

# TREE LANGUAGES DEFINED IN FIRST-ORDER LOGIC WITH ONE QUANTIFIER ALTERNATION

MIKOŁAJ BOJAŃCZYK AND LUC SEGOUFIN

Warsaw University  
*e-mail address*: bojan@mimuw.edu.pl

INRIA - LSV  
*homepage* : <http://www-rocq.inria.fr/~segoufin>

---

ABSTRACT. We study tree languages that can be defined in  $\Delta_2$ . 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  $\Delta_2$ . This characterization is in terms of algebraic equations. Over words, the class of word languages definable in  $\Delta_2$  forms a robust class, which was given an effective algebraic characterization by Pin and Weil [11].

## 1. INTRODUCTION

We say a logic has a decidable characterization if the following decision problem is decidable: “given as input a finite automaton, decide if the recognized language can be defined using a formula of the logic”. Representing the input language by a finite automaton is a reasonable choice, since many known logics (over words or trees) are captured by finite automata.

This type of problem has been successfully studied for 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

---

Supported by Polish government grant no. N206 008 32/0810.

Work partially funded by the AutoMathA programme of the ESF.

been found for fragments of first-order logic with restricted quantification and 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 positions that do not have label  $a$ ” can be defined by the  $\Sigma_2(<)$  formula

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

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 combination 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  $\text{DA}$ ; 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.

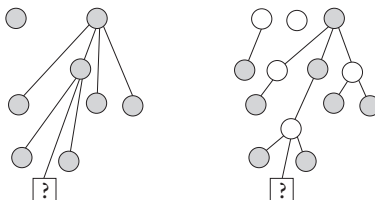
## 2. NOTATION

**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  $t_1, \dots, t_n$  is a finite sequence of trees, then  $t_1 + \dots + t_n$  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  $s, t, u, \dots$ . 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 an ancestor of  $y$  or, equivalently, that  $y$  is a strict descendant of  $x$ . As usual, we write  $x \leq y$  when  $x = y$  or  $x < y$ . 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  $\square$ , 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 contexts  $p$ , 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  $p = \square + s$  and  $q = \square + t$ .) 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. 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 contains 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  $\Sigma_2$  formula is a formula of the form

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

where  $\gamma$  is quantifier free. Languages defined in  $\Sigma_2$  are closed under disjunction and conjunction, but not necessarily negation. The negation of a  $\Sigma_2$  formula is called a  $\Pi_2$  formula, equivalently this is a formula whose quantifier prefix is  $\forall^* \exists^*$ . A forest property is called  $\Delta_2$  if it can be expressed both by a  $\Sigma_2$  and a  $\Pi_2$  formula. We will use  $\Sigma_2(<)$  and  $\Sigma_2(<, <_{\text{lex}})$  to specify which predicates are used in the signature, similarly for  $\Pi_2$  and  $\Delta_2$ .

**2.3. 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  $\Delta_2(<)$ .

One could ask, what about tree languages? There are two possible notions of a tree language  $M$  definable in  $\Delta_2(<)$ :

- (1) There exists a  $\Delta_2(<)$  formula  $\varphi$  such that for any *tree*  $t$ ,  $t \in M$  if and only if  $\varphi$  holds in  $t$ . In other words, there exists a  $\Delta_2(<)$  definable forest language  $L$  such that  $M$  is the intersection of  $L$  with the set of all trees.

- (2) There exists a  $\Delta_2(<)$  formula  $\varphi$  such that for any forest  $t$ ,  $t \in M$  if and only if  $\varphi$  holds in  $t$ . In other words, the language  $M$  is a  $\Delta_2(<)$  definable forest language that happens to contain only trees.

Our algorithm can tell us which tree languages are definable in  $\Delta_2(<)$  in the second notion. It turns out, however, that the two notions are equivalent (also for  $\Delta_2(<, <_{\text{lex}})$ ). The reason is that the forest property of “being a tree” is definable in  $\Delta_2(<)$ . 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. Hence a solution of the problem for forest languages also gives a solution for tree languages.

As noted earlier, the corresponding problem for words was solved by Pin and Weil: 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  $\omega$  means that the identity holds for sufficiently large powers (in different settings,  $\omega$  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.

**2.4. Basic properties of  $\Pi_1$  and  $\Sigma_2$ .** 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.

We begin by describing the expressive power of purely universal formulas  $\Pi_1$ .

**Lemma 2.1.** *A forest language is closed under pieces if and only if it is definable in  $\Pi_1$ . Likewise for context languages.*

*Proof.* Let  $L$  be a forest language that is closed under pieces, and let  $\alpha : A^\Delta \rightarrow (H, V)$  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  $\Pi_1(<, <_{\text{lex}})$ .

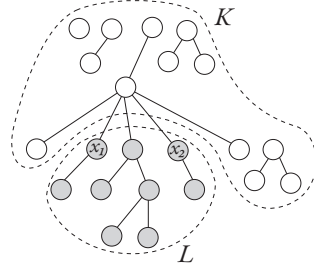
If, additionally, the language  $L$  is commutative, then we do not need to worry about the lexicographic order when talking about the pieces in  $T$ , only the descendant order is relevant.  $\square$

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

**Lemma 2.2.** *Let  $K, K'$  be context languages and  $L$  a forest language. 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  $KK' = \{qq' : q \in K, q' \in K'\}$ .*

*Proof.* We only do the proof for  $KL$ , the others are treated similarly. The formula places two existentially quantified variables  $x_1$  and  $x_2$  for the leftmost and rightmost root of the trees that are expected to form the forest in  $L$ , that is two sibling nodes such that  $x_1 \leq_{\text{lex}} x_2$ .



Note that it is possible to express in  $\Sigma_2(<)$  that two existentially quantified nodes are siblings. One quantifies existentially a common ancestor and uses universal quantification for checking that this common ancestor is a parent of both nodes. Once  $x_1$  and  $x_2$  are available, membership in  $KL$  is obtained by relativizing all quantification in the  $\Sigma_2(<, <_{\text{lex}})$  formula for  $L$  to the nodes that are either descendants of  $x_2$  or between  $x_1$  and  $x_2$  in the lexicographic order  $<_{\text{lex}}$ , and doing a dual relativization in the  $\Sigma_2(<, <_{\text{lex}})$  formula for  $K$ .  $\square$

We define  $\Sigma_2$  forest expressions and  $\Sigma_2$  context expressions by mutual recursion:

- Any forest (respectively, context) language that is closed under pieces is a  $\Sigma_2$  forest (respectively, context) expression. For any label  $a \in A$ ,  $\{a\Box\}$  is a  $\Sigma_2$  context expression, likewise for  $\{\Box\}$ .
- If  $K, K'$  are  $\Sigma_2$  context expressions and  $L, L'$  are  $\Sigma_2$  forest expressions, then
  - $K \cdot K'$  is a  $\Sigma_2$  context expressions;
  - $L + L'$  is a  $\Sigma_2$  forest expression;
  - $K \cdot L$  is a  $\Sigma_2$  forest expression.

From Lemma 2.2 and Lemma 2.1 it follows immediately that  $\Sigma_2$  forest and context expressions are definable in  $\Sigma_2(<, <_{\text{lex}})$ .

### 3. 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,  $\square \cdot h = h$  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  $(\square + g)$  and  $(g + \square)$  defined by  $(\square + g)h = h + g$ ,  $(g + \square)h = g + h$  for all  $h \in H$ .

A morphism  $\alpha : (H_1, V_1) \rightarrow (H_2, V_2)$  of forest algebras is actually a pair  $(\alpha_H, \beta_V)$  of monoid morphisms 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  $\alpha$ .

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  $A^\Delta$ .

We say that a forest algebra  $(H, V)$  *recognizes* a forest language  $L \subseteq H_A$  if there is a morphism  $\alpha : A^\Delta \rightarrow (H, V)$  and a subset  $X$  of  $H$  such that  $L = \alpha^{-1}(X)$ . A forest language is regular 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  $\omega$  when  $M$  is understood from the context) such that for all element  $x$  of  $M$ ,  $x^\omega$  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  $u^\omega$  and  $\omega(g)$  for the corresponding idempotents.

Given  $L \subseteq H_A$  we define an equivalence relation  $\sim_L$  on  $H_A$  by setting  $s \sim_L s'$  if and only if for every context  $p \in V_A$ ,  $ps$  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  $A^\Delta$ , 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  $v \in V$  is a piece of a context  $w \in V$  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

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

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 contexts  $v \preceq w$ .

#### 4. CHARACTERIZATION OF $\Delta_2(<, <_{\text{lex}})$

In this section we present the main result of this paper:

**Theorem 4.1** (Effective characterization of  $\Delta_2$  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  $\Delta_2$  identity,*

$$v^\omega w v^\omega = v^\omega \quad \text{for } w \preceq v. \quad (4.1)$$

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  $\Delta_2$  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].  $\square$

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

**Lemma 4.3.** *Let  $\varphi$  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  $\varphi$  are closed under replacing  $p^n p^n$  with  $p^n q p^n$ .*

*Proof.* Assume that  $\varphi$  is  $\exists x_1 \cdots x_k \forall y_1 \dots y_l \psi$ , with  $\psi$  quantifier-free. Any first-order definable tree language is aperiodic, 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  $\varphi$ . We want to show that the forest  $rp^n q p^n s$  also satisfies  $\varphi$ . By aperiodicity, it is sufficient to show that for some numbers  $i, j > m$ ,  $xp^i qp^j s$  satisfies  $\varphi$ .

Because  $t$  satisfies  $\varphi$  we can fix  $k$  nodes  $x_1, \dots, x_k$  that make  $\forall y_1 \dots y_l \psi$  true. By the choice of  $n$ ,  $t$  can be decomposed as  $xp^i p^l p^j s$  such that  $i, j \geq m$  and the middle  $p^l$  part contains no quantified variables. We show that  $t' = xp^i p^l q p^j s$  satisfies  $\varphi$ , 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  $x_1, \dots, x_k$  that we fixed above. We show that this valuation, when seen as nodes of  $t'$ , makes  $\varphi$  true. Indeed, for any valuation for the variables  $y_1, \dots, y_l$  in  $t'$ , we show that the valuation makes the quantifier-free part  $\psi$  true. This is obvious if none of the  $y_i$  are in  $q$  because  $\varphi$  holds in  $t$  and the insertion of  $q$  does not affect the relationship  $<$  and  $<_l$  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. Therefore, because  $q$  is a piece of  $p$ , the part of  $p^l q$  of  $t'$  that contains variable can be mapped to the part  $p^l$  of  $t$  without affecting the  $<$  and  $<_l$  relationship. Hence  $\psi$  must hold with the new valuation.  $\square$

The rest of the paper contains the more difficult implication of Theorem 4.1. This main proposition is given below.

**Proposition 4.4.** *Fix a morphism  $\alpha : A^\Delta \rightarrow (H, V)$ , with  $(H, V)$  satisfying the  $\Delta_2$  identity. For each  $h \in H$ , the set  $L_h$  of forests  $t$  with type  $\alpha(t) = h$  is definable by a  $\Sigma_2$  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  $\Sigma_2$  expressions allow union, the above proposition shows that any language recognized by  $\alpha$  can be defined by a  $\Sigma_2$  forest expression. In particular, if  $L$  is recognized by  $\alpha$ , then both  $L$  and its complement can be defined by  $\Sigma_2$  forest expressions, and consequently formulas of  $\Sigma_2(<, <_{\text{lex}})$ . Since the complement of an  $\Sigma_2(<, <_{\text{lex}})$  formula is a  $\Pi_2(<, <_{\text{lex}})$  formula, we get the right to left implication in Theorem 4.1.

The rest of this section is devoted to showing Proposition 4.4. The proof is by induction on two parameters. The first parameter is the size of the algebra. The second parameter is the position of  $h$  in a certain pre-order defined below. The second parameter corresponds to a bottom-up pass through the forest, as the types  $h$  that are closer to the induction base are types of forests that are close to the leaves.

This induction proof is presented in Section 5. However, the induction breaks down for certain types, which are treated with a different technique. This technique is presented in Section 7.

## 5. TYPES OUTSIDE $H_{\perp}$

We now define the pre-order on  $H$ , which is used in the induction proof of Proposition 4.4. We say that a type  $h$  is *reachable from* a type  $g$  if there is a context  $v \in V$  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 any type  $g \in H$ . Note that  $H_{\perp}$  is not empty, since it contains the type  $h_1 + \dots + h_n$ , for any enumeration  $H = \{h_1, \dots, h_n\}$ .

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.

Let then  $h \in H$  be a forest type. By induction, using Proposition 4.4, for each type  $g \not\sim h$  from which  $h$  is reachable, we have a  $\Sigma_2$  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  $\Sigma_2$  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

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

We say that a context type  $v$  stabilizes  $h$  if it belongs to the stabilizer of  $h$ . The key lemma is that the  $\Delta_2$  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  $\text{stab}(h) = \text{stab}(h')$ . Assume  $v \in \text{stab}(h)$ . 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  $\Delta_2$  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  $v \in \text{stab}(h')$ .

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

Recall now the piece order on forest types from the end of Section 3, 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  $\Delta_2$  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  $v$  such that  $h = vuh$ . We are looking for a context  $w$  such that  $wu'h = h$ . From  $h = vuh$  we get  $h = (vu)^\omega h$ . Hence by the  $\Delta_2$  identity we have  $h = (vu)^\omega u' (vu)^\omega h = (vu)^\omega u' h$  as desired.  $\square$

We now consider two possible cases: either  $h + h \sim h$ , or not. Equivalently, we could have asked if  $h + \square$  stabilizes  $h$ , or if  $\square + h$  stabilizes  $h$ . In the first case we will conclude by induction on the size of the algebra while in the second case we will conclude by induction on the reachability pre-order. These 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  $(G, \text{stab}(h))$  is a forest algebra.*

*Proof.* We need to show that the two sets are closed under all operations.

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

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  $v \in \text{stab}(h)$  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,  $G$  is a proper subset of  $H$ , and 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  $\Sigma_2$  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  $\Sigma_2$  forest expression. Our goal is to give a  $\Sigma_2$  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

$$L_h = \bigcup_{u \in \text{stab}(h)} \left( \bigcup_{\substack{a \in A, g \in G \\ uag=h}} K_u a L_g \quad \cup \quad \bigcup_{\substack{g_1, g_2 \in G \\ u(g_1+g_2)=h}} K_u (L_{g_1} + L_{g_2}) \right) \quad (5.1)$$

Note that  $L_g, L_{g_1}$  and  $L_{g_2}$  can all be written as  $\Sigma_2$  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  $\Sigma_2$  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 concatenation of the form  $K_1 \cdots K_m$  where each  $K_i$  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  $\Sigma_2$  context expression for each language  $K_u$  in (5.1). The singleton languages, and the languages closed under pieces are  $\Sigma_2$  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 (5.1) 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  $\Sigma_2$  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  $\Sigma_2$  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:

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

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

$$(mn)^\omega m (mn)^\omega = (mn)^\omega \tag{6.1}$$

*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  $B_i^*$  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 (6.1).*

*Proof.* The only if implication is shown using a standard Ehrenfeucht-Fraïssé argument, we only consider the if implication.

Let then  $L \subseteq A^*$  be a language recognized by a morphism  $\beta : A^* \rightarrow M$ , with  $M$  satisfying (6.1). 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  $\Pi_2(<)$ .  $\square$

We now proceed to the proof of Proposition 6.1. The proof is the same as that of Thérien and Wilke, i.e. 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 original proof of Thérien 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  $\Sigma_2$  *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  $\Sigma_2$  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 (6.1), is:

$$m \sim_{\mathcal{R}} n \wedge m \sim_{\mathcal{L}} n \Rightarrow m = n \quad \text{for } m, n \in M . \quad (6.2)$$

We will also use the following property of monoids satisfying (6.1), which can be proved along the lines of Lemma 5.1.

$$m \sim_{\mathcal{R}} n \sim_{\mathcal{R}} mk \Rightarrow nk \sim_{\mathcal{R}} n \quad \text{for } m, n, k \in M . \quad (6.3)$$

**Lemma 6.3.** *For all  $m \in M$ , the language  $U_m = \{w : m\beta(w) \sim_{\mathcal{R}} m\}$  is definable by a  $\Sigma_2$  word expression.*

*Proof.* Thanks to (6.3), a word belongs to  $U_m$  if and only if all of its letters are in

$$A_m = A \cap \beta^{-1}(N) \quad \text{where } N = \{n : mn \sim_{\mathcal{R}} m\} .$$

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  $n^\omega kn^\omega = n^\omega$ . 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 (6.3), and hence  $k \in N$ .  $\square$

Note that by (6.3) 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  $\Sigma_2$  word expression:*

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

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  $\beta(u) \sim m$ . Hence we have:

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

Since  $\Sigma_2$  word expressions are closed under union and concatenation, the above language is definable by a  $\Sigma_2$  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  $\Sigma_2$  word expression for  $\{w : \beta(w) \sim_{\mathcal{L}} m\}$ . But we also know from (6.2) 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  $\Sigma_2$  word expressions under intersection.

*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  $\Sigma_2$  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  $n = \beta(u_1) \sim k$ . We therefore have:

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

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  $\Sigma_2$  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 (6.3) 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  $\beta'$  be the restriction of  $\beta$  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  $\Sigma_2$  word expression for  $U_{n,k}$ . This concludes the proof of this lemma as  $\Sigma_2$  word expressions are closed under concatenation and union.  $\square$

**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. Recall that for a context type  $v \in V$ , we are looking for 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  $\beta : B \rightarrow V$ , which is simply  $\alpha$  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  $n_1, \dots, n_k$  are elements of  $V$ , and  $N_1, \dots, N_k$  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 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  $\Sigma_2$  word expression, since it distributes across concatenation.)

## 7. TYPES IN $H_\perp$

Recall that in Section 5, we managed to find a  $\Sigma_2$  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. Loosely speaking, we will try to determine how a forest  $t$  affects the type of  $pt$ , for a given context  $p$ . The idea is that the bigger the context  $p$ , the smaller the effect of  $t$  on the type of  $pt$ . The size of the context  $p$  is the parameter in our induction, which will start with large contexts  $p$  and then progress on through to smaller contexts (a smaller context has the hole closer to the root).

The formal statement of the main result, Proposition 7.1, is stated in terms of an equivalence relation  $\equiv_v$ . Given a context type  $v \in V$  and two forest types  $h, h'$ , we write

$$h \equiv_v h' \quad \text{if} \quad \forall u \in V \quad vuh = vuh' . \quad (7.1)$$

We extend this equivalence relation to context types, by

$$w \equiv_v w' \quad \text{if} \quad \forall u \in V \quad \forall h \in H \quad vuw h = vuw' h . \quad (7.2)$$

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  $\equiv_v$  is definable by a  $\Sigma_2$  forest expression.*

From Proposition 7.1 we immediately obtain a  $\Sigma_2$  forest expression for  $L_h$ , as the equivalence class of  $\equiv_v$  containing  $h$ , where  $v = \square$ . 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  $\Sigma_2$  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 included in  $H \setminus H_\perp$ , then it can easily be defined by the disjunction of all the  $\Sigma_2$  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 contexts. We say that a context  $u$  is a *prefix* of a context  $v$  if there exists a context  $w$  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 contexts that have no proper extension, and ending at the context  $v = \square$  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  $v$ , there is only one equivalence class of  $\equiv_v$ , and this class is defined by the set of all forests, which is a  $\Sigma_2$  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:

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

When  $u \in \text{stab}(v)$ , we say that  $u$  stabilizes  $v$ . As for stabilizers of forest types, the  $\Delta_2$  identity implies that the stabilizer  $\text{stab}(v)$  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. In the proof of Lemma 7.3, we use the  $\Delta_2$  identity, while Lemma 7.2 holds already under a weaker assumption, namely that the monoid  $H$  is aperiodic. Nevertheless, the lemma below has a simpler proof, and it will also be referenced later on. Note that  $H_\perp$  is a single class relative of the reachability equivalence relation  $\sim$ . This is because for any  $f, f' \in H_\perp$ , we have

$$f \sim f' + f \sim f' .$$

**Lemma 7.3.** *If both  $H_\perp + \square$  and  $\square + H_\perp$  intersect  $\text{stab}(v)$ , then the context  $v$  is constant.*

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

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

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

However, the context on the left side of the identity is constant, since any forest type  $h \in H$  is a piece of  $f$ , and therefore by (4.1) we have

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

□

**7.2. The induction step.** We now proceed to the induction step in Proposition 7.1. Recall that our goal is to find a  $\Sigma_2$  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.

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

We use a similar notation  $[K]_v$  for languages of contexts. We say that a forest language  $L^\uparrow$  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  $\Sigma_2$  forest expression.*

The above result concludes Proposition 7.1. To see this, consider an equivalence class consisting of forest types  $f_1, \dots, f_n$ . 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  $\Sigma_2$  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  $\Sigma_2$  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 either  $\square + H_\perp$  or  $H_\perp + \square$  is disjoint with the stabilizer of  $v$ . Without loss of generality we assume

$$\square + H_\perp \cap \text{stab}(v) = \emptyset. \quad (7.3)$$

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 of type in  $H_\perp$  can be decomposed as  $t = ps$ , such that the type of the context  $p$  stabilizes  $v$ , and the forest  $s$  is such that either*

- (1)  $s = 0$ ; or
- (2)  $s = as'$  with the type of the context  $a\square$  not stabilizing  $v$ ; or
- (3)  $s = s_1 + s_2$  with the types of the contexts  $s_1 + \square$  and  $\square + s_2$  not stabilizing  $v$ .

*Proof.* A context  $p$  is called a *prefix* of a forest  $t$  if  $t = ps$  for some forest  $s$ . Let  $p$  be a prefix of  $t$  that has a type that stabilizes  $v$ , and maximal (with respect to context extensions) for that property. Let  $s$  be such that  $t = ps$ . We show that  $s$  satisfies one of the properties listed in the statement of the lemma. If  $s$  is empty, we get item (1). If  $s$  is a tree,  $s = as'$ , then by choice of  $p$ , the type of  $a\square$  cannot stabilize  $v$  and therefore (2) holds. Assume now that  $s$  is a forest consisting of (at least two) trees, with the decomposition into trees being  $t_1 + \dots + t_n = s$ . By choice of  $p$ , the type of the context  $\square + t_n$  cannot stabilize  $v$ . Likewise, the type of the context  $t_1 + \dots + t_{n-1} + \square$  cannot stabilize  $v$ . Therefore we get item (3) by picking  $s_1 = t_1 + \dots + t_{n-1}$  and  $s_2 = t_n$ .  $\square$

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

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

where  $Y_f$  stands for the set of all forests  $s$  that have type  $f$  and that satisfy one of the conditions (1)-(3) of Lemma 7.5. To get the  $v$ -overapproximation of  $L_h$  will use  $v$ -overapproximations for the smaller expressions above, as stated by the following two lemmas. The statement of the first lemma, concerning  $Y_f$ , is straightforward.

**Lemma 7.6.** *For every  $f \in G$ , some  $v$ -overapproximation  $\hat{Y}_f$  of  $Y_f$  can be defined by a  $\Sigma_2$  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  $u \in \text{stab}(v)$ , one can define a  $\Sigma_2$  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, though, 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

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

is a  $v$ -overapproximation of  $L_h$ .

The first property required from  $v$ -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

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

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

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

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 algebra.

**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  $\Sigma_2$  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  $\Sigma_2$  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

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

Containing such a subforest can be expressed by a  $\Sigma_2$  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  $\Sigma_2$  forest expressions thanks to the section on types outside  $H_\perp$ .  $\square$

**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  $\Sigma_2$  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

$$K_u = \sum_i K_{i,1} \cdots K_{i,n_i} \tag{7.4}$$

The problem with the expression above is that it may use, in some of the subexpressions  $K_{i,j}$ , languages  $M_f + \square$  or  $\square + M_f$  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 that there is a  $\Sigma_2$  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 (7.4).

This only problem with the overapproximation is when  $K_{i,j}$  is of the form  $M_f + \square$  or  $\square + M_f$ , for  $f \in H_\perp$ . In all the other cases,  $K_{i,j}$  is known to be definable by a  $\Sigma_2$  context expression, and no overapproximation is needed. Note that by (7.3), the expressions  $\square + L_f$  cannot be used, since a forest type from  $H_\perp$  cannot appear to the right of the hole in a context 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  $M_f + \square$  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

$$[L]_{v+} = \bigcap_{u \notin \text{stab}(v)} [L]_{vu} \quad (7.5)$$

By the induction assumption in Proposition 7.1, each equivalence class of  $\equiv_{v+}$  is definable by a  $\Sigma_2$  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  $K = M_f + \square$  satisfies (\*) with  $\hat{K} = [L_f]_{v+} + \square$ . Assume that the type of  $s$  is in  $H_\perp$ . Then we have:

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

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

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

This inclusion holds because we have:

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

In the second equality, we used the definition of  $[L_f]_{v+}$  and the assumption that  $\square + \alpha(s)$  does not stabilize  $v$ . The latter follows from assumption (7.3) since the type of  $s$  is in  $H_\perp$ .

**7.2.2. Proof of Lemma 7.6.** In this section, we need to show that for every type  $f \in H_\perp$ , a  $v$ -overapproximation of  $Y_f$  can be defined by a  $\Sigma_2$  forest expression. Recall that the language  $Y_f$  was defined based on a case distinction in Lemma 7.5, being a union of three components, one for each of the three cases in the lemma. As  $v$ -overapproximation are closed under union, for each of these components, we provide a  $v$ -overapproximation defined by a  $\Sigma_2$  forest expression.

The case (1) is trivial.

For case (2) we have a union of sets

$$a \cdot L_h \quad \text{where the type of } a \text{ does not stabilize } v, h \in H, \text{ and } ah = f. \quad (7.6)$$

It may be the case that  $h$  belongs to  $H_\perp$  and therefore we have no  $\Sigma_2$  forest for 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  $\Sigma_2$  forest expression for  $[L_h]_{va}$ . But then the  $\Sigma_2$  forest expression  $a \cdot [L_h]_{va}$  is a  $v$ -overapproximation of  $a \cdot L_h$ .

It remains to consider the case of (3). In this case we have a union of sets

$$L_{h_1} + L_{h_2} \quad h_1 + \square, \square + h_2 \notin \text{stab}(v) . \quad (7.7)$$

We consider two cases.

The first case is when  $h_1 \notin H_\perp$ . Therefore we have a  $\Sigma_2$  forest expression for  $L_{h_1}$ . In this case we claim that

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

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

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

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  $g_2 \equiv_{v+} h_2$ . Since  $h_1 + \square$  does not stabilize  $v$ , this means that  $g_2 \equiv_{v(h_1 + \square)} h_2$ , and therefore also  $h_1 + g_2 \equiv_v h_1 + h_2$ .

The second case is when  $h_1 \in H_\perp$ . Then we have:

$$H_\perp + \square \cap \text{stab}(v) = \emptyset . \quad (7.8)$$

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

$$([L_{h_1}]_{v+} \cap L_\perp) + ([L_{h_2}]_{v+} \cap L_\perp) . \quad (7.9)$$

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

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

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

$$v(g_1 + g_2) = v(h_1 + h_2) .$$

Since  $t_1 \in L_\perp$ , then by (7.8) the context type  $g_1 + \square$  does not stabilize  $v$ . Therefore, we can use the assumption on  $g_2 \equiv_{v+} h_2$  to infer

$$v(g_1 + g_2) = v(g_1 + \square)g_2 = v(g_1 + \square)h_2 = v(g_1 + h_2) .$$

In a similar way, we use  $h_2 \in H_\perp$ , the assumption (7.3), and  $g_1 \equiv_{v+} h_1$ , to complete the proof of this case, and of Lemma 7.6.

$$v(g_1 + h_2) = v(\square + h_2)g_1 = v(\square + h_2)h_1 = v(h_1 + h_2) .$$

## 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  $\Delta_2$  with the descendant order).

*A forest language is definable in  $\Delta_2(<)$  if and only if its syntactic forest algebra satisfies the  $\Delta_2$  identity, as well as horizontal commutativity:*

$$h + g = g + h . \quad (8.1)$$

The “only if” implication is easy: we have already shown that the  $\Delta_2$  identity must hold in the syntactic forest algebra of a language definable in  $\Delta_2(<, <_{\text{lex}})$ , even more so for a language definable in  $\Delta_2(<)$ . 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  $\Delta_2$  identity, then both  $L$  and its complement can be defined by  $\Sigma_2$  forest expressions. We apply this result also in our case. The problem is that the  $\Sigma_2$  forest expressions are not commutative, and thus need not be definable in  $\Sigma_2(<)$ . We will show, however, that their commutative closure can be define 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  $\Sigma_2$  forest expression is definable in  $\Sigma_2(<)$ .*

Before we prove this proposition, we remark that this is not a completely generic result. For instance, the language “Each node is a leaf or has two children, and some leaf has an even number of ancestors” is definable in  $\Sigma_3(<, <_{\text{lex}})$  and is horizontally commutative (the formula comes from Potthoff [12]). However, it cannot be defined by a  $\Sigma_3(<)$  formula (actually, not even by any first-order formula with just the descendant order).

The proof of the above proposition is by induction on the size of the  $\Sigma_2$  forest expression.

The base case is when the  $\Sigma_2$  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 2.1, which showed that a language  $L$  that is closed under pieces is definable in  $\Pi_1$ . If we take the commutative closure of  $L$ , we get a commutative language closed under pieces. In the proof of Lemma 2.1, we constructed the  $\Pi_1$  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  $\Sigma_2$  expressions: union, intersection, (horizontal) concatenation

$$L + L' \tag{8.2}$$

and (vertical) composition

$$K \cdot L \quad K \cdot K' \tag{8.3}$$

for a forest languages  $L, L'$  and context languages  $K, K'$ . Note that  $\Sigma_2(<)$  is not closed under the operations above. 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 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 (8.2) and (8.3) can be defined in  $\Sigma_2(<)$ . We only do the cases  $L + L'$  and  $K \cdot K'$ , the others are done the same way.

**Lemma 8.3.** *If forest languages  $L, L'$  are defined 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.

Consider now the general case in  $L \oplus L'$ . Any 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 the  $M_i$  are tree languages definable in  $\Sigma_2(<)$ . Since  $\oplus$  distributes across union, it suffices to show 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, we can assume that there are no tree languages, and only a formula for  $L_0 \oplus L'_0$  is needed. But the language  $L_0 \oplus L'_0$  is closed under pieces, and therefore it is definable in  $\Pi_1(<)$ .  $\square$

**Lemma 8.4.** *If context languages  $K, K'$  are defined 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  $\{a\Box\}$ . The formula places a variable on node corresponding to  $a\Box$ , and relativizes the formula for the remaining context language to the remaining nodes.

Consider now the general case. The language  $K$  can be decomposed as a finite union of languages of the form

$$\hat{K} \odot \{a\Box\} \odot (\Box \oplus L) \quad \text{or} \quad \Box \oplus L .$$

Likewise,  $K'$  can be decomposed as a finite union of languages

$$(\Box \oplus L') \odot \{a'\Box\} \odot \hat{K}' \quad \text{or} \quad \Box \oplus L' .$$

Since the operation  $\odot$  distributes across union, we can assume that  $K$  and  $K'$  are simply of the forms above, and not finite unions. We have four cases to consider, we only do the most difficult one

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

This language is the same as

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

The  $\Sigma_2(<)$  formula puts an existentially quantified variable  $x$  on the node corresponding to  $a\Box$ , another existentially quantified variable  $x'$  on the node corresponding to  $a'\Box$ , and then runs three subformulas, for  $\hat{K}$ ,  $\Box \oplus L \oplus L'$ , and  $\hat{K}'$ , on the remaining nodes, relativizing the quantification to nodes that are, respectively, not descendants of  $x$ , descendants of  $x$  but not of  $x'$ , and, finally, descendants 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  $\Delta_2$  if and only if both  $L$  and its complement are definable in  $\Sigma_2$ . We conjecture that, as in the case for words [1], the characterization of  $\Sigma_2(<, <_{\text{lex}})$  requires replacing the equivalence in the  $\Delta_2$  identity by a one-sided implication, which says that a language definable in  $\Sigma_2(<, <_{\text{lex}})$  is closed under replacing  $v^\omega$  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.