Implementing a Fair Monodic Temporal Logic Prover. *AI Commun.*, 23(2-3):69–96, 2010.

[19] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems - Safety.* Springer, 1995.

[20] N. Markey. Temporal Logic with Past is Exponentially More Succinct. *Bulletin of the EATCS*, 79:122–128, 2003.

[21] M. Reynolds. More Past Glories. In *Proc. of the 15^{th} IEEE Symposium on Logic in Computer Science*, pages 229–240. IEEE, 2000.

[22] M. Reynolds. An axiomatization of PCTL*. *Information and Computation*, 201:72–119, 2005.

[23] M. Reynolds. A New Rule for LTL Tableaux. In *Proc. of the 7^{th} International Symposium on Games, Automata, Logics and Formal Verification*, volume 226 of *EPTCS*, pages 287–301, 2016.

[24] K. Y. Rozier and M. Y. Vardi. LTL Satisfiability Checking. In *Proc. of the 13^{th} International SPIN Symposium*, pages 149–167, 2007.

[25] K. Y. Rozier and M. Y. Vardi. A Multi-encoding Approach for LTL Symbolic Satisfiability Checking. In *Proc. of the 17^{th} International Symposium on Formal Methods*, pages 417–431, 2011.

[26] V. Schuppan and L. Darmawan. Evaluating LTL Satisfiability Solvers. In *Proc. of the 9^{th} International Symposium on Automated Technology for Verification and Analysis*, pages 397–413, 2011.

[27] S. Schwendimann. A new one-pass tableau calculus for PLTL. In *Proc. of the 7^{th} International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, volume 1397 of *LNCS*, pages 277–292. Springer, 1998.

[28] A. P. Sistla and E. M. Clarke. The Complexity of Propositional Linear Temporal Logics. *Journal of ACM*, 32(3):733–749, 1985.

[29] M. Suda and C. Weidenbach. A PLTL-Prover Based on Labelled Superposition with Partial Model Guidance. In *Proc. of the 6^{th} International Joint Conference on Automated Reasoning*, volume 7364 of *LNCS*, pages 537–543. Springer, 2012.

[30] M. Y. Vardi and P. Wolper. Reasoning About Infinite Computations. *Information and Compututation*, 115(1):1–37, 1994.

[31] G. Venkatesh. A Decision Method for Temporal Logic Based on Resolution. In *Proc. of the 5^{th} Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 206 of *LNCS*, pages 272–289. Springer, 1985.

[32] P. Wolper. The Tableau Method for Temporal Logic: An Overview. *Logique et Analyse*, 28:119–136, 1985.