Forest Algebras

Mikołaj Bojańczyk[1]
Igor Walukiewicz[2]

[1] Institute of Informatics
Warsaw University
Banacha 2, 02-097 Warsaw, Poland

[2] CNRS LaBRI
Université de Bordeaux
351, Cours de la Libération
33405 Talence, France

Abstract

There are at least as many interesting classes of regular tree languages as there are of regular word languages. However, much less is known about the former ones. In particular, very few decidable characterizations of tree language classes are known. For words, most known characterizations are obtained using algebra. With this in mind, the present paper proposes an algebraic framework for classifying regular languages of finite unranked labeled trees.

If in a transformation semigroup we assume that the set being acted upon has a semigroup structure, then the transformation semigroup can be used to recognize languages of unranked trees. This observation allows us to examine the relationship connecting tree languages with standard algebraic concepts such as aperiodicity idempotency, or commutativity. The new algebraic setting is used to give several examples of decidable algebraic characterizations.

1 Introduction

There is a well-known decision problem in formal language theory:

> Decide if a given a regular language of finite binary trees can be defined by a formula of first-order logic with three relations: ancestor, left and right successor.

If the language is a word language (there is only one successor relation in this case) the problem is known to be decidable thanks to fundamental results of Schützenberger [14] and McNaughton and Papert [11]. The problem is also decidable for words when only the successor relation is available [18, 1].

However, no algorithm is known for the case of tree languages, see [9, 13, 3, 2] for some results in this direction.

There is a large body of work on problems of the type: decide if a given regular word language can be defined using such and such a logic [6, 12, 15, 19, 20, 22]. Most of the results have been obtained using algebraic techniques of semigroup theory. Recently, there has even been some progress for tree languages [21, 8, 5, 2]. There is, however, a feeling that we still do not have the right algebraic tools to deal with tree languages. In this paper we propose an algebraic framework, called forest algebras, and study the notion of recognizability in this framework. We want it to be as close to the word case as possible to benefit from the rich theory of semigroups. We show how standard notions, such as aperiodicity, idempotency, or commutativity, can be used in our framework to characterize classes of tree languages.

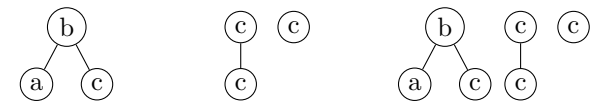
Forest algebras are defined for forests (ordered sequences) of unranked trees, where a node may have more than two (ordered) successors. This more general (more general than, say, binary trees) setting is justified by cleaner definitions, where semigroup theory can be used more easily.

We begin our discussion of forest algebras with the free forest algebra. Just as the free monoid is the set of words, the free forest algebra is going to be the set of forests. For finite words, there is one natural monoid structure: concatenation of words with the empty word as a neutral element. For forests there is also a concatenation operation that puts one forest after the other (see Figure 1). This operation though, has a very limited power as the depth of the resulting forest is the maximum of the depths of the arguments. One needs also some kind of vertical composition that makes forests grow. This requires a notion of a context, which is a forest with a single hole in some leaf. Contexts can be composed by putting one of them in the hole of the other (see Figure 2). Moreover, by putting a forest in the hole of a context we obtain again a forest. Summarizing, for unranked, ordered, finite forests there are two natural monoids:

- *Horizontal monoid.* Forests with concatenation, and the empty tree as a neutral element.
- *Vertical monoid.* Contexts with context composition, and the context with a hole in the root as a neutral element.

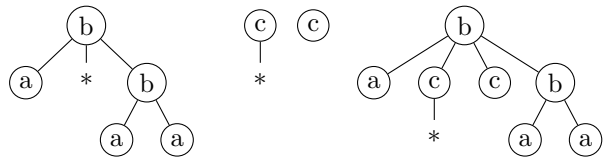
The two monoids are linked by an action of contexts on forests: if p is a context and t is a forest then pt is a forest obtained by putting t in the hole of p in the same way as the contexts are composed.

In the case of words, a language of finite words induces a congruence, the Myhill-Nerode equivalence relation, which has finite index whenever the language is regular. The same concepts apply to forest algebras, except that we get two congruences: one for the vertical semigroup and one for the



A forest t_1 A forest t_2 The resulting forest $t_1 + t_2$

FIGURE 1. Forest concatenation



A context p A context q The resulting context pq

FIGURE 2. Context composition

horizontal semigroup. A regular language of finite forests can be thus seen as one where both congruences are of finite index.

An important property of a forest algebra is that it is a special case of a transformation semigroup. Recall that a *transformation semigroup* is a semigroup along with an action over a set. In the forest algebra, the acting semigroup is the set of contexts, while that set acted upon is the set of forests (which itself is equipped with a semigroup structure).

There is a well-developed theory of transformation semigroups that is useful in classifying regular word languages. We hope that this theory might extend to the case of trees and this paper presents first steps in this direction. To illustrate how forest algebra can be used in classifying regular languages, we show how two language classes—forest languages determined by the labels occurring in a forest, and forest languages definable by a Σ_1 formula—can be described in terms of forest algebra. We also present a more involved example: languages definable in the temporal logic EF.

Acknowledgments We would like to thank Olivier Carton, Jean-Eric Pin, Thomas Schwentick, Luc Segoufin and Pascal Weil for their helpful comments. Special thanks are due to Howard Straubing, for correcting several errors in a previous version, and suggesting some improved definitions.

2 Preliminaries

The set of trees and forests over a finite alphabet A is defined as follows:

- an empty tree, denoted 0, is a tree (and therefore also a forest);
- if s, t are forests, then $s + t$ is a forest; moreover $+$ is associative;
- If s is a forest, then as is a tree (and also a forest) for every $a \in A$.

The empty tree is a neutral element for the operation $+$ of forest concatenation. This operation is in general non-commutative. A tree is a forest of the form as, where s is a forest. We denote trees, as well as forests, by s, t and u. Most of the time we will be working with forests and we will say explicitly when a variable denotes a tree.

It will be convenient to interpret a forest as a partial function $t : \mathbb{N}^+ \to A$ with a finite domain (the roots of this forest are the nodes from $\mathbb{N}$). Elements of this finite domain are called *nodes* of t. (The domain is closed under nonempty prefixes, and if $y < y'$ are natural numbers with $x \cdot y'$ in the domain, then also $x \cdot y$ belongs to the domain.) This function assigns to each node its *label*. If x, y are two nodes of t, we write $x \leq y$ ($x < y$) if x is a (proper) prefix of y (i.e x is closer to the root than y). If x is a maximal node satisfying $x < y$, then we call x the *parent* of y and we call y a *successor* of x. (Each node has at most one parent, but may have many successors.) Two nodes are *siblings* if they have the same parent. A *leaf* is a node without successors. The *subtree of t rooted in the node x*, denoted $t|_x$, assigns the label $t(x \cdot y)$ to a node $0 \cdot y$. The *successor forest* of a node is the forest of subtrees rooted in that node's successors.

An *A-context* is an $(A \cup \{*\})$-forest, where $*$ is a special symbol not in A. Moreover, $*$ occurs in exactly one leaf, which is called the *hole*. We use letters p, q to denote contexts. When p is a context and t is a forest, pt is the forest obtained from p by replacing the hole with t (see Figure 2). Similarly we define the composition of two contexts p, q – this is the context $p \cdot q$ that satisfies $(p \cdot q)t = p(qt)$ for every forest t. The neutral element of context composition is a context, denoted 1, consisting only of a single node labeled $*$.

3 Forest algebras

In this section we formally define a forest algebra. We give some examples and explore some basic properties.

A *forest algebra* (H, V, act, in_l, in_r) consists of two monoids H, V, along with an action $act : H \times V \to H$ of V on H and two operations $in_l, in_r : H \to V$. We denote the monoid operation in H by $+$ and the monoid operation in V by $\cdot$. The neutral elements of the two monoids will be denoted respectively: 0 and 1. Instead of writing $act(h, v)$, we write vh (notice a reversal of arguments). A forest algebra must satisfy the following axioms:

action $(v \cdot w)h = v(wh)$;

insertion $in_l(g)h = g + h$ and $in_r(g)h = h + g$;

faithfulness for every two distinct $v, w \in V$ there is $h \in H$ with $vh \neq wh$;

We call V the *vertical monoid* and H the *horizontal monoid*. Thanks to the action axiom it is unambiguous to write vwh. Most of the time we will omit the act, in_l, in_r from (H, V, act, in_l, in_r) and write (H, V), just as we identify a monoid with its carrier set. We will also sometimes write $h + 1$ instead of $in_l h$, and $1 + h$ instead of $in_r h$.

Example 3.1. Let H be any monoid. Let V be the set H^H of all transformations of H into H, with composition as the operation. To obtain a forest algebra from (H, V) it suffices to add the action and in_l, in_r. We can take the action of V on H to be just function application. The operations in_l and in_r are then determined by the insertion axiom. Faithfulness can be easily verified.

Note 3.2. As mentioned earlier, we have chosen to write the action on the left, while the standard practice in the algebraic study of languages of words is to write it on the right. That is, we write $act(h, v)$ as vh, while in most papers on monoids and word languages one would see hv. We feel that this choice is justified by the difference in the way words and trees are denoted. In the case of words, writing the action on the right is justified by the way words are written (with the first letter on the left) as well as the way finite automata read the input (from left to right). For example, if one wants to calculate the action of a word abb on a state q of an automaton, one writes $qf_a f_b f_b$; where f_a, f_b are the actions associated with the corresponding letters. Using standard functional notation this would give $f_b(f_b(f_a(q)))$. Hence, writing action on the right saves tiresome reversal of the word. For trees the situation is different. Usually, one describes trees with terms. So $a(t_1 + t_2)$ denotes a tree with the root a and two subtrees t_1 and t_2. If we were writing actions on the right, the value of this tree would be denoted by $(h_1 + h_2)v_a$, where h_i is the value of t_i and v_a is the value of a. In consequence, writing the action to the right corresponds to writing terms in reverse Polish notation. Writing the action on the right would thus force us either: to do the conversion into reverse Polish notation each time we go from trees to algebra, or to write trees in reverse Polish notation. The authors think that both options are more troublesome than the choice of writing action on the left.

Note 3.3. Despite additive notation for monoid $(H, +)$, we do not require $+$ to be commutative. Having H commutative would be equivalent to saying that the order of siblings in a tree is not relevant. Although in all the

examples given in this paper + will be commutative, one can easily find examples when it will not be the case. A prominent one is first-order logic with order on siblings.

Note 3.4. The axioms of forest algebra imply the existence of strong links between horizontal and vertical monoids. The first observation is that every element of h of H is of the form $v0$ for some $v \in V$. Indeed, it is enough to take $in_l h$ for v. Moreover, the mappings $in_l, in_r : H \to V$ are monoid morphisms as $in_l(h_1 + h_2) = in_l(h_1)in_l(h_2)$ and $in_l(0) = 1$.

A *morphism* between two forest algebras (H, V) and (G, W) is a pair of monoid morphisms $(\alpha : H \to G, \beta : V \to W)$ with additional requirements ensuring that the operations are preserved:

$$\alpha(vh) = \beta(v)\alpha(h)$$
$$\beta(in_l(h)) = in_l(\alpha(h)) \quad \text{and} \quad \beta(in_r(h)) = in_r(\alpha(h))$$

Note 3.5. The morphism α is determined by β via

$$\alpha(h) = \alpha(h + 0) = \alpha(in_l(h)0) = \beta(in_l(h))\alpha(0) \ ,$$

where $\alpha(0)$ must be the neutral element in G by the assumption on α being a monoid morphism. So it is enough to give a morphism β and verify if together with the uniquely determined α they preserve the operations.

Given an alphabet A, we define the *free forest algebra* over A, which is denoted by A^Δ, as follows:

- The horizontal monoid is the set of forests over A.
- The vertical monoid is the set of contexts over A.
- The action is the substitution of forests in contexts.
- The in_l function takes a forest and transforms it into a context with a hole to the right of all the roots in the forest. Similarly for in_r but the hole is to the left of the roots.

Observe that in_l and in_r are uniquely determined by insertion axioms, once the action is defined. The following lemma shows that free forest algebra is free in the sense of universal algebra.

Lemma 3.6. The free forest algebra A^Δ is a forest algebra. Moreover, for every forest algebra (H, V), every function $f : A \to V$ can be uniquely extended to a morphism $(\alpha, \beta) : A^\Delta \to (H, V)$ such that $\beta(a(*)) = f(a)$ for every $a \in A$.

Proof. That A^Δ is a forest algebra can be easily verified. We define a homomorphism by induction on the size of a tree/context:

$$\begin{aligned} \alpha(0) &= 0 & \beta(*) &= 1 \\ \alpha(at) &= f(a)(\alpha(t)) & \beta(a(p)) &= f(a)\beta(p) \\ \alpha(t_1 + t_2) &= \alpha(t_1) + \alpha(t_2) & \beta(t_1 + p + t_2) &= in_l(\alpha(t_1))in_r(\alpha(t_2))\beta(p) \end{aligned}$$

Directly from the definition it follows that α, β is a unique possible extension of f to a homomorphism. It can be checked that the two mappings are well defined. It is clear that α preserves $+$ operation. One shows that $\beta(pq) = \beta(p)\beta(q)$ by induction on the size of p. The preservation of the action property: $\alpha(pt) = \beta(p)\alpha(t)$ is also proved by induction on p. Finally, $\beta(in_l(t)) = \beta(t + *) = \alpha(t_1) + \beta(1) = in_l(\alpha(t))\beta(1) = in_l(\alpha(t))$. Q.E.D.

We now proceed to define languages recognized by forest algebras.

Definition 3.7. A set L of A-forests is said to be *recognized* by a surjective morphism $(\alpha, \beta) : A^\Delta \to (H, V)$ if L is the inverse image $\alpha^{-1}(G)$ of some $G \subseteq H$. The morphism (α, β) is said to recognize L, the set G is called the *accepting set*, and L is said to be recognized by (H, V).

Generally, we are interested in the case when (H, V) is finite; in this case we say that L is *recognizable*.

Example 3.8. Consider the set L of forests with an even number of nodes. We present here a finite forest algebra (H, V) recognizing L. Both H and V are $\{0, 1\}$ with addition modulo 2. The action is also addition; this defines the insertion functions uniquely. The recognizing morphism maps a context onto 0 if it has an even number of nodes. The accepting set is $\{0\}$.

Example 3.9. A language L of A-forests is called *label-testable* if the membership $t \in L$ depends only on the sets of labels that occur in t. The appropriate forest algebra is defined as follows. Both H and V are the same monoid: the set $P(A)$ with union as the operation. This determines the action, which must also be also union. We can take as a recognizing morphism a function that maps a context to the set of its labels.

Note 3.10. Another way to look at a forest algebra is from the point of view of universal algebra. In this setting, a forest algebra is a two-sorted algebra (with the sorts being H and V) along with two constants (neutral elements for H and V) and five operations: (i) monoid operations in H and V, (ii) the action vh of V on H and (iii) the two insertion operations in_l and

in_r. Forest algebras cannot be defined equationally due to the faithfulness requirement.

The universal algebra viewpoint gives us definitions of such concepts as subalgebra, cartesian product, quotient, morphism. The requirement of faithfulness is not preserved by homomorphic images and quotients. This implies that every time we take a quotient we need to check if the result is a faithful algebra.

3.1 Syntactic algebra for forest languages

Our aim now is to establish the concept of a syntactic forest algebra of a forest language. This is going to be a forest algebra that recognizes the language, and one that is optimal among those that do.

Definition 3.11. We associate with a forest language L two equivalence relations on the free forest algebra A^Δ:

- Two A-forests s, t are *L-equivalent* if for every context p, either both or none of the forests ps, pt belong to L.
- Two A-contexts p, q are *L-equivalent* if for every forest t, the forests pt and qt are L-equivalent.

Lemma 3.12. Both L-equivalence relations are congruences with respect to the operations of the forest algebra A^Δ.

Proof. We first show that L-equivalence for forests is a congruence with respect to concatenation of forests. We will consider only concatenation to the right. We show that if s and s' are L-equivalent, then so are the forests $s+t$ and $s'+t$, for every forest t. Unraveling the definition of L-equivalence, we must show that for every context p we have: $p(s+t) \in L$ iff $p(s'+t) \in L$. Taking $q = p \cdot in_r(t)$ we get $qs = p(in_r(t)s) = p(s+t)$. In consequence:

$$p(s+t) \in L \quad \text{iff} \quad q(s) \in L \quad \text{iff} \quad q(s') \in L \quad \text{iff} \quad p(s'+t) \in L \ ,$$

where the middle equivalence follows from L-equivalence of s and s'. The proof for the concatenation to the left is analogous.

We now proceed to show that L-equivalence for contexts is a congruence with respect to context composition. We need to show that if two contexts p and p' are L-equivalent, then so are the contexts pq and $p'q$ for any context q (and similarly for the concatenation to the left). We need to show that for every forest t and every context q',

$$q'pqt \in L \quad \text{iff} \quad q'p'qt \in L \ .$$

The above equivalence follows immediately from the L-equivalence of p and p': it suffices to consider qt as a tree that is plugged into the contexts p and p'.

In a similar way, one shows that L-equivalence is a congruence with respect to the action pt and the insertions $in_l(t)$, $in_r(t)$. Q.E.D.

Definition 3.13. The *syntactic forest algebra* for L is the quotient of A^Δ with respect to L-equivalence, where the horizontal semigroup H^L consists of equivalence classes of forests over A, while the vertical semigroup V^L consists of equivalence classes of contexts over A. The *syntactic morphism* (α^L, β^L) assigns to every element of A^Δ its equivalence class in (H^L, V^L).

The above lemma guarantees that the quotient is well defined. In this quotient, faithfulness holds thanks to the definition of L-equivalence over contexts. The action and insertion axioms are also satisfied (as it is a quotient of a forest algebra). Hence, it is a forest algebra. We claim that this forest algebra satisfies the properties required from the syntactic forest algebra of L.

Proposition 3.14. A language L of A-forests is recognized by a the syntactic morphism (α^L, β^L). Moreover, any morphism $(\alpha, \beta) : A^\Delta \to (H, V)$ that recognizes L can be extended by a morphism $(\alpha', \beta') : (H, V) \to (H^L, V^L)$ so that $\beta' \circ \beta = \beta^L$.

Proof. The first part follows immediately by taking as an accepting set the set of L-equivalence classes of all the elements of L. The second statement follows from the observation that if two A-forests or contexts have the same image under (α, β) then they are L-equivalent. Q.E.D.

Note that in general the syntactic forest algebra may be infinite. However, Proposition 3.14 shows that if a forest language is recognized by some finite forest algebra, then its syntactic forest algebra must also be finite. In this case the syntactic forest algebra can be also easily computed. The procedure is the same as for syntactic monoids. Given a finite forest algebra (H, V) and a subset $G \subseteq H$ one marks iteratively all the pairs of elements that are not equivalent with respect to G. First, one marks all pairs consisting of an element of G and of an element of $H \setminus G$. Then one marks a pair $(h_1, h_2) \in H \times H$ if there is a $v \in V$ such that (vh_1, vh_2) is already marked. One marks also a pair of vertical elements (v_1, v_2) if there is a horizontal element h with $(v_1 h, v_2 h)$ already marked. This process continues until no new pairs can be marked. The syntactic forest algebra is the quotient of the given algebra by the relation consisting of all the pairs that are not marked. In Section 3.3 we will show that recognizability is equivalent with being accepted by the standard form of automata. In particular the proof of Proposition 3.19 gives a way of constructing a forest algebra from automaton. Together with the above discussion this gives a method of constructing a syntactic forest algebra for the language accepted by a given tree automaton.

3.2 Forest algebras and tree languages

Forest algebras give a natural definition of recognizable forest languages (Definition 3.7). However, tree languages are studied more often than forest languages. In this section we describe how a forest algebra can be used to recognize a language of unranked trees.

Definition 3.15. Given a tree language L over A and a letter $a \in A$, the *a-quotient*, denoted $a^{-1}L$, is the set of forests t that satisfy $at \in L$. A language L of A-trees is *tree-recognized* by a morphism $(\alpha, \beta) : A^\Delta \to (H, V)$ if $a^{-1}L$ is recognized by (α, β) for all $a \in A$.

Note that the above definition does not say anything about trees with only one (root-leaf) node; but these are finitely many and irrelevant most of the time. In particular, regular languages are closed under adding or removing a finite number of trees.

Example 3.16. A tree language of the form: "the root label is $a \in A$" is tree-recognized by any forest algebra. This because all the quotients $b^{-1}L$ for $b \in A$ are either empty (when $b \neq a$) or contain all forests (when $b = a$).

The above definition of recognizability induces a definition of syntactic forest algebra for a tree language L. Consider the intersection of all $(a^{-1}L)$-equivalences for $a \in A$. This is a congruence on A^Δ as it is an intersection of congruences. It is easy to check that the result is a faithful algebra.

Note 3.17. There is an alternative definition of tree-recognizability. In the alternative definition, we say that a tree language L is tree-recognized by a forest algebra (H, V) if there is a forest language K recognized by (H, V) such that L is the intersection of K with the set of trees. Under this alternative definition, there is no correct notion of syntactic algebra. For instance, the tree language "trees whose root label is a" can be tree-recognized by two forest algebras that have no common quotient tree-recognizing this language. Indeed, these may be forest algebras for two different forest languages that agree on trees.

Note 3.18. Yet another alternative definition of tree-recognizability says that L is tree-recognized iff it is recognized. In this case, the forest algebra must keep track of what is a single tree, and what is a forest. As a result, it becomes impossible to characterize some languages by properties of their syntactic algebras. For instance, consider the tree language "all trees". The horizontal monoid of this language has three elements: $H = \{0, 1, 2+\}$ which keep track of the number of trees in the forest. The vertical monoid has the transformations

$$V = \{h \mapsto h, h \mapsto h + 1, h \mapsto h + 2, h \mapsto 1, h \mapsto 2+\} .$$

The first three are contexts with the hole in the root, and the last two have the hole in a non root node. It is already inconvenient that the simplest possible tree language needs a non-trivial algebra. Furthermore, this same algebra recognizes the language "trees over $\{a, b\}$ where a does not appear in the root". The recognizing morphism maps the label a to $h \mapsto h + 2$ and the label b to $h \mapsto 1$. One can suggest several logics that can describe the first language but not the second, for all these logics characterizations in terms of syntactic algebras will be impossible.

3.3 Automata over forests

We would like to show that our definition of recognizability is equivalent with the standard notion of regular languages, i.e., languages accepted by automata. There are numerous presentations of automata on finite unranked trees and forests; here we will use one that matches our algebraic definitions.

A *forest automaton* over an alphabet A is a tuple

$$\mathcal{A} = \langle (Q, 0, +),\ A,\ \delta : (A \times Q \to Q),\ F \subseteq Q \rangle$$

where $(Q, 0, +)$ is a finite monoid; intuitively a set of states with an operation of a composition of states.

The automaton assigns to every forest t a value $t^{\mathcal{A}} \in Q$, which is defined by induction as follows:

- if t is an empty forest, then $t^{\mathcal{A}}$ is 0;
- if $t = as$, then $t^{\mathcal{A}}$ is defined to be $\delta(a, s^{\mathcal{A}})$, in particular if t is a leaf then $t^{\mathcal{A}} = \delta(a, 0)$;
- if $t = t_1 + \cdots + t_n$ then $t^{\mathcal{A}}$ is defined to be $t_1^{\mathcal{A}} + \cdots + t_n^{\mathcal{A}}$; observe that the $+$ operation in the last expression is done in $(Q, 0, +)$.

A forest t is *accepted by* $\mathcal{A}$ if $t^{\mathcal{A}} \in F$.

Proposition 3.19. A forest language is recognized by a finite forest algebra if and only if it is the language of forests accepted by some forest automaton.

Proof. Take a tree language L recognized by a morphism $(\alpha, \beta) : A^{\Delta} \to (H, V)$. That is $L = \alpha^{-1}(F)$ for some $F \subseteq H$. For the "only if" part, we need to show how it can be recognized by an automaton. Let $\mathcal{A} = \langle H, A, \delta : A \times H \to H, F \rangle$ where H is the horizontal monoid, $F \subseteq H$ is as above, and δ is defined by

$$\delta(a, h) = \beta(a)h \qquad \text{for } a \in A\ .$$

By induction on the size of the forest one can show that $t^{\mathcal{A}} = \alpha(t)$. Thus $\mathcal{A}$ recognizes the language of forests L.

For the other direction, suppose that we are given an automaton $\mathcal{A} = \langle (Q, 0, +), A, \delta, F\rangle$. We consider a forest algebra (H, V) where H is $(Q, 0, +)$ and V is the function space $H \to H$ with function composition as the operation and the identity as the neutral element. The action is function application and the insertions are uniquely determined. It is easy to see that (H, V) is a forest algebra. Consider now the unique homomorphism $(\alpha, \beta) : A^\Delta \to (H, V)$ with

$$\beta(a) = \delta(a) \qquad \text{for } a \in A \ ;$$

observe that each $\delta(a)$ is a function from H to H. This homomorphism might not be surjective as required by Definition 3.7, in this case we only keep the part of the algebra used by the homomorphism. By induction on the height of the forest one can show that $t^{\mathcal{A}} = \alpha(t)$. Q.E.D.

Actually, the above notion of automaton can be refined to a notion of (H, V) automaton for any forest algebra (H, V). Such an automaton has the form:

$$\mathcal{A} = \langle H,\ A,\ \delta : A \to V,\ F \subseteq H\rangle$$

thus the only change is that now states are from H and $\delta(b)$ is an element of from V while before it was a function from $Q \to Q$. We can do this because using the action *act* of the forest algebra, each $v \in V$ defines a function $act(v) : H \to H$.

By the same reasoning as before, every language accepted by a (H, V) automaton is recognized by the algebra (H, V). Conversely, every language recognized by (H, V) is accepted by some (H, V) automaton. This equivalence shows essential differences between algebras and automata. Algebras do not depend on alphabets, while alphabets are explicitly declared in the description of an automaton. More importantly, the structure of the vertical semigroup is not visible in an automaton: in an automaton we see only generators of the vertical semigroup.

It may be worth to compare the above automata model with unranked tree automata (UTA's) [17, 10]. The only difference of any importance between these models is that UTA's have transition function of the form $\delta : \Sigma \times Q \to Reg(Q)$, i.e., to each pair of state and a letter, a UTA assigns a regular language over the alphabet Q. A tree whose root is labeled a can be assigned a state q if the sequence of states assigned to its children is in the regular language $\delta(q, a)$. In our case regular languages are represented by monoids. More precisely, we use one monoid structure on states to simultaneously recognize all regular word languages that appear in transitions. The two automata models have the same expressive power, and effective translations can be easily presented. Note that since we use monoids, there may be an exponential growth when translating from a UTA to a forest automaton.

3.4 Other possible variants of forest algebra

For words, one can use either monoids or semigroups to recognize word languages. In the first case, the appropriate languages are of the form $L \subseteq A^*$, while the second case disallows the empty word, and only languages $L \subseteq A^+$ are considered.

For forests, the number of choices is much greater. Not only do we have two sorts (forests and contexts) instead of just one (words), but these sorts are also more complex. This requires at least two choices:

- Is the empty forest a forest? Here, we say yes.
- Is the empty context a context? Here, we say yes.

We can also put some other restrictions on a position of the hole in the context, for example that it cannot have siblings, or that it cannot be in the root. Each combination of answers to the above questions gives rise to an appropriate definition of a forest algebra, as long as the correct axioms are formulated.

We do not lay any claim to the superiority of our choices. The others are just as viable, but this does not mean that they are all equivalent. The difference becomes visible when one tries to characterize algebras by equations. For example, the equation $vh = vg$ in our setting implies $h = g$ because this equation should be valid for all assignments of elements to variables, and in particular we can assign the identity context to v. But then, $h = g$ says that the horizontal monoid is trivial. If we did not allow contexts with the hole in a root, this equation would describe forest languages where membership of a forest depends only on the labels of its roots.

One may also ask what would happen if we had dropped the vertical structure. We could work with pairs of the form (H, Z) where Z is just a set and not a semigroup, but still we could have an action of Z on H. Such pairs correspond to automata where the alphabet is not fixed. For such objects we do not need to require insertion axioms as these axioms talk about the structure of the vertical semigroup which is not present here. All the theory could be developed in this setting but once again equations would have different meaning in this setting. In particular we would not have any way to refer explicitly to vertical composition. We refrain from doing this because we think that the structure of the vertical semigroup is important.

4 Simple applications

In this section we present two straightforward characterizations of forest languages. Both are effective, meaning that the conditions on the forest algebra can be effectively tested. The first characterization—of label testable

languages—illustrates how a property of the context monoid can have important implications for the forest monoid. The second characterization—of languages definable by a Σ_1 formula—shows that we can also consider language classes that are not closed under boolean operations.

In the following we will very often express properties of algebras by equations. An equation is a pair of terms in the signature of forest algebras over two types of variables: horizontal variables (h, g, ...), and vertical variables (v, w,...). These terms should be well typed in an obvious sense and should have the same type: both should be either of the forest type, or of the context type. An algebra *satisfies* an equation if for any valuation assigning elements of the horizontal monoid to horizontal variables, and elements of the vertical monoid to vertical variables, the two terms have the same value. In this way an equation expresses a propery of algebras.

We say a forest language is *label testable* if the membership in the language depends only on the set of labels that appear in the forest.

Theorem 4.1. A language is label testable if and only if its syntactic algebra satisfies the equations:

$$vv = v \qquad vw = wv \ .$$

Proof. The only if part is fairly obvious, we only concentrate on the if part. Let then L be a language recognized by a morphism $(\alpha, \beta) : A^\Delta \to (H, V)$, with the target forest algebra satisfying the equations in the statement of the theorem. We will show that for every forest t the value $\alpha(t)$ depends only on the labels appearing in t.

We start by showing that the two equations from the statement of the theorem imply another three. The first is the idempotency of the horizontal monoid:

$$h + h = h \ .$$

This equation must hold in any forest algebra satisfying our assumption because of the following reasoning which uses the idempotency of the vertical monoid:

$$h + h = (h + 1)(h + 1)0 = (h + 1)0 = h \ .$$

(In the above, $h+1$ denotes the context $in_l(h)$.) The second is the commutativity of the horizontal monoid:

$$h + g = g + h \ .$$

The argument uses commutativity of the vertical monoid:

$$h + g = (h + 1)(g + 1)0 = (g + 1)(h + 1)0 = g + h \ .$$

Finally, we have an equation that allows us to flatten the trees:

$$v(h) = h + v0 \ .$$

The proof uses once again the commutativity of the vertical monoid:

$$v(h) = v(h+1)0 = (h+1)v0 = h + v0 \ .$$

The normal form of a forest will be a forest $a_1 0 + \cdots + a_n 0$, where each tree contains only one node, labeled a_i. Furthermore, the labels $a_1, \ldots, a_n$ are exactly the labels used in t, sorted without repetition under some arbitrary order on the set A. Using the three equations above one can show that every forest has the same value under α as its normal form. Starting from the normal form one can first use idempotency to "produce" as many copies of each label as the number of its appearances in the tree. Then using the last equation and the commutativity one can reconstruct the tree starting from leaves and proceeding to the root. Q.E.D.

Note 4.2. If we omit the equation $vv = v$, we get languages that can be defined by a boolean combination of clauses of the forms: "label a occurs at least k times", or "the number of occurrences of label a is $k \mod n$".

We now present the second characterization. A Σ_1 *formula* is a formula of first-order logic, where only existential quantifiers appear in the quantifier prenex normal form. The logic we have in mind uses the signature allowing label tests (a node x has label a) and the descendant order (a node x is a descendant of a node y). The following result shows which forest languages can be defined in Σ_1:

Theorem 4.3. Let L be a forest language, and let (α, β) be its syntactic morphism. A language L is definable in Σ_1 if and only if $vh \in \alpha(L)$ implies $vwh \in \alpha(L)$.

Proof. The only if implication is an immediate consequence of the fact that languages defined in Σ_1 are closed under adding nodes. We will now show the if implication. Below, we will say that a forest s is a piece of a forest t if s can be obtained from t by removing nodes (i.e. the transitive closure of the relation which reduces a forest pqs to a forest ps).

Let L be a language recognized by a morphism $(\alpha, \beta) : A^\Delta \to (H, V)$, with α satisfying the property in the statement of the theorem. For each $h \in H$, let T_h be the set of forests that are assigned h by α, but have no proper piece with this property. Using a pumping argument, one can show that each set T_h is finite. We claim that a forest belongs to L if and only if it contains a piece $t \in T_h$, with $h \in \alpha(L)$. The theorem follows from this claim, since the latter property can be expressed in Σ_1.

The only if part of the claim is obvious: if a forest t belongs to L, then by definition it contains a piece from $T_{\alpha(t)}$, since $\alpha(t)$ belongs to $\alpha(L)$. For the if part of the claim, we need to use the property of α stated in the theorem: if t contains a piece s with $\alpha(s) \in \alpha(L)$, then by iterative application of the implication $vh \in \alpha(L) \Rightarrow vwh \in \alpha(L)$, we can show that $\alpha(t)$ also belongs to $\alpha(L)$, and hence t belongs to L. Q.E.D.

5 Characterization of EF

In this section we show how forest algebras can be used to give a decidable characterization of a known temporal logic for trees. The logic in question, called EF, is a fragment of CTL where EF is the only temporal operator allowed. Decidability of this fragment for the case of binary trees is known [5], and several alternative proofs have already appeared [23, 7]. Here, we would like to show how our setting—which talks about forests—can be used to show decidability of a logic over trees.

5.1 The logic EF

EF is a temporal logic that expresses properties of trees. The name EF is due to the unique temporal operator in the logic — EF — which stands for Exists (some path) Further down (on this path). Formulas of EF are defined as follows:

- If a is a letter, then a is a formula true in trees whose root label is a.
- EF formulas are closed under boolean connectives.
- If φ is an EF formula, then $\mathsf{EF}\varphi$ is an EF formula true in trees having a *proper* subtree satisfying φ.

We write $t \vDash \varphi$ to denote that a formula φ is true in a tree t.

Restricting to proper subtrees in the definition of EF gives us more power, since the non-proper operator can be defined as $\varphi \vee \mathsf{EF}\varphi$.

We need to deal with a mismatch due to the fact that EF is defined over trees and our algebraic setting works with forests. For this, we need to define how forest languages can be defined in EF.

Definition 5.1. A *tree language L is definable in* EF iff there is an EF formula α with $L = \{t : t \vDash \alpha\}$. A *forest language L is definable in* EF if for some $a \in A$ the tree language $\{at : t \in L\}$ is definable in EF.

Notice that the choice of a in the above definition does not matter. The following observation shows that we can use forest definability to decide tree definability.

Lemma 5.2. A tree language L is EF definable iff for every $a \in A$ the forest language $a^{-1}L$ is EF definable (as a language of forests).

Proof. Suppose L is a tree language defined by a formula φ. This formula is boolean combination for formulas starting with EF and formulas of the form b for some $b \in A$. It is easy to see that φ can rewritten as a conjunction of implications $\bigwedge_{b \in A} b \Rightarrow \varphi_b$, where φ_b, for all $b \in A$, is a boolean combination of formulas starting with EF. Then φ_a defines the forest language $a^{-1}L$.

For the other direction suppose that for each $a \in A$ the forest language $a^{-1}L$ is EF-definable. So there is a formula φ_a and a letter $b \in A$ such that $bt \vDash \varphi_a$ iff $t \in a^{-1}L$. We can, if necessary, modify φ_a into φ'_a with the property that $bt \vDash \varphi_a$ if and only if $at \vDash \varphi'_a$. The tree language L is then defined by $\bigwedge_{a \in A} a \Rightarrow \varphi'_a$. Q.E.D.

As the main result of this section, we present two equations and show that a forest language is definable by an EF formula if and only if its syntactic forest algebra satisfies these equations. In particular, it is decidable if a regular tree language can be defined in EF.

Theorem 5.3. A forest language is definable in EF if and only if its syntactic forest algebra satisfies the following equations, called the EF equations:

$$g + h = h + g \tag{1.1}$$

$$vh = h + vh\ . \tag{1.2}$$

Equation (1.1) states that the horizontal monoid is commutative. In other words, membership of a forest in the language does not depend on order of siblings. Equation (1.2) is specific to EF and talks about interaction between two monoids. This equation also shows an advantage of our setting: the equation can be that simple because we need not to worry about the degree of vertices, and we can compare not only trees but also forests. The proof of the theorem is split across the following two subsections.

Note 5.4. One can also consider the logic EF^*, where the EF modality is replaced by its non-strict version EF^*. A formula $\mathsf{EF}^*\varphi$ is equivalent to $\varphi \vee \mathsf{EF}\varphi$. As mentioned before, this logic is strictly weaker than EF. For example, one cannot express in EF^* that a tree consists only of one leaf. Recently, a decidable characterization of EF^* was given in [23, 7]. The logic $\mathsf{EF}^*\varphi$ is not as well-behaved in our algebraic setting as EF. The problem is that one cannot tell if a forest language is definable in EF^* just by looking at its syntactic forest algebra. For an example, consider the language defined by the formula $\mathsf{EF}^*(b \wedge \mathsf{EF}^* c)$, over the alphabet $\{a, b, c\}$. The syntactic forest algebra for this language can also recognize the language of flat forests (where every tree consists only of the root). But the latter language is not EF^* definable.

5.2 Correctness

We show that the syntactic algebra of a forest language definable in EF must satisfy the EF equations. The basic idea is to prove that any language definable in EF is recognized by a forest algebra satisfying the EF equations. We will then be able to conclude that the syntactic algebra must also satisfy these equations, as it is a morphic image of any algebra recognizing the language.

Assume then that a forest language L over an alphabet A is defined by a formula φ. The EF-*closure* of φ, denoted $cl_{\mathsf{EF}}(\varphi)$, is the set of all subformulas of φ of the form $\mathsf{EF}\psi$ for some ψ.

Given a forest t and $a \in A$ we define a *forest type* of t (with respect to our fixed φ):

$$FT_\varphi(t) = \{\psi \in cl_{\mathsf{EF}}(\varphi) : at \vDash \psi\} \ .$$

It is clear that this definition does not depend on the choice of a, so we do not include it in the notation.

We now define an equivalence relation on forests by saying that two forest are φ-equivalent if their FT_φ values are the same. We denote this relation by $\sim_\varphi$. The relation can be extended to contexts by saying that two contexts p, q are $\sim_\varphi$ equivalent if for every nonempty forest t, the forests pt and qt are $\sim_\varphi$ equivalent.

Lemma 5.5. The relation $\sim_\varphi$ is a congruence of the free forest algebra A^Δ.

Proof. It is clear that $\sim_\varphi$ is an equivalence relation on forests and contexts. We need to show that it is a congruence. The first preparatory step is to show by induction on the size of a context p that for any two forests $t_1 \sim_\varphi t_2$ we have $pt_1 \sim_\varphi pt_2$.

Using this we can now show that $\sim_\varphi$ preserves the action. Suppose that $p_1 \sim_\varphi p_2$ and $t_1 \sim_\varphi t_2$. Then $p_1 t_1 \sim_\varphi p_1 t_2 \sim_\varphi p_2 t_2$; where the second equivalence follows directly from the definition of $\sim_\varphi$ for contexts.

Next, we deal with monoid operations in H and V. From the definition it easily follows that if $s_1 \sim_\varphi t_1$ and $s_2 \sim_\varphi t_2$ then $s_1 + s_2 \sim_\varphi t_1 + t_2$. For the contexts take $p_1 \sim_\varphi p_2$ and $q_1 \sim_\varphi q_2$. For an arbitrary tree t we have: $q_1 p_1 t \sim_\varphi q_1 p_2 t \sim_\varphi q_2 p_2 t$. The first equivalence follows from the property proved in above, as $p_1 t \sim_\varphi p_2 t$.

Finally, we deal with the insertion operations. Take $s_1 \sim_\varphi s_2$ and an arbitrary tree t. We have $(in_l(s_1))t = s_1 + t \sim_\varphi s_2 + t = (in_l(s_2))t$. Q.E.D.

Lemma 5.6. The quotient $A^\Delta / \sim_\varphi$ is a forest algebra, and it recognizes L. Equations (1.1) and (1.2) are satisfied in the quotient.

Proof. For $A^\Delta / \sim_\varphi$ to be a forest algebra we must check if it is faithful. To check faithfulness take p, q which are not in $\sim_\varphi$ relation. Then there is a tree t such that $pt \not\sim_\varphi qt$ which gives: $[p][t] = [pt] \not\sim_\varphi [qt] = [q][t]$.

The language L is recognized by a canonical homomorphism assigning to each context its equivalence class, and the accepting set consisting of equivalence classes of trees from L. To show that it is correct we need to show that if two trees are equivalent then either both or none of them satisfies φ. This follows from the observation that φ is equivalent to a formula of the form $a \Rightarrow \varphi'$ where φ' is a boolean combination of some formulas form $cl_{\mathsf{EF}}(\varphi)$.

A straightforward inspection shows that the equations are satisfied. For example, the fact that the trees vh and $h + vh$ have the same FT_φ value follows directly from the definition of the value. Q.E.D.

As the syntactic algebra for L is a morphic image of any other algebra recognizing L (cf. Proposition 3.14), all equations satisfied in $A^\Delta/\sim_\varphi$ must hold also in the syntactic algebra.

Corollary 5.7. The syntactic algebra of an EF definable forest language satisfies the equations (1.1) and (1.2).

5.3 Completeness

In this section we show that if a forest algebra satisfies the two EF equations, then every forest language recognized by this algebra can be defined in EF. This gives the desired result, since the syntactic algebra of L recognizes L.

From now on we fix a forest algebra (H, V) that recognizes a forest language L via a morphism

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

We assume that the forest algebra (H, V) satisfies the two EF equations (1.1) and (1.2). We will show that L can be defined using an EF formula.

We first show that the EF equations imply two other properties:

$$h = h + h \tag{1.3}$$

$$w(vw)^\omega = (vw)^\omega \ . \tag{1.4}$$

These state idempotency of the horizontal monoid, and L-triviality of the vertical monoid, respectively. We need to explain the ω notation, though. In each finite semigroup (and hence in each monoid) S, there is a power $n \in \mathbb{N}$ such that all elements $s \in S$ satisfy $s^n = s^n s^n$. We refer to this power as ω, and use it in equations. In particular, every finite semigroup satisfies the equation: $s^\omega = s^\omega s^\omega$. The reader is advised to substitute "a very large power" for the term ω when reading the equations.

The idempotency of the horizontal monoid follows directly from the equation $vh = h + vh$, by taking v to be the neutral element of the vertical monoid. Observe that we always have $1h = h$, as $h = u0$ for some u and then $1(u0) = (1u)0 = u0$. The proof for the other equation is more complicated.

Lemma 5.8. For each $v, w \in V$, we have $w(vw)^\omega = (vw)^\omega$

Proof. First we show that the EF equations imply aperiodicity for the context monoid:

$$v^\omega = vv^\omega \ .$$

Indeed, by applying the first equation repeatedly to $v^\omega v^\omega$, we obtain:

$$v^\omega = v^\omega v^\omega = v^\omega + vv^\omega + vvv^\omega + \cdots + v^\omega v^\omega$$

Likewise for $vv^\omega v^\omega$:

$$vv^\omega = vv^\omega v^\omega = vv^\omega + vvv^\omega + vvvv^\omega + \cdots + v^\omega v^\omega + vv^\omega v^\omega$$

If we cancel out $vv^\omega v^\omega = vv^\omega$, and use idempotency and commutativity of H, we obtain the desired equality $v^\omega = vv^\omega$.

We now proceed to show the statement of the lemma.

$$w(vw)^\omega = (vw)^\omega + w(vw)^\omega = vw(vw)^\omega + w(vw)^\omega = vw(vw)^\omega = (vw)^\omega \ .$$

In the first and third equation we use $vh = h + vh$, while in the second and fourth we use aperiodicity. Q.E.D.

The main idea of the proof is to do an induction with respect to what forests can be found inside other forests. Given $g, h \in H$, we write $g \leq h$ there is some context $u \in V$ such that $h = ug$. We write $g \sim h$ if $\leq$ holds both ways. Here are three simple properties of these relations. The first is a direct consequence of the second EF equation. The other two require a short calculation.

Lemma 5.9. If $g \leq h$ then $g + h = h$.

Lemma 5.10. If $g \sim h$ then $g = h$. In particular, $\leq$ is a partial order.

Proof. Assume that $g \neq h$. If $g \sim h$ then there are contexts v, w such that $h = wg$ and $g = vh$. Iterating this process ω-times we obtain

$$h = wvh = (wv)^\omega h$$

But then, by applying Lemma 5.8, we get

$$h = (wv)^\omega h = v(wv)^\omega h = g \ .$$

Q.E.D.

Lemma 5.11. If $g_1 \leq h_1$ and $g_2 \leq h_2$ then $g_1 + g_2 \leq h_1 + h_2$.

Proof. By assumption $h_1 = v_1 g_1$ and $h_2 = v_2 g_2$. Then, by using commutativity of H and equation (1.2), we get

$$h_1 + h_2 = v_1 g_1 + v_2 g_2 = v_1 g_1 + g_1 + v_2 g_2 + g_2 \geq g_1 + g_2 \ .$$

The last inequality is a consequence of the property $g + h \geq g$ which follows from the definition of the order as $g + h = (1 + h)g$. Q.E.D.

The next proposition is the main induction in the completeness proof:

Proposition 5.12. For every $h \in H$, there is an EF formula φ_h such that for every forest t and letter a we have

$$at \vDash \varphi_h \quad \text{iff} \quad \alpha(t) = h \ .$$

Proof. The proof is by induction on the depth of h in the order $\leq$, i.e. on the number of f satisfying $f < h$ (as usual, $<$ denotes the strict version of $\leq$).

Consider first the base case, when h is minimal for $\leq$; which by the way implies that $h = 0$ is the identity of the horizontal monoid. How can a forest t satisfy $\alpha(t) = h$? All leaves need to have labels $a \in A$ satisfying $\alpha(a) = h$; this can be easily tested in EF. Second, all internal nodes need to have labels $a \in A$ satisfying $\alpha(a)h = h$; this can also be tested in EF. These conditions are clearly necessary, but thanks to idempotency $h + h = h$, they are also sufficient. It remains to say how these conditions can be expressed in EF. The formula $\exists tt$ says that a node has a proper subtree, i.e., that a node is an internal node. So, the formula $\bigwedge_{b \in B} \neg \exists b \wedge \exists tt$ expresses the fact that no internal node has the label from a set B. Similarly one can say that no leaf has a label form B.

We now proceed with the induction step. We take some $h \in H$ and assume that the proposition is true for all $f < h$. We claim that a forest t satisfies $\alpha(t) = h$ iff the following three conditions hold:

- The forest t contains a *witness*. There are two types of witness. The first type, is a forest of a form $s_1 + s_2$ with $\alpha(s_1) + \alpha(s_2) = h$ but $\alpha(s_1), \alpha(s_2) < h$. The second type is a tree of the form as, with $\alpha(s) < h$ and $\beta(a)\alpha(s) = h$.
- For all subtrees as of t with s containing a witness, $\beta(a)(h) = h$ holds.
- For all subtrees as of t with $\alpha(s) < h$ we have $\beta(a)\alpha(s) \leq h$; moreover, for all subtrees s_1 and s_2 of t, with $\alpha(s_1) < h$ and $\alpha(s_2) < h$ we have $\alpha(s_1 + s_2) \leq h$.

These conditions can be easily written in EF using formulas φ_f for all $f < h$. So it remains to show that they are equivalent to $\alpha(t) = h$.

Suppose that the three conditions hold. By the first condition $\alpha(t) \geq h$. If $\alpha(t)$ were strictly greater than h then there would be a minimal size subtree s of t with $\alpha(s) \not\leq h$. It cannot be of the form $s_1 + s_2$ because, by Lemma 5.11, if $\alpha(s_1), \alpha(s_2) \leq h$ then $\alpha(s_1) + \alpha(s_2) \leq h$. So this minimal tree should be of the form as. It cannot be the case that $\alpha(s) = h$ because of the second property. If $\alpha(s) < h$ then the third property guarantees $\beta(a)\alpha(s) \leq h$, a contradiction.

Suppose now that $\alpha(t) = h$. It is clear that a minimal subtree of t which has the value h is a witness tree satisfying the first property. The second property is obvious. Regarding the third property, it is also clear that for every subtree of the form as if $\alpha(s) < h$ then $\beta(a)\alpha(s) \leq h$. It remains to check that for every two subtrees s_1, s_2 with $\alpha(s_1), \alpha(s_2) < h$ we have $\alpha(s_1) + \alpha(s_2) \leq h$. Take two such subtrees and a minimal tree containing both of them. If it is, say s_2, then $\alpha(s_1) < \alpha(s_2)$ and $\alpha(s_1) + \alpha(s_2) = \alpha(s_2) < h$. Otherwise, s_1 and s_2 are disjoint, and the minimal subtree has the form $b(t_1 + t_2 + t_3)$ with t_1 containing s_1, and t_2 containing s_2 (due to commutativity, the order of siblings does not matter). Now we have $\alpha(s_1) \leq \alpha(t_1)$ and $\alpha(s_2) \leq \alpha(t_2)$ which gives $\alpha(s_1 + s_2) \leq \alpha(t_1 + t_2) \leq \alpha(t) = h$ by Lemma 5.11. Q.E.D.

6 Conclusions and future work

This work is motivated by decidability problems for tree logics. As mentioned in the introduction, surprisingly little is known about this subject. We hope that this paper represents an advance, if only by making more explicit the algebraic questions that are behind these problems. Below we discuss some possibilities for future work.

Wherever there is an algebraic structure for recognizing languages, there is an Eilenberg theorem. This theorem gives a bijective mapping between classes of languages with good closure properties (language varieties) and classes of monoids with good closure properties (monoid varieties). It would be interesting to see how this extends to trees, i.e. study varieties forest algebras. Indeed, we have used equations to characterize EF, in particular the appropriate class of forest algebras will satisfy all closure properties usually required of a variety. The next step is to a develop variety theory, and check what classes of forest algebras can be defined using equations. Under a certain definition, it can be shown that first-order definable languages form a variety, so does CTL*, and chain logic. There are also logics that do not correspond to varieties; we have given EF^* as an example. This situation is well known in the word case: for some logics one needs to work in monoids, for others in semigroups. In the case of trees the choice is bigger. For exam-

ple, a characterization of EF^* requires to forbid contexts consisting of just a hole. Another example is a characterization of first-order logic with two variables [4] where the empty tree is excluded.

A related topic concerns $\mathcal{C}$-varieties [16]. This is a notion from semigroup theory, which — among others — does away with the tedious distinction between semigroup and monoid varieties. It would be interesting to unify the variants mentioned above in a notion of $\mathcal{C}$-variety of forest algebras.

There are of course classes of tree languages — perhaps even more so in trees than words — that are not closed under boolean operations: take for instance languages defined by deterministic top down automata, or Σ_1 definable languages presented here. In the case of words, ordered semigroups extend the algebraic approach to such classes. It would be interesting to develop a similar concept of ordered forest algebras.

The logics considered in this paper cannot refer to the order on siblings in a tree. It would be worthwhile to find correct equations for logics with the order relation on siblings. It is also not clear how to cope with trees of bounded branching. One can also ask what is the right concept of forest algebras for languages of infinite trees.

References

[1] D. Beauquier and J-E. Pin. Factors of words. In *ICALP'89*, volume 372 of *LNCS*, pages 63 – 79, 1989.

[2] M. Benedikt and L. Segoufin. Regular languages definable in FO. In *STACS'05*, volume 3404 of *LNCS*, pages 327 – 339, 2005. See the corrected version on the authors web page.

[3] M. Bojańczyk. *Decidable Properties of Tree Languages*. PhD thesis, Warsaw University, 2004.

[4] M. Bojanczyk. Two-way unary temporal logic over trees. In *LICS'07*, 2007. To appear.

[5] M. Bojanczyk and I. Walukiewicz. Characterizing EF and EX tree logics. *Theoretical Computer Science*, 358(2-3):255–272, 2006.

[6] J. Cohen, D. Perrin, and J-E. Pin. On the expressive power of temporal logic. *Journal of Computer and System Sciences*, 46(3):271–294, 1993.

[7] Z. Esik and I. Szabolcs. Some varieties of finite tree automata related to restricted temporal logics. *Fundamenta Informaticae*, 2007. To appear.

[8] Z. Esik and P. Weil. On certain logically defined tree languages. In *FSTTCS'03*, volume 2914 of *LNCS*, pages 195–207, 2003.

[9] U. Heuter. First-order properties of trees, star-free expressions, and aperiodicity. In *STACS'88*, volume 294 of *LNCS*, pages 136–148, 1988.

[10] L. Libkin. Logics for unranked trees: an overview. *Logical Methods in Computer Science*, 2:1–31, 2006.

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

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

[13] A. Potthoff. First-order logic on finite trees. In *Theory and Practice of Software Development*, volume 915 of *LNCS*, pages 125–139, 1995.

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

[15] H. Straubing. *Finite Automata, Formal Languages, and Circuit Complexity*. Birkhäuser, Boston, 1994.

[16] H. Straubing. On logical descriptions of regular languages. In *LATIN'02*, volume 2286 of *LNCS*, pages 528 – 538, 2002.

[17] J.W. Tatcher. Characterizing deriviation trees of context-free grammars through a generalization of finite automata theory. *J. Counput. Syst. Sci.*, 1:317–322, 1967.

[18] D. Thérien and A. Weiss. Graph congruences and wreath products. *Journal of Pure and Applied Algebra*, 36:205–215, 1985.

[19] D. Thérien and T. Wilke. Temporal logic and semidirect products: An effective characterization of the Until hierarchy. In *FOCS'96*, pages 256–263, 1996.

[20] D. Thérien and T. Wilke. Over words, two variables are as powerful as one quantifier alternation. In *ACM Symposium on the Theory of Computing*, pages 256–263, 1998.

[21] T. Wilke. Algebras for classifying regular tree languages and an application to frontier testability. In *ICALP'93*, volume 700 of *LNCS*, pages 347–358, 1993.

[22] T. Wilke. Classifying discrete temporal properties. In *STACS'99*, volume 1563 of *LNCS*, pages 32–46, 1999.

[23] Z. Wu. A note on the characterization of TL[EF]. *Information Processing Letters*, 102(2-3):28–54, 2007.