Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 206 (2008) 59–73

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Systematic Semantic Tableaux for PLTL

J. Gaintzarain, M. Hermo, P. Lucio and M. Navarro[1]

Dpto. de Lenguajes y Sistemas Informáticos.
Facultad de Informática. Universidad del País Vasco.
20080-San Sebastián, Spain.

Abstract

The better known methods of semantic tableaux for deciding satisfiability in propositional linear temporal logic generate graphs in addition to classical trees. The test of satisfaction is made from the graph and it does not correspond with the application of rules in any calculus for PLTL. We present here a new method of semantic tableaux without using additional graphs. The method is based on a new complete finitary sequent calculus for PLTL which allows us to incorporate all the information in a tree. This approach makes our tableaux better suited for completely automatic theorem proving.

Keywords: Propositional Linear Temporal Logic, Tableaux, Satisfiability

1 Introduction

Temporal logics constitute a well-known topic of study in theoretical computer science. One of the most basic and important types of temporal logic is the Propositional (Linear) Temporal Logic (PLTL), which contains logical operators for reasoning about discrete, linear sequences of states. Tableaux are common mechanisms used in most decision procedures for the validity problem and also in tableaux-based completeness proofs for temporal logics.

The semantic tableaux for PL (Propositional Logic) are very simple: the formula is decomposed into its sub-formulas according to certain rules, resulting in a tree-like tableau where each branch is terminated by a leaf with a complementary pair of formulas (a closed branch) or by a leaf containing a set of non-contradictory literals (an open branch). Each open branch represents a model for the given formula.

However, in PLTL, the same approach is not enough, since each formula must be analyzed in an infinite sequence of states. For instance the formula $\phi\,\mathcal{U}\,\psi$ is analyzed as follows: either ψ holds now or else ϕ holds now and $\phi\,\mathcal{U}\,\psi$ holds in the next state.

[1] This work has been partially supported by Spanish Project TIN2004-079250-C03.

1571-0661© 2008 Elsevier B.V. Open access under CC BY-NC-ND license.

doi:10.1016/j.entcs.2008.03.075

Therefore, some mechanism must control the repeated appearances of the formulas and identify periodic situations in finite time [5]. The usual way to proceed is to build a tableau-graph, divide it into its strongly connected components, and check *fulfilling* paths among them to ensure that eventually in the future ψ holds for every $\phi\,\mathcal{U}\,\psi$.

While the application of rules in decomposing the formula can correspond to a deduction in usual sequent calculus for PLTL, the second phase of the procedure that checks for the fulfillment property does not correspond with rules for such calculus.

We can only mention one paper [9] that avoids the second phase in the construction by adding extra information into the nodes of the tableaux. Some of this information must be synthesized bottom-up and it is needed because a single branch cannot be open or closed; it may be open in connection with some other branches.

In this paper we present a systematic tableaux method that, similarly to the PL case, allows us to build a tableau as a finite tree with open and closed branches. If all branches are closed then the formula has no models. If the formula is satisfiable then some of its models can be obtained from the open branches. These models (in general) are ultimately periodic, that is, they terminate in a cycle. Our approach is simpler than [9] since each branch does not depend on other branches.

This approach is based on the use of a particular rule for *until* formulas ($\phi\,\mathcal{U}\,\psi$) to "remember" the context when the unfolding of such formulas occurs. By using this rule, the fulfilling check is incorporated into the construction of the tree. Moreover, this particular rule belongs to the set of rules in a new sound and complete sequent calculus for PLTL [3]. Therefore, each application of a rule in the tableaux is indeed an application of the corresponding rule in the mentioned calculus. In contrast with this calculus, called $\mathcal{FC}$, other sequent calculus [6,7,8] prevent this correspondence.

This paper is organized as follows. Section 2 is a basic introduction to PLTL. In section 3, we introduce the method of semantic tableaux. More precisely, we present the rules for constructing tableaux, the algorithm of construction, and several properties of this algorithm. Section 4 shows some examples, while section 5 presents the soundness and the completeness proof of the method.

2 PLTL: Language and Model Theory

In what follows, we refer to the PLTL language, with syntax and semantics similar to [3].

A PLTL-formula is built using the constant proposition $false$, propositional variables (denoted by lowercase letters $p, q, \ldots$) from a set Prop, the classical connectives $\neg$ and $\wedge$, and the temporal connectives $\circ$ and $\mathcal{U}$. A lowercase Greek letter ($\varphi, \psi, \chi, \gamma, \ldots$) denotes a formula and an uppercase one ($\Phi, \Gamma, \Psi, \Omega, \ldots$) denotes a finite set of PLTL-formulas. Those of the form p and $\neg p$, where $p \in$ Prop, are called *literals*. As usual, other connectives can be defined in terms of the previous ones: $true \equiv \neg false$, $\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$, $\diamond\varphi \equiv true\,\mathcal{U}\,\varphi$, $\Box\varphi \equiv \neg\diamond\neg\varphi$. In the rest of this paper, we employ *formula* instead of PLTL-formula. Formulas of the form

$\varphi\,\mathcal{U}\,\psi$ (also $\diamond\varphi$ and $\neg\Box\varphi$) are called *eventualities*. Those of the form $\circ\varphi$ and $\neg\circ\varphi$ are called *next* formulas.

The operator next translates any set of formulas into another (possibly empty) set of formulas $\mathsf{next}(\Phi) = \{\gamma \mid \circ\gamma \in \Phi\} \cup \{\neg\gamma \mid \neg\circ\gamma \in \Phi\}$.

Definition 2.1 A PLTL-*structure* $\mathcal{M}$ is a pair $(\mathbb{N}, V_{\mathcal{M}})$ where $\mathbb{N}$ is the set of natural numbers and $V_{\mathcal{M}} : \mathbb{N} \to 2^{\mathsf{Prop}}$ maps each state $n \in \mathbb{N}$ into a subset of Prop.

Intuitively, $V_{\mathcal{M}}$ specifies which propositional variables are necessarily true in each state.

Definition 2.2 The truth of a formula φ in the state j of a PLTL-structure $\mathcal{M}$, which is denoted by $\langle\mathcal{M}, j\rangle \models \varphi$, is inductively defined as follows:

$\langle\mathcal{M}, j\rangle \not\models false$

$\langle\mathcal{M}, j\rangle \models p$ iff $p \in V_{\mathcal{M}}(j)$ for $p \in \mathsf{Prop}$

$\langle\mathcal{M}, j\rangle \models \neg\varphi$ iff $\langle\mathcal{M}, j\rangle \not\models \varphi$

$\langle\mathcal{M}, j\rangle \models \varphi \wedge \psi$ iff $(\langle\mathcal{M}, j\rangle \models \varphi$
and $\langle\mathcal{M}, j\rangle \models \psi)$

$\langle\mathcal{M}, j\rangle \models \circ\varphi$ iff $\langle\mathcal{M}, j+1\rangle \models \varphi$

$\langle\mathcal{M}, j\rangle \models \varphi\,\mathcal{U}\,\psi$ iff $\langle\mathcal{M}, k\rangle \models \psi$ for some $k \geq j$ and $\langle\mathcal{M}, i\rangle \models \varphi$ for every $j \leq i < k$.

This is extended to sets in the usual way: $\langle\mathcal{M}, j\rangle \models \Phi$ iff $\langle\mathcal{M}, j\rangle \models \varphi$ for all $\varphi \in \Phi$. We say that $\mathcal{M}$ is a model of Φ, in symbols $\mathcal{M} \models \Phi$, iff $\langle\mathcal{M}, 0\rangle \models \Phi$. A satisfiable set of formulas has at least one model, otherwise it is unsatisfiable.

3 Semantic Tableaux for PLTL

In this section we present a method for semantic tableaux which, given a temporal formula, searches systematically for a model. If a model is found, the formula is satisfiable; otherwise, it is unsatisfiable.

3.1 Rules for Constructing a Semantic Tableau

We use the following α-, β-, X-rules in the construction of semantic tableaux for, respectively, α-formulas (conjunctions), β-formulas (disjunctions) and *next* formulas. α_1 denotes the (set of) conjuncts of a α-formula, β_1, β_2 denote the (sets of) disjuncts of a β-formula and X_1 denotes the application of the operator next to a set of formulas.

	α	α_1
$(r1)$	$\neg\neg\varphi$	φ
$(r2)$	$\varphi \wedge \psi$	φ, ψ

	β	β_1	β_2
$(r3)$	$\neg(\varphi \wedge \psi)$	$\neg\varphi$	$\neg\psi$
$(r4)$	$\neg(\varphi\,\mathcal{U}\,\psi)$	$\neg\varphi, \neg\psi$	$\varphi, \neg\psi, \neg\circ(\varphi\,\mathcal{U}\,\psi)$
$(r5)$	$\varphi\,\mathcal{U}\,\psi$	ψ	$\varphi, \neg\psi, \circ(\varphi\,\mathcal{U}\,\psi)$
$(r6)$	$\Gamma, \varphi\,\mathcal{U}\,\psi$	Γ, ψ	$\Gamma, \varphi, \neg\psi, \circ((\Gamma^* \wedge \varphi)\,\mathcal{U}\,\psi)$ where $\Gamma^* = \neg(\bigwedge_{\gamma\in\Gamma}\gamma)$

	X	X_1
$(r7)$	Γ	$\mathsf{next}(\Gamma)$

All rules but $(r6)$ are the usual ones in temporal tableaux construction (see for instance [1]). Rule $(r6)$ is introduced in [3], where a new sound and complete calculus for PLTL is presented. This rule allows us to "remember" the context Γ when unfolding of $\varphi\,\mathcal{U}\,\psi$ is done. More concretely, it forces some formula in the context to change in future worlds (while ψ is not obtained).

Although it is enough to use the minimal set of operators given above, we will use all operators in the examples in section 4. Below the usual β-rules for $\varphi \vee \psi$, $\Diamond\varphi$ and $\neg\Box\varphi$, and α-rules for $\neg(\varphi \vee \psi)$, $\Box\varphi$ and $\neg\Diamond\varphi$ are shown. Note that they can be derived from the above rules. In particular, the rules with context Γ are derived from rule $(r6)$.

α	α_1
$\Box\varphi$	$\varphi, \circ\Box\varphi$
$\neg\Diamond\varphi$	$\neg\varphi, \neg\circ\Diamond\varphi$
$\neg(\varphi \vee \psi)$	$\neg\varphi, \neg\psi$

β	β_1	β_2
$\varphi \vee \psi$	φ	ψ
$\Diamond\varphi$	φ	$\neg\varphi, \circ\Diamond\varphi$
$\Gamma, \Diamond\varphi$	Γ, φ	$\Gamma, \neg\varphi, \circ(\Gamma^*\,\mathcal{U}\,\varphi)$
$\neg\Box\varphi$	$\neg\varphi$	$\varphi, \neg\circ\Box\varphi$
$\Gamma, \neg\Box\varphi$	$\Gamma, \neg\varphi$	$\Gamma, \varphi, \circ(\Gamma^*\,\mathcal{U}\,\neg\varphi)$

3.2 Preclosure and Closure of a Temporal Formula

Let φ be the formula whose satisfiability we wish to check.

Definition 3.1 The set of components in φ, *Comp*(φ), is the smallest set of formulas defined as follows:

- $\varphi \in Comp(\varphi)$
- If $\gamma \in Comp(\varphi)$, then all the formulas appearing in the parts $\alpha_1, X_1, \beta_1, \beta_2$ of rules $(r1), \ldots, (r7)$, except $(r6)$, that can be applied to γ, are in $Comp(\varphi)$.

Next we define the pre-closure of φ.

Definition 3.2 $PCL(\varphi) = Comp(\varphi) \cup \{\neg\gamma \mid \gamma \in Comp(\varphi)\} \cup \{\gamma \mid \neg\gamma \in Comp(\varphi)\}$

Note that the pre-closure of a formula does not consider those produced by the application of rule $(r6)$. We will define the closure adding these formulas to the pre-closure.

Definition 3.3 Let $D(\varphi)$ be the following set of formulas

$$D(\varphi) = \{\neg(\bigwedge_{\psi \in Q} \psi) : Q \subseteq PCL(\varphi)\} \cup \{false\}$$

Let $C(\varphi)$ be the set of all possible conjunctions of elements in $D(\varphi)$. That is,

$$C(\varphi) = \{ \bigwedge_{D \in S} D : S \subseteq D(\varphi)\}$$

The closure of φ , $CL(\varphi)$, is defined as follows

$$CL(\varphi) = PCL(\varphi) \cup C(\varphi) \cup \mathcal{A}$$

where

$$\mathcal{A} = \bigcup_{\substack{(\gamma\,\mathcal{U}\,\psi) \in \mathcal{PCL}(\varphi) \\ \mathcal{C} \in \mathcal{C}(\varphi)}} \{\circ((\mathcal{C} \wedge \gamma)\,\mathcal{U}\,\psi),\ (\mathcal{C} \wedge \gamma)\,\mathcal{U}\,\psi\}$$

From the above definition, the following holds:

Proposition 3.4 *$CL(\varphi)$ has finite cardinality. Actually, if $|PCL(\varphi)| = n$ then $|D(\varphi)| \in O(2^n)$ and both $|C(\varphi)|, |CL(\varphi)| \in O(2^{2^n})$.*

3.3 Systematic Construction of a Semantic Tableau

A tableau $\mathcal{T}$ is a tree where each node n is labeled with a set of formulas $F(n)$. The root is labeled with the singleton set $\{\varphi\}$, for the formula φ whose satisfiability we wish to check. The children of a node are obtained by applying the rules $(r1), \ldots, (r7)$.

Definition 3.5 Let $\mathcal{T}$ be a tableau and p be a path in $\mathcal{T}$ from nodes $n_1, n_2, \ldots, n_j$. Any eventuality $\gamma_1\,\mathcal{U}\,\gamma_2 \in F(n_i)$, with $1 \leq i \leq j$, is fulfilled in p if there exists k, with $i \leq k \leq j$, such that $\gamma_2 \in F(n_k)$.

To determine which rule $(r5)$ or $(r6)$ to apply to an eventuality in a node, it is necessary to "distinguish" eventualities. The rule $(r6)$ is applied only to "distinguish" eventualities, in other case the rule $(r5)$ is used. If a node does not contain any distinguished eventuality, then the algorithm distinguishes one of them and rule $(r6)$ is chosen to be applied to it. Each node of the tableau has at most one distinguished eventuality.

Each branch of $\mathcal{T}$ can be seen as divided into *stages*, where each stage is a set of consecutive nodes which are obtained by applying α- or β-rules. When the X-rule is applied, we move from one stage to the following one in the branch.

Given two nodes in the same branch of $\mathcal{T}$, $n1$ and $n2$, that are labeled with the same set of formulas Φ (i.e. $F(n1)$=$F(n2)$=Φ), the path between such nodes is called a *loop*.

The construction of the tableau is as follows.

Input: A $PLTL$ formula φ
Output: A semantic tableau $\mathcal{T}$ for φ

Algorithm:
The tableau is built inductively by repeatedly choosing an unmarked leaf l labeled with a set of formulas $F(l)$ and applying one of the following points in the order given.

(i) Check if there is either the formula $false$ or a complementary pair of formulas $\{\varphi, \neg\varphi\}$ in $F(l)$. If so, mark the leaf *closed* ($\times$).

(ii) If $F(l)$ is a set of literals, then mark the leaf *open* ($\odot$).

(iii) If $F(l) = F(l')$ for l' an ancestor of l, take the oldest ancestor of l that is labeled with $F(l)$ (denote it by l''). Now check if each eventuality in the path between l'' and l is fulfilled in such path. If this is the case, mark the leaf *open* ($\odot$).

(iv) Otherwise, choose $\varphi \in F(l)$ which is not a *next* formula.

- If the formula is an α-formula ($\varphi = \alpha$), create a new node l' as a child of l and label l' with
$$F(l') = (F(l) - \{\alpha\}) \cup \{\alpha_1\}$$
- If the formula is a β-formula ($\varphi = \beta$), create two new nodes l' and l'' as children of l. Label l' with
$$F(l') = (F(l) - \{\beta\}) \cup \{\beta_1\}$$
and label l'' with
$$F(l'') = (F(l) - \{\beta\}) \cup \{\beta_2\}$$
In this case, if φ is an eventuality, then
  – If φ is the distinguished eventuality, then apply the special rule $(r6)$ to φ. Distinguish the formula that is inside the *next* formula in β_2.
  – If φ is not distinguished, but there is another distinguished formula, then

apply rule $(r5)$ to φ. Maintain the existing distinguished formula in β_1 and β_2.

- Otherwise, distinguish the formula φ. Apply the special rule $(r6)$ to φ and distinguish the formula that is inside the *next* formula in β_2.

(v) If $F(l)$ consists only of literals and *next* formulas, the operator next is applied. That is, let

$$\{\circ\varphi_1, \ldots, \circ\varphi_m, \neg\circ\varphi_{m+1}, \ldots, \neg\circ\varphi_n\}$$

be the set of *next* formulas in $F(l)$. Create a new node l' as a child of l and label l' with

$$F(l') = \{\varphi_1, \ldots, \varphi_m, \neg\varphi_{m+1}, \ldots, \neg\varphi_n\}$$

The construction terminates when every leaf is marked × or $\odot$. □

A tableau whose construction has terminated is called a *completed* tableau. A completed tableau is *closed* if all leaves are marked ×. Otherwise, it is *open*.

3.4 Properties of this Construction

Here, we give some conditions that our algorithm must hold to ensure its termination, as well as other properties which are necessary for section 5.

Definition 3.6 A node in the tableau is *inconsistent* if it contains a formula and its negation, or the constant $false$.

Proposition 3.7 *If a formula γ and its negation $\neg\gamma$ belong to the same stage in a branch of the tableau, and γ does not contain an eventuality that is distinguished in the stage, then this stage finishes in an inconsistent node.*

Proof. It can be easily proven by structural induction on the formula γ. Note that there are two base cases: for γ being a literal p and a *next* formula $\circ\alpha$, since these formulas must remain (after its first appearance) in all following nodes of the stage. □

Proposition 3.8 *All contexts accumulated in a distinguished eventuality in an open branch are pairwise different.*

Proof. Let n be a node with $F(n) = \Gamma \cup \{(\Gamma^*{}_i \wedge \ldots \wedge \Gamma^*{}_1 \wedge \alpha)\,\mathcal{U}\,\beta\}$, and $n+1$ the node obtained as the β_2-part of applying rule $(r6)$ to the distinguished eventuality in n. That is, $F(n+1) = \Gamma \cup \{\neg\beta, \Gamma^*{}_i \wedge \ldots \wedge \Gamma^*{}_1 \wedge \alpha, \circ((\Gamma^* \wedge \Gamma^*{}_i \wedge \ldots \wedge \Gamma^*{}_1 \wedge \alpha)\,\mathcal{U}\,\beta)\}$ where the last formula contains the eventuality that remains distinguished in all nodes of this stage. By applying i times rule $(r2)$, the branch is extended until node $F(n+i+1) = \Gamma \cup \{\neg\beta, \Gamma^*{}_i, \ldots \Gamma^*{}_1, \alpha, \circ((\Gamma^* \wedge \Gamma^*{}_i \wedge \ldots \wedge \Gamma^*{}_1 \wedge \alpha)\,\mathcal{U}\,\beta)\}$.

Let us suppose that $\Gamma^* \in \{\Gamma^*{}_i, \ldots \Gamma^*{}_1\}$ for $\Gamma = \{\gamma_1, \ldots, \gamma_p\}$. Then $\{\gamma_1, \ldots, \gamma_p, \neg(\gamma_1 \wedge \ldots \wedge \gamma_p)\} \subseteq F(n+i+1)$ and therefore, by applying $p-1$ times rule $(r3)$, the branch split (in some moment) into p branches containing each one a complementary pair of formulas (γ_j and $\neg\gamma_j$) in the same stage. Moreover, each formula γ_j, for j in $1, \ldots, p$, cannot contain any distinguished eventuality. By Proposition 3.7, all these branches are closed. □

Note that although there can be some formulas in the tableau $\mathcal{T}$ for φ that do not belong to $CL(\varphi)$ because contexts are repeated, the previous proposition proves that all formulas in any open branch of $\mathcal{T}$ belong to $CL(\varphi)$.

The following proposition concerns the open branches containing a loop.

Proposition 3.9 *For every eventuality $\alpha \,\mathcal{U}\, \delta$ which is distinguished inside a loop, it holds that δ also belongs to (some node in) the loop.*

Proof. Let $\alpha \,\mathcal{U}\, \delta$ be distinguished in a node of a loop. Then the rule $(r6)$ is applied to $\alpha \,\mathcal{U}\, \delta$ within some context Γ_1. If δ does not belong to the loop, this means that each application of such rule in the loop yields to obtain the β_2-formulas; in particular, in the following stages $(i \geq 1)$, distinguished formulas are obtained: $\gamma_1 = (\Gamma^*{}_1 \wedge \alpha) \,\mathcal{U}\, \delta$, $\gamma_2 = (\Gamma^*{}_2 \wedge \Gamma^*{}_1 \wedge \alpha) \,\mathcal{U}\, \delta$, $\ldots$, $\gamma_i = (\Gamma^*{}_i \wedge \ldots \wedge \Gamma^*{}_1 \wedge \alpha) \,\mathcal{U}\, \delta$, for some contexts Γ_i.

By Proposition 3.8, these formulas are all new (they are syntactically growing), and in each node n of stage i either γ_i or $\circ\gamma_{i+1}$ is in the set $F(n)$. This contradicts the existence of a repeated node. Therefore δ must belong to the loop. □

Next, the construction of the tableau is open to be implemented in different ways. However, any implementation must ensure that it is not possible to find a never distinguished unfulfilled eventuality in an open branch.

Remark 3.10 The use of a *fair strategy* for distinguishing the eventualities in each branch of the tableau, is essential for proving that the construction finishes.

Theorem 3.11 *Any semantic tableau for a temporal formula φ, that distinguishes eventualities with a fair strategy, is finite.*

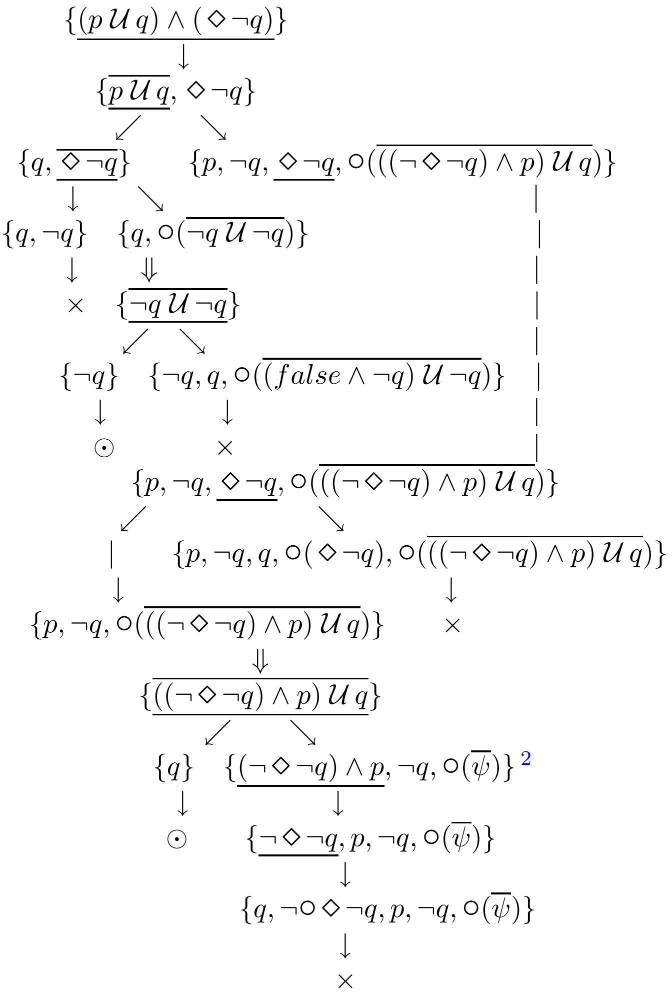
Proof. Suppose that a tableau for φ contains an infinite branch p. Then the sets of formulas labeling the nodes in p are included in $CL(\varphi)$ which, by Proposition 3.4, has finite cardinality. Then there are only finitely many possible different nodes in p. Thus, there must exist a node occurring infinite many times in p. This means that there is an unfulfilled eventuality, and by proposition 3.9, this eventuality is never distinguished. But this contradicts the fact that the strategy for distinguishing eventualities is fair. Therefore, the tableau cannot contain an infinite branch. □

To conclude, it is worth saying that the implementation of the algorithm must build the tableau incrementally using a deep-first strategy. Thus, when a node is marked open, the algorithm stops providing a model for the formula.

4 Examples of Semantic Tableaux

In this section, we give some examples of tableaux. The distinguished formulas are overlined. The formula which a rule is applied to is underlined. The application of operator next is drawn with ⇓ (instead of ↓) to better mark the stages in each branch. For purposes of visual clarity, we sometimes omit the application of trivial rules like $(r1)$, which eliminates double negations. In these examples, all connectives are used and consequently all α and β-rules. In addition, Γ^* is written $\bigvee_{\gamma\in\Gamma}\neg\gamma$ instead of $\neg(\bigwedge_{\gamma\in\Gamma}\gamma)$.

The first example shows an open tableau without loops for the formula $(p\,\mathcal{U}\,q)\wedge(\Diamond\neg q)$.


$\{\underline{(p\,\mathcal{U}\,q)\wedge(\Diamond\neg q)}\}$
$\{\underline{\overline{p\,\mathcal{U}\,q}}, \Diamond\neg q\}$
$\{q, \underline{\overline{\Diamond\neg q}}\}$
$\{p, \neg q, \underline{\Diamond\neg q}, \circ(\overline{((\neg\Diamond\neg q)\wedge p)\,\mathcal{U}\,q})\}$
$\{q, \neg q\}$
$\{q, \circ(\overline{\neg q\,\mathcal{U}\,\neg q})\}$
×
$\{\underline{\overline{\neg q\,\mathcal{U}\,\neg q}}\}$
$\{\neg q\}$
$\{\neg q, q, \circ(\overline{(false\wedge\neg q)\,\mathcal{U}\,\neg q})\}$
$\odot$
×
$\{p, \neg q, \underline{\Diamond\neg q}, \circ(\overline{((\neg\Diamond\neg q)\wedge p)\,\mathcal{U}\,q})\}$
$\{p, \neg q, q, \circ(\Diamond\neg q), \circ(\overline{((\neg\Diamond\neg q)\wedge p)\,\mathcal{U}\,q})\}$
$\{p, \neg q, \circ(\overline{((\neg\Diamond\neg q)\wedge p)\,\mathcal{U}\,q})\}$
×
$\{\underline{\overline{((\neg\Diamond\neg q)\wedge p)\,\mathcal{U}\,q}}\}$
$\{q\}$
$\{\underline{(\neg\Diamond\neg q)\wedge p}, \neg q, \circ(\overline{\psi})\}$ [2]
$\odot$
$\{\underline{\neg\Diamond\neg q}, p, \neg q, \circ(\overline{\psi})\}$
$\{q, \neg\circ\Diamond\neg q, p, \neg q, \circ(\overline{\psi})\}$
×

Note that each branch finishing with the mark $\odot$, produces a finite structure from which a model can be constructed (see Figure 1).

The different stages in a branch correspond to the different worlds in the structure associated to such a branch.

The second example shows a closed tableau for the formula $p\,\mathcal{U}\,q\wedge\neg\circ\Diamond q\wedge\neg q$. Note that each branch finishes with the mark ×. The formula $p\,\mathcal{U}\,q\wedge\neg\circ\Diamond q\wedge\neg q$ has no models.

[2] where $\psi = (false\wedge(\neg\Diamond\neg q)\wedge p)\,\mathcal{U}\,q$

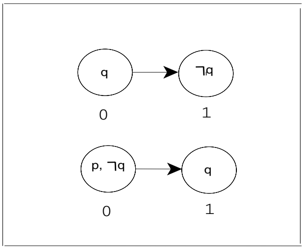


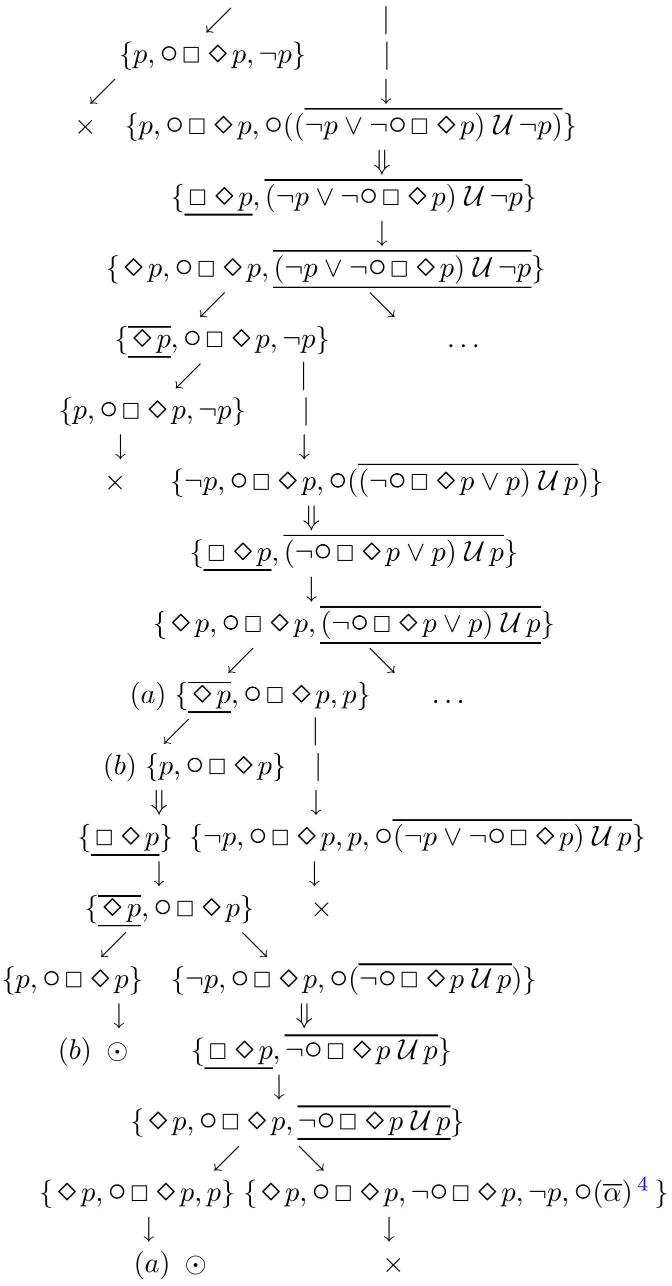
Fig. 1. Two structures for $\{(p\,\mathcal{U}\,q) \wedge (\Diamond\neg q)\}$

$$\begin{tikzcd}
 & \{\underline{p\,\mathcal{U}\,q \wedge \neg\circ\Diamond q \wedge \neg q}\} \arrow[d] & \\
 & \{p\,\mathcal{U}\,q, \underline{\neg\circ\Diamond q \wedge \neg q}\} \arrow[d] & \\
 & \{\overline{\underline{p\,\mathcal{U}\,q}}, \neg\circ\Diamond q, \neg q\} \arrow[dl] \arrow[ddd, dash] & \\
\{q, \neg\circ\Diamond q, \neg q\} \arrow[d] & & \\
\times & & \\
 & \{p, \neg q, \neg\circ\Diamond q, \circ(\overline{((q \vee \circ\Diamond q) \wedge p)\,\mathcal{U}\,q})\} \arrow[d, Rightarrow] & \\
 & \{\underline{\neg\Diamond q}, \overline{((q \vee \circ\Diamond q) \wedge p)\,\mathcal{U}\,q}\} \arrow[d] & \\
 & \{\neg q, \neg\circ\Diamond q, \overline{\underline{((q \vee \circ\Diamond q) \wedge p)\,\mathcal{U}\,q}}\} \arrow[dl] \arrow[d] & \\
\{\neg q, \neg\circ\Diamond q, q\} \arrow[d] & \{\neg q, \neg\circ\Diamond q, \underline{(q \vee \circ\Diamond q) \wedge p}, \circ(\overline{\delta})\}\text{[3]} \arrow[d] & \\
\times & \{\neg q, \neg\circ\Diamond q, \underline{(q \vee \circ\Diamond q)}, p, \circ(\overline{\delta})\} \arrow[dl] \arrow[dr] & \\
\{\neg q, \neg\circ\Diamond q, q, p, \circ(\overline{\delta})\} \arrow[d] & & \{\neg q, \neg\circ\Diamond q, \circ\Diamond q, p, \circ(\overline{\delta})\} \arrow[d] \\
\times & & \times
\end{tikzcd}$$

The last example shows (part of) an open tableau with loops (indicated (a) and (b)) for the formula $\Box\Diamond p \wedge \Diamond\neg p$. Each branch finishing with the mark $\odot$ produces a model (see Figure 2).

$$\begin{tikzcd}
 & \{\underline{\Box\Diamond p \wedge \Diamond\neg p}\} \arrow[d] & \\
 & \{\underline{\Box\Diamond p}, \Diamond\neg p\} \arrow[d] & \\
 & \{\overline{\underline{\Diamond p}}, \circ\Box\Diamond p, \Diamond\neg p\} \arrow[dl] \arrow[dr] & \\
\{p, \circ\Box\Diamond p, \overline{\underline{\Diamond\neg p}}\} & & \dots
\end{tikzcd}$$

[3] where $\delta = [(q \vee \circ\Diamond q) \wedge (q \vee \circ\Diamond q) \wedge p]\,\mathcal{U}\,q$


$\{p, \circ\Box\Diamond p, \neg p\}$
$\times$
$\{p, \circ\Box\Diamond p, \circ((\neg p \vee \neg\circ\Box\Diamond p)\,\mathcal{U}\,\neg p)\}$
$\{\Box\Diamond p, (\neg p \vee \neg\circ\Box\Diamond p)\,\mathcal{U}\,\neg p\}$
$\{\Diamond p, \circ\Box\Diamond p, (\neg p \vee \neg\circ\Box\Diamond p)\,\mathcal{U}\,\neg p\}$
$\{\Diamond p, \circ\Box\Diamond p, \neg p\}$
$\dots$
$\{p, \circ\Box\Diamond p, \neg p\}$
$\times$
$\{\neg p, \circ\Box\Diamond p, \circ((\neg\circ\Box\Diamond p \vee p)\,\mathcal{U}\,p)\}$
$\{\Box\Diamond p, (\neg\circ\Box\Diamond p \vee p)\,\mathcal{U}\,p\}$
$\{\Diamond p, \circ\Box\Diamond p, (\neg\circ\Box\Diamond p \vee p)\,\mathcal{U}\,p\}$
(a) $\{\Diamond p, \circ\Box\Diamond p, p\}$
$\dots$
(b) $\{p, \circ\Box\Diamond p\}$
$\{\Box\Diamond p\}$
$\{\neg p, \circ\Box\Diamond p, p, \circ(\neg p \vee \neg\circ\Box\Diamond p)\,\mathcal{U}\,p\}$
$\{\Diamond p, \circ\Box\Diamond p\}$
$\times$
$\{p, \circ\Box\Diamond p\}$
$\{\neg p, \circ\Box\Diamond p, \circ(\neg\circ\Box\Diamond p\,\mathcal{U}\,p)\}$
(b) $\odot$
$\{\Box\Diamond p, \neg\circ\Box\Diamond p\,\mathcal{U}\,p\}$
$\{\Diamond p, \circ\Box\Diamond p, \neg\circ\Box\Diamond p\,\mathcal{U}\,p\}$
$\{\Diamond p, \circ\Box\Diamond p, p\}$
$\{\Diamond p, \circ\Box\Diamond p, \neg\circ\Box\Diamond p, \neg p, \circ(\overline{\alpha})$ [4] $\}$
(a) $\odot$
$\times$

5 Soundness and Completeness of Semantic Tableaux

In this section we prove that our algorithm is sound and complete for proving the satisfiability of PLTL formulas. Soundness is given in Theorem 5.2 and completeness in Theorem 5.3.

[4] where $\alpha = [(\neg\Diamond p \vee \neg\circ\Box\Diamond p) \wedge (\neg\circ\Box\Diamond p)]\,\mathcal{U}\,p$

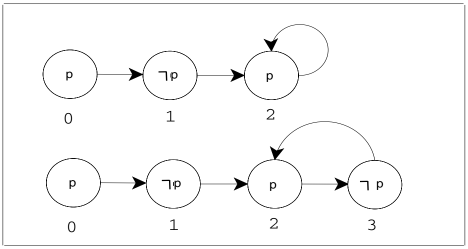


Fig. 2. Two models for $\{\square\ \diamond p \wedge\ \diamond \neg p\}$

Lemma 5.1 *The following facts hold for α- and β- formulas:*

- $\Phi \bigcup \{\alpha\}$ *satisfiable* $\Longleftrightarrow \Phi \bigcup \{\alpha_1\}$ *satisfiable*
- $\Phi \bigcup \{\beta\}$ *satisfiable* $\Longleftrightarrow \Phi \bigcup \{\beta_1\}$ *satisfiable or* $\Phi \bigcup \{\beta_2\}$ *satisfiable*
- *Given a consistent (without any complementary pair of formulas) set of formulas* Φ *consisting of literals and* next *formulas:* Φ *satisfiable* $\Longleftrightarrow$ *next*(Φ) *satisfiable*

Theorem 5.2 *If there exists a closed tableau for φ then φ is unsatisfiable.*

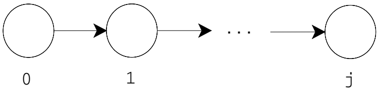
Proof. Let $\mathcal{T}$ be a closed tableau for φ. The set of formulas labeling each leaf is inconsistent and therefore unsatisfiable. By the previous lemma, each node in $\mathcal{T}$ is then labeled with a unsatisfiable set of formulas, in particular the root. Therefore φ is unsatisfiable. □

Theorem 5.3 *If there exists an open tableau for φ then φ is satisfiable.*

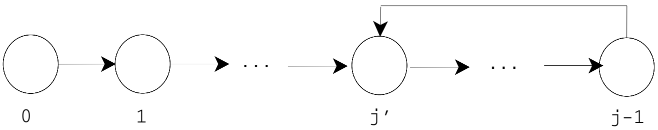
Proof. Let $\mathcal{T}$ be an open tableau for φ. There exists a leaf n in $\mathcal{T}$, marked *open*, labeled with a set of formulas $F(n)$. Let R be the branch in $\mathcal{T}$ from the root until leaf n and let j be the stage in R the node n belongs to. We shall build from R a model for φ.

Consider the structure $\mathcal{M}$ with a sequence of states $0, 1, 2, \ldots$, where $\langle \mathcal{M}, i\rangle$ is the set of all literals appearing in all nodes (or equivalently in the last node) of the stage i in R, for $i = 0, 1, \ldots$. Distinguish two main cases:

(i) The leaf n is marked *open* because $F(n)$ is a (consistent) set of literals. Then $\mathcal{M}$ is a finite structure that can be seen as follows:



(ii) The leaf n is marked *open* because the set of formulas $F(n)$ coincides with the set of formulas in some ancestor nodes of n in R. Let m be the oldest ancestor node of n in R such that $F(m) = F(n)$. Then all eventualities in the path between m and n are fulfilled in the path. Let j' be the stage in R the node m belongs to, for some $0 \le j' < j$. Then $\mathcal{M}$ is an infinite structure with a final loop that can be seen as follows:



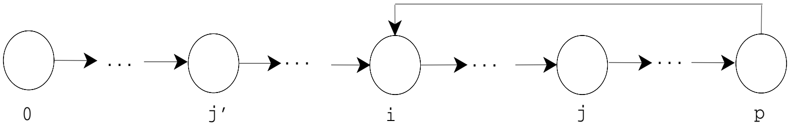
In the next auxiliar lemma we shall prove that every formula ϕ in a node of stage i in R is satisfied in the structure $\mathcal{M}$, that is, $\langle \mathcal{M}, i\rangle \models \phi$. Then, in particular, $\langle \mathcal{M}, 0\rangle \models \varphi$.
Finally, it is well known that $\mathcal{M}$ can be extended to an interpretation by adding to $\langle \mathcal{M}, i\rangle$ all propositional variables p (from φ) such that neither p nor $\neg p$ are in the stage i in R, for every $i \geq 0$. □

Now we prove the auxiliary lemma used in the proof of Theorem 5.3.

Lemma 5.4 *For every formula ϕ appearing at (a node of) stage i in R, it holds that $\langle \mathcal{M}, i\rangle \models \phi$.*

Proof. Let $\phi \in_i R$ denote that ϕ appears at (a node of) stage i in R, with $i = 0, 1, \ldots, j$. We proceed by structural induction on ϕ.

- The base case, ϕ literal, holds by construction of $\mathcal{M}$.
- Cases $\neg\neg\phi_1$, $\phi_1 \wedge \phi_2$, $\circ\phi_1$, $\neg\circ\phi_1$ and the case $\neg(\phi_1 \wedge \phi_2)$ can be easily proven by induction hypothesis on ϕ_1 and ϕ_2.
- Case $\neg(\phi_1 \,\mathcal{U}\, \phi_2)$.
  - When $\mathcal{M}$ is a finite structure, it is obvious that for some k in $i \ldots j$ it holds $\neg\phi_1 \in_k R$. Otherwise, the leaf n contains the formula $\neg(\phi_1 \,\mathcal{U}\, \phi_2)$ which is not a literal. Also $\neg\phi_2 \in_s R$ for every s in $i \ldots k$. Then, by induction, $\langle \mathcal{M}, k\rangle \models \neg\phi_1$ and $\langle \mathcal{M}, s\rangle \models \neg\phi_2$, for every s in $i \ldots k$. Therefore $\langle \mathcal{M}, i\rangle \models \neg(\phi_1 \,\mathcal{U}\, \phi_2)$.
  - When $\mathcal{M}$ is an infinite structure with a final loop, let us first suppose that stage i is not inside the loop, that is, suppose $i < j'$. Hence, either the same occurs as in the previous case, or else $\neg\phi_2 \in_s R$ for every $s \geq i$. Both facts yield to $\langle \mathcal{M}, i\rangle \models \neg(\phi_1 \,\mathcal{U}\, \phi_2)$.
  - When $\mathcal{M}$ is an infinite structure with a final loop, but stage i is inside the loop, we must also take into account the stages from j until the second occurrence of i (these are also future worlds for i).
    This situation can be converted to the previous one just considering that the structure $\mathcal{M}$ can be seen as the structure $\mathcal{M}'$ defined $\langle \mathcal{M}', k\rangle = \langle \mathcal{M}, k\rangle$ for k in $0 \ldots j-1$, and $\langle \mathcal{M}', j+s\rangle = \langle \mathcal{M}, j'+s\rangle$ for s in $0 \ldots (j-j'-1)$
    That is, $\mathcal{M}'$ is the following structure, where $p = 2j - j' - 1$. Then it holds: $\neg(\phi_1 \,\mathcal{U}\, \phi_2) \in_i R \Longrightarrow \langle \mathcal{M}', i\rangle \models \neg(\phi_1 \,\mathcal{U}\, \phi_2) \iff \langle \mathcal{M}, i\rangle \models \neg(\phi_1 \,\mathcal{U}\, \phi_2)$.



- Case $\phi_1 \,\mathcal{U}\, \phi_2$.

- When $\mathcal{M}$ is a finite structure, it is obvious that for some k in $i \dots j$ it holds $\phi_2 \in_k R$. Otherwise, the leaf n contains a formula which is not a literal. Also $\phi_1 \in_s R$ for every s in $i \dots k-1$. Then, by induction, $\langle \mathcal{M}, k \rangle \models \phi_2$ and $\langle \mathcal{M}, s \rangle \models \phi_1$, for every s in $i \dots k-1$. Therefore $\langle \mathcal{M}, i \rangle \models (\phi_1 \,\mathcal{U}\, \phi_2)$.
- When $\mathcal{M}$ is an infinite structure with a final loop, we can consider again two situations depending on stage i to be before or inside the loop.
  If stage i is inside the loop then, by construction of the algorithm, the formula ϕ_2 belongs to the loop where, until this moment, always is ϕ_1. It yields, again by induction, to $\langle \mathcal{M}, i \rangle \models (\phi_1 \,\mathcal{U}\, \phi_2)$.

  If stage i is before the loop and the formula ϕ_2 does not belong to some stage before the loop, then there must exist an eventuality $(\Delta \wedge \phi_1) \,\mathcal{U}\, \phi_2$ in the first node of stage j', for some (possibly empty) conjunction Δ of contexts. Besides, $\phi_1 \in_s R$ for every s in $i \dots (j'-1)$. Now the previous situation can be applied to $(\Delta \wedge \phi_1) \,\mathcal{U}\, \phi_2$ since the stage j' is inside the loop. Then $\langle \mathcal{M}, j' \rangle \models ((\Delta \wedge \phi_1) \,\mathcal{U}\, \phi_2)$ which implies that $\langle \mathcal{M}, i \rangle \models (\phi_1 \,\mathcal{U}\, \phi_2)$. □

6 Conclusions and Further Work

The development of automated deduction systems for propositional temporal logic has followed two main proof-theoretical approaches: tableaux [10] and resolution [2]. We have focused here on the first field, introducing a new method for semantic tableaux. While most of the previous decision algorithms for $PLTL$ have been presented as two-phase procedures:

– A tableau procedure that creates a graph.

– A procedure that checks whether the graph fulfills all eventualities.

Here, we have presented a tableau method where derivations result in tree structures rather than general graphs. Consequently, our method has the following two main advantages:

First, it avoids the second phase which requires the creation of the graph. Since it involves only the first phase, the procedure stops as soon as a model is detected, thus avoiding the construction of the complete tableau.

Secondly, each application of a rule in the tableaux is indeed an application of the corresponding rule in a new sequent calculus called $\mathcal{FC}$ [3]. Given any valid propositional temporal formula, a refutational proof in $\mathcal{FC}$ can be directly built from the closed tableau obtained by our method.

With respect to implementation, we have built a first prototype (in Java) that allows us to see, step by step, the construction of the tableau. When the initial formula is unsatisfiable, the running time of our algorithm can be high. In fact, the Proposition 3.4 gives an upper bound on the time complexity (worst case) of our method. But in practice, although it is still in test phase, its behaviour appears to be good enough. We are also working on the mechanization of the calculus $\mathcal{FC}$ in the generic proof-assistant Isabelle (cf. `http://isabelle.in.tum.de`) in order to allow the interactive formalization of $\mathcal{FC}$-proofs for temporal properties.

Additionally, we also plan to work on applying this new approach to make resolution. Up to now, the best resolution methods need to incorporate rules that only can be implemented using an invariant formula [2]. As in the tableaux case, we believe that the use of $\mathcal{FC}$ could improve the known resolution algorithms.

References

[1] M. Ben-Ari, *Mathematical Logic for Computer Science*, Springer, second edition, (2001).

[2] M. Fisher, *A Resolution Method for Temporal Logic*, IJCAI, (1991), 99–104.

[3] J. Gaintzarain, M. Hermo, P. Lucio, M. Navarro and F. Orejas, *A Cut-Free and Invariant-Free Sequent Calculus for PLTL*, 21 International Workshop CSL 2007. LNCS 4646. Springer-Verlag, (2007), 481–495.

[4] Y. Kesten, Z. Manna, H. McGuire and A. Pnueli, *A Decision Algorithm for Full Propositional Temporal Logic*, CAV'93: Proceedings of the 5th International Conference on Computer Aided Verification, Springer-Verlag, (1993), 97–109.

[5] O. Lichtenstein and A. Pnueli, *Propositional Temporal Logics: Decidability and Completeness*, Logic Journal of the IGPL, volume **8**, number 1, (2000).

[6] B. Paech, *Gentzen-Systems for Propositional Temporal Logics*, in CSL, (1988), 240–253. http://dblp.uni-trier.de.

[7] R. Pliuskevicius, *Investigation of Finitary Calculus for a Discrete Linear Time Logic by means of Infinitary Calculus*, Baltic Computer Science, (1991), 504–528, http://dblp.uni-trier.de.

[8] A. Szalas, *Temporal Logic of Programs: A Standard Approach*, Time and Logic. A Computational Approach. Bolc, L. and Szałas, A. UCL Press Ltd. (1995), 1–50.

[9] S. Schwendimann, *A New One-Pass Tableau Calculus for PLTL*, In Proceedings Tableaux 98. LNAI **1397**, (1998), 277–291.

[10] P. Wolper, *Temporal Logic Can Be More Expressive*, Information and Control, **56**, 1–2, (1983),72–99.