A New One-Pass Tableau Calculus for PLTL

Stefan Schwendimann

Institut für Informatik und
angewandte Mathematik
University of Berne
Neubrückstr.10
CH-3012 Bern
E-mail: schwendi@iam.unibe.ch
Phone: +41 31 6313317

Abstract. The paper presents a one-pass tableau calculus PLTL_T for the propositional linear time logic PLTL. The calculus is correct and complete and unlike in previous decision methods, there is no second phase that checks for the fulfillment of the so-called eventuality formulae. This second phase is performed *locally* and is incorporated into the rules of the calculus. Derivations in PLTL_T are cyclic trees rather than cyclic graphs. When used as a basis for a decision procedure, it has the advantage that only one branch needs to be kept in memory at any one time. It may thus be a suitable starting point for the development of a parallel decision method for PLTL.

1 Introduction

Temporal logic has proved to be a useful formalism for reasoning about execution sequences of programs. It can be employed to formulate and verify properties of concurrent programs, protocols and hardware (see for instance [1], [13], [14]). A prominent variant is the propositional linear time logic PLTL where the decision problem is known to be PSPACE-complete [15]. In most of the previous publications the decision algorithm itself has been presented as a 2-phase procedure:

1. A tableau procedure that creates a graph.
2. A procedure that checks whether the graph fulfills all eventuality formulae.

The second phase usually leads to an analysis of the strongly connected components (SCC) of the graph (see e.g. [16]). Typical descriptions of this 2-phase method can be found in [17] and [9] where, in both cases, the second phase is not treated formally.

The tableau method presented in [12] is claimed to be incremental, where 'incremental' means that only reachable nodes are created (this is also true for [17] and [9]). However, it is essentially still a 2-phase procedure. The focus there is on providing a refined method for linear temporal logic with past time operators.

The above methods can treat the verification problem directly as a logical implication '*spec* $\rightarrow$ *prop*', where *spec* is the PLTL formula representing a specification and *prop* the formula representing a property to be verified. The essence of the problem is to show the validity of this implication in PLTL.

An alternative approach uses state-based methods (also referred to as 'model checking'). One possibility is to translate both the specification (e.g. of a protocol) and the negation of the property into labeled generalized Büchi automata, where the property automaton is also generated by a tableau-like procedure. A second phase then checks whether the language accepted by the synchronous product of the two automata is empty. Once again, in general, this involves an SCC analysis. In [7] it is claimed that the check for emptiness can be done 'on-the-fly' during the generation of the product: the tableau-like procedure builds the property graph in a depth-first manner choosing only successors that 'match' the current state of the protocol. Validity can also be checked using this method. However, it is not clear from the description whether the procedure remains 'on-the-fly' when there is no protocol to 'match'. In [2] it is shown how a generalized Büchi automaton can be transformed into a classical Büchi automaton for which the emptiness check reduces to a simple cycle detection scheme. So in the area of state-based methods similar attempts have been made to intermix the two phases and to avoid a standard SCC analysis.

Here we present a one-pass tableau calculus which checks locally, on-the-fly, for the fulfillment of eventuality formulae on a branch-by-branch basis. No second phase is required. It can also be used for an incremental depth-first search where only reachable states are created. Derivations in this calculus result in (cyclic) tree-like structures rather than general graphs. Thus, the analysis of strongly connected components reduces to the detection of 'isolated subtrees', a task which is very simple and which can therefore be incorporated easily into the calculus. The new aspects basically consist of:

1. A branch-based loop check that ensures termination.
2. A part that synthesizes the essential information gleaned from expanding the subtrees of a node.

The 2-phase methods require the creation of a fully expanded tableau, which is often exponential in the size of the initial formula. Since our method involves only one pass and is complete, we can stop as soon as a (counter-) model is detected, thus, (sometimes) avoiding a fully expanded tableau. A further advantage is that only one branch of the derivation tree needs to be considered at any stage. Therefore, the calculus PLTL_T is a natural analogue of the tableau and Gentzen-style sequent calculi for various modal logics, for instance K, KT and S4 (see e.g. [6], [8], [3]), where derivations are also trees, where it is always sufficient to consider one branch at any one time and where a check for loops is sometimes required to guarantee termination (see e.g. [11]).

While the two phases of the previous methods are an obstacle for parallelization, the branch-by-branch treatment offers natural possibilities for concurrent search. Of course, at the end, the resultant parts would need to be combined, but until then the processors could work independently on different subtrees without extra-communication.

There is of course a caveat. Since a naive derivation in PLTL_T essentially unfolds a graph into a tree, the run-time may be significantly higher, especially for examples where the graphs have (relatively) few nodes and many edges. Clearly, the calculus must be applied in combination with suitable pruning and caching techniques. Algorithmic aspects, however, are beyond the scope of this paper. We will focus on the new definitions and the key lemmata and theorems. Simpler observations are stated as propositions without proofs.

2 Syntax

In the following we deal with an extension $\mathcal{L}$ of the language for classical propositional logic. It comprises: 1. Countably many propositional variables $p_0, p_1, \ldots$. 2. The propositional constants true and false. 3. The connectives $\neg$, $\wedge$, $\vee$, X (neXt time), F (sometime), G (generally), $\mathcal{U}$ (until), and $\mathcal{B}$ (before). As auxiliary symbols we have parentheses and commas. The formulae of $\mathcal{L}$ are inductively defined: 1. The propositional variables and constants are formulae. 2. If A and C are formulae, then $(\neg A)$, $(\mathsf{X}\, A)$, $(\mathsf{F}\, A)$, $(\mathsf{G}\, A)$, $(A \wedge C)$, $(A \vee C)$, $(A\, \mathcal{U}\, C)$, and $(A\, \mathcal{B}\, C)$ are formulae.

The set of propositional variables is denoted by Var and the set of all formulae by Fml. As metavariables for propositional variables we use P, Q, and as metavariables for formulae A, C, D, possibly with subscripts. Propositional variables are also called *positive literals*; if P is a propositional variable then $\neg P$ is a *negative literal*. As metavariable for positive literals we use P and as metavariable for literals M, possibly with subscripts. In order to increase readability, we omit outer parentheses and define the unary connectives to take precedence over all binary connectives. For example, we write $\mathsf{F}\,(p_7\, \mathcal{U}\, p_1) \wedge (p_0\, \mathcal{B}\, \neg \mathsf{X}\, p_1)$ for the formula $((\mathsf{F}\,(p_7\, \mathcal{U}\, p_1)) \wedge (p_0\, \mathcal{B}\, (\neg(\mathsf{X}\, p_1))))$.

3 Semantics

Definition 1. *A* PLTL-model *is a pair* $\langle S, L\rangle$, *where* S *is an infinite sequence of states* $(s_i)_{i \in \mathbb{N}} = s_0\, s_1 \ldots$ *and* $L : S \to \mathrm{Pow}(\mathrm{Var})$ *is a function which assigns to each state a set of propositional variables.* L *is called a 'labeling'.*

Definition 2. *Let* $\mathcal{M} = \langle S, L\rangle$ *be a* PLTL*-model,* $s_i \in S$*, and* $A \in \mathcal{L}$*. The relation '*$\mathcal{M}$ satisfies A at state s_i*', formally* $\mathcal{M}, s_i \models A$*, is inductively defined:*

1. $\mathcal{M}, s_i \models \mathsf{true}$ *and* $\mathcal{M}, s_i \not\models \mathsf{false}$.
2. $\mathcal{M}, s_i \models P$ *iff* $P \in L(s_i)$.
3. $\mathcal{M}, s_i \models \neg A$ *iff* $\mathcal{M}, s_i \not\models A$.
4. $\mathcal{M}, s_i \models A \wedge C$ *iff* $\mathcal{M}, s_i \models A$ *and* $\mathcal{M}, s_i \models C$.
5. $\mathcal{M}, s_i \models A \vee C$ *iff* $\mathcal{M}, s_i \models A$ *or* $\mathcal{M}, s_i \models C$.
6. $\mathcal{M}, s_i \models \mathsf{X}\, A$ *iff* $\mathcal{M}, s_{i+1} \models A$.

7. $\mathcal{M}, s_i \models \mathsf{G}\, A$ *iff* $\mathcal{M}, s_{i+j} \models A$ *for all* $j \geq 0$.
8. $\mathcal{M}, s_i \models \mathsf{F}\, A$ *iff there exists a* $j \geq 0$ *such that* $\mathcal{M}, s_{i+j} \models A$.
9. $\mathcal{M}, s_i \models A \mathcal{U} C$ *iff there exists a* $j \geq 0$ *such that* $\mathcal{M}, s_{i+j} \models C$ *and* $\mathcal{M}, s_{i+k} \models A$ *for all* $0 \leq k < j$.
10. $\mathcal{M}, s_i \models A \mathcal{B} C$ *iff for all* $j \geq 0$ *with* $\mathcal{M}, s_{i+j} \models C$ *there exists a* $0 \leq k < j$ *with* $\mathcal{M}, s_{i+k} \models A$.

If $\mathcal{M}, s_i \models A$ *for all* $s_i \in S$, *we write* $\mathcal{M} \models A$. *A formula* A *is* PLTL-satisfiable *iff there exists a* PLTL-*model* $\mathcal{M} = \langle S, L \rangle$ *and a state* $s_i \in S$ *such that* $\mathcal{M}, s_i \models A$. *A formula* A *is* PLTL-valid *iff* $\mathcal{M} \models A$ *for all* PLTL-*models* $\mathcal{M} = \langle S, L \rangle$. *Then we write* PLTL $\models A$.

Formulae which contain the symbol $\neg$ only immediately before positive literals are called formulae in *negation normal form*. The PLTL-valid equivalences $(\neg \mathsf{X}\, A \leftrightarrow \mathsf{X} \neg A)$, $(\neg \mathsf{G}\, A \leftrightarrow \mathsf{F} \neg A)$, $(\neg(A \mathcal{U} C) \leftrightarrow (\neg A \mathcal{B} C))$, and $(\neg(A \mathcal{B} C) \leftrightarrow (\neg A \mathcal{U} C))$ allow us to push the negation inwards and to obtain for any formula an equivalent formula in negation normal form. In the following we restrict ourselves to formulae in negation normal form.

Definition 3. *The* complement $\overline{A}$ *of a formula* A *in negation normal form is inductively defined as follows. 1.* $\overline{\mathsf{true}} := \mathsf{false}$ *and* $\overline{\mathsf{false}} := \mathsf{true}$. *2.* $\overline{P} := \neg P$ *and* $\overline{\neg P} := P$. *3.* $\overline{A \wedge C} := (\overline{A} \vee \overline{C})$ *and* $\overline{A \vee C} := (\overline{A} \wedge \overline{C})$. *4.* $\overline{\mathsf{G}\, A} := \mathsf{F}\, \overline{A}$ *and* $\overline{\mathsf{F}\, A} := \mathsf{G}\, \overline{A}$. *5.* $\overline{A \mathcal{B} C} := \overline{A} \mathcal{U} C$ *and* $\overline{A \mathcal{U} C} := \overline{A} \mathcal{B} C$.

Definition 4. *We classify the formulae in negation normal form: 1. Propositional constants, literals and formulae of the form* $\mathsf{X}\, A$ *are called* elementary. *2. All other formulae are called* non-elementary *and can be represented either as* α*-formulae (conjunctions) or as* β*-formulae (disjunctions) according to the following tables:*

α	α_1	α_2
$A \wedge C$	A	C
$\mathsf{G}\, A$	A	$\mathsf{X}\, \mathsf{G}\, A$
$A \mathcal{B} C$	$\overline{C}$	$A \vee \mathsf{X}\, (A \mathcal{B} C)$

β	β_1	β_2
$A \vee C$	A	C
$\mathsf{F}\, D$	D	$\mathsf{X}\, \mathsf{F}\, D$
$C \mathcal{U} D$	D	$C \wedge \mathsf{X}\, (C \mathcal{U} D)$

β*-formulae of the form* $\mathsf{F}\, D$ *and* $C \mathcal{U} D$ *are also called* eventuality *formulae or* eventualities *for short; in order for these formulae to hold at a certain state in a model, there must be a future state where* D *'eventually' holds.*

In the following we use α, α_1, α_2 to denote an α-formula and its conjuncts and β, β_1, β_2 to denote a β-formula and its disjuncts. Moreover, we assume for the rest of the paper that there are no formulae of the form $\mathsf{F}\, D$; they can be written as $\mathsf{true} \mathcal{U} D$.

Definition 5. *We define the* closure $cl(A)$ *for any formula* A *in negation normal form: 1.* A *is in* $cl(A)$. *2. If* $\neg P$ *is in* $cl(A)$, *then* P *is in* $cl(A)$. *3. If* $\mathsf{X} B$ *is in* $cl(A)$, *then* B *is in* $cl(A)$. *4. If* α *is in* $cl(A)$, *then* α_1 *and* α_2 *are in* $cl(A)$. *5. If* β *is in* $cl(A)$, *then* β_1 *and* β_2 *is in* $cl(A)$.

Stefan Schwendimann. A New One-Pass Tableau Calculus for PLTL. In *Proceedings, TABLEAUX'98, Oisterwijk, The Netherlands*, LNAI 1397, pages 277 – 291, Springer 1998

The closure of a formula is essentially the set of all subformulae augmented with the α_2 and β_2 parts of the temporal connectives. It is also called the Fischer-Ladner closure [5]. Before we turn to Hintikka structures for PLTL, we define some properties for more general 'labeling' functions which assign to states sets of formulae rather than sets of variables.

Definition 6. *Let S be a (possibly finite) sequence of states $s_0\, s_1\, \ldots$, L a function $L : S \to \mathrm{Pow}(\mathrm{Fml})$, and $s_i \in S$.*

1. *Propositional consistency properties:*
   (PC0) false *is not in $L(s_i)$.*
   (PC1) If a literal M is in $L(s_i)$, then its complement $\overline{M}$ is not in $L(s_i)$.
   (PC2) If α is in $L(s_i)$, then α_1 and *α_2 are in $L(s_i)$.*
   (PC3) If β is in $L(s_i)$, then β_1 or *β_2 is in $L(s_i)$.*
2. *Local consistency property:*
   (LC) If $\mathsf{X}\, A$ is in $L(s_i)$ and s_i is not the last state if S is finite, then A is in $L(s_{i+1})$.

We say that L fulfills one of the above properties if the respective condition is satisfied for all *states s_i of the sequence S.*

In the next definition we describe the set of eventualities that are not 'satisfied' in a sequence of states.

Definition 7. *Let S be a (possibly finite) sequence of states $s_0\, s_1 \ldots$ and $L : S \to \mathrm{Pow}(\mathrm{Fml})$ a labeling. Then the set $open(S, L)$ of eventualities is defined as:*

$$open(S,L) := \{C\, \mathcal{U}\, D \mid \exists i\, (C\, \mathcal{U}\, D \in L(s_i)) \text{ and } \forall j \geq i\, (D \notin L(s_j))\}.$$

The following definition of a (pre-)Hintikka structure can be found in the literature (e.g. [4]).

Definition 8. *A* pre-Hintikka *structure $\mathcal{H}$ is a pair $\langle S, L\rangle$, where S is a sequence of states $(s_i)_{i\in\mathbb{N}} = s_0\, s_1 \ldots$ and $L : S \to \mathrm{Pow}(\mathrm{Fml})$ is a labeling function that fulfills the properties (PC0-3) and (LC).*

By restricting the labeling function L to variables, we can associate with each pre-Hintikka structure $\mathcal{H} = \langle S, L\rangle$ a model $\mathcal{M}_{\mathcal{H}} := \langle S, L{\restriction}\mathrm{Var}\rangle$.

Definition 9. *We say that a pre-Hintikka structure $\mathcal{H} = \langle S, L\rangle$ is a* Hintikka *structure if $open(S, L) = \emptyset$, that is, if we have for any state s_i and any eventuality $C\, \mathcal{U}\, D$: If $C\, \mathcal{U}\, D \in L(s_i)$, then there exists a $j \geq i$ with $D \in L(s_j)$.*

$\mathcal{H}$ is said to be a (pre-)Hintikka structure for a formula A *if $A \in L(s_0)$. We say that $\mathcal{H}$ is a* complete *(pre-)Hintikka structure for A if for all i: $L(s_i) = \{C \mid C \in cl(A)$ and $\mathcal{M}_{\mathcal{H}}, s_i \models C\}$.*

Note that any Hintikka structure for A can be made into a complete Hintikka structure for A by adding to $L(s_i)$ all formulae of the closure that are satisfied at s_i. The following standard theorem relates the existence of Hintikka structures to the existence of models.

Theorem 10. *A formula A in negation normal form is* PLTL*-satisfiable iff there exists a Hintikka structure for A.*

Proof. See for instance [9].

In the following we deal with a set W of words over an alphabet S. We write ws for the concatenation of a word w and a single element $s \in S$. Similarly, we write ww' for the concatenation of the two words w and w'. w and w' may also be the empty word. Now we introduce a new type of structures which are essentially trees with loops on their branches.

Definition 11. *A* loop tree *is a tuple $\mathcal{T} = \langle W, S, L, R\rangle$ where:*

1. *S is a finite set.*
2. *W is a finite set of finite words over S where:*
   (a) *If $w = s_0 s_1 \dots s_k \in W$, then $s_i \neq s_j$ for all $0 \leq i < j \leq k$.*
3. *R is a binary relation on W with the following properties:*
   (a) *$(w, ws) \in R$ for all $w, ws \in W$.*
   (b) *If $w \in W$ and $ws \notin W$ for all $s \in S$, then there exists a word $w' \in W$ such that w' is a prefix of w and $(w, w') \in R$.*
   (c) *If $(w, w') \in R$, then either w' is of the form ws or w' is a prefix of w.*
4. *$L : W \rightarrow \mathrm{Pow(Fml)}$ is a labeling function with the property: $L(ws) = L(w's)$ for all $ws, w's \in W$.*

The set S can be viewed as a set of nodes and the words W as directions how to reach these nodes. The conditions say that a word should contain a node only once, and that words which cannot be extended are related to a prefix. This means that we basically have a tree-like structure with loops back on the branches where at the end of each branch we have at least one loop back. The arrows in Fig. 1 correspond to the relation R. The labeling is controlled by the last node of a word. A word is essentially the last node plus the information how it is reached. Therefore words will also be called states.

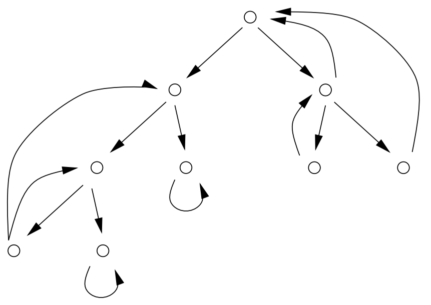


Fig. 1. Example of a loop tree.

Definition 12. *Let $\mathcal{T} = \langle W, S, L, R\rangle$ be a loop tree.*

1. *If $ws \in W$ and $w \notin W$, then ws is called a* root *of $\mathcal{T}$.*
2. *A* path *through $\mathcal{T}$ is a finite or infinite R-sequence of states $w_0, \dots, w_i, w_{i+1}, \dots$, where $(w_i, w_{i+1}) \in R$ for all w_i of the sequence (except the last one if the sequence is finite).*

Stefan Schwendimann. A New One-Pass Tableau Calculus for PLTL. In *Proceedings, TABLEAUX'98, Oisterwijk, The Netherlands*, LNAI 1397, pages 277 – 291, Springer 1998

3. *A* loop branch *of $\mathcal{T}$ is a finite path $w_0, w_1, \dots, w_k, w_{k+1}$ through $\mathcal{T}$ where w_0 is a root and for all $i < k$ $w_{i+1} = w_i s_i$ for some $s_i \in S$. The last state w_{k+1} is identical to a previous state, i.e. $w_{k+1} = w_j$ for a $j \leq k$, and it is called the* loop state *of the branch. The suffix path $w_j, w_{j+1}, \dots, w_k, w_{k+1}$ is called the* loop *of the branch. We say that a path π* visits *the loop branch or simply the loop if w_k, w_{k+1} occurs in π (as a pair of consecutive states).*
4. *If $\pi = w_0, \dots, w_j, w_{j+1}, \dots, w_k, w_{k+1}$ is a loop branch, the set $open_{\inf}(\pi, L)$ is defined as:*

$$\begin{aligned} open_{\inf}(\pi, L) := \{C\,\mathcal{U}\,D \mid\ & C\,\mathcal{U}\,D \in open(\pi, L) \\ & and\ \forall i, (j \leq i \leq k \Rightarrow D \notin L(w_i))\}. \end{aligned}$$

5. *The function* $\mathrm{depth}_{\mathcal{T}} : W \to \mathbb{N}$ *is defined as follows: 1.* $\mathrm{depth}_{\mathcal{T}}(w) := 0$ *for any root w of $\mathcal{T}$. 2.* $\mathrm{depth}_{\mathcal{T}}(ws) := \mathrm{depth}_{\mathcal{T}}(w) + 1$ *for any w, $ws \in W$.*

Remarks: 1. Note that a loop tree may contain several roots and may therefore represent several tree-like structures. 2. A loop branch is defined to contain the backward loop. Therefore a 'physical branch' can contain several loop branches that share a common prefix path (see Fig. 1). In particular, loops may also start at non-leaf nodes. 3. Obviously, $open_{\inf}(\pi, L)$ is a subset of $open(\pi, L)$. It denotes the eventualities of π which are not satisfied on the loop itself even if it is visited infinitely many times.

Proposition 13. *If $w_j, w_{j+1}, \dots, w_k, w_{k+1}$ is a loop ($w_{k+1} = w_j$) and a path π visits it repeatedly (i.e. multiple occurrences of w_k, w_{k+1} on π), then obviously all other states of the loop $w_{j+1}, \dots, w_{k-1}$ must occur in π between two occurrences of w_k, w_{k+1}, although not necessarily in a row.*

Definition 14. *Let $\mathcal{T} = \langle W, S, L, R\rangle$ be a loop tree. The* subtree of $\mathcal{T}$ at $w \in W$ *is a structure $\mathcal{T}' = \langle W', S', L', R'\rangle$ defined as follows: 1. $S' := S$. 2. $W' := \{ww' \mid ww' \in W\}$. 3. $R' := \{(ww', ww'') \mid (ww', ww'') \in R\}$. 4. $L' := L{\restriction}W'$.*
We say that $\mathcal{T}'$ is an isolated *subtree of $\mathcal{T}$ if $(w', v) \notin R$ for any $w' \in W'$ and $v \in W \setminus W'$.*

An isolated subtree is obviously a loop tree. Whether or not a subtree is isolated can be determined easily by checking the loop states of the loop branches that pass through the subtree's root.

Lemma 15. *Let $\mathcal{T} = \langle W, S, L, R\rangle$ be a loop tree and $\mathcal{T}' = \langle W', S', L', R'\rangle$ the subtree at $w \in W$. Then we have: $\mathcal{T}'$ is isolated iff*

$$\begin{aligned} \mathrm{depth}_{\mathcal{T}}(w) \leq \min(\{\mathrm{depth}_{\mathcal{T}}(w') \mid\ & w' \textit{is a loop state of a} \\ & \textit{loop branch of } \mathcal{T} \textit{ containing } w\}). \end{aligned}$$

Proof. If $\mathcal{T}'$ is isolated, then no loop branch of $\mathcal{T}$ containing w can have a loop state outside $\mathcal{T}'$. Since w is the root of $\mathcal{T}'$, the depth of a loop state must be greater or equal than the depth of w.

Conversely, if the depth of a loop state is greater or equal than the depth of w, then it must belong to $\mathcal{T}'$ since a branch may only loop back on itself. Therefore $\mathcal{T}'$ must be isolated.

Definition 16.

1. *A* pre-Hintikka-tree *is a loop tree $\mathcal{T} = \langle W, S, L, R\rangle$ where L fulfills the properties (PC0-3) and (LC) for all paths through $\mathcal{T}$.*
2. *A* Hintikka-tree for a formula *A is a pre-Hintikka-tree $\mathcal{T} = \langle W, S, L, R\rangle$ with the additional property that there exists an infinite path $\pi = w_0, w_1, \ldots$ through $\mathcal{T}$ with $A \in L(w_0)$ and $open(\pi, L) = \emptyset$.*

Proposition 17. *Let $\pi = w_j, w_{j+1}, \ldots, w_k, w_{k+1}$ be the loop ($w_{k+1} = w_j$) of a pre-Hintikka-tree $\mathcal{T} = \langle W, S, L, R\rangle$. Then we have: If an eventuality $C \,\mathcal{U}\, D$ is in $open_{\inf}(\pi, L)$, then $C \,\mathcal{U}\, D$ and $\mathsf{X}\,(C \,\mathcal{U}\, D)$ are in $L(w_i)$ for all i with $j \leq i \leq k$.*

The following lemma states that the open eventualities of a path depend in a simple way on the unfulfilled eventualities of single loop branches.

Lemma 18. *Let π be an infinite path through the pre-Hintikka-tree $\mathcal{T} = \langle W, S, L, R\rangle$ and $\pi_1, \ldots, \pi_m$ be the loops of $\mathcal{T}$ that are visited infinitely many times by π. Then we have:*

$$open(\pi, L) = \bigcap_{i=1\ldots m} open_{\inf}(\pi_i, L).$$

Proof. $\supset$: Let $C \,\mathcal{U}\, D$ be in $\bigcap_{i=1\ldots m} open_{\inf}(\pi_i, L)$. There is a point in time after which only the loops $\pi_1, \ldots, \pi_m$ and, therefore, only states from $\pi_1, \ldots, \pi_m$ are visited. If $C \,\mathcal{U}\, D \in open_{\inf}(\pi_i, L)$, then D is not in any state of π_i, and by Proposition 17 we know that $C \,\mathcal{U}\, D$ is in each state of π_i. Therefore $C \,\mathcal{U}\, D$ must be in $open(\pi, L)$.

$\subset$: Let $C \,\mathcal{U}\, D$ be in $open(\pi, L)$. Then there is a state s in π such that for any future state s' the formula D is not in $L(s')$ but $\mathsf{X}\,(C \,\mathcal{U}\, D)$ is in $L(s')$. This implies that for any state s'' from $\pi_1, \ldots, \pi_m$ the formula D is not in $L(s'')$ and $\mathsf{X}\,(C \,\mathcal{U}\, D)$ is in $L(s'')$ since by Proposition 13 all these states are visited by π after s. Therefore $C \,\mathcal{U}\, D$ is in $open_{\inf}(\pi_i, L)$ for all π_i ($i = 1 \ldots m$).

Theorem 19. *There is a Hintikka structure for a formula A iff there exists a Hintikka-tree for A.*

Proof. The direction from right to left is obvious. If $\mathcal{T} = \langle W, S, L, R\rangle$ is a Hintikka-tree for A then simply choose a path $\pi = w_0 w_1 \ldots$ through $\mathcal{T}$ with $A \in L(w_0)$ and $open(\pi, L) = \emptyset$. $\langle \pi, L\rangle$ is then a Hintikka structure for A.

For the direction from left to right assume that $\mathcal{H} = \langle S, L\rangle$ is a Hintikka structure for A with $S = s_0 s_1 \ldots s_i s_{i+1} \ldots$. First, we introduce an equivalence relation $\sim$ on the elements of S: $s_i \sim s_j$ iff $L(s_i) \cap cl(A) = L(s_j) \cap cl(A)$. The equivalence class of s_i is denoted by $[s_i]$. We construct a Hintikka-tree $\mathcal{T} = \langle W, S', L', R\rangle$ for A in the following way (w, w', w'' may be the empty word):

1. $S' := S/{\sim}$.
2. W and R are defined inductively:

Stefan Schwendimann. A New One-Pass Tableau Calculus for PLTL. In *Proceedings, TABLEAUX'98, Oisterwijk, The Netherlands*, LNAI 1397, pages 277 – 291, Springer 1998

(a) $[s_0]$ is an element of W.
(b) If $w[s_i]$ is an element of W, then we distinguish two cases:
   i. If $w[s_i]$ contains a state equivalent to s_{i+1}, that is, if $w[s_i] = w'[s_j]w''$ and $s_j \sim s_{i+1}$, then $(w[s_i], w'[s_j])$ is in R (a loop).
   ii. Otherwise $w[s_i][s_{i+1}]$ belongs to W and $(w[s_i], w[s_i][s_{i+1}])$ is in R.

3. The labeling L' is defined as $L'(w[s_i]) := L(s_i) \cap cl(A)$.

The structure $\mathcal{T}$ is obviously a loop tree. S' is finite since $cl(A)$ is finite, L' satisfies (PC0-3) and (LC), and by the construction there is a path $\pi = w_0, w_1, \ldots$ through $\mathcal{T}$ (corresponding to $s_0 s_1 \ldots$) with $w_0 = [s_0]$, $A \in L'(w_0)$ and $open(\pi, L') = \emptyset$. Therefore $\mathcal{T}$ is a Hintikka-tree for A.

4 The Calculus PLTL_T

We present a Tableau-like calculus for PLTL that is complete and correct with respect to the PLTL semantics. It operates on so-called prestates which contain the full information needed to decide satisfiability of formulae in negation normal form.

In the following we use Γ and Σ for finite sets of formulae in negation normal form, and Λ for sets of *literals* (and possibly constants). We also write A, Γ for the set $\{A\} \cup \Gamma$, and Γ, Σ for the union $\Gamma \cup \Sigma$, and $\mathsf{X}\,\Gamma$ is used for the set $\{\mathsf{X}\,A \mid A \in \Gamma\}$.

For lists we have the following conventions: We use $*$ for the concatenation of lists and [] for the empty list. If M is a list, then we write $len(M)$ for the length of M and $M[i]$ for the i^{th} element of M $(1 \leq i \leq len(M))$. If M is a list of tuples, then we write $M[i]_j$ to denote the projection to the j^{th} element of $M[i]$.

Definition 20. *A* prestate *is a triple* $(\Gamma, Save, Res)$*, also written as* $\Gamma \mid Save \mid Res$ *where:*

1. *Γ is a finite set of formulae in negation normal form.*
2. *Save is a structure to store history information. It is a pair (Ev, Br), also written as $Ev\,;\,Br$, where Ev is a set of formulae in negation normal form representing the currently satisfied eventualities, Br is a list of pairs (Γ', Ev') representing the current branch, and Γ' and Ev' correspond to the Γ and Ev parts of previous prestates.*
3. *Res is a structure to store partial result information. It is a pair (n, uev), where n is a natural number indicating the 'earliest' prestate reachable by the current one, and uev is a set of eventuality formulae in negation normal form. It represents the unfulfilled eventualities of the current branch.*

A prestate is said to be a state *if Γ is of the form $\Lambda, \mathsf{X}\,\Sigma$, that is, if Γ consists only of elementary formulae.*

According to the above definition, $\Gamma \mid Ev\,;\,Br \mid (n, uev)$ is the extended notion for an abstract prestate. To focus on the locally relevant parts of a prestate, we use '...' for the 'unimportant' parts (e.g. $\Gamma \mid \ldots \mid \ldots$). If '...' appears at the same position in the numerator and the denominator(s) of a rule, then we mean that the corresponding parts are the same.

Definition 21. *The Tableau calculus PLTL_T is defined as follows:*
a) Terminal rules:

$$\mathsf{false}, \varGamma \mid Ev\,;\, Br \mid (len(Br), \{\mathsf{false}\}) \qquad (\mathsf{false})$$

$$P, \neg P, \varGamma \mid Ev\,;\, Br \mid (len(Br), \{\mathsf{false}\}) \qquad (contr)$$

$$\varLambda, \mathsf{X}\,\varSigma \mid Ev\,;\, Br \mid (k, uev) \qquad (loop)$$

where in (loop) there exists an i, $1 \leq i \leq len(Br)$, such that:
1. $\varLambda, \mathsf{X}\,\varSigma = Br[i]_1$.
2. $k = i-1$ and $uev = \{C\,\mathcal{U}\,D \mid C\,\mathcal{U}\,D \in \varSigma$ and $D \notin (\cup_{j=i+1}^{len(Br)} Br[j]_2 \cup Ev)\}$.

b) α-rules:

$$\frac{\alpha, \varGamma \mid \ldots \mid \ldots}{\alpha_1, \alpha_2, \varGamma \mid \ldots \mid \ldots}\ (\alpha)$$

c) β-rules:

$$\frac{A \vee B, \varGamma \mid \ldots; Br \mid (n, uev)}{A, \varGamma \mid \ldots; Br \mid (n_1, uev_1) \qquad B, \varGamma \mid \ldots; Br \mid (n_2, uev_2)}\ (\vee)$$

$$\frac{C\,\mathcal{U}\,D, \varGamma \mid Ev\,;\, Br \mid (n, uev)}{D, \varGamma \mid Ev \cup \{D\}\,;\, Br \mid (n_1, uev_1) \qquad C, \mathsf{X}\,(C\,\mathcal{U}\,D), \varGamma \mid Ev\,;\, Br \mid (n_2, uev_2)}\ (\,\mathcal{U}\,)$$

where in $(\vee)$ and $(\,\mathcal{U}\,)$:
1. $n = \min(n_1, n_2)$.
2. $(m := len(Br) - 1)$.

$$uev = \begin{cases} \emptyset & \textit{if } uev_1 = \emptyset \textit{ or } uev_2 = \emptyset, \\ \{\mathsf{false}\} & \textit{if } n_1 > m \textit{ and } n_2 > m \ (\textit{and } uev_1 \neq \emptyset,\ uev_2 \neq \emptyset), \\ uev_1 & \textit{if } n_1 \leq m \textit{ and } n_2 > m \ (\textit{and } uev_2 \neq \emptyset), \\ & \textit{or if } uev_2 = \{\mathsf{false}\}, \\ uev_2 & \textit{if } n_1 > m \textit{ and } n_2 \leq m \ (\textit{and } uev_1 \neq \emptyset), \\ & \textit{or if } uev_1 = \{\mathsf{false}\}, \\ uev_1 \cap uev_2 & \textit{otherwise.} \end{cases}$$

d) Nexttime rule:

$$\frac{\varLambda, \mathsf{X}\,\varSigma \mid Ev\,;\, Br \mid \ldots}{\varSigma \mid \emptyset\,;\, Br * ((\varLambda, \mathsf{X}\,\varSigma), Ev) \mid \ldots}\ (\,\mathsf{X}\,)$$

In order to ensure termination, the α- and β-rules and the nexttime rule are restricted to prestates that are not instances of a terminal rule. We call α in (α), $A \vee B$ in $(\vee)$, and $C\,\mathcal{U}\,D$ in $(\,\mathcal{U}\,)$ the decomposed formula *of the respective rule.*

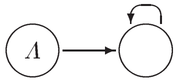
Remark 22.

- The main difference to a modal calculus is the result part which is synthesized bottom-up (from children to parents). It is needed because a single branch need not be 'open' or 'closed'; it may be 'open' in connection with some other branches.
- (loop): The sequence $Br[i]_1, \ldots, Br[len(Br)]_1, (\varLambda, \mathsf{X}\,\varSigma)$ corresponds to the loop of a branch. uev is defined to be the set $open_{\mathrm{inf}}(.,.)$ of eventualities that are not satisfied on this loop branch (see proposition 17).

Stefan Schwendimann. A New One-Pass Tableau Calculus for PLTL. In *Proceedings, TABLEAUX'98, Oisterwijk, The Netherlands*, LNAI 1397, pages 277 – 291, Springer 1998

- (β-rules): A β-rule corresponds to a branching of the tree. n is set to the minimum depth of the states to which branches of the two subtrees can loop back. uev is the minimal set of eventualities that are left open by any infinite path visiting only loops of the subtree below this β node.
- (Nexttime rule): The current state and the eventualities that are satisfied by the current state are appended to the branch Br.
- Note that the sets Γ, Λ, and Σ may be empty. For instance, if Σ is empty in the numerator of (X), we obtain the following fragment of a tableau which ends in a basically empty instance of (loop). On the right the corresponding model is shown.

$$\begin{array}{cl} \Lambda \mid Ev \,;\, Br \mid \ldots & \\ \hline \emptyset \mid \emptyset \,;\, Br * (\Lambda, Ev) \mid \ldots & (\mathsf{X}) \\ \hline \emptyset \mid \emptyset \,;\, Br * (\Lambda, Ev) * (\emptyset, \emptyset) \mid (len(Br) + 1, \emptyset) \quad \text{(loop)} & (\mathsf{X}) \end{array}$$



Definition 23. *A* tableau for a prestate *ps is a tree of prestates with root ps and where the sons of a node (prestate) correspond to an application of a* PLTL_T *rule to the node. We say that the tableau is* expanded*, if each leaf node is an instance of a terminal rule.*

Let A be a formula and n the number of subformulae of A. Then it is clear that any tableau for $A \mid \ldots \mid \ldots$ is finite. There are $2^{O(n)}$ many subsets of $cl(A) \cup cl(\overline{A})$. Each Γ of a prestate $\Gamma \mid \ldots \mid \ldots$ is such a subset, and since the terminal rules must be applied whenever they can be applied, the number of different prestates on each branch is finite. Therefore, the total number of prestates in any tableau for $A \mid \ldots \mid \ldots$ is finite and any expansion will eventually terminate.

Proposition 24.

a) For every formula A there is an $n \in \mathbb{N}$ and a set $uev \subseteq$ Fml such that there is an expanded tableau for $A \mid \emptyset \,;\, Br \mid (n, uev)$.

b) If in a tableau the set uev of a prestate is empty , then the set uev of the root of the tableau is also empty.

Example 25. We show the essential branch of a tableau for the satisfiable property $\mathsf{G}\,\mathsf{F}\,p \wedge \mathsf{G}\,\mathsf{F}\,\neg p$ (recall that $\mathsf{F}\,p$ can be written as $\mathsf{true}\,\mathcal{U}\,p$). The α- and β-rules are applied until we reach a state with only elementary formulae. The currently decomposed formula is in parentheses. It is left to the reader to fill in the missing *Save* and *Res* parts.

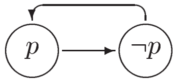
$$\begin{array}{cl} (\mathsf{G}\,\mathsf{F}\,p \wedge \mathsf{G}\,\mathsf{F}\,\neg p) \mid \ldots \mid \ldots & \\ \hline (\mathsf{G}\,\mathsf{F}\,p), \mathsf{G}\,\mathsf{F}\,\neg p \mid \ldots \mid \ldots & \\ \hline \mathsf{F}\,p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,p, (\mathsf{G}\,\mathsf{F}\,\neg p) \mid \ldots \mid \ldots & \\ \hline (\mathsf{F}\,p), \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,p, \mathsf{F}\,\neg p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,\neg p \mid \ldots \mid \ldots & \\ \hline p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,p, (\mathsf{F}\,\neg p), \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,\neg p \mid \{p\} \,;\, . \mid \ldots & \mathrm{Sub}_1 \\ \hline p, \neg p, \ldots \qquad p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,p, \mathsf{X}\,\mathsf{F}\,\neg p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,\neg p \mid \{p\} \,;\, . \mid \ldots & \\ \hline (\mathsf{G}\,\mathsf{F}\,p), \mathsf{F}\,\neg p, \mathsf{G}\,\mathsf{F}\,\neg p \mid \ldots \mid \ldots & (\mathsf{X}) \\ \hline \mathsf{F}\,p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,p, \mathsf{F}\,\neg p, (\mathsf{G}\,\mathsf{F}\,\neg p) \mid \ldots \mid \ldots & \end{array}$$

Stefan Schwendimann. A New One-Pass Tableau Calculus for PLTL. In *Proceedings, TABLEAUX'98, Oisterwijk, The Netherlands*, LNAI 1397, pages 277 – 291, Springer 1998

$$\begin{array}{ccc}
 & \overline{(\mathsf{F}\,p), \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,p, \mathsf{F}\,\neg p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,\neg p \mid \ldots \mid \ldots} & \\
\mathrm{Sub}_2 & \overline{\mathsf{X}\,\mathsf{F}\,p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,p, (\mathsf{F}\,\neg p), \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,\neg p \mid \ldots \mid \ldots} & \\
 & \overline{\mathsf{X}\,\mathsf{F}\,p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,p, \neg p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,\neg p \mid \{\neg p\}\,;\,.\mid \ldots} & \mathrm{Sub}_3 \\
 & \overline{\mathsf{F}\,p, (\mathsf{G}\,\mathsf{F}\,p), \mathsf{G}\,\mathsf{F}\,\neg p \mid \ldots \mid \ldots}\ (\mathsf{X}) & \\
 & \overline{\mathsf{F}\,p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,p, (\mathsf{G}\,\mathsf{F}\,\neg p) \mid \ldots \mid \ldots} & \\
 & \overline{(\mathsf{F}\,p), \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,p, \mathsf{F}\,\neg p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,\neg p \mid \ldots \mid \ldots} & \\
 & \overline{p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,p, (\mathsf{F}\,\neg p), \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,\neg p \mid \{p\}\,;\,.\mid \ldots} & \mathrm{Sub}_4 \\
\overline{p, \neg p, \ldots} & \overline{p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,p, \mathsf{X}\,\mathsf{F}\,\neg p, \mathsf{X}\,\mathsf{G}\,\mathsf{F}\,\neg p \mid \ldots \mid (0, \emptyset)} & \text{(loop)}
\end{array}$$

The highlighted prestates above the (X)'s are the *states* of the tableau; the first one (at 'state' depth 0) satisfies p and the second one satisfies $\neg p$. The essential branch ends in an instance of (loop), where in the *Res* part the 0 refers to the depth 0 of the first state and the $\emptyset$ indicates that all eventualities (the only candidate to check stems from $\mathsf{X}\,\mathsf{F}\,\neg p$) are satisfied on this loop. $\mathrm{Sub}_{1\ldots4}$ stand for other branches in the expanded tableau.

The corresponding model is very simple:



Definition 26. *The* loop tree $\mathcal{T} = \langle W, S, L, R\rangle$ for an expanded tableau *is defined in the following way:*

1. *S is the set of all states (not prestates!) of the tableau and the set of leaf nodes which are not instances of (loop).*
2. *W is the set of paths (in terms of S) to the elements of S in the tableau.*
3. *R:*
   (a) *(w, ws) is in R for all $w, ws \in W$.*
   (b) *(ws, ws) is in R if s is an instance of* (false) *or* (*contr*)*. That is, we draw a loop to the last state itself if it is inconsistent.*
   (c) *If $w \in W$ is a path to a state which is the last state before an instance of (loop) in the tableau, then w must be of the form $w'sw''$ where s is the referenced loop state. Then $(w, w's)$ is in R.*
4. *$L(ws)$ is the set Γ if $s = \Gamma \mid \ldots \mid \ldots$ plus all the formulae that are decomposed in the tableau between w and ws.*

We could also (formally) omit the classical contradictions from the loop tree (and the states which have only contradictory prestates below), and we would obtain a pre-Hintikka-tree. However, the relevant information is always in the result part.

Lemma 27. *Let $\mathcal{T} = \langle W, S, L, R\rangle$ be the loop tree for an expanded tableau. Then we have for all $ws \in W$, $s = \Lambda, \mathsf{X}\,\Sigma \mid Ev\,;\,Br \mid (n, uev)$:*
a) L fulfills (PC0-3) and (LC) for ws if s is not an instance of (false) *or (contr).*
b) $\mathrm{depth}_{\mathcal{T}}(ws) = len(Br)$.
c) $n = \min(\{\mathrm{depth}_{\mathcal{T}}(v) \mid v$ *is a loop state of a branch* $\pi = \ldots ws \ldots\}$.
d) The subtree of $\mathcal{T}$ at ws is isolated iff $n \geq len(Br)$.

Proof. Follows from the definition of the calculus and Definition 26. The part d) follows from b) and c) and Lemma 15.

Note that the β-rule applications in the tableau are between two states. The lower state is at depth $len(Br)$. The conditions in the β-rules, however, control the result synthesis for the upper state which is at depth $len(Br) - 1$.

Theorem 28 (Correctness). *If A is a formula in negation normal form and if there exists an n such that there is a expanded tableau for $A\,|\,\emptyset\,;\,[]\,|\,(n, \emptyset)$, then A is satisfiable.*

Proof. (Sketch) We basically show that for any prestate $ps = \ldots \,|\, \ldots \,|\,(n, uev)$ in the tableau with $uev \neq \{\mathsf{false}\}$ there is a pre-Hintikka structure $\mathcal{H}_{ps} = \langle S, L\rangle$ for A with $open(S, L) = uev$. We represent the pre-Hintikka structure as a loop tree with one single branch π starting with the path that leads to ps. We proceed bottom-up, that is, by induction on the depth of the tableau subtree with root ps. The main case involves a linearization of two loops into a single one as depicted in Fig. 2 (the capital letters denote sections of the path). Lemma 18 ensures that the set of open eventualities is the intersection of the corresponding sets of the two loops, according to the condition in the β-rules.

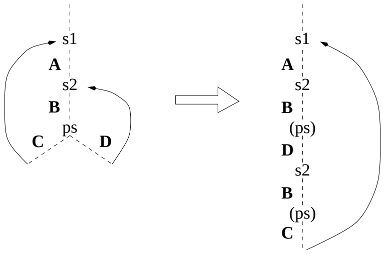


Fig. 2. Loop linearization.

Lemma 29. *Let $\mathcal{T} = \langle W, S', L', R\rangle$ be the loop tree for an expanded tableau for $A\,|\,\emptyset\ \ []\,|\,(n_A, uev_A)$. Then we have: If there is a infinite path π through $\mathcal{T}$ with $open(\pi, L) = \emptyset$, and if L fulfills (PC0-3) and (LC) on π, then uev_A must be $\emptyset$.*

Proof. (Sketch) We use Lemma 18, 15 and Lemma 27 b),c) and proceed by induction on the depth of the subtree visited by π.

Theorem 30 (Completeness). *If a formula A in negation normal form is satisfiable, then there exists a tableau for $A\,|\,\emptyset\,;\,[]\,|\,(n, \emptyset)$ for some $n \in \mathbb{N}$.*

Proof. If A is satisfiable, there exists a complete Hintikka structure $\mathcal{H} = \langle S, L\rangle$ for A by Theorem 10. Let S be the sequence $s_0 s_1 \ldots$, and let $\mathcal{T} = \langle W, S', L', R\rangle$ be the loop tree for an expanded tableau for $A\,|\,\emptyset\ \ []\,|\,(n, uev)$. We define inductively a map $\varphi : S \to W$ which provides us with a path $\pi = \varphi(s_0)\varphi(s_1)\ldots$ through $\mathcal{T}$ with the following properties:

Stefan Schwendimann. A New One-Pass Tableau Calculus for PLTL. In *Proceedings, TABLEAUX'98, Oisterwijk, The Netherlands*, LNAI 1397, pages 277 – 291, Springer 1998

(a) $A \in L'(\varphi(s_i))$ if $i = 0$.
(b) $L'(\varphi(s_i)) \subseteq L(s_i)$.
(c) If $C\,\mathcal{U}\,D \in L'(\varphi(s_i))$ and $D \in L(s_i)$, then $D \in L'(\varphi(s_i))$.

$i = 0$: First, A is in $L'(w)$ for every root $w \in W$ since the loop tree stems from a tableau for $A \mid \ldots \mid \ldots$, and A is also in $L(s_0)$, since $\mathcal{H}$ is a Hintikka structure for A. Second, there must exist a root $w_0 \in W$ with $L'(w_0) \subseteq L(s_0)$ since in the tableau there is a root state for each possible decomposition of A, and $L(s_0)$ must contain at least one set of decomposed formulae ($L(s_0)$ contains A and L fulfills (PC2) and (PC3)). Third, we can choose a w_0 so that for each $C\,\mathcal{U}\,D \in L'(w_0)$ with $D \in L(s_0)$ the decomposition $\{C\,\mathcal{U}\,D, D\}$ rather than $\{C\,\mathcal{U}\,D, \mathsf{X}(C\,\mathcal{U}\,D)\}$ is a subset of $L'(w_0)$. Set $\varphi(s_0) = w_0$.

$i \rightarrow i+1$: Assume that we have defined the map up to $\varphi(s_i)$. We define the sets $\mathit{next} := \{C \mid \mathsf{X}C \in L(s_i)\}$ and $\mathit{next}' := \{C \mid \mathsf{X}C \in L'(\varphi(s_i))\}$. We know that $\mathit{next} \subseteq L(s_{i+1})$ since L fulfills (LC), and that for every successor w of $\varphi(s_i)$ $\mathit{next}' \subseteq L'(w)$ (see the $(\,\mathsf{X}\,)$ rule of PLTL_T). Moreover, because of (b), we have $\mathit{next}' \subseteq \mathit{next}$. Again, since in the tableau there is a successor for each possible decomposition of next', and since $L(s_{i+1})$ must contain at least one decomposition, there must exist a $w_{i+1} \in W$ so that (b) and (c) are fulfilled if we set $\varphi(s_{i+1})$ to w_{i+1}.

Obviously, L' fulfills (PC0-3) and (LC) on $\pi = \varphi(s_0)\varphi(s_1)\ldots$. Suppose now that there exists an eventuality $C\,\mathcal{U}\,D \in \mathit{open}(\pi, L')$. Then there exists a state $\varphi(s_i)$ so that $C\,\mathcal{U}\,D \in L'(\varphi(s_j))$ and $D \notin L'(\varphi(s_j))$ for all $j \geq i$. However, because of (b) and (c) this would mean that $C\,\mathcal{U}\,D$ is in $\mathit{open}(S, L)$ as well, which is a contradiction. Applying the previous lemma 29 concludes the proof of the theorem.

5 Conclusion

We have presented a new one-pass tableau calculus for PLTL which works, as most modal calculi, on trees rather than graphs . The representation is minimal but complete, that is, it can be used directly as the basis for a decision procedure without a second phase. It has inherent advantages compared to previous approaches: 1. Only one branch needs to be considered at any one time. This makes it into a natural candidate for parallelization. 2. A simple linearization of loops allows to actually extract linear models in a canonical way. Having the details of the eventuality checking incorporated in a formal way, the calculus is also a good starting point for theoretical investigations, for instance the verification of pruning techniques. These are certainly simpler to check when the underlying structure is a tree. A decision procedure based on PLTL_T has been implemented and tested and will be publicly available as a part of the Logics Workbench [10] version 1.1.

Stefan Schwendimann. A New One-Pass Tableau Calculus for PLTL. In *Proceedings, TABLEAUX'98, Oisterwijk, The Netherlands*, LNAI 1397, pages 277 – 291, Springer 1998

References

1. M. Browne, E. Clarke, D. Dill, and B. Mishra. Automatic verification of sequential circuits using temporal logic. *IEEE Transactions on Computers*, 35:1035–1044, December 1986.
2. C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory-efficient algorithms for the verification of temporal properties. In *Formal Methods in System Design*, volume 1, pages 275–288, 1992.
3. M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors. *Handbook of Tableau Methods*, chapter Tableau Methods for Modal and Temporal Logics. Kluwer, to appear. (currently available as technical report TR-ARP-15-95, Australian National University (ANU)).
4. E. Emerson. Temporal and modal logic. In J. v. Leeuwen, editor, *Handbook of Theoretical Computer Science. Volume B*, pages 995–1072. Elsevier, 1990.
5. M.J. Fischer and R.L. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18:194–211, 1979.
6. M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*. Reidel, Dordrecht, 1983.
7. R. Gerth, D. Peled, M. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In P. Dembinski and M. Sredniawa, editors, *Protocol Specification Testing and Verification*, volume XV, pages 3–18. Chapman & Hall, 1996.
8. R. Goré. *Cut-free Sequent and Tableau Systems for Propositional Normal Modal Logic*. PhD thesis, Computer Laboratory, University of Cambridge, England, 1992.
9. G. Gough. Decision procedures for temporal logic. Technical Report UMCS-89-10-1, Department of Computer Science, University of Manchester, 1989.
10. A. Heuerding, G. Jäger, S. Schwendimann, and M. Seyfried. Propositional logics on the computer. In P. Baumgartner, R. Hähnle, and J. Posegga, editors, *Theorem Proving with Analytic Tableaux and Related Methods*, LNCS 918, pages 310–323, 1995.
11. A. Heuerding, M. Seyfried, and H. Zimmermann. Efficient loop-check for backward proof search in some non-classical propositional logics. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Tableaux 96*, LNCS 1071, pages 210–225, 1996.
12. Y. Kesten, Z. Manna, H. McGuire, and A.Pnueli. A decision algorithm for full propositional temporal logic. In *Computer Aided Verification*, LNCS 697, pages 4–35. Springer, 1993.
13. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer Verlag, 1991.
14. K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
15. A. Sistla and E. Clarke. The complexity of propositional linear temporal logic. *Journal of the Association for Computing Machinery*, 32(3):733–749, 1985.
16. R. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal Computing*, 1(2):146–160, 1972.
17. P. Wolper. The tableau method for temporal logic: an overview. *Logique et Analyse*, 110-111:119–136, 1985.

Stefan Schwendimann. A New One-Pass Tableau Calculus for PLTL. In *Proceedings, TABLEAUX'98, Oisterwijk, The Netherlands*, LNAI 1397, pages 277 – 291, Springer 1998