

Model Theoretic Syntax and Parsing: An Application to Temporal Logic

Adi Palm¹

*Department of General Linguistics
University of Passau
94030 Passau, Germany*

Abstract

In general, model-theoretic approaches to syntax aim to describe the syntactic structures of natural languages by means of logical formulae so that the structures are models of these formulae. However, in practice, the use of such descriptions for processing issues involves so much complexity, that they appear to be more or less useless for such tasks. We present a simple formalism, which is based on temporal logic, to specify sets of labeled binary trees. Then we show how to employ such temporal formulae in an Earley-style parsing formalism using labeled tableau methods. This may reduce the complexity, since the resulting parsing formalism only considers the models whose leaves represent the input string.

1 Introduction

Inspired by constraint-based and principle-based grammar formalisms, several model-theoretic approaches [8] to syntax have attracted attention over the last decade. Insofar as they differ from one another, they do so in their underlying logical framework; for instance Kracht [19] employs propositional dynamic logic, Palm [21,22] temporal logic and a fragment of propositional dynamic logic, Blackburn et al. [3] modal logic, Rogers [24] weak monadic second order logic. Such a formalism considers syntactic structures to be the models of a particular formula specifying the syntactic properties of a language. Typically, these structures are ordered trees whose nodes are labeled with symbols or propositions of a finite domain.

Although these formalisms tell us a lot about the properties and the nature of such trees, we do not hear much about operational aspects like parsing or generation. In the approach presented here, we introduce a parsing method

¹ Email: palm@uni-passau.de

that employs a logical description of trees as its grammatical base. The underlying logical framework is a particular version of propositional branching time logic, since this formalism already provides tree-shaped models. Although the generative power of temporal logic lies somewhere between dynamic and modal logic, it should be sufficient for most linguistic applications to stay within the scope of context-free languages. On the basis of this formalism, we construct a tableau calculus that builds valid tree models for a given formula. Afterwards, we present an Earley-based parsing formalism for temporal formulae that employs this tableau calculus to dynamically generate the grammatical information which is necessary for parsing an input string.

2 A Propositional Temporal Logic for Binary Trees

According to its basic nature, branching temporal logic provides a canonical method to describe certain sets of trees. In such a framework the notion of time is organized into discrete points of time where each of them owns alternative possible successors. If we further assume that branching time lines never join together and that there is a unique starting point, we obviously obtain tree-like structures. As a common application, such trees are considered to be the alternative computations of finite-state concurrent systems, leading to *computation tree logic* CTL [5,6] and related formalisms. However, we must take into account the basic difference between CTL structures and the trees considered here: the number of tree nodes is finite and there is a linear order imposed on the children of a node. Further, we restrict our attention to binary trees, since binary constituent structures typically occur in natural languages. Thus, they are commonly used by natural language grammar formalisms e.g. Head-driven Phrase Structure Grammar (HPSG), Government and Binding (GB), Minimalist Program (MP), Categorical Grammar (CG).

Looked at in formal terms CTL and related formalisms follow Arthur Prior's [23] view, where temporal logic is used as a particular version of modal logic which is more powerful than ordinary modal logic but which is also properly embedded in propositional dynamic logic. Moreover, most versions of computation tree logic distinguish between paths and nodes, which is essential for specifying certain properties of (infinitely running) concurrent systems. In the framework considered here, the situation is less complex, since due to the finiteness we could show that each leaf in a finite tree uniquely represents a path and, hence, a path might be viewed as a leaf with particular properties. Hence the conditions on path can be transformed into conditions on tree nodes. As a consequence the syntax used for our formalism resembles more propositional linear time logic, despite of the fact that we actually describe branching structures.

Given this view of trees, we only require the modalities that access to the next immediate temporal states and the until-operator. For a binary tree, we need two next-operators accessing the next left and the next right state.

Correspondingly, our language of propositional binary branching temporal logic $PBTL_2$ employs three basic modalities: the two operators \mathcal{O}_1 and \mathcal{O}_2 , which access the first (left) and second (right) successor, and the until-operator \mathcal{U} that access to a sequence of nodes until a certain condition becomes true.

Definition 2.1 (*Syntax of $PBTL_2$*) Given a finite set of propositions $\mathcal{P} = \{p, q, r, \dots\}$, the set of $PBTL_2$ formulae (over \mathcal{P}) is defined as follows:

$$WFF := \top \mid \perp \mid p \mid (\varphi) \mid \neg\varphi \mid \varphi \wedge \psi \mid \mathcal{O}_1\varphi \mid \mathcal{O}_2\varphi \mid \psi\mathcal{U}\varphi$$

Based on these operators, we can define the Boolean connectives \vee , \rightarrow , \leftrightarrow in the usual way and, in addition, the temporal operators *next* $\mathcal{O}\varphi \equiv (\mathcal{O}_1\varphi) \vee (\mathcal{O}_2\varphi)$, *eventually* $\diamond\varphi \equiv \top\mathcal{U}\varphi$ and *henceforth* $\square\varphi \equiv \neg(\top\mathcal{U}\neg\varphi)$. As mentioned above the underlying structures of $PBTL_2$ consist of ordered finite binary trees, where each node is labeled with a set of propositions of a given set \mathcal{P} . In contrast to standard temporal logic, we assume that every branch of the tree ends at a leaf node and includes at most a finite number of nodes. We represent such a structure as a finite subset $t \subseteq \{1, 2\}^*$ which is closed under prefixes, i.e. for all $w.a \in t$ with $w \in \{1, 2\}^*$ and $a \in \{1, 2\}$ we have $w \in t$, where ε denotes the root. In addition, a right successor always requires a corresponding left successor, i.e. for all $w.2 \in t$ there is $w.1 \in t$. We interpret $PBTL_2$ formulae over triples (t, h, n) representing nodes $n \in t$ in a labeled binary tree domain $T = (t, h)$ where $t \subseteq \{1, 2\}^*$ is a tree domain and $h : t \rightarrow 2^{\mathcal{P}}$ is a labeling function:

Definition 2.2 (*Semantics of $PBTL_2$*)

- $(t, h, n) \models \top$ and $(t, h, n) \not\models \perp$.
- $(t, h, n) \models p$ iff $p \in h(n)$ for every proposition $p \in \mathcal{P}$.
- $(t, h, n) \models \neg\varphi$ iff $(t, h, n) \not\models \varphi$.
- $(t, h, n) \models \varphi \wedge \psi$ iff $(t, h, n) \models \varphi$ and $(t, h, n) \models \psi$.
- $(t, h, n) \models \mathcal{O}_1\varphi$ iff $n.1 \in t$ and $(t, h, n.1) \models \varphi$.
- $(t, h, n) \models \mathcal{O}_2\varphi$ iff $n.1, n.2 \in t$ and $(t, h, n.2) \models \varphi$.
- $(t, h, n) \models \psi\mathcal{U}\varphi$ iff there is $a = a_1 \cdots a_k \in \{1, 2\}^k$ for some $k \geq 0$ such that $(t, h, na) \models \varphi$ and $(t, h, na_1 \cdots a_i) \models \psi$ for all $0 < i < k$.

The tree $T = (t, h)$ satisfies a $PBTL_2$ formula φ if and only if the root of T satisfies φ , i.e. $(t, h, \varepsilon) \models \varphi$. As an alternative to this root-based semantic of $(t, n) \models \varphi$, we could employ the universal interpretation, where all nodes n of (t, h) must satisfy φ but which would be equivalent to $(t, h) \models \square\varphi$. For our purpose we prefer the root-based semantic since this interpretation enables us to construct trees starting from the root, while separating and analyzing a formula.

Another aspect we have not taken into account so far concerns the expressive power of $PBTL_2$. In temporal terms, this formalism only considers the ‘future’ of a state but ignores its past. Nevertheless, there are some re-

sults concerning the strong correspondence that holds between temporal logic, past temporal logic and first-order logic. For the linear case this can be obtained from [18,14] and for binary trees from [17]. If this were applied to $PBTL_2$, additional past operators would not increase the overall expressivity. For instance, if we want to state that all nodes satisfying some condition φ must have a parent satisfying ψ , we could employ the non-past formulation $\neg\varphi \wedge \Box\varphi(\bigcirc\varphi \rightarrow \psi)$. Accordingly, the models of $PBTL_2$ are the set of binary trees described by a corresponding version of first-order logic on binary trees.

3 A Tableau-based Construction of Tree Models

Now we introduce a method that generates the tree models of a given $PBTL_2$ formula. Our solution follows similar approaches to constructing corresponding tree automata [10,2,27]. Unfortunately, these approaches employ variants of alternating tree automata for infinite trees, which allow linear time model checking, but require in general exponential time for generating at least one tree. Therefore, we employ a tableau-based method that constructs plain tree models which relates to similar methods in the CTL framework [1,11,13,28,20]. Typically, a tableau calculus denotes a proof method, however, it can also be viewed as a formalism that constructs models for a given formula. Within a modal framework we can extend the tableau formulae with certain labels that indicate the underlying reachability relation. Consequently, such a labeled tableau system may provide the models of a given formula. The construction fails if all branches of the tableau are closed. But every branch that is not closed represents a model of the underlying formula where the labels of the formulae occurring in this branch represent the internal structure of that model. Afterwards, we use this way of constructing tree models to handle the parsing actions of a modified Earley-parser for $PBTL_2$.

In formal terms, a tableau calculus denotes a system of rules that extends a given set of formulae in a particular manner. The result is a tree-like structure, where each branch denotes a certain alternative. For a detailed discussion of tableau methods in a modal or a temporal framework, we refer to [4,15]. A tableau rule with the name (α) consists of a premise \mathcal{P} and a finite number of alternative conclusions $\mathcal{C}_1 | \dots | \mathcal{C}_k$ written as

$$(\alpha) \frac{\mathcal{P}}{\mathcal{C}_1 | \dots | \mathcal{C}_k}$$

In our approach, the premise consists of a single labeled formula, while the conclusions are sets of labeled formulae. We read a tableau rule downward, i.e. “if the premise is satisfiable then so is at least one of the conclusions”. A finite set of tableau rules, which are identified by their rule names, denotes a *tableau calculus* \mathcal{TC} . Within our approach, a given non-atomic formula matches exactly the premise of a single tableau rule. Therefore no other tableau rule can be applied to this formula. For a given finite set of formulae $X = \{\varphi_1, \dots, \varphi_k\}$ we construct the tableau tree $\mathcal{TC}(X)$ according to the

following steps:

- Each tree node represents a single formula, where we arrange the initial formulae $\varepsilon::\varphi_1, \dots, \varepsilon::\varphi_k$ in an arbitrary vertical linear sequence and mark them as active nodes.
- Atomic formulae $\sigma::p$ and $\sigma::\neg p$ for some $p \in \mathcal{P}$ are always inactive and can never be activated.
- If an active formula $\sigma::\varphi$ in an open branch \mathcal{B} matches the premise of a rule, then we add its conclusions as active nodes at the end of the branch so that a rule with n conclusions leads to an n branching node. We only add labeled formulae that do not already occur on this branch, except this would rule out a certain alternative. In this case we still add this formula but mark it as inactive [16].
- After we handled all open branches of the active formula under consideration, this formula becomes inactive.
- If we add $\sigma::\perp$ to any branch or if we add $\sigma::\neg\psi$ to some branch including $\sigma::\psi$, then the branch under consideration is marked as closed. A node also becomes inactive if all branches including it are closed.

A tableau is closed if all branches are closed. A branch is finished if either it is closed or no rule can be applied to one of its formulae (without introducing new formulae on this branch). The tableau is finished if all branches are finished. To express the immediate dominance relation of the tree models, we associate each formula with a particular label σ where the labels $\sigma.1$ and $\sigma.2$ indicate the left and the right successors of a node. These labels distinguish formulae that are assigned to different tree nodes and they express how the tableau formulae are structurally related to each other.

To construct the tableau rules, we reconsider the (intended) semantics of the $PBTL_2$ formulae. Obviously, the rules should handle each type of formula uniquely by separating it into corresponding subformulae obeying the particular structure of binary trees and the formal properties of temporal and modal logic. For each operator, except the one for negation, we consider the original positive version and its negated counterpart. Thus, we must handle the negation operators only for the propositions in \mathcal{P} , the Boolean constants \top and \perp and the atomic next-operators $\mathcal{O}_1\top$ and $\mathcal{O}_2\top$:

$$\begin{array}{ccc}
 (\wedge) \frac{\sigma::\varphi \wedge \psi}{\sigma::\varphi, \psi} & (\neg\wedge) \frac{\sigma::\neg(\varphi \wedge \psi)}{\sigma::\neg\varphi \mid \sigma::\neg\psi} & (\neg) \frac{\sigma::\neg\neg\varphi}{\sigma::\varphi} \\
 \\
 (\mathcal{O}_1) \frac{\sigma::\mathcal{O}_1\varphi}{\sigma::\neg leaf_1} & (\mathcal{O}_2) \frac{\sigma::\mathcal{O}_2\varphi}{\sigma::\neg leaf_1, \neg leaf_2} & \\
 \sigma.1::\varphi & \sigma.2::\varphi &
 \end{array}$$

true if $\neg\psi$, $\neg\varphi$ and ψ , $\neg\varphi$, $leaf_1$, $leaf_2$ are inconsistent. Correspondingly, we add a further closing condition to our tableau formalism which ensures that the branch including $\sigma::\neg(\psi\mathcal{U}\varphi)$ closes in this case.

Theorem 3.1 *\mathcal{TC} is complete, i.e. for every finite binary tree (t, h) and every $PBTL_2$ formula φ it is true that:*

$$(t, h) \in T_{\mathcal{TC}}(\varphi) \quad \text{iff} \quad (t, h) \models \varphi$$

Proof (Sketch) By the way of constructing the rules presented above, we obtain, more or less obviously, the conclusion that every finite binary tree $(t, h) \in T_{\mathcal{TC}}(\varphi)$ satisfies the formula φ . However, we need more effort to establish the opposite direction, claiming that every finite binary tree satisfying some $PBTL_2$ formula φ can be obtained by this tableau calculus. We must show that for every finite binary tree $T = (t, h)$ satisfying φ there is an open branch $\mathcal{B}_T \in \mathcal{TC}(\varphi)$ such that \mathcal{B}_T represents (t, h) . By induction on the tree structure we show that we can construct an open branch \mathcal{B}_T , where all nodes n in T meet the following conditions:

- (i) If $p \in h(n)$ then $n::\neg p \notin \mathcal{B}_T$.
- (ii) If $p \notin h(n)$ then $n::p \notin \mathcal{B}_T$.
- (iii) If n is not in \mathcal{B}_T then there is an ancestor n_p such that n_p is an Ω -node, i.e. $n_p::a \notin \mathcal{B}_T$ for any $a \in \{leaf_1, \neg leaf_1, leaf_2, \neg leaf_2\}$

The last condition indicates the case where the structure below n_p is arbitrary. For the induction we assume an unfinished tableau where we extend \mathcal{B} in an appropriate manner by employing tableau rules. We start the induction by considering the case $(t, h, \varepsilon) \models \varphi$. Since φ has a model and our tableau calculus is correct, $\mathcal{TC}(\varphi)$ includes at least one open branch. Due to the initialization, every branch of the tableau $\mathcal{TC}(\varphi)$ starts with $\varepsilon::\varphi$. Thus, as demanded, there is at least one open branch including $\varepsilon::\varphi$. Now we consider some nodes $n = a_1 \dots a_{i+1}$ in T with $a_1, \dots, a_{i+1} \in \{1, 2\}$. If n_p is a valid label in \mathcal{B} , we show that we can expand \mathcal{B} by employing individual tableau rules so that it includes the appropriate formulae wearing the label n . Otherwise, if n_p is absent in \mathcal{B} , it must meet the condition (iii) and so does n and any other descendant of n_p . Now we show how to determine the appropriate n -labeled formulae, but only for the case, where $n = n_p.1$ and $n_p::\rho\mathcal{U}\psi$. The proof for $n = n_p.2$ and the remaining formulae is similar. From the tableau construction follows that $\varepsilon::\varphi$ entails $n_p::\rho\mathcal{U}\psi$ and consequently $(t, h, n_p) \models \rho\mathcal{U}\psi$. Now we can expand \mathcal{B} in three alternative ways using the \mathcal{U} -rule. The first alternative $n_p::\psi$ makes no statements concerning n . Thus, if no other formula adds a (negated) auxiliary label, n_p is an Ω -node. The second one adds $n::\rho\mathcal{U}\psi$ and the third one implicitly adds $n::\top$. According to the semantics of $PBTL_2$ as specified in Def. 2.2, n_p must satisfy ψ or it must have a successor $n_p.a$ with $a \in \{1, 2\}$ that satisfies ρ and $\rho\mathcal{U}\psi$. If the first one is true, we select the first alternative. Otherwise we select the second or the third alternative, if $a = 1$ or $a = 2$, respectively. Afterwards it could be necessary to employ

some propositional rules (\wedge), (\vee), (\neg). However their application would never close the branch \mathcal{B} . Consequently, the node n meets the conditions we claimed above for the resulting expanded branch. \square

4 An Earley-Parser for $PBTL_2$

The tableau formalism introduced above can be employed to construct all binary trees that are models for a given $PBTL_2$ formula. It is further possible to create a tree automaton generating this set of trees, where the states $q \in Q$ correspond to sets of formulae wearing the same label and where the transition relation $\delta \subseteq Q \times 2^{\mathcal{P}} \times Q \times Q$ can be obtained as follows:

$$\delta(q, S, q_1, q_2) \text{ iff } q = \sigma(\mathcal{B}) \wedge q_1 = \sigma.1(\mathcal{B}) \wedge q_2 = \sigma.2(\mathcal{B}) \wedge S \cup \sigma(\mathcal{B}) \not\vdash \perp$$

for some tableau branch \mathcal{B} and some label σ where $\sigma(\mathcal{B}) := \{\varphi \mid \sigma :: \varphi \in \mathcal{B}\}$ and $S \cup \sigma(\mathcal{B}) \not\vdash \perp$ expresses that the propositions $p \in S$ are consistent with $\sigma(\mathcal{B})$. However, the number of states might be exponential with respect to the size of the underlying $PBTL_2$ formulae, and so might be the resulting tree automaton. Although when a tree automaton and, hence, the corresponding context-free grammar is used, parsing an input sentence requires, at most, cubic time with respect to the length of the input, this time could be exponential with respect to the length of the underlying $PBTL_2$ formula. We try to avoid this exponential growth by selecting only those parts of the transitions that are necessary for parsing a given input sentence. Nevertheless, in general, exponential growth cannot be avoided. Such cases occur if the input imposes only a weak restriction on the possible structures. For instance, the formula $\Box(p_1 \vee \dots \vee p_k)$ with $\mathcal{P} = \{p_1, \dots, p_k\}$ accepts any input string, but it has k alternatives for each node. Since $PBTL_2$ works downward in a tree, we employ a top-down parsing formalism, namely an Earley-parser [9]. Instead of employing the rules of a context-free grammar, we dynamically generate triples of sets of consistent $PBTL_2$ formulae using the tableau formalism defined above. For a given $PBTL_2$ formula φ we assume to have the whole finished tableau $\mathcal{TC}(X)$, but actually we only dynamically generate the required parts of it.

Now let A be a set of all $PBTL_2$ formulae wearing the same label σ in some branch of $\mathcal{TC}(X)$, where $X = \{\varphi_1, \dots, \varphi_m\}$ is the given finite set of $PBTL_2$ formulae. Then, by applying the \mathcal{TC} -rules to A , we can construct alternative pairs of sets B and C , representing the left and right successor of A . Provided that A is \mathcal{TC} -saturated, i.e. no more \mathcal{TC} -rules can be applied to it, we do not need any information concerning the branch or the label σ , and the sets B and C are \mathcal{TC} -saturated, too. For an absent successor, the corresponding set B or C is empty where the auxiliary propositions $leaf_1$ and $leaf_2$ must be set accordingly. If A denotes a proposition, i.e. no temporal operator occurs and, hence, no auxiliary proposition, the structure below it is arbitrary, which is expressed by the special symbol Ω ; thus, we set $B = C = \Omega$. Hence, if A , B and C meet these conditions, we write $\langle A, B, C \rangle \in \mathcal{TC}(X)$. Moreover,

$root(\mathcal{TC}(X))$ denotes the set of initial sets which are the \mathcal{TC} -saturated sets of formulae wearing the label ε .

Basically, an Earley-type parser works on a set of parsing items. Within our approach a parsing item $\langle p, A, B, C, i, j \rangle$ states that we have recognized the part $w_{i+1} \dots w_j$ of the input string $w = w_1 \dots w_n$ (with $0 \leq i \leq j \leq n$) for a subtree with the root A and the left and right successor B and C , respectively. The position $p \in \{l, m, r\}$ indicates the position of the ‘dot’: $p = l$, the left position, i.e. no part of the input string was recognized; $p = m$, the middle, i.e. the part below B has been recognized, but the part below C still needs to be recognized; $p = r$, i.e. the right position, the whole subtree has been recognized successfully. The set of parsing items I for an input string $w_1 \dots w_n$ is built by the following Earley-parsing operations:

- *Initialize*
For all $\langle A, B, C \rangle \in root(\mathcal{TC}(X))$ let $\langle l, A, B, C, 0, 0 \rangle \in I$
- *Expand*
If $\langle l, A, B, C, i, i \rangle \in I$ then $\langle l, B, B_1, B_2, i, i \rangle \in I$
for each $\langle B, B_1, B_2 \rangle \in \mathcal{TC}(X)$
If $\langle m, A, B, C, i, j \rangle \in I$ then $\langle l, C, C_1, C_2, j, j \rangle \in I$
for each $\langle C, C_1, C_2 \rangle \in \mathcal{TC}(X)$
- *Shift*
If $\langle l, A, \emptyset, \emptyset, i, i \rangle \in I$ and $w_i \models A$ then $\langle r, A, \emptyset, \emptyset, i, i+1 \rangle \in I$.
If $\langle m, A, B, \emptyset, i, j \rangle \in I$ then $\langle r, A, B, \emptyset, i, j \rangle \in I$.
If $\langle l, A, \Omega, \Omega, i, i \rangle \in I$ then $\langle r, A, \Omega, \Omega, i, k \rangle \in I$ for $i \leq k \leq n$
- *Complete*
If $\langle l, A, B, C, i, i \rangle, \langle r, B, B_1, B_2, i, j \rangle \in I$
then let $\langle m, A, B, C, i, j \rangle \in I$
If $\langle m, A, B, C, i, j \rangle, \langle r, C, C_1, C_2, j, k \rangle \in I$
then let $\langle r, A, B, C, i, k \rangle \in I$
- *Accept*
There is $\langle A, B, C \rangle \in root(\mathcal{TC}(X))$ with $\langle r, A, B, C, 0, n \rangle \in I$

The operation *Initialize* introduces the root properties to the item set. Next we recursively employ the expand operation to predict the possible left descendants of the root. After there is no further possible prediction, we read the first input symbol using the shift operation. A successful shift moves the ‘dot’ from the left to the right position in the corresponding item. Moreover, every shift possibly triggers a complete operation which completes an item predicted previously. In addition, this can also trigger further complete operations. Likewise, we repeat this scheme of expand, shift and complete operations until the whole input string has been proceeded. We accept the input string if I includes the item $\langle r, A, B, C, 0, n \rangle$ for some $\langle A, B, C \rangle \in root(\mathcal{TC}(X))$. Actually, each item $\langle r, A, B, C, i, j \rangle \in I$ describes a set of (sub-)trees $T_I(A, i, j)$ over $w_{i+1} \dots w_j$, where the root satisfies A . This set can be obtained as follows:

$$\begin{aligned}
T_I(i, k, A) = & \{\tau[A, \emptyset, \emptyset] \mid \langle r, A, \emptyset, \emptyset, i, k \rangle \in I\} \cup \\
& \{\tau[A, \Omega, \Omega] \mid \langle r, A, \Omega, \Omega, i, k \rangle \in I\} \cup \\
& \{\tau[A, T_I(B, i, k)] \mid \langle r, A, B, \emptyset, i, k \rangle \in I\} \cup \\
& \{\tau[A, T_I(B, i, j), T_I(C, j, k)] \mid \langle r, A, B, C, i, k \rangle \in I, i < j < k\}
\end{aligned}$$

where the tree constructor $\tau[A, t_1, t_2]$ generates a tree such that the root wears the label A and has the left and right subtrees t_1 and t_2 , respectively. Moreover, $\tau[A, B, \emptyset]$ denotes a single branching root, $\tau[A, \emptyset, \emptyset]$ denotes a leaf, and $\tau[A, \Omega, \Omega]$ denotes a an Ω -tree. Consequently, $T_I(A, 0, n)$ is the set of parse trees for the input string $w_1 \cdots w_n$.

Our parsing method combines tableau resolution with a standard Earley-parser using forward chaining techniques. To establish the intended behaviour we must show that the parser is correct, i.e. every resulting parse tree (t, h) satisfies X , and more important, the parser is complete, i.e. it generates every tree that matches the input string and that satisfies the underlying tree formulae.

Theorem 4.1 *The Earley-based parser for $PBTL_2$ is correct and complete.*

Proof (Sketch) Since the expand operation employs the tableau rules, it only predicts parts of the tree that occur in the corresponding branch of the tableau $\mathcal{TC}(X)$. However, each parse tree consists of completed items that were predicted previously, so it must be in $T_{\mathcal{TC}}(X)$ and, according to Theorem 3.1, it satisfies φ . For the completeness we employ the method presented by Sikkel [25] that shows by induction that the parser can generate every item $\langle p, A, B, C, i, j \rangle$ that corresponds to tree node n . In our case this node means some label σ with $A = \sigma(\mathcal{B})$, $B = \sigma.1(\mathcal{B})$, $C = \sigma(\mathcal{B})$ for some branch $\mathcal{B} \in \mathcal{TC}(X)$ where the leaves of \mathcal{B} match the input string w . Basically, we must determine the minimal number $d(\langle p, A, B, C, i, j \rangle)$ of parsing operations that are necessary to generate this item. Then, by induction on this number, any item generated by k minimal operations is generated by items requiring less than k operations. To employ this method we must carry out some modifications. One concerns the number of context-free derivations, the second one concerns the use of context-free rules, since either of these issues does not directly match our formalism. Obviously, the number of derivation steps s in a context-free derivation $A \Rightarrow^s w_i \dots w_j$ is equal to the number of non-leaf nodes in the corresponding subtree. Moreover we can consider any triple $\langle A, B, C \rangle \in \mathcal{TC}(\varphi)$ as a context-free rule $A \rightarrow BC$. When using these modifications, we can apply the proof of completeness presented in [25] to our approach. \square

Finally, we make some remarks concerning the complexity of our parsing formalism for $PBTL_2$ formulae. Obviously, the Earley-based part requires at most cubic time with respect to the length of the input string. However, the crucial value results from the number of triples $\langle A, B, C \rangle \in \mathcal{TC}(X)$ provided by the tableau. Their number is at most exponential with respect to the length

of the given formula, since the sets A, B, C correspond to subsets of the set of subformulae obtained from the given formula. Once again we consider the tableau as a tree-automaton, that includes an exponential number of states q_A in the worst case. Thus it is possible to consider an exponential number of triples $\langle A, B, C \rangle$ while parsing an input string. Now we can interpret the resulting set of parsing items as the result obtained by intersecting the tree automaton for the grammar and the tree automaton generating all trees over the given input string w , which has been already suggested in [7]. For the second tree automaton, we only require states for each pair (i, j) with $0 \leq i < j \leq n$ so that a state $q_{i,j}$ spans over the section $w_{i+1} \dots w_j$ of the input string. The number of such states is bound by n^2 . After intersecting both tree automata, we obtain states consisting of pairs of states q_A and $q_{i,j}$, which correspond to the parsing items. However, the exponential worst case scenario remains. But this may not apply to every input string and every underlying $PBTL_2$ formula φ . For instance, if φ includes strong structural restrictions that only allow a small number of structures over (a part of) the input string, then only a small section of the states must be considered. In general, we can approximately state that the more structural restrictions we have, the less is the complexity and the less is the number of possible valid trees for some input string. However, in natural language we commonly obtain only a small number of possible syntactic structures for an input sentence. Thus one could expect a complexity far below exponential for such applications.

5 Concluding Remarks

We have sketched an Earley-like parsing formalism that is based on a model-theoretic description of syntax by means of a particular version of branching time logic. Although this approach makes use of the strong relationship between branching time and top-down parsing, this is not a general limitation. Obviously, one might capture other methods like head-corner parsing by introducing appropriate temporal modalities.

For evaluation purposes, some parts of this approach have been implemented in Prolog. However, the results we have obtained so far are too weak to make any general statements in this respect. Nevertheless, the simplicity of the Earley-formalism presented above offers several points of departure for improving time and space complexity. Another area of improvement would involve combing it with (partially) lexicalized grammars, where lexical structures are given as additional formulae that can be included in the expand operation, which must be adopted correspondingly.

Another way improving the complexity may result from restricting the syntax of $PBTL_2$ as proposed in [12] for the case of CTL. However such a restriction includes a maximal number of nested temporal modalities. Therefore we must check whether such a constraint also applies to the syntax of natural languages. Nevertheless this is in general true for the examples proposed in

the model-theoretic approach mentioned in [19,3,24,21,22] .

References

- [1] Ben-Ari, M., Z. Manna and A. Pnueli, *The temporal logic of branching time*, in: *Proceedings of the 8th ACM Symposium on Principles of Programming Languages*, 1981, pp. 164–176.
- [2] Bernholtz, O., M. Vardi and P. Wolper, *An automata-theoretic approach to branching-time model checking*, in: D. Dill, editor, *Computer Aided Verification, Proc. 6th Int. Conference, Stanford CA, June 1994*, LNCS **818** (1994), pp. 142–155.
- [3] Blackburn, P., W. Meyer-Viol and M. de Rijke, *A proof system for finite trees*, in: H. K. Büning, editor, *Computer Science Logic*, LNCS **1092** (1996), pp. 86–105.
- [4] Catach, L., *TABLEAUX: a general theorem prover for modal logics*, *Journal of Automated Reasoning* **7** (1991), pp. 489–510.
- [5] Clarke, E. and E. Emerson, *Designs and synthesis of synchronization skeletons using branching time temporal logic*, in: *Proc. Logic of Programs Workshop*, LNCS **131** (1981), pp. 52–71.
- [6] Clarke, E., E. Emerson and A. Sistla, *Automatic verification of finite-state concurrent systems using temporal logic specifications*, *ACM Trans. Programming Languages Systems* **8** (2) (1986), pp. 244–263.
- [7] Cornell, T., *Parsing and grammar engineering with tree automata*, in: D. Heylen, A. Nijholt and G. Scollo, editors, *Algebraic Methods in Language Processing AMoLP 2000*, Iowa City, Iowa, 2000, pp. 267–274.
- [8] Cornell, T. and J. Rogers, *Modell theoretic syntax*, in: L. L.-S. Cheng and R. Sybesma, editors, *The Glot International State of the Article Book 1*, *Studies in Generative Grammar* **48**, Mouton de Gruyter, Berlin, 2000 pp. 101–125.
- [9] Earley, R., *An efficient context-free parsing algorithm*, *Communications of the ACM* **13** (1970).
- [10] Emerson, E., *Automata, tableaux and temporal logic*, in: R. Parikh, editor, *Proceedings Conference on Logic of Programs*, LNCS **193** (1985), pp. 79–88.
- [11] Emerson, E. and J. Halpern, *Decision procedures and expressiveness in the temporal logic of branching time*, in: *Proceedings of the 14th ACM Symposium on Computing*, 1982, pp. 169–180.
- [12] Emerson, E., T. Sadler and J. Srinivasan, *Efficient temporal satisfiability*, *Journal of Logic and Computation* **2** (1992), pp. 173–210.
- [13] Emerson, E. and A. Sistla, *Deciding branching time logic*, *Information and Control* **61** (1984), pp. 175–201.

- [14] Gabby, D., A. Pnueli, A. Shelah and J. Stavi, *The temporal analysis of fairness*, in: *Proceedings of 7th Ann. ACM Sympos. on Principle on Programming Languages*, 1980, pp. 163–167.
- [15] Goré, R., *Tableau methods for modal and temporal logic*, in: M. D’Augustino, D. Gabbay, R. Hähnle and J. Posegga, editors, *Handbook of Tableau Methods*, Kluwer, Dordrecht, 1999 pp. 297–396.
- [16] Goré, R., “Errata and Addendum for Tableau Methods for Modal and Temporal Logic,” Automated Reasoning Project and Department of Computer Science, Australian National University, Canberra, 2000.
URL <http://arp.anu.edu.au/~rpg/publications.html>
- [17] Hafer, T. and W. Thomas, *Computation tree logic CTL* and path quantifiers in the monadic theory of the binary trees*, in: T. Ottmann, editor, *Automata, Languages and Programming*, LNCS **267** (1987), pp. 269–279.
- [18] Kamp, J., “Tense Logic and the Theory of Linear Order,” Ph.D. thesis, University of California, Los Angeles (1968).
- [19] Kracht, M., *Syntactic codes and grammar refinement*, *Journal of Logic, Language and Information* **4** (1995), pp. 41–60.
- [20] Lichtenstein, O. and A. Pnueli, *Propositional temporal logics: Decidability and completeness*, *Logic Journal of the IGPL* **8** (2000), pp. 55–85.
- [21] Palm, A., *The expressivity of tree languages for syntactic structures*, in: H.-P. Kolb and U. Mönnich, editors, *The Mathematics of Syntactic Structures*, De Gruyter, Berlin, 1999 pp. 50–90.
- [22] Palm, A., *Propositional tense logic for finite trees*, in: *Proceedings of 6th Meeting on Mathematics of Language (MOL6)*, 1999, pp. 285–300.
- [23] Prior, A., “Past, Present and Future,” Oxford University Press, 1967.
- [24] Rogers, J., “A Descriptive Approach to Language-Theoretic Complexity,” CSLI Publication, Stanford, CA., 1998.
- [25] Sikkel, K., *Parsing schemata and correctness of parsing algorithms*, *Theoretical Computer Science* **199** (1997), pp. 87–103.
- [26] Smorynski, C., “Self Reference and Modal Logic,” Springer Verlag, Berlin Heidelberg New York, 1985.
- [27] Vardi, M., *Alternating automata: Unifying truth and validity checking for temporal logics*, in: W. McCune, editor, *Proceedings of the 14th International Conference on Automated deduction*, LNAI **1249** (1997), pp. 191–206.
URL <http://www.cs.rice.edu/~vardi/papers/cade97.ps.gz>
- [28] Wolper, P., *The tableau method for temporal logic*, *Logique et Analyse* **28** (1985), pp. 119–136.