Tree languages defined in first-order logic with one quantifier alternation

Mikolaj Bojańczyk, Luc Segoufin

Tree languages defined in first-order logic with one quantifier alternation

Mikołaj Bojańczyk, Luc Segoufin
Warsaw University, INRIA - LSV

March 9, 2010

Abstract

We study tree languages that can be defined in Δ_2. These are tree languages definable by a first-order formula whose quantifier prefix is $\exists^*\forall^*$, and simultaneously by a first-order formula whose quantifier prefix is $\forall^*\exists^*$. For the quantifier free part we consider two signatures, either the descendant relation alone or together with the lexicographical order relation on nodes. We provide an effective characterization of tree and forest languages definable in Δ_2. This characterization is in terms of algebraic equations. Over words, the class of word languages definable in Δ_2 forms a robust class, which was given an effective algebraic characterization by Pin and Weil [11].

1 Introduction

We say a logic $\mathcal{L}_1$ has a decidable characterization inside a logic $\mathcal{L}_2$ if the following decision problem is decidable: "given as input a formula of the logic $\mathcal{L}_2$, decide if it is equivalent to some formula of the logic $\mathcal{L}_1$". We are interested in the case when the logic $\mathcal{L}_2$ is MSO on words or trees, and $\mathcal{L}_1$ represents some fragment of $\mathcal{L}_2$.

This type of problem has been successfully studied in the case when $\mathcal{L}_2$ is MSO on finite words. In other words $\mathcal{L}_2$, represents the class of regular word languages. Arguably best known is the result of McNaughton, Papert and Schützenberger [13, 9], which says that the following two conditions on a regular word language L are equivalent: a) L can be defined in first-order logic with order and label tests; b) the syntactic semigroup of L does not contain a non-trivial group. Since condition b) can be effectively tested, the above theorem gives a decidable characterization of first-order logic. This result demonstrates the importance of this type of work: a decidable characterization not only gives a better understanding of the logic in question, but it often reveals unexpected connections with algebraic concepts. During several decades of research, decidable characterizations have been found for fragments of first-order logic with restricted quantification and various signatures (typically subsets of the order relation and the successor relation), as well as a large group of temporal logics, see [10] and [17] for references.

An important part of this research has been devoted to the quantifier alternation hierarchy, where each level counts the alterations between $\forall$ and $\exists$ quantifiers in a first-order formula in prenex normal form. The quantifier free part of such a formula is built using a binary predicate $<$ representing the linear order on the word. Formulas that have $n-1$ alternations (and therefore n blocks of quantifiers) are called $\Sigma_n(<)$ if they begin with $\exists$, and $\Pi_n(<)$ if they begin with $\forall$. For instance, the word property "some position has label a" can be defined by a $\Sigma_1(<)$ formula $\exists x.\ a(x)$, while the language "nonempty words with at most two positions that do not have label a" can be defined by the $\Sigma_2(<)$ formula

$$\exists x_1 \exists x_2 \forall y \quad (y \neq x_1 \wedge y \neq x_2) \quad \Rightarrow \quad a(y)\ .$$

A lot of attention has been devoted to analyzing the low levels of the quantifier alternation hierarchy for word languages. The two lowest levels are easy: a word language is definable in

$\Sigma_1(<)$ (resp. $\Pi_1(<)$) if and only if it is closed under inserting (removing) letters. Both properties can be tested in polynomial time based on a recognizing automaton, or semigroup. However, just above $\Sigma_1(<), \Pi_1(<)$, and even before we get to $\Sigma_2(<), \Pi_2(<)$, we already find two important classes of languages. A fundamental result, due to Simon [15], says that a language is defined by a Boolean combination of $\Sigma_1(<)$ formulas if and only if its syntactic monoid is $\mathcal{J}$-trivial. Above the Boolean combination of $\Sigma_1(<)$, we find $\Delta_2(<)$, i.e. languages that can be defined simultaneously in $\Sigma_2(<)$ and $\Pi_2(<)$. As we will describe later on, this class turns out to be surprisingly robust, and it is the focus of this paper. Another fundamental result, due to Pin and Weil [11], says that a regular language is in $\Delta_2(<)$ if and only if its syntactic monoid is in DA. The limit of our knowledge is level $\Sigma_2(<)$: it is decidable if a language can be defined on level $\Sigma_2(<)$ [1, 11], but there are no known decidable characterization for Boolean combinations of $\Sigma_2(<)$, for $\Delta_3(<)$, for $\Sigma_3(<)$, and upwards.

For trees even less is known. No decidable characterization has been found for what is arguably the most important proper subclass of regular tree languages, first-order logic with the descendant relation, despite several attempts. Similarly open are chain logic and the temporal logics CTL, CTL* and PDL. However, there has been some recent progress. In [5], decidable characterizations were presented for some temporal logics, while Benedikt and Segoufin [2] characterized tree languages definable in first-order logic with the successor relation (but without the descendant relation).

This paper is part of a program to understand the expressive power of first-order logic on trees, and the quantifier alternation hierarchy in particular. The idea is to try to understand the low levels of the quantifier alternation hierarchy before taking on full first-order logic (which is contrary to the order in which word languages were analyzed). We focus on two signatures. The first signature contains unary predicates for label tests and the ancestor order on nodes, denoted $<$. The second signature assumes that the trees have an order on siblings, which induces a lexicographical linear order on nodes, denoted $<_{\text{lex}}$. Both signatures generalize the linear order on words. As shown in [4], there is a reasonable notion of concatenation hierarchy for tree languages that corresponds to the quantifier alternation hierarchy. Levels $\Sigma_1(<)$ and $\Pi_1(<)$ are as simple for trees as they are for words. A recent result [7] extends Simon's theorem to trees, by giving a decidable characterization of tree languages definable by a Boolean combination of $\Sigma_1(<)$ formulas, and also a decidable characterization of Boolean combinations of $\Sigma_1(<, <_{\text{lex}})$ formulas. There is no known decidable characterization of tree languages definable in $\Sigma_n(<)$ for $n \geq 2$.

The contribution of this paper is a decidable characterization of tree languages definable in $\Delta_2(<)$, i.e. definable both in $\Sigma_2(<)$ and $\Pi_2(<)$. We also provide a decidable characterization of tree languages definable in $\Delta_2(<, <_{\text{lex}})$.

As we signaled above, for word languages the class $\Delta_2(<)$ is well studied and important, with numerous equivalent characterizations. Among them one can find [11, 16, 14, 8]: a) word languages that can be defined in the temporal logic with operators F and F^{-1}; b) word languages that can be defined by a first-order formula with two variables, but with unlimited quantifier alternations; c) word languages whose syntactic semigroup belongs to the semigroup variety DA; d) word languages recognized by two-way ordered deterministic automata; e) a certain form of "unambiguous" regular expressions.

It is not clear how to extend some of these concepts to trees. Even when natural tree counterparts exist, they are not equivalent. For instance, the temporal logic in a) can be defined for trees—by using operators "in some descendant" and "in some ancestor". This temporal logic was studied in [3], however it was shown to have an expressive power incomparable with that of $\Delta_2(<)$. A characterization of $\Delta_2(<)$ was left as an open problem, one which is solved here.

We provide an algebraic characterization of tree languages definable in $\Delta_2(<)$ and in $\Delta_2(<, <_{\text{lex}})$. This characterization is effectively verifiable if the language is given by a tree automaton. It is easy to see that the word setting can be treated as a special case of the tree setting. Hence our characterization builds on the one over words. However the added complexity of the tree setting makes both formulating the correct condition and generalizing the proof quite nontrivial.

2 Trees forests and languages

In this section, we present some basic definitions regarding trees. We also present the formalism of forest algebra, which is used in our characterizations.

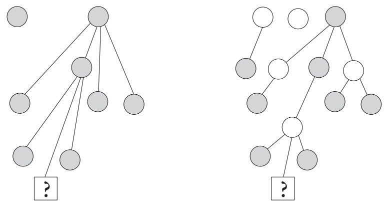
2.1 Trees, forests and contexts

In this paper we work with finite, unranked, ordered trees and forests over a finite alphabet A. Formally, these are expressions defined inductively as follows: If s is a forest and $a \in A$, then as is a tree. If $t_1, \ldots, t_n$ is a finite sequence of trees, then $t_1 + \cdots + t_n$ is a forest. This applies as well to the empty sequence of trees, which is called the *empty forest,* and denoted 0 (and which provides a place for the induction to start). Forests and trees alike will be denoted by the letters $s, t, u, \ldots$ When necessary, we will remark which forests are trees, i.e. contain only one tree in the sequence.

The notion of node, as well as the descendant and ancestor relations are defined in the usual way. We write $x < y$ to say that x is a strict ancestor of y or, equivalently, that y is a strict descendant of x. As usual, we write $x \leq y$ when $x = y$ or $x < y$. The parent of a node x is its immediate ancestor. Two nodes x and y are *siblings* if they have the same parent. We also use the lexicographic order on nodes, written $<_{\text{lex}}$. Recall that $x <_{\text{lex}} y$ holds if either $x < y$, or there are nodes $x' \leq x$ and $y' \leq y$ such that x' is a sibling to the left of y'.

If we take a forest and replace one of the leaves by a special symbol $\Box$, we obtain a *context.* Contexts will be denoted using letters p, q, r. A forest s can be substituted in place of the hole of a context p, the resulting forest is denoted by ps. There is a natural composition operation on contexts: the context qp is formed by replacing the hole of q with p. This operation is associative, and satisfies $(pq)s = p(qs)$ for all forests s and contexts p and q.

We say a forest s is an *immediate piece* of a forest s' if s, s' can be decomposed as $s = pt$ and $s' = pat$ for some context p, some label a, and some forest t. The reflexive transitive closure of the immediate piece relation is called the *piece* relation. We write $s \preceq t$ to say that s is a piece of t. In other words, a piece of t is obtained by removing nodes from t. We extend the notion of piece to contexts. In this case, the hole must be preserved while removing the nodes. The notions of piece for forests and contexts are related, of course. For instance, if p, q are contexts with $p \preceq q$, then $p0 \preceq q0$. Also, conversely, if $s \preceq t$, then there are contexts $p \preceq q$ with $s = p0$ and $t = q0$. (For instance, one can take $p = \Box + s$ and $q = \Box + t$.) The picture below depicts two contexts, the left one being a piece of the right one, as can be seen by removing the white nodes.



We will be considering three types of languages in the paper: *forest languages* i.e. sets of forests, denoted L; *context languages*, i.e. sets of contexts, denoted K, and *tree languages*, i.e. sets of trees, denoted M. Note that a forest language can contain trees.

2.2 Forest algebras

Forest algebras were introduced by Bojańczyk and Walukiewicz as an algebraic formalism for studying regular tree languages [6]. Here we give a brief summary of the definition of these algebras and their important properties. A forest algebra consists of a pair (H, V) of finite monoids, subject to some additional requirements, which we describe below. We write the operation in V multiplicatively and the operation in H additively, although H is not assumed to be commutative. We accordingly denote the identity of V by $\Box$ and that of H by 0. We require that V act on the left of H. That is, there is a map $(h,v) \mapsto vh \in H$ such that $w(vh) = (wv)h$ for all $h \in H$ and $v, w \in V$. We further require that this action be *monoidal,* that is, $\Box \cdot h = h$ for all $h \in H$, and that it be *faithful,* that is, if $vh = wh$ for all $h \in H$, then $v = w$. Finally we require that for every $g \in H$, V contains elements $(\Box + g)$ and $(g + \Box)$ defined by $(\Box + g)h = h + g, (g + \Box)h = g + h$ for all $h \in H$.

A morphism $\alpha : (H_1, V_1) \to (H_2, V_2)$ of forest algebras is actually a pair (α_H, α_V) where α_H is a monoid morphism between H_1 and H_2 and α_V is a monoid morphism between V_1 and V_2, such that $\alpha_H(vh) = \alpha_V(v)\alpha_H(h)$ for all $h \in H$, $v \in V$. However, we will abuse notation slightly and denote both component maps by α.

Let A be a finite alphabet, and let us denote by H_A the set of forests over A, and by V_A the set of contexts over A. Each of these is a monoid, with the operations being forest concatenation and context composition, respectively. The pair (H_A, V_A), with forest substitution as action, forms a forest algebra, which we denote A^Δ.

We say that a forest algebra (H, V) *recognizes* a forest language $L \subseteq H_A$ if there is a morphism $\alpha : A^\Delta \to (H, V)$ and a subset X of H such that $L = \alpha^{-1}(X)$. A forest language is regular, i.e. recognized by any of the many equivalent notions of automata for unranked trees that can be found in the literature, if and only if it is recognized by a finite forest algebra [6].

Given any finite monoid M, there is a number $\omega(M)$ (denoted by ω when M is understood from the context) such that for all elements x of M, x^ω is an idempotent: $x^\omega = x^\omega x^\omega$. Therefore for any forest algebra (H, V) and any element u of V and g of H we will write u^ω and ωg for the corresponding idempotents. The element u^ω is idempotent with respect to the operation in V. The element ωg, which is the same as $(\Box + g)^\omega 0$ and the same as $(g + \Box)^\omega 0$, is idempotent with respect to the operation in H.

Given a forest language $L \subseteq H_A$ we define an equivalence relation $\sim_L$ on H_A by setting $s \sim_L s'$ if and only if for every context $p \in V_A$, the forests ps and ps' are either both in L or both outside of L. We further define an equivalence relation on V_A, also denoted $\sim_L$, by setting $p \sim_L p'$ if for all $s \in H_A$, $ps \sim_L p's$. This pair of equivalence relations defines a congruence of forest algebras on A^Δ, and the quotient (H_L, V_L) is called the *syntactic forest algebra* of L. Each equivalence class of $\sim_L$ is called a *type.*

We now extend the notion of piece to elements of a forest algebra (H, V). The general idea is that a context type $v \in V$ is a piece of a context type $w \in V$ if one can construct a term (using elements of H and V) which evaluates to w, and then take out some parts of this term to get v. Let (H, V) be a forest algebra. We say $v \in V$ *is a piece* of $w \in V$, denoted by $v \preceq w$, if there is an alphabet A such that $\alpha(p) = v$ and $\alpha(q) = w$ hold for some morphism

$$\alpha : A^\Delta \to (H, V)$$

and some contexts $p \preceq q$ over A. The relation $\preceq$ is extended to H by setting $g \preceq h$ if $g = v0$ and $h = w0$ for some context types $v \preceq w$.

3 Logic

The focus of this paper is the expressive power of first-order logic on trees. A forest can be seen as a logical relational structure. The domain of the structure is the set of nodes. (We allow empty domains, which happens when an empty forest 0 is considered.) We consider two different signatures. Both of them contain a unary predicate P_a for each symbol a of the alphabet A, as well as a binary predicate $<$ for the ancestor relation. Furthermore, the second signature also contains a binary predicate $<_{\text{lex}}$ for the lexicographic order on nodes. A formula without free variables over these signatures defines a set of forests, these are the forests where it is true. We are particularly interested in formulas of low quantifier complexity. A Σ_2 formula is a formula of the form

$$\exists x_1 \cdots x_n \, \forall y_1 \cdots y_m \quad \gamma \ ,$$

where γ is quantifier free. Languages defined in Σ_2 are closed under disjunction and conjunction, but not necessarily negation. The negation of a Σ_2 formula is called a Π_2 formula, equivalently this is a formula whose quantifier prefix is $\forall^*\exists^*$. A forest property is called Δ_2 if it can be expressed both by a Σ_2 and a Π_2 formula. We will use $\Sigma_2(<)$ and $\Sigma_2(<, <_{\text{lex}})$ to specify which predicates are used in the signature, similarly for Π_2 and Δ_2.

With limited quantification, the choice of signature is a delicate question. For instance, adding a child relation changes the expressive power.

3.1 The problem

We want an algorithm deciding whether a given regular forest language is definable in $\Delta_2(<, <_{\text{lex}})$ and another one for deciding whether it is in $\Delta_2(<)$.

As noted earlier, the corresponding problem for words was solved by Pin and Weil [11]: a word language L is definable in $\Delta_2(<)$ if and only if its syntactic monoid $M(L)$ belongs to the variety DA, i.e. it satisfies the identity

$$(mn)^\omega = (mn)^\omega m (mn)^\omega$$

for all $m, n \in M(L)$. The power ω means that the identity holds for sufficiently large powers (in different settings, ω is defined in terms of idempotent powers, but the condition on sufficiently large powers is good enough here). Since one can effectively test if a finite monoid satisfies the above property (it is sufficient to verify the power $|M(L)|$), it is decidable whether a given regular word language is definable in $\Delta_2(<)$. We assume that the language L is given by its syntactic monoid and syntactic morphism, or by some other representation, such as a finite automaton, from which these can be effectively computed.

We will show that a similar characterization can be found for forests; although the identities will be more involved. For decidability, it is not important how the input language is represented. In this paper, we will represent a forest language by a forest algebra that recognizes it. Forest algebras are described in the next section.

3.2 Tree languages.

We give an algorithm which says when a forest language belongs to a class $\mathcal{L}$, which is either $\Delta_2(<)$ or $\Delta_2(<, <_{\text{lex}})$. What about tree languages? There are two ways of getting a class of tree languages from a class of forest languages $\mathcal{L}$.

1. The class of tree languages that belong to $\mathcal{L}$.
2. The class of tree languages of the form $L \cap T_A$, where A is an alphabet, T_A is the set of all trees over alphabet A, and $L \in \mathcal{L}$ is a forest language over A.

Our algorithm gives a decision procedure under the first definition. The usual understanding of tree languages definable in $\Delta_2(<)$ or $\Delta_2(<, <_{\text{lex}})$ corresponds to the second definition.

Fortunately, the two definitions are equivalent when $\mathcal{L}$ is either $\Delta_2(<)$ or $\Delta_2(<, <_{\text{lex}})$. This is because in both cases, $\mathcal{L}$ is closed under intersection and contains the languages T_A.

Closure under intersection is immediate. Why does $\mathcal{L}$ contain the languages T_A? Since $\Delta_2(<)$ is the less powerful logic, it suffices to show how to define T_A using a $\Sigma_2(<)$ formula, and also using a $\Pi_2(<)$ formula. The $\Sigma_2(<)$ formula says there exists a node that is an ancestor of all other nodes, while the $\Pi_2(<)$ formula says that for every two nodes, there exists a common ancestor.

In general, the definitions of tree language classes are not equivalent. Consider as $\mathcal{L}$ the class of forest languages defined by purely existential formulas. In particular, if a language $L \in \mathcal{L}$ contains a tree t, then it also contains the forest $t + t$. This means that under the first definition, the only tree languages we get are the empty tree languages. Under the second definition, we get some more tree languages, such as "trees with at least two nodes".

3.3 Basic properties of Π_1 and Σ_2

Most of the proofs in the paper will work with $\Sigma_2(<)$ or $\Sigma_2(<, <_{\text{lex}})$ formulas. We present some simple properties of such formulas in this section.

Apart from defining forest languages, we will also be using formulas to define languages of contexts. To define a context language we use formulas with a free variable; such a formula is said to hold in a context if it is true when the free variable is mapped to the hole of the context. For instance, the formula $\forall y \; y < x \Rightarrow a(y)$ with a free variable x defines the set of contexts where every ancestor of the hole has label a.

We begin by describing the expressive power of purely universal formulas Π_1.

Lemma 3.1 *A forest language is closed under pieces if and only if it is definable in* Π_1*. Likewise for context languages.*

proof : It is clear that a forest language definable in Π_1 is closed under pieces as the models of a Π_1 formula are closed under substructures.

For the converse, let L be a forest language that is closed under pieces, and let $\alpha : A^\Delta \to (H, V)$ be its syntactic algebra. Thanks to a pumping argument, any forest has a piece with the same type, but at most $|H|^{|H|}$ nodes. Let T be the finite set of forests with at most $|H|^{|H|}$ nodes that are outside L. Thanks to the pumping argument, a forest belongs to L if and only if it has no piece in the set T. The latter is a property that can be expressed in $\Pi_1(<, <_{\text{lex}})$.

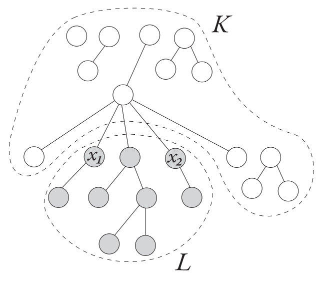
If, additionally, the language L is commutative, then we do not need to worry about the lexicographic order when talking about the pieces in T, only the descendant order is relevant. $\square$

We now turn to the formulas from Σ_2. We begin with $\Sigma_2(<, <_{\text{lex}})$, since it has the better closure properties.

Lemma 3.2 *Let K, K' be context languages and L, L' be forest languages. If these languages are all definable in $\Sigma_2(<, <_{\text{lex}})$, then so are*

1. *the forest language $KL = \{qt : q \in K, t \in L\}$,*
2. *the forest language $L + L' = \{t + t' : t \in L, t' \in L'\}$,*
3. *the context language $KK' = \{qq' : q \in K, q' \in K'\}$,*
4. *the forest language $L \cap L' = \{t : t \in L, t \in L'\}$,*
5. *the context language $K \cap K' = \{q : q \in K, q \in K'\}$,*

proof : We only do the proof for KL, the others are treated similarly. When does a forest t belong to KL? There must exist two siblings x_1 and x_2 such that the set, call it X, of gray nodes in the picture below



describes a forest in L, and the other nodes describe a context in K. Below we define this property more precisely, and show that it can be defined in $\Sigma_2(<, <_{\text{lex}})$.

First, we want to say that the nodes x_1 and x_2 are siblings. This can be expressed by a formula, call it $\alpha(x_1, x_2)$, of $\Sigma_2(<, <_{\text{lex}})$. The formula quantifies existentially a common ancestor and uses universal quantification to check that this common ancestor is a parent of both nodes:

$$\alpha(x_1, x_2) = \exists x \; \forall y \; x_1 \leq_{\text{lex}} x_2 \wedge x < x_1 \wedge x < x_2 \wedge (y < x_1 \to y \leq x) \wedge (y < x_2 \to y \leq x_2)$$

Next, we describe the set X. Membership $x \in X$ is defined by a quantifier-free formula

$$\beta(x) = (x_1 \leq_{\text{lex}} x \wedge x \leq_{\text{lex}} x_2) \vee (x > x_2).$$

Next, we say what it means for the part inside X describes a forest in L. Suppose that the forest language L is defined by a formula φ of $\Sigma_2(<, <_{\text{lex}})$. To say that X describes a forest in L we use the formula φ^β obtained from φ by restricting quantification to nodes satisfying β. Note that φ^β has free variables x_1, x_2 from β. Since β is quantifier-free, the formula φ^β is also in $\Sigma_2(<, <_{\text{lex}})$.

Finally, we say that the part outside X describes a context in K. The idea is that the hole of this context corresponds to the set X. The logical formula is constructed below. Suppose that $\phi(x)$ is a formula of $\Sigma_2(<, <_{\text{lex}})$ that describes K. Note that this formula has a free variable, as with formulas for contexts, which corresponds to the hole. Let $\phi^{\neg\beta}(x)$ be the formula obtained from $\phi(x)$ by restricting quantification to nodes not satisfying β. The remaining question is: which node should we use for x? Any node from X will do, since for each node $y \notin X$, all nodes from X have the same relationship to y, with respect to the descendant and lexicographic orders. We use the node x_1 for x.

Summing up, the formula for KL is written below.

$$\exists x_1 \exists x_2 \; \alpha(x_1, x_2) \;\wedge\; \phi^{\neg\beta}(x_1) \;\wedge \varphi^\beta.$$

$\square$

3.4 Σ_2 expressions.

In the proofs, it will sometimes be more convenient to use a type of regular expression instead of formulas. These are called Σ_2 *forest expressions* and Σ_2 *context expressions*, and are defined below by mutual recursion:

- Any forest (respectively, context) language that is closed under pieces is a Σ_2 forest (respectively, context) expression. For any label $a \in A$, $\{a\square\}$ is a Σ_2 context expression. Likewise for $\{\square\}$, the language containing only the empty context.
- If K, K' are Σ_2 context expressions and L, L' are Σ_2 forest expressions, then
  - $K \cdot K'$ is a Σ_2 context expression;
  - $L + L'$ is a Σ_2 forest expression;
  - $K \cdot L$ is a Σ_2 forest expression.
  - $L \cup L'$ is a Σ_2 forest expression.
  - $K \cup K'$ is a Σ_2 forest expression.
  - $L \cap L'$ is a Σ_2 forest expression.
  - $K \cap K'$ is a Σ_2 forest expression.

From Lemmas 3.1 and 3.2 it follows that languages defined by Σ_2 forest and context expressions are definable in $\Sigma_2(<, <_{\text{lex}})$.

4 Characterization of $\Delta_2(<, <_{\text{lex}})$

The main result of this paper is the following theorem:

Theorem 4.1 (Effective characterization of Δ_2 with descendant and lexicographic orders) *A forest language is definable in $\Delta_2(<, <_{\text{lex}})$ if and only if its syntactic forest algebra satisfies the following identity, called the Δ_2 identity,*

$$v^\omega w v^\omega = v^\omega \qquad \textit{for } w \preceq v \;. \tag{1}$$

Before we prove the main theorem, we state and prove an important corollary.

Corollary 4.2 *It is decidable whether a forest language can be defined in* $\Delta_2(<, <_{\text{lex}})$.

proof : We assume that the language is represented as a forest algebra. This representation can be computed based on other representations, such as automata or monadic second-order logic.

Once the forest algebra is given, the Δ_2 identity can be tested in polynomial time by searching through all elements of the algebra. The relation $\preceq$ can be computed in polynomial time, using a fixpoint algorithm as in [3]. □

The following lemma gives the easier implication from the main theorem.

Lemma 4.3 *Let* φ *be a formula of* $\Sigma_2(<, <_{\text{lex}})$ *and let* $q \preceq p$ *be two contexts. For* $n \in \mathbb{N}$ *sufficiently large, forests satisfying* φ *are closed under replacing* $p^n p^n$ *with* $p^n q p^n$.

proof : Assume that φ is $\exists x_1 \cdots x_k \forall y_1 \ldots y_l \psi$, with ψ quantifier-free. Any first-order definable tree language is aperiodic [13], i.e. there is a number m such that any context p^i can be replaced by p^j without affecting membership in the language, for any $i, j \geq m$. We set $n = 2m + (k+1)(l+1)$.

Consider a forest $t = rp^n p^n s$ that satisfies φ. We want to show that the forest $rp^n q p^n s$ also satisfies φ. By aperiodicity, it is sufficient to show that for some numbers $i, j > m$, $rp^i q p^j s$ satisfies φ.

Because t satisfies φ we can fix k nodes $x_1, \ldots, x_k$ that make $\forall y_1 \ldots y_l \psi$ true. By the choice of n, t can be decomposed as $rp^i p^l p^j s$ such that $i, j \geq m$ and the middle p^l part contains none of the nodes $x_1, \ldots, x_k$. We show that $t' = rp^i p^l q p^j s$ satisfies φ, which will conclude the proof of the lemma.

We identify the nodes of t with the nodes of t' outside the inserted context q. Consider the valuation of the variables $x_1, \cdots, x_k$ that we fixed above. We show that this valuation, when seen as nodes of t', makes φ true. Indeed, for any valuation for the variables $y_1, \cdots, y_l$ in t', we show that the valuation makes the quantifier-free part ψ true. This is obvious if none of the y_i are in q because φ holds in t and the insertion of q does not affect the relationship $<$ and $<_{\text{lex}}$ between the selected nodes. If some of the y_i are in q then one of the contexts p in the middle block p^l of t' does not contain any variable. As removing the context p that does not contain any variable does not affect the relationship $<$ and $<_{\text{lex}}$ between the selected nodes, $\psi(x_1, \cdots, x_k, y_1, \cdots, y_l)$ holds on $rp^i p^l q p^j s$ iff it holds on $rp^i p^{l-1} q p^j s$. Moreover, because q is a piece of p, replacing q by p does not affect the relationship $<$ and $<_{\text{lex}}$ between the selected nodes and therefore $\psi(x_1, \cdots, x_k, y_1, \cdots, y_l)$ holds on $rp^i p^{l-1} q p^j s$ iff it holds on $rp^i p^{l-1} p p^j s = t$. Hence ψ must hold with the new valuation. □

The rest of the paper contains the more difficult implication of Theorem 4.1 which is a consequence of the proposition below.

Proposition 4.4 *Fix a morphism* $\alpha : A^\Delta \to (H, V)$, *with* (H, V) *satisfying the* Δ_2 *identity. For each* $h \in H$, *the set* L_h *of forests* t *with type* $\alpha(t) = h$ *is definable by a* Σ_2 *forest expression, and thus also by a formula of* $\Sigma_2(<, <_{\text{lex}})$.

Before proving this proposition, we show how it concludes the proof of Theorem 4.1. Since Σ_2 expressions allow union, the above proposition shows that any language recognized by α can be defined by a Σ_2 forest expression. In particular, if L is recognized by α, then both L and its complement can be defined by Σ_2 forest expressions, and consequently formulas of $\Sigma_2(<, <_{\text{lex}})$. Since the complement of a $\Sigma_2(<, <_{\text{lex}})$ formula is a $\Pi_2(<, <_{\text{lex}})$ formula, we get the right-to-left implication in Theorem 4.1.

The rest of the paper is devoted to showing Proposition 4.4. The proof is by induction on two parameters. For the second parameter, we need to define a pre-order on H. We say that a type h is *reachable from* a type g if there is a context type $v \in V$ such that $h = vg$. If h and g are mutually reachable from each other, then we write $h \sim g$. Note that $\sim$ is an equivalence relation. Note also that if g is reachable from h, then h is a piece of g. We write $H_\perp$ for the set of types h that can be reached from every type $g \in H$. Note that $H_\perp$ is not empty, since it contains the type $h_1 + \cdots + h_n$, for any enumeration $H = \{h_1, \ldots, h_n\}$.

The proof of Proposition 4.4 is by induction on the size of the algebra (H, V) and then on the position of h in the reachability pre-order. The two parameters are ordered lexicographically, the most important parameter being the size of the algebra. That is we will either decrease the size of the algebra or stay within the same algebra, but go from a type g to a type h such that h is reachable from g but not vice versa. As far as h is concerned, the induction corresponds to a bottom-up pass, where types close to the leaves are treated first.

Part of the induction proof is presented in Section 5. However, the induction breaks down for types from $H_\perp$, which are treated in Section 7.

5 Types outside $H_\perp$

In this section we prove Proposition 4.4 for forest types outside $H_\perp$. We fix such a forest type h for the rest of the section. By induction assumption, for each type $g \not\sim h$ from which h is reachable, we have a Σ_2 forest expression defining the language L_g of forests of type g. (The case when there are no such types g corresponds to the induction base, which is treated the same way as the induction step.) In this section we assume that h is outside $H_\perp$, and we will produce a Σ_2 forest expression for L_h. The case where h is in $H_\perp$ will be treated in Section 7.

In the following, we will be using the *stabilizer* of h, defined as

$$stab(h) = \{v : vh \sim h\} \subseteq V \ .$$

We say that a context type v stabilizes h if it belongs to the stabilizer of h. The key lemma is that the Δ_2 identity implies that the stabilizer is a submonoid of V.

Lemma 5.1 *The stabilizer of h only depends on the $\sim$-class of h. In particular, it is a submonoid of V.*

proof : We need to show that if $h \sim h'$ then $stab(h) = stab(h')$. Assume $v \in stab(h)$. Then $vh \sim h$. Hence we have u_1, u_2, u_3 such that $h = u_1vh, h = u_2h'$ and $h' = u_3h$. This implies that $h' = u_3u_1vu_2h'$ and therefore $h' = (u_3u_1vu_2)^\omega h'$. From the Δ_2 identity we have that

$$h' = (u_3u_1vu_2)^\omega h' = (u_3u_1vu_2)^\omega v(u_3u_1vu_2)^\omega h' = (u_3u_1vu_2)^\omega vh' \ .$$

Hence h' is reachable from vh'. Since vh' is clearly reachable from h', we get $vh' \sim h'$ and $v \in stab(h')$.

To see that $stab(h)$ is a submonoid consider $v, v' \in stab(h)$. We need to show that $vv' \in stab(h)$. Let $h' = v'h$. Because $v' \in stab(h)$ we have $h' \sim h$. As $v \in stab(h) = stab(h')$ we have $vh' \sim h'$ and hence $vv'h \sim h$. $\Box$

Recall now the piece order on forest types from the end of Section 2.2, which corresponds to removing nodes from a forest. We say a set $F \subseteq H$ of forest types is *closed under pieces* if any piece of a forest type $f \in F$ also belongs to F. A similar definition is also given for sets of context types. Another consequence of the Δ_2 identity is:

Lemma 5.2 *Each stabilizer is closed under pieces.*

proof : We need to show that if u stabilizes h, then each piece u' of u also stabilizes h. By definition of the stabilizer we have a context type v such that $h = vuh$. We are looking for a context type w such that $wu'h = h$. From $h = vuh$ we get $h = (vu)^\omega h$. Hence by the Δ_2 identity we have $h = (vu)^\omega u'(vu)^\omega h = (vu)^\omega u'h$ as desired. $\Box$

We now consider two possible cases: either $h+h \sim h$, or not. Equivalently, we could have asked if $h+\Box$ stabilizes h. Again equivalently, we could have asked if $\Box+h$ stabilizes h. When $h+h \sim h$, we will conclude by induction on the size of the algebra. When $h + h \not\sim h$, we will conclude by induction on the reachability pre-order. These cases are treated separately in Sections 5.2 and 5.1, respectively.

5.1 $h + h \sim h$

Let G be the set of pieces of h. By assumption that $h + h \sim h$, we know that both $h + \square$ and $\square + h$ stabilize h.

Lemma 5.3 *If $h + h \sim h$ then $(G, stab(h))$ is a forest algebra.*

proof: We need to show that the two sets are closed under all operations:

$$\begin{aligned} stab(h)stab(h) &\subseteq stab(h) \\ G + G &\subseteq G \\ \square + G, G + \square &\subseteq stab(h) \\ stab(h)G &\subseteq G \end{aligned}$$

The first of the above inclusions follows from Lemma 5.1. For the second inclusion, we note that $h+h$ is a piece of h by assumption on $h+h \sim h$. In particular, each forest type in $G+G$ is a piece of h. For the third inclusion, $h + h \sim h$ implies that both $h + \square$ and $\square + h$ stabilize h. Since by Lemma 5.2 the stabilizer is closed under pieces, we get the third inclusion. For the last inclusion, consider $v \in stab(h)$ and $g \in G$. We need to show that $vg \in G$. This holds because vg is a piece of vh, which is a piece of h as $vh \sim h$. $\square$

Recall that in this section we are dealing with the case when h is outside $H_\perp$, i.e. there are some forest types that can be reached from h but not vice versa. In this case we show that G is a proper subset of H. To see this, we show that for $h_\perp \in H_\perp$, $h_\perp \notin G$. Assume for contradiction that $h_\perp \in G$. Then $h_\perp$ is a piece of h and because $h + \square \in stab(h)$ and $stab(h)$ is closed under pieces (Lemma 5.2), we infer $h_\perp + \square \in stab(h)$ and h is reachable from $h_\perp$, a contradiction.

Therefore the algebra from the above lemma is a proper subalgebra of the original (H, V). Furthermore, this algebra contains all pieces of h; so it still recognizes the language L_h; at least as long as the alphabet in the morphism is reduced to include only letters that can appear in h. We can then use the induction assumption on the smaller algebra to get a Σ_2 forest expression for L_h.

5.2 $h + h \not\sim h$

For $v \in V$, we write K_v for the set of contexts of type v. For $g \in H$, we write L_g for the set of forests of type g, and M_g for the set of trees of type g.

Let G be the set of forest types g such that h is reachable from g but not vice-versa. By induction assumption, for each $g \in G$, the language L_g is definable by a Σ_2 forest expression. Our goal is to give a Σ_2 forest expression for L_h.

Lemma 5.4 *Any forest t of type h can be decomposed as $t = ps$, with s a forest whose type is reachable from h, and which furthermore is:*

1. *A tree $s = as'$ with the type of s' in G; or*
2. *A forest $s = s_1 + s_2$ with the types of s_1, s_2 in G.*

proof: Consider decompositions of t as $t = ps$. Among such decompositions, take a decomposition where the forest s has a type reachable from h, but s has no subforest with a type reachable from h. Such a decomposition always exists as t is of type h. If s is a tree, we get case (1), if s is a forest, we get case (2). $\square$

Note that in the above lemma, the type of the context p must stabilize h, since both the type of s and the type of the whole forest t are in the class of h. Therefore, thanks to the above lemma, the set L_h of forests with type h can be decomposed as

$$L_h = \bigcup_{u \in stab(h)} \Big(\bigcup_{\substack{a \in A, g \in G \\ uag = h}} K_u a L_g \quad \cup \quad \bigcup_{\substack{g_1, g_2 \in G \\ u(g_1 + g_2) = h}} K_u(L_{g_1} + L_{g_2}) \Big) \qquad (2)$$

Note that L_g, L_{g_1} and L_{g_2} can all be written as Σ_2 forest expressions thanks to the induction assumption from Proposition 4.4. The only thing that remains is showing that the context language K_u can be defined by a Σ_2 context expression. For this, we use the following proposition.

Proposition 5.5 *For any $v \in V$, the language K_v of contexts of type v is defined by a finite union of concatenations of the form $K_1 \cdots K_m$ where each context language K_i is either:*

1. *A singleton language $\{a\Box\}$ for some $a \in A$; or*
2. *A context language closed under pieces; or*
3. *A context language $\Box + M_g$ or $M_g + \Box$ for some $g \in H$.*

The proof of this proposition will be presented in Section 6. Meanwhile, we show how the proposition gives a Σ_2 context expression for each language K_u in (2). The singleton languages, and the languages closed under pieces are Σ_2 context expressions by definition. The only potential problem is with the languages $\Box + M_g$ or $M_g + \Box$ that appear in the proposition. Since the context types u that appear in (2) stabilize h, the forest type g has to be such that $\Box + g$ or $g + \Box$ stabilizes h. In either case, g cannot be reachable from h, since $h + h \not\sim h$ and the stabilizer is closed under pieces. As h is obviously reachable from g, the language L_g is definable by a Σ_2 forest expression thanks to the induction assumption from Proposition 4.4. Finally, M_g is the intersection of L_g with the set of all trees, which is definable by a Σ_2 forest expression.

6 Treating contexts like words

In this section, we prove Proposition 5.5. The basic idea is that a context is treated as a word, whose letters are smaller contexts. The proof strategy is as follows. First, in Section 6.1, we present the characterization of $\Delta_2(<)$ for words, which was shown by Pin and Weil in [11]. This characterization is slightly strengthened to include what we call stratified monoids, which are used to model the contexts that appear in Proposition 5.5. Then, in Section 6.2, we apply the word result, in its strengthened form, to prove Proposition 5.5.

6.1 $\Delta_2(<)$ for words

In this section we present the characterization of $\Delta_2(<)$ for words, extended to stratified monoids. A *stratified monoid* is a monoid M along with a pre-order $\preceq$ that satisfies the following property:

$$m^\omega n m^\omega = m^\omega \qquad \text{for } n \preceq m \ .$$

A subset $N \subseteq M$ is called *downward closed under* $\preceq$ if for every $n \in N$, and every $m \preceq n$, we also have $m \in N$.

Proposition 6.1 *Let A be an alphabet (possibly infinite), and let $\beta : A^* \to M$ be a morphism into a stratified monoid $(M, \preceq)$ that satisfies the identity*

$$(mn)^\omega m (mn)^\omega = (mn)^\omega \tag{3}$$

For any $m \in M$, the language $\beta^{-1}(m)$ is defined by a finite union of expressions

$$A_0^* B_1 A_1^* \cdots B_i A_i^*$$

where each B_j is of the form $A \cap \beta^{-1}(n)$ for some $n \in M$, and each A_j is of the form $A \cap \beta^{-1}(N)$ for some $N \subseteq M$ downward closed under $\preceq$.

The difference between the above result and the main technical result in Pin and Weil is twofold. First, we use infinite alphabets here. Second, we use stratified monoids to get a stronger conclusion, where the letters in the blocks A_i^* are downward closed. Both differences are necessary for our application to context languages.

Our proof is a straightforward adaptation of a proof of Thérien and Wilke in [16], which analyzed the languages recognized by semigroups in DA.

Before proving this result, we remark how Proposition 6.1 gives the characterization of $\Delta_2(<)$ presented by Pin and Weil:

Corollary 6.2 (Pin and Weil [11]) *A word language (over a finite alphabet) is definable* $\Delta_2(<)$ *if and only if its syntactic monoid satisfies the identity (3).*

proof : The only if implication is shown using a standard Ehrenfeucht-Frass argument, we only consider the if implication.

Let then $L \subseteq A^*$ be a language recognized by a morphism $\beta : A^* \to M$, with M satisfying (3). We can see this M as a stratified monoid under the identity pre-order. By applying Proposition 6.1, we see that each inverse image $\beta(m)$ is defined by an expression as in Proposition 6.1 (the downward closure is a vacuous condition, since the order is trivial). Since each such expression is clearly expressible in $\Sigma_2(<)$, we get that L is definable in $\Sigma_2(<)$. Furthermore, by Proposition 6.1 also the complement of L is definable in $\Sigma_2(<)$, and therefore L is also definable in $\Pi_2(<)$. □

We now proceed to the proof of Proposition 6.1. The proof is by induction on the size of $\beta(A) \subseteq M$ or, equivalently, the number of elements in the monoid that correspond to single letters. In the proof of Thrien and Wilke, the induction was simply on the size of the alphabet, but this will not work here, since the alphabet is infinite.

We use the term Σ_2 *word expression* for the word expressions as in the statement of the Proposition 6.1. It is not difficult to show that languages defined by Σ_2 word expressions are closed under union, intersection and concatenation.

We will use the following notation. Given two elements m and n of M we say that $m \sim_{\mathcal{L}} n$ if there exist $k, l \in M$ such that $m = kn$ and $n = lm$. We say that $m \sim_{\mathcal{R}} n$ if there exist $k, l \in M$ such that $m = nk$ and $n = ml$. These are the left and right Green's relations.

A classical consequence of aperiodicity, itself a consequence of (3), is:

$$m \sim_{\mathcal{R}} n \ \wedge \ m \sim_{\mathcal{L}} n \ \Rightarrow \ m = n \qquad \text{for } m, n \in M \ . \tag{4}$$

We will also use the following property of monoids satisfying (3), which can be proved along the lines of Lemma 5.1.

$$m \sim_{\mathcal{R}} n \sim_{\mathcal{R}} mk \ \Rightarrow nk \sim_{\mathcal{R}} n \qquad \text{for } m, n, k \in M \ . \tag{5}$$

Lemma 6.3 *For all $m \in M$, the language $U_m = \{w : m\beta(w) \sim_{\mathcal{R}} m\}$ is definable by a Σ_2 word expression.*

proof : Let A_m be the set of letters a of A such that $m\beta(a) \sim_{\mathcal{R}} m$. In other words, $A_m = A \cap \beta^{-1}(N)$, where N is the set $\{n : mn \sim_{\mathcal{R}} m\}$.

We will show that $U_m = A_m^*$. Stated differently, a word belongs U_m if and only if all of its letters belong to A_m.

Thanks to (5), for all $n \sim_{\mathcal{R}} m$ we have $n\beta(a) \sim_{\mathcal{R}} m$. Hence by induction on the length of $w \in A_m^*$ we can prove $w \in U_m$. For the converse implication, let w be a word outside A_m^*, of the form $w = w_1 a w_2$ with $w_1 \in A_m^*$ and $a \notin A_m$. Then $m\beta(w) = n\beta(a)n'$ for some n, n', and from the discussion above we have $n \sim_{\mathcal{R}} m$. Hence by (5) and by hypothesis on a, $n\beta(a) \not\sim_{\mathcal{R}} m$ and w cannot be in U_m.

To conclude, we need to show that N is closed under $\preceq$. Indeed, let $k \preceq n$ and let $n \in N$. By assumption on the monoid being stratified, we have $n^\omega k n^\omega = n^\omega$. In particular, we have

$$mn^\omega kn^\omega \sim_{\mathcal{R}} mn^\omega \sim_{\mathcal{R}} m \ .$$

From the above it follows that $mn^\omega k \sim_{\mathcal{R}} m$, which gives $mk \sim_{\mathcal{R}} m$ by (5), and hence $k \in N$. □

Note that by (5) we have $U_n = U_m$ whenever $m \sim_{\mathcal{R}} n$.

Lemma 6.4 *For any $m \in M$, the following language can be defined by a Σ_2 word expression:*

$$V_m \quad = \quad \{a_1 \cdots a_i : \ \beta(a_1 \cdots a_i) = m \text{ and } \beta(a_1 \cdots a_{i-1}) \not\sim_{\mathcal{R}} m\} \ ,$$

Before we give the proof, we show how it concludes the proof of Proposition 6.1. Consider the set $\{w : \beta(w) \sim_{\mathcal{R}} m\}$. Each word w in this set can be written as uv where u is the smallest prefix of w such that $\beta(u) \sim_{\mathcal{R}} m$. Hence we have:

$$\{w : \beta(w) \sim_{\mathcal{R}} m\} = \bigcup_{n : n \sim_{\mathcal{R}} m} V_n U_m \ .$$

Since Σ_2 word expressions are closed under union and concatenation, the above language is definable by a Σ_2 word expression thanks to Lemmas 6.3 and 6.4. Using a symmetric version of Lemma 6.3 and Lemma 6.4 for $\sim_{\mathcal{L}}$, we can get an Σ_2 word expression for $\{w : \beta(w) \sim_{\mathcal{L}} m\}$. But we also know from (4) that

$$\beta^{-1}(m) = \{w : \beta(w) \sim_{\mathcal{R}} m\} \cap \{w : \beta(w) \sim_{\mathcal{L}} m\}$$

and the result follows by closure of Σ_2 word expressions under intersection.

proof : [Proof of Lemma 6.4] We say that m is *a prefix* of n if there exists $k \in M$ such that $n = mk$. This defines a pre-order in M. The proof is by induction on the position of m relative to this pre-order.

The induction base is when m has no proper prefixes: If $m = nk$ then $n \sim_{\mathcal{R}} m$. In this case the language V_m contains at most the empty word, since the condition on $a_1 \cdots a_{i-1}$ is infeasible. Clearly both languages $\emptyset$ and $\{\epsilon\}$ are Σ_2 word expressions.

Assume now that m is not minimal. Each word w of V_m can be written as ua where $a \in A$, $\beta(u) = k$ and $k\beta(a) = m$. Furthermore, u can be written as u_1u_2 where u_1 is the smallest prefix of u such that $n = \beta(u_1) \sim_{\mathcal{R}} k$. We therefore have:

$$V_m = \bigcup V_n U_{n,k} a \qquad \text{with } U_{n,k} = \{w \in A^* : n\beta(w) = k\}$$

where the union is taken for $n, k \in M$ such that $n \sim_{\mathcal{R}} k$, $k \not\sim_{\mathcal{R}} m$ a prefix of m and for $a \in A$ with $k\beta(a) = m$. By induction the language V_n is definable by a Σ_2 word expression. It is also clear that $U_{n,k} \subseteq U_n$. Recall from the proof of Lemma 6.3 that $U_n = A_n^*$ where $A_n = A \cap \beta^{-1}(N)$ for some $N \subseteq M$. Therefore we also have $\beta(U_{n,k}) \subseteq A_n^*$. From (5) and the fact that $k\beta(a) = m$ we know that $a \notin A_n$, therefore A_n is a proper subset of A. Let β' be the restriction of β to A_n. We have $U_{n,k} = \bigcup_{nx=k} \beta'^{-1}(x)$, from induction on the size of the alphabet in Proposition 6.1 we obtain a Σ_2 word expression for $U_{n,k}$. This concludes the proof of this lemma as Σ_2 word expressions are closed under concatenation and union. $\Box$

6.2 Proof of Proposition 5.5

We now proceed to show how the word result stated in Proposition 6.1 can be lifted to the context result in Proposition 5.5. Proposition 5.5 says that for any context type $v \in V$, the set of contexts of type v is described by a finite union of expressions of the form $K_1 \cdots K_m$ where each K_i is either: a singleton language $\{a\Box\}$; or closed under pieces; or an expression $\Box + M_g$ or $M_g + \Box$.

The basic idea is that we treat the context as a word over an infinite alphabet, which we call B. This alphabet has two kinds of letters. Both kinds are contexts:

- Contexts of the form $a\Box$, for $a \in A$.
- Contexts of the form $t + \Box$ or $\Box + t$, for t a tree over A.

Consider now the morphism $\beta : B^* \to V$, which is simply α restricted to the contexts in B^*. Every context p in K_v can be decomposed as $p = b_1 \cdots b_m \in B^*$. In particular, we have

$$K_v = \beta^{-1}(v) \ .$$

We can treat V as a stratified monoid, by using the piece relation $\preceq$ as the pre-order. By applying Proposition 6.1, we see that the inverse image $\beta^{-1}(v)$ can be presented as a finite union of expressions of the form:

$$(B \cap \beta^{-1}(N_0))^*(B \cap \beta^{-1}(n_1)) \cdots (B \cap \beta^{-1}(n_k))(B \cap \beta^{-1}(N_k))^* \ ,$$

where $n_1, \ldots, n_k$ are elements of V, and $N_1, \ldots, N_k$ are a subsets of V that are downward closed under $\preceq$.

We need to show that the expressions used above are of the three forms allowed by Proposition 5.5. Consider first an expression $(B \cap \beta^{-1}(W))^*$, where $W \subseteq V$ is closed under pieces. Since W is closed under pieces (as a set of context types), then so is the language $(B \cap \beta^{-1}(W))^*$ (as a set of contexts). Consider next an expression of the form $B \cap \beta^{-1}(n)$. This context language is a union of languages of the first (singleton) and third ($\Box + M_g$ or $M_g + \Box$) types described in Proposition 5.5. The union is not a problem for a Σ_2 word expression, since union distributes across concatenation.

7 Types in $H_\perp$

Recall that in Section 5, we managed to find a Σ_2 forest expression for each set L_h, assuming h was outside $H_\perp$. Our techniques failed for forest types $h \in H_\perp$, i.e. forest types reachable from every other forest type. In this section, we deal with these forest types.

In order to deal with the types from $H_\perp$, we will have to do a different induction, this time on context types. This induction, stated in Proposition 7.1, is expressed in terms of an equivalence relation $\equiv_v$. Given a context type $v \in V$ and two forest types h, h', we write

$$h \equiv_v h' \qquad \text{if} \quad \forall u \in V \ vuh = vuh' \ . \tag{6}$$

We extend this equivalence relation to context types, by

$$w \equiv_v w' \qquad \text{if} \quad \forall u \in V \ \forall h \in H \ vuwh = vuw'h \ . \tag{7}$$

By abuse of notation, we also lift the equivalence relation $\equiv_v$ to forests, considering two forests s, t equivalent when their forest types are equivalent. It is this meaning that is used in the statement below.

Proposition 7.1 *For any context type v, every equivalence class of forests under $\equiv_v$ is forest language definable by a Σ_2 forest expression.*

From Proposition 7.1 we immediately obtain a Σ_2 forest expression for L_h, as the equivalence class of $\equiv_v$ containing h, where $v = \Box$. Hence the proof of Proposition 7.1 ends the proof of Proposition 4.4. We note that the proof of Proposition 7.1 will be using the Σ_2 forest expressions L_h for types $h \notin H_\perp$ that have been developed in Section 5. In particular if an equivalence class of $\equiv_v$ is contained in $H \setminus H_\perp$, then it can easily be defined by the disjunction of all the Σ_2 forest expressions corresponding to each type. The difficulty is to handle equivalence classes that intersect $H_\perp$.

The rest of Section 7 is devoted to proving Proposition 7.1. The proof uses the following pre-order on context types. We say that a context type u is a *prefix* of a context type v if there exists a context type w such that $v = uw$ (we also say that v is an *extension* of u). We overload the use of $\sim$ and denote by $\sim$ the equivalence relation induced by the prefix pre-order, i.e. $v \sim w$ holds if v is both a prefix and an extension of w. The proof of Proposition 7.1 is by induction on the position of v in the prefix pre-order, starting with context types that have no proper extension, and ending at the context type $v = \Box$ that has no proper prefix.

7.1 The induction base

The base of the induction in the proof of Proposition 7.1 is when the context type v has no proper extension, i.e. $v \sim w$ holds for all extensions w of v. We will show that such a context type is necessarily *constant*, i.e. $vg = vh$ holds for all forest types $g, h \in H$. This gives the induction base, since for a constant context type v, there is only one equivalence class of $\equiv_v$, and this class is, by definition, the set of all forests, which can be defined by a Σ_2 forest expression (it is closed under pieces).

Lemma 7.2 *A context type has no proper extension if and only if it is constant.*

The if direction is immediate: if a context type v is constant, then $vu = v$ holds for all context types u, and therefore v has no proper extension. For the converse implication, as well as in the rest of Section 7, we will use the notion of stabilizers for context types:

$$stab(v) = \{w : vw \sim v\} \ .$$

When $u \in stab(v)$, we say that u *stabilizes* v. As for stabilizers of forest types (recall Lemma 5.1 and Lemma 5.2), the Δ_2 identity implies that the stabilizer $stab(v)$ is a submonoid of V and it is closed under pieces. The following lemma implies Lemma 7.2, since its assumptions are met by a context type without proper extensions. Recall that $H_\perp$ is the equivalence class of $\sim$ that contains all types reachable from any other type.

Lemma 7.3 *If both $H_\perp + \Box$ and $\Box + H_\perp$ intersect $stab(v)$, then the context type v is constant.*

proof : First note that if some context type in $H_\perp + \Box$ stabilizes v, then all context types in $H_\perp + \Box$ stabilize v, likewise for $\Box + H_\perp$. This is because the stabilizer is closed under pieces, and every type in $H + \Box$ is a piece of every type in $H_\perp + \Box$.

Let $f = h_1 + \cdots + h_n$, for some arbitrary enumeration $h_1, \ldots, h_n$ of H. As we noted above, both $f + \Box$ and $\Box + f$ stabilize v. Therefore,

$$v(\omega f + \Box + \omega f) \sim v \ .$$

The context type $(\omega f + \Box + \omega f)$ is constant, since any forest type $h \in H$ is a piece of f, and therefore by (1) we have

$$\omega f + h + \omega f = (f + \Box)^\omega \cdot (h + \Box) \cdot (f + \Box)^\omega \cdot 0 = (f + \Box)^\omega \cdot (f + \Box)^\omega \cdot 0 = \omega f + \omega f \ .$$

Hence the context type v is constant as it is equal to $wv(\omega f + \Box + \omega f)$ for some context type w.

$\Box$

7.2 The induction step

We now proceed to the induction step in Proposition 7.1. Recall that our goal is to find a Σ_2 forest expression for every equivalence class of $\equiv_v$. For a forest language L, we denote by $[L]_v$ the union of equivalence classes of $\equiv_v$ that intersect L, i.e.

$$[L]_v = \{t : t \equiv_v s \text{ for some } s \in L\} \quad \supseteq L \ .$$

We use a similar notation $[K]_v$ for languages of contexts. We say that a forest language is a *v-overapproximation* of a forest language L if it contains L, but is contained in $[L]_v$. In other words, a v-overapproximation may add forests to L, but it adds no new forest types, at least as far as the context type v is concerned. Note that a language may have several v-overapproximations.

Proposition 7.4 *For every $h \in H$, some v-overapproximation of L_h can be defined by a Σ_2 forest expression.*

The above result concludes the proof of Proposition 7.1. To see this, consider an equivalence class consisting of forest types $f_1, \cdots, f_n$. The set of forests with a type in the class is by definition equal to $\bigcup L_{f_i}$. By definition of v-overapproximation this set is also equal to $\bigcup \hat{L}_{f_i}$ where $\hat{L}_{f_i}$ is any v-overapproximation of L_{f_i}. Hence it is definable by a Σ_2 forest expression by Proposition 7.4.

The rest of this section is devoted to proving Proposition 7.4.

When h is outside $H_\perp$, then L_h itself, which is its own v-overapproximation, is definable by a Σ_2 forest expression by the results from the previous sections. The problem is when h is in $H_\perp$. Because v has some proper extension, from Lemma 7.3 we know that at least one of $\Box + H_\perp$ or $H_\perp + \Box$ is disjoint with the stabilizer of v. Without loss of generality we assume

$$\Box + H_\perp \ \cap \ stab(v) \quad = \quad \emptyset \ . \tag{8}$$

In other words, if the type of a context stabilizes v, then it is possible that some tree to the left of the hole has a type in $H_\perp$, however all trees to the right of the hole must have types outside $H_\perp$.

In order to obtain a v-overapproximation of L_h for $h \in H_\perp$ we will use the following decomposition of forests with types in $H_\perp$.

Lemma 7.5 *Any forest t of type in $H_\perp$ has a decomposition $t = ps$ where p is a context whose type stabilizes v, and s is a forest of type in $H_\perp$ that has one of the two forms below.*

1. *$s = as'$ with the type of the forest s' outside $H_\perp$; or*
2. *$s = as'$ with the type of the context $a\Box$ not stabilizing v; or*
3. *$s = s_1 + s_2$ with the types of the forests s_1, s_2 outside $H_\perp$; or*
4. *$s = s_1 + s_2$ and*
   - *if s_1 has type in $H_\perp$, then the type of $\Box + s_2$ does not stabilize v.*
   - *if s_2 has type in $H_\perp$, then the type of $s_1 + \Box$ does not stabilize v.*

proof: Consider the set D of all possible pairs (p, s) such that $t = ps$, the type of p preserves v and the type of s is in $H_\perp$. Take a pair $(p, s) \in D$ that is maximal in the following sense: if q is a nonempty context, then D has no pair with pq on the first coordinate.

Suppose s is a tree of the form $s = as'$. If s' has a type outside $H_\perp$, we get item (1). If s' has type in $H_\perp$, then by maximality, the context type of $pa\Box$ does not stabilize v, and therefore $a\Box$ has a type that does not stabilize v, so we get item (2).

Suppose s is a forest of at least two trees. Consider any partition of s into two nonempty forests $s = s_1 + s_2$. If both s_1, s_2 have type outside $H_\perp$, then we get item (3). Otherwise, we get case (4) by maximality of (p, s). $\Box$

From Lemma 7.5, we have for $h \in H_\perp$:

$$L_h = \bigcup_{u \in stab(v)} \ \bigcup_{\substack{f \in H_\perp \\ uf = h}} K_u \cdot Y_f$$

where Y_f stands for the set of all forests s that have type $f \in H_\perp$ and that satisfy one of the conditions (1)-(4) of Lemma 7.5. To get the v-overapproximation of L_h we will use v-overapproximations for the smaller expressions above, as stated by the following two lemmas. The first lemma is concerned with languages of the form Y_f.

Lemma 7.6 *For every $f \in H_\perp$, some v-overapproximation $\hat{Y}_f$ of Y_f can be defined by a Σ_2 forest expression.*

For the second lemma, concerning K_u, we need a more careful statement. The overapproximation that we give is not really an overapproximation of K_u, but it is an overapproximation that works as long as a forest of type in $H_\perp$ is inserted into the hole.

Lemma 7.7 *For any $u \in stab(v)$, one can define a Σ_2 context expression $\hat{K}_u$ such that for any forest s of type in $H_\perp$, $\hat{K}_u s$ is a v-overapproximation of $K_u s$.*

These two lemmas are proved in Sections 7.2.2 and 7.2.1, respectively. First we show how they complete the proof of Proposition 7.4. We write $L_\perp$ for the set of all forests with a type in $H_\perp$. We claim that the following language

$$\hat{L}_h = \bigcup_{u \in stab(v)} \bigcup_{\substack{f \in H_\perp \\ uf = h}} \hat{K}_u \cdot (\hat{Y}_f \cap L_\perp)$$

is a v-overapproximation of L_h.

The first property required from a v-overapproximation, $L_h \subseteq \hat{L}_h$, is immediate. For the second part, $\hat{L}_h \subseteq [L_h]_v$, we need a bit more effort. We show a stronger result, namely that for any u and f as in the summation above, we have

$$\hat{K}_u(\hat{Y}_f \cap L_\perp) \quad \subseteq \quad [K_u Y_f]_v$$

This completes the proof of $\hat{L}_h \subseteq [L_h]_v$, since $[\_]_v$ distributes across union. To prove the above, we apply the properties of $\hat{K}_u$ and $\hat{Y}_f$ to get

$$\hat{K}_u(\hat{Y}_f \cap L_\perp) \quad \subseteq \quad [K_u \cdot (\hat{Y}_f \cap L_\perp)]_v \quad \subseteq \quad [K_u \cdot \hat{Y}_f]_v \quad \subseteq \quad [K_u \cdot [Y_f]_v]_v$$

To complete the proof, we would like to replace $[Y_f]_v$ by Y_f in the last expression above. This can be done thanks to the following easily verifiable consequence of the fact that $\equiv_v$ is a congruence for forest algebras.

Fact 1 *For any set of contexts K and set of languages L, we have $[K[L]_v]_v = [KL]_v$.*

Thanks to Lemmas 7.7 and 7.6, the only thing keeping $\hat{L}_h$ from being defined by a Σ_2 forest expression is the language $L_\perp$. We deal with this language in the following lemma.

Lemma 7.8 *The language $L_\perp$ is definable by a Σ_2 forest expression.*

proof : A subforest of t is a forest s such that $t = ps$ for some context p. Take a forest $t \in L_\perp$ and consider a subforest s of t that is in $L_\perp$, but has no proper subforests in $L_\perp$. Then either s is a tree as' with $s' \notin L_\perp$, or $s = s_1 + s_2$ with $s_1, s_2 \notin L_\perp$. Therefore, a forest is in $L_\perp$ if and only if it has a subforest in

$$\bigcup_{\substack{a \in A, g \notin H_\perp \\ ag \in H_\perp}} aL_g \quad \cup \quad \bigcup_{\substack{g_1, g_2 \notin H_\perp \\ g_1 + g_2 \in H_\perp}} L_{g_1} + L_{g_2} \ .$$

Containing such a subforest can be expressed by a Σ_2 forest expression, by prefixing the set above with the set of all contexts. The expressions for L_g, L_{g_1} and L_{g_2} are Σ_2 forest expressions by the results from the section on types outside $H_\perp$. □

7.2.1 Proof of Lemma 7.7

Our goal in this section is to prove Lemma 7.7, which says that for any context type u stabilizing v, there is a Σ_2 context expression $\hat{K}_u$ such that for any forest s with a type in $H_\perp$, $\hat{K}_u s$ is a v-overapproximation of $K_u s$.

We apply Proposition 5.5 to get an expression for the context language K_u of the form

$$K_u = \bigcup_i K_{i,1} \cdots K_{i,n_i} \tag{9}$$

The problem with the expression above is that it may use, in some of the subexpressions $K_{i,j}$, languages $M_f + \Box$ or $\Box + M_f$ that involve forest types $f \in H_\perp$, and we do not know how to describe types in $H_\perp$. This is where the overapproximation comes in. We show that if the languages for types in $H_\perp$ are overapproximated, then the result satisfies the properties required by Lemma 7.7. A more detailed argument is described below.

We say a context language K satisfies (*) if it has the property required from K_u by Lemma 7.7, namely

> (*) there is a Σ_2 context expression $\hat{K}$ such that for any forest s with a type in $H_\perp$, the language $\hat{K}s$ is a v-overapproximation of Ks.

It is not difficult to see that property (*) is preserved by unions and compositions of context languages. In particular, in order to prove Lemma 7.7, it suffices to show (*) is satisfied by all languages $K_{i,j}$ that appear in (9).

The only problem with the overapproximation is when $K_{i,j}$ is of the form $M_f + \Box$ or $\Box + M_f$, for $f \in H_\perp$. In all the other cases, $K_{i,j}$ is known to be definable by a Σ_2 context expression, and no overapproximation is needed. Note that by (8), the expressions $\Box + L_f$ cannot be used, since a forest type from $H_\perp$ cannot appear to the right of the hole in a context type that stabilizes v. Therefore, to complete the proof of Lemma 7.7, it remains to show that for any $f \in H_\perp$, the context language $M_f + \Box$ satisfies (*).

In the following, we use an equivalence relation $\equiv_{v+}$. This is defined to be the intersection of all equivalence relations $\equiv_{vu}$, for context types u that do not stabilize v (and hence v is a strict prefix of vu). For a forest language L, we write

$$[L]_{v+} = \bigcap_{u \notin stab(v)} [L]_{vu} \tag{10}$$

By the induction assumption in Proposition 7.1, each equivalence class of $\equiv_{v+}$ is definable by a Σ_2 forest expression and therefore so is each language $[L]_{v+}$, as a union of equivalence classes of $\equiv_{v+}$. We will show that, for any $f \in H_\perp$, the context language $K = M_f + \Box$ satisfies (*) with $\hat{K} = [L_f]_{v+} + \Box$. Assume that the type of s is in $H_\perp$. Then we have:

$$Ks = M_f + s \quad \subseteq \quad \hat{K}s = [L_f]_{v+} + s \quad \subseteq \quad [Ks]_v = [L_f + s]_v \ .$$

The first inequality is clear. For the second inequality, we need to show that

$$v \cdot \alpha([L_f]_{v+} + s) \qquad \subseteq \qquad v \cdot \alpha(L_f + s)$$

This inclusion holds because we have:

$$v \cdot \alpha([L_f]_{v+} + s) = v \cdot (\Box + \alpha(s)) \cdot \alpha([L_f]_{v+}) = v \cdot (\Box + \alpha(s)) \cdot \alpha(L_f) = v \cdot \alpha(L_f + s)$$

In the second equality, we used the definition of $[L_f]_{v+}$ and the assumption that $\Box + \alpha(s)$ does not stabilize v. The latter follows from assumption (8) since the type of s is in $H_\perp$.

7.2.2 Proof of Lemma 7.6

In this section, we show that for every type $f \in H_\perp$, a v-overapproximation of Y_f can be defined by a Σ_2 forest expression. Recall that the language Y_f was defined based on a case distinction in Lemma 7.5, and therefore it can be decomposed into a union of four languages, one for each of the four cases in the lemma. As v-overapproximations are closed under union, for each of these languages, we provide a v-overapproximation defined by a Σ_2 forest expression.

For the languages corresponding to cases (1) and (3), we use the assumption that L_h can be defined by a Σ_2 expression for every type $h \notin H_\perp$. The interesting cases are (2) and (4).

The language corresponding to case (2) is a union of forest languages of the form

$$a \cdot L_h \tag{11}$$

ranging over letters a such that $a\Box$ does not stabilize v, and forest types h with $ah = f$. We treat each language $a \cdot L_h$ separately.

It may be the case that h belongs to $H_\perp$ and therefore we have no Σ_2 forest expression for L_h. However, we can use overapproximation. As a does not stabilize v, we can apply the induction assumption in Proposition 7.1 and obtain a Σ_2 forest expression for $[L_h]_{v+}$. But then the Σ_2 forest expression $a \cdot [L_h]_{v+}$ is a v-overapproximation of $a \cdot L_h$. It clearly contain $a \cdot L_h$ so it remains to show that it is included in $[aL_h]_v$. To see this consider a forest $t \in [L_h]_{v+}$ of type g and an arbitrary $u \in V$. As a does not stabilize v, ua does not stabilize v. Hence by the choice of g we have $vuag = vuah$ and at is in $[aL_h]_v$.

It remains to consider the case of (4), where have a union of sets

$$L_{h_1} + L_{h_2} \tag{12}$$

ranging over h_1, h_2 that satisfy the two implications in item (4) of Lemma 7.5. We do each pair h_1, h_2 separately. We consider three subcases.

The first subcase is when $h_1 \notin H_\perp$. Therefore we have a Σ_2 forest expression for L_{h_1}. In this case we claim that

$$L_{h_1} + [L_{h_2}]_{v+}$$

is a v-overapproximation of of $L_{h_1} + L_{h_2}$. The first requirement of v-overapproximation,

$$L_{h_1} + L_{h_2} \quad \subseteq \quad L_{h_1} + [L_{h_2}]_{v+} \ ,$$

is immediate. For the other requirement, we need to show that for any forests $t_1 \in L_{h_1}$ and $t_2 \in [L_{h_2}]_{v+}$, the type of $t_1 + t_2$ is $\equiv_v$-equivalent to $h_1 + h_2$. From $t_1 \in L_{h_1}$, we know that the type of t_1 is h_1, but all we know about t_2 is that its type g_2 satisfies $g_2 \equiv_{v+} h_2$. Consider an arbitrary $u \in V$. Since $h_1 + \Box$ does not stabilize v, we also have $u(h_1 + \Box)$ does not stabilize v. Hence from $g_2 \equiv_{v+} h_2$ we get $vu(h_1 + g_2) = vu(h_1 + h_2)$.

The second subcase, when $h_2 \notin H_\perp$, is treated as above by symmetry.

The third subcase is when both h_1 and h_2 are in $H_\perp$. As a consequence of $h_2 \in H_\perp$ and the second condition of item (4) in Lemma 7.5 is that

$$H_\perp + \Box \quad \cap \quad stab(v) \quad = \quad \emptyset \ . \tag{13}$$

We claim that a v-overapproximation of $L_{h_1} + L_{h_2}$ is

$$([L_{h_1}]_{v+} \cap L_\perp) \quad + \quad ([L_{h_2}]_{v+} \cap L_\perp) \ . \tag{14}$$

As before, the problem boils down to showing that for any forests

$$t_1 \in [L_{h_1}]_{v+} \cap L_\perp \qquad t_2 \in [L_{h_2}]_{v+} \cap L_\perp \ ,$$

the types g_1 of t_1 and g_2 of t_2 satisfy $g_1 + g_2 \equiv_v h_1 + h_2$. In other words, we need to show that for an arbitrary $u \in V$

$$vu(g_1 + g_2) = vu(h_1 + h_2) \ .$$

Since $t_1 \in L_\perp$, then by (13) the context type $g_1 + \Box$ does not stabilize v and therefore the same holds for $u(g_1 + \Box)$. Hence, we can use the assumption on $g_2 \equiv_{v+} h_2$ to infer

$$vu(g_1 + g_2) = vu(g_1 + \Box)g_2 = vu(g_1 + \Box)h_2 = vu(g_1 + h_2) \ .$$

In a similar way, we use $h_2 \in H_\perp$, the assumption (8), and $g_1 \equiv_{v+} h_1$, to complete the proof of this case, and of Lemma 7.6.

$$vu(g_1 + h_2) = vu(\Box + h_2)g_1 = vu(\Box + h_2)h_1 = vu(h_1 + h_2) \ .$$

8 No lexicographic order

In this section, we consider the logic $\Delta_2(<)$ where only the descendant order, and not the lexicographic order, is available. We give an effective characterization in the following theorem.

Theorem 8.1 (Effective characterization of Δ_2 with the descendant order only) *A forest language is definable in $\Delta_2(<)$ if and only if its syntactic forest algebra satisfies the Δ_2 identity, as well as horizontal commutativity:*

$$h + g = g + h \ . \tag{15}$$

The "only if" implication is easy: we have already shown that the Δ_2 identity must hold in the syntactic forest algebra of a language definable in $\Delta_2(<, <_{\text{lex}})$, and $\Delta_2(<)$ is a fragment of $\Delta_2(<, <_{\text{lex}})$. Horizontal commutativity must also hold: the logic only has the descendant relation, and therefore its formulas are invariant under rearranging sibling subtrees.

The "if" implication is a minor variation on the work done in the previous sections. Recall that we proved before that if the syntactic forest algebra of a language L satisfies the Δ_2 identity, then both L and its complement can be defined by Σ_2 forest expressions. We apply this result also in our case. The problem is that the Σ_2 forest expressions are not commutative, and thus need not be definable in $\Sigma_2(<)$. We will show, however, that their commutative closure can be defined in $\Sigma_2(<)$. Here, we use the term *commutative closure of* L for the smallest language that contains L and is closed under rearranging sibling subtrees.

Proposition 8.2 *The commutative closure of a Σ_2 forest expression is definable in $\Sigma_2(<)$.*

Before we prove this proposition, we remark that this is not a completely generic result. For instance consider the following language over a one letter alphabet: "Each node is a leaf or has two children, and some leaf has an even number of ancestors". This language is definable in $\Sigma_3(<, <_{\text{lex}})$ and is horizontally commutative (the formula comes from Potthoff [12]). However, this language cannot be defined in $\Sigma_3(<)$. Actually, an Ehrenfeucht-Fraissé argument shows that every first-order formula, that has quantifier depth n and only uses the descendant order, will give the same result for all balanced binary trees of depths larger than 2^n.

The proof of the above proposition is by induction on the size of the Σ_2 forest expression.

The base case is when the Σ_2 expression is either $\{a\Box\}$, or a language that is closed under pieces. In the first case, the language is clearly definable in $\Sigma_2(<)$. In the second case, we revisit the proof of Lemma 3.1, which showed that a language L that is closed under pieces is definable in Π_1. If we take the commutative closure of L, we get a commutative language closed under pieces. In the proof of Lemma 3.1, we constructed the Π_1 formula by forbidding a finite number of pieces; in the commutative case the formula does not need to worry about the order of siblings in the forbidden pieces.

In the induction step, we have to consider the operations that are allowed by Σ_2 expressions: union, intersection, (horizontal) concatenation

$$L + L' \tag{16}$$

and (vertical) composition

$$K \cdot L \quad K \cdot K' \tag{17}$$

for a forest languages L, L' and context languages K, K'. Union and intersection are easy. Concatenation and composition are more problematic. Actually, $\Sigma_2(<)$ is not closed under these two operations. For instance, the languages

$$\begin{aligned} K &= \{a + \Box, \Box + a\} \\ L &= \{b + c, c + b\} \end{aligned}$$

are both definable in $\Sigma_2(<)$, but their concatenation

$$KL = \{a + b + c, a + c + b, b + c + a, c + b + a\}$$

is not, since it does not contain the forest $b + a + c$.

Nevertheless, if we use commutative closure, the problem disappears. That is, we will show that if the languages K, K', L, L' are definable in $\Sigma_2(<)$, then the commutative closure of each of the languages in (16) and (17) can be defined in $\Sigma_2(<)$. We only do the cases $L + L'$ and $K \cdot K'$, the language $K \cdot L$ is done the same way.

Lemma 8.3 *If forest languages L, L' are definable in $\Sigma_2(<)$, then so is the commutative closure of $L + L'$.*

proof : We write $L \oplus L'$ for the commutative closure of $L + L'$.

Consider first the case when L' is a *tree* language definable in $\Sigma_2(<)$. In this case, the formula for $L \oplus L'$ formula places an existentially quantified variable over the root of one tree, and then relativizes the formulas for L and L', respectively, to the nodes the are not descendants (respectively, are descendants), of this existentially quantified root.

For the general case, we use the following lemma on forest languages definable in $\Sigma_2(<)$.

Lemma 8.4 *Every forest language L definable in $\Sigma_2(<)$ can be written as a finite union of languages $L_0 \oplus M_1 \oplus \cdots \oplus M_n$, where L_0 is a forest language definable in $\Pi_1(<)$, and $M_1, \ldots, M_n$ are tree languages definable in $\Sigma_2(<)$.*

proof : The statement of the lemma immediately follows from the following claim on formulas of $\Sigma_2(<)$. We claim that any formula φ of $\Sigma_2(<)$ is equivalent to a finite disjunction of formulas of the form

$$\exists z_1 \ldots \exists z_n \quad \psi \wedge \bigwedge_{i \in \{1,\ldots,n\}} (\forall y \ \neg(y < z_i)) \wedge \psi_i$$

where $\psi \in \Pi_1(<)$, and $\psi_1, \ldots, \psi_n \in \Sigma_2(<)$ are formulas such that

- Each formula ψ_i has all quantification relativized to descendants of z_i.
- The formula ψ has all quantification relativized to nodes that are not descendants of any of the nodes $z_1, \ldots, z_n$.

The idea, of course, is that the z_i describe the roots of the trees that contain the existentially quantified nodes $x_1, \ldots, x_m$ in the original formula φ of $\Sigma_2(<)$. The straightforward proof of the claim is omitted. The finite disjunction ranges over all possible repartitions of the nodes $x_1, \ldots, x_m$ into distinct trees. $\square$

From this normal form, since $\oplus$ distributes across union, it suffices to give a $\Sigma_2(<)$ formula for languages of the form

$$L_0 \oplus M_1 \oplus \cdots \oplus M_n \quad \oplus \quad L'_0 \oplus M'_1 \oplus \cdots \oplus M'_n \ ,$$

where the M languages are tree languages definable in $\Sigma_2(<)$ and where L_0, L'_0 are forest languages definable in $\Pi_1(<)$. By the technique shown at the beginning of the proof, it is sufficient to obtain a formula for $L_0 \oplus L'_0$. But the language $L_0 \oplus L'_0$ is closed under pieces, and therefore it is definable in $\Pi_1(<)$. $\square$

Lemma 8.5 *If context languages K, K' are definable in $\Sigma_2(<)$, then so is the commutative closure of $K \cdot K'$.*

proof : We write $K \odot K'$ for the commutative closure of $K \cdot K'$.

Consider first the case when either K or K' is a language $\{a\square\}$. The formula places a variable on node corresponding to $a\square$, and relativizes the formula for the remaining context language to the remaining nodes.

Consider now the general case. Again, we use a normal form lemma for languages definable in $\Sigma_2(<)$. This lemma is prove the same way as Lemma 8.4.

Lemma 8.6 *Every context language K definable in $\Sigma_2(<)$ can be written as a finite union of languages of the kinds*

$$\hat{K} \odot \{a\square\} \odot (\square \oplus L) \qquad or \qquad \square \oplus L$$

where $\hat{K}$ and L are context language and forest languages definable in $\Sigma_2(<)$.

We now use Lemma 8.6 to finish the proof of Lemma 8.6. We want to show that $K \odot K'$ is definable in $\Sigma_2(<)$. We apply Lemma 8.6 to the languages K and K'. Since the operation $\odot$ distributes across union, we can assume that the unions describing K and K' use just one language, of either of the two kinds. We have four cases to consider, we only do the most difficult one

$$\hat{K} \odot \{a\square\} \odot (\square \oplus L) \quad \odot \quad (\square \oplus L') \odot \{a'\square\} \odot \hat{K}' \ .$$

This language is the same as

$$\hat{K} \odot \{a\square\} \odot (\square \oplus L \oplus L') \odot \{a'\square\} \odot \hat{K}' \ .$$

Let $\psi_{\hat{K}}$ be the $\Sigma_2(<)$ formula defining the context language $\hat{K} \odot \{a\square\}$, obtained from the first case considered in the proof. Let $\psi_{\hat{K}'}$ be the $\Sigma_2(<)$ formula defining $\{a'\square\} \odot \hat{K}'$ obtained in the same way. Both of these formulas have a free variable, which describes the hole of the context. Using Lemma 8.3 we also have a $\Sigma_2(<)$ formula $\psi_{L\oplus L'}$ defining $L \oplus L'$.

The desired $\Sigma_2(<)$ formula puts an existentially quantified variable x on the node corresponding to $a\square$, another existentially quantified variable x' on the node corresponding to $a'\square$, and then runs three subformulas, for $\hat{K}$, $\square \oplus L \oplus L'$, and $\hat{K}'$, on the remaining nodes, appropriately relativizing the quantification. More specifically, this is the formula

$$\exists x\ \exists x' \quad a(x) \wedge a'(x) \wedge \text{parent}(x,x') \wedge \varphi_{\hat{K}}(x) \wedge \varphi_{\hat{K}'}(x') \wedge \varphi_{L\oplus L'}(x,x')$$

where $\text{parent}(x,x')$ is the Π_1 formula stating that x is a proper ancestor of x' and there are no nodes in between, $\varphi_{\hat{K}}(x)$ is constructed from $\psi_{\hat{K}}$ by relativizing all quantification to the ancestors of x, $\varphi_{\hat{K}'}(x)$ is constructed from $\psi_{\hat{K}'}$ by relativizing all quantification to the descendants of x, and $\varphi_{L\oplus L'}(x,x')$ is constructed from $\psi_{L\oplus L'}$ by relativizing all quantification to the nodes that are neither ancestor of x nor descendant of x'. $\square$

9 Discussion

In this paper we considered a signature with the descendant and lexicographic orders. It would be interesting to know what happens in the presence of other predicates such as the closest common ancestor, next sibling or child.

Probably the most natural continuation of this work would be an effective characterization of $\Sigma_2(<)$ or $\Sigma_2(<,<_{\text{lex}})$. Note that this would strengthen our result: a language L is definable in Δ_2 if and only if both L and its complement are definable in Σ_2. We conjecture that, as in the case for words [1], the characterization of $\Sigma_2(<,<_{\text{lex}})$ requires replacing the equivalence in the Δ_2 identity by a one-sided implication, which says that a language definable in $\Sigma_2(<,<_{\text{lex}})$ is closed under replacing v^ω by $v^\omega w v^\omega$, for $w \preceq v$.

References

[1] M. Arfi. Opérations polynomiales et hiérarchies de concaténation. *Theor. Comput. Sci.*, 91(1):71–84, 1991.

[2] M. Benedikt and L. Segoufin. Regular tree languages definable in FO and in FO+mod. To appear in *ACM Transactions on Computational Logic (TOCL)*. 2009.

[3] M. Bojańczyk. Two-way unary temporal logic over trees. In *Logic in Computer Science*, pages 121–130, 2007.

[4] M. Bojańczyk. Forest expressions. In *Computer Science Logic*, volume 4646 of *Lecture Notes in Computer Science*, pages 146–160, 2007.

[5] M. Bojańczyk and I. Walukiewicz. Characterizing EF and EX tree logics. *Theoretical Computer Science*, 358(2-3):255–273, 2006.

[6] M. Bojańczyk and I. Walukiewicz. Forest algebras. In *Automata and Logic: History and Perspectives*, pages 107 – 132. Amsterdam University Press, 2007.

[7] M. Bojańczyk, L. Segoufin, and H. Straubing. Piecewise testable tree languages. In *Logic in Computer Science*, 2008

[8] K. Etessami, M. Y. Vardi, and T. Wilke. First-order logic with two variables and unary temporal logic. *Inf. Comput.*, 179(2):279–295, 2002.

[9] R. McNaughton and S. Papert. *Counter-Free Automata.* MIT Press, 1971.

[10] J.-É. Pin. Logic, semigroups and automata on words. *Annals of Mathematics and Artificial Intelligence*, 16:343–384, 1996.

[11] J.-É. Pin and P. Weil. Polynomial closure and unambiguous product. *Theory Comput. Systems*, 30:1–30, 1997.

[12] A. Potthoff First-order logic on nite trees. In *TAPSOFT*, volume 915 of *Lecture Notes in Computer Science*, pages 125–139, 1995.

[13] M. P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.

[14] T. Schwentick, D. Thérien, and H. Vollmer. Partially-ordered two-way automata: A new characterization of DA. In *Devel. in Language Theory*, pages 239–250, 2001.

[15] I. Simon. Piecewise testable events. In *Automata Theory and Formal Languages*, pages 214–222, 1975.

[16] D. Thérien and T. Wilke. Over words, two variables are as powerful as one quantifier alternation. In *STOC*, pages 256–263, 1998.

[17] T. Wilke. Classifying discrete temporal properties. In *Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 32–46, 1999.