An Implementation of the Context-Based Tableau[1]

J. Gaintzarain, J. A. Hernandez and P. Lucio[2]

Department of Computer Languages and Systems
The University of the Basque Country
Spain

Abstract

In this paper we report on a prototype that implements a one-pass tableau method for Propositional Linear Temporal Logic (shortly, PLTL). It is well known that PLTL is decidable and that PLTL worst-case is PSPACE. The first tableau method for PLTL is due to P. Wolper in 1983 and it is a *two-pass method*. In the first pass, it generates an auxiliary graph. This graph is checked and (possibly) pruned in a second phase of the refutation procedure. The first *one-pass tableau method* for PLTL was introduced by S. Schwendimann in 1998, and it is based on checking, on-the-fly and branch-by-branch, the fulfillment of the eventuality formulas. The worst-case complexity of Wolper's method is EXPTIME, whereas Schwendimann's method is 2EXPTIME. However, according to published experiments, the implementation of the latter one-pass has outperformed the implementation of the earlier tableau method. The reason seems to be that 2EXPTIME worst-case rarely occurs in practice.
In this paper, we present a prototype that is a Haskell implementation of the tableau method for PLTL that was introduced by J. Gaintzarain et al. in 2007. This more recent method is also one-pass, but it does not perform any check of eventualities, instead it uses a mechanism to force the fulfillment of eventualities, if it is possible, or otherwise it forces a contradiction. The practical improvement of this kind of tools is a great challenge, since the worst-case complexity of this new one-pass method is also bounded to 2EXPTIME.

Keywords: Propositional Linear Temporal Logic, Tableaux, Satisfiability, Implementation, Haskell.

1 Introduction

Tableau systems are refutational proof methods that play a prominent role in automated reasoning. In addition for decidable logics, tableau methods

[1] This work has been partially supported by Spanish Project TIN2007-66523 and by Basque Project LoRea GIU07/35.

[2] Emails: jose.gaintzarain@ehu.es, jonan.h@gmail.com, paqui.lucio@ehu.es

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

serve as decision procedures for the satisfiability of (sets of) formulas. The propositional linear temporal logic, also named as PLTL, is decidable. The first tableau method for PLTL was introduced by P. Wolper in [24] and it is a *two-pass method*. In the first pass, it generates an auxiliary graph. This graph is checked and (possibly) pruned in a second phase that analyzes whether the so-called *eventualities* are fulfilled. An eventuality is a formula that asserts that something does eventually hold, e.g. $\Diamond\varphi$ (in words, φ eventually holds in the future) and $\chi\,\mathcal{U}\,\varphi$ (in words, χ holds from now until φ eventually holds). For example, to fulfill $\Diamond\varphi$ or $\chi\,\mathcal{U}\,\varphi$, φ must eventually be satisfied. Hence, any path in the graph that includes $\Diamond\varphi$ or $\chi\,\mathcal{U}\,\varphi$, but does not include φ, is pruned. At the end, an empty graph means unsatisfiability. Since Wolper's seminal paper [24], several authors (e.g. [11,2,20]) have found inspiration in Wolper's tableau to design tableau methods for different temporal and modal logics. [3] In particular, Wolper's two-pass tableau has been used in the development of decision procedures or proof techniques for logics that extends PLTL to some decidable fragment of the first-order temporal logic (e.g.[19]), or to the branching case or with other features, such as agents, knowledge, etc (e.g. [8]). Regarding implementations of this approach, we are aware of the Janssen's procedure ([14]) ("satisfiability" function in the PLTL module of the LogicsWorkbench Version 1.1) and of the McGuire et al. procedure ([18]) (inside the STeP system [21]).

The first one-pass tableau method for PLTL was developed in [22] and it avoids the second pass by adding extra information to the nodes in the tableau. Some of this information must be synthesized bottom-up and it is needed because the fulfillment of an eventuality in a single branch depends on the other branches. Hence, it carries out an on-the-fly checking of the fulfillment of every eventuality in every branch. Following [10], the method presented in [22] will be called as the *on-the-fly method*. An implementation of the on-the-fly method is incorporated as the "model" function in the PLTL module of the Logics Workbench Version 1.1. This on-the-fly tableau method has been successfully applied to other logics such as e.g. CTL ([1]) and PDL ([10]).

More recently, a new one-pass tableau method was introduced in [5] (see also [6]) that is not based on the on-the-fly check of eventualities. Instead, it is based on the fact that if a set of formulas $\Delta \cup \{\varphi\,\mathcal{U}\,\psi\}$ is satisfiable, there must exist a model $\mathcal{M}$ (with states $s_0, s_1 \ldots$) that is minimal in the following sense:

$$\mathcal{M} \textit{ satisfies either } \Delta \cup \{\psi\} \textit{ or } \Delta \cup \{\varphi, \circ((\varphi \wedge \neg\Delta)\,\mathcal{U}\,\psi)\}$$

In other words, in a minimal model such that $s_0 \models \neg\psi$, the so-called *context* Δ cannot be satisfied from the state s_1 until the state where ψ is true. In order to trust the above fact, consider a model $\mathcal{M}'$ with states $s'_0, s'_1 \ldots$

[3] The interested reader is referred to [9] for a good survey.

such that $s'_0 \models \neg\psi$ and $s'_j \models \psi$ for some $j \geq 1$, but there are some states between s'_1 and s'_j that satisfy Δ, namely $s'_{k_1}, \ldots, s'_{k_n}$. Then, let k be the greatest $j \in \{k_1, \ldots, k_n\}$ such that $s'_j \models \Delta$. Then, the structure given by $s'_k, s'_{k+1}, \ldots$ is also a model of $\Delta \cup \{\varphi \mathcal{U} \psi\}$ that is minimal in the above sense. On the basis of this idea, in [6] a new kind of temporal deduction was proposed as two dual systems of tableaux and sequents, respectively. Regarding tableaux, the crucial rule for getting a one-pass method is the one related to minimal models, which allows to split a branch containing a node labelled by $\Delta \cup \{\varphi \mathcal{U} \psi\}$ into two branches respectively labelled by $\Delta \cup \{\psi\}$ or $\Delta \cup \{\varphi, \circ((\varphi \wedge \neg\Delta) \mathcal{U} \psi)\}$. By means of this context-dependent rule, and provided that the number of possible contexts $\Delta_0, \Delta_1, \ldots$ is finite, the fulfillment of ψ cannot be indefinitely postponed, without getting a contradiction. We mean that, by keeping the succesive formulas of the form $(\varphi \wedge \neg\Delta_0 \wedge \ldots \wedge \neg\Delta_n) \mathcal{U} \psi$ as designated for the application of this crucial rule, the process should reach a node whose context is one –namely Δ_j– of the $\Delta_0, \ldots, \Delta_n$. Hence, to satisfy $\Delta_j \cup \{(\varphi \wedge \neg\Delta_0 \wedge \ldots \wedge \neg\Delta_n) \mathcal{U} \psi\}$, either $\Delta_j \cup \{\psi\}$ or $\Delta_j \cup \{\varphi \wedge \neg\Delta_0 \wedge \ldots \wedge \neg\Delta_n\}$ must be satisfied. But the second set of formulas is clearly unsatisfiable since $1 \leq j \leq n$. Due to the fact that contexts play the just explained significant role in this method, we call it the *context-based tableau method*.

It is well known (since [23]) that the worst-case complexity for PLTL is PSPACE. The two-pass method works in EXPTIME, hence it is optimal, while the worst-case complexity of both –on-the-fly and context-based– one-pass methods is 2EXPTIME. However, the practical performance of tableau methods for PLTL do not comply with the worst-case complexity results. Indeed, the experiments in [12] and [7] showed that the on-the-fly method outperforms the two-pass method. We conjecture that this phenomenon is due to the fact that the 2EXPTIME worst-case behaviour rarely arises for both –on-the-fly and context-based– methods. A non-experimental assessment on worst-case probability of the three methods seems to be very interesting, but not easy. In this paper, we provide a system description of our implementation of the context-based method in Haskell. This is the version 1.0 of the system TTM that is available at http://www.sc.ehu.es/jiwlucap/TTM.html. Some work remain to be done on improving this prototype and on experimental analysis for comparing our context-based method with other decision methods for PLTL, in particular with both the two-pass and the on-the-fly tableau method.

Outline. To make the paper self-contained, it begins with short to introductions the logic PLTL (Section 2) and to the context-based tableau method (Sections 3). For more details the reader is referred to [5,6]. The description of our prototype system is given in Section 4 where we report on the system architecture and also on some Haskell implementation details. Section 5 is de-

voted to the functionalities that the prototype provides to the user. Finally, in Section 6, an overview of our project's status and future direction is given.

2 Basics of PLTL

PLTL-formulas are built using propositional variables ($p, q, \ldots$), classical connectives $\neg$ and $\wedge$, and the temporal connectives $\circ$ (for next) and $\mathcal{U}$ (for until). We also use the constant proposition $\mathbf{F}$ for falsehood. Other connectives are defined in terms of the previous ones: $\mathbf{T} \equiv \neg\mathbf{F}$, $\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$, $\varphi \mathcal{R} \psi \equiv \neg(\neg\varphi \mathcal{U} \neg\psi)$, $\Diamond\varphi \equiv \mathbf{T}\mathcal{U}\varphi$, $\Box\varphi \equiv \neg\Diamond\neg\varphi$. Note that $\Box\varphi \equiv \mathbf{F}\mathcal{R}\varphi$. The defined connectives will be used as abbreviations for readability. PLTL-formulas of the form $\varphi\mathcal{U}\psi$ and $\Diamond\varphi$ are called *eventualities*.
Formally, a PLTL-*structure* $\mathcal{M}$ is a pair $(S_\mathcal{M}, V_\mathcal{M})$ such that $S_\mathcal{M}$ is a denumerable sequence of states $s_0, s_1, s_2, \ldots$ and $V_\mathcal{M}$ is a map $V_\mathcal{M} : S_\mathcal{M} \to 2^{\mathsf{Prop}}$. Intuitively, $V_\mathcal{M}(s)$ specifies which atomic propositions are (necessarily) true in the state s.
The formal semantics of PLTL-formulas is given by the truth of a formula φ in the state s_j of a PLTL-structure $\mathcal{M}$ is denoted by $\langle\mathcal{M}, s_j\rangle \models \varphi$ and is inductively defined as follows:

- $\langle\mathcal{M}, s_j\rangle \not\models \mathbf{F}$
- $\langle\mathcal{M}, s_j\rangle \models p$ iff $p \in V_\mathcal{M}(s_j)$ for $p \in \mathsf{Prop}$
- $\langle\mathcal{M}, s_j\rangle \models \neg\varphi$ iff $\langle\mathcal{M}, s_j\rangle \not\models \varphi$
- $\langle\mathcal{M}, s_j\rangle \models \varphi \wedge \psi$ iff $\langle\mathcal{M}, s_j\rangle \models \varphi$ and $\langle\mathcal{M}, s_j\rangle \models \psi$
- $\langle\mathcal{M}, s_j\rangle \models \circ\varphi$ iff $\langle\mathcal{M}, s_{j+1}\rangle \models \varphi$
- $\langle\mathcal{M}, s_j\rangle \models \varphi\mathcal{U}\psi$ iff there exists $k \geq j$ such that $\langle\mathcal{M}, s_k\rangle \models \psi$ and for every $j \leq i < k$ it holds $\langle\mathcal{M}, s_i\rangle \models \varphi$.

This semantics is extended to the defined connectives using each definition.
From the semantical point of view, (finite) sets of formulas are equivalent to their conjunction. Given a set $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ we will use $\bigwedge\Phi$ to denote $\varphi_1 \wedge \ldots \wedge \varphi_n$ and $\neg\Phi$ to denote the formula $\neg\bigwedge\Phi$ or equivalently $(\neg\varphi_1 \vee \ldots \vee \neg\varphi_n)$. In particular, when Φ is empty, $\neg\Phi$ and $\bigwedge\Phi$ are the constants $\mathbf{F}$ and $\mathbf{T}$, respectively.
A PLTL-structure $\mathcal{M}$ is *cyclic* if its (infinite) sequence of states $S_\mathcal{M}$ is a path over a cyclic sequence of states. Any satisfiable PLTL-formula has a cyclic model ([3]). Indeed, the system TTM finds (and returns to the user) a cyclic model for every satisfiable input.

Rule	α	$A(\alpha)$
$(\neg\neg)$	$\neg\neg\varphi$	$\{\varphi\}$
$(\wedge)$	$\varphi \wedge \psi$	$\{\varphi, \psi\}$
$(\neg\circ)$	$\neg\circ\varphi$	$\{\circ\neg\varphi\}$

Rule	β	$B_1(\beta)$	$B_2(\beta)$
$(\neg\wedge)$	$\neg(\varphi \wedge \psi)$	$\{\neg\varphi\}$	$\{\neg\psi\}$
$(\neg\,\mathcal{U}\,)$	$\neg(\varphi\,\mathcal{U}\,\psi)$	$\{\neg\varphi, \neg\psi\}$	$\{\varphi, \neg\psi, \neg\circ(\varphi\,\mathcal{U}\,\psi)\}$
β-$(\mathcal{U}\,)$	$\varphi\,\mathcal{U}\,\psi$	$\{\psi\}$	$\{\varphi, \neg\psi, \circ(\varphi\,\mathcal{U}\,\psi)\}$

Rule	κ	$C_1(\kappa)$	$C_2(\kappa, \Delta)$
κ-$(\mathcal{U}\,)$	$\varphi\,\mathcal{U}\,\psi$	$\{\psi\}$	$\{\varphi, \neg\psi, \circ((\varphi \wedge \neg\Delta)\,\mathcal{U}\,\psi)\}$

Fig. 1. The TTM-rules

Rule	α	$A(\alpha)$
$(\Box)$	$\Box\varphi$	$\{\varphi, \circ\Box\varphi\}$

Rule	β	$B_1(\beta)$	$B_2(\beta)$
$(\vee)$	$\varphi \vee \psi$	$\{\varphi\}$	$\{\psi\}$
$(\,\mathcal{R}\,)$	$\varphi\,\mathcal{R}\,\psi$	$\{\varphi, \psi\}$	$\{\neg\varphi, \psi, \circ(\varphi\,\mathcal{R}\,\psi)\}$
β-$(\Diamond)$	$\Diamond\varphi$	$\{\varphi\}$	$\{\neg\varphi, \circ\Diamond\varphi\}$

Rule	κ	$C_1(\kappa)$	$C_2(\kappa, \Delta)$
κ-$(\Diamond)$	$\Diamond\varphi$	$\{\varphi\}$	$\{\neg\varphi, \circ((\neg\Delta)\,\mathcal{U}\,\varphi)\}$

Fig. 2. The derived TTM-rules

3 The Context-Based Tableau Method

In this section we recall the context-based tableau method that was introduced in [5,6] and give a complete example of tableau construction.
A tableau for a finite set of formulas Φ is a tree-like structure where each node n is labelled with a set of formulas $L(n)$. The root is labelled with the

set Φ whose satisfiability we wish to check. The children of a node n are obtained by applying one of the rules to one of the formulas in $L(n)$. Nodes are organized in branches, so that the rules serve to either enlarge the branch (with one new child) or split the branch with two new children. A tableau rule is applied to a set of formulas $L(n)$ labelling a node n that is the last node of a branch. Each rule application requires the selection of a designated formula from $L(n)$. By means of the α-, β- and κ-rules in Fig. 1, each formula named as α is decomposed in a unique set, called $A(\alpha)$, and any formula β is decomposed into two constituent sets $B_1(\beta)$ and $B_2(\beta)$. By application of an α-rule, we enlarge the branch with a node with label $(L(n) \setminus \{\alpha\}) \cup A(\alpha)$. A β-rule application splits the branch with two nodes with respective labels $(L(n) \setminus \{\beta\}) \cup B_1(\beta)$ and $(L(n) \setminus \{\beta\}) \cup B_2(\beta)$. In other words, the application scheme for the α- and β-rules are

$$\begin{array}{c} \Delta, \alpha \\ | \\ \Delta, A(\alpha) \end{array} \qquad\qquad \begin{array}{c} \Delta, \beta \\ \diagup \quad \diagdown \\ \Delta, B_1(\beta) \quad \Delta, B_2(\beta) \end{array}$$

For example, the following is a pre-tableau (it is not completed) that first uses β-($\mathcal{U}$) and then ($\wedge$) in the right branch.

$$\begin{array}{c} s, (p \wedge q)\,\mathcal{U}\, r \\ \diagup \quad\quad \diagdown \\ s, r \quad s, p \wedge q, \neg r, \circ((p \wedge q)\,\mathcal{U}\, r) \\ \qquad\qquad\qquad | \\ \qquad\qquad\quad s, p, q, \neg r, \circ((p \wedge q)\,\mathcal{U}\, r) \end{array}$$

The κ-rules work similar to the β-rules for splitting branches, but the second constituent $C_2(\kappa, \Delta)$, depends not only on the designated κ-formula, but also on the context. We call the *context* to Δ, that is the remaining formulas in the node of the designated eventuality.

$$\begin{array}{c} \Delta, \kappa \\ \diagup \quad \diagdown \\ \Delta, C_1(\kappa) \qquad \Delta, C_2(\kappa, \Delta) \end{array}$$

For example, if we use in the above pre-tableau κ-($\mathcal{U}$) instead of β-($\mathcal{U}$) (where the context is $\{s\}$) the pre-tableau is

$$\begin{array}{c} s, (p \wedge q)\,\mathcal{U}\, r \\ \hline\hline \\ s, r \qquad s, p \wedge q, \neg r, \circ((p \wedge q \wedge \neg s)\,\mathcal{U}\, r) \\ \Big| \\ s, p, q, \neg r, \circ((p \wedge q \wedge \neg s)\,\mathcal{U}\, r) \end{array}$$

A formula is *elementary* whenever it is either a next-formula (i.e. a formula of the form $\circ\varphi$) or a literal (atom or negated atom). For instance $\{s, p, q, \neg r, \circ((p \wedge q \wedge \neg s)\,\mathcal{U}\, r)\}$ is a set of elementary formulas.
The rules in Fig. 1 are applied to nodes with at least one formula that is not elementary, For sets of elementary formulas, the TTM system includes the following rule that is called (unnext)

$$\begin{array}{c} \ell_1, \ldots, \ell_m, \circ\varphi_1, \ldots, \circ\varphi_n \\ \Big| \\ \varphi_1, \ldots, \varphi_n \end{array}$$

where the ℓ_i are literals and $m, n \geq 0$. For instance

$$\begin{array}{c} \{s, p, q, \neg r, \circ((p \wedge q \wedge \neg s)\,\mathcal{U}\, r)\} \\ \Big| \\ \{(p \wedge q \wedge \neg s)\,\mathcal{U}\, r\} \end{array}$$

Note that, the rule (unnext) simulates the jump from one state to the next one. Note also that for $n = 0$, the rule (unnext) produces the empty set, then (unnext) can only be applied to the empty set, that yields again the empty set. This is a especial kind of rear-cycle in a model.
There are also derived rules (see Fig. 2) for the defined connectives ($\vee$, $\Diamond$, $\Box$, $\mathcal{R}$) that are obtained from the basic rules in Fig. 1 by using the definition of each connective.
Tableaux are constructed with the aim of refuting the initial set of formulas. A node n is consistent iff $\mathbf{F} \notin L(n)$ and there is no φ such that $\{\varphi, \neg\varphi\} \subseteq L(n)$. Otherwise, n is *inconsistent*. When a branch contains an inconsistent node we say that it is *closed*. Any closed branch is trivially unsatisfiable. Branches that are not closed are said to be *open*.
The implemented algorithm consists in a systematic extension/splitting of branches using the TTM-rules. When no other rule can be applied, the rule (unnext) is used to jump to a new stage. The algorithm ends when either a

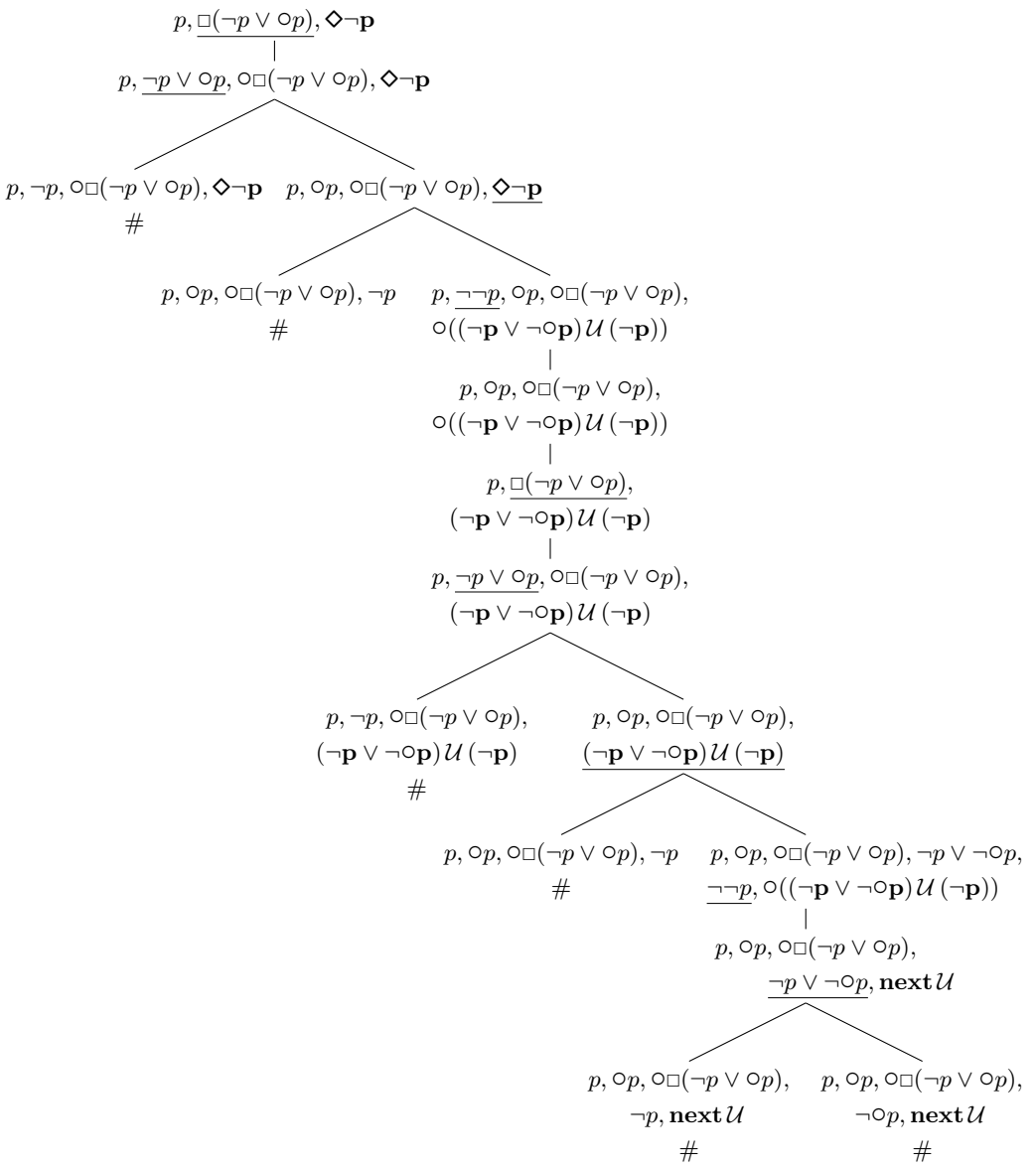

Fig. 3. A Closed Tableau

cyclic branch is detected or every branch is closed. In the former case, the model given by the open branch is returned as output. Otherwise, the output is a message stating that the input set is unsatisfiable.
Regarding the application of the TTM-rules, the algorithm keeps one selected eventuality (if there is any) to which the rule κ-($\mathcal{U}$) is applied. More precisely,

- Between each two applications of the rule (unnext) exactly one (if any) eventuality is marked as selected.
- The selection function is fair in the sense that no eventuality occurring in a branch could remain non-selected indefinitely.

- When the rule κ-($\mathcal{U}$) is applied to a node labelled by Δ and a designated $\varphi \mathcal{U} \psi$, the formula $\circ((\varphi \wedge \neg\Delta)\mathcal{U}\psi)$ –in the right branch– is elementary. Hence, it is not further decomposed in the current state. Then, $(\varphi \wedge \neg\Delta)\mathcal{U}\psi$ will become the selected eventuality after the application of (unnext).

The non-selected eventualities are decomposed using the rule β-($\mathcal{U}$) and for the remaining nonelementary formulas there is a unique rule to be applied.

Example 3.1 The figure 3 shows a closed tableau for the set of formulas $p, \Box(\neg p \vee \circ p), \Diamond\neg p$. The selected eventualities are shown in bold font. The tableau is obtained by the successive application of the following rules (each rule is applied to the underlined formula, excepting (unnext) that is applied to the whole set): ($\Box$), ($\vee$), κ-($\Diamond$), ($\neg\neg$), (unnext), ($\Box$), ($\vee$), κ-($\mathcal{U}$), ($\neg\neg$) and ($\vee$).
In the application of κ-($\Diamond$), the context Δ is formed by the formulas $\{p, \circ p, \circ\Box(\neg p \vee \circ p)\}$. However, the last formula has not been considered to write $\circ((\neg p \vee \neg\circ p)\mathcal{U}(\neg p))$ in the output of the rule κ-($\Diamond$). This is an optimization that was already explained in [6] and has been implemented in our prototype. The basic idea is the fact that when the context contains a formula stating that something always holds, the negation of this statement would never be satisfied. In the later application of κ-($\mathcal{U}$), the context is again the same set $\{p, \circ p, \circ\Box(\neg p \vee \circ p)\}$. Textually using the rule κ-($\mathcal{U}$), the formula $\circ(((\neg p \vee \neg\circ p) \wedge (\neg p \vee \neg\circ p))\mathcal{U}(\neg p))$ must be in its output. Our prototype is aware of context repetition, giving $\circ((\neg p \vee \neg\circ p)\mathcal{U}(\neg p))$ instead. This formula is called "`next`$\mathcal{U}$" in Fig. 3 due to the lack of space since the concrete eventuality is irrelevant to close the tableau.

4 System Description

The system TTM is implemented in Haskell ([16,15]) and compiled with Glasgow Haskell Compiler ([17]), resulting in an efficient cross-platform implementation packaged for Linux, Windows, and Mac OS. This prototype is available at http://www.sc.ehu.es/jiwlucap/TTM.html In this section we describe the architecture of the prototype TTM (version 1.0) and we also give some Haskell implementation details.

Regarding architecture, Fig. 4 shows a diagram with the packages and modules involved in the subsystems of the prototype TTM which have been built using the Haskell Cabal building and packaging system (see http://www.haskell.org/cabal/). The input layer –which is responsible of parsing the input set of PLTL-formulas into the Haskell internal representation– is shown on the left-hand part. When a syntactical error is found, the parser reports it to the user through the user interface. This layer is constructed

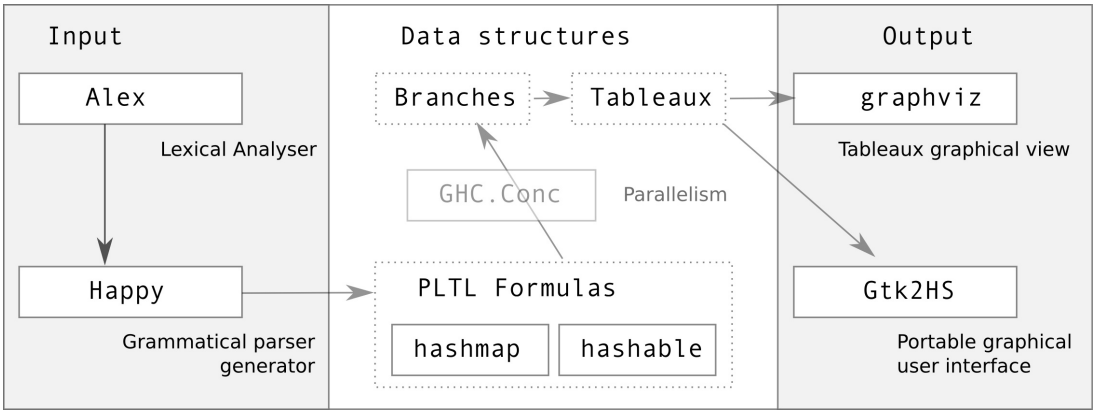

Fig. 4. The TTM System Architecture

with the aid of the *Alex* (http://www.haskell.org/alex/) and *Happy* packages (http://www.haskell.org/happy/).
The right-hand part of Fig. 4 represents the graphical user interface architecture. First, the *graphviz* package (http://projects.haskell.org/graphviz/) is used for visualizing graphically the output tableau that is built by using the rules and following the algorithm explained in the previous Section 3. A partial view of the graphical tableau that the user can generate is given in Fig. 5. Second, the Haskell binding to Gtk+, called *Gtk2Hs* (www.haskell.org/haskellwiki/Gtk2Hs), is used for constructing the cross-platform graphical user interface.
The core algorithm constructs a tableau, for the input set of PLTL-formulas.

```
data TableauNode = TableauNode {
                         markedFormulas      :: Set,
                         unmarkedFormulas    :: Set,
                         selection           :: [PLTLFormula],
                         eventualies         :: Set
                   }

type Branch      = [TableauNode]
type Tableau     = [Branch]
```

A tableau is represented in Haskell as a list of branches, where a branch is a list of nodes. Each node contains four items of information. The first is the set of formulas to which a rule has already been applied. We say they are marked. The second is the set of unmarked formulas of the node. The third is the list of selected eventualities, that can be empty of a singleton, whereas the fourth is the set of all the eventualities that still remain selectable in the branch to which the node belongs.

```
solveTableau :: Tableau -> Tableau
solveTableau []     = []
solveTableau (b:bs) = solveBranch ++ solveTableau bs
```

```
        where solveBranch = if isCycling b
                            then [b]
                            else if consistentBranch b
                                 then solveTableau (extendBranch b)
                                 else []
```

Following the method explained in the previous Section 3, the refutation stops as soon as the first open branch is found. A tableau branch is open whenever it represents a model and every model is cyclic. The model represented by this open branch is returned to the user. Hence, in the satisfiable case it avoids the computation of the remaining branches. Otherwise, the system must construct the whole closed tableau.

```
extendBranch :: Branch -> Tableau
extendBranch b
    | nonElementary == []
        = [b ++ [unnext lastNode]]
    | (not . null) selectedEv && (not . isNext . head) selectedEv
        = [b ++ [n] | n <- applyRule (head selectedEv) lastNode]
    | otherwise
        = [b ++ [n] | n <- applyRule (head nonElementary) lastNode]
      where lastNode          = last b
            selectedEv        = selection lastNode
            nonElementary     = nonElementaryFormulas lastNode
```

The construction of the tableau is carried out by extending the branches. To this end, if the last node is exclusively formed by elementary formulas, the rule (unnext) is used to obtain the node by which the branch is enlarged. Otherwise, a tableau rule must be applied to the last node. Depending on whether there is a selected eventuality that is not a formula of the form $\circ\varphi$ (called a next-formula) or not, the formula designated for the rule application is either the selected formula (the head of the list selectedEv) or the head of the list of all nonelementary formulas in the node.

```
applyRule :: PLTLFormula -> TableauNode -> [TableauNode]
applyRule formula node
 = case formula of
     (And p q)  -> newNodes [[p,q]]
     (Or p q)   -> newNodes [[p],[q]]
     (Eventually p)
         -> if [Eventually p] == selectedEv
               then let newEv = negateContext 'U' p
                    in  newNodes_rule2 ([p],[Not p,Next newEv],Next newEv)
               else newNodes [[p],[Not p,Next(Eventually p)]]
        .
        .
        .
```

The application of a tableau rule to a designated formula an a node depends on the designated formula and gives the list of the new nodes that must be used to split/enlarge the branch. For formulas of the form (And p q) and (Or p q) the function newNodes respectively constructs the nodes corresponding to the sets [[p,q]] and [[p],[q]]. However for eventualities (e.g. $\Diamond p$ in the code above)

the calculation of the new nodes depends on whether the designated formula is the selected one or not, to construct the new nodes that correspond to the application of the rule κ-($\Diamond$) or β-($\Diamond$), respectively. The function newNodes_rule2 is identical to newNodes excepting that the selected eventuality in the new nodes remains unchanged by the latter whereas the former changes the selection to be the eventuality newEv (which is the eventuality constructed in the rule κ-($\Diamond$)).

In the early stages of the prototype, it performed competitively in spite of the lack of optimization. We believe that Haskell laziness has been a great advantage, since it was able to construct tableaux with several thousands of closed branches in a few minutes and with a low space complexity. This is due to the lazy evaluation (of Haskell) that keeps exactly one tableau branch at a time. Our first prototype showed that a huge explosion of identical branches occurs very frequently. The most significant optimization techniques used in the current version of the prototype is devoted to avoid the re-calculation of closed sub-tableaux (or sub-trees). The optimization consists in storing unsatisfiable sets of formulas, which are the root of closed sub-tableaux that have been already constructed. Consequently, we refactor data structures for efficiently loading and comparing branches, which are sets of sets of formulas. Hence, the data structures used to represent the sets of PLTL-formulas play an important role in the overall performance. At each step of the tableau construction, multiple set operations take place like e.g. checking for inconsistencies or adding the constituents of a formula after a rule application. Two representations were mainly taken into consideration (the packages for both can be found in http://hackage.haskell.org/packages/archive/containers/latest/doc/html):

- The package Data.Set (in /Data-Set.html) –which is based on a balanced binary tree– requires the base type only to be ordered and comparable [4] that is straight forward on a Haskell algebraic data type.
- The package Hashmap (in /Data-HashSet.html) provides a *persistent immutable* set implemented over a hashmap (Data.IntMap.IntMap). The data type requires, besides being comparable, a hashing operator which maps any object to an integer. The hashing operator uses a hashing composition function to combine the hashes of complex objects.

The current version 1.0 of ttm uses the package Hashmap to construct hash tables that represent sets of PLTL-formulas. This choice is due to the fact that hash tables provide optimal set operations –that are performed very frequently in ttm–, while allowing optimal sequential access to the items of a set. Indeed, using hash tables, we represent sets of PLTL-formulas in a hierarchical way that allows efficient access to e.g. all the elementary formulas in the set, or

[4] For proper PLTL-formula comparison our system transforms the formulas to a canonical form.

all the formulas whose outermost connective is ○, or all the eventualities, etc. To gain efficiency in this kind of access to some particular subset of a set of PLTL-formulas is crucial for the context-based tableau method explained in Section 3. Furthermore, recent developments of Haskell on implementation of hash tables allow efficient updating that does not copy the entire structure, but only the changed path. Therefore, in TTM this can be used to share data across different branches.

5 The ttm Functionality

This section is devoted to the usage of the TTM tool. The system TTM has been integrated with two different interfaces for different user purposes. The package TTMGUI provides the user with a *Graphical User Interface*, whereas the package TTMCLI is a *Command Line Interface* for TTM executions.

The graphical interface TTMGUI consists of three main sets of components,

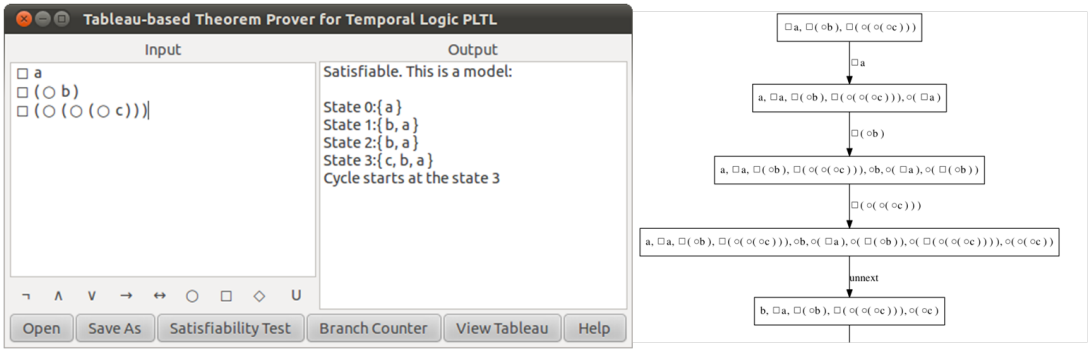

Fig. 5. The TTM Graphical User Interface

which respectively provide a formula editor, an output view, and a set of buttons that enables the user to obtain different outputs from the input formulas. The *formula editor* makes a syntactical check of the PLTL formulas entered by the user, helps inserting the PLTL symbols encoded in Unicode character set (UTF-8 [5]), and allows to load set of formulas into text files and also to open previously loaded files. The interface includes buttons for the following subset of the TTM capabilities: the satisfiability test (including model calculation when the test results is positive), the counter of the number of generated branches, and the view of the whole tableau. The *output view* serves to display the computed results in pretty and useful ways. Fig. 5 depicts the image of TTMGUI after entering the input in the left-hand and then clicking the buttons "Satisfiability Test" and "View Tableau". For space reasons, in the

[5] The UCS Transformation Format - 8 is a multibyte character encoding for Unicode backward-compatible with ASCII.

left-hand of Fig. 5 we show a partial view of the whole tableau.

Fig. 6. The TTM Command Line Interface

The command line interface TTMCLI provides an interactive mode and a non-interactive mode. The interactive mode provides a prompt where the user can insert a sequence of commands. However, the non-interactive mode only runs a single command specified through the available options. The interface TTMCLI was a requirement to create a set of tools which help the testing and analysis of the behavior of the different optimizations of the TTM system. There are commands for testing the satisfiability (getting a model, if possible) of a PLTL formula (*model command*), for computing the number of branches (*branches command*), for viewing the whole tableau (*tableau-view command*), for exporting the tableau in dot language[6] (*tableau-dot command*), for dynamically changing the behavior of TTM e.g. changing the data structures used to represent the sets of PLTL formulas (e.g. binary trees: Data.Set or hash maps: package hashmap), and for disabling/enabling some optimizations like branch repetition removal and formula comparison using negation normal form. Fig. 6 shows the result of testing the satisfiability of the input set in file `sat2.txt`, which contains the set in the left-hand side of Fig. 5.

6 Conclusion and Future Work

We have presented the version 1.0 of the system TTM –which is available at http://www.sc.ehu.es/jiwlucap/TTM.html– that implements the context-based tableau method for PLTL introduced in [5,6].
Some refactoring of the data structures used in the very preliminary version of the prototype have already proved to be valuable for efficiency. However, more experimentation and comparison with other implementations remain to

[6] DOT is a plain text graph description language

be done. In a near future, we plan to use the ideas of "scientific benchmarking" proposed in [12] to compare the system TTM with other implementations of decision procedures for PLTL. Firstly, we shall compare TTM with the two tableau-based implementations included in PLTL module of the Logics Workbench Version 1.1 (http://www.lwb.unibe.ch/): the two-pass tableau of Janssen ([14]) and the on-the-fly tableau of Schwendimann ([22]). Secondly, we also plan to experiment and compare with the system TRP++ ([13]) that is also a PLTL theorem prover that it is not tableau-based but it implements the temporal resolution procedure introduced by Fisher ([4]). We hope that all these experiments would test our implementation of the context-based method and would help us to identify potential improvements whose implementation would result in a measurably better prototype. We plan to be ready to present some profiling analysis and benchmarks at the workshop.
Additionally, we plan to experiment with the parallel computation of tableau branches. Concretely, we are considering the GHC parallelization package (http://hackage.haskell.org/packages/archive/parallel/latest/doc/html/Control-Parallel.html) as a good tool for this end.

References

[1] P. Abate, R. Goré, and F. Widmann. One-pass tableaux for computation tree logic. In *Proceedings of the 14th international conference on Logic for programming, artificial intelligence and reasoning*, LPAR'07, pages 32–46, Berlin, Heidelberg, 2007. Springer-Verlag.

[2] B. Banieqbal and H. Barringer. Temporal logic with fixed points. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification, Altrincham, UK, April 8-10, 1987, Proceedings*, volume 398 of *Lecture Notes in Computer Science*, pages 62–74. Springer, 1987.

[3] A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. In *Highly Dependable Software*, volume 58 of *Advances in Computers*. Academic Press, 2003.

[4] M. Fisher. A resolution method for temporal logic. In John Mylopoulos and Raymond Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI), Sydney, Australia.*, pages 99–104, 1991.

[5] J. Gaintzarain, M. Hermo, P. Lucio, and M. Navarro. Systematic semantic tableaux for PLTL. In E. Pimentel, editor, *Proceedings of the Seventh Spanish Conference on Programming and Computer Languages (PROLE 2007), Selected Papers*, volume 206 of *Electronic Notes in Theoretical Computer Science*, pages 59–73, Amsterdam, The Netherlands, The Netherlands, 2008. Elsevier Science Publishers B. V.

[6] J. Gaintzarain, M. Hermo, P. Lucio, M. Navarro, and F. Orejas. Dual systems of tableaux and sequents for PLTL. *Journal of Logic and Algebraic Programming*, 78(8):701–722, 2009.

[7] V. Goranko, A. Kyrilov, and D. Shkatov. Tableau tool for testing satisfiability in ltl: Implementation and experimental analysis. *Electr. Notes Theor. Comput. Sci.*, 262:113–125, 2010.

[8] V. Goranko and D. Shkatov. Tableau-based decision procedure for full coalitional multiagent temporal-epistemic logic of linear time. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 969–976, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[9] R. Gore. *Tableau methods for modal and temporal logics*, pages 297–396. Kluwer Academic Publishers, 1999.

[10] R. Goré and F. Widmann. An optimal on-the-fly tableau-based decision procedure for pdl-satisfiability. In *Proceedings of the 22nd International Conference on Automated Deduction*, CADE-22, pages 437–452, Berlin, Heidelberg, 2009. Springer-Verlag.

[11] G. D. Gough. *Decision Procedures for Temporal Logic.* Master's thesis. Department of Computer Science, University of Manchester, England, 1984.

[12] U. Hustadt and R. A. Schmidt. Scientific benchmarking with temporal logic decision procedures. In D. Fensel, F. Giunchiglia, D. McGuinness, and M.-A. Williams, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Eighth International Conference (KR'2002)*, pages 533–544. Morgan Kaufmann, 2002.

[13] Ullrich Hustadt and Boris Konev. Trp++2.0: A temporal resolution prover. In Franz Baader, editor, *Automated Deduction - CADE-19, 19th International Conference on Automated Deduction Miami Beach, FL, USA, July 28 - August 2, 2003, Proceedings*, volume 2741 of *Lecture Notes in Computer Science*, pages 274–278. Springer, 2003.

[14] G. Janssen. Logics for digital circuit verification - theory, algorithms, and applications, phd thesis, eindhoven university of technology, the netherland, 1999.

[15] S. Peyton Jones, editor. *Haskell 98 Language and Libraries: The Revised Report.* http://haskell.org/, September 2002.

[16] S. Peyton Jones and J. Hughes, editors. *Report on the programming language Haskell 98. Available from the Haskell homepage: http://www.haskell.org.* September 1999.

[17] Simon L. Peyton Jones, Cordy Hall, Kevin Hammond, Jones Cordy, Hall Kevin, Will Partain, and Phil Wadler. The glasgow haskell compiler: a technical overview, 1992.

[18] Y. Kesten, Z. Manna, H. Mcguire, and A. Pnueli. A decision algorithm for full propositional temporal logic. In *Proc. CAV'93, volume 697 of LNCS*, pages 97–109. Springer-Verlag, 1993.

[19] R. Kontchakov, C. Lutz, F. Wolter, and M. Zakharyaschev. Temporalising tableaux. *STUDIA LOGICA*, 76(1):91–134, 2004.

[20] O. Lichtenstein and A. Pnueli. Propositional temporal logics: Decidability and completeness. *Logic Journal of the IGPL*, 8(1), 2000.

[21] Z. Manna, A. Anuchitanukul, N. Bjorner, A. Browne, E. Chang, M. Colon, L. de Alfaro, H. Devarajan, H. Sipma, and T. Uribe. Step: The stanford temporal prover. technical report stan-cs-tr-94-1518, computer science department, stanford university, 1994.

[22] S. Schwendimann. A new one-pass tableau calculus for PLTL. In *Analytic Tableaux and Related Methods*, volume 1397 of *Lecture Notes in Computer Science*, pages 277–292, 1998.

[23] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.

[24] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1–2):72–99, 1983.